REPORT ON OpenDreamKit DELIVERABLE D2.3

Review on emerging technologies

ERIK BRAY, LUCA DE FEO, VIVIANE PONS, TOM WIESING

Due on	31/08/2016 (M12)
Delivered on	??/09/2016
Lead	Université Paris-Sud (UPSud)
Progress on and finalization of this deliverable has been tracked publicly at: https://github.com/OpenDreamKit/OpenDreamKit/issues/43	

DELIVERABLE DESCRIPTION, AS TAKEN FROM GITHUB ISSUE #43 ON 2016-09-19

- **WP2:** Community Building, Training, Dissemination, Exploitation, and Outreach
- **Lead Institution:** Université Paris-Sud
- **Due:** 2016-08-31 (month 12)
- **Delivered:** 2016-09-??
- **Nature:** Report
- **Task:** T2.4
- **Proposal:** p.39
- **Final report**

This deliverable reviews emerging technologies that did not exist, or were not sufficiently visible at the time the OpenDreamKit proposal was written. Its goal is to inform other Work Packages on technologies that have a potential impact on the achievement of their tasks, and to suggest corrective actions to be undertaken when needed. A secondary goal for this review is to inform the general public about technologies related to OpenDreamKit.

Parts of this deliverable appeared, or will appear, on OpenDreamKit's weblog, and we will keep posting reviews there as new technologies emerge.

CONTENTS

1. Emerging technologies external to OpenDreamKit

This section is about technologies being developed outside of OpenDreamKit.

1.1. Anaconda

Anaconda is a free open source Python-based distribution for scientific computing, powered by the Conda package manager. Anaconda is also the name of a family of derived solutions sold by Continuum Analytics, however here we will only refer to the free source distribution by the name of Anaconda.

Anaconda is already extremely popular; it is thus slightly misleading to include it in a report on *emerging technologies*. However we mention it here because of its impact on OpenDreamKit and on WP3 in particular.

One of the main goals of WP3 is to make Sage and its components easily packaged and distributed. This is to be implemented in deliverables D3.6: "Open package repository for Sage" and D3.10: "Packaging for major Linux distributions". Because of Anaconda's size and popularity, its packages are another obvious target for OpenDreamKit. WP3 is already exploring the possibilities to make Sage and its components available as Anaconda packages. For details and reviews of other packaging and build systems, see the notes about Sage packaging taken at Sage Days 77.

1.2. Docker for Mac and Windows

Docker is an emerging technology for packaging software in so-called containers that can run processes isolated from the rest of the operating system. It is the midway point between a complete virtual machine for a single system and a process running with limited user rights.

Docker provides so-called software images that package software. These images can then be executed inside a docker container. Furthermore, Docker provides a service called DockerHub that allows users to upload and publish their own images. This allows developers to bundle their software and distribute it easily. Furthermore, since each container runs isolated from the rest of the systems, the developers do not have to rely on any kind of other system configuration.

Within the OpenDreamKit project this allows us to bundle mathematical software systems (such as GAP, Sage, etc) and distribute them in a manner that is both accessible to users and easily maintainable for developers (see D3.1: "Virtual images and containers"). Furthermore the isolated aspect of docker containers allows us to easily integrate multiple systems together without having to make additional assumptions about the user's specific setup — we can just run all systems in one docker container. This way users can install entire mathematical software stacks that the OpenDreamKit project aims to provide easily.

Docker was originally a Linux only application — it relied on a lot of functionality provided by the Linux kernel. To make it available on Windows and Mac the Docker developers provide a virtual machine, called Docker Toolbox, that runs a streamlined Linux system with Docker pre-installed. This makes it possible for Windows and Mac users to run docker containers; however it introduces an additional layer of abstraction that comes with some disadvantages. The additional virtualisation slows down docker containers and faces technical limitations when wanting to integrate with the host system. It also requires users to install virtualisation software before being able to run any kind of Docker Image. Even though Docker Toolbox automatically installs VirtualBox, this is a separate application that adds load to users machines.

