



Deliverable No. 5.1.3

The final CHIC technical architecture

Grant Agreement No.:	600841
Deliverable No.:	D5.1.3
Deliverable Name:	The final CHIC technical architecture
Contractual Submission Date:	30/11/2016
Actual Submission Date:	31/03/2017

Dissemination Level



PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	CHIC
Project Full Name:	Computational Horizons In Cancer (CHIC): Developing Meta- and Hyper-Multiscale Models and Repositories for In Silico Oncology
Deliverable No.:	D5.1.3
Document name:	The final CHIC technical architecture
Nature (R, P, D, O) ¹	P
Dissemination Level (PU, PP, RE, CO) ²	RE
Version:	2.0
Actual Submission Date:	31/03/2017
Editor:	Giorgos Zacharioudakis
Institution:	FORTH
E-Mail:	gzaxar@ics.forth.gr

ABSTRACT:

This document reports on the final version of the CHIC technical architecture and it describes the most important views of the architecture based on the IEEE 1471 standard. We describe the functionality of the CHIC platform, its functional components and their programming interfaces, the type of information handled and the technical details behind the implementation of this architecture.

KEYWORD LIST:

CHIC technical architecture, views, viewpoints, requirements, functionality, functional components, interactions, information, deployment, technical resources, programming interfaces.

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement no 600841.

¹ R=Report, P=Prototype, D=Demonstrator, O=Other

² PU=Public, PP=Restricted to other programme participants (including the Commission Services), RE=Restricted to a group specified by the consortium (including the Commission Services), CO=Confidential, only for members of the consortium (including the Commission Services)

The author is solely responsible for its content, it does not represent the opinion of the European Community and the Community is not responsible for any use that might be made of data appearing therein.

MODIFICATION CONTROL			
Version	Date	Status	Author
1.0	12/06/2014	Initial version submitted to EC	Giorgos Zacharioudakis
1.1	19/02/2016	Updated version, draft	Giorgos Zacharioudakis
1.2	29/07/2016	Draft	Giorgos Zacharioudakis
1.3	07/11/2016	Draft	Giorgos Zacharioudakis
1.4	13/01/2017	Draft	Giorgos Zacharioudakis
1.5	24/03/2017	Draft, internal review	Giorgos Zacharioudakis
2.0	31/03/2017	Final version submitted to EC	Giorgos Zacharioudakis

List of contributors

- Georgios Stamatakos, ICCS
- Fay Misichroni, ICCS
- Nikolaos Touser, ICCS
- Manolis Tsiknakis, TEI-C & FORTH
- Kostas Marias, FORTH
- Giorgos Zacharioudakis, FORTH
- Stelios Sfakianakis, FORTH
- Ioannis Karatzanis, FORTH
- Elias Neri, Custodix
- Marco Viceconti, USFD
- Daniele Tartarini, USFD
- Debora Testi, CINECA
- Simone Bna, CINECA
- Philippe Buechler, UBERN
- Roman Niklaus, UBERN
- Pierre Grenon, UCL

- Norbert Graf, USAAR
- Feng Dong, BED
- Nigel Mcfarlane, BED

Contents

1	EXECUTIVE SUMMARY.....	8
2	INTRODUCTION.....	9
3	ARCHITECTURAL APPROACH.....	10
3.1	ARCHITECTURAL DESCRIPTION - THE IEEE 1471 STANDARD	10
3.2	VIEWS OF THE ARCHITECTURE.....	11
3.3	VIEWS OF THE CHIC ARCHITECTURE	13
4	ARCHITECTURAL DRIVERS IN CHIC	15
4.1	STAKEHOLDERS.....	15
4.2	GOALS, CONSTRAINTS, PRINCIPLES, CONCERNS, REQUIREMENTS OF THE CHIC STAKEHOLDERS	16
4.3	EVOLUTION AND REFINEMENT OF THE ARCHITECTURE	18
5	FUNCTIONAL VIEW	20
5.1	INTRODUCTION - HIGH LEVEL VIEW.....	20
5.2	PRESENTATION LAYER	22
5.3	SERVICE LAYER	30
5.4	DATA LAYER	64
5.5	INFRASTRUCTURE LAYER	147
6	DEPLOYMENT VIEW.....	155
6.1	INTRODUCTION – HIGH LEVEL VIEW	155
6.2	PRESENTATION LAYER	155
6.3	SERVICE LAYER	158
6.4	DATA LAYER	167
6.5	INFRASTRUCTURE LAYER	171

1 Executive Summary

This document reports on the design of the CHIC technical architecture. We describe the CHIC technical architecture following the IEEE 1471 - ISO/IEC 42010:2007 approach. In order to describe the architecture we select a set of views and viewpoints such as the Functional, the Information, the Security and the Deployment view that provide the necessary information to understand what the system does, its main functional components, their programming interfaces, their user interfaces, what information is stored or exchanged in the system and the technical details of the reference implementation and the execution environment.

2 Introduction

The CHIC project developed a software platform consisting of tools, services and infrastructure that support the creation of multiscale cancer hypermodels (integrative models). CHIC aspires to make a breakthrough in multiscale cancer modeling through greatly facilitating multi-modeller cancer hypermodelling and its clinical adaptation and validation. Standardization of model description and model “fusion” are two of the core means to achieve this goal. The creation of such elaborate hypermodels is expected to accelerate the clinical translation of multiscale cancer models and oncosimulators following their prospective clinical validation (*in silico* oncology).

Towards this goal, the CHIC project developed a suite of tools, services and infrastructure that supports accessibility and reusability of VPH mathematical and computational hypermodels. These include a hypermodelling infrastructure consisting primarily of a hypermodelling editor and a hypermodelling execution environment, an infrastructure for semantic metadata management, a hypermodel repository, a clinical data repository, a metadata repository and an *in silico* trial repository for the storage of executed simulation scenarios. All these tools and services are connected through appropriate security services. Multiscale models and data are semantically annotated using the ontological and annotating tools. An image processing and visualization toolkit have been developed, as well as cloud and virtualization services have been deployed to support this platform. The CHIC tools, services, infrastructure and repositories may provide the community with a collaborative interface for exchanging knowledge and sharing work in an effective and standardized way and a number of open source features and tools will enhance usability and accessibility.

The tools and facilities described above comprise a complex software system, which calls for a systematic approach to design, implement and document its architecture. In this document, we describe the software architecture using the approach of the ISO/IEC 42010:2007 standard.

The rest of this document is organized as following: In chapter 3 we briefly present our architectural approach. In chapter 4 we present the architectural drivers behind the CHIC platform, its stakeholders and their concerns. In chapter 5 we elaborate on the Functional view of the CHIC architecture and its functional components. In chapter 6 we present the Deployment view of the architecture, the technical details and the execution environment of the platform.

According to this conceptual model, a system has **an architecture** and this can be described in an **architectural description (AD)**. Note the distinction between the architecture of a system, which is conceptual, from the description of this architecture, which is concrete. The architectural description is defined as “a collection of products to document an architecture”. The AD can be divided into one or several **views**. Each view covers one or more **stakeholder concerns**.

- A view is defined as “a representation of a whole system from the perspective of a related set of concerns”. A view is created according to rules and conventions defined in a **viewpoint**.
- A **viewpoint** is defined as “a specification of the conventions for constructing and using a view. A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis”.

An AD selects one or more viewpoints for use and this choice depends on the concerns of the stakeholders that need to be addressed by the architectural description. A view may consist of one or more models and a model may participate in one or more views. Each such model is defined according to the methods established in the corresponding viewpoint definition. The AD aggregates the models, organized into views.

The IEEE 1471/ISO/IEC 42010:2007 standard defines a set of requirements for conforming architectural descriptions that can be summarized as:

- Identification, version, and overview information of an architectural documentation (AD)
- Identification of the system stakeholders and their concerns
- Specification of each viewpoint that has been selected and the rationale for those selections
- One or more architectural views
- A record of all known inconsistencies among the AD’s required constituents
- A rationale for selection of the architecture

It is evident from the above that this standard is largely based on the definition of the most important viewpoints and the corresponding views but it does not provide any concrete definition of those. The selection of specific views and viewpoints is largely dependent on the specific architect - the stakeholders, requirements, constraints etc.

3.2 Views of the architecture

For the selection of specific views and viewpoints of the architecture, there are a number of different architectural frameworks supporting and proposing different views and viewpoints, such as the 4+1

views model⁴, the Reference Model of Open Distributed Processing (RM-ODP)⁵, the Zachman framework⁶, the Department of Defense Architecture Framework (DoDAF)⁷, etc.

In the 4+1 model, the views that are used to describe an architecture are:

- The **Logical** view, which describes the functionality of the system.
- The **Development** view, which describes the implementation of the system.
- The **Process** view, which describes the runtime behaviour, such as concurrency and performance issues.
- The **Physical** view, which describes the deployment and the physical topology of the system.
- The **Scenarios** or **Use Case** view, which is a central view (the “+1”) that describes the system from the end user viewpoint and is used both for illustrating as well as validating the general architecture.

In the work of Rozanski and Woods⁸ they prescribed a similar set of viewpoints to be used in documenting software architectures. They have essentially extended the 4+1 model by providing the **Information** viewpoint to deal with data related concerns, like structure, ownership, distribution, etc. and the **Operational** viewpoint in order to describe how the system is installed, monitored etc. Their six viewpoints are the following:

- The **functional** view documents the system’s functional elements, their responsibilities, interfaces, and primary interactions. A functional view is the cornerstone of most architecture documents and is often the first part of the documentation that stakeholders try to read. It drives the shape of other system structures such as the information structure, concurrency structure, deployment structure, and so on. It also has a significant impact on the system’s quality properties, such as its ability to change, its ability to be secured, and its runtime performance.
- The **information** view documents the way that the architecture stores, manipulates, manages, and distributes information. The ultimate purpose of virtually any computer system is to manipulate information in some form, and this viewpoint develops a complete but broad view of static data structure and information flow. The objective of this analysis is to answer the important questions around content, structure, ownership, latency, references, and data migration.

⁴ Kruchten, Philippe. “Architectural Blueprints - The 4+1 View Model of Software Architecture.” IEEE Software, 12(6):42–50, November 1995.

⁵ Reference Model of Open Distributed Processing. ITU-T Rec. X.901-X.904 / ISO/IEC 10746, <http://www.rm-odp.net/>

⁶ Zachman framework. <http://www.zachman.com/about-the-zachman-framework>

⁷ Department of Defense Architecture Framework (DoDAF). <http://dodcio.defense.gov/dodaf20.aspx>

⁸ Rozanski, Nick, and Woods, Eoin. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2nd Edition. Addison Wesley, 2011.

- The **concurrency** view describes the concurrency structure of the system and maps functional elements to concurrency units to clearly identify the parts of the system that can execute concurrently and how this is coordinated and controlled. This entails the creation of models that show the process and thread structures that the system will use and the inter-process communication mechanisms used to coordinate their operation.
- The **development** view describes the architecture that supports the software development process. Development views communicate the aspects of the architecture of interest to those stakeholders involved in building, testing, maintaining, and enhancing the system.
- The **deployment** view describes the environment into which the system will be deployed, including capturing the dependencies the system has on its runtime environment. This view captures the hardware environment that the system needs, the technical environment requirements for each element, and the mapping of the software elements to the runtime environment that will execute them.
- The **operational** view describes how the system will be operated, administered, and supported when it is running in its production environment. For all but the simplest systems, installing, managing, and operating the system is a significant task that must be considered and planned at design time. The aim of the operational view is to identify system-wide strategies for addressing the operational concerns of the system's stakeholders and to identify solutions that address these.

3.3 Views of the CHIC architecture

Based on the architectural drivers of the CHIC project, the stakeholders, the requirements, the constraints and the general principles that are given in more detail in chapter 3, we used the following subset of the views of the Rozanski and Woods model for the architecture description:

- The **Functional view**, to describe the functionality and the functional elements of the architecture
- The **Information view**, to describe the type of information, its flow and its management in the system
- The **Deployment view**, to describe the technical implementation details, the execution environment and the deployment configuration

In the initial version of the technical architecture, due to the high importance of the security aspects in the CHIC project in order to address the legal and security requirements, we had included a dedicated **Security view** to describe the security issues of the architecture. However, in this final version of the architecture we do not elaborate on the security aspects since they are covered in exhaustive detail in CHIC deliverable **D5.2.2 “Final version of security tools and guidelines”**. Similarly, in the initial version of the architecture we elaborated on the **Information view** but in this final version it is not described in deep since all details regarding the design and implementation of the CHIC Repositories can be found in the deliverable **D8.4 “Report on the final system”**.

A view that is often used for the description of the architecture, as proposed also by Rozanski and Woods, is the **Operational** view. This information however, depends on the deployment model of the architecture and the specific operational context and for the current deployment model most of the necessary information is given in the deployment view. For other deployment models, we discuss the effects and the operational consequences in **D5.1.2 “Deployment models of the CHIC technical architecture and its private cloud”**.

4 Architectural drivers in CHIC

As a general principle, the development of a system is successful when it delivers the functionality requested by its users. So the most important factors to take into account when designing the architecture of a system are the stakeholders involved, their requirements and any existing constraints, such as technical or legal issues that must be met. Below we present the drivers behind the design of the CHIC architecture, as defined in the Technical Annex of the project, the deliverables of the project and the feedback we have collected from the consortium partners.

4.1 Stakeholders

The general groups of stakeholders involved or concerned about the CHIC project are the CHIC consortium partners, the patients, the users (current or future) of the developed tools and the European Commission.

The consortium partners as well as the external users of the CHIC platform and tools can be further categorized into distinct stakeholder groups; the clinical partners, such as clinicians, clinical researchers and data providers, and the technical partners such as the modellers, researchers in various domains, especially in the clinical, biological or bioinformatics domain, the software developers and the administrators of the final platform.

While each stakeholder might have different concerns or requirements over the CHIC platform, some of them might share in parallel more than one role. For example a clinical partner might both be a clinical data provider and a clinical researcher; a technical partner may both be a model provider and a software developer. For this reason, for the sake of clarity and better understanding of each stakeholder category, the requirements and the functionality they need, the categorization given below is based on the distinct *virtual* roles participating in the project than the actual end-users or stakeholders. Due to simplicity, the architecture does not enforce any separation of the roles in the functionality of the system. The separation of functionality, or access control, is achieved in the implementation of the architecture by introducing relevant mechanisms and policies (described in briefly in paragraph 5.3.3 and in more detail in deliverable D5.2.2 “Final version of security tools and guidelines”).

Below we list a brief description of the stakeholders; a more elaborate description is given later in the following section.

Table 1 Stakeholders of CHIC

Stakeholder	Remarks
Data providers	The data providers are the end users who will be providing data into the CHIC platform. The main tasks of those users will be to de-identify, upload and annotate data for usage by other users or components of the platform.

Model providers	The model providers upload, annotate and execute already available models. They also develop and execute hypermodels (integrative models) using the CHIC platform.
Researchers	This is a general user group which contains clinicians, bioinformaticians, mathematicians, physicists etc. whose main task is to design, compose or execute hypermodels in CHIC. They edit, manage, execute, publish hypermodels or use the visualization and imaging tools to conduct their research.
Clinicians	The clinicians search and execute hypermodels or imaging tools that are already available in the platform and they operate them on appropriate data sets in order to find answers into specific clinical questions.
Legal partners	The legal partners of the consortium are neither clinical nor technical partners; they can be seen as a representative both of the patients, the EC and the State, making sure that all legal aspects are met.
Software developers	They develop and test the IT infrastructure (development phase)
Administrators	They operate the IT infrastructure (production/operation phase)
Patients	They consent to give their personal data for research, or they are interested in having access to results of research which might relate to their own disease.
External users	This user category contains all users outside of the CHIC consortium. External users may want to access and use the CHIC IT platform or results of research conducted with it. They are not officially related to the CHIC project.
European Commission	The EU is funding the CHIC project and has a contract with the CHIC consortium members for the delivery of the final CHIC outcome.

4.2 Goals, constraints, principles, concerns, requirements of the CHIC stakeholders

4.2.1 Data providers

Data providers are usually clinicians or clinical researchers who share data in order to be able to use it for their own research, and to the benefit of their patients. The goal of the data providers, inside the CHIC platform, is to be able to easily upload and share the data they own. Their main concern is to be able to protect the anonymity and the legal rights of the persons whose data are used in CHIC and also to retain, as much as possible, the right to use/update/withdraw these data –in other words, to have clear terms of ownership and control.

4.2.2 Model providers

Model providers come from different domains; they often are IT researchers, physicists, mathematicians, bioinformaticians, biologists, clinical researchers. They develop models using a variety of technologies (outside the context of CHIC) and they want to upload, share, and execute them in the context of CHIC in order to validate them, or to expand them by finding other models and composing hypermodels (integrative models). A main concern that they have, similar to the data providers, is to retain control over what they share or its results –in other words, the intellectual property rights (IPR) management. An important goal also for CHIC is to ensure the technical compatibility between the technologies that the modellers use and the ones supported by CHIC.

4.2.3 Researchers

The researchers group is a general group consisting of people from different domains, similarly to the model providers. Their main goal into the CHIC platform is to find available models and data sets that relate to their scientific research, to combine those models, forming hypermodels (integrative models) and to execute them. Their main concerns relate to the scientifically sound outcome of this integrative approach and all the related engineering aspects in order to achieve this task.

4.2.4 Clinicians

The clinicians are a distinct subset of the researchers' user group, whose main goal and concern is to find answers into concrete clinical questions. They usually don't have as a goal to develop models or hypermodels, but to find and execute already developed tools in order to gain knowledge that will help their patients.

4.2.5 Legal partners

The legal partners, share concerns with the data and model providers in terms of the IPR management, the legal constraints concerning the handling of personal and sensitive data, and the adherence to the legal constraints of the overall platform, such as the auditing, authentication and authorization mechanisms. This group of stakeholders can be seen as a representative of the patients, the EC and the State, who makes sure that all procedures followed as well as the final system meet the legal and ethical constraints.

4.2.6 Software developers

The software developers, the IT partners of the consortium have as primary goal to deliver the CHIC platform with all the necessary functionality, with the time and budget constraints that are imposed

by their contractual obligations with the rest of the consortium partners and the EC. Their primary concerns are to overcome the IT engineering difficulties and deliver a state of the art infrastructure.

4.2.7 Administrators

The administrators can be considered as a subset of the IT partners whose main task is to run the CHIC infrastructure when it enters a production phase. The primary concern of those users is having an infrastructure that works robustly and the availability of tools that will make their job easier for the management of the whole platform.

4.2.8 Patients

This user group is not officially involved in the CHIC project, but their concerns are implicitly shared through the data providers, the legal partners and the EC. The main concern of this user group is respecting their personal data and having their informed consent in whatever clinical practice concerns them, and at the same time to be able to benefit from state of the art research and clinical trials that could help them into their individual clinical case.

4.2.9 External users

This user group is the general public who are not officially related with the CHIC. Researchers from the wider scientific community who are interested in gaining knowledge from the CHIC research, sharing data or models, or interested in collaborating in whatever means. The CHIC platform should be built having an open architecture, using standards and exposing information and knowledge with such a way that will promote the research collaboration and that will disseminate knowledge and experience.

4.2.10 European Commission

The European Commission is the stakeholder who, on behalf of the taxpayers, is concerned over having a functioning system delivered timely, with the financial and other contractual restrictions met, and by ensuring that the overall project meets the research goals that are foreseen and highly expected.

4.3 Evolution and refinement of the architecture

The architecture aims to address the requirements and concerns of the stakeholders, so it is natural to be refined and adapted in the development course if any requirements change and as feedback from the end users is collected and re-iterated in the design process.

Some major requirement modifications that emerged in the course of the project that required major changes in the initial architecture are the following:

- Private cloud. The initial CHIC Technical Annex had the plan to deploy a *private cloud* for the design and implementation of the project tools and components and then to migrate all services and data to a *public cloud*, in a more stable, robust and resource-rich operational environment. However, critical review comments that we received from the project reviewers strongly advised the consortium to modify this planning and to not deploy any project data, especially sensitive biomedical data, into any public cloud; irrespective from the fact that the design of the architecture would have installed all the necessary policies, mechanisms and precautions for such a utilization of 3rd party infrastructure. This modification in the design constituted a major turn in the course of the project that affected not only the infrastructure layer of the architecture but also a number of the technical components.
- Focus on the Clinical functionality of the project. The initial Technical Annex had the vision to implement a prototype of a technical platform that would provide functionality to a variety of end user categories, such as modellers, researchers, clinicians, bioinformaticians. However, taking into account the time, budget and resource constraints of the project this vision proved that it probably was too ambitious; such an endeavour would go beyond the scope of the project and wouldn't have allowed to focus and elaborate on many technical or research issues of the project. Thus, the reviewers suggested that the project should primarily focus on delivering the clinical functionality of the project. Again, this requirement modification changed the focus of the project. A number of technical tools were given higher priority in the implementation and some new functionality was introduced in order to address mainly the requests and the functionality required from the clinical end users. Most important modification was the introduction of the CRAF tool, in order to provide the clinical functionality of the project through a desktop application. This modification downgraded the importance of other tools such as the Data Upload Tool, the Hypermonitor (hypermodelling execution monitoring) and the User Portal, since CRAF integrated their functionality that was addressed to clinical users.

5 Functional view

5.1 Introduction - High level view

In this section we describe the Functional view of the system; the way that the system interacts with the end users or other systems and fulfils their requirements. The functionality of the system results from the functionality of its constituent functional components. Some functional components, mostly the ones in the user interface (presentation) layer, are meant to be used by end-users so their functionality is provided by a graphical user interface, while other components are meant to be used or interact programmatically with other software components and consequently they provide an application programming interface (API).

In order to describe the functional view of the system, we need a list of its functional elements, their responsibilities, their interactions and their interfaces, either user interfaces or programmatic interfaces.

In Figure 2 we depict a high level view of the main functional components of the CHIC platform, organized into layers and functional groups:

- On top is the **Presentation layer** which includes the components that provide a graphical user interface (GUI). These components are the point of access and interaction of the end users with the CHIC platform. Two major subgroups on this layer are the desktop-based and the web-based applications.
- Below is located the **Service layer** which includes the back-end services, the tools and components that provide business logic to the CHIC platform and mostly interact with other components via application programming interfaces (API).
- The **Data layer** sits below, and includes the repositories of the system. The functionality of these components is described in this section but an elaborate description of the information they store can be found in the corresponding deliverable D8.4 “Report on the final system” that describes the CHIC repositories.
- The **Infrastructure layer** is located on the bottom, providing support and physical resources to the other functional components.

