Project Acronym: MuG
Project title: Multi-Scale Complex Genomics (MuG)
Call: H2020-EINFRA-2015-1
Topic: EINFRA-9-2015
Project Number: 676556
Project Coordinator: Institute for Research in Biomedicine (IRB Barcelona)
Project start date: 1/11/2015
Duration: 36 months

Deliverable 5.2: Computational infrastructure components implementation

Lead beneficiary: Barcelona Supercomputing Center (BSC-CNS)
Dissemination level: PUBLIC

Due date: 01/11/2017
Actual submission date: 01/12/2017

Document history

Version	Contributor(s)	Partner	Date	Comments
0.1	Josep Ll. Gelpí	BSC	23/11/2017	First draft
0.2	Javier Alvarez Laia Codó Dmitry Repchevsky	BSC	29/11/2017	Second draft
1.0			1/12/2017	Approved by Supervisory Board

Table of contents

1. EXECUTIVE SUMMARY

MuG Virtual Research Environment should provide the members of the 3D/4D genome community with an adequate combination of relevant information, data, and computational tools. The combination should help, with a friendly access, the researcher to analyse data, either from repositories, or obtained from experiment or simulation; combine and compare such analysis results with related studies and reference data.

MuG VRE prototype was presented in Sept 2016, and described, together with all design considerations in D5.1. This document describes the implementation of the software components in the first beta release of MuG Virtual Research Environment portal (https://vre.multiscalegenomics.eu). In brief, the portal in based in a central workspace that allow the user to find together data and tools related to research operations in 3D/4D genomics. User is offered a series of tool and visualization options and may analyze together data coming from different levels of the 3D/4D genomics ecosystem. The portal backend is responsible to channel the analysis or simulation operations to the appropriate infrastructure, manage the execution, and collect the results back to the workspace. MuG VRE is implemented in two cloud systems at IRB, and BSC premises, and was presented last 15th November 2017, in the conference "Multidimensional Genomics: The 3D/4D organization of chromatin" and is open to users and developers. The document is organized as follows: Section 3 will recall the design guidelines and highlight the most relevant improvements; section 4 describes in detail the implementation of the software components and the state of the infrastructure; section 5 details of the present offer of data and applications; and a glimpse of expected improvements during the last year of the project (section 6). Usage policies both for users and developers are summarized in section 7, and finally additional information as usage statistics recovered so far, addresses for software repositories, and data types and formats understood by MuG VRE, are included in the Annexes section.

2. INTRODUCTION

3D/4D genomics community is a highly heterogeneous community where researches focus their work in a specific scale of the problem without usually accessing to the others. The main reason for such situation is the heterogeneity of data types and tools (see D3.1 for a more formal discussion). MuG Virtual Research Environment has been designed to cover this heterogeneity with a common infrastructure that allow users to work at their respective level of expertise but also provide a seamless access to the other levels with the necessary degree of integration among data and tools. In summary, MuG VRE puts together data coming from atomistic simulations, genome annotation, middle and high scale 3D genomics, and cell biology imaging data, and establishes the necessary relationships among the different levels to build an integrated view of the biological phenomena under study. The computational infrastructure should assure interoperability of analysis tools and generate an integrated environment with a seamless transition among the available data levels. The design of MuG VRE has been split in several components, a 3D/4D browser (WP3), a data infrastructure (WP4), and a collection of interoperable analysis tools (WP6), all components supported by a computational infrastructure (WP5). MuG VRE computational infrastructure described here has the mission of managing the above components, and integrate them in a single user environment, assuring the best efficiency in data mobilization and process. The chosen strategy (see D5.1), will allow VRE members i) browse the available data in an integrated way, ii) incorporate raw data to the VRE that will perform the appropriate analysis,

and incorporate results to MuG's data repository, iii) use the VRE as an analysis infrastructure using the available tools on existing or uploaded data, and iv) download data in the appropriate formats for in-house further analysis.

MuG VRE infrastructure design was originally described in D5.1. The initial prototype has been active since Oct 2016, and has been used as a test bed to develop software components and developments produced in the project and the protocol and components for the integration of analysis and simulation tools. After this period, the original design has been reconsidered and updated, and the components of the infrastructure following the final design have been implemented as MuG VRE first beta version, and released to the community (15th Nov 2017). We present here details of such implementation, the present state of the infrastructure, and the roadmap of evolution in the last year of the MuG project. MuG VRE is available at http://vre.multiscalegenomics.eu .

3. UPDATE OF COMPUTATIONAL INFRASTRUCTURE INITIAL DESIGN

MuG computational infrastructure has been designed to fulfil the following principles (taken from D5.1):

1. Flexible environment, able to adapt to the specific needs of the analysis tools (from WP6), both in terms of software requirements, or computational resources.
2. Software scheduler(s), able to manage analysis workflows, and computational resources in a transparent and adaptable manner. This will be an elastic infrastructure with automatic adaptation to user loads.
3. Multi-scale execution. Analysis workflows could be executed either at the cluster level, in HPC environments, or distributed infrastructures like EGI, and eventually in the forthcoming European Science Cloud (EOSC) ecosystem.
4. Web-based access centered in the MuG multi-scale browser (designed in WP3). This will be complemented by programmatic access using well-established interfaces including Galaxy. User access will integrate the Authentication and Authorization Infrastructure being designed within the Elixir initiative.
5. The infrastructure will be eventually interfaced to European e-infrastructures, including the EGI for computation, and EUDAT for shared storage.

Figure 1 shows a general schema of MuG VRE infrastructure. The original design was largely maintained and most changes constitute a refinement of the implementation based in the upgrade of the software components. In particular the following updates are worth to be mentioned here. Full details of the improvements will be described in the following sections.

- User workspace has been re-structured. User workspace constitutes the organization center for the complete activity on the MuG VRE. The workspace is now presented as a collection of analysis projects. This makes easier the access to the data and results, and allows to intuitively filter the workspace contents and provide integrated presentations of the analysis results.
- PMES software scheduler has been rebuilt. PMES can now be controlled through a REST interface. This simplifies the interaction between the workspace backend and the PMES scheduler, allowing both systems to be physically separated. This is relevant as it opens the possibility of remote scheduling of tool's executions and makes possible to evolve to a truly

distributed VRE. Also PMES can now fully replace the use of traditional queuing systems (like SGE) to manage execution demand.

- Building of the Virtual Machines have been improved to make then usable in different cloud infrastructures.
- A protocol for the integration of tools in the VRE has been designed (see section 5.2). Python based skeletons for new tools are now available, what simplifies the addition of new tools, and also makes easier the communication of such tools with the workspace, as all tools share a common interface to communicate with the VRE workspace.
- A data management plan (DMP) has been put in place (see D4.5). Data management inside MuG VRE is being updated to the new protocol. Once completed MuG workspace will be available through a uniform REST API, shared by all MuG components. This will again simplify data transmission and will empower the distribution of the workload among several cloud systems, and the availability of MuG's data to third party
- User authentication have been derived to a centralized server based on Keycloak [1] software, allowing to access to VRE using a variety of identity providers.

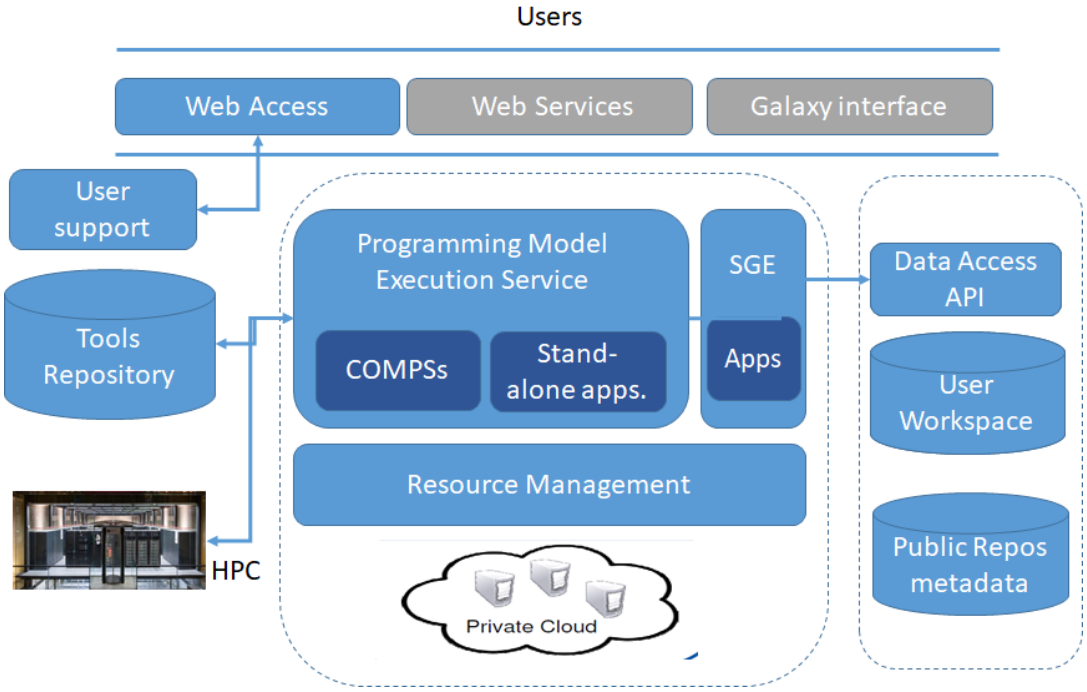

Figure 1. Layout of MuG's computational infrastructure

4. PRESENT IMPLEMENTATION OF MuG SOFTWARE COMPONENTS

The following section describes individually the implementation of software components used the initial installation and their specific function.

4.1. MuG Cloud deployments

MuG VRE infrastructure has been designed as a fully virtualized environment. This layout allows to deploy new instances of the VRE Backend in new cloud infrastructures with minimal overhead. Besides, tools deployed as virtual machines, allows to configure an elastic infrastructure, to cover peaks of demand, or to configure complex workflow schemes. MuG VRE has been deployed in two cloud infrastructures based on OpenNebula [2] (IRB and BSC), and the KVM hypervisor [3] (see Table 1). Additionally, a small instance at the EMBL-EBI's Embassy cloud has been deployed for testing purposes.

The generation of Virtual Machines has been adapted to make them compatible with the deployment in both openNebula and openStack [9] cloud managers, allowing their use in a wider set of cloud platforms, including Elixir Compute Platform and EGI providers.

Table 1. Present deployments of MuG-VRE

Institution	**Cloud infrastructure**	**Specifications**	**Deployed software**
IRB	OpenNebula	84 core, 1,5TB RAM	Production VRE Development VRE
BSC	OpenNebula	96 core, 1TB RAM, 90 TB storage	Development VRE Authentication VM
EMBL-EBI	OpenStack	16 core, 64 RAM, 1 TB storage	PyCOMPS testing VMs Selected tools

4.2. Process management

4.2.1. Sun Grid Engine queuing system / oneflow

Sun Grid Engine (SGE) [4] was designed to manage distributed software executions in heterogeneous computational environments. SGE is used normally in cluster based infrastructures as a general process scheduler. Capabilities of SGE include, among other, resource management, remote execution, parallel execution management, interactive processes, monitoring and accounting, integration with Amazon EC2 or Hadoop. MuG VRE backend uses SGE to manage applications where no complex workflows are necessary, although peaks of demand requiring the deployment of additional workers may be expected. To adapt to MuG general infrastructure (Figure 1), a specific connection with OpenNebula cloud manager has been set up through the use of oneFlow [5], a component of the OpenNebula framework that allows managing Multi-VM application and auto-scaling. Figure 2 shows a schema of the structure implemented in MuG VRE.

