Evolution of the VISPA-project

Martin Erdmann[1], *Benjamin* Fischer[1*], *Lukas* Geiger[1], *Erik* Geiser[1], *Dennis Daniel Nick* Noll[1], *Yannik Alexander* Rath[1], *Marcel* Rieger[1], *Felix* Schlüter[1,2], *David Josef* Schmidt[1], *Martin* Urban[1], and *Ralf Florian* von Cube[1,2]

[1]RWTH Aachen University
[2]Karlsruhe Institute of Technology

Abstract. VISPA (Visual Physics Analysis) is a web-platform that enables users to work on any secure shell (SSH) reachable resource using just their web-browser. It is used successfully in research and education for HEP data analysis. The emerging JupyterLab is an ideal choice for a comprehensive, browser-based, and extensible work environment and we seek to unify it with the efforts of the VISPA-project. The primary objective is to provide the user with the freedom to access any external resources at their disposal, while maintaining a smooth integration of preconfigured ones including their access permissions. Additionally, specialized HEP tools, such as native format data browsers (ROOT, PXL), are being migrated from VISPA- to JupyterLab-extensions as well. We present these concepts and their implementation progress.

1 Introduction

Any scientific workflow generally encompasses a plethora of different tasks, each with their own tools – ranging from general purpose to highly specialized. The challenge is to provide access to a working environment which serves the full range of requirements. Consequently, a web-browser based implementation is an ideal choice as it is highly portable while still enabling feature-rich user interfaces.

The VISPA software [1–5] currently provides all the functions needed to achieve this: an extensible user interface, a web server, and the methods to access any SSH reachable resource. However, with the maturing of the open-source project JupyterLab [6], a high-quality and well-maintained alternative for a user interface implementation is available. Therefore, it is sensible to consolidate the efforts of the VISPA-project and restructure it, in order to harness these new developments.

This article is structured as follows. Section 2 outlines the goals of VISPA and discuss the general concept chosen to tackle them with. Section 3 briefly reviews the current implementation and discusses considerations regarding its scope. Finally, section 4 introduces the restructuring effort, referred to as future VISPA, and details its core software design principles.

[*]presenter, e-mail: benjamin.fischer@rwth-aachen.de

2 Goals and Concept

The VISPA-project aims to provide a working environment which meets the demands of scientific workflows. In particular, the implementation must fulfill a number of requirements divided in three overarching properties:

Portable It should ...

... be accessible from anywhere without the need for any installation.
... persist personal settings such that they are retained when switching devices.
... be possible to continue a work session on a different device.

Unrestricted It should ...

... have general purpose tools: file browser, editor, and terminal.
... enable a user to use any SSH accessible custom resource.

Extensible It should ...

... have a user interface with a well defined plug-in mechanism, in particular for foreign data-format display, e.g. `.root`-files.
... support the usage of configuration templates, e.g. a CERN authenticated user may be provided with a partially configured LXPLUS setup.

SSH was chosen over other protocols or APIs due to its widespread availability alongside the following feature set. It is inherently secured through its authentication and encryption, but still can multiplex the transport of any payload. In particular, it supports two very useful features: TCP/IP tunneling and SFTP-based file access without the need, overhead, and complexities of remote-side bootstrapping.

This set of requirements is best fulfilled by a web-based application, as it has a practically unmatched portability since an already ubiquitous modern web-browser is the only needed client side software. The server can then persist settings and sessions of authenticated users, while also acting as broker for the SSH connections to the resources. However, this authentication is merely used to uniquely identify a user for providing persistence across devices. The sensitive credentials for the authentication with the resources are only forwarded to the resource and never stored.

The server and the resources executing the users' workload are deliberately separated, as they are of different concerns. It should be noted that each client is establishing private SSH connections to their resources using their own credentials. This provides the desired high flexibility and good scalability for the user's workload-carrying environment.