Recently Docker started to build native versions for Windows and Mac. These versions do not rely on Linux functionality — instead they leverage functionality provided by Windows and OS X operating systems natively. In terms of the OpenDreamKit project these are a big step in terms of usability — they make it significantly easier for users of such systems to run a Docker container. Users can now install Docker just as they would install any other software on their machine. Among speed and resource advantages, these versions will make it easier

for developers to create docker-powered applications and Docker containers because of better integration between host system and containers; for example the file system of the physical machine can be mounted inside containers more easily. As a side note, the Windows version of Docker only works on Windows 10 Professional and Enterprise editions and requires some manual configuration — a setting in the BIOS has to be changed (for more information see Docker for Windows — Getting Started Documentation). This is much less effort than was required previously, however it is not quite ready for adoption by inexperienced users yet. The Docker developers have stated clearly their intention to make the Docker experience in Windows and OSX as easy and streamlined as it is for Linux. When the users will be able to run Docker without the need for manual configuration or a high-end edition of Windows, we expect many components of OpenDreamKit to be avaible via Docker containers on Windows and OSX as easily as they are now for Linux.

Docker for Windows and Mac has been in a private Beta since March 2016 and has recently become available as a public Beta. Interested readers can find more information on the Docker Blog.

When this technology reaches maturity, it will impact deliverables D3.1: "Virtual images and containers" and D3.7: "One-click install SAGE distribution for Windows with Cygwin 32bits and 64bits". D3.1 is already delivered; the recommended action for its continuous maintenance is to add Windows and Mac Docker containers to the ones already distributed. The recommended action for D3.7 is to reconsider Cygwin as a platform for distributing one-click installs of SAGE on Windows; this recommendation has already been enacted: progress on an experimental Docker-based installer is being tracked at `https://github.com/sagemath/docker-images/issues/1`.

1.3. Windows Subsystem for Linux

One of the goals of the OpenDreamKit project is to improve support for open source mathematics software on a wider range of hardware platforms and operating systems (see **T3.1**). Among the largest portability challenges is improving installation and operation of such softwares on Microsoft Windows—still the dominant OS in many user communities, especially on desktop and laptop computers. Despite there being many large communities of Windows users, most open source software developers have traditionally preferred UNIX-like software development environments. The UNIX environment differs in many significant ways from Windows, such that support for Windows has often been neglected by those developers.

1.3.1. *Introducing Windows Subsystem for Linux.* In late March of 2016, at its annual developers' conference, Microsoft announced a surprising new technology. Dubbed Windows Subsystem for Linux (WSL), this new feature premiering in the Windows 10 "Anniversary Update" would add a Linux system call compatibility layer to the Windows NT kernel, and a Windows-native port of the popular "bash" shell. Furthermore, in partnership with Canonical – creators of the popular Ubuntu – Linux distribution, the WSL supports Ubuntu's "apt" package repository, giving Windows users access to a large swath of open source software built for Ubuntu, but running directly on Windows.

In short, what this means, is that Windows users will now have a Microsoft-supported Unix-like shell environment, and the ability to run Linux-based software directly on Windows, without a virtual machine. This would have been unthinkable to most even a decade ago.

1.3.2. *Why porting UNIX software to Windows is hard.* Software that is compiled from languages like C and C++, often favored by researchers, is generally built in such a way that the compiled *binaries* support a specific operating system. Each OS has a particular *binary format*—the way the program is organized on disk and copied into memory at runtime. So any compiled software built for that OS has to be arranged in the binary format for that OS in order for the OS to know

how to interpret and execute it. It is not typical for one OS to be able to understand binaries for another OS. For example, software built for Linux uses the ELF binary format; normally if one tried to run a program built for Linux on Windows, which only understands the PE format, it will not be recognized as a valid executable.

An even deeper complication to writing portable software is the system calls— software run by users interacts with the operating system to perform low-level operations such as writing to disk, or making network connections, through special functions provided by the operating system called "system calls". Modern UNIX-like operating systems follow, to an extent, the POSIX standard for system calls, allowing them to be generally more interoperable. Windows, on the other hand, has its own system call definitions that are not necessarily in one-to-one correspondence with POSIX system calls. As such, a program built for Linux has no idea how to communicate with a Windows operating system.

This can be a problem even on higher-level interpreted languages like Python. Although writing code in Python abstracts away most operating system differences, Python code *can* still access OS-specific features such as system calls, and this is sometimes necessary to access more advanced OS features needed by some scientific software. So Python code that uses Linux-specific features, for example, can only run on a version of the Python *interpreter* built for Linux.