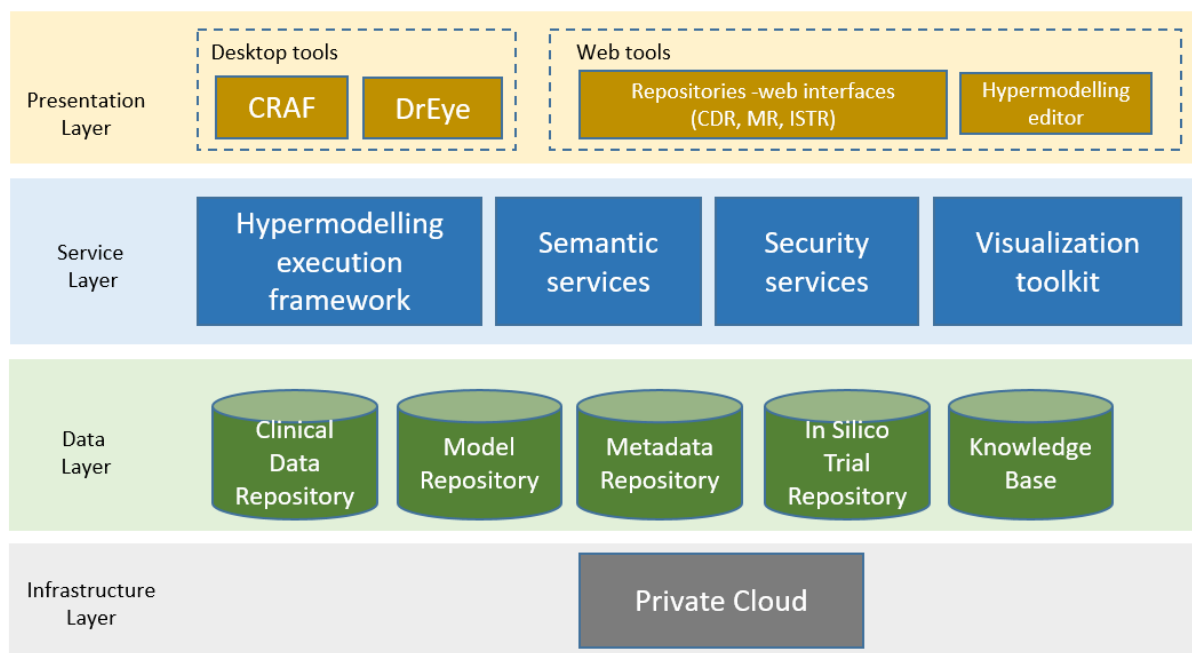


Figure 2 : Functional view: A high level view of the CHIC functional components

The layering described above is a conceptual separation which serves in order to better illustrate the functional groups and their role in the overall functionality of the platform. However, there is not any strict separation of the interaction between the various layers; the user applications also interact with the data integration layer and the repositories, and the service layer also interacts directly with the infrastructure layer. The implementation of these functional groups during the development phase of CHIC is roughly depicted in the separation of work packages in the Technical Annex of CHIC:

ID	Functional groups	WP(-s)
1	Integrated platform (CRAF, Portal)	WP10
2	Image processing and visualisation toolkits	WP9
3	Hypermodelling infrastructure	WP7
4	Data management system (data and model repositories)	WP8
5	Cloud infrastructure (storage and computation)	WP5, WP10
6	Semantics framework (annotation of data and models)	WP7, WP8
7	Security framework	WP5

In order to describe the functional view of the system we present below an elaborate list of its functional elements and the functionality they provide, their interactions with other CHIC components and their interfaces, either user interface (UI) or programmatic interface (API).

5.2 Presentation layer

5.2.1 Clinical Research Application Framework (CRAF)

5.2.1.1 Description

The aim of the Clinical Research Application Framework (CRAF) is to provide a suite of tools in the clinical domain for the clinicians to run the CHIC models that have been verified for clinical use in order to gain potentially valuable information for the best treatment or diagnosis of their patients. Its main responsibility therefore is to request the execution of the CHIC hypermodels for the given patient data and to present their results to the clinical users. In addition to that, CRAF is also responsible for the upload of the patient data to the clinical domain of CHIC and their proper categorization and indexing, and in general for the provision of a unified interface to the whole CHIC platform for the clinical purposes.

5.2.1.2 Functionality

ID	Functionality
1	Allow the enrolment of patients by the authorized clinical staff and the upload of the relevant clinical data.
2	Support the selection of the most relevant hypermodels to run based on the profile and characteristics of a given patient.
3	Support the execution of the selected hypermodels for a specific patient based on a relevant clinical question and the presentation of their results.

5.2.1.1 CHIC components interaction

The main controller of CRAF is responsible for aggregating information from a variety of CHIC components in order to provide a unified interface and a single point of access for the CHIC clinical users (see Figure 3). These components include:

- The Model Repository, which is the model registry of CHIC. CRAF retrieves the list of the available, clinically relevant, hyper models from this repository and presents them to the user to be selected.
- The VPH-HF execution framework that is responsible for the actual model execution using the user supplied patient data.
- The inSilico Trial Repository, which stores the results of the executions alongside with the input data used and any other relevant information, such as starting and ending time, the identity of the users that triggered the execution, etc.
- The Clinical Data Repository for the storage and the indexing of the patient data that the users upload through CRAF. These sets of data are subsequently used for the execution of the CHIC hypermodels, so CRAF also keeps enough information about what have been uploaded, by whom, etc.
- The Semantic services and the semantic infrastructure are the curator of rich metadata annotations using domain specific ontologies for the models, their parameters, and their

outputs. CRAF contacts the semantic infrastructure to acquire these annotations and make intelligent decisions on the values of the parameters of the hypermodels or the specific patient data that should be used for each execution.

- The security related set of services, such as the Identity Provider/Secure Token Service, the Authorization Service, and the Personal Information Management Service (PIMS) are vital for providing the security layer in the CHIC clinical domain. CRAF contacts the Authorization Service in order to make authorization decisions based on the users’ identity and roles before any access or modification to the sensitive patient data and prior to any model execution. No patient specific (identifying) information is stored in CRAF. Instead, during the registration of a new patient, it forwards patient demographics and identifiers to PIMS and gets back the patient pseudonym. All data managed by CRAF are tagged with the patient pseudonyms. Whenever, for usability purposes, it needs to present the patients’ demographics to the user, CRAF contacts PIMS again to do the reverse mapping, from the pseudonym to the real patient identity.

In addition to the above components, CRAF also integrates with the auditing and monitoring services of the cloud infrastructure so that the administrators of the platform and responsible personnel have a detailed view of its health and usage.

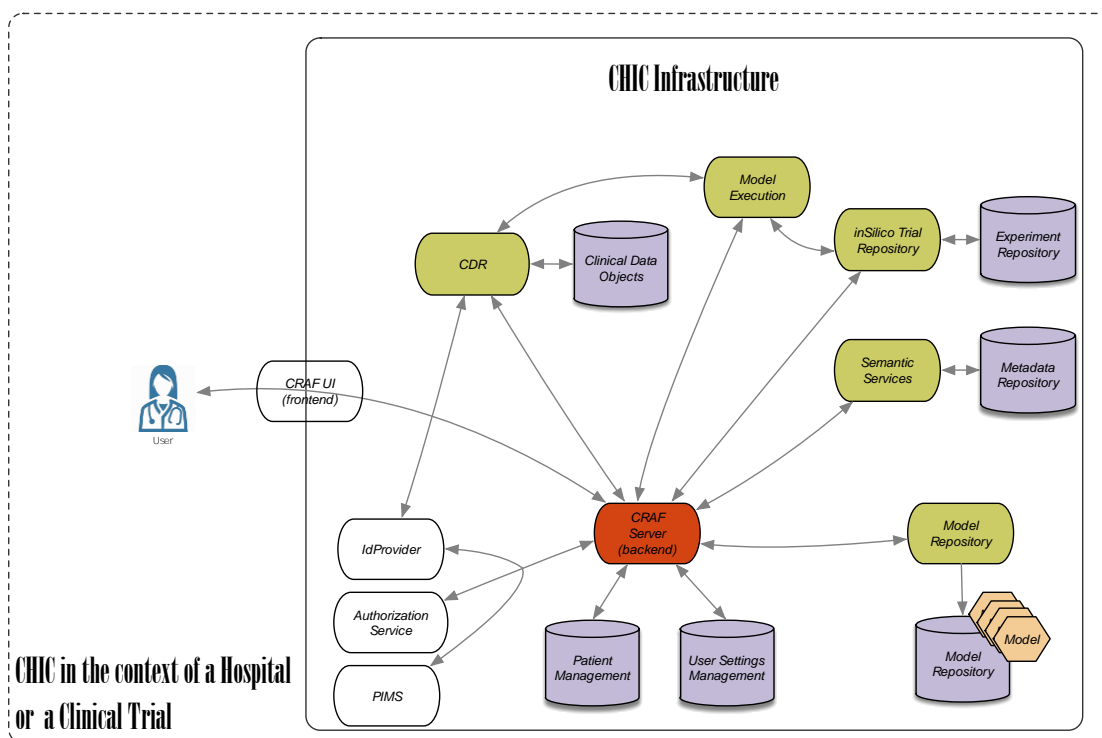


Figure 3 The architecture of the CHIC clinical domain and CRAF as its core

5.2.1.2 API interface/User Interface

The end-user facing application of CRAF is accessible through a web browser, although a desktop version of it is also available but it’s considered deprecated. CRAF does not expose any programmatic

interface (API) apart from the one used for its own purposes, i.e. the communication between the client HTML and Javascript based interface and the server side.

The following pictures show some of its functionality and the user interface it supports. More details about its features and interface can be found in Deliverables 6.2 (“Initial standardized cancer hypermodels”) and 10.5 (“The CHIC Clinical Research integrated platform”).

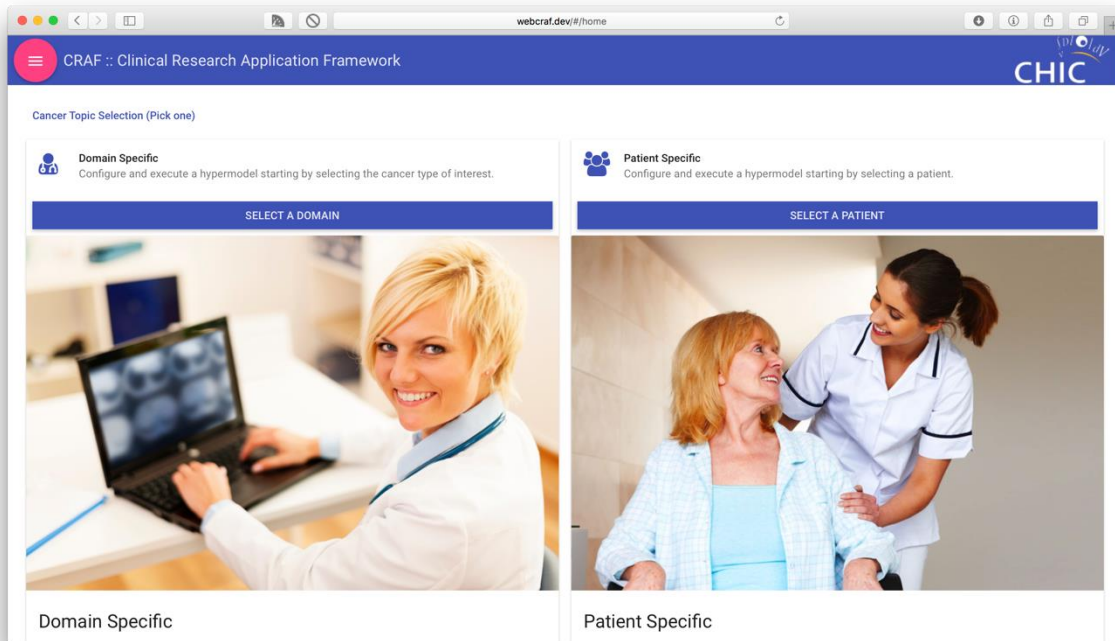


Figure 4 The main CRAF window

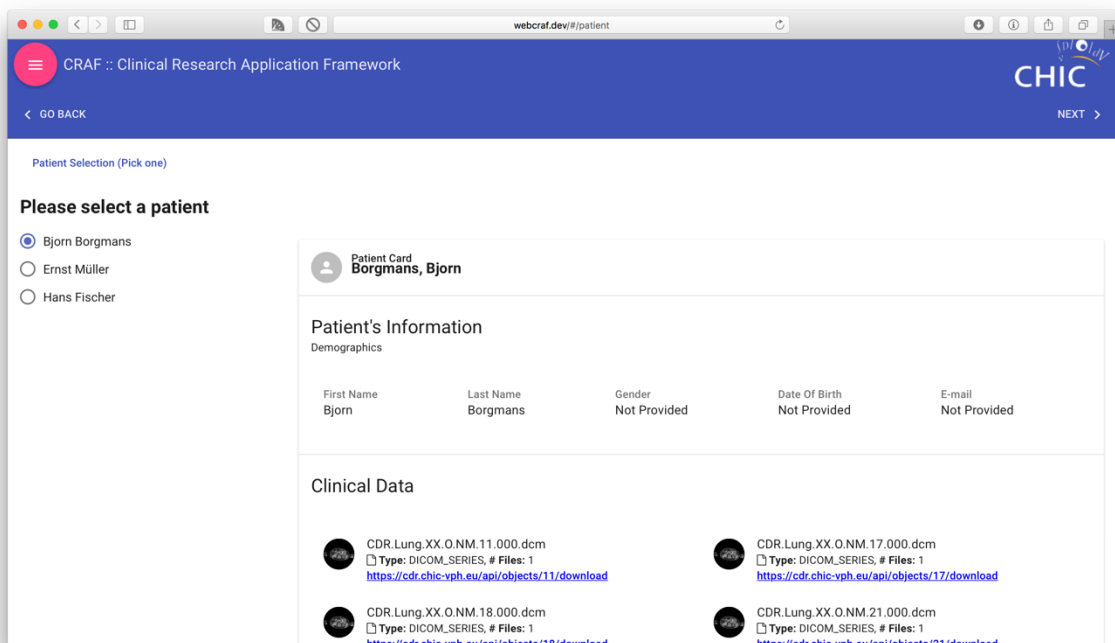


Figure 5 Patient selection

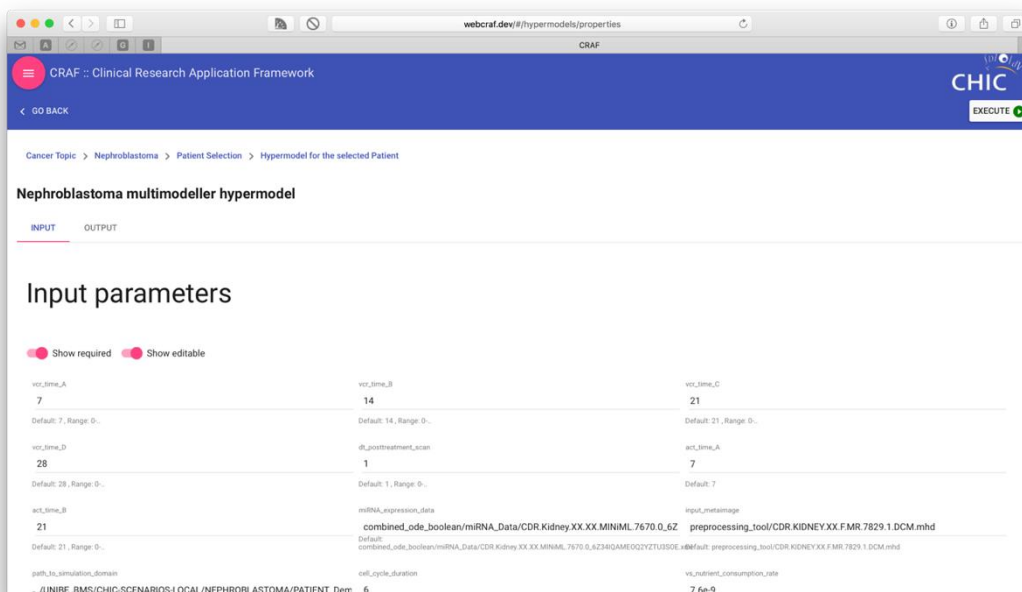


Figure 6 Overview of the input/output parameters before the execution of the chosen hypermodel for the selected patient and the chosen question

5.2.2 DrEye - Image processing toolkit

5.2.2.1 Description

The Doctor Eye suite serves as a flexible and extendable basis for the image processing toolkit. The suite offers a plugin-environment, for which a variety of image processing modules are being developed (e.g. for skull-stripping of brain images). The stand-alone software can process and visualize multimodal medical images (such as multimodal Magnetic Resonance Images) for qualitative as well as quantitative three-dimensional analysis of tumor shape. The plugins are developed using the state-of-the-art image processing library “Insight Registration & Segmentation Toolkit” (ITK) from Kitware.

5.2.2.2 Functionality

ID	Functionality
1	Loading and Writing of medical image data. Supported formats: <ul style="list-style-type: none"> • DICOM (.dcm) • Metainage with embedded header (.mha) • Metainage with separate header (.mhd + .raw)
2	Visualization of medical image data.
3	General image processing capabilities (<ul style="list-style-type: none"> • image smoothing, • resampling, • and more...

4	<p>Statistical analysis on selected regions of interest.</p> <ul style="list-style-type: none"> • Surface/volume estimation. • Computation of several statistics magnitudes • Histogram charts • and more...
5	<p>Offers a variety of tools for definition and refinement of manual annotations (Magic wand, Contouring tool and many manual selection tools for accurate delineations with voxel precision.).</p>
6	<p>Extendable through plugins</p>

5.2.2.1 CHIC components interaction

The DrEye application interacts mainly with the CCGVis component for the visualization of tumour and simulations. Implicit, manual interaction also takes place with the Clinical Data Repository for downloading of patient data and the CRAF application for the uploading of the segmentation results back to the CHIC platform and through the CHIC security services.

5.2.2.2 User Interface

Doctor Eye requires .NET framework (4.0. or higher). The integrated image processing routines rely on the Insight Registration and Segmentation Toolkit (ITK) library.

The figure below shows a screenshot of the graphical user interface of Dr. Eye (Version 4.7). As an exemplary case a T1-weighted post-contrast MR image of the brain is loaded with an annotated tumour (glioblastoma).

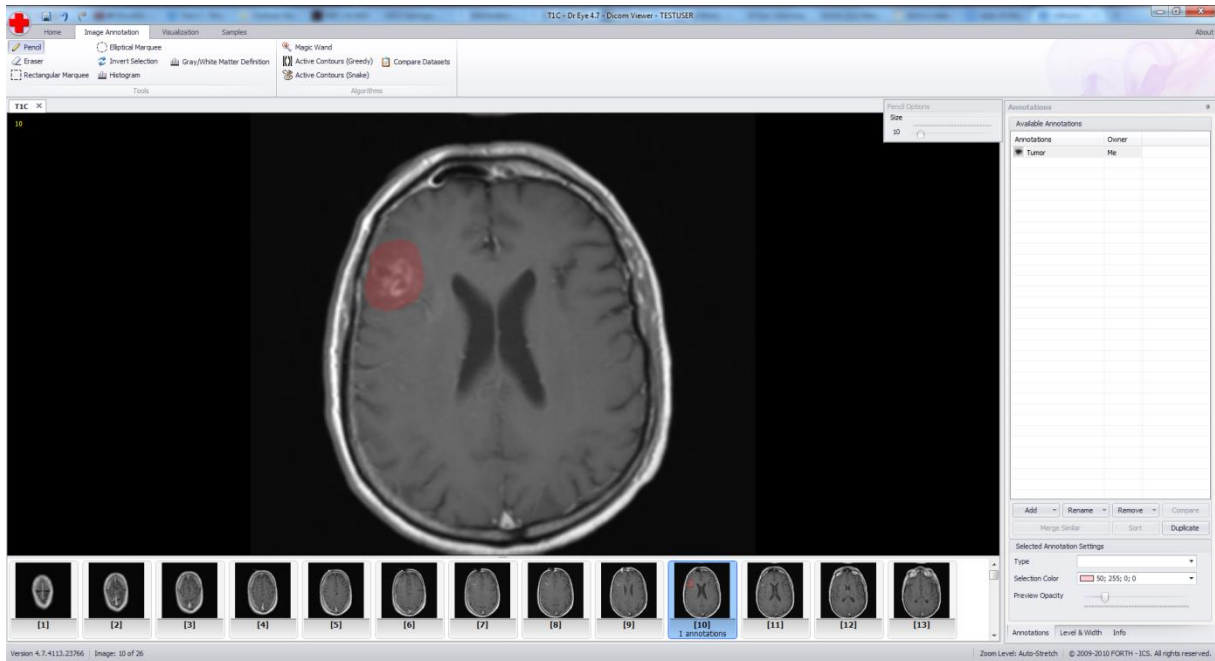


Figure 7 Screenshot of DrEye

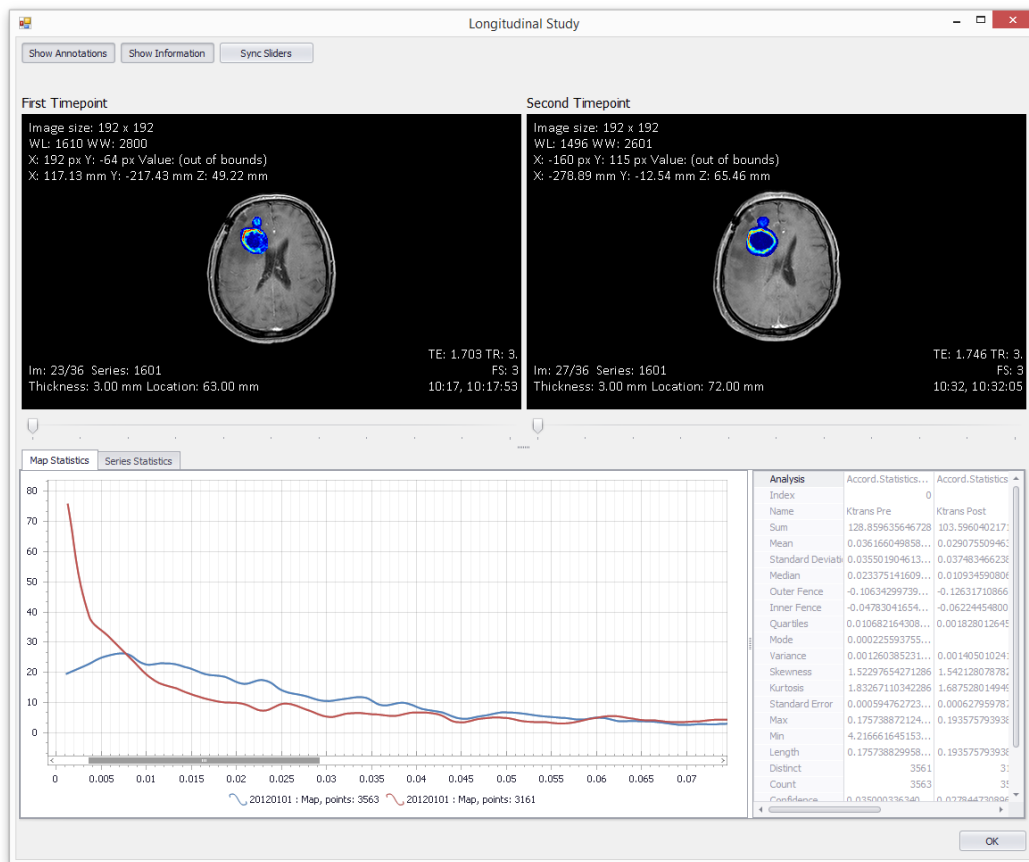


Figure 8 Side by side comparison of DICOM slides on two different time points, overview of the histograms and of the statistics magnitudes for the corresponding ROIs.