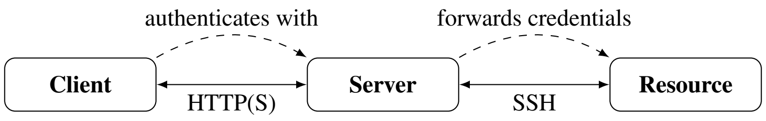

Figure 1. Schema of the three application parts and the client's authentication interactions (dashed).

3 The VISPA software

The VISPA software [5] follows the previously outlined schema and consequently consists of three components: client, server, and resource, where the latter is referred to as workspace.

The entire project is organized as one Python package and uses the CherryPy framework [7]. Stored data, e.g. users and their respective personal preferences, are organized using object-relational mapping (ORM) as provided by the SQLAlchemy toolkit [8]. Additional tools for processing of intermediate markup languages are only needed during development.

The client-side is supplied with HTML, CSS, and JavaScript/ECMAScript by the server. At runtime, further data is requested or submitted via AJAX (asynchronous JavaScript and XML). Server-side sent events and latency-critical low-bandwidth communication can be transmitted via WebSocket API or polling as a fall-back. Numerous external JavaScript libraries are used, among them jQuery, VueJS, ACE, and RequireJS [9–12].

The remote resources, here referred to as workspaces, are accessed through SSH as implemented by Paramiko [13]. Program execution, pseudo-terminal allocation, and file(-system) access are facilitated by the native implementations of the SSH protocol. Additionally, RPyC [14] is used to implement transparent remote procedure calls (RPC) within Python, which avoids the need to implement additional APIs for the communication between the server and the workspace. This necessitates the availability of Python on the remote side. The required Python packages to operate this "foothold" are automatically transferred from the server alongside the bootstrapping code. Using the RPC-mechanism additional functionality is implemented, such as file-system events and recursive directory listings.

The extension mechanism enables the integration of additional separable functionality, in the form of Python packages, which can extend any of the aforementioned three components of the VISPA software. Through these APIs an extension can interact with other extensions and use commonly needed functions, such as file manipulation and user interface controls. The built-in extensions implement basic tools, namely file-browser, editor, and terminal, and also provide the user interface for aggregated preferences, workspace configuration, and user management.

The VISPA software is successfully employed in research, outreach, and education with up to around 100 concurrent users in undergraduate courses and workshops [15–18].

4 Future VISPA

Future VISPA is a restructuring initiative which aims to consolidate our development efforts by relying on JupyterLab to provide the majority of the user interface. Consequently, the primary focus then becomes to facilitate the SSH-based connectivity.

JupyterHub [19] is an established similar project but follows a very different core design principle: First, its resource's access and allocation is heavily preconfigured by the operating administrator allowing only limited flexibility on the user's side. Secondly, network access to any TCP-port on the remote resource is required, which is usually only permissible in a well isolated system due to security considerations. Overall, it is better suited for providing controlled access to resources known in advance for example in the scope of a lecture or workshop which uses the universities local computing cluster.

In contrast, future VISPA aims at broader scope with increased flexibility for the user, especially since scientific work often has heterogeneous resources and requirements.

4.1 Structure

Future VISPA follows the same structure as outlined in section 2 and figure 1: The central server operates as a connection broker while also supplying the client with needed markup and code. However, instead of implementing and providing the work-environment on its own, it merely spawns third party software, e.g. a JupyterLab instance, on the remote resource and delegates any traffic accordingly by tunneling HTTP requests through the established SSH connection.

Throughout the entire project, flexibility is ensured by modularizing functional parts as much as feasible. That means that the server consists of multiple components namely authentication, data storage, connection-broker implementation, and static data delivery, each of which has a limited service-scope and the ability to be disabled or exchanged as desired. But also the mechanism that defines and operates every connection is modularized in a very granular manner through so-called *pipelets*, which can be combined by users to form the desired connections.