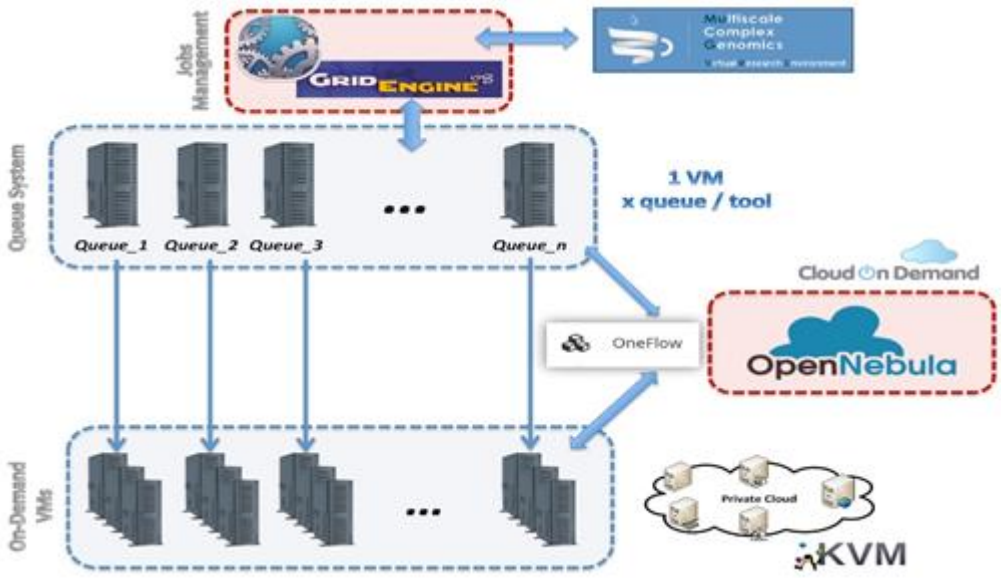

Figure 2. Layout of the integration of Sun Grid engine in MuG computational infrastructure

Each VRE tool execution is send to a separated SGE queue populated with multiple instances of the VM where the tool implementation is encapsulated. The availability of these instances is controlled by oneFlow, who dynamically deploys them according to a set of configurable system metrics like the waiting time of the jobs, or the VM load. In this way, SGE queue workers can automatically grow or shrink on demand, with the only restriction of having at least one VM already deployed and ready to accepts jobs in each SGE queue.

4.2.2. COMPSs programming model

COMPS Superscalar (COMPSs) [6] is a programming model and runtime designed to simplify the development and execution of distributed applications. COMPSs applications are programmed in a completely sequential manner, but contain code annotations that identify certain methods as tasks that can be executed in a remote location. Using these annotations, COMPSs runtime is able to automatically detect and exploit the inherent parallelism of the application, and to execute it on various distributed platforms, such as Grids, Clouds, and clusters.

COMPSs runtime implements a master-worker architecture that can be seen in Figure 3. Master and workers are processes that can run on different virtual machines (VM) or physical nodes depending on the characteristics of the underlying infrastructure. COMPSs runtime manages the available computational resources in a completely transparent manner and, in the case of elastic infrastructures such as Clouds, the runtime can dynamically create and destroy workers to tailor the computational capacity to the application workload.

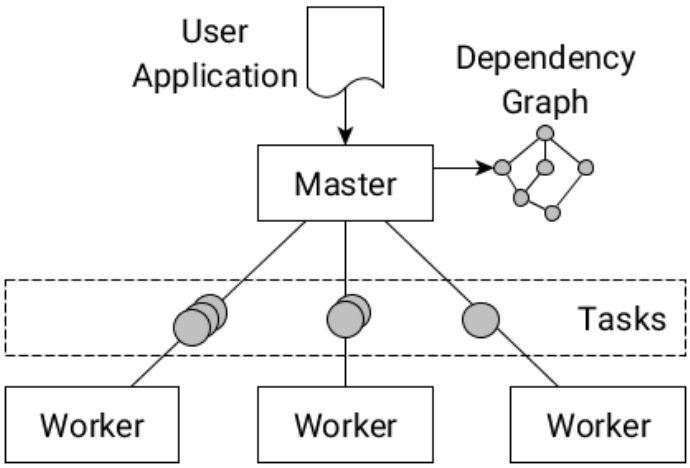

Figure 3. COMPSs master-worker architecture.

COMPSs runtime is based on the Java programming language. However, COMPSs also supports C/C++ and Python applications through bindings. In the context of the MuG project, users mainly employ the Python binding (also known as PyCOMPSs) due to their familiarity with this programming language. COMPSs applications consist of a main program and a set of annotated methods. The main program is the entry point of the application and is executed by the COMPSs master, whereas annotated methods are executed remotely by workers. The main program of an application is executed sequentially, and the master generates and stores a task object every time that it encounters a call to an annotated method. Task objects consist of the annotated method code, and a description of its input and output variables. These variables define the data dependencies between tasks and thus the order in which tasks can be executed. For example, if task *T1* writes a variable that is read by task *T2*, we say that there is a read-after-write dependency between *T1* and *T2* that forces *T2* to be executed only after *T1* has finished. Task objects are stored in a directed acyclic graph, called the *dependency graph,* where nodes represent tasks and edges represent the dependencies between them. As tasks become dependency-free, they are scheduled for execution in an available worker. The scheduling algorithm maximizes data locality by allocating tasks where their input data is stored whenever possible. However, if a task cannot be executed where its input data is located, the necessary data transfers are performed between workers before task execution.

4.2.3. Programming Model Enactment Service

The Programming Model Enactment Service (PMES) [7] controls the execution of jobs in an underlying Cloud platform through an Open Cloud Computing Interface (OCCI) [8] Server (Figure 4). The PMES offers a REST interface with four main operations to manage jobs:

- `createActivity`: to launch new jobs
- `terminateActivity`: to cancel one or more jobs
- `getActivityStatus`: to get the status of one or more jobs
- `getActivityReport`: to obtain a report of one or more jobs

The PMES supports two types of jobs: single and COMPSs jobs. Single jobs consist of the execution of a single command on a VM, while COMPSs jobs involve the execution of a COMPSs application using one or more VMs. Single jobs provide an easy way of running already existing applications in the Cloud, while COMPSs jobs allow for the execution of large parallel workflows. In the case of single jobs, the PMES manages the only VM employed, whereas in the case of COMPSs jobs, the PMES manages the COMPSs master VM, and the COMPSs runtime creates additional worker VMs if necessary.

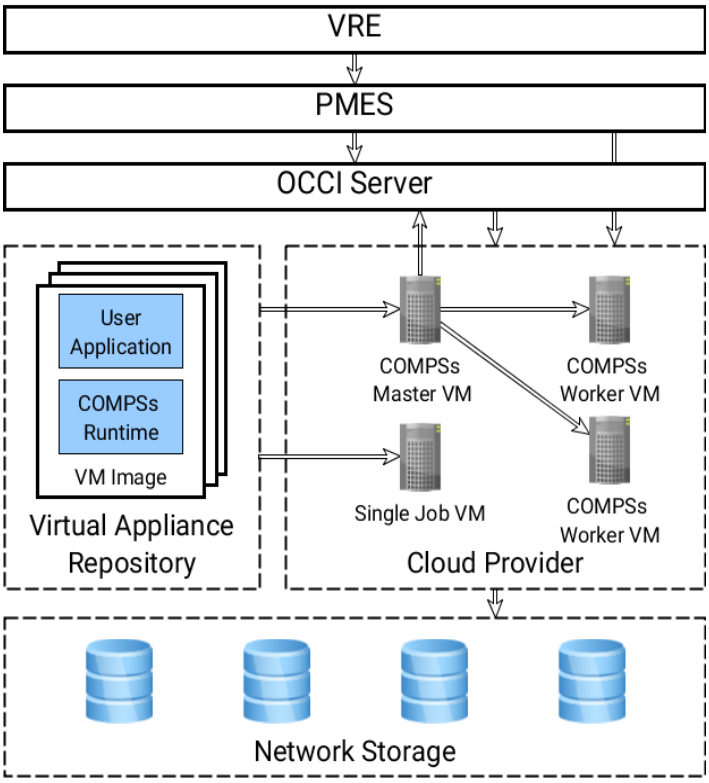

Figure 4. Overview on the PMES execution infrastructure.

Images to create VMs are obtained from a Virtual Appliance Repository. Each of these VM images contains all the binaries and libraries necessary for running a specific user application (or set of applications). In this manner, binaries and libraries do not need to be installed every time that a job is executed. In the case of COMPSs applications, the VM image also contains the COMPSs runtime.

VM images do not contain any application input data, as this is dynamically read from a Network Storage system accessible from all VMs. In Mug implementation this storage system consists of a private partition where users can store sensitive data, and a public partition where users can make data available to others, or data from public repositories can be cached. Both input and output application data is read and written from the Network Storage so that costly data transfers are avoided.

PMES job life cycle consists of three main phases: VM creation and contextualization, application execution, and VM destruction. This life cycle thus begins when a `createActivity` request is received. `createActivity` requests contain a JSON document that specifies the characteristics of the job to run. An example of this JSON document can be seen in Figure 5. Among other information, this JSON document provides the computational requirements of the job (i.e., CPU, memory, and storage), the name of the virtual image to deploy in the Cloud infrastructure, the job type, the application that needs to be executed and its arguments, and the mount points of the shared storage in the VM.

After receiving a `createActivity` request, the PMES asks the OCCI Server for the creation of a new VM with the characteristics specified in the JSON document. The OCCI Server then contacts the Cloud Provider to deploy and contextualize the new VM. Contextualization is carried out through cloud-init, and consists of setting up the VM network, creating a user with the adequate permissions, generating SSH keys, and mounting the Network Storage partitions that makes available the user's workspace files and the public data (more details in 4.3.1). The OCCI Server then informs PMES of the IP address of the newly created VM, so that PMES can access the VM through SSH and execute the tool application ("app" object in Figure 5). In the case of COMPSs jobs, the PMES also dynamically generates the required COMPSs configuration files, and transfers them to the VM. In addition, the COMPSs runtime may contact

the OCCI Server at execution time to create new worker VMs depending on the job settings specified in the JSON document and on the application computational load.

The PMES monitors the whole job life cycle, and periodically updates the job status and report with new information. This information can be consulted at any time by means of the `getActivityStatus` and `getActivityReport` requests. Once the application finishes, or after receiving a `terminateActivity` request, the PMES orders the OCCI Server to destroy the previously created VM, and the job life cycle ends. In the case of COMPSs jobs, the destruction of worker VMs is managed by the COMPSs runtime also through the OCCI Server. The OCCI Server thus abstracts the PMES and COMPSs runtime from the underlying infrastructure, and allows the execution of applications using any OCCI compliant Cloud middleware, such as OpenNebula [2] and OpenStack [9].

```
[
    {
        "jobName": "processGenome_run0",
        "wallTime": "1440",
        "memory": 12,
        "cores": 4,
        "minimumVMs": 1,
        "maximumVMs": "1",
        "limitVMs": "1",
        "initialVMs": 1,
        "numNodes": "1",
        "disk": "1.0",
        "type": "COMPSs"
        "mountPoints": [
            {
            "target": "/MUG_USERDATA/",
            "device": "/MuG userdata/MuGUSER59e5ead574743",
            "permissions": "rw"
            },
            {
            "target": "/MUG_PUBLIC/",
            "device": "/MuG/MuG_public",
            "permissions": "r"
            }
        ],
        "user": {
            "username": "vre21af1",
            "credentials": {
                "pem": "pmes.pem",
                "key": "pmes.key",
                "uid": "33",
                "gid": "33",
                "token": ""
            }
        },
        "img": {
            "imageName": "uuid_mg-process_62",
            "imageType": "small"
        },
        "app": {
            "name": "process genome",
            "target": "/usr/local/code/mg-process-fastq",
            "source": "process_genome.py",
            "args": {
              "config": "/MUG USERDATA/processGenome run0/.config.json",
              "in metadata": "/MUG USERDATA/processGenome run0/.input metadata.json",
              "out_metadata":"/MUG_USERDATA/processGenome_run0/.results.json"
            }

        },
        "compss flags": {
            "flag": " --summary --base log dir=/MUG USERDATA/processGenome run0"
        }
    }
]
```

Figure 5. Example of a JSON document included in a createActivity request.