5.2.3 Hypermodelling editor

5.2.3.1 Description

The hypermodelling editor is the end user application used by modellers, researchers, and other domain experts for the construction of the hypermodels. The objective of this editor is to provide a user-friendly environment where the models and data are presented in a visual way and the users can integrate them into higher level modelling constructs.

Since the Editor is targeting the hypermodel design tasks, it mainly interacts with the model repositories, the VPH-HF execution framework, and the semantic infrastructure. In the following paragraphs we provide details on these interactions.

5.2.3.2 Functionality

ID	Functionality
1	List the available hypo and hyper models by contacting the relevant model repositories. This functionality is further enhanced by the ability to filter the available models based on user submitted search terms, annotation properties, etc. that are translated to proper semantic queries.
2	Provide a visual representation of the models in an intuitive graphical interface. In addition to the basic descriptive metadata such as the model descriptions, authors, citations, etc., it also presents the model’s inputs, parameters, and outputs with their semantic and syntactic type information.
3	Provide information about compatible datasets that can be used for the invocation of the selected models based on the semantic and syntactic information that models and data are annotated with. Such annotation information and relevant metadata are retrieved by the corresponding CHIC model and data repositories.
4	Support the visual linking and fusion of the models for the construction of higher level, more complex models (hyper-models). The semantic based descriptions of the models (meta models) are taken into account in order to facilitate this hyper-model construction.
5	Storage and retrieval of the built hyper-models with complete provenance and version control.
6	Export the designed hypermodels in the hypermodelling language adopted in CHIC
7	Submit the hypermodels for instantiation as computational entities and their subsequent execution to the Hypermodelling Orchestrator.

5.2.3.1 CHIC components interaction

The following is the complete list of the CHIC architectural elements that the Hypermodelling Editor interacts with:

- **Model Repository.** The Editor retrieves the model information from the Model Repository and presents it to the user as a list of available hypomodels to be used in the design of a new hypermodel. It also stores the designed hypermodels as new models in the Model Repository so that they can become hypomodels for even more complex hypermodels.
- **Semantic Services.** These services are used by the Editor in order to retrieve semantic metadata for the models and their parameters. Such rich semantic annotations are then used in order to guide the construction of new hypermodels and validate the integration and linking of different hypomodels.
- **VPH-HF model execution services.** The Editor submits the new hypermodels in the serialization format that is compatible with the CHIC Hypermodelling Language to the execution services so that they become executable artifacts of the architecture. When a given hypermodel is deployed in the execution framework, the Editor and the rest of the CHIC platform (most prominently, CRAF) can request its execution for some specified input parameter values and data.
- **Security services.** Being part of the CHIC platform, the Editor complies with the security framework of CHIC for the users' authentication and authorization. The Editor contacts the CHIC Identity Provider for the user authentication using the SAML-compliant mechanisms and the Secure Token Service (STS) for getting tokens to contact the backend CHIC services.

5.2.3.2 API interface/User Interface

The editor is accessible through a web user interface integrated into the CHIC portal. Additionally, it supports a web service read-only API for retrieving the constructed hypermodels and relevant information, such as the participating (hypo) models and their usage. This API is secured (authenticated and authorized) according to the adopted security framework in CHIC.

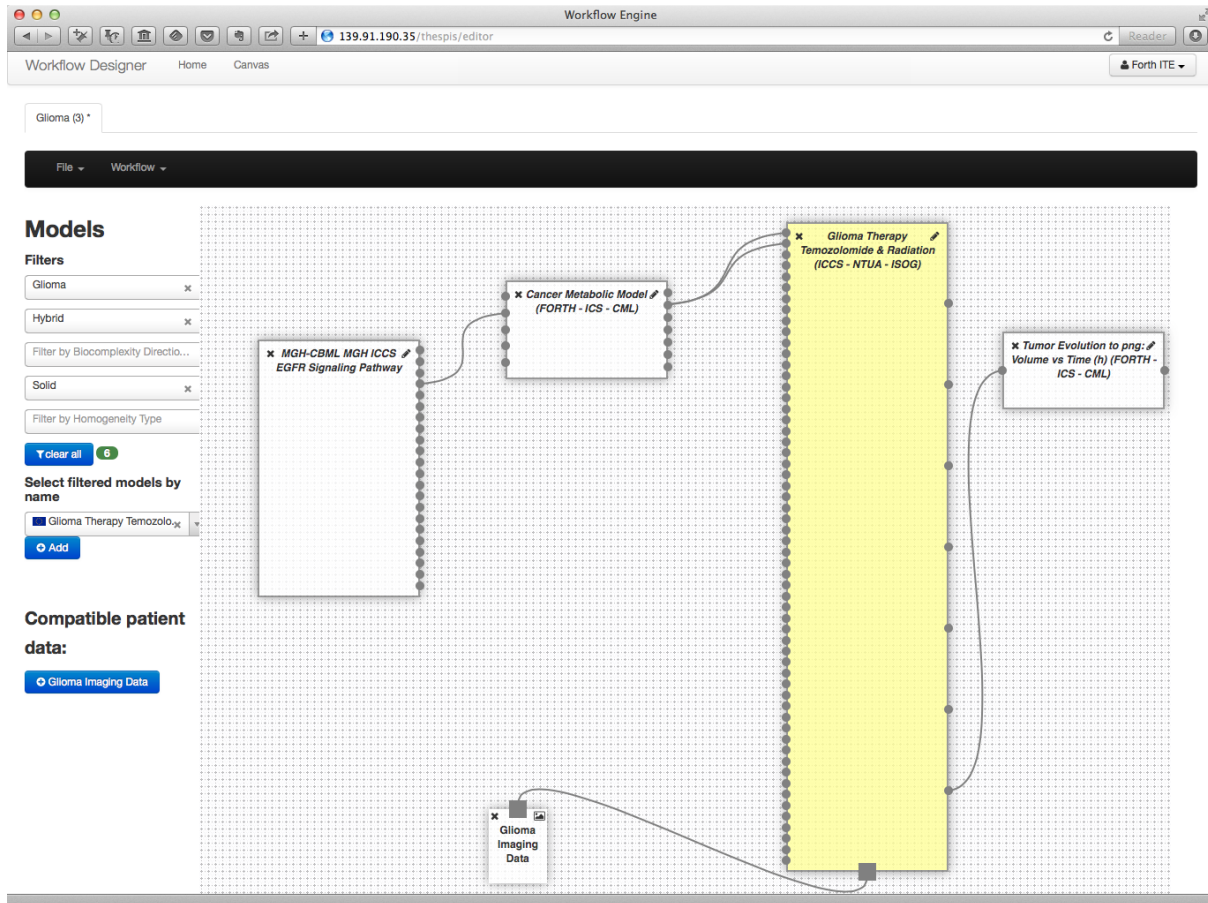


Figure 9 User interface of the hypermodelling editor.

5.3 Service layer

5.3.1 Hypermodelling execution framework

5.3.1.1 Description

The **VPH Hypermodelling Framework** (VPH-HF) is a collaborative computational platform providing a complete Problem Solving Environment⁹ (PSE) to execute, on distributed architectures, sophisticated predictive models involving patient medical data or specialized repositories. It is based on a fully-fledged prototype developed in a previous VPH project, the Osteoporotic VPH¹⁰ (VPH-OP), which addressed the estimation of bone fracture risk due to osteoporosis. For further information, the interested reader can refer to CHIC deliverable D7.4¹¹.

⁹ “Computer as thinker/doer: problem-solving environments for computational science,” IEEE Computational Science and Engineering. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=326669>. [Accessed: 29-Oct-2014].

¹⁰ <http://www.vphop.eu>

¹¹ FP7 - ICT - 600841, CHIC - Computational Horizons in Cancer, WP7: Hypermodelling infrastructure, D7.4 Final Hypermodelling framework deployed on test node, 2016

The aim of VPH-HF is to improve the effectiveness of diagnosis, prognosis and treatment of specific diseases in clinical practice with the ultimate objective to foster the personalized medicine paradigm and perform in silico clinical trials¹². In particular, the VPH-HF is customized for the oncological needs targeting two primary users: the **clinician** and the **researcher**. The former needs easy out-of-the-box software tools to analyse patient medical data and simulate cancer behaviour to address specific clinical questions. The latter has a broader profile that includes the creation, modification, and validation of complex integrative models¹³. Researchers can populate the CHIC data and model repositories with their experimental data and provide the integrative/predictive models implemented in the computational format of their convenience.

Following the use case described above, the VPH Hypermodelling Framework has been designed and developed as a technology with the aim to provide services and tools to allow:

- **Integration of models**, which can be developed with different software tools or libraries and be deployed in the PSE on different hardware and/or operating systems;
- Communication between the models, which can be classified into two types:
 - the **control flow**, which is the set of instructions that needs to be passed from one sub-model to another or to the system for its execution;
 - the **data flow**, which is the data input-output of each sub-model; in order to define this, the data formats used in both input and output by each sub-model have to be identified.

The underlying assumption is that a hypermodel (i.e. an integrative model or a composition/orchestration of models) can be described as a workflow where its composite hypomodels (i.e. models) are connected to produce an output result from a given input and data from repositories and/or patient specific data. Therefore, a workflow can be represented as a graph where the nodes are models or data repositories, while connections are data or control flows. Two models are connected when an output of the first is an input of the second, while data repositories can be connected to any of the models. In order to build a workflow, hypermodels and hypomodels can be considered as black boxes with a standardized abstract interface exposing input and output ports and control data flow. This interface is well defined within the CHIC project and it ensures the interoperability between all the provided hyper- and hypomodels: it is called **Component Model Generic Stub**. The VPH-HF is compliant with this interface and provides a software implementation that follows the Wrapper pattern¹⁴. It allows the actual integration of any of the computational instances of the models in a workflow including data and control flow by adapting the parameters from the format used in the actual model to the standard one of the Component Model Generic Stub Interface.

¹² G. Clermont, J. Bartels, R. Kumar, G. Constantine, Y. Vodovotz, and C. Chow, "In silico design of clinical trials: A method coming of age," *Critical Care Medicine*, vol. 32, no. 10, pp. 2061–2070, Oct. 2004.

¹³ J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. Ben Belgacem, D. Groen, and A. G. Hoekstra, "Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment," *Journal of Computational Science*, vol. 5, no. 5, pp. 719–731, Sep. 2014.

¹⁴ Coveney, PV; Saksena, RS; Zasada, SJ; McKeown, M; Pickles, S; (2007) The application hosting environment: Lightweight middleware for grid-based computational science. *COMPUT PHYS COMMUN* , 176 (6) 406 - 418. [10.1016/j.cpc.2006.11.011](https://doi.org/10.1016/j.cpc.2006.11.011).

5.3.1.2 Functionality

ID	Functionality
1	Deploy automatically the models in the computational infrastructure in order to be run from VPH-HF.
2	Authorise the execution requests and the access to the necessary input data using the CHIC authentication services.
3	Orchestrate the execution of a hypermodel with a given input set, managing both the control flow and the data flow among the hypomodels that compose it.
4	Maintain a global log of all hypomodel executions and of their orchestration, to allow debugging and execution monitoring.
5	Replicate in the in silico trials repository the results of the hypermodel execution for a given input set.

5.3.1.1 CHIC components interaction

The hypermodelling execution framework interacts with the following CHIC components:

1. **Authentication Service:** VPH-HF is a non-browser REST client of the CHIC authentication service
2. **CRAF:** CRAF interacts with VPH-HF for the submission and monitor of hypermodel executions
3. **Hypermodelling editor:** the hypermodelling editor is responsible for the creation of hypermodels in the xMML language which is one of the format accepted by VPH-HF for the submission of workflows
4. **CHIC repositories (Model repository, Clinical Data repository, InSilicoTrial repository):** VPH-HF interacts with the Clinical Data repository for the retrieval of input medical data for the workflow execution; interacts with the InSilicoTrial repository for the update of the status and the upload of the output of workflow executions; interacts with the Model repository to get descriptive model information which are used for the automatic deployment of models.

5.3.1.2 API interface/User Interface

5.3.1.2.1 User Interface

The **User Interfaces** are the components which are used by the end-users to interact with the back-end of the VPH hypermodelling framework to allow the creation, submission, and monitoring of a hypermodel execution. The CRAF application is an external component of the VPH-HF that allows selection and execution of existing hypermodels in the cancer domain in order to address clinical questions. From CRAF it is also possible to monitor the execution of the hypermodel and to visualize the output results. Inside the hypermodelling framework a basic web interface (called “Admin panel-dashboards”) was designed and implemented to execute existing hypermodels on available patient’s data and a dashboard to monitor the execution, retrieve the results and analyse the logs. The latter was implemented in order to provide a simple and direct interface to VPH-HF for independent testing and not intended to be used directly by the CHIC users.

The final version of the VPH-HF admin panel-dashboard allows the user to:

- login into the system (by connecting to the authentication service);
- see the available hypomodels, hypermodels and input datasets (by connecting to the registry service);
- add new input dataset/hypomodel/hypermodel with the core metadata (by connecting to the storage/registry services);
- submit and start a hypermodel for execution to the WMS with the necessary data (by connecting to the Director service);
- see logs/status of the already submitted executions (by connecting to the Director);
- retrieve the output of the workflow execution (by connecting to Director/SMS).

Figure 10 shows a snapshot of the admin panel-dashboard.

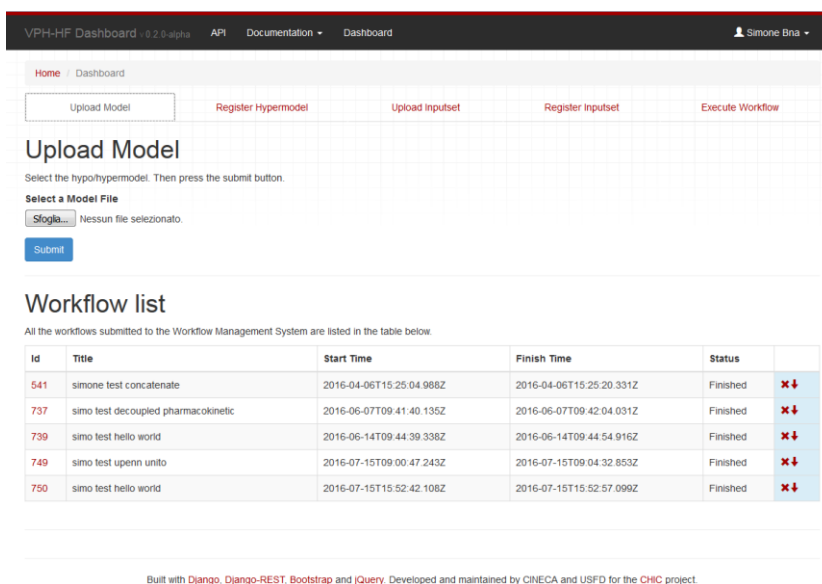


Figure 10 VPH-HF admin panel-dashboard

5.3.1.2.2 API Interface

One of the components of the hypermodelling framework that interacts with the rest of the CHIC platform is the Director service. **Director** acts as an interface between the end-user application (CRAF, dashboard) and the backend hypermodel technology and exposes the APIs to access the different VPH-HF functionalities needed from outside the framework. It aims to provide the functionalities to manage the request of a hypo/hypermodel execution. In particular, it submits the execution to the workflow management service passing it the required inputs/parameters, polls the workflow management service to get information on the status and other parameters of the hypo/hypermodel execution and calls the storage services to save the output in a storage repository.

The description of the APIs are reported below (as reported in Deliverable D7.4).

HTTP Method: GET URL: /api/director/workflowlist/		
Description	Get a list of all the workflows submitted to Director by the user	
Returns	200 http status code and a Json object containing the list of workflow objects if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: POST URL: /api/director/workflowlist/		
Description	Submit a new workflow	
Encoding	multipart/form-data	
Parameters passed through request body	workflow_title	Not Required – title of the workflow
	workflow_description	Not Required – description of the workflow
	workflow_comment	Not Required – comments about the workflow
	experiment_id	Required
	subject_out_id	Required
	model_url	Required
	inputset_url	Required
Returns	201 http status code and a Json object containing the new created workflow object if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET URL: /api/director/workflowlist/{id}/		
Description	Get the model data of the workflow {id}	
Returns	200 http status code and a Json object containing the workflow object if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the workflow {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: DELETE URL: /api/director/workflowlist/{id}/		
Description	Delete the workflow {id}	
Returns	204 http status code if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the workflow {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET URL: /api/director/workflowlist/{id}/status		
Description	Get the status of the workflow run	
Returns	200 http status code and a Json object containing the workflow status of the workflow object {id} if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not	

	verified	
	404 http status code if the workflow {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: PUT URL: /api/director/workflowlist/{id}/status		
Description	Change the status of the workflow	
Encoding	multipart/form-data	
Parameters passed through request body	workflow_status	Required
Returns	200 http status code and a Json object containing the workflow status of the workflow object {id} if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the workflow {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET URL: /api/director/workflowlist/{id}/modellog		
Description	Get the logs of the workflow	
Returns	200 http status code and a Json object containing the logs of the hypomodels run in the workflow {id} if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the workflow {id} is not found	
	500 http status code if an internal server error occurs	

HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>
-------------	---------------------	--

HTTP Method: GET URL: /api/director/workflowlist/{id}/createtime		
Description	Get the time when the workflow has been created	
Returns	200 http status code and a Json object containing the start time of the workflow object {id} if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the workflow {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET URL: /api/director/workflowlist/{id}/starttime		
Description	Get the time when the workflow has been started	
Returns	200 http status code and a Json object containing the start time of the workflow object {id} if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the workflow {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET URL: /api/director/workflowlist/{id}/finishtime	
Description	Get the time when the workflow {id} has been finished

Returns	200 http status code and a Json object containing the finish time of the workflow object {id} if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the workflow {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET URL: /api/director/workflowlist/{id}/expirytime		
Description	Get the time when the workflow {id} has been expired	
Returns	200 http status code and a Json object containing the expiry time of the workflow object {id} if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the workflow {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET URL: /api/director/workflowlist/{id}/exitcode		
Description	Get the exit code of the workflow run {id}	
Returns	200 http status code and a Json object containing the exit code of the workflow object {id} if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the workflow {id} is not found	

	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

CRAF communicates with the Director of the VPH-HF server using the SAML standard data format over the HTTPS protocol in order to submit and execute a hypermodel. This is a synchronous communication over the HTTP protocol where the response contains the identification (UUID) of the newly created run of the corresponding hyper model. On the other hand, since the exact completion time point is not known, and a continuous “polling” mechanism would be too inefficient, the execution status information is sent asynchronously. In particular, CRAF receives back notifications from VPH-HF about the status of the hypermodel (OPERATING, FINISHED_SUCCESSFULLY, FINISHED_ERRONEOUSLY) with RabbitMQ as intermediary. Following this approach CRAF and VPH-HF are operationally decoupled. VPH-HF sends all the notifications to RabbitMQ using a topic and “durable **exchange**” named “vphhf” with a **routing key** set to “workflow.<workflow_uuid>.status”, where <workflow_uuid> is a unique identifier of the workflow run. CRAF instead creates a message queue bound to the ‘vphhf’ exchange with a binding key set to workflow.*.status” in order to listen to the status of all the workflows.

The **Registry** application is the component of the hypermodelling framework that provides to the CHIC platform a list of all the models accessible within the VPH-HF instance and the automatic deployment of models stored in the Model Repository within the hypermodelling infrastructure. The models that are successfully deployed can be included in a hypermodel workflow and executed in the computational infrastructure. The description of the APIs are reported below (in addition to what has been reported in deliverable 7.4).

HTTP Method: GET		URL: /api/registry/modellist/mr-chic-vph-eu
Description	Get the list of models registered in the Model repository	
Returns	200 http status code and a Json object containing the list of model objects	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET URL: /api/registry/modellist/mr-chic-vph-eu/{id}		
Description	Get the data of the model {id} registered in the Model Repository	
Returns	200 http status code and a Json object containing the model object	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the model {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET URL: /api/registry/modellist/mr-chic-vph-eu/{id}/binariesandscripts/deploy		
Description	Deploy the model {id} registered in the Model Repository in the hypermodelling infrastructure	
Returns	200 http status code	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the model {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET URL: /api/registry/modellist/mr-chic-vph-eu/{id}/tests/run		
Description	Run the unit tests for the deployed model {id} registered in the Model Repository	
Returns	200 http status code	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	

	404 http status code if the model {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

The last two APIs delegates to a job queue the tasks of deploying the binaries and scripts and running the unit tests since immediate results cannot be computed. Following the Publish/Subscribe pattern, the status and the message error are notified back asynchronously. All the notifications are sent to RabbitMQ using a topic and “durable **exchange**” named “vphhf” with a **routing key** set to “model.<model_id>.deployment.detailed_status” for the deployment notifications and a routing key set to “model.<model_id>.tests.detailed_status” for the unit test report notifications, where <model_id> is the id of the model registered in the Model repository. The content of the message is a JSON object containing the status of the task and the message error (the message error is “null” if the task finishes successfully).

5.3.2 Visualization toolkit

5.3.2.1 Description

The visualization toolkit developed within CHIC will consist on a set of modular functionalities allowing the different users to perform from basic to advanced visualization operations. The intention is to allow end-users, such as clinicians and practitioners, to access the toolkit through the plug-in enabled software.

5.3.2.2 Functionality

ID	Functionality
1	Visualization of the model and data repository structures
2	Visualization of the information related to models such as their parameters and execution results

5.3.2.1 CHIC components interaction

In the CHIC architecture, CCGVis is launched by DrEye.

5.3.2.2 User Interface

CCGVis is the CHIC tool for visualizing and comparing tumours and simulations. CCGVis can import, register and visualize medical data, segmentation data and simulations. It can import various

formats, singly or as time series, including dicom, mha, nifti and CHIC simulations. Visualizations include slice and orthoslice views in 2D and 3D, isosurfaces in 3D, comparisons between real and simulated tumours, and plots of tumour growth. CCGVis can be executed as a standalone desktop application, or it can be launched from another application with command-line arguments. Input and output data is exchanged via the local file system, in directories specified by the command line arguments.

The user interface consists of a display window on the left, a widget panel for interactive control on the right, and menus for loading data and launching visualization tasks. The display window and widget panel are empty on startup.