A comprehensive overview over these structures and their relations is shown in figure 2, which is discussed in detail in the following sections.

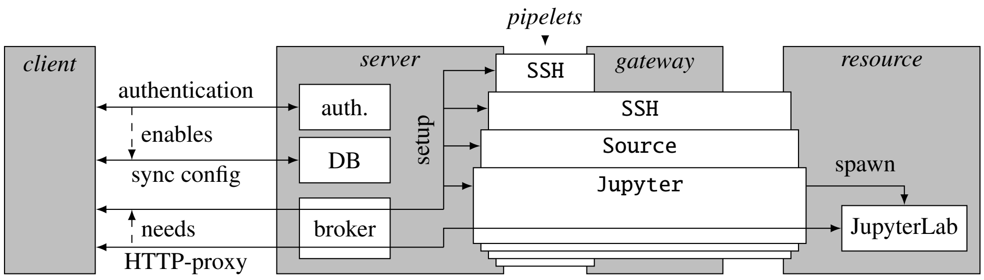

Figure 2. Structure of future VISPA with example client interaction and thereby instantiated *pipelets*. Multiple such interactions can occur simultaneously without interfering with each other. The gray boxes show different functional domains of which all but the *server* are chosen at the users discretion. The *pipelets* are depicted as a stack of white boxes with typewriter font and are stacked on top of their logical predecessors as described in section 4.2. The remaining white boxes represent further processes involved in the chosen use case, of which the *server* parts are detailed in section 4.3.

4.2 Pipelets

All *pipelets* implement a compact API providing command execution and socket opening capablities. Different classes of *pipelets* each implement one specific task and are instantiable with configurable options. Each class can then modify or override parts of this API, while the base implementation merely forwards calls to its logical predecessor *pipelet*, which is provided during instantiation. Some *pipelets*, usually the last of such a stack, may expose a HTTP endpoint directly to the client. All *pipelet* instances are run on the *server* inside the connection broker. Additionally, each *pipelet* defines user interface components which are needed to query instantiation configuration from the user and expose runtime information to the user.

To illustrate these abstract definitions, a typical stack of *pipelets* is outlined, with the order indicating logical predecessors. In this example a JupyterLab is to be run on a *resource* which is only accessible through a *gateway* host. This is also depicted in figure 2.

1. An `SSH` *pipelet* establishes the SSH connection between the *server* and the remote *gateway*. The credentials are queried from the user as needed.
2. Another `SSH` *pipelet* instance establishes a SSH connection to the actual *resource*. It uses the socket opening API provided by the first `SSH` *pipelet*, which is natively implemented through the SSH protocol. This effectively tunnels through the first *pipelet* by starting a new TCP/IP connection between *gateway* and *resource*. In many cases, this

is not necessary, but in others it enables access to devices that are not directly reachable through the Internet. All following *pipelets* will not be aware that this SSH connection is passing through a tunnel.

3. Then a `Source` *pipelet* wraps the command execution API such that the actual commands are preceded by sourcing the requested scripts. These are present on the *resource*, e.g. a `~/.bash_profile` and set up the environment e.g. by modifying the `PATH` variable as needed. This, too, is an optional step of the given example.
4. Finally, a `Jupyter` *pipelet* starts a JupyterLab instance on the *resource* and exposes Jupyters HTTP server to the client in a proxy-like manner. It internally takes care of the propagation of the security-token, TCP-port, URL-prefix, and other options.

It is important to note, that the *pipelet* implementations are not limited to the above mentioned cases. For example, instead of JupyterLab one could also start TensorBoard, a monitoring and debugging tool for TensorFlow [20], by using the corresponding *pipelet* class.