4.3. Data repository components

4.3.1. Database managers

MuG VRE user's data is divided in two types of repositories. Metadata is held in a MongoDB [9,10] database in the MMB-IRB MongoDB server following the data model as defined in the data management plan (DMP), which it is mainly based on a collection of data types, files types among other file related attributes like the path where the file can be found in the file system (see D4.5).

The MongoDB server not only holds the metadata referring to user's files, but also the necessary data to correctly define tools and visualizers and how they interact with user's files. MongoDB also keeps track of user management, job execution, and other VRE functionalities like help, sample data collections, etc. The MongoDB server where MuG data is hosted, contains also reference data as a full copy of Protein Data Bank [11] , and Uniprot [12], and the trajectory database BiGNASim [13].

Data itself is stored in a standard filesystem in its original format. The filesystem is shared with the virtualized environments via the network file system protocol. The filesystem layout is organized per user so that the privacy of data is maintained. In fact, process managers specifically mount to the deployed VM only the data belonging to the user executing the application. On the contrary, public repository data is mounted read-only (more details on the contextualization in 4.2.3).

4.3.2. Integration of remote repositories

MuG aims to ease the access of users to relevant public data repositories, where MuG related studies are been maintained (see D4.2 for details of such repositories). In the present version if MuG, metadata from selected studies of ArrayExpress [14] have been stored in the MongoDB metadata repository. Metadata stored correspond to that can be obtained in a automatic way from ArrayExpress REST API, and allows user to browse and search for specific studies using MuG VRE interface, and download data into the personal workspace for further analysis. See Figure 6 for a screenshot of MuG interface to ArrayExpress metadata.

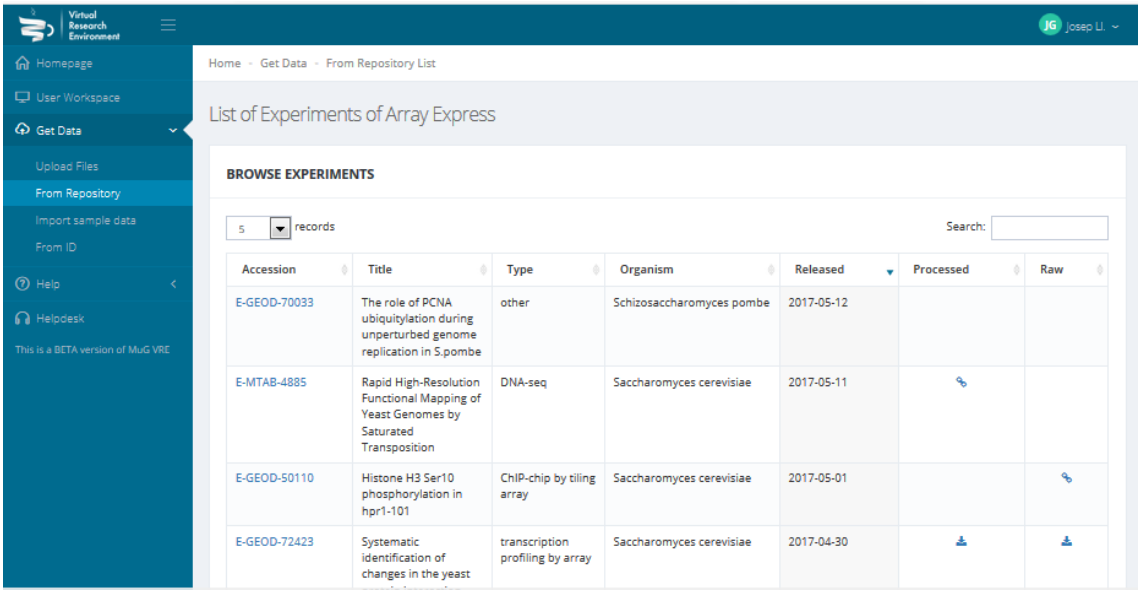

Figure 6. Screenshot of MuG interface to access ArrayExpress studies.

4.4. User access interfaces

4.4.1. Authentication

MuG VRE should assure a complete data privacy with respect to users data and activities. To this end, access to the workspace and tools, either interactively or through REST APIs is made using an encrypted channel (https, ssh), and users are authenticated on every access to the VRE.

MuG VRE uses Keycloak v3.3 identity server for the authentication. Keycloak implements OpenID Connect 1.0 which allows for the Web access a standard username/password authentication based on the code authorization flow of OAuth2, and a token based authentication for the MuG REST services such as DMP APIs based on the implicit OAuth2 flow. VRE displays the authentication tokens in use and allow to refresh them (see Figure 7.b) so that the user is able to authorize himself to the publicly available DMP services via REST.

To ease user registration, additional external identity providers like Google and LinkedIn are accepted, and Elixir Authorization and Authentication Infrastructure (AAI) is being integrated in short. Once the authentication through those providers has taken place, MuG VRE creates an internal user record with all security considerations in place independently on the identity providers used.

Figure 7. (a) VRE Login page. (b) User profile details including authorization tokens. (c) Schema of the MuG centralized authorization service based on Keycloak

4.4.2. Personal workspace

MuG VRE personal workspace is the central environment for user activity. It is based on a filesystem-based layout (see 4.4.2.2) where uploaded data and analysis results are available. The workspace gives also access to analysis and simulation tools, selected according to data types and file types (formats), recovering results as soon as they are available.

4.4.2.1. Getting data

Users can populate the workspace in several ways (see figure 8)

- Direct upload: Files from user's local computer can be uploaded directly in the workspace through a HTTPS protocol. The amount of data that can be uploaded in this way is limited due to the technical limitations of the protocol.
- Create files: A text editor is available to create simple plain text files. This is intended for data or metadata of reduced format that can be simply be typed in.
- Upload from an External URL: MuG VRE is able to access any given URL to download data into the workspace. This is the recommended procedure to include bulky data, as the procedure is performed in the background and no limit in size applies, being only limited by the user's quota available in their workspace. This option is also recommended for obtaining data from public repositories.
- From repository: A selected series of studies from public repositories are available for browsing in MuG VRE (see 4.3.2). Data from such studies can be incorporated directly into the workspace for further processing.

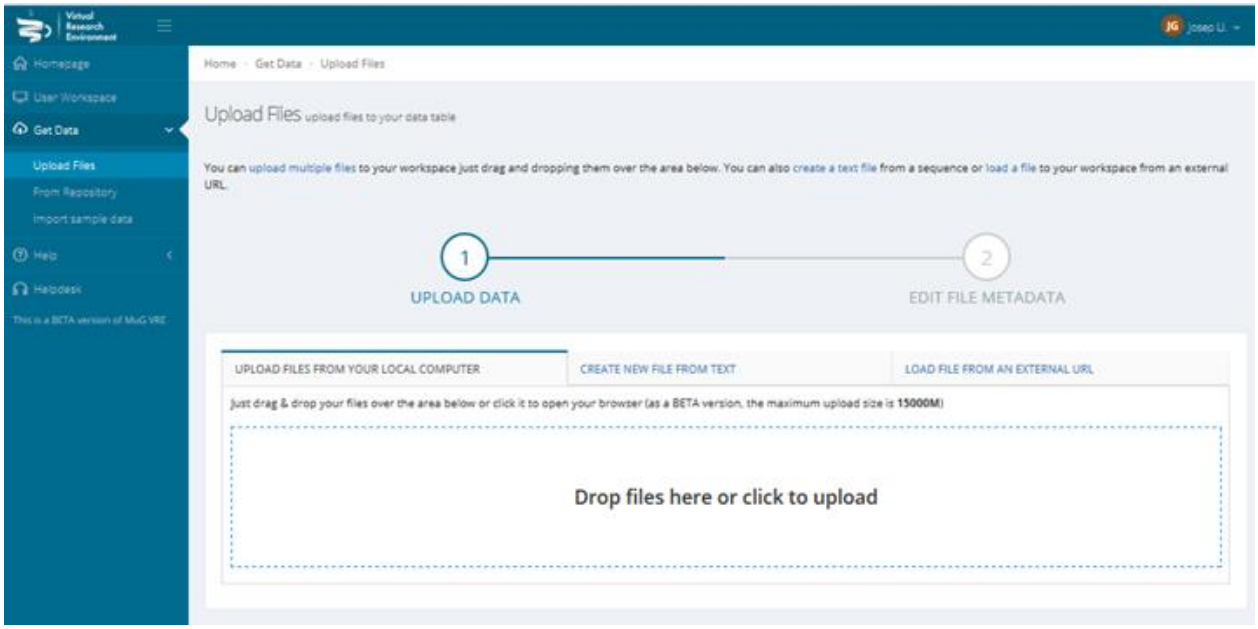

Figure 8. Options to upload data into the workspace

MuG data files should be "validated" after upload. Validation includes a number of internal check on formats, but also requires the user to fill in a series of metadata items. These include especially data type and format selecting from a predefined list (see Annex 8.1). Data types and formats enable MuG VRE to select the appropriate set of tools and visualizers usable with the uploaded files. Metadata for files obtained from MuG tools are automatically obtained from the tools metadata manifest (see 5.4).

4.4.2.2. The workspace

MuG VRE workspace (see figure 9) is organized with a file system layout with an intuitive look-and-feel. There are two types of data object: files and folders grouping files. The *Uploads* folder include all data uploaded by the user in either manner (direct, edit, or URL). Data from repositories that is grouped under *Repository* folder. The remaining folders correspond to projects, the result of executing a tool or a workflow analysis. A new folder is generated for any new process started in the VRE. Next to data, type

and format is stated. Files can be filtered by any of the fields (name, format, data type, or project). Also, a tools based filter allows to select only valid data input for a given tool.

Files are provided with three interactive toolkits (that may not appear when not appropriate). Those toolkits contain the following options:

- File toolkit: Download data or folder, edit metadata, delete, pack and compress.
- Visualization toolkit: Available visualizers for the specific data type and format.
- Tools toolkit: Selected tools for the specific data type and format.

The contents of Visualization and Tools toolkits are adapted specifically to the file, using the available metadata.

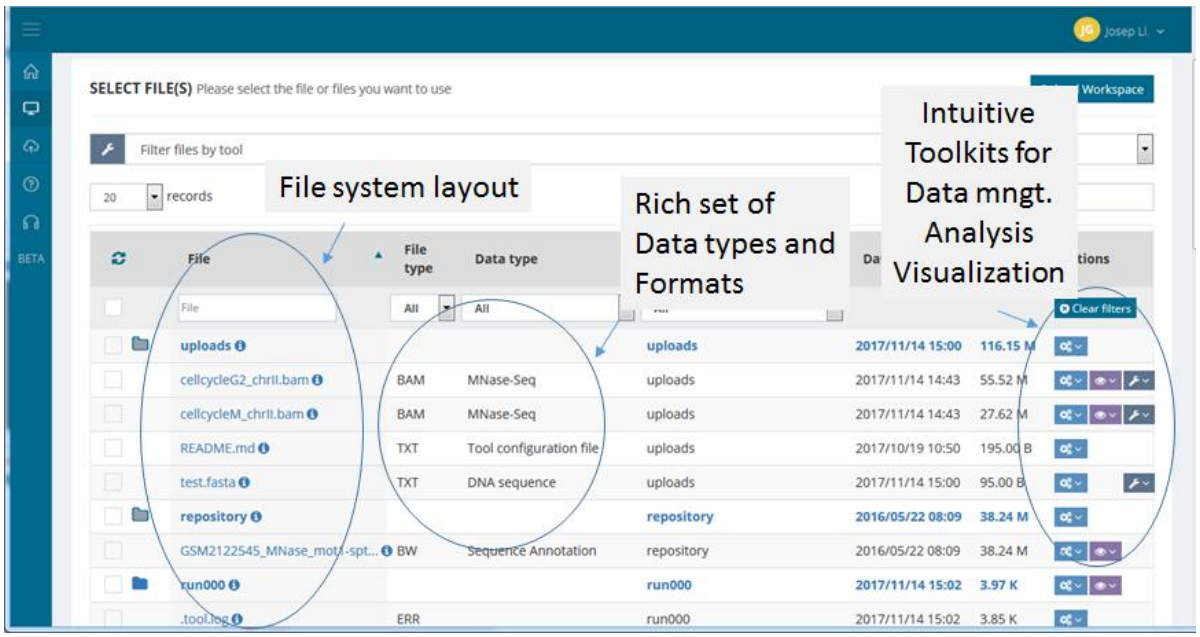

Figure 9. MuG VRE personal workspace

Tools toolkit allows to launch tools directly. For those procedures requiring more that one input file, files can be selected anywhere in the workspace, and added to the execution list. Selection of a specific tool triggers a configuration screen (see Figure 10 for an example using PyDockDNA tool) where user can assign the selected data files to the appropriate input parameters of the tool, define the necessary settings and launch the tool. Progress of the execution can be followed in the main workspace.