A third difficulty has to do with minor differences in user interface standards. For example, a common issue in Windows support is its different standard for representing file paths. While Windows paths contain a "drive letter" and uses the backslash ("\") to separate between folders (e.g. `C:\Windows\cmd.exe`), UNIX-like systems have no concept of a "drive letter", and use forward-slashes ("/") (e.g. `/bin/bash`). Issues like this can cause many small, but pervasive bugs when porting software between operating systems.

1.3.3. *How WSL gets around it.* The Windows Subsystem for Linux does two main things:

(1) It enables the Windows NT kernel to understand the ELF binary format, and *translate* it, as closely as possible, to the binary format used by Windows.
(2) It implements a sizeable subset of the POSIX system call standard on top of Windows. Although Windows' own system calls do not map directly the POSIX, because Microsoft has access to how its underlying operating system is implemented, they are able to implement the POSIX interface on top of the lower-level details of their NT kernel.

WSL also provides its own *bash* shell—a command-line interface favored by many users of Linux. This provides a UNIX-like command-line interface within Windows, also has an underlying system for transparently translating things like file paths between the Windows and UNIX formats.

The ultimate goal is to be able to take a program compiled and built on a Linux system, copy it over to Windows, and allow it to run without any modifications, with all the system-level translations completely transparent to the user. Targeting Linux software *specifically* makes this possible, because the system interfaces it will use are well-specified and *predictable* in most cases. This is as opposed to running a virtual machine, in which an entire separate operating system is run in order to run software on that OS, and which needs to be able to run any arbitrary OS.

This is direct support for Linux software in Windows itself—there is no virtualization.

This is also an improvement over previous efforts at supporting Linux software on Windows, such as Cygwin. Because Cygwin is third-party software it cannot modify the Windows NT kernel itself. It does not support ELF binaries: to run some software with Cygwin it has to be *recompiled* to the native PE binaries understood by Windows. It also does its best to provide emulation of POSIX *system calls*, but it has to do this by building them on to of the NT system

calls which, as noted above, is not a one-to-one mapping. WSL, on the other hand, provides support directly from the operating system for POSIX and other Linux system calls.

1.3.4. *What it means for OpenDreamKit.* Because WSL allows binaries built for Linux to run directly on Windows, it makes much of the enormous repository of software built for Ubuntu (and potentially other Linux distributions) immediately available to run on Windows. No recompilation has to be performed or anything. At least, that is the goal—as we'll see below it is still not fully realized.

For example, Ubuntu's software repository already includes builds of many of the packages that are central to OpenDreamKit, such as GAP, PARI, and some smaller packages including many of the dependencies of SAGE. SAGE itself has an unofficial Ubuntu package—this has been found so far to nominally "work" on WSL, but there have been found to be many bugs. That said, a great deal of other mathematical software—especially that which is less dependent on OS-specific features, should already work out of the box.

An additional potential advantage for WSL (indeed, one of the project's goals as detailed in an Ars Technica article[1]) is to make the development tools and command-line interfaces favored by UNIX-oriented developers available on Windows. This makes it possible, in principle, to develop software like SAGE the same way on both Windows and Linux.

In some sense this could be an end-run around OpenDreamKit's goal of better supporting Windows—Microsoft has already done the lion's share of the work for us. But there is more to be done, and it may not be an end-all be-all solution.

1.3.5. *Caveats.* As mentioned in the previous section, while some OpenDreamKit software has been found to work in WSL, it is not without issues. Many bugs were found in running SAGE on WSL (and even more when trying to compile it). This is not unexpected—the current release is marked "beta" by Microsoft, and they fully acknowledge that it is buggy and incomplete.

Second, Microsoft has made it clear in several statements[2] that the WSL and "Bash for Windows" are to be considered tools for developer convenience *only*. It is not intended for use in a server infrastructure nor, presumably, as a means of distributing/installing software for end-users (i.e. who are agnostic about how the software is implemented). Although one could take the cynical view that this just Microsoft's way of protecting its own server products, there are also some practical reasons for this:

(1) As a developer tool, the WSL + Bash for Windows are not easy for casual users to install. First, it is only available on Windows 10 with the recent (as of writing) "Anniversary Update". Not all users are on Windows 10 yet. It also requires having an account on Microsoft's developer network, and for their Windows to be configured to "developer mode" in order to receive development-related updates, plus a few extra steps. This can also involve some sizeable downloads. This is not especially onerous for a developer, but is not a series of steps that can or should be asked of the "casual" or first-time user just to install some software.