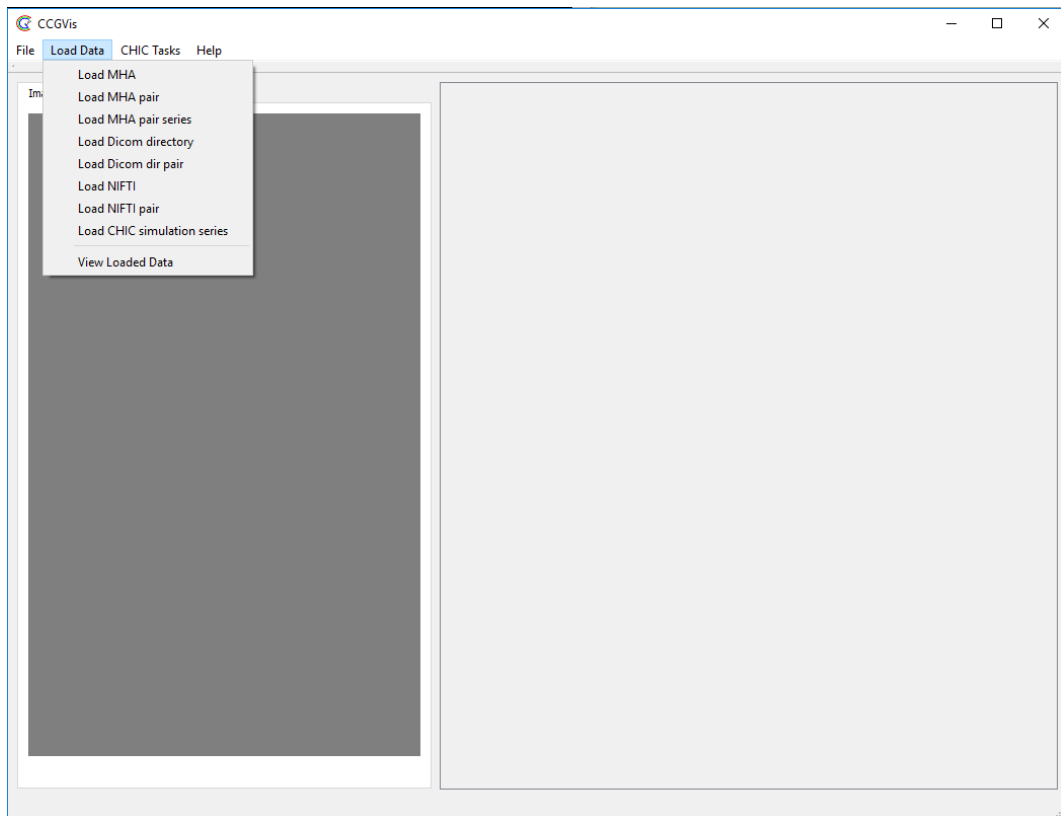


Figure 11 CCGVis user interface

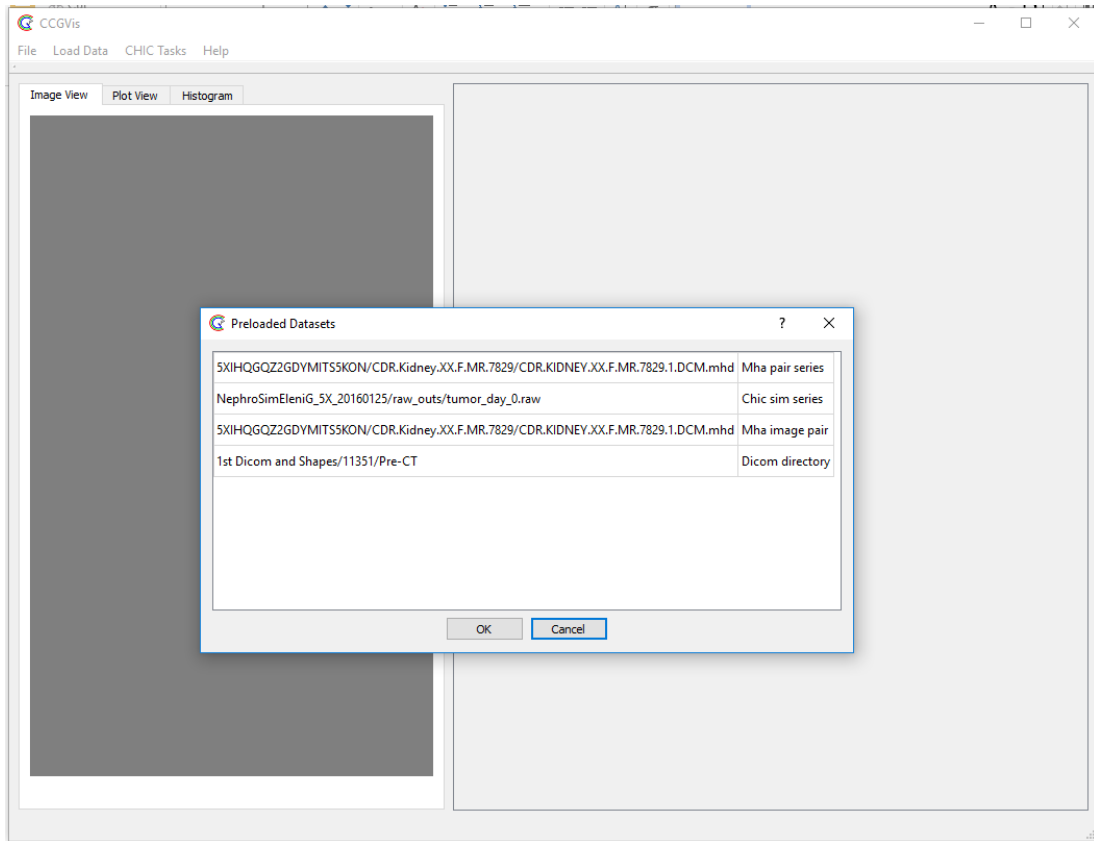


Figure 12 CCGVis loading data

The “Load Data” menu allows the user to manually import datasets. Data formats are mha/mhd, nifti, dicom and CHIC simulation format. Datasets can be single images, image/segmentation pairs, or time series. The “View Loaded Data” menu item displays the current list of loaded datasets.

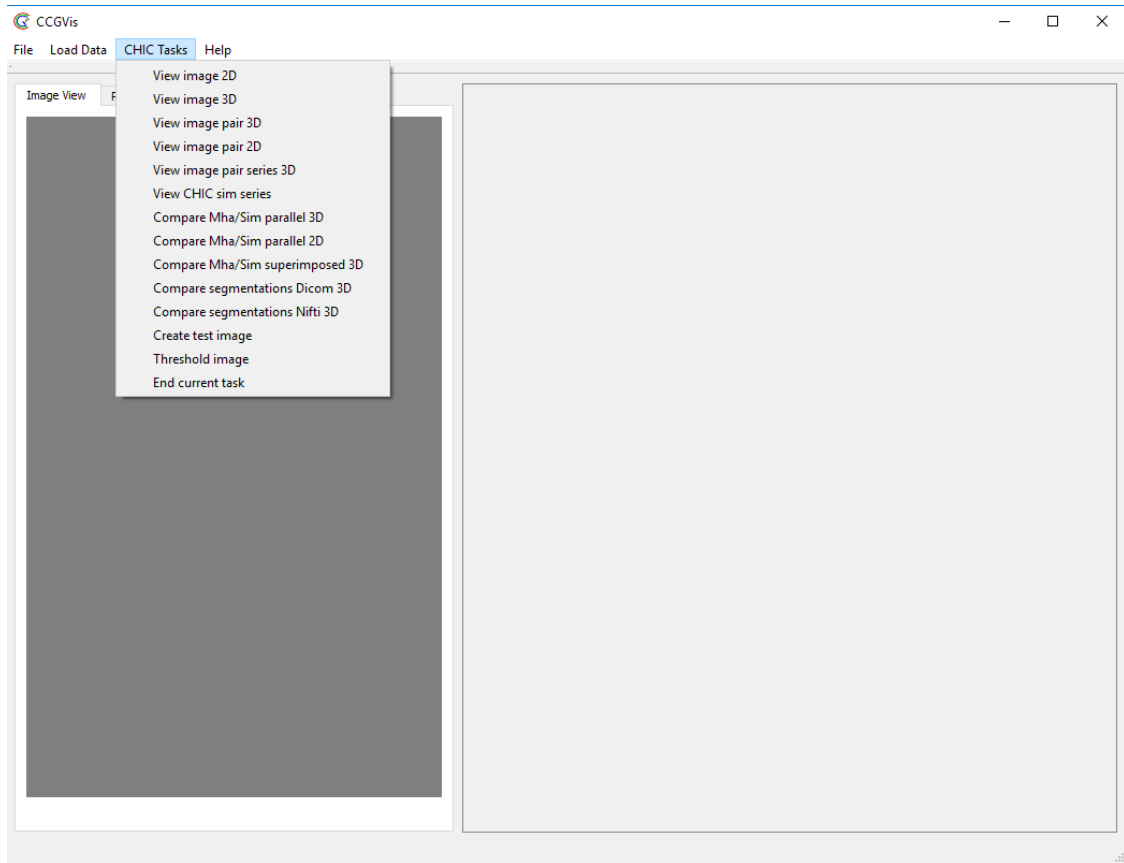


Figure 13 CCGVis task execution

The “CHIC Tasks” menu displays the list of visualization tasks. Selecting a task invokes a search of the loaded data for compatible input data, and then displays the visualization. If more than one compatible dataset is found, the user is asked to choose from the list. The “End current task” item clears the visualization and returns to the default blank screen.

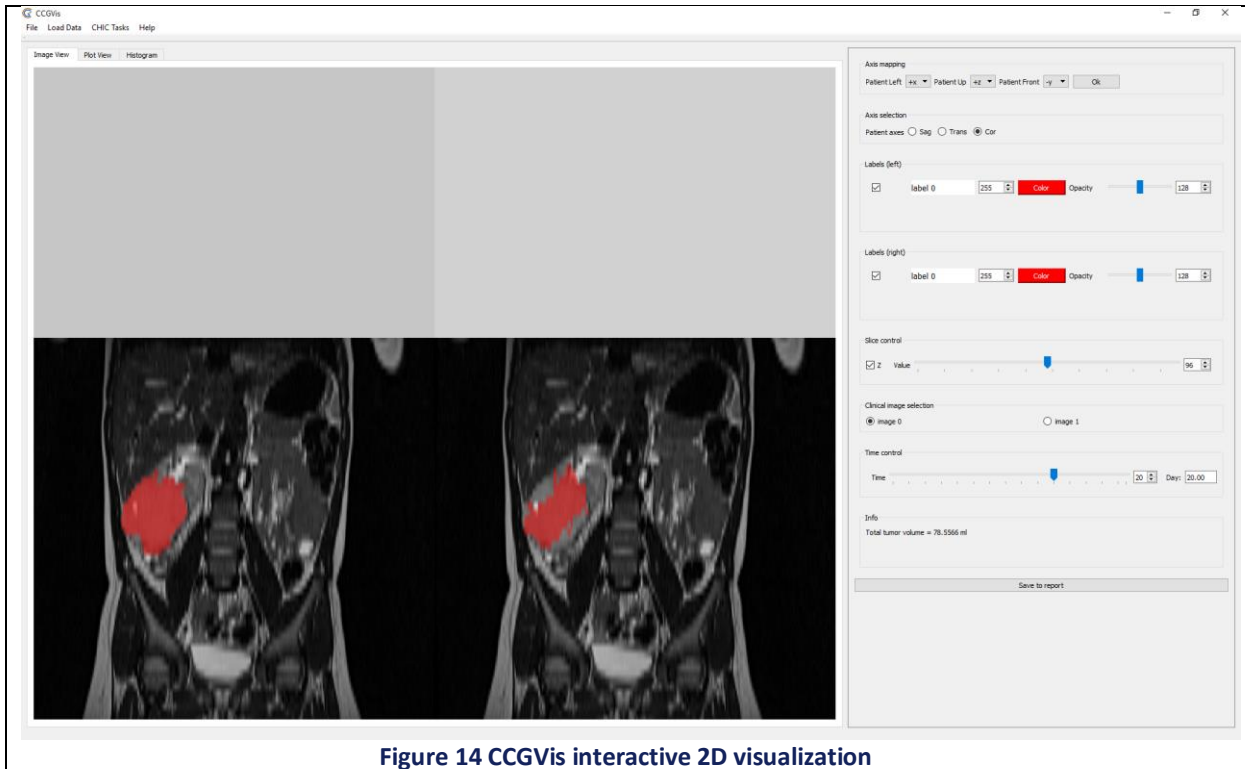


Figure 14 CCGVis interactive 2D visualization

The clinical tumour (left) and the simulation (right) can be compared in parallel with an interactive 2D visualization. The tumour volume is displayed in an info panel near the lower right.

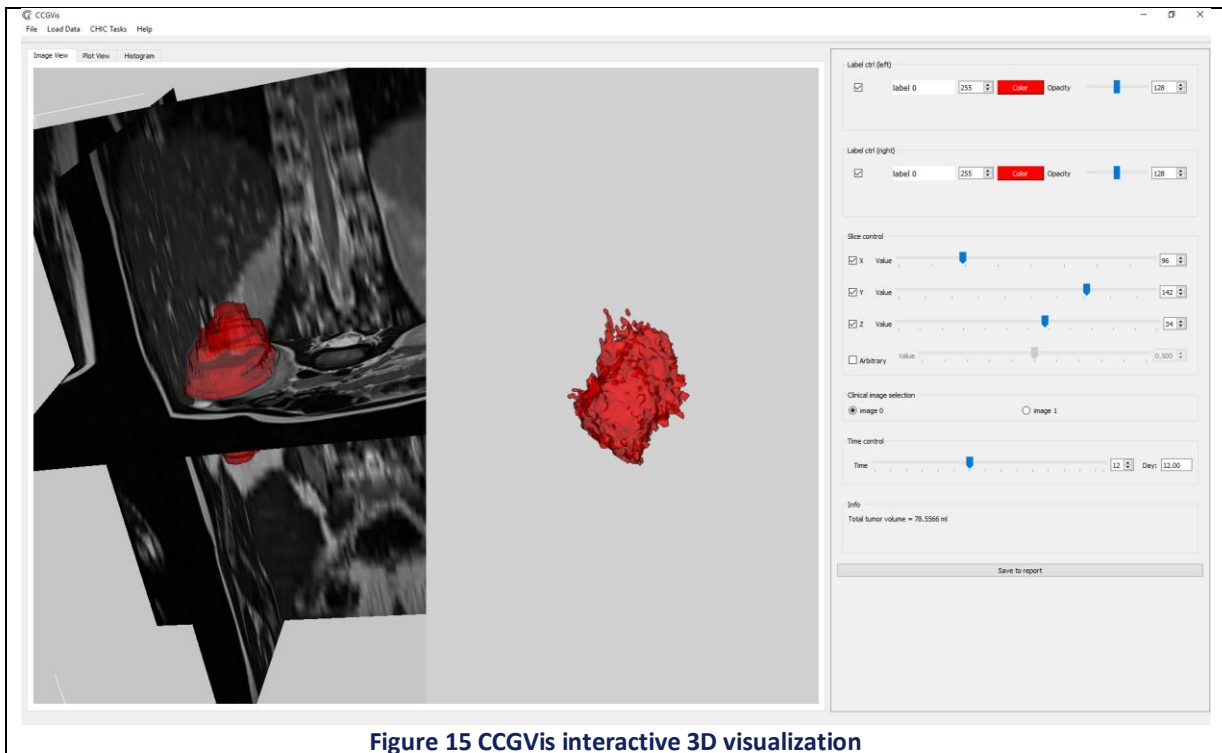


Figure 15 CCGVis interactive 3D visualization

The clinical tumour (left) and the simulation (right) can be compared in parallel with an interactive 3D visualization.

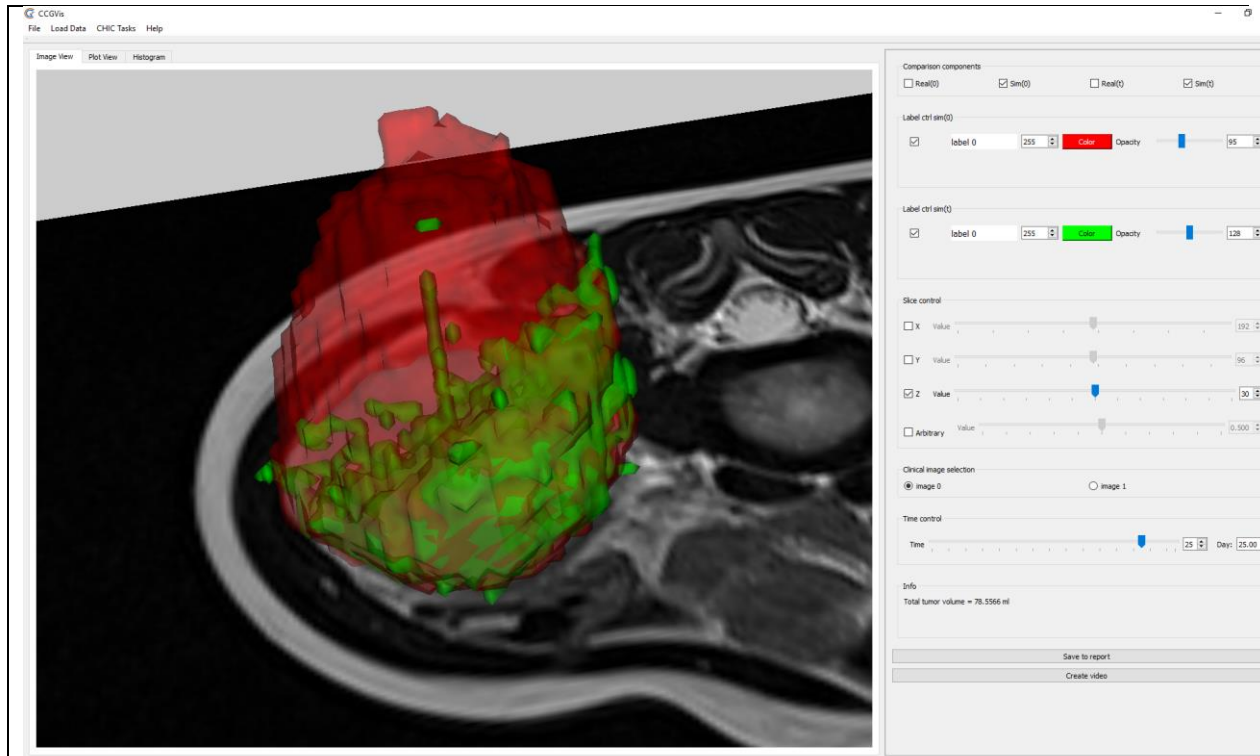
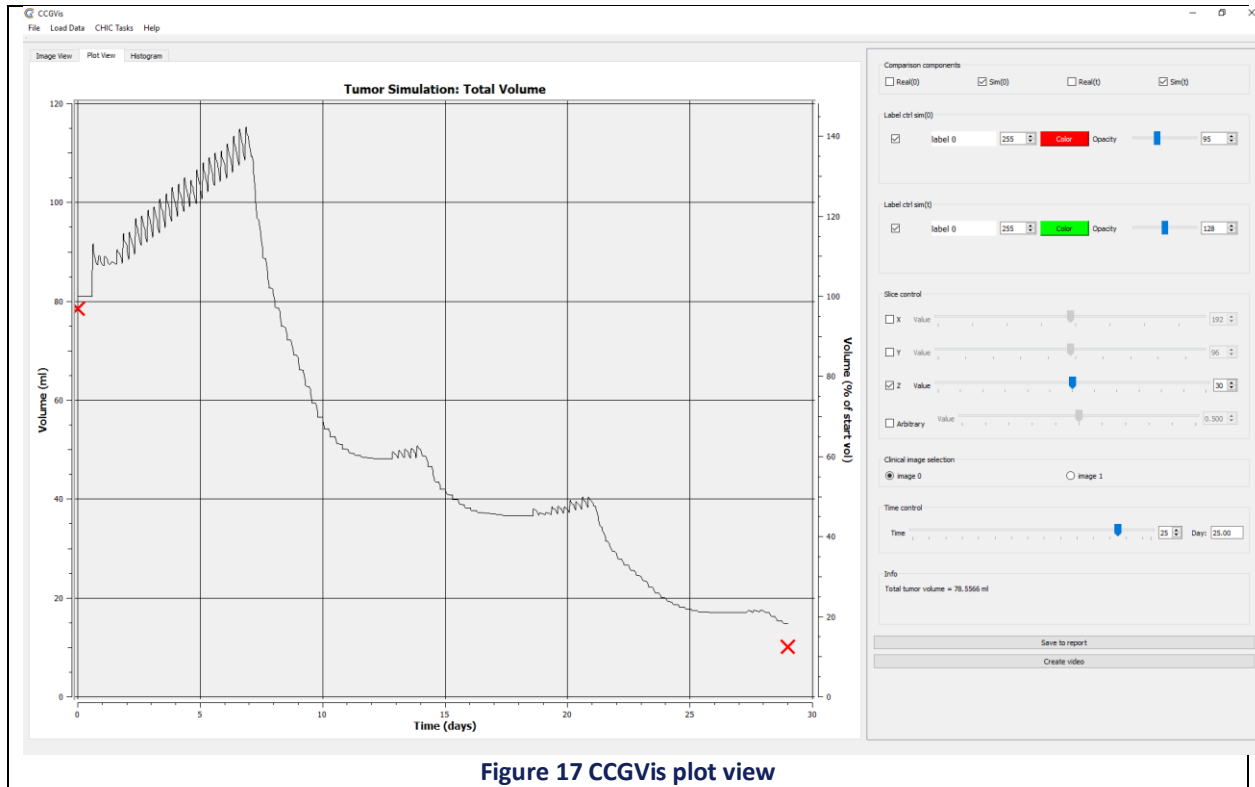
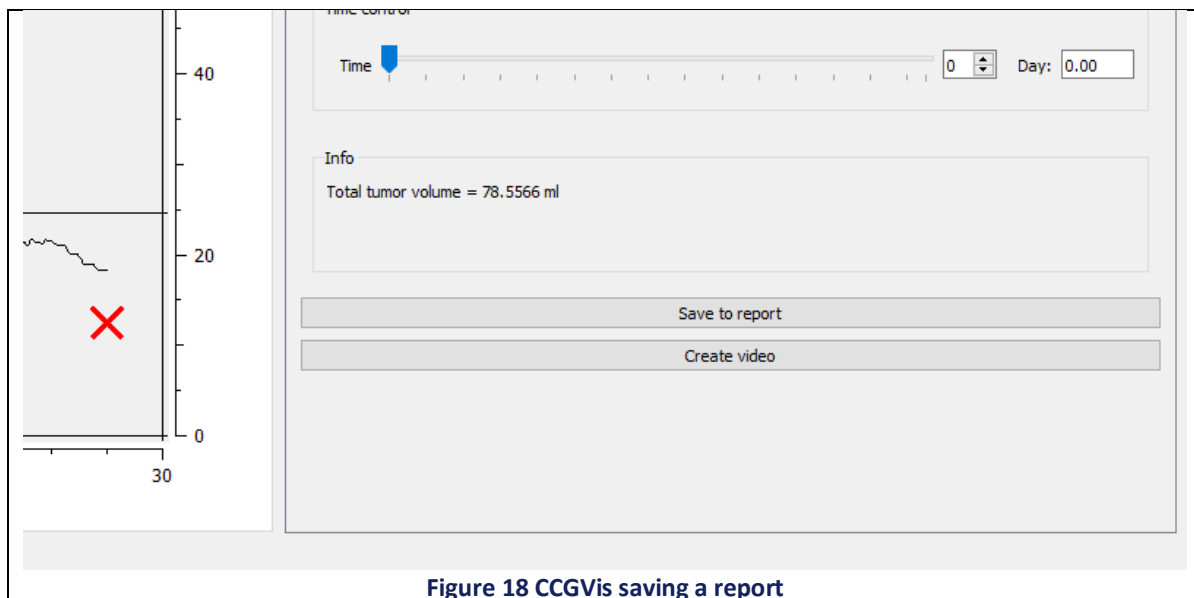


Figure 16 CCGVis comparison of simulation using superimposed isosurfaces

The clinical images and simulations can be compared with each other at different timesteps in 3D as superimposed isosurfaces. This shows the day 25 simulation (green) compared with the baseline tumour on day 0 (red).