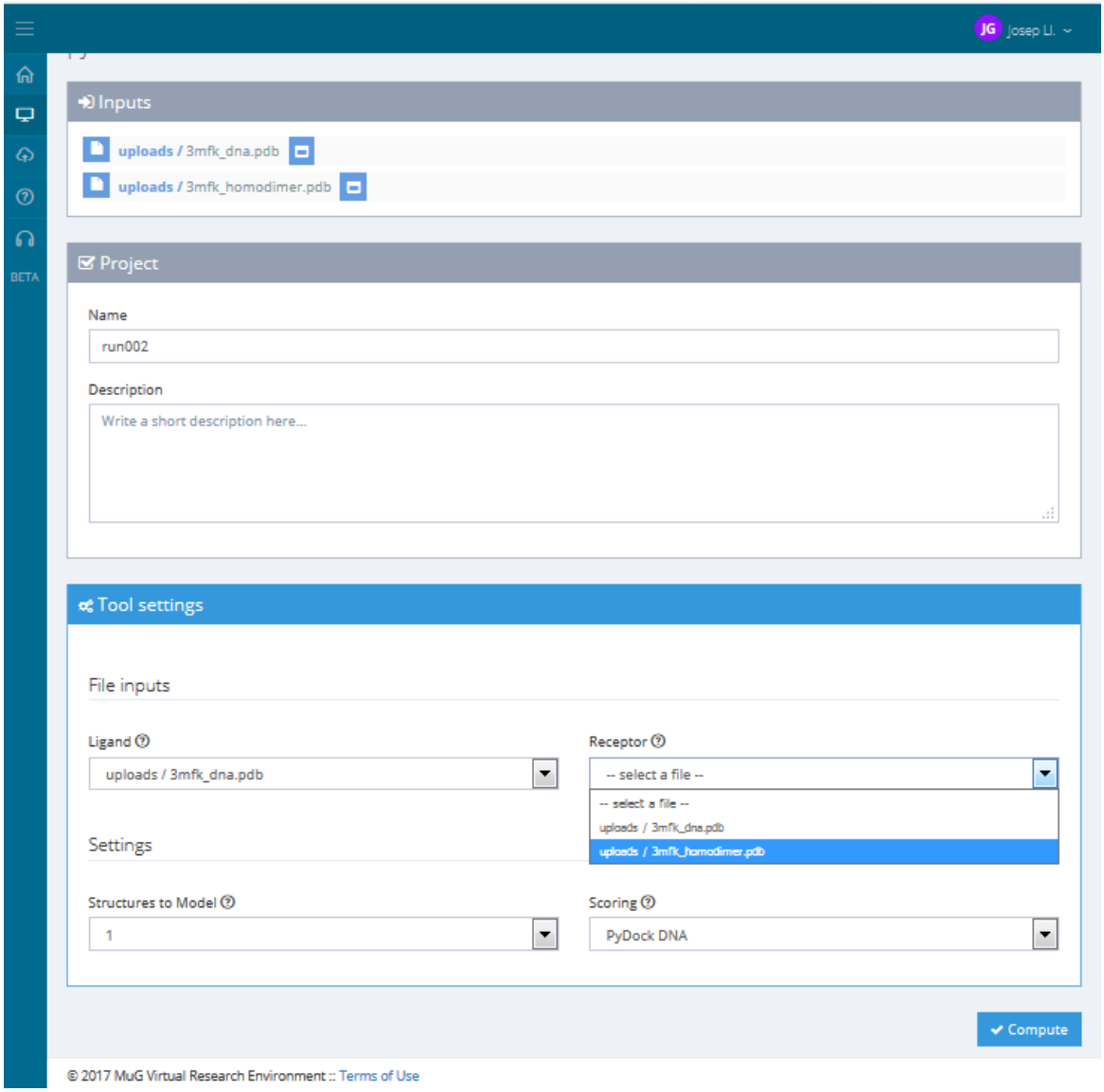

Figure 10. Configuration screen for pyDockDNA.

Finally (figure 11) the MuG VRE provides constant tracking of the state of the operations performed and the available space in the VRE.

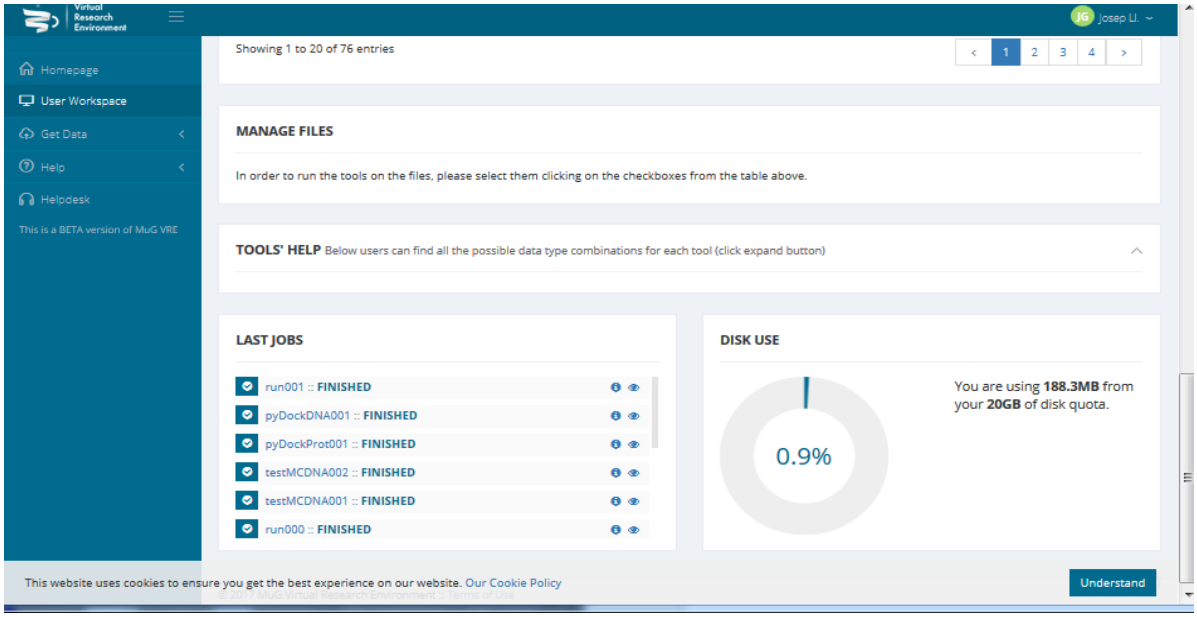

Figure 11. Personal workspace log and workspace status.

4.5. User support tools

4.5.1. Discussion forum

A forum based on Discourse [15] software package is integrated in the VRE in order to enhance scientific discussions relevant for the MuG community. Long-form chat rooms are organized by fields and tools, and users can post comments or doubts, discuss the particularities of their system, propose hot topics, etc.

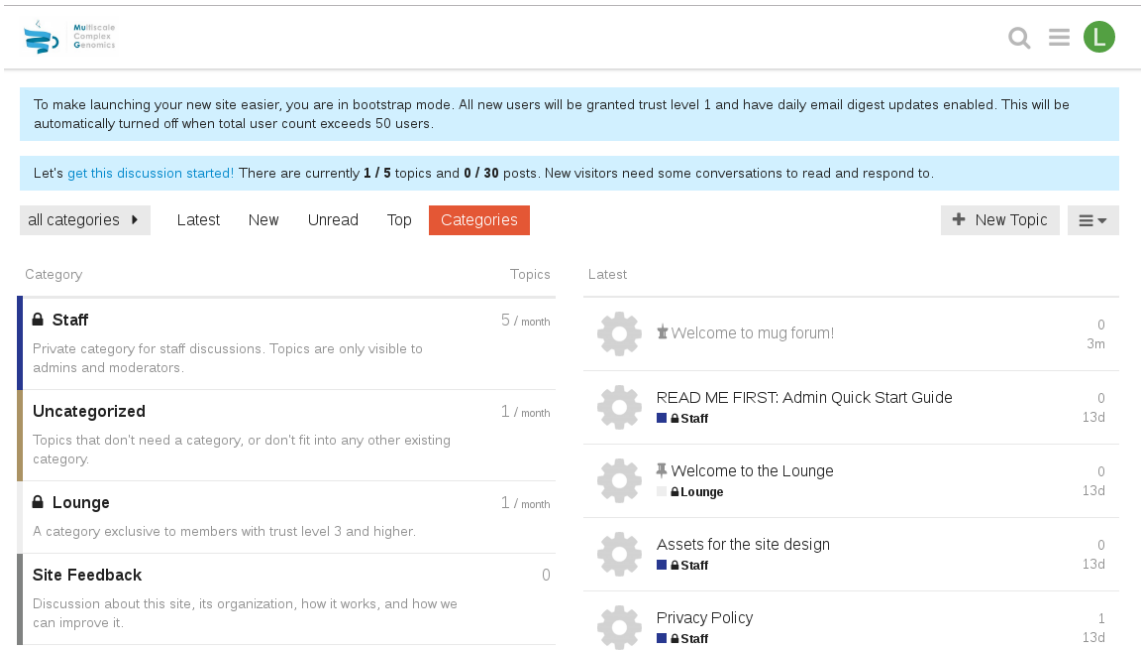

Figure 12. Discussion forum main page

4.5.2. Helpdesk

A mail based ticketing system is set up in VRE to put in contact the end user with both, tool developers, and the VRE development team. It opens a channel for resolving doubts and issues related to the VRE behaviour, the VRE implementation of any of the offered tools, and also to gather proposals and new suggestions. Mails are directly addressed to the tool author/s as well as to the site admins.

5. APPLICATIONS AND DATA OFFER

5.1. Analysis and Simulation tools

The present table summarizes the tools integrated or in the way of being integrated in the VRE together with some of their implementation details.

Table 2. Offer of tools available at MuG VRE.