(2) Despite having support directly in the kernel, the WSL is something of a walled garden. It is not possible to run native Windows applications from within the Windows *bash* prompt. Nor is it possible (in any transparent sense) to interact with Linux applications from native Windows applications. This is probably required, on some level, to maintain a clean abstraction.

Finally, it is not currently supported to run GUI applications on top of WSL, in part because that requires a lot more than just system call compatibility. While not supported officially

[1] http://arstechnica.com/information-technology/2016/04/why-microsoft-needed-to-make-windows-run-linux-software/

[2] https://blogs.windows.com/buildingapps/2016/03/30/run-bash-on-ubuntu-on-windows/

by Microsoft, some hobbyists have made progress on it though, by integrating with existing X server implementations for Windows[3]. For many mathematical softwares this is a non-issue—they are text based: numbers in; numbers out. Additionally, graphical interfaces for interactive research environments are increasingly moving to the web (see for example JUPYTER or SAGEMATHCLOUD). In such cases the GUI elements have been moved out to the web browser and the backend typically runs "headlessly": it has no reliance on the system's desktop interface.

1.3.6. *Conclusion.* The Windows Subsystem for Linux represents a major step in the right direction for Microsoft. It shows that they are listening to the needs of the broader software developer community (not just those who work exclusively on Windows) and that they have some interest in cooperating with the open source software community (this has also been demonstrated in several other ways in recent years).

For the purposes of OpenDreamKit, this work will make *development* of open mathematical software more accessible to a wider community. Although this may not improve accessiblity for casual end-users, many users of open research software tend to become *de facto* developers as well, as the more they use the software the more interested they become in modifying it for their own purposes. Making it possible for Windows users to do development on otherwise UNIX-oriented software, without leaving their personal desktop environments, is appealing. Being able to compile one's own software is also important for some highly optimized numerical software, which tunes itself at compile time to the computer it is being built on, sometimes with dramatic results.

In conclusion, we do not recommend any corrective action on OpenDreamKit's deliverables. Although WSL does not yet provide a fully reliable immediate solution for porting OpenDreamKit software to Windows, we encourage partners involved in **T3.1** to keep an eye on it as it evolves.

1.4. Cap'n proto

Cap'n proto is a multi-language serialization protocol and toolkit, providing *zero-cost encoding/decoding*. Cap'n proto is an open source project of Sandstorm.io.

Cap'n proto works by fixing a portable, efficient, memory layout for its data structures. This way, data can be serialized and transferred simply by coping the raw data in memory. Cap'n proto ships with an official C++ implementation, and many contributed implementations in other languages (notably C, Python, Java, ...). Each implementation gives access to Cap'n proto data structures through the language's native APIs, thus abstracting away all the protocol's complexity and making data access very efficient.

Serialization is a core component of any complex system. It allows communication *inter-process*, *inter-node*, and across time. The specific design of Cap'n proto has the potential to allow even *inter-language* shared-memory communication inside the same process, something that is usually done in a language-dependent and non-portable way.

Because of its focus on efficiency, Cap'n proto has potential applications in WP3 and WP5. We recommend the partners involved in these work-packages to closely follow this technology.

1.5. Binder

Binder is a free cloud service that lets users define a *running environment* (e.g., by a Docker file, a Python `requirements.txt` or a Conda environment), and obtain a link to a cloud instance of the running environment, together with a Jupyter frontend.

This is similar to the tmpnb service, except that the repository owner defines the running environment. The computing power for the default instance is provided by The Freeman Lab at

[3] http://www.pcworld.com/article/3055403/windows/windows-10s-bash-shell-can-run-graphic
html

HHMI Janelia Research Campus to support open science. However Binder is open source and could be deployed elsewhere.

Relevance to OpenDreamKit. Binder is a promising approach to make it as easy as possible for our users to share publicly their Jupyter notebooks. OpenDreamKit could help by:

- Contributing ready-to-use environment descriptions for our favorite sofware (GAP, ...) Docker containers developed in D3.1 are probably a good starting point.
- Finding infrastructure support (universities, EGI, ...) to run more instances of Binder.

2. GÉANT Open Education Resource project

The goal of D2.7: "Community-curated indexing tool (open source)" is to provide a (open source) community-curated indexing tool for resources (documentation, tutorials, courses, notebooks, ...) related to mathematical software.

eduOER is a searchable, metadata-driven, multilingual, indexing service for educational multimedia content. The eduOER service is being developed by the "Real Time Communications and Multimedia Management" service activity of the GN4-1 project partly funded by the European Commission. Its alpha version was released in March 2016.