The “Plot View” tab shows a graph of simulated tumour volume against time. Data points (red crosses) show the actual volume of the clinical tumour.



The “Save to report” button saves an output image of the current image or plot in the main window. A caption describing the image is added to a text file. The “Create video” button creates and saves an animated 360° video of a 3D visualization. The outputs are sent to the output directory specified in the command line arguments.

5.3.2.3 API interface

Command line arguments allow data to be preloaded and tasks to be launched on execution. This allows CCGVis to be launched from a batch file or from another program. Command line flags are also available for setting the input and output directories.

<i>Valid flags</i>
--Scenario: "Lung", "Nephroblastoma", "Prostate", "Glioblastoma". Not currently used.
--Dataset: Starts a new dataset.
--Format str: specifies format of dataset.
--ImageSegPair: (optional) indicates that the following 2 files are a pair: image plus segmentation. A pair counts as one data item with two components.
--TimeSeries n: (optional) the dataset is a time series containing n items.
--InputDir: the next item specifies a general default path for input data.
--OutputDir: the next item is the path where CCGVis will put its output.
--Task str: specify task to run on launch
--ShowMaximised: request window maximised on launch
--AutoVideo: create video output when task is launched without user interaction.

<i>Valid format strings</i>	
"Mha"	"DicomDir"
"Nifti"	"ChicSimRaw"

<i>Valid task strings</i>	
"ViewImage2D"	"ViewDicomDir2D"
"ViewImage3D"	"ViewDicomDirOrtho"
"ViewImagePair2D"	"ViewMhaPair2D"
"ViewImagePair3D"	"ViewMha"
"ViewImagePairSeries3D"	"ViewMhaSeries"
"ViewChicSim"	"CompareMhaChicSimOrtho"
"CompareMhaChicSim2D"	"CompareMhaChicSimSuperimposed"
"CompareTwoDicomSegmentations3D"	"CompareTwoNiftiSegmentations3D"
"DoNothing"	

<i>Command line Examples</i>
1) Load single image in dicom format (note that filenames must be full paths) --Dataset --Format DicomDir <file1>
2) Load image and its segmentation, both in Mha format --Dataset --Format Mha --ImageSegPair <imfile> <segfile>
3) Load mha time series consisting of 3 image/segmentation pairs. --Dataset -TimeSeries 3 --Format Mha -ImageSegPair <im1> <seg1> <im2> <seg2> <im3> <seg3>
4) Load CHIC simulation series (no need for TimeSeries flag as it can't be anything else) --Dataset --Format ChicSimRaw <dir>
5) Load 2 datasets: an mha time series and its corresponding simulation --Dataset -TimeSeries 2 --Format Mha -ImageSegPair <im1> <seg1> <im2> <seg2> --Dataset --Format ChicSimRaw <pathSim>
6) Set CCGVis default data directory and leave user to browse --InputDir <path>
7) Specify directory for CCGVis output. --OutputDir <path>
8) Load 2 datasets as in (5) and launch 2D comparison view. Launch with window maximised. --Dataset -TimeSeries 2 --Format Mha -ImageSegPair <im1> <seg1> <im2> <seg2> --Dataset --Format ChicSimRaw <pathSim>


```

--OutputDir <path>
--Task CompareMhaChicSim2D
--ShowMaximised
9) Load 2 datasets as in (5) and launch superimposed isosurface comparison view.
Create automatic video output on launch.
--Dataset -TimeSeries 2 --Format Mha -ImageSegPair <im1> <seg1> <im2>
<seg2>
--Dataset --Format ChicSimRaw <pathSim>
--OutputDir <path>
--Task CompareMhaChicSim2D
--ShowMaximised
--AutoVideo

```

5.3.3 Security services framework

5.3.3.1 Description

The security framework is not a concrete software component but a set of tools, which deal with the security aspects of CHIC’s technological platform, ranging from user authentication, authorization, and auditing, to data integrity and privacy.

The security tools and policies of this framework enforce the legal and regulatory compliance and encompass the appropriate auditing mechanisms which are needed by the legislation.

The security framework contains the following tools & components:

1. Authentication and Identity Management Components

- a. **Identity and Access Management Site (IAM)**
User enrolment and management site and services through which users can be enrolled, revoked, edited ...
- b. **Identity Provider (IdP)**
A service which is responsible for authentication within CHIC. It provides identity assertions to web sites accessed from a browser.
- c. **Secure Token Service (STS)**
A service which is responsible for authentication within CHIC. It provides identity assertions to services accessed from a non-browser client.

2. Authorization components

- a. **Policy Decision Point (PDP)**
A PDP is the entity which takes authorization decisions. A PDP accepts authorization requests.
- b. **Policy Administration Point (PAP)**
A PAP is the endpoint which manages policies. The PAP provides the PDP with all policies required to produce an authorization decision. The PAP also consists of management services where authorization rules can be configured generating authorization policies.
- c. **Policy Information Point (PIP)**
Policies evaluated by the PDP need information (attributes) about the actors to be able to take an authorization decision. Most attributes are typically provided through the authorization requests. A PIP is responsible for providing the missing attributes to the PDP.

d. Policy Enforcement Point (PEP)

A PEP is the component which integrates with application code. The PEP is responsible for creating the authorization request and sends it to the PDP.

3. Audit Service

4. Security gateway/proxy

5. Integration modules and extensions

- a. Various integration modules are available within CHIC to integrate JAVA, PHP and .NET applications into the security framework.
- b. Extensions
- c. Liferay Extension

6. De-identification tool (CATS)

An elaborate documentation of the security framework is given in the CHIC deliverable D5.2.2 “Final version of security tools and guidelines”. In this section we give a brief overview of the functionality of the security framework for the sake of clarity and coherence when referring to it while discussing the rest of the architecture.

5.3.3.2 IAM (Identity and Access Management site)

5.3.3.2.1 Description

Through the CHIC central Identity and Access Management site, users and user credentials can be created and managed. IAM allows in addition the attributes of a user such as his roles, rights and group memberships to be managed.

5.3.3.2.2 Functionality

ID	Functionality
1	User enrolment (registration and activation)
2	Credential recovery (both username as password)
3	Management of user roles and rights, group memberships, attributes
4	User account revocation
5	Profile management allow users to manage their own demographic information/profile.

5.3.3.2.3 Programmatic/User Interface

IAM features a web based graphical user interface towards end users and user administrators.

IAM publishes web services to allow programmatic integration so that users can be enrolled and managed through other CHIC applications.

5.3.3.3 IdP (Identity Provider)

5.3.3.3.1 Description

The IdP is responsible for authentication users who access CHIC from a browser. It provides identity assertions to web sites accessed from a browser.

5.3.3.3.2 Functionality

ID	Functionality
1	User Authentication (Web Browser)
2	Issue Identity Assertions (SAML)

5.3.3.3.3 Programmatic/User Interface

HTTP interfaces as defined in the SAML Web Browser SSO Profile. More detailed information on security profiles can be found in D5.2.2.

5.3.3.4 STS (Secure Token Service)

5.3.3.4.1 Description

The STS is responsible for authenticating users and clients who wish to access CHIC through web services. Web services are typically accessed through non browser clients. The STS will issue, after successful authentication, tokens that can be used by a client to access a CHIC web services. The STS can also validate, renew and cancel tokens.

5.3.3.4.2 Functionality

ID	Functionality
1	User Authentication (web service client)
2	Issue Identity Assertions (SAML tokens)
3	Validate Identity Assertions
4	Renew Identity Assertions
5	Cancel Identity Assertions

5.3.3.4.3 Programmatic/User Interface

The STS provides web services that can be used by a client to issue, renew, validate and cancel SAML tokens.

5.3.3.5 PDP (Policy Decision Points)

5.3.3.5.1 Description

A PDP is the entity which takes authorization decisions by responding on authorization requests.

5.3.3.5.2 Functionality

ID	Functionality
1	Take authorization decisions: permit or deny

5.3.3.5.3 Programmatic/User Interface

A web services interface is available allowing other components and services to call the authorization service.

5.3.3.6 PAP (Policy Administration Point)

5.3.3.6.1 Description

The PAP's management interfaces allow the configuration of authorization rules through which policies are generated. These are then provided to the PDP so that the PDP can use them to produce authorization decisions.

5.3.3.6.2 Functionality

ID	Functionality
1	Create Authorization policies
2	Provide the PDP with Authorization policies

5.3.3.6.3 Programmatic/User Interface

A web based graphical user interface is available through which the authorization policies can be managed. Web services allow the PDP to query the PAP for policies.

5.3.3.7 PIP (Policy Information Point)

5.3.3.7.1 Description

Policies evaluated by the PDP need information (attributes) about the actors to be able to take authorization decision. Most attributes are typically provided through the authorization request. A PIP is responsible for providing the missing attributes to the PDP.

Within CHIC any component which could provide information on the subject or resource of a request is able to act as a PIP. Possible PIPs are the central user repository and the portal.

5.3.3.7.2 Functionality

ID	Functionality
1	Query a subject or resource attribute

5.3.3.7.3 Programmatic/User Interface

A Web Service should be published by a PIP, which would be query-able by the PDP.

5.3.3.8 PEP (Policy Enforcement Point)

5.3.3.8.1 Description

A PEP is a software component or library which integrates with applications that should create authorization requests and call the PDP.

5.3.3.8.2 Functionality

ID	Functionality
1	Create an authorization request, send it to the PDP and parse the PDP response.

5.3.3.8.3 Programmatic/User Interface

The PEP is typically a programming interface that can be integrated with application code.

5.3.3.9 Audit Services

5.3.3.9.1 Description

Through the CHIC Audit Interfaces secured CHIC services should log all actions performed. These audit logs allow the detection of system breaches or the misuse of access rights. In additions audit logs can be used through usage patterns to automatically identify suspicious behaviour.

5.3.3.9.2 Functionality

ID	Functionality
1	Provides a central audit bus to which all CHIC services can send audit logs.
2	Provides a management interface through which audit messages can be viewed and searched.

5.3.3.9.3 Programmatic/User Interface

Web Services (and other protocol endpoints such as syslog) are available through which audit messages can be logged. A web based graphical user interface provides a view on the logged audit messages.

5.3.3.10 Security Gateway

5.3.3.10.1 Description

Not every service provider can easily integrate with the CHIC security framework. To enable these services to be integrated with the security framework a security gateway is available which is

responsible for handling all security protocols. Identity information is then passed by the gateway to backend service providers through HTTP headers.

5.3.3.10.2 Functionality

ID	Functionality
1	Handle the security protocols to hide it from backend applications. This implies the following functionality <ol style="list-style-type: none"> 1. Authentication 2. Authorization 3. Auditing

5.3.3.10.3 Programmatic/User Interface

The security gateway is deployed as a gateway or proxy in between the application server and the client. It thus mimics the backend application’s interface.

5.3.3.11 Security Integration Modules and extensions

5.3.3.11.1 Description

Modules and libraries are available within CHIC which integrate the security framework in various programming languages (e.g. Java, PHP, .NET), web server implementations (e.g. Apache HTTP, Apache Tomcat) and applications (e.g. Liferay). These modules are described in more detail in deliverable D5.2.2.

5.3.3.12 De-Identification

5.3.3.12.1 Description

The legal requirements and constraints of the CHIC stakeholders require us to de-identify all private and patient-specific information from the clinical data that will be used in the context of CHIC. The de-identification functionality is implemented by CATS. This component provides all the necessary functionality to meet these requirements. The de-identification services are described in more detail in deliverable D4.3.1 “Development of the data protection and copyright framework for CHIC first iteration”.

5.3.3.12.2 Functionality

ID	Functionality
1	Upload data to CHIC
2	De-identify uploaded data
3	Manage and create privacy profiles (a privacy profile defines how a data file should be de-identified).

5.3.3.12.3 Programmatic/User Interface

CATS consists of a web based graphical user interface and a set of rich clients.

Programmatic REST and SOAP interfaces are also available through which files can be uploaded. Through a CATS library client-side pseudonymisation functionality can be integrated in client applications.

5.3.4 Semantic services

5.3.4.1 Ontology-based semantic services

5.3.4.1.1 Description

Ontology-based semantic services include:

1. Ontology oriented: services providing access to the knowledge base.
2. Metadata oriented: services that provide access to the metadata stored in the metadata repository.
 - 2.1. accessing the metadata only
 - 2.2. accessing the metadata through query expansion via access to the knowledge base (i.e., using services from #1 and #2.1)

5.3.4.1.2 Functionality

ID	Functionality
1	Query knowledge base
2	Add terms to the knowledge base
3	Query of the metadata repository
4	Query the metadata repository using search term expansion i.e. using #1

5.3.4.1.1 API Interface

Ontology-based semantic services are wrapping services around the Metadata Repository and around the Knowledge Base.

The base line for these solutions is adapted from the output of the VPH project RICORDO.

REST services are used to access the Metadata Repository.

REST services are used to access the Knowledge Base.

A dedicated RDF database back end is deployed and can be interacted with directly through its SPARQL endpoint. In addition to this, a convenience middleware (RDF store services) is available to perform query and write operations to the store via a set of predefined templates. This support is discussed in the RDFstore documentation reproduced here as the API is derived from these elements.

SPARQL is the query language for RDF data. In our context, a template is a SPARQL query which can comport up to ten parameters. Rdfstore reduces SPARQL to a matter of filling-in-the-blanks, namely, one blank for each parameter. Templates can be written specifically to answer specific metadata management needs. Furthermore, a given template may therefore be used while varying the values of its parameters.

Example: The SPARQL query to find all things which are "part-of" the class "acids"

```
SELECT DISTINCT ?part
      WHERE
      {
        ?part      <http://example.com/ontology#part-of>
        <http://example.com/ontology#acids>
      }
```

Now suppose you want a generic form for "find all things 'part-of' the class 'X'", where the end-user fills in X.

Example: Create a template file with a name like "get_parts_of.txt" with contents

```
SELECT DISTINCT ?part
      WHERE
      {
        ?part      <http://example.com/ontology#part-of>
        <[0]>
      }
```

Here, [0] is a variable. Other available variables are [1] through [9].

Templates should be stored in a template directory in the form of a text file. When you run Rdfstore, use the command line to tell Rdfstore which directory the templates are stored in (unless you use the default directory). The template's name (minus ".txt") will become part of Rdfstore's GUI. Assuming the template in the above example has been loaded by Rdfstore, the template can be accessed at an address like

`http://yoururl.org:20060/get_parts_of/?0=acids`

Adding template to a running Rdfstore instance is not supported and the addition of templates requires restarting Rdfstore.

At the beginning of a template file, certain special commands can be issued.

Example: You can give a name to a variable, as in the following example

```
# 0 = whole
SELECT DISTINCT ?part
WHERE
{
    ?part <http://example.com/ontology#part-of> <[0]>
}
```

In this example, the command is that first line, #0 = whole. It says that the name of the variable 0 is 'whole' (so the template is searching for 'parts' of the 'whole'). This is how the Rdfstore demo GUI knows which placeholder text to put in the different form fields.

The other type of command you can use here is a preprocessor command, as in the following example:

Example: Preprocessor command

```
#0 = whole
#Preprocessor0 =
http://localhost:20080/terms/%s?longURI=yes&json=yes
    SELECT DISTINCT ?part
    WHERE
    {
        ?part <http://example.com/ontology#part-of> <[0]>
    }
```

The command,

```
# Preprocessor0 =
http://localhost:20080/terms/%s?longURI=yes&json=yes
```

indicates that the contents of variable 0 will be passed through the indicated preprocessor. For example, if the user enters 'FMA_50801' for variable 0, Rdfstore will replace the '%s' in the Preprocessor0 string with 'FMA_50801' to get the URL:

```
http://open-physiology.org:20080/terms/FMA_50801?longURI=yes&json=yes
```

which points to OWLKB and gets a list of subclasses of FMA_50801. Rdfstore will use that list of subclasses, and query the triplestore for all things which are part-of any subclass of FMA_50801.

API type	Description	Example
RDFstore API	Rdfstore has a dynamic API. The API is defined by the templates loaded when Rdfstore is started. For each template, there	. If the template is named X.txt, and depends on parameters [0], [1], and [2], then the API command looks like: http://localhost:20060/X/?0=fill_this_in

	is a corresponding API command	&1=also_fill_this&2=this_too
Query	A low level command allows wrapping (URL encoded) SPARQL queries.	select ?x ?y ?z where {?x ?y ?z} limit 10 http://localhost.org:20060/Raw_SPARQL/?0=s elect%20%3Fx%20%3Fy%20%3Fz%20where%20{%3Fx%20%3Fy%20%3Fz}%20limit%2010
Insertion	A low level command allows inserting a triple (SPAQRL INSERT DATA):	http://localhost:20060/Insert_Triple_%28Fuseki%29/?0=a&1=b&2=c
Deletion	A low level command allows inserting a triple (SPAQRL INSERT DATA):	http://localhost.org:20060/Delete_Triple_%28Fuseki%29/?0=a&1=b&2=c

5.3.4.2 Folksonomy semantic services

5.3.4.2.1 Description

The folksonomy service provides annotation and tag management, which enables the user to add free tags to the component and the model. The generated tags will be consolidated to revise the ontology.

5.3.4.2.2 Functionality

ID	Functionality
1	Add tags to a resource identified by URI
2	Delete tags
3	Edit tags
4	Query tags by resource, user
5	Generate community tags

5.3.4.2.1 API interface/User Interface

The folksonomy service is provided as a web-based client with RESTful APIs. The web client allows the user

- to add tags to a resource with suggestions from existing tags and ontology terms (e.g. Gene Ontology, Cell Ontology), see Figure 19 for a snapshot of the initial version;
- to view tags, edit or delete them, see Figure 20 for a snapshot of the initial version;

- to view community tags, see Figure 21 for a snapshot of the initial version.

The RESTful APIs provide the following functions:

- add a tag to a resource;
- edit/delete a tag by id;
- view all available tags;
- view an individual tag;
- view tags grouped by resource;
- view tags grouped by user.

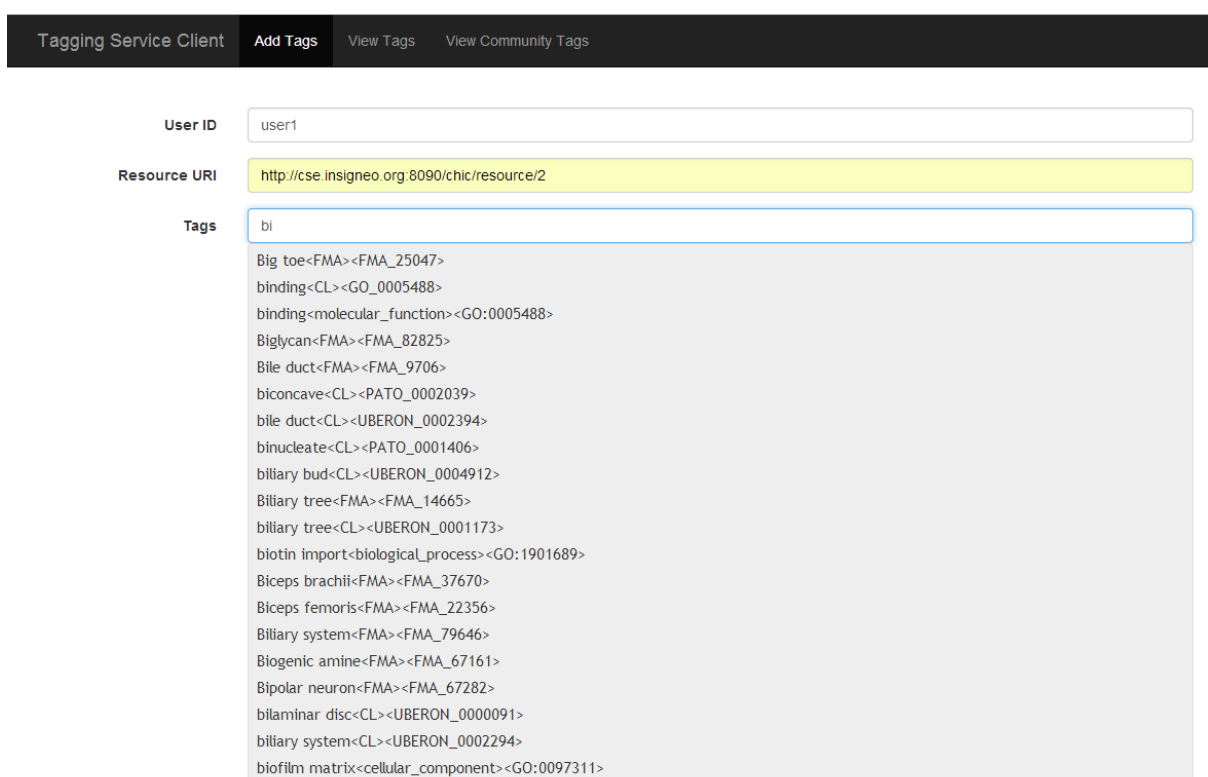


Figure 19 Folksonomy service snapshot – add a tag to a resource

Tagging Service Client
Add Tags
View Tags
View Community Tags

filter by

	Tag	Resource	Provider
<input type="checkbox"/>	ODE	http://cse.insigneo.org:8090/chic/resource/1	xia
<input type="checkbox"/>	model	http://cse.insigneo.org:8090/chic/resource/1	xia
<input type="checkbox"/>	tumor	http://cse.insigneo.org:8090/chic/resource/2	xia
<input type="checkbox"/>	protein	http://cse.insigneo.org:8090/chic/resource/2	xia
<input type="checkbox"/>	ODE	http://cse.insigneo.org:8090/chic/resource/2	xia
<input type="checkbox"/>	EDU	http://cse.insigneo.org:8090/chic/resource/2	xia
<input type="checkbox"/>	protein	http://cse.insigneo.org:8090/chic/resource/1	xia
<input type="checkbox"/>	cell cycling	http://cse.insigneo.org:8090/chic/resource/1	xia
<input type="checkbox"/>	test resource	http://cse.insigneo.org:8090/chic/resource/2	xia
<input type="checkbox"/>	Feng	http://cse.insigneo.org/chic/resource/1	Feng
<input type="checkbox"/>	models	http://cse.insigneo.org/chic/resource/1	Feng
<input type="checkbox"/>	model	http://cse.insigneo.org/chic/resource/2	04568
<input type="checkbox"/>	biology	http://cse.insigneo.org/chic/resource/2	04568
<input type="checkbox"/>	test port	http://cse.insigneo.org:8090/chic/resource/2	xia
<input type="checkbox"/>	port test	http://cse.insigneo.org:8090/chic/resource/2	xia

Figure 20 Folksonomy service snapshot – view/edit/delete tags

Tagging Service Client
Add Tags
View Tags
View Community Tags

Tag	Occurance
hypomodel	9

Figure 21 Folksonomy service snapshot – view community tags

5.3.4.3 Knowledge base

5.3.4.3.1 Description

The knowledge base is a store for ontologies. Ontologies specify the theory of a domain that is relevant to the annotation of CHIC resources.