Tool name	Category	Description	Implementation
Chromatin Dynamics	-Chromatin -DNA	Chromatin Dynamics provides a user friendly way to create individual 'beads-on-a-string' like representations of a chromatin fiber.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single
MC-DNA	-DNA	MC-DNA is a tool to rapidly create static and dynamic B-DNA conformations of a sequence of interest. With the use of a Monte Carlo algorithm this tool runs up to 50x faster than conventional Molecular Dynamics providing similar accuracy. MC-DNA provides a three-dimensional all-atom representation of the DNA structure with the underlying sequence of interest.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single
MDWeb (MD Energy refinement) [16]	-DNA -Protein -RNA	MDWeb is based on well known simulation programs like Amber, NAMD and Gromacs, and a series of preparation and analysis tools, joined together in a common interface.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single
NAFlex [17]	-DNA -RNA	NAFlex provides a friendly environment to analyse your own generated molecular dynamics trajectories of nucleic acid structures.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single

Nucleosome Dynamics [18]	-DNA	Nucleosome positioning plays a major role in transcriptional regulation and most DNA-related processes. The nucleosome dynamics server offers different tools to analyze nucleosome positioning from MNase-seq experimental data and perform comparative experiments to account for the transient and dynamic nature of nucleosome positioning under different cellular states.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single
3D Consensus	-DNA -Interactions -Protein	Analyse a protein-DNA complex 3D structure to identify interactions and study their impact on specific binding by integrating experimental data on the protein's DNA specificity. 3DConsensus allows the interpretation of experimental data on DNA-binding specificity of a protein through the analysis of a 3D structure of the complex.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single
Process Genome [19]	-DNA	Pipeline for generating index files for a genomic sequence. Once the index files have been generated for a given assembly then they can be used by different pipelines/tools as they are required. Based on the FASTA file of a genomic sequence index files are generated for the following indexers: Bowtie2, BWA, GEM	• Process manager PMES • tool skeleton mg-tool API • job type PyCOMPSs
pyDock [20]	-Interactions -Protein	pyDock is a tool for the structural prediction of protein-protein interactions.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single
pyDockDNA [21]	-DNA -Interactions -Protein	pyDockDNA is a tool for the structural prediction of protein-DNA interactions.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single
TADBit map filter and parse [22]	-Chromatin -DNA	TADbit is a complete Python library to deal with all steps to analyze, model and explore 3C-based data. This TADbit step maps and filters Hi-C read FASTQ files obtains a pseudo BAM with the aligned reads.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single
Tadbit Normalize [22]	-Chromatin -DNA	TADbit is a complete Python library to deal with all steps to analyze, model and explore 3C-based data. This TADbit step normalize the aligned Hi-C reads.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single
Tadbit Segment	-Chromatin -DNA	TADbit is a complete Python library to deal with all steps to analyze, model and explore 3C-based	• Process manager SGE-oneflow

[22]		data. This TADbit step finds Topologically Associating Domains (TAD)s and segments	• tool skeleton custom wrapper • job type single
Tadbit Binning [22]	-Chromatin -DNA	TADbit is a complete Python library to deal with all steps to analyze, model and explore 3C-based data. This TADbit step bins interaction matrices.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single
Tadbit Modeling [22]	-Chromatin -DNA	TADbit is a complete Python library to deal with all steps to analyze, model and explore 3C-based data. This TADbit step builds an ensemble of 3D models from the interaction matrices, able to be explored and visualized in TADkit.	• Process manager SGE-oneflow • tool skeleton custom wrapper • job type single
Process ChiP-seq [19] *IN PROGRESS*	-DNA	Pipeline for processing ChIP-seq sequence reads to identify regions of DNA-protein interactions. Sequences are aligned to the genomic sequence using BWA, BioBamBam2 is used to filter out experimental artifacts and MACS2 is used for the analysis of the alignments to identify regions of DNA-protein interaction.	• Process manager PMES • tool skeleton mg-tool API • job type PyCOMPSs
Process RNA-seq [19] *IN PROGRESS*	-RNA	Align RNA-seq data pipeline. Gene expression calling with Kallisto.	• Process manager PMES • tool skeleton mg-tool API • job type PyCOMPSs
Process WGBS [19] *IN PROGRESS*	-DNA	Align WGBS (Whole-Genome Bisulfite Sequencing) data. Uses BS Seeker2 and Bowtie2	• Process manager PMES • tool skeleton mg-tool API • job type PyCOMPSs

5.2. Protocol for integration of tools into MuG VRE

The modular and portable design of the VRE computational platform has lead to a complete virtualization of the analyzes and pipelines integrated in VRE. Tools live encapsulated in virtual machines, and the VRE core acts as a framework that delivers to them the input files and their metadata, sets up the deployment procedure of the VMs, monitors the tool execution, and eventually gathers the output files and their metadata once the execution has finished. In order to perform all these procedures, a protocol defining how the VRE core communicates with the virtualized tools has been established. From the point of view of a tool developer the protocol conforms the guidelines on how to integrate a new tool in the VRE.

The protocol covers two sides of the integration, first the registration of a tool in the VRE, and second, the execution of the same.

5.2.1. Tools registration

Metadata of tools available in the VRE is stored in the MongoDB tools collection (see section 4.3), to allow the system to adequately manage the applications and the types of data to be used or produced. To incorporate a tool into the VRE, the developer needs to prepare, along with the tool VM itself, a the *tool configuration* JSON file. Tool requirements and particulars are included in this document. Examples of the required metadata are the type of input files accepted by the tool, the arguments, the expected output files, the type of application (single, COMPSs), the MuG cloud/s infrastructure in which the tool VM is installed, the identifier of the tool VM, the application callable to be invoked inside the VM, the computational resources (cores, memory) or type of process manager (PMES, SGE-oneflow) that should be used. Check the schema and a example of the *tool configuration* JSON in Annex 8.2. With all this information the VRE is able to:

- Suggest the tool given a set of input files in the user workspace
- Create the web form so that the user fills in the arguments before executing the tool
- Invoke the application callable via any of the process managers (PMES or SGE-oneflow) following the procedure specified in the section 4.2
- Register the tool results in the DMP so output files are findable in the workspace for the user
- Recognize the ownership of the tool, so that tool developers have the adequate administrative permissions over their tools

5.2.2. Tools execution

Once the tool is properly defined, it is ready to be launched by the VRE execution engine. Figure 13 covers the complete life cycle of a tool execution in VRE, and summarizes the data flow carried out in each step.

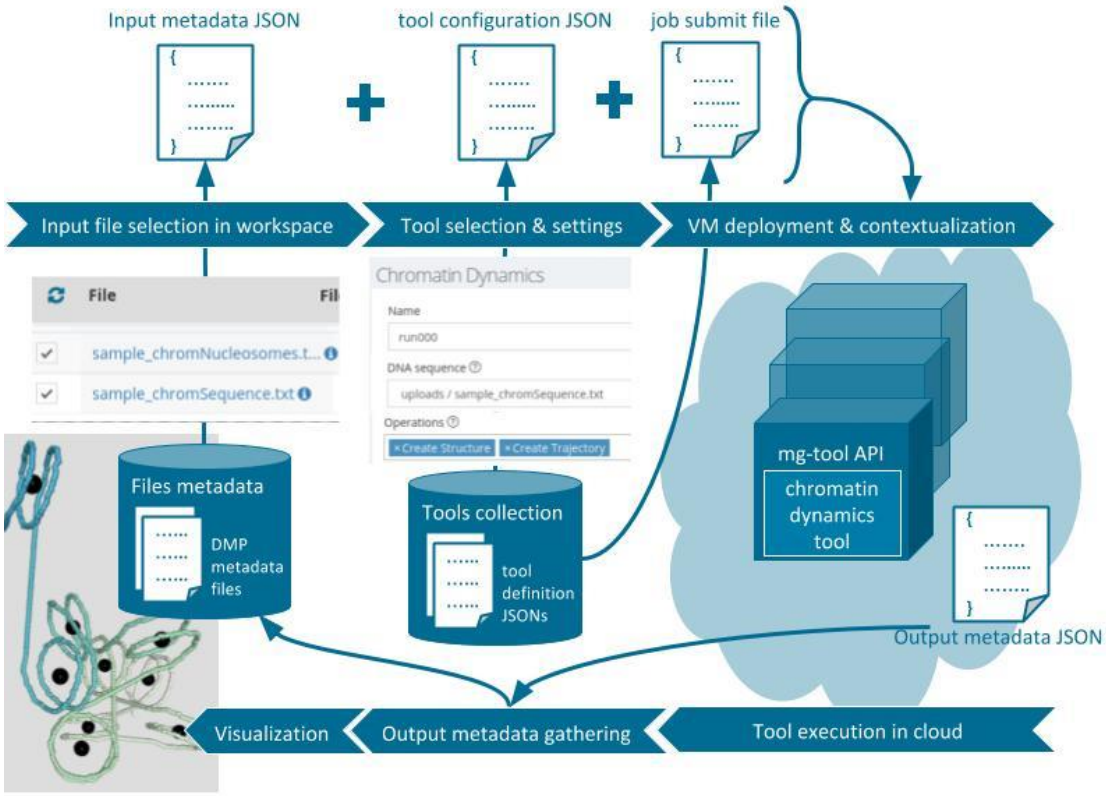

Figure 13: Life cycle of a tool execution in VRE, and how the information is transferred from the VRE user to the virtualized tools, and back to the VRE user.

The end user, via the web interface, defines the value that input files and arguments take in a particular execution. Such information is transferred to the tool via two files called *input metadata* JSON and *tool configuration* JSON (examples in Annex 8.2). The first contains the metadata corresponding to the input files, which among other attributes like data types and formats, it includes the file path as it is to be seen by the virtualized environment. The second file contains the parameter values for the application. When the user clicks the "Compute" button, and according to the *tool definition* JSON, one of the two process managers supported by VRE (PMES, SGE-oneflow) will be triggered as described in section 4.2. In short, if SGE-oneflow is the election, a *submit* BASH file invoking the application callable will be submitted to the queue, and the tool VM as part of that queue will accept and start the *submit* file execution (example in Annex 8.2). If PMES is the selected, a REST call to the *create activity* endpoint will be performed, and the tool VM will be deployed, contextualized, and finally, the application callable will be executed. Both launchers end up executing in the tool VM a specific command line whose executable is the application callable, and whose arguments are invariably:

[*application callable*]

--in_metadata [*input metadata* JSON]

--out_metadata [*output metadata* JSON]

--config [*tool configuration* JSON]

If the application is not single but of type PyCOMPSs, the process manager will invoke the application callable using PyCOMPSs libraries. In general to an instance of the the mg-tool API (https://github.com/Multiscale-Genomics/mg-tool-api), a tool skeleton developed by WP6, whose target is to ease tool integration by implementing a homogenous layer on top of the application code that both, transparently deals with VRE communication, and absorbs possible application heterogeneities. However, for some tools still to be migrated to mg-tool, a customized script that honors the previously defined command line can be also executed.

Once the execution is finished, the last step performed by the VRE engine is to gather the tool output files. At this stage results exists in the file system, but not in the metadata DMP repository unless the appropriate metadata is supplied beforehand (data types, formats). In this case, when the tool has a fixed number and name of result files, this metadata can be set as part of the *tool definition* JSON. However, if results are dynamic, the tool should create an additional file (example in Annex 8.2) called *output metadata* JSON file, containing such metadata that is imported by VRE, allowing to incorporate the results to the workspace.

5.3. Data visualization

Data visualizers allow VRE users to interactively analyse their data inside the workspace. Following with the modular philosophy of VRE, visualizers are treated in a similar way than tools are. They are again defined in a MongoDB collection where accepted data types and formats are specified. However, they are not installed as separated VMs but installed together with the VRE core.

Table 3 shows the available visualizers in MuG VRE

Visualizer	Description	Supported Data

NGL Viewer [23]	NGL Viewer is a web application for molecular visualization. WebGL is employed to display molecules like proteins and DNA/RNA with a variety of representations.	3D Structures and MD trajectories (PDB, DCD)
JBrowser [24]	JBrowse is a fast, embeddable genome browser built completely with JavaScript and HTML5, with optional run-once data formatting tools written in Perl.	Genome sequence annotation related formats (BAM, BW, GFF, GFF3)
TADKit [25]	TADkit creates interactive 3D representations of chromatin conformations modeled from 3C-based interaction matrices. The user can overlay 1D and 2D tracks of genomic data to these 3D views to directly evaluate the relationship between the 3D structure of the genome and its biological function.	HiC analysis data processed with TADBit (JSON, TXT)

5.4. MuG Data repository

Table 4 summarizes the data currently available at MuG repositories. Data is accessible through the specific interfaces. Table 5 summarized the additional annotation tracks available at JBrowse visualizer.