4.3 Server

The *pipelets* are tied together in the server by the *connection broker*. It exposes a RESTful API to the *client* for instantiating and closing these *pipelets*, and performs cleanups based on inactivity timeouts. Unauthorized use can be prevented by checking for valid authentication tokens, such as signed cookies or JSON Web Token (JWT), while active *pipelets* are isolated through random universally unique identifiers (UUID). This *connection broker* is configured by a list of available *pipelet* classes, which is shared with the *server* during asset bundling in order to provide the proper user interface components to the *client*.

Each user's configuration, i.e. *pipelet* instance configuration and possible runtime information, is stored in a *database* (*DB*) on the server in order to enable persistence across devices. This requires a user *authentication* mechanism, which is designed to support multiple configurable authentication methods, e.g. OpenID and LDAP. Still, entirely anonymous usage can be enabled as well.

Besides the aforementioned API endpoints, the *server* also delivers the static markup and code to the *client*. It should be noted, that all the described functional groups (database, authentication, broker, and static data delivery) are fully separable. They partially share configuration, e.g. token-signing keys, but are otherwise only connected to each other due to the *client*'s communication. Consequently, they can be replicated for load balancing purposes and even replaced by an alternative implementation.

5 Conclusion

The current VISPA software implementation provides users with a browser-based working environment. The ability to flexibly access any SSH reachable resource is an essential feature needed by oftentimes heterogeneous scientific workflows.

Recent developments, in particular the user-readiness of JupyterLab, prompted a reassessment that has led to the restructuring of VISPA. Such external projects are relied upon to provide a good portion of the needed functionality, in particular a basic yet extensible user interface. Here, specialized tools such as foreign data format rendering can be implemented through JupyterLab extensions. At the same time, the VISPA-project is able to focus on developing its distinctive features, the facilitation of the connectivity. At its heart, VISPA follows a modular structure, to the extent that even the connections themselves are separated into so-called *pipelets*, providing flexibility, extensibility, and scalability. Through this, users can access and leverage complex computing environments and even utilize custom applications such as TensorBoard.

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 and the Bundesministerium für Bildung und Forschung (BMBF).

References

[1] M. Erdmann et al., Journal of Physics: Conference Series **1085**, 042044 (2018)
[2] M. Erdmann et al., Journal of Physics: Conference Series **898**, 072045 (2017)
[3] M. Erdmann et al., Journal of Physics: Conference Series **762**, 012008 (2016)
[4] H.P. Bretz et al., Journal of Instrumentation **7**, T08005 (2012), `1205.4912`
[5] *VISPA Web* [software], `https://forge.physik.rwth-aachen.de/projects/vispa-web`
[6] Project Jupyter, *JupyterLab is Ready for Users*, `https://blog.jupyter.org/jupyterlab-is-ready-for-users-5a6f039b8906` (accessed 2018-10-17)
[7] *CherryPy* [software], `https://cherrypy.org/`
[8] *SQLAlchemy* [software], `http://www.sqlalchemy.org/`
[9] *jQuery* [software], `https://jquery.com/`
[10] *VueJS* [software], `https://vuejs.org/`
[11] *ACE* [software], `https://ace.c9.io/`
[12] *RequireJS* [software], `https://requirejs.org/`
[13] *Paramiko* [software], `http://www.paramiko.org/`
[14] *RPyC* [software], `https://github.com/tomerfiliba/rpyc`
[15] *Workshop on Big Data Science in Astroparticle Research*, Aachen, Germany (2017, 2018), `https://indico.scc.kit.edu/event/344`,`https://indico.scc.kit.edu/event/277`
[16] D. van Asseldonk et al., Nuclear and Particle Physics Proceedings **273-275**, 2581 (2016)
[17] D. van Asseldonk et al., Journal of Physics: Conference Series **664**, 032031 (2015)
[18] M. Erdmann et al., European Journal of Physics **35**, 035018 (2014)
[19] *JupyterHub* [software], `https://github.com/jupyterhub/jupyterhub`
[20] M. Abadi et al., *TensorFlow: Large-scale machine learning on heterogeneous systems* [software] (2015), `https://www.tensorflow.org/`