5.3.4.3.2 Functionality

ID	Functionality
1	Storage of ontologies

2	Inferencing
---	-------------

5.3.4.3.1 API Interface

The base line for these solutions is adapted from the output of the VPH project RICORDO. In RICORDO, ontologies were maintained in the language OWL and a knowledge base was maintained in a dedicated OWL database system that facilitated OWL reasoning.

OWLKB launches a server listening for connections and responding to the following types of requests.

NOTE: The "eqterms" type of request is special. Unlike the other commands, "eqterms" will actually create a new class and add it to the selected ontology, if no equivalent class already exists. This is one of the main features of OWLKB, creation of so called *composite terms*.

In the following, the string "[SERV]" stands for the address of the OWLKB server, for example: <http://localhost:20080>, i.e. on port 20080 of localhost.

Command	Description	Example
subterms	Finds all subterms of the indicated term. For example, "amino acid" is a subterm of "acid".	[SERV]/subterms/CHEBI_33709
parents	Finds all the direct parents (i.e., the direct superclasses) of the indicated term	[SERV]/parents/CHEBI_33709
children	Finds all the direct children (i.e., the direct subclasses) of the indicated term.	[SERV]/children/CHEBI_33709
siblings	Finds all siblings of the indicated term. A 'sibling' is defined to be an immediate subterm of an immediate superterm of the indicated term.	[SERV]/siblings/CHEBI_33709
subhierarchy	Finds all subterms of the indicated term, and displays them in a hierarchical format (using JSON).	[SERV] /subhierarchy/CHEBI_33709
eqterms	Finds all terms equivalent	[SERV]/eqterms/

	<p>to the indicated term. For example, the class of all "animal cells" (CL_0000548) capable of some "reproductive process" (GO_0022414) is equivalent to the class of all "germ line stem cells" (CL_0000039). If there are no equivalent terms, a new class is created, defined to be equivalent to the indicated term. The new class is saved to the ontology (unless saving to hard-drive was disabled by command-line argument).</p>	<p>CL_0000548+and+(capable_of+some+GO_0022414)</p>
terms	<p>Finds all terms and all subterms of the indicated term. Note that unlike "eqterms", this API command will not create a new class if no equivalent classes are found.</p>	<p>[SERV] /terms/ CL_0000548+and+(capable_of+some+GO_0022414)</p>
instances	<p>Finds all instances of the indicated class. For example, "IN-VITRO-CCTYPE" might be an instance of "TYPE-OF-CLINICAL-CONTEXT". (This is, of course, only for ontologies that include named individuals; otherwise "instances" will always return the empty result set.)</p>	<p>[SERV]/instances/TYPE-OF-CLINICAL-CONTEXT</p>
labels	<p>Finds all labels annotated to the indicated term (specifically, all <code>rdfs:label</code>'s). For example, the label "Brain" is annotated to FMA_50801.</p>	<p>[SERV]20080/labels/FMA_50801</p>
search	<p>Finds all classes in the ontology with the given label (specifically, the given <code>rdfs:label</code>). Note that</p>	<p>[SERV]/search/Brain</p>

	<p>this is an exact, case-sensitive search--a search for "Brai" or "brain" will not return "Brain" for instance.</p>	
addlabel	<p>Adds a label to a class that was created with "eqterms". For syntax, see the example above. To be more precise, the label which is added is an <code><rdfs:label></code>. Multiple labels can be added for a single class. This command triggers OWLKB to save changes to the ontology to the hard drive (unless saving has been disabled via command line).</p>	<p>[SERV]/addlabel/RICORDO12345=volume+of+blood+in+aorta</p>

Furthermore, there are three ways to coerce data into JSON format:

1. Include an URL parameter 'json'.

Example: `http://localhost:20080/subterms/FMA_50801?json`

2. Include an URL parameter 'verbose'. In addition to changing the command output to json, this also causes the command to send additional information (most importantly, it will send labels along with terms).

Example: `http://localhost:20080/siblings/FMA_50801?verbose`

3. Send a request header "Accept: application/json". This has the same effect as method number 1 from above.

Example: `curl --header "Accept: application/json" "http://localhost:20080/subterms/CHEBI_33709"`

Additionally, because of backward-compatibility considerations, the default form of OWLKB results is sparse (including nothing but raw terms in most cases, whereas the user is probably interested in the labels of those terms as well). In order to get labels along with terms, use the 'verbose' URL parameter. Note that this will also coerce the results into JSON format.

Example: `http://localhost:20080/subterms/CHEBI_33709?verbose`

Finally, the strength of OWLKB is that in all the API commands where a term is expected, a compound term can be indicated using Manchester Syntax. Of course, when passing Manchester Syntax in an URL, it should be url-encoded.

Here are some examples of Manchester Syntax (we've replaced spaces with +'s so these examples can be used in URLs):

All subclasses of (GO_0000111 intersect GO_0000112): "GO_0000111+and+GO_0000112"

All things that are GO_0000111 and part-of some GO_0000112: "GO_0000111+and+part-of+some+GO_0000112"

All things that are (GO_0000111 intersect GO_0000112) and part-of some GO_0000113: "(GO_0000111+and+GO_0000112)+and+part-of+some+GO_0000113"

All things that are GO_0000111 and part-of some (GO_0000112 intersect GO_0000113): "GO_0000111+and+part-of+some+(GO_0000112+and+GO_0000113)"

5.4 Data layer

5.4.1 Model repository

5.4.1.1 Description

The Model Repository permanently hosts multiscale cancer models that have been developed in the context of the CHIC project. It also hosts tools such as linkers and data transformation tools which are needed for the construction of hypermodels. For each model, the Model Repository contains all the related information, including descriptive information (abstract and detailed description, references, etc.), input and output parameters (for proper linking with other models and tools), source files, documentation and executables of the models. Moreover, information about model authorship, ownership, and access permissions are also stored in the Model Repository database. In order for the user to be able to interact with the Repository, a web-based interface has been designed and implemented. Apart from the aforementioned graphical interface, many web services have been developed so as to be able to expose the contents of the Repository to other tools developed in the CHIC project, such as the hypermodelling Editor, the CRAF (Clinical Research Application Framework) and the Hypermodelling Framework. Up to now, more than 10 hypomodels, 4 hypermodels and 1 tool have been permanently and safely stored in the Repository in the context of CHIC project, and all this information is available to the user either through the user interface of the Model Repository¹⁵, or through the user interface of the other CHIC components, such as the Hypermodelling Editor and the CRAF. The user is now able to store in an elegant and user-friendly way new models in the Model Repository through a five-step wizard, or even browse, view, change and delete the content of the Repository.

5.4.1.2 Functionality

ID	Functionality
----	---------------

¹⁵ <https://mr.chic-vph.eu>

1	Adding a new (hyper)model or tool, including all its related information (descriptive information, input and output parameters, executable and documentation, files etc). The functionality is provided to the end user through a five-step wizard.
2	Updating any information stored in the repository. The functionality is provided to the end user through a graphical user interface and to other components through web services.
3	Deleting a (hyper)model or tool. The functionality is provided to the end user through a graphical user interface and to other components through web services.
4	Filtering stored (hyper)models according to their categorization. The aforementioned categorization constitutes a metamodeling description of the corresponding model based on the CHIC 13 perspective approach. The functionality is provided to the end user through a graphical user interface and to other components through web services.
5	Storing hypermodels constructed by the hypermodeling editor using web services.
6	Automatically storing into the CHIC RDF triplestore information related to the categorization of the models depending on the perspective from which they are viewed in the basic science context. The aforementioned information is represented in the form of subject-predicate-object expressions.
7	Publishing events to the other CHIC components (Hypermodelling Framework, CRAF, Hypermodelling Editor) through AMQP protocol, whenever the content of the Repository changes.
8	Deployment of the binaries of the models into the Hypermodelling Infrastructure in order for them to be tested by the Hypermodelling Framework.
9	Automatically “freezing” the successfully deployed models. According to the “freezing” mechanism, all successfully deployed models cannot be changed anymore.
10	Cloning the “freezed” models. Since the “freezed” models cannot be updated anymore by their owners, the Model Repository offers the service of cloning. According to the aforementioned service, the user is able to clone a “freezed” model into a new one with identical information but with different title. The clone is linked to its ancestor through a pointer and its version is incremented by one.
11	Access control to the model repository (Single sign on)

5.4.1.1 CHIC components interaction

The Model Repository exposes its content to the other CHIC components through the already developed web services. Thereafter, the Hypermodelling Framework, the CRAF (Clinical Research Application Framework) and the Hypermodelling Editor are able to retrieve, update and delete the content of the Model Repository. Table 2 briefly outlines the interaction of the Model Repository with the rest of the CHIC components and Figure 22 presents the interconnections.

Table 2 Interactions of the Model Repository with the other CHIC components

Interaction of the Model Repository with the other CHIC components	
Interaction with CRAF	CRAF retrieves information related to hypermodels (description, parameters, etc.)
	The Model Repository notifies CRAF through AMQP protocol whenever the content of the Repository changes
Interaction with the Hypermodelling Editor	The Hypermodelling Editor retrieves information related to hypomodels (description, parameters, etc.)
	The Hypermodelling Editor stores in the Model Repository new hypermodels in the form of xMML
	The Model Repository notifies the Hypermodelling Editor through AMQP protocol whenever the content of the Repository changes
Interaction with the Hypermodelling Framework	The Hypermodelling Framework retrieves the binaries and the dependencies of the stored hypomodels.
	The Hypermodelling Framework retrieves the descriptive language of the stored hypermodels in the form of xMML
	The Model Repository notifies the Hypermodelling Framework through AMQP protocol whenever the content of the Repository changes
Interaction with the CHIC RDF triplestore	The Model Repository stores into the CHIC RDF triplestore information related to the categorization of the models depending on the perspective from which they are viewed in the basic science context

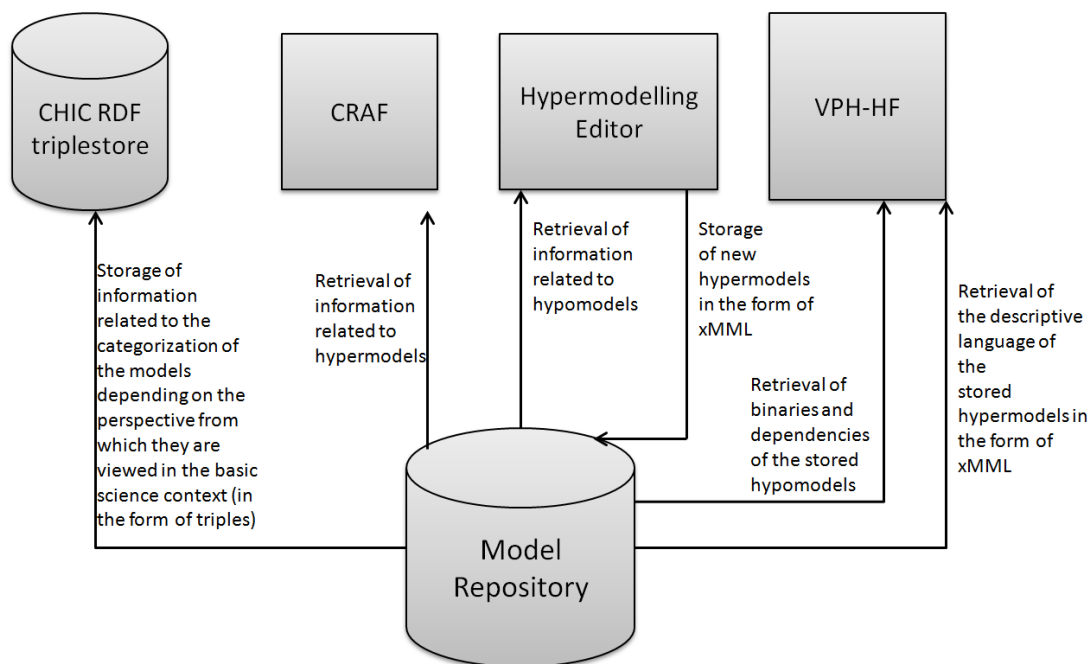


Figure 22 Interactions of the Model Repository with the other CHIC components

5.4.1.2 User interface

A web based interface has been designed and implemented in order to allow users (modellers, researchers) to interact with the Repository. The aforementioned interface makes the interaction with the user self-explanatory, efficient, enjoyable and user-friendly. It has been given special emphasis during the development of the Model Repository to provide a user interface where the user will need to provide minimal input to achieve the desired output and where the Repository will minimize undesired outputs to the user.

Figure 23 presents the main page of the Model Repository. As shown in the aforementioned figure, right after the authentication and authorization processes, the user is able to store a new model through a wizard, or browse the content of the Repository in order to view or even update the models that have been stored.

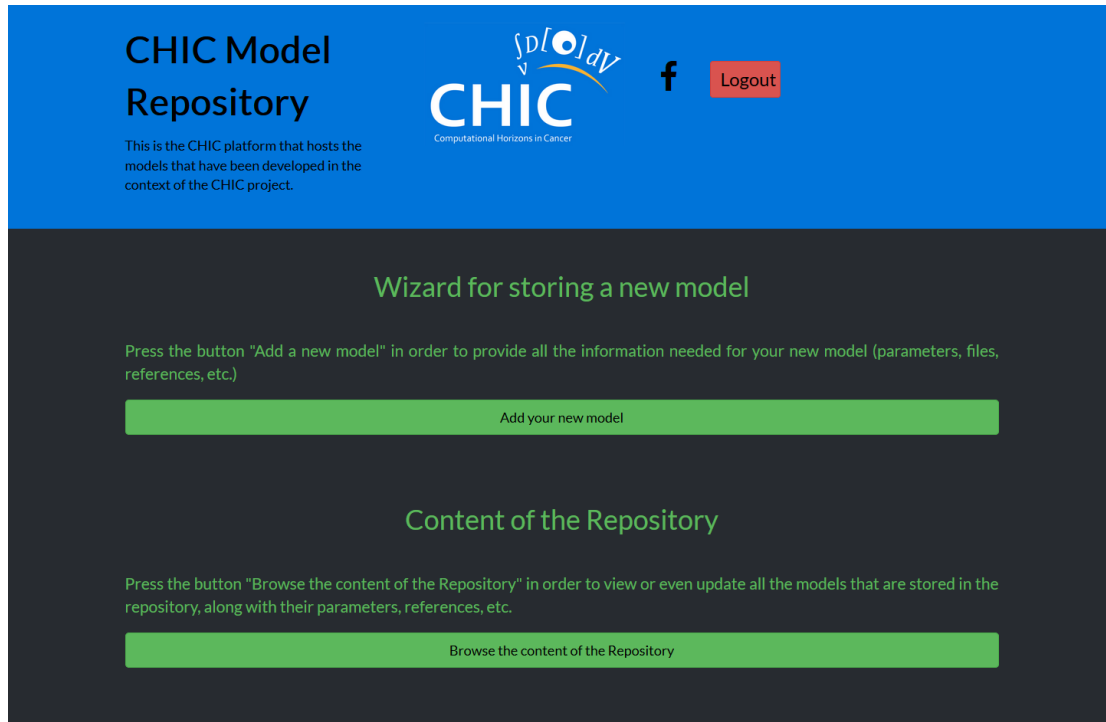


Figure 23 The main page of the Model and Tool Repository

The wizard that has been implemented for the Model Repository allows the user to store a new model through a single page. More specifically, the user is able through this wizard to store all the related information of the new model, including:

- Basic information of the new model (title, description, additional comments, etc.).
- Definition of the input and output parameters of the new model.
- Categorization of the new model based on the 13 Perspectives that have been designed within CHIC.
- References related to the new model (journal articles, conference proceedings, etc.)

Figure 24 presents the third step of the wizard where the user is able to upload a variable number of files associated with the new model (source code, executable, documentation, etc.), while Figure 25 presents the fourth step of the wizard. As shown in Figure 25, the user is able through the fourth step to categorize their new model based on the 13 perspectives that have been defined within CHIC.

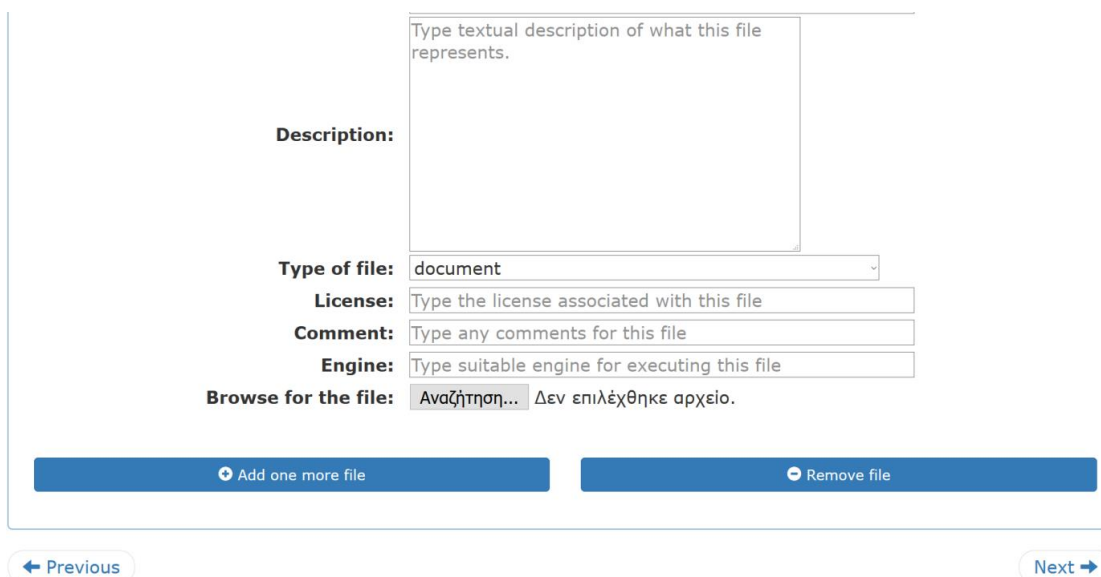


Figure 24 The third step of the wizard. The user uploads a variable number of files related to the new model (source code, executables, documentation, etc.)

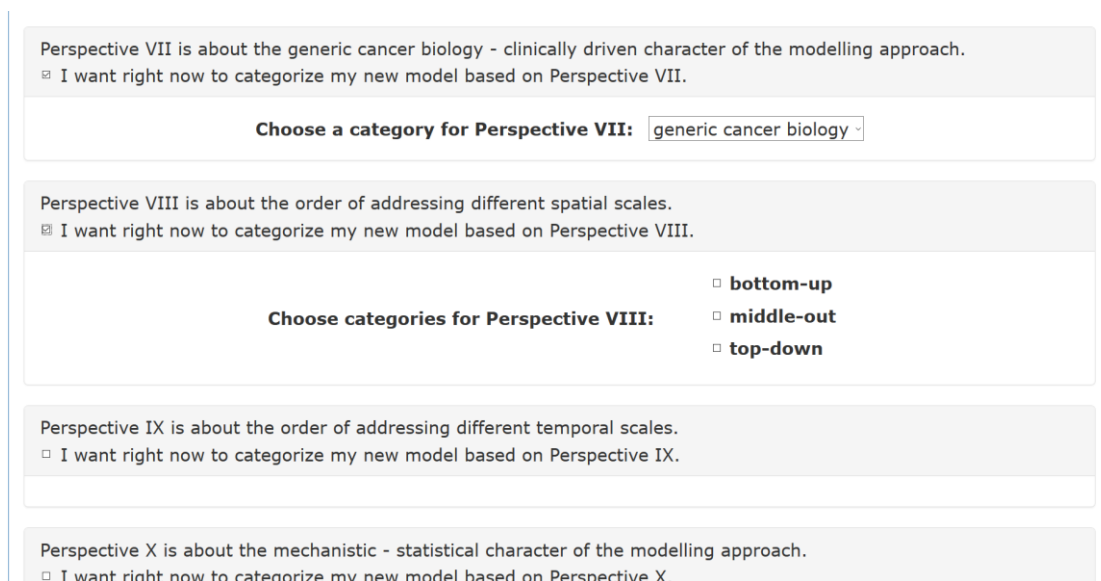


Figure 25 The fourth step of the wizard. The user categorizes their new model based on the 13 perspectives that have been defined within CHIC.

Apart from the wizard that has been developed in order to facilitate the storing procedure of a new model, the user is also able through the Model Repository to browse or even update the available models and their related information (parameters, categorization, references, files, etc.). Based on the feedback received mainly from some modelling partners (WP6), the user interface of the Model Repository has been improved and now the user is able to view the content of the Repository in a more elegant way. In addition to the advance concerning the graphics and the illustration of the corresponding web pages, the new design aims to facilitate the interactions between the Repository

and the user in a way that common tasks and activities can be accomplished easily and efficiently. For instance, if required, the user is able to view in the same page all the information related to a specific model (parameters, categorization, etc.). Moreover, since the filtering of the models based on their categorization is now feasible through the Repository, the user can easily and instantly view the models of his choice without browsing the full content of the database.

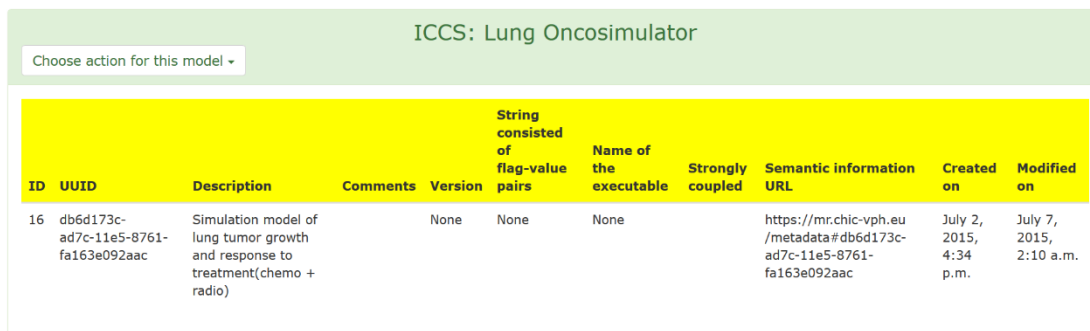
A screenshot of the content of the Model Repository is presented in Figure 26. As shown in the aforementioned figure, the available models are rendered by using tables and panels. The basic information of each model (unique identifier, name of the executable, description, etc.) is available in the corresponding table and pagination is being used in order to browse all the models. As shown in the screenshot of figure 5, the Model Repository displays the descriptive information of the models named Wilms Oncosimulator, Nephroblastoma phenomenological hypermodel and Lung Oncosimulator.