Table 4. Data currently available at MuG repositories

Data Set	Origin and status	Comments
Reference Databases		
Protein Data Bank (MMB-IRB)	RCSB. Weekly update	MongoDB, REST API
Uniprot (MMB-IRB)	EMBL-EBI. Monthly update	MongoDB, REST API
Reference Genomes	EMBL-EBI, Ensembl	Raw files, and specific application formats
Reference Annotation Tracks	Diverse	Displayed in JBrowse. See Table 5
MuG specific data and metadata		
ArrayExpress Nucleosome related experiments	EMBL-EBI ArrayExpress (metadata test set)	MongoDB. Web access
MuG specific simulation set	MuG partners	Available through BigNASim engine at MuG VRE web site

Nucl. Acids Flexibility Data	MuG partners	Available at MuG VRE web site

Table 5. Annotation tracks available a MuG VRE's genome browser

Data	***Source***
Saccharomyces cerevisiae	
Gene and Gene predictions	Saccharomyces Genome Database [26]
Gene structure / UTRs / transcribed regions	Yassour et al, 2009 [27]
Gene Models / introns / 5' 3' UTR's / unannotated transcripts	Nagalakshimi et al. 2008 [28]
Transcription Start sites	Zhang, Z and Dietrich FS. 200 [29]
Chromatin modifications	Kirmizis A. et al. 2007 [30]
Nucleosome positions	Mavrich et al. 2008 [31]
Digital genomic footprinting	Hesselberth et al. 2009 [32]
H2A.Z nucleosome positions	Albert et al. 2007 [33]
H2A/H2B, H2A.Z/H2A.Z, H2A.Z/H2B log2 ChIp chip ratio	Guillemette et al. 2005 [34]
H3K4ac_set1D_on_WT, set1D_H3K4ac_on_H3, WT_H3K4ac_on_H3, WT_H3K4me3_on_H3	Guillemette et al. 2011 [35]
anti-Ac, H2AK7aci, H2BK16ac, H3K14ac, H3K18ac, H3K4me1, H3K4me2, H3K4me3, H3K9ac, H4K12ac, H4K16ac, H4K5ac, H4K8ac, mock, RNA PolII ChIP_chip	Liu et al. 2005 [36]
predicted average nucleosome occupancy, predicted nucleosome potential score, nucleosome sequence read count	Field et al. 2008 [37]
nucleosome positions, nucleoatac signal, nucleosome calling occurrences	Schep et al. 2015 [38]
ORC, Mcm2p binding, ARS sequences	Xu et al. 2006 [39]
ORC, ARS, Nucleosome positioning	Eaton et al. 2010 [40]
TATA_elements	Rhee and Ough 2012 [41]
Bur1, Cet1 (Capping enzyme), Ctk1 Elf1, Kin28 (TFIIH), Paf1, Pcf11, Ser2P (RNA Pol II), Ser5P (RNA Pol II), Ser7P (RNA Pol II), Rpb3 (RNA Pol II), Spn1 (lws1), Spt16, Spt4, Spt5, Spt6, Spt6deltaC, Tfg1 (TFIIF), TFIIB	Mayer et al. 2010 [42]
Gal4, Phd1, Rap1, Reb1	Rhee and Pugh 2011 [43]
Nucleosome architecture through cell cycle	Deniz et al. 2016 [44]
Drosophila melanogaster	
Genes, Transcripts	
Chromatin types through protein binding sites	Filion et al. 2010 [45]
Nucleosome organization	Mavrich et al 2008 [46]
Homo sapiens	
Refseq Genes	
Gencode Genes	

6. DEVELOPMENT ROADMAP

6.1. User workspace

Current workspace is the central point of the VRE, as it is where input and output files are listed, and where tools and visualizers are selected. It reflects DMP metadata files, that in turn, are synchronized with the local file system.

Future changes at the workspace go in the direction of globalizing the workspace, converting it into a single virtual data space. A more integrative space where user can access data in local and remote MuG instances, and selected public data, irrespective of the geographic location. Having this single data space requires the implementation of a distributed data model, solving security issues related the administration of file permissions and group roles in the DMP, and building the necessary infrastructure for selecting the appropriate cloud instance for tool execution and making data accessible for the process (see discussion in section 6.3).

6.2. Computational layer

MuG computational infrastructure is currently composed by three cloud implementations (BSC, IRB, EBI-embassy). All MuG software components are ready to collaborate and operate remotely, hence the interconnection of these environments is being set up and tested. As a proof of concept, the development VRE instance located in the IRB is remotely deploying tools at the EBI-embassy infrastructure. The communication is based on REST calls by which VRE posts to the remote PMES server the tool job request, and the remote PMES instance deploys in the EBI-embassy cloud the targeted VM tool, and triggers the execution using EBI provided data. The standardization and development of this schema will allow to extend the number of supported e-infrastructures, opening the possibility of including other European platforms like EGI.

6.3. Data and Storage

The main MuG VRE is in in the process of fully migrating to the recently established DMP data model, based in the use of micro-services to manage data. After the migration is finished, all instances to MuG, including the main VRE workspace will act as any other DMP API client, opening the possibility to share data through a REST interface. However, apart from the management of files metadata, data itself needs to be carefully handled among MuG infrastructures. Data transfer and replication should be minimized in order to optimize procedure and resources, and data security and privacy need to be preserved along the whole process.

The current data management plan covers data transfer based on REST services, which gives to VRE the chance to smartly manage the resources across the infrastructures and ensure that data redundancy rules fit the especial requirements of our system. In MuG, the data load of the three infrastructures is not balanced, neither is their computational resources or their repository accessibility, and furthermore, the resource's end clients are diverse, they may accept streaming data (NGL viewer, simple HTTP downloads, etc) or may require to stage it in advance (tools executions itself, custom visualizers, etc).

Considering these particularities, there exists several software solutions that can complement the DMP strategy, so that VRE can delegate part of this data handling task. oneData [47], iRODS [48] (used in EUDAT), or Owncloud globalize data access in distributed environments, and their inclusion is being studied. Redundancy rules of these engines are partially configurable, and the accessibility to them as local directories is also solved by most of them. However, challenges ahead include the dynamic and user-specific contextualization of these systems.

7. MuG USAGE POLICIES

7.1. User access policy

Users sign in for free and access to the fully featured version of VRE. Registration is open, and as detailed in section 4.4.1, can use either local MuG users, or external ID providers. Once logged in, all tools and visualizers currently integrated in the VRE are widely available with no restrictions. However, each tool implementation respects the licensing rules of the original application or pipeline code, hence, a scenario where a certain tool is reserved to specific users is possible. Additionally, special conditions for highly demanding users can be negotiated.

Regarding data storage, VRE guarantees a private and secure space for user's personal data. VRE terms of use (https://dev.multiscalegenomics.eu/applib/getTermsOfUse.php) defines the security policy complied, where this data is, and who is responsible for it. *A priori*, the assigned quota is the same for all users, 20GB, although extra space can be granted individually to specific users if they request so via the help-desk section.

VRE defines three different user roles that modulate the administrative tasks a user has rights on. *Common* users have no extra privileges, while *tool developer* users and *admin* users can better monitor tool's operations running behind the interface. Furthermore, *admin* users have a complete panel for controlling the infrastructure usage (quotas, mails sent, etc) and administrating the rest of user's privileges

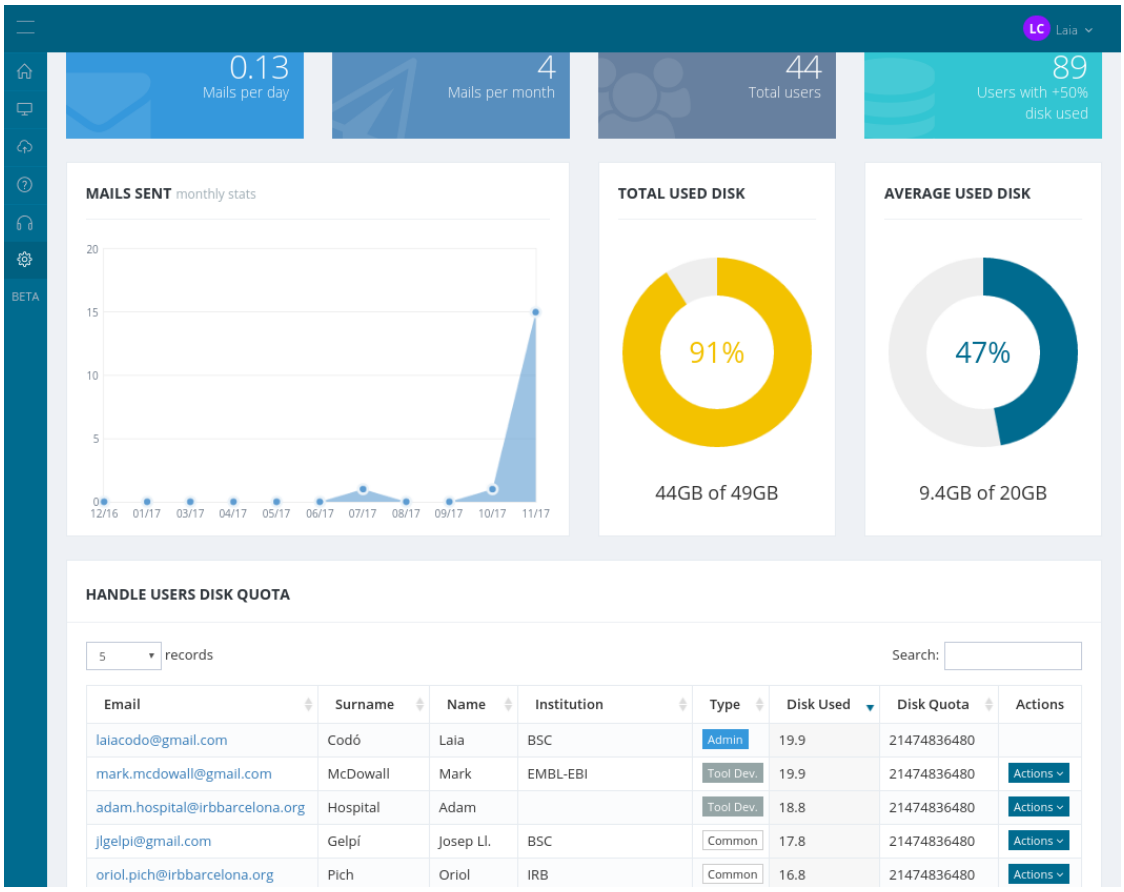

Figure 14: Main panel control for admin users

7.2. Tool developer accounts

Tool developer users have special access requirements, as they need to understand what's going on behind the VRE interface when fixing and debugging their tools within the platform. For helping on this procedure, tool developers can:

- list the tools owned by the user, and visualize their *tool definition* JSONs to check the running configuration
- consult in a simple pop up attached to each execution folder in the workspace, all the execution associated files (*input metadata* JSON, *configuration tool* JSON, *submit* file and *output metadata* JSON) that contains the data being transferred between VRE and the tool VM (Figure 15)
- check absolute paths for their files
- visualize the raw DMP metadata for all their files
- create and edit the tool help pages using a Bootstrap Markdown editor [49] able to easily style the text and uploads images.