There is a clear overlap between the goals of D2.7 and eduOER, which could justify offloading the contents of D2.7 to eduOER. However, there are also some major differences:

- Role: eduOER is an aggregator (it aggregates metadata from third-party repositories), D2.7 is a repository (of URLs + metadata).
- Content: eduOER is audio-video only (although support for other contents may be added in the future), D2.7 is text-oriented (although any format is supported in principle).
- Metadata: eduOER aggregates metadata from participating repositories, metadata in D2.7 is user-generated.
- Search: eduOER offers search on metadata, D2.7 requires metadata and full-text search. However eduOER can perform full-text search if full-text-extraction is provided as metadata.
- Social: eduOER has no social interaction (however the frontend component of eduOER has some planned social features). D2.7 is community curated in the sense that entries are reviewed, commented and scored by humans.

We recommend that the partners involved in D2.7 keep surveying the state of eduOER, in view of a possible partial or total adoption of the technology, or at least in view of interoperability.

3. Emerging technologies internal to OpenDreamKit

This section being about technologies developed internally by OpenDreamKit, its goal is mainly to inform the general public. However, some task leaders may get some useful insights on technologies they have only been following from a distance.

3.1. **JupyterLab**

At the SciPy 2016 conference, Brian Granger and Jason Grout presented the next generation of the Jupyter Notebook application: JupyterLab. The presentation was followed by a post on Jupyter's blog[4]. JupyterLab is in pre-alpha stage, and is available on GitHub[5].

[4] http://blog.jupyter.org/2016/07/14/jupyter-lab-alpha/
[5] https://github.com/jupyter/jupyterlab

3.1.1. *What is JupyterLab?* JupyterLab captures a lot of what has been learned from the usage patterns of the Notebook application over the last 5 years and seeks to build a clean and robust foundation that will offer not only an improved user interface and experience, but also a flexible and extensible environment for interactive computing.

Today's Notebook application includes not only support for Notebooks but also a file manager, a text editor, a terminal emulator, a monitor for running Jupyter processes, an IPython cluster manager and a pager to display help.

But the underlying code is not the cleanest to extend and providing a more responsive and flexible UI atop it is difficult. JupyterLab is a next-generation architecture to support all of the above tools, but with a flexible and responsive UI that adapts easily to multiple workflow needs, thanks to its user-controlled layout that ties together many tools under a single roof. The entire JupyterLab is built as a collection of plugins that talk to kernels for code execution and that can communicate with one another.

3.1.2. *JupyterLab in OpenDreamKit.* The way JupyterLab is being built enables building different applications, such as making other non-notebook webpages (e.g. documentation) interactive. This fits the OpenDreamKit philosophy perfectly: rather than building one unique VRE, JupyterLab ships a modular environment to build VREs tailored to each user's needs.

JupyterLab has the potential to be the *one-size-fits-all* standard for graphical user interfaces delivered by OpenDreamKit. Given the very large projected user base of JupyerLab, this will make adoption OpenDreamKit products easier for end-users. WP4 (user interfaces) will be especially involved in integrating JupyterLab into its demonstrators.

We strongly encourage all partners working on delivering a fully integrated VRE, such as SAGEMATHCLOUD (see next section), to keep assessing the maturity of JupyterLab, and its potential to replace their currently planned UI.

3.2. SAGEMATHCLOUD

Part of OpenDreamKit's mission is to work on user-interfaces for better collaboration and also component architectures. This is why the SAGEMATHCLOUD platform is of special interest for us. The goal of **T3.6** is even to have a deeper look into its code base.

3.2.1. *What is* SAGEMATHCLOUD*?* SAGEMATHCLOUD is an online platform which allows the creation of **collaborative scientific projects** including many scientific softwares and tools like SAGE, Jupyter, SciPy, Julia, Latex, and more.

Its codebase is open-source, distributed under the GNU General Public License. The platform is run by a private company (SageMath Inc.) created by William Stein who is also the initiator of the SAGE software. The platform offers both free and paying premium accounts.