ICCS Wilms Oncosimulator

ID	UUID	Description	Comments	Version	String consisted of flag-value pairs	Name of the executable	Strongly coupled	Semantic information URL	Created on	Modified on
50	04e3c5aa-ad45-11e5-bd32-fa163e092aac	The model simulates the spatiotemporal response of wilms cancer to combined chemotherapy treatment with the regimens Vincristine and Actinomycin. Based on discrete time and space stochastic cellular automata				./wilms_oncosimulator /bin/oncosimulator	Yes	https://mr.chic-vph.eu/metadata#04e3c5aa-ad45-11e5-bd32-fa163e092aac	Dec. 28, 2015, 11:26 a.m.	Jan. 22, 2016, 11:56 p.m.

Nephroblastoma phenomenological hypermodel

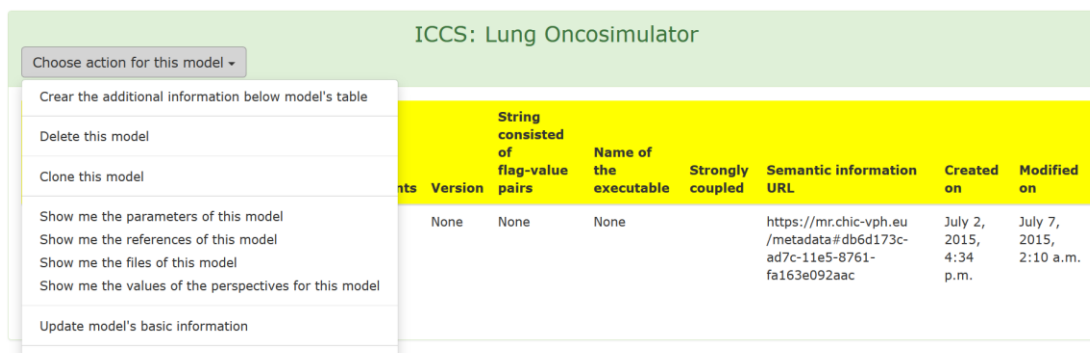
ID	UUID	Description	Comments	Version	String consisted of flag-value pairs	Name of the executable	Strongly coupled	Semantic information URL	Created on	Modified on
60	42acaa08-c052-11e5-bafa-fa163e092aac	Phenomenological hypermodel					No	https://mr.chic-vph.eu/metadata#42acaa08-c052-11e5-bafa-fa163e092aac	Jan. 21, 2016, 5:18 p.m.	Jan. 21, 2016, 6:36 p.m.



ID	UUID	Description	Comments	Version	String consisted of flag-value pairs	Name of the executable	Strongly coupled	Semantic information URL	Created on	Modified on
16	db6d173c-ad7c-11e5-8761-fa163e092aac	Simulation model of lung tumor growth and response to treatment(chemo + radio)		None	None	None		https://mr.chic-vph.eu/metadata#db6d173c-ad7c-11e5-8761-fa163e092aac	July 2, 2015, 4:34 p.m.	July 7, 2015, 2:10 a.m.

Figure 26 A screenshot of part of the content of the Model Repository

Apart from browsing the available models, the user is also able through the same page to perform some actions on the desired model. For instance, they are able to delete the model, view the parameters of the model, view the categorization of the model, update the parameters, etc. Figure 27 presents all the available actions that can be applied to a model, after pressing the button “Choose action for this model”



Actions	Version	String consisted of flag-value pairs	Name of the executable	Strongly coupled	Semantic information URL	Created on	Modified on
None	None	None	None		https://mr.chic-vph.eu/metadata#db6d173c-ad7c-11e5-8761-fa163e092aac	July 2, 2015, 4:34 p.m.	July 7, 2015, 2:10 a.m.

Figure 27 The user may apply many actions to a model (view the parameters, view the files, etc.)

For a complete guide regarding the user interface of the Model and Tool Repository, the reader may refer to the CHIC deliverable “D8.4 Report on the final system”.

5.4.1.3 API Interface

The Model and Tool Repository makes use of RESTful web services. The web services of the Model Repository are mainly based on the interfaces described in CHIC deliverable “D10.2 – Design of the orchestration platform, related components and interfaces”. The description of the web service, the HTTP method used, the parameters of the service, the URL and the returned object of the service are all described in the following tables. Each table is related to a specific RESTful web service.

5.4.1.3.1 Model/Tool

The following web services (Table 3 - Table 8) should be used whenever the client needs to store, retrieve or delete descriptive information (title, description, comments) of the model/tool.

Table 3 Information for calling storeTool web service


storeTool		
Description	This method stores the basic descriptive information of the model/tool and returns the id	
URL	https://mr.chic-vph.eu/model_app/storeTool	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
Parameters passed through request body	title=	Required - Title of the model/tool
	description=	Not required – Description of the model/tool
	comment=	Not required – Comments on the model/tool
	version=	Required – version of the model/tool (version should be in the format X.X where X is an integer)
	semtype=	Not required – url representing semantic information about this model/tool
	extra_parameters=	Not required – string consisted of flag-value pairs that should be included in the command line argument list of the model
	executable_path=	Not required – The relative path of the executable inside the compressed package
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Example Response		
The JSON object returned by method storeTool has one key, named id, and one value which is associated with this key.		

Table 4 Information for calling getAllTools web service




getAllTools		
Description	This method returns all the models/tools and the corresponding descriptive information stored (id, uuid, title, description, comment, version, semtype, executable_path, extra_parameters, strongly_coupled). It returns null when no model/tool stored in the repository.	
URL	https://mr.chic-vph.eu/model_app/getAllTools	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
Parameters	No parameters required	
Returns	200 OK & JSON object 400 http status code if bad request 401 http status code if no SAML token inside HTTP header 403 http status code if SAML token not verified 500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getAllTools are as many as the different models/tools stored in the model/tool repository. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of mr_tool entity (see figure 3) and each value of this nested object represents the information of the corresponding column</p>		

Table 5 Information for calling getToolById web service

getToolById		
Description	This method returns the descriptive information stored under the id (uuid, title, description, comment, version, semtype, strongly_coupled, executable_path, extra_parameters) and null when not existing	
URL	https://mr.chic-vph.eu/model_app/getToolById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	

Parameter (parameter should be passed through the URL – query string parameter)	id=	Required – Id of the model/tool
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method getToolById has thirteen keys named title, uuid, description, comment, version, strongly_coupled, extra_parameters, executable_path, semtype, created_on, created_by, modified_on and modified_by, and thirteen values associated with those keys.		

Table 6 Information for calling getToolByParameterId web service

getToolByParameterId		
Description	This method returns the descriptive information of the model/tool (mr_tool table) to which the given parameter belongs.	
URL	https://mr.chic-vph.eu/model_app/getToolByParameterId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
Parameter (parameter should be passed through the URL – query string parameter)	id=	Required – id of the given parameter
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		

The JSON object returned by method `getToolByParameterId` has fourteen keys named `id`, `uuid`, `title`, `description`, `comment`, `version`, `strongly_coupled`, `extra_parameters`, `executable_path`, `semtype`, `created_on`, `created_by`, `modified_on` and `modified_by`, and fourteen values associated with those keys.

Table 7 Information for calling `getToolByUuid` web service



getToolByUuid		
Description	This method returns the descriptive information stored under the <code>uuid</code> (<code>id</code> , <code>title</code> , <code>description</code> , <code>comment</code> , <code>version</code> , <code>semtype</code> , <code>strongly_coupled</code> , <code>executable_path</code> , <code>extra_parameters</code>) and null when not existing	
URL	https://mr.chic-vph.eu/model_app/getToolByUuid	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
Parameter (parameter should be passed through the URL – query string parameter)	<code>uuid=</code>	Required – <code>uuid</code> of the model/tool
Returns	200 OK & JSON object 400 http status code if bad request 401 http status code if no SAML token inside HTTP header 403 http status code if SAML token not verified 500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method <code>getToolByUuid</code> has thirteen keys named <code>id</code> , <code>title</code> , <code>description</code> , <code>comment</code> , <code>version</code> , <code>strongly_coupled</code> , <code>extra_parameters</code> , <code>executable_path</code> , <code>semtype</code> , <code>created_on</code> , <code>created_by</code> , <code>modified_on</code> and <code>modified_by</code> , and thirteen values associated with those keys.		

Table 8 Information for calling `deleteToolById` web service


deleteToolById		
Description	This method deletes the descriptive information, the files, the parameters, and property values of a model/tool.	

URL	https://mr.chic-vph.eu/model_app/deleteToolById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – id of model/tool
Returns	200 OK if model/tool has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

5.4.1.3.2 Parameter

The following web services (Table 9 - Table 15) should be used whenever the client needs to store, retrieve or delete information related to parameters (name, description, data_type, data_range, etc.).

Table 9 Information for calling storeParameter web service

storeParameter		
Description	This method stores the parameter information of a tool and returns the id	
URL	https://mr.chic-vph.eu/model_app/storeParameter	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	tool_id=	Required - id of the tool to which the parameter belongs
	name=	Required – name of the parameter
	description=	Not Required – description of the parameter
	data_type=	Required – the type of the parameter (number, string, file)
	unit=	Not Required – the units in which the parameter is

		represented (only applicable if the parameter is a number)
	data_range=	Required – Data range of the parameter <ul style="list-style-type: none"> • Discrete values example: value1,value2,value3 • Min value example: 3- • Max value example: -10 • Min max values example: 3-5
	default_value=	Required – the value that will be used if a parameter value is not provided to the tool
	is_mandatory=	Required – 1 if the parameter is mandatory, 0 if it is optional
	is_output=	Required – 1 if the parameter is output, 0 if it is input
	is_static=	Required – 1 if the parameter is static, 0 if it is dynamic
	comment=	Not Required – comments on the parameter
	semtype=	Not required – url representing semantic information about this parameter
	flag=	Not required – the flag which accompanies the parameter in the command line argument list
Returns	200 OK & JSON object *	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method storeParameter has one key, named id, and one value which is associated with this key.		

Table 10 Information for calling deleteParameter web service



deleteParameter		
Description	This method deletes a certain parameter	
URL	https://mr.chic-vph.eu/model_app/deleteParameter	
Encoding	application/x-www-form-urlencoded	
HTTP method	Delete	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – id of the parameter
Returns	200 OK if parameter has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

Table 11 Information for calling getParametersByToolId web service

getParametersByToolId		
Description	This method returns the information of all the parameters of a given tool	
URL	https://mr.chic-vph.eu/model_app/getParametersByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	tool_id=	Required – the id of the tool to which the parameters belong
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	

HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getParametersByToolId</code> are as many as the different parameters belonging to the tool. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the <code>mr_parameter</code> entity (see figure 3) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 12 Information for calling `getParameterById` web service


getParameterById		
Description	This method returns the descriptive information of the parameter stored under the given id (<code>mr_parameter</code> table).	
URL	https://mr.chic-vph.eu/model_app/getParameterById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
Parameter (parameter should be passed through the URL – query string parameter)	id=	Required – id of the given parameter
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The Json object returned by method <code>getParameterById</code> has eighteen keys named <code>tool</code>, <code>name</code>, <code>description</code>, <code>data_type</code>, <code>unit</code>, <code>flag</code>, <code>uuid</code>, <code>data_range</code>, <code>default_value</code>, <code>is_mandatory</code>, <code>is_output</code>, <code>is_static</code>, <code>comment</code>, <code>semtype</code>, <code>created_on</code>, <code>created_by</code>, <code>modified_on</code>, <code>modified_by</code> and eighteen values associated with those keys.</p>		

Table 13 Information for calling getMandatoryParametersByToolId web service




getMandatoryParametersByToolId		
Description	This method returns the information of the mandatory parameters of a given tool	
URL	https://mr.chic-vph.eu/model_app/getMandatoryParametersByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	tool_id=	Required - the id of the tool to which the mandatory parameters belong
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getMandatoryParametersByToolId are as many as the different mandatory parameters belonging to the tool. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the mr_parameter entity (see figure 3) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 14 Information for calling getInputParametersByToolId web service

getInputParametersByToolId		
Description	This method returns the information of the input parameters of a given tool	
URL	https://mr.chic-vph.eu/model_app/getInputParametersByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter	tool_id=	Required – the id of the tool to

should be passed through the URL – query string parameter		which the input parameters belong
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response The keys of the JSON object returned by method getInputParametersByToolId are as many as the different input parameters belonging to the tool. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the mr_parameter entity (see figure 3) and each value of the nested JSON object represents the information of the corresponding column.		

Table 15 Information for calling getOutputParametersByToolId web service

getOutputParametersByToolId		
Description	This method returns the information of the output parameters of a given tool	
URL	https://mr.chic-vph.eu/model_app/getOutputParametersByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	tool_id=	Required – the id of the tool to which the output parameters belong
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response The keys of the JSON object returned by method getOutputParametersByToolId are as many as the		

different output parameters belonging to the tool. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the mr_parameter entity (see figure 3) and each value of the nested JSON object represents the information of the corresponding column.

5.4.1.3.3 Property

The following web services (Table 16 - Table 22) should be used whenever the client needs to store, retrieve or delete information related to properties (perspectives) (property name, property value, property description, property comments).

Table 16 Information for calling storeProperty web service


storeProperty		
Description	This method stores the basic descriptive information of a property (perspective) and returns the id	
URL	https://mr.chic-vph.eu/model_app/storeProperty	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	name=	Required – the name of the property
	description=	Not required – description of the property
	comment=	Not required – comments on the property
	semtype=	Not required – url representing semantic information about this property
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method storeProperty has one key, named id, and one value which is associated with this key.		

Table 17 Information for calling getAllProperties web service



getAllProperties		
Description	This method returns all the properties (perspectives) and the corresponding descriptive information stored (id, name, description, comment, semtype)	
URL	https://mr.chic-vph.eu/model_app/getAllProperties	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS	No parameters required	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getAllProperties are as many as the different properties (perspectives) stored in the model/tool repository. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the mr_property entity (see figure 3) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 18 Information for calling getPropertyById web service

getPropertyById		
Description	This method returns the descriptive information stored under the property (perspective) id (name, description, comment)	
URL	https://mr.chic-vph.eu/model_app/getPropertyById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the property

Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method getPropertyById has four keys named name, description, comment, semtype, and four values associated with those keys.		

Table 19 Information for calling storePropertyValue web service


storePropertyValue		
Description	This method stores the value of a property for a tool and returns the id	
URL	https://mr.chic-vph.eu/model_app/storePropertyValue	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	tool_id=	Required – the id of the tool
	property_id=	Required – the id of the property
	value=	Required – the value of the property
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method storePropertyValue has one key, named id, and one value which is associated with this key.		

Table 20 Information for calling deletePropertyValue web service




deletePropertyValue		
Description	This method deletes the property value for a certain tool	
URL	https://mr.chic-vph.eu/model_app/deletePropertyValue	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the record which holds the property value
Returns	200 OK if property value has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

Table 21 Information for calling getPropertyValuesByToolId web service

getPropertyValuesByToolId		
Description	This method retrieves all the property (perspective) – value pairs for a given tool	
URL	https://mr.chic-vph.eu/model_app/getPropertyValuesByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	tool_id=	Required – the id of the tool with which the property – value pairs are associated
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	

HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getPropertyValuesByToolId</code> are as many as the different properties (perspectives) that describe or/and classify the given tool. Each value associated with a specific key is represented by a nested JSON object. The keys of the aforementioned nested JSON object are named <code>name</code>, <code>description</code>, <code>comment</code>, <code>value</code>, <code>semtype</code>.</p>		


Table 22 Information for calling `deletePropertyById` web service

deletePropertyById		
Description	This method deletes the property (perspective) which is associated with the given id and the corresponding values	
URL	https://mr.chic-vph.eu/model_app/deletePropertyById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the record which holds property’s descriptive information
Returns	200 OK if property has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

5.4.1.3.4 Reference

The following web services (Table 23 - Table 25) should be used whenever the client needs to store, retrieve or delete information related to references (reference title, reference authors, reference type, etc.).

Table 23 Information for calling storeReference web service

storeReference		
Description	This method stores information of the reference. The reference should be associated with a model/tool.	
URL	https://mr.chic-vph.eu/model_app/storeReference	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	tool_id=	Required – the id of the tool with which the reference is associated
	title=	Required – the title of the reference
	type=	Required – the type of the reference (book, journal article, etc.)
	creator=	Required – the creator(s) of the resource
	issued=	Required - the date of formal issuance
	bibliographic_citation=	Not required – the bibliographic citation of the resource
	is_part_of=	Not required – the related resource that this resource is part of
	source=	Not required – the related resource from which the described resource is derived from
	doi=	Not required – digital object identifier of the resource
	pmid=	Not required – the pubmed identifier
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML

	token>
Json Response	
The JSON object returned by method storeReference has one key, named id, and one value which is associated with this key.	

Table 24 Information for calling deleteReferenceById web service



deleteReferenceById		
Description	This method deletes a specific reference	
URL	https://mr.chic-vph.eu/model_app/deleteReferenceById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the reference
Returns	200 OK if reference has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

Table 25 Information for calling getReferencesByToolId web service


getReferencesByToolId		
Description	This method returns all the references of a given tool	
URL	https://mr.chic-vph.eu/model_app/getReferencesByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	tool_id=	Required – the id of the tool with which the references are associated
Returns	200 OK & JSON object	

	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getReferencesByToolId</code> are as many as the different references which are associated with the given tool. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the <code>mr_reference</code> entity (see figure 3) and each value of the nested JSON object represents the information of the corresponding column.</p>		

5.4.1.3.5 File

The following web services (Table 26 - Table 31) should be used whenever the client needs to store, retrieve or delete information related to files (title of file, description of file, the file itself, etc.).

Table 26 Information for calling storeFile web service

storeFile		
Description	This method stores the file information and returns the id	
URL	https://mr.chic-vph.eu/model_app/storeFile	
Encoding	Multipart/form-data	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	tool_id=	Required – the id of the tool with which the file is associated
	title=	Required – the title of the file
	description=	Not required – description of the file
	kind=	Not required – defines what this file is (document, source code, binary, etc.)
	license=	Not required – the license associated with this file
	Sha1sum=	Not required – the sha1 checksum of the file
	comment=	Not required – comments on

		the file
	engine=	Not required – the engine that is suitable for executing this file
	file=	Required – the actual file (blob)
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method storeFile has one key, named id, and one value which is associated with this key.		

Table 27 Information for calling deleteFile web service


deleteFile		
Description	This method deletes a certain file	
URL	https://mr.chic-vph.eu/model_app/deleteFile	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the file
Returns	200 OK if file has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

Table 28 Information for calling getFileById web service



getFileById		
Description	This method returns the given file (attachment)	
URL	https://mr.chic-vph.eu/model_app/getFileById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the file
Returns (Content-Type: application/force-download Content-Disposition: attachment)	200 OK & attachment	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

Table 29 Information for calling getPackageByToolId web service

getPackageByToolId		
Description	This method returns the file (attachment) which is of kind "compressed package with binary and dependencies" and belongs to the model with id=tool_id. This method returns 200 O.K. + attachment.	
URL	https://mr.chic-vph.eu/model_app/getPackageByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	tool_id=	Required – the id of the model/tool to which the "compressed package with binary and dependencies" belongs
Returns (Content-Type:	200 OK & attachment	
	400 http status code if bad request	

application/force-download Content-Disposition: attachment)	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

Table 30 Information for calling getFilesOfKind web service



getFilesOfKind		
Description	This method returns the information of all the files of a specific kind of a given tool	
URL	https://mr.chic-vph.eu/model_app/getFilesOfKind	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS (parameters should be passed through the URL – query string parameter)	tool_id=	Required – the id of the tool
	kind=	Required - kind of file (document, source code, binary, etc.)
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getFilesOfKind are as many as the different files of a specific kind which are associated with the given tool. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the mr_file entity (see figure 3) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 31 Information for calling getFilesByToolId web service

getFilesByToolId		
Description	This method returns information (only metadata, not attachment) for all the files that are associated with the given model/tool.	
URL	https://mr.chic-vph.eu/model_app/getFilesByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS (parameters should be passed through the URL – query string parameter)	tool_id=	Required – the id of the tool with which the files are associated
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The keys of the JSON object returned by method getFilesByToolId are as many as the different files that are associated with the given model/tool. Each value associated with a specific key is represented by a nested JSON object. The keys of the aforementioned nested JSON object are named id, title, description, kind, source, license, sha1sum, comment, engine, created_on, created_by, modified_on, modified_by and 13 values associated with those keys.		

5.4.2 Clinical data repository

5.4.2.1 Description

The clinical data repository hosts all the medical data produced or collected by the CHIC project. The data provided by the clinical environment pass through de-identification and (pseudo)-anonymization processes, as described in the following chapter. Additionally, there are appropriate interfaces that allow to import and export the contents of the clinical data repository. In this way the data can be sustained after the expiration of the project’s lifetime and reused and exploited continuously within the limits allowed by the legal framework of the project. The clinical data repository contains for each patient all the relevant medical data including imaging data, clinical data, histological data and genetic data.

5.4.2.2 Functionality

ID	Functionality
1	Provide an interface to store and retrieve pseudonymized clinical data.
2	Provide data annotation such as ontology or object type/modality.
3	Linking of datasets to find related data.
4	Semantically driven search to find datasets containing the required anatomical structures.
5	Access control to the clinical data repository (Single sign on, sharing of information)

5.4.2.1 CHIC components interaction

5.4.2.1.1 Authentication

The clinical data repository makes use of the security framework described in deliverable “D5.2.2 – Final version of security tools and guidelines”. Therefore, the users are not directly authenticated by the clinical data repository (Service Provider) itself but rather by the CHIC authentication broker (Identity Provider) to support Single Sign-On (SSO). This procedure is called brokered authentication.

The CHIC security framework further distinguishes between brokered authentication for web services including REST and for web sites. As the clinical data repository provides complete access to the features of the database with the help of REST interfaces, the Security Token Service (STS) provided by CHIC is fully integrated in the authentication process. Before calling a REST interface of the clinical data repository the client needs to send a SOAP (Simple Object Access Protocol) request containing an RST (RequestSecurityToken) to the STS. The STS then returns the identity assertion as a SAML token, embedded in a RSTR (RequestSecurityTokenResponse). The SAML token can then be passed to the REST interface through the HTTP authorization header.