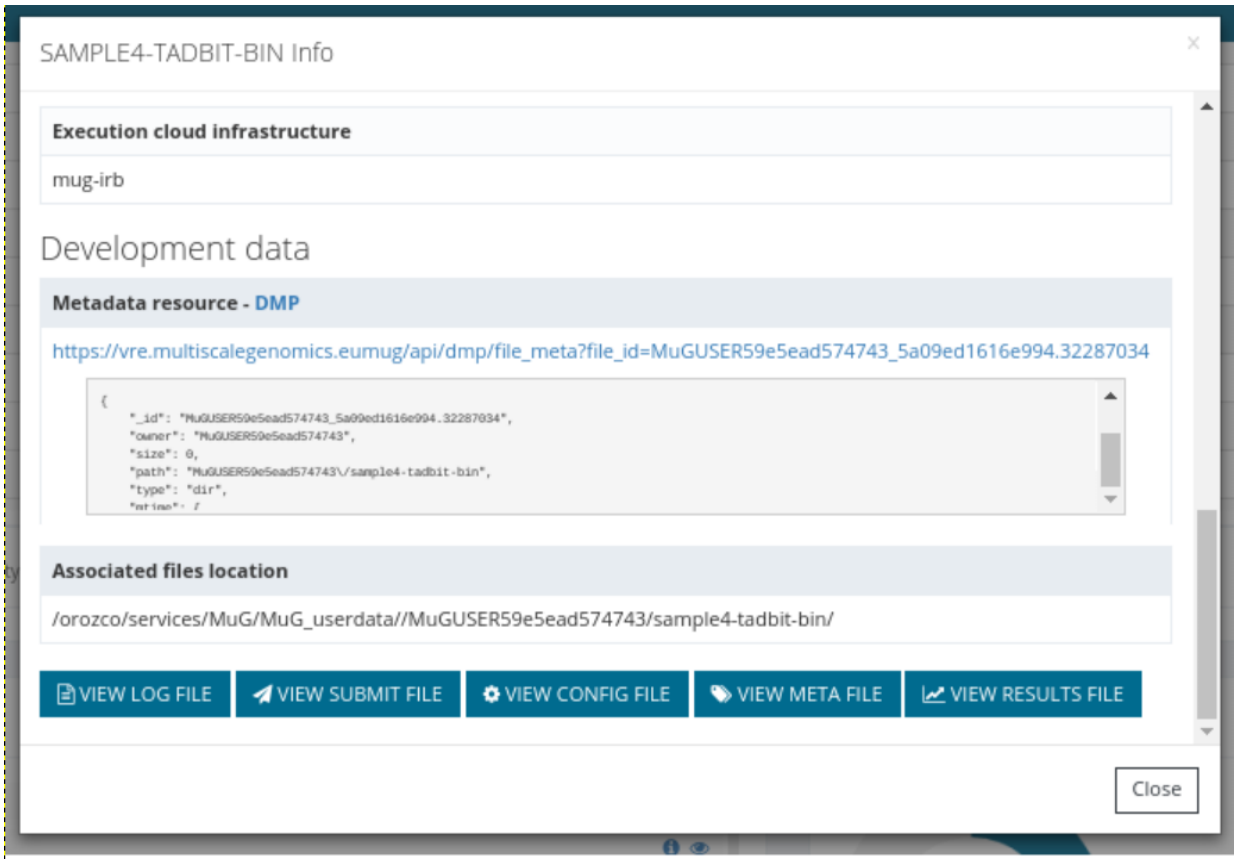

Figure 15: Extra information available for tool developers when visiting file details in the workspace

8. ANNEXES

8.1. DMP data model: data types and file types

MuG data management plan (DMP) includes a data model for files metadata that, as described in D4.5, includes the *data type* and the *file type* among other attributes to define the content and the format of user's file. Here, the complete collection of *data types* supported by VRE, and their associated *file types*.

Data Type Identifier	Data Type Name	Associated File Types
chromatin_3dmodel	Chromatin 3D structure	PDB
chromatin_3dmodel_ensemble	Ensemble of chromatin 3D structures	JSON
chromatin_compartments	Chromatin compartments data	TXT
chromatin_tads	Chromatin TADs	BED, TXT,
chromatin_traj	Chromatin trajectory	DCD
configuration_file	Tool configuration file	JSON, TXT, TSV
data_atac_seq	ATAC-Seq	FASTQ, BAM, BED, WIG
data_chip_seq	ChIP-Seq	BED, FASTQ, BAM, TSV
data_dna_methylation	DNA methylation	FASTQ, WIG, TSV
data_fish	FISH data	LIF, TIFF, PNG
data_mnase_seq	MNase-Seq	FASTQ, BAM, BED
data_rna_seq	RNA-Seq	FASTQ, TSV, HDF5, JSON
data_wgbs	Whole Genome Bisulfite Sequencing	FASTQ, BAM, BAI, WIG, TSV, TXT
docking_ranking	Docking ranking score	CSV, TXT, TSV
hic_biases	HiC Biases	PICKLE
hic_contacts_coverage	HiC contacts coverage	WIG, BW, TXT
hic_contacts_differential	HiC differential contacts	TSV
hic_contacts_matrix	HiC contact matrix	TXT, HDF5
hic_contacts_peaks	HiC contact peaks	TSV
hic_directionality	HiC directionality index	TXT
hic_reads	HiC sequencing reads	FASTQ
hic_sequences	HiC aligned reads	BAM
hic_tads_scale	HiC TADs scaling factor	WIG
md_restart	MD restart file	RST, CPT
na_md_atom_traj_coords	Nucleic acid MD trajectory coordinates	XTC, NETCDF, MDCRD

na_md_atom_traj_top	Nucleic acid MD trajectory topology	TOP, TPR, PARMTOP, PDB
na_md_cg_traj	Nucleic acid MD CG trajectory	MDCRD
na_structure	Nucleic acid 3D structure	PDB
na_traj	Nucleic acid trajectory	DCD, MDCRD
na_traj_coords	Nucleic acid trajectory coordinates	XTC, NETCDF, MDCRD
na_traj_top	Nucleic acid topology	TOP, TPR, PARMTOP, PDB
nucleosome_dynamics	Nucleosome dynamics	BW, GFF3, BED, WIG, RDATA
nucleosome_free_regions	Nucleosome free regions	BW, GFF3, BED, WIG
nucleosome_gene_phasing	Nucleosome phasing	BW, GFF3, BED, WIG
nucleosome_positioning	Nucleosome positioning	BW, GFF3, BED, WIG, TXT
nucleosome_stiffness	Nucleosome stiffness	BW, GFF3, BED, WIG
prot_dna_specificity	Protein-DNA specificity	TSV
prot_dna_structure	Protein-DNA complex structure	PDB
prot_structure	Protein 3D structure	PDB
sequence_annotation	Sequence Annotation	BED, BB, BEDGRAPH, WIG, BW, GFF, GFF3, GTF, VCF, TBI
sequence_dna	DNA sequence	FASTA, TXT
sequence_genomic	Genomic sequence	FASTA
sequence_mapping_index_bowtie	Bowtie2 index files	BT2, TXT
sequence_mapping_index_bwa	BWA index files	AMB, ANN, BWT, PAC, SA
sequence_mapping_index_gem	Sequence mapping index	GEM
sequence_mapping_index_kallisto	Kallisto index file	IDX
sequence_prot	Protein sequence	FASTA
sequence_rna	RNA sequence	FASTA
structure	3D structure	PDB
tool_intermediate_file	Tool Intermediate file	TAR
tool_statistics	Tool summary file	TAR
tss_classification_by_nucleosomes	Nucleosome TSS	BW, GFF3, BED, WIG

8.2. Documents, Software and data models

JSON schema and example of tool definition configuration file, compulsory for registering a now tool in VRE

1. tool definition JSON - schema https://github.com/Multiscale-Genomics/VRE_tool_jsons/blob/dev/tool_specification/tool_schema.json
2. tool definition JSON - example

https://github.com/Multiscale-Genomics/VRE_tool_jsons/blob/dev/tool_specification/examples/pydockdna.json

JSON examples for the configuration files sent between VRE and tool VMs during the tool life cycle execution

3. input metadata JSON - example https://github.com/Multiscale-Genomics/VRE_tool_jsons/blob/dev/tool_execution/sample_project/myPydockProject/.input_metadata.json
4. configuration tool JSON - example https://github.com/Multiscale-Genomics/VRE_tool_jsons/blob/dev/tool_execution/sample_project/myPydockProject/.config.json
5. submit file - examples https://github.com/Multiscale-Genomics/VRE_tool_jsons/blob/dev/tool_execution/sample_project/myPydockProject/.submit
6. Output metadata JSON - example https://github.com/Multiscale-Genomics/VRE_tool_jsons/blob/dev/tool_execution/sample_project/myPydockProject_out/.results.json

8.3. Usage statistics

Last year usage statistics reflect the major VRE events (April 2017 workshop, November 2017 Demo and Release). Coinciding with these events, visitors and new users increase, currently reaching 43 registered users.

Total number of registered users ● Admin users ● Tool developers ● Common users	43 2 12 29
Total used space / total quota	54/860 GB

Figure 13. VRE page views during last year filtering out partner institution domains

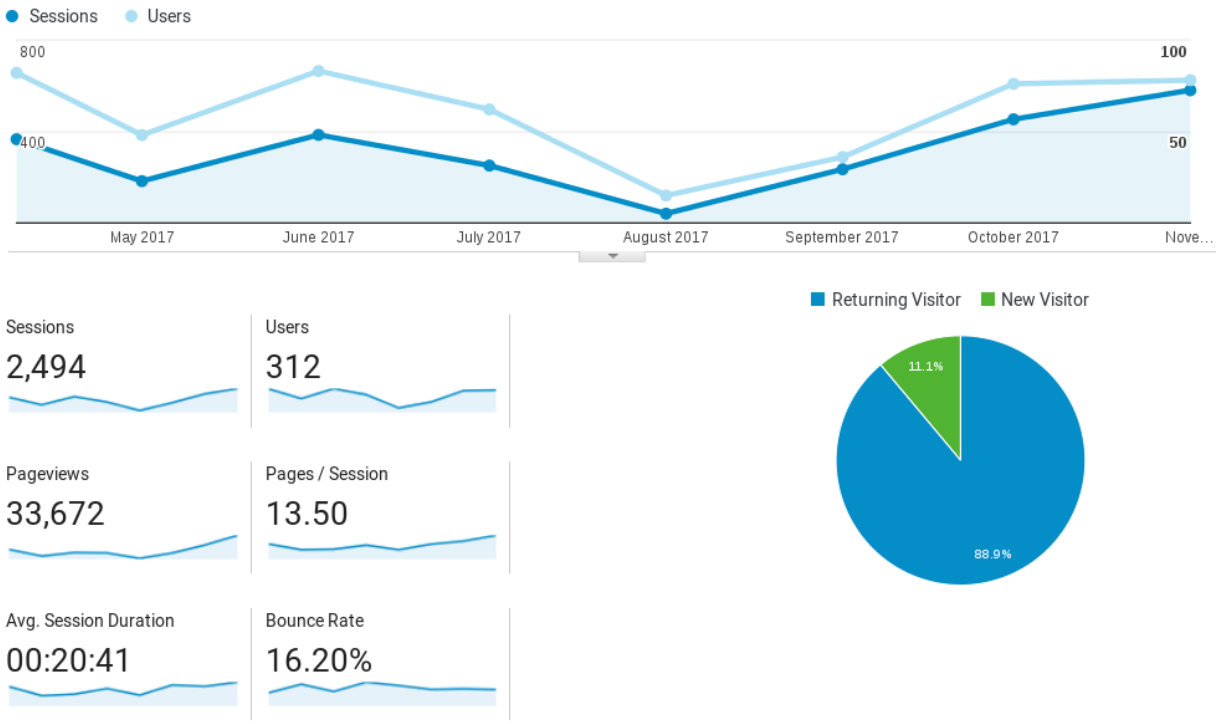

Figure 13. Last year VRE sessions (time user actively engaged in the website) and users (who have initiated at least one session)