Projects. The main tool of the SAGEMATHCLOUD platform is the possibility to create **projects** from which you can access the many features. A single user can create as many projects as needed. Each project is an **independant Linux virtual machine**. It thus comes with a full file system and an **online terminal** that allows you to run Linux commands. The storage of each project is limited by default but can be extended on premium accounts. You can access the files through the SAGEMATHCLOUD web interface or also through ssh.

One key feature is that each project can be **shared by multiple users**. This allows sharing access to the files and also **real time editing** though the platform. Single files or folders can also be made **public**. A link is then provided which allows either viewing or downloading the files (even without a SAGEMATHCLOUD account) and also an easy way to copy onto a different SAGEMATHCLOUD project owned by the viewer.

Softwares. When you create a SAGEMATHCLOUD project, your Linux virtual machine comes with many softwares and tools especially useful for mathematicians and scientists in general. We list here the most important ones.

- SAGE and SAGE **worksheets**. As the name indicates, the platform was primarily developed as a replacement for the old SAGE notebook server to allow collaborative online work using SAGE. The SAGE software is of course installed by default on the virtual machine and one can run SAGE through the online terminal. The platform also offers its own SAGE **worksheet filetype** to edit and run SAGE code in a cell-type system (as in the Jupyter notebook or the old SAGE notebook) mixed with other cell types like text and HTML. This is used to create interactive worksheets that can be easily shared and copied.
- **Jupyter**. SAGEMATHCLOUD includes a Jupyter notebook interface with many kernel options (Python 2, Python 3, Anaconda, SAGE, R, Julia, and more). On top of the usual interface, SAGEMATHCLOUD's Jupyter offers **real time** synchronization among multi users.
- **Latex**. The common document preparation system Latex is installed on the virtual machine. It also offers a multi user editor with real time synchronization and a dual view of both the Latex source code and pdf output.

3.2.2. *Sharing and teaching with* SAGEMATHCLOUD.

Accessibility. The great advantage of SAGEMATHCLOUD is that it offers a **complete scientific environment** without the usual setting up hassle. It makes the different software very easy to access independently of the user personal system as long as there is an access to a good Internet connection. As an example, a mathematician can share a demo of code (in a Jupyter or a SAGE notebook) that could be used directly by its collaborators. Of course, the Internet access is itself a limit. Given poor network access, for example but not only in some developing countries, the latency can considerably reduce the usability of the system.

Teaching. When teaching is concerned, the sharing facilities of SAGEMATHCLOUD come very useful. Moreover, the platform offers a course managing system. The principle is as follows: the teacher has acces to a "main project" containing the class material; every student has its own project which is shared with the teacher. The course management system allows for automatic actions like:

- Create all the student projects where the teacher is automatically added as a collaborator.
- Create assignments by copying some material from the main project to the students projects.
- Collecting, grading, and returning assignments by copying back and forth between the students projects and the main project.

An *assignment* is just a folder. It can have multiple content depending on the class. Of course, the system is especially interesting when the assignment is given within an **interactive worksheet** and can then be achieved by the student directly on the interface. SAGEMATHCLOUD then becomes a very good interface to initiate students to the many scientific softwares it offers.

3.2.3. SAGEMATHCLOUD *and OpenDreamKit.* The many features of SAGEMATHCLOUD make it a very interesting project for OpenDreamKit to look at. Indeed, it offers one of the leading technologies for scientists in terms of cloud project management, teaching and sharing facilities. In particular it showcases a collection of **features** that have been selected and adopted by a wide community.

It also has some limits which we would like to address through our project:

- **Accessibility**. As previously mentioned, the cloud based interface can not be easily accessed in places where the Internet connection is not good enough. One solution would be to have clear easy-to-follow instructions on how to install a SAGEMATHCLOUD platform in a local institution or on a personal machine. This is to be taken care of in D3.2: "Understand and document SAGEMATHCLOUD backend code." and D3.4: "*Personal* SAGEMATHCLOUD: single user version of SAGEMATHCLOUD distributed with SAGE."
- **Interoperability and file formats**. At the moment, the SAGEMATHCLOUD platform offers two file formats for interactive worksheet: the Jupyter one and a home-made SAGE worksheet one. It is not possible to run the SAGE worksheets elsewhere than on the platform. Especially, there is no way to run a SAGE worksheet on a local SAGE installation. It is not yet clear what a long term unified worksheet solution would be and it is part of the OpenDreamKit project to work on this question. The technical choices made for the SAGE worksheets are interesting to investigate in this regard, as well as, file conversions and so on.