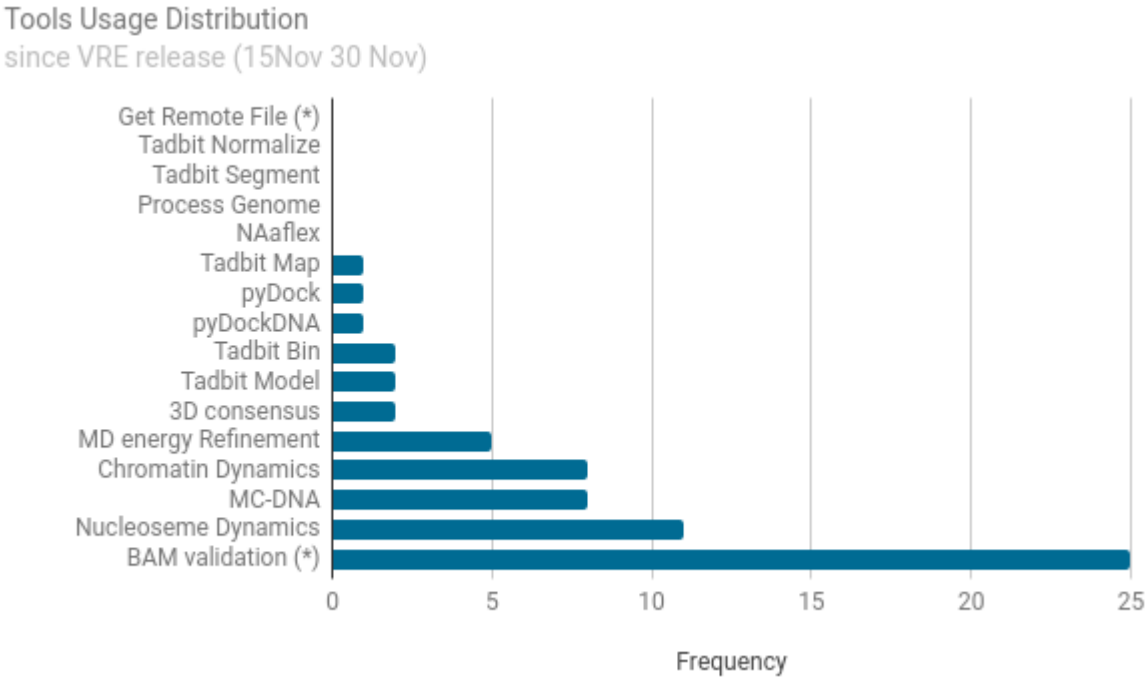

Figure 14. VRE tools usage since the release (last 15 days). (*) Correspond to those tools internally launched by VRE when I. importing a remote resource into workspace, II. importing a BAM file

9.REFERENCES

1. Keycloak Authorization server. Available: http://www.keycloak.org/
2. OpenNebula – Flexible Enterprise Cloud Made Simple. Available: https://opennebula.org/
3. KVM. Available: https://www.linux-kvm.org
4. Open Grid Scheduler. SourceForge. Available: https://sourceforge.net/projects/gridscheduler/
5. OneFlow — OpenNebula 4.12.1 documentation. Available: http://docs.opennebula.org/4.12/advanced_administration/application_flow_and_auto-scaling/oneapps_overview.html
6. Lezzi D, Rafanell R, Carrión A, Espert IB, Hernández V, Badia RM. Enabling e-Science Applications on the Cloud with COMPSs. Lecture Notes in Computer Science. 2012. pp. 25–34. doi:10.1007/978-3-642-29737-3_4
7. Lordan F, Tejedor E, Ejarque J, Rafanell R, Álvarez J, Marozzo F, et al. ServiceSs: An Interoperable Programming Framework for the Cloud. Int J Grid Util Comput. 2013;12: 67–91. doi:10.1007/s10723-013-9272-5
8. Open Cloud Computing Interface – Open Community. Available: http://occi-wg.org/
9. OpenStack Open Source Cloud Computing Software. Available: https://www.openstack.org/
10. MongoDB for GIANT Ideas. Available: https://www.mongodb.com/index
11. Berman HM. The Protein Data Bank. Nucleic Acids Res. 2000;28: 235–242. doi:10.1093/nar/28.1.235
12. The UniProt Consortium. UniProt: the universal protein knowledgebase. Nucleic Acids Res. 2016;45: D158–D169. doi:10.1093/nar/gkw1099
13. Hospital A, Andrio P, Cugnasco C, Codo L, Becerra Y, Dans PD, et al. BIGNASim: a NoSQL database structure and analysis portal for nucleic acids simulation data. Nucleic Acids Res. 2016;44: D272–8. doi:10.1093/nar/gkv1301
14. Kolesnikov N, Hastings E, Keays M, Melnichuk O, Amy Tang Y, Williams E, et al. ArrayExpress update—simplifying data submissions. Nucleic Acids Res. 2014;43: D1113–D1116. doi:10.1093/nar/gku1057
15. Discourse - Civilized Discussion. Available: https://discourse.org/
16. Hospital A, Andrio P, Fenollosa C, Cicin-Sain D, Orozco M, Gelpí JL. MDWeb and MDMoby: an integrated web-based platform for molecular dynamics simulations. Bioinformatics. 2012;28: 1278–1279. doi:10.1093/bioinformatics/bts139
17. Hospital A, Faustino I, Collepardo-Guevara R, González C, Gelpí JL, Orozco M. NAFlex: a web server for the study of nucleic acid flexibility. Nucleic Acids Res. 2013;41: W47–55. doi:10.1093/nar/gkt378
18. nucleR. In: Bioconductor. Available: http://bioconductor.org/packages/nucleR/
19. Multiscale-Genomics. Multiscale-Genomics/mg-process-fastq. In: GitHub. Available: https://github.com/Multiscale-Genomics/mg-process-fastq
20. Cheng TM-K, Blundell TL, Fernandez-Recio J. pyDock: electrostatics and desolvation for effective scoring of rigid-body protein-protein docking. Proteins. 2007;68: 503–515. doi:10.1002/prot.21419
21. Multiscale-Genomics. Multiscale-Genomics/pydockdna_tool. Available: https://github.com/Multiscale-Genomics/pydockdna_tool
22. Serra F, Baù D, Goodstadt M, Castillo D, Filion GJ, Marti-Renom MA. Automatic analysis and 3D-modelling of Hi-C data using TADbit reveals structural features of the fly chromatin colors. PLoS Comput Biol. 2017;13: e1005665. doi:10.1371/journal.pcbi.1005665
23. Rose AS, Hildebrand PW. NGL Viewer: a web application for molecular visualization. Nucleic Acids Res. 2015;43: W576–9. doi:10.1093/nar/gkv402
24. Buels R, Yao E, Diesh CM, Hayes RD, Munoz-Torres M, Helt G, et al. JBrowse: a dynamic web platform for genome visualization and analysis. Genome Biol. 2016;17: 66. doi:10.1186/s13059-

016-0924-1
25. TADbit @ CNAG/CRG. Available: http://sgt.cnag.cat/3dg/tadbit
26. Saccharomyces Genome Database. Available: http://www.yeastgenome.org
27. Yassour M1, Kaplan T, Fraser HB, Levin JZ, Pfiffner J, Adiconis X, Schroth G, Luo S, Khrebtukova I, Gnirke A, Nusbaum C, Thompson DA, Friedman N, Regev A. Ab initio construction of a eukaryotic transcriptome by massively parallel mRNA sequencing. Proc Natl Acad Sci U S A. 2009. 106(9):3264-9.
28. Nagalakshmi U1, Wang Z, Waern K, Shou C, Raha D, Gerstein M, Snyder M. The transcriptional landscape of the yeast genome defined by RNA sequencing. Science. 2008 6;320(5881):1344-1349.
29. Zhang Z, Dietrich FS. Mapping of transcription start sites in Saccharomyces cerevisiae using 5' SAGE. Nucleic Acids Res. 2005. 33(9):2838-51
30. Kirmizis A, Santos-Rosa H, Penkett CJ, Singer MA, Vermeulen M, Mann M, Bähler J, Green RD, Kouzarides T. Arginine methylation at histone H3R2 controls deposition of H3K4 trimethylation. Nature. 2007; 449(7164):928-932.
31. Mavrich TN, Ioshikhes IP, Venters BJ, Jiang C, Tomsho LP, Qi J, Schuster SC, Albert I, Pugh BF A barrier nucleosome model for statistical positioning of nucleosomes throughout the yeast genome. Genome Res. 2008 18(7):1073-1083.
32. Hesselberth JR, Chen X, Zhang Z, Sabo PJ, Sandstrom R, Reynolds AP, Thurman RE, Neph S, Kuehn MS, Noble WS, Fields S, Stamatoyannopoulos JA. Global mapping of protein-DNA interactions in vivo by digital genomic footprinting. Nat Methods. 2009. 6(4):283-289.
33. Albert I, Mavrich TN, Tomsho LP, Qi J, Zanton SJ, Schuster SC, Pugh BF. Translational and rotational settings of H2A.Z nucleosomes across the Saccharomyces cerevisiae genome. Nature. 2007;446(7135):572-576.
34. Guillemette B, Bataille AR, Gévry N, Adam M, Blanchette M, Robert F, Gaudreau L. Variant histone H2A.Z is globally localized to the promoters of inactive yeast genes and regulates nucleosome positioning. PLoS Biol. 2005; 3(12):e384.
35. Guillemette B, Drogaris P, Lin HH, Armstrong H, Hiragami-Hamada K, Imhof A, Bonneil E, Thibault P, Verreault A, Festenstein RJ. H3 lysine 4 is acetylated at active gene promoters and is regulated by H3 lysine 4 methylation. PLoS Genet. 2011;7(3):e1001354.
36. Liu CL, Kaplan T, Kim M, Buratowski S, Schreiber SL, Friedman N, Rando OJ. Single-nucleosome mapping of histone modifications in S. cerevisiae. PLoS Biol. 2005; 3(10):e328.
37. Field Y, Kaplan N, Fondufe-Mittendorf Y, Moore IK, Sharon E, Lubling Y, Widom J, Segal E. Distinct modes of regulation by chromatin encoded through nucleosome positioning signals. PLoS Comput Biol. 2008; 4(11):e1000216.
38. Schep AN, Buenrostro JD, Denny SK, Schwartz K, Sherlock G, Greenleaf WJ. Structured nucleosome fingerprints enable high-resolution mapping of chromatin architecture within regulatory regions. Genome Res. 2015; 25(11):1757-1770.
39. Xu W, Aparicio JG, Aparicio OM, Tavaré S. Genome-wide mapping of ORC and Mcm2p binding sites on tiling arrays and identification of essential ARS consensus sequences in S. cerevisiae. BMC Genomics. 2006 26;7:276.
40. Eaton ML, Galani K, Kang S, Bell SP, MacAlpine DM. Conserved nucleosome positioning defines replication origins. Genes Dev. 2010;24(8):748-753
41. Rhee HS, Pugh BF. Genome-wide structure and organization of eukaryotic pre-initiation complexes. Nature. 2012 18;483(7389):295-301.
42. Mayer A, Lidschreiber M, Siebert M, Leike K, Söding J, Cramer P. Uniform transitions of the general RNA polymerase II transcription complex. Nat Struct Mol Biol. 2010; 17(10):1272-1278.
43. Rhee HS, Pugh BF Comprehensive genome-wide protein-DNA interactions detected at single-nucleotide resolution. Cell. 2011 9;147(6):1408-1419.
44. Deniz Ö, Flores O, Aldea M, Soler-López M, Orozco M. Nucleosome architecture throughout the cell cycle. Sci Rep. 2016 28;6:19729.
45. Filion GJ, van Bemmel JG, Braunschweig U, Talhout W, Kind J, Ward LD, Brugman W, de Castro

IJ, Kerkhoven RM, Bussemaker HJ, van Steensel B. Systematic protein location mapping reveals five principal chromatin types in Drosophila cells. Cell. 2010 15;143(2):212-24.
46. Mavrich TN, Jiang C, Ioshikhes IP, Li X, Venters BJ, Zanton SJ, Tomsho LP, Qi J, Glaser RL, Schuster SC, Gilmour DS, Albert I, Pugh BF. Nucleosome organization in the Drosophila genome Nature. 2008 15;453(7193):358-62.
47. Onedata. Available: https://onedata.org
48. iRODS. Available: https://irods.org/
49. bootstrap-markdown-editor. Available: https://github.com/inacho/bootstrap-markdown-editor