The following procedure is needed in order to supply a SAML token to the clinical data repository:

1. Get the SAML token from the CHIC Security Token Service.
2. ZLIB (RFC 1950) compress the retrieved SAML token.
3. Base64 (RFC 4648) encode the compressed SAML token.
4. Supply an "Authorization" header with content "SAML auth=" followed by the encoded string.

The brokered authentication for the clinical data repository web site makes use of SAML Web Browser SSO Profile as suggested by the CHIC security framework. The SAML Web Browser SSO Profile is initiated by an end user who visits the protected clinical data repository web site, also called a Service Provider (SP). The SP redirects the user to the assertion provider (also called Identity Provider (IdP)) passing through an authentication request. The IdP will request the user to authenticate and upon successful authentication the IdP will issue an identity assertion for the user containing all information on the user needed by the SP to authenticate and authorise him. The

assertion is then sent back to the SP that will use it to determine whether the user is allowed to access the requested resource.

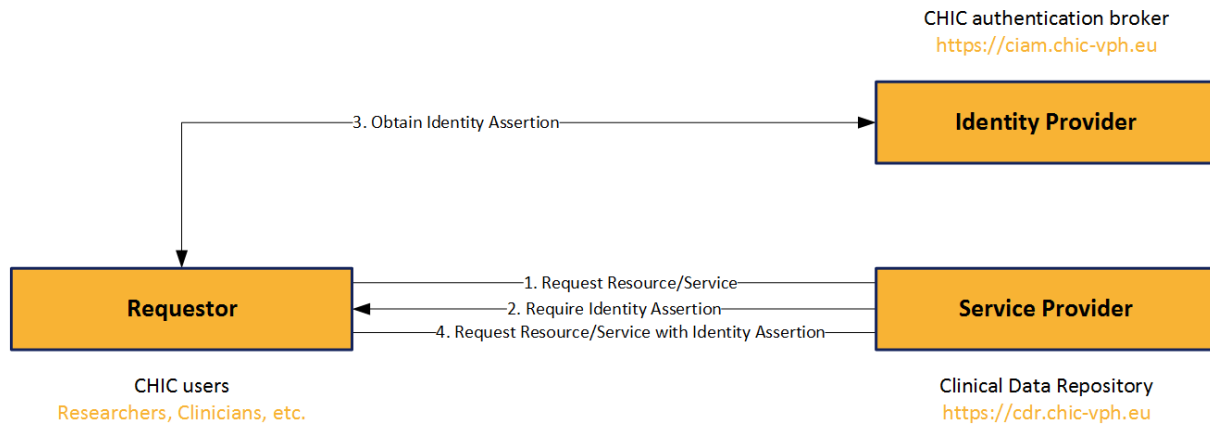


Figure 28 Brokered Authentication Flow with the clinical data repository

5.4.2.1.2 Data flow

The intention of the data flow described below (Figure 29) is to limit the additional workload on the clinical side, while providing all the relevant information with the data. The trial data collection on the clinical side is not covered. Once the data is uploaded to the CHIC infrastructure it must be impossible to know the origin of the data nor any of the patient information without being authorized to translate the used pseudonyms by the TTP. A good example for this workflow is a cryptographic hash function. A cryptographic hash function is considered practically impossible to invert if only the hash value is known. However, the same input data will result always in the same hash code. This one-way concept can be applied to the general workflow for data upload in the CHIC context.

With this concept in mind, it is straightforward to understand the problems and limitations of the data upload pipeline. A critical requirement to meet is to keep track of the patient throughout the anonymization process. Therefore, it is necessary that the same pseudonym is used for the same patient across different file formats. A unique patient identifier will ensure that the repository receiving the data is able to keep the links between datasets obtained from the same patient, even if the data are uploaded at different time points. The information is critical for the CHIC platform, since the modelling tool and other services rely on all the information collected on each individual patient. Another important aspect is the fact that datasets cannot be properly annotated once uploaded to the CHIC infrastructure, because the data uploader does not know where the dataset is actually stored. Due to the pseudonymization requirement, it is necessary to perform the annotation prior to data upload and to transfer this information together with the data file.

Therefore, the system must find a compromise between the requirements associated to the data protection, limited time available by the clinician to process the data and the information necessary to run the in-silico trials in the CHIC infrastructure. The analysis of these constraints resulted in the following proposition for the data upload workflow:

- A special trial-patient-identifier will be used across all the datasets collected on each patient.

- The datasets will be annotated before upload in a way that ensures a reliable extraction of the meta-information by the repositories.
- The semantic annotations will be stored in a triplestore (providing generalized search functionalities) only after retrieval of the meta-information by the clinical data repository.

The general workflow for data upload involves 6 distinct parties which are briefly described in Table 32 and the general workflow is illustrated in Figure 29.

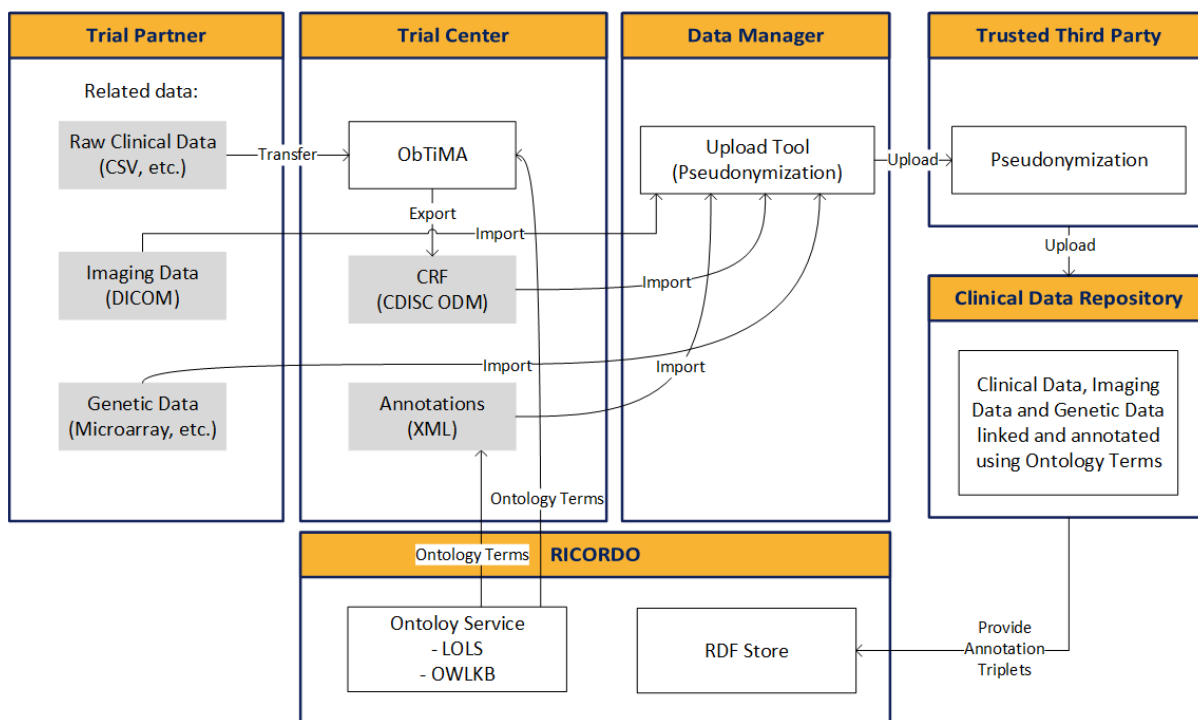


Figure 29: The general workflow for data upload

The steps required to store clinical data in the CHIC environment, including the related semantic annotation is the following:

1. The trial partner enters the patient in the trial center which generates the special trial-patient-identifier.
2. The trial partner provides the data to the trial center.
 - a. The trial partner enters the clinical study data available in raw format into the tool provided by the trial center (e.g. ObTiMA).
 - b. The trial partner provides the imaging data to the trial center.
 - c. The trial partner provides the genetic data to the trial center.
3. The trial center makes sure that the trial-patient-identifier is used accordingly.
 - a. The trial center exports the study data in standardized format (e.g. ObTiMA to CDISC ODM).
 - b. The trial center adds the trial-patient-identifier to the imaging data and creates the annotation file.
 - c. The trial center adds the trial-patient-identifier to the genetic data and creates the annotation file.
4. The trial center creates the special file containing annotations and other metadata.

5. The trial center imports the data in the upload tool.
6. The upload tool applies the first pseudonymization round.
7. The upload tool uploads the data to the trusted third party.
8. The trusted third party applies the second pseudonymization round.
9. The trusted third party uploads the data to the data repository.
10. The data repository extracts the annotations and provides them to the triplestore.

Table 32 Parties involved in the general workflow for data upload

Party	Description
Trial Partner	The trial partner conducts the clinical trial and gathers all data to be stored in the CHIC infrastructure.
Trial Center	The trial center coordinates the clinical trial and ensures that the unique trial-patient-identifier is used across all supported file formats accordingly.
Data Manager	The data manager is responsible to upload compliant data provided by the trial center to the trusted third party after a first pseudonymization round.
Trusted Third Party	The trusted third party accepts data uploaded by the data manager and uploads it to the data repository after a second pseudonymization round.
Data Repository	The data repository stores clinical, imaging and genetic data. Related data is linked and annotated with ontology terms.
RICORDO	RICORDO provides services to search ontology terms, to store annotation triples, to conduct semantically driven search queries and to perform automated semantic reasoning.

5.4.2.1.3 Auditing

Auditing is an examination of the management controls within an information technology (IT) infrastructure. The evaluation of obtained evidence determines if the information systems are safeguarding assets, maintaining data integrity, and operating effectively to achieve the desired goals or objectives. In order to track activities by individual people, systems, accounts or other entities so-called audit trails are required. An audit trail (also called audit log) is a security-relevant chronological record, set of records, and/or destination and source of records that provide documentary evidence of the sequence of activities that have affected at any time a specific operation, procedure, or event.

The clinical data repository makes use of the updated audit data model called XDASv2¹⁶ introduced in deliverable D5.2. An implementation in C# of the audit data model has been published¹⁷ as open-source on the GitHub platform under the MIT license.

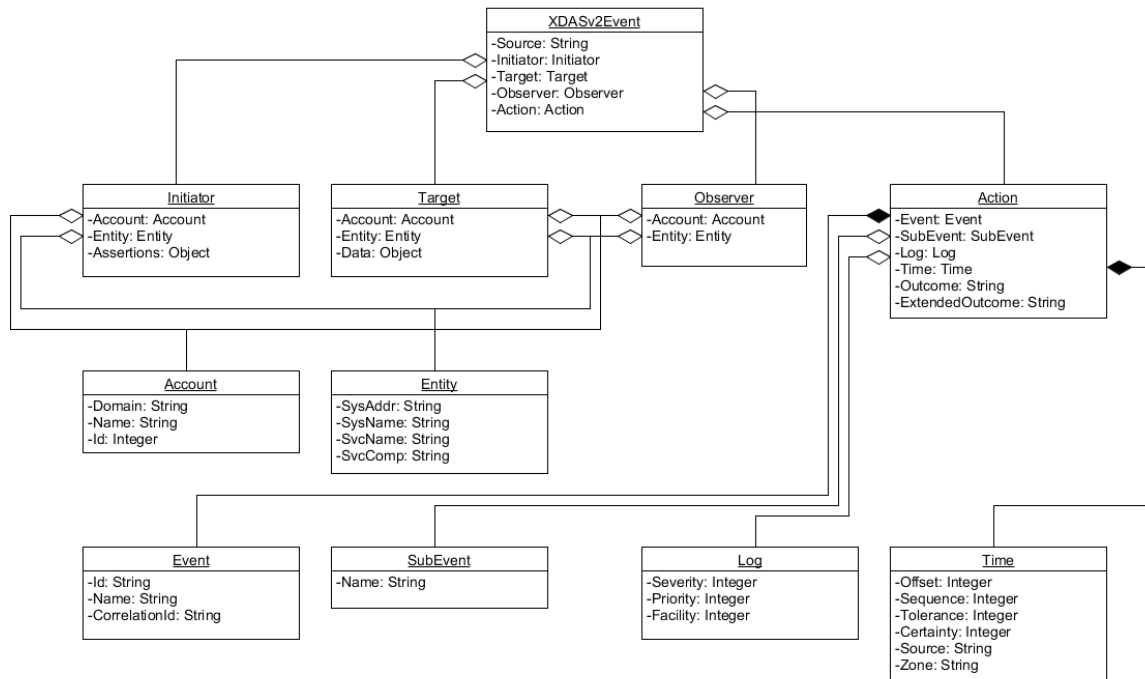


Figure 30: The audit data model XDASv2 used by the clinical data repository for auditing.

The architecture of the auditing within the clinical data repository has been designed to support different and multiple audit systems at the same time. As illustrated in Figure 31 the clinical data repository relies on the elastic stack. The elastic stack consists of Filebeat, Logstash, Elasticsearch and Kibana. All previously listed components are licensed under the Apache License Version 2.0. This ensures the legal compliance with other dependencies of the clinical data repository.

¹⁶ XDASv2, “The XDASv2 specification provides a standardized classification for audit events.”, [Online]. Available: https://www.netiq.com/documentation/edir88/pdfdoc/edirxdas_admin/edirxdas_admin.pdf. [Accessed 9 August 2016].

¹⁷ XDASv2Net, “XDASv2Net is a .NET library containing the model of the XDASv2 specification.”, [Online]. Available: <https://github.com/niklr/XDASv2Net>. [Accessed 9 August 2016].

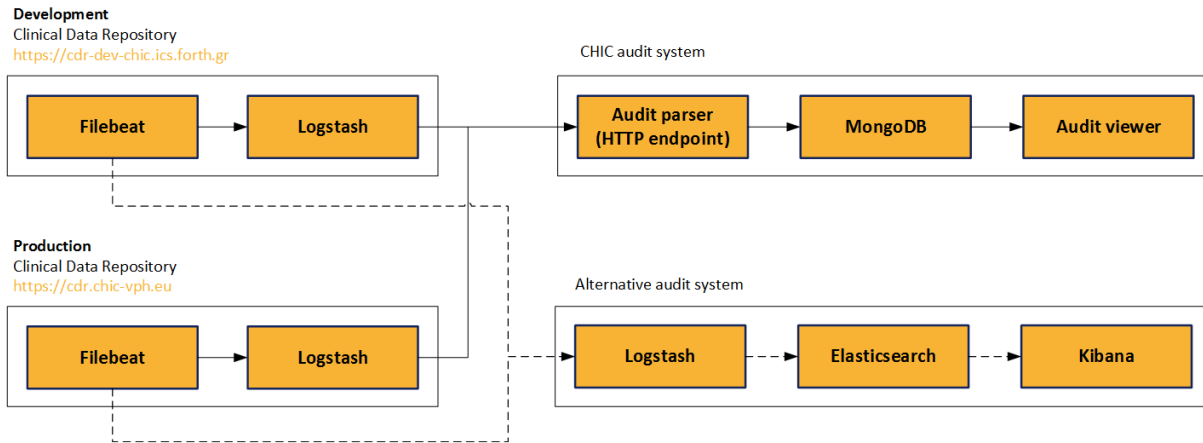


Figure 31: The components of the audit systems and the interactions with the clinical data repository

Filebeat is a lightweight, open-source shipper for log file data. As the next-generation Logstash Forwarder, Filebeat tails logs and quickly sends this information to Logstash for further parsing and enrichment or to Elasticsearch for centralized storage and analysis.

Logstash is an open-source data collection engine with real-time pipelining capabilities. Logstash can dynamically unify data from disparate sources and normalize the data into destinations of choice.

Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows to store, search, and analyze big volumes of data quickly and in near real time. It is generally used as the underlying engine/technology that powers applications that have complex search features and requirements.

Kibana is an open-source analytics and visualization platform designed to work with Elasticsearch. It can be used to search, view, and interact with data stored in Elasticsearch indices. Kibana makes it easy to perform advanced data analysis and to visualize data in a variety of charts, tables, and maps.

5.4.2.1.4 Semantic integration with RICORDO

The clinical data repository is one of the CHIC components that makes use of the higher level services, which are provided by the CHIC semantic infrastructure. It can be accessed either by website or by web service. The former is geared towards end users and the latter for third-party applications but both use the same core. The common core relies on a relational database which makes use of the Structured Query Language (SQL). Standard file formats as described in D8.1 supported by the clinical data repository include DICOM, MetaImage, Analyze and Nifti for medical imaging data, CDISC ODM XML for clinical data and MINiML XML for genetic / molecular data. One objective is to extract selected metadata from the files during the upload process to the clinical data repository automatically. Another objective is to let users such as clinicians, researchers and others annotate the objects of the clinical data repository manually. Both objectives have in common that the annotations will be exported to the semantics infrastructure provided by RICORDO within CHIC.

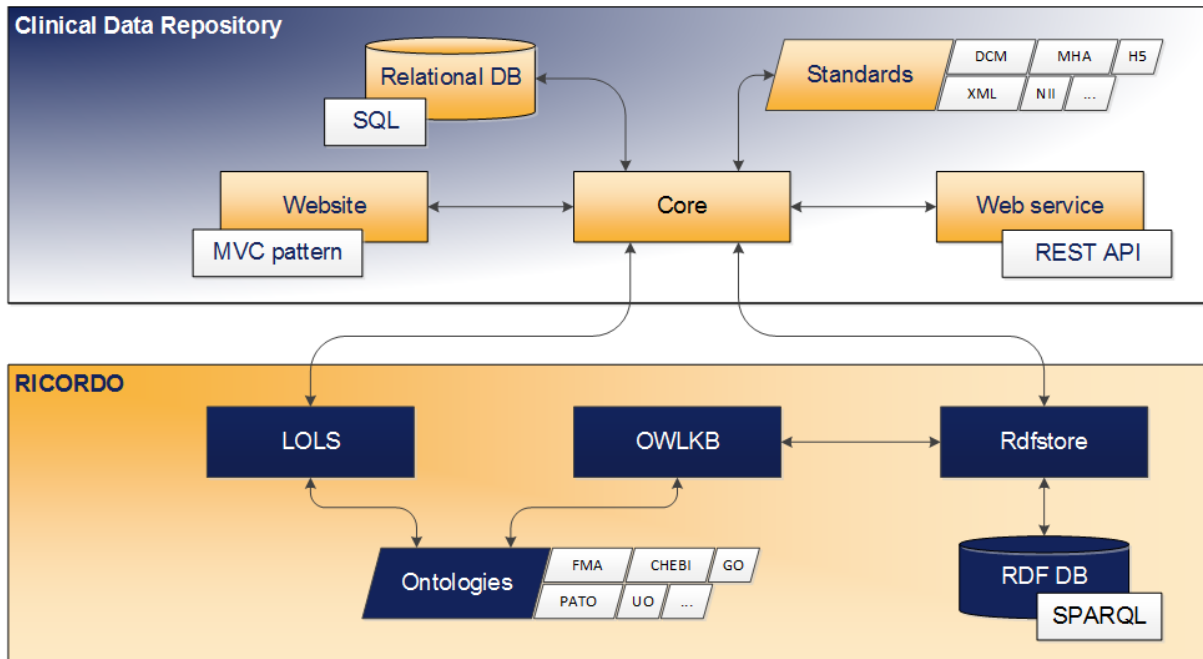


Figure 32: A visual representation of interactions between clinical data repository and RICORDO components.

RICORDO offers three components called LOLS, Rdfstore and OWLKB which are relevant within CHIC. The intended purpose of Local Ontology Lookup Service (LOLS) is to translate between standardized (but not human readable) identifier strings used for triplestores, and human-readable labels describing them for a given set of ontologies. Rdfstore is a metadata wrapper based on templates serving as a messenger between SPARQL endpoint and end-user, obviating the need to learn complicated SPARQL syntax. OWLKB is a semantic reasoner which enables to query semantic data loaded from an ontology. Both components LOLS and OWLKB have the same set of ontologies in common.

5.4.2.1.4.1 Interactions with the Local Ontology Lookup Service

A connection with LOLS is required to enable clinicians, researchers and others to annotate objects of the clinical data repository manually. An exemplary use case is the annotation of an object with anatomical regions. As shown in Figure 33, the user starts to type the name of the anatomical region and the autocomplete function offered by LOLS returns a list of matching entries. The user selects the correct entry from the list which completes this step of the annotation process. In this case, it would not make sense to present matching entries other than those from the Foundational Model of Anatomy (FMA) ontology to the user. Therefore, the crucial functionality to filter the range of ontologies to be searched by the autocomplete function is required.

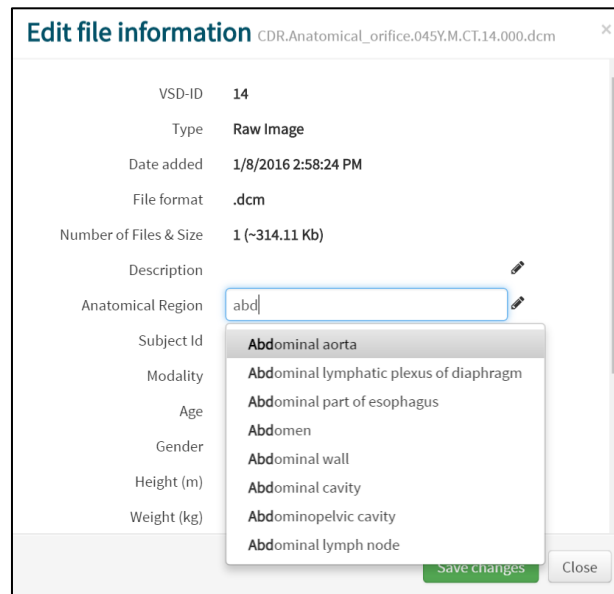


Figure 33: User dialog to annotate an object with anatomical regions using the autocomplete function offered by the Local Ontology Lookup Service (LOLS).

5.4.2.1.4.2 Interactions with the Rdfstore

Metadata can be extracted during the upload process by the clinical data repository automatically, if standard file formats are used. However, in the majority of cases the extracted metadata is not in the form to be stored directly in the Rdfstore. Therefore, the metadata must be processed to triples before being exported to the Rdfstore. This is one of the reasons the clinical data repository stores the extracted metadata in the relational database. Another reason is the export process itself which requires a reliable retry logic. Last but not least, the clinical data repository needs to be able to display the information associated with each object without fetching it from the Rdfstore every time. The Rdfstore itself already offers the functionality to add and delete triples in order to enable interactions with the clinical data repository.

Adding and deleting triples to/from the Rdfstore is merely a means to an end. The main objective is to leverage the powerful search capabilities offered by its very nature of the semantic technology. For this purpose the Rdfstore offers an extensible template system which can be used for querying. A simple query such as “get all objects having more than one file” can be achieved by the Rdfstore directly. Once the query involves information stored in an ontology such as “get all objects which are part of FMA Head” the Rdfstore relies on the semantic reasoner offered by OWLKB. A direct interaction between the clinical data repository and the OWLKB is not intended.

To enable interactions with the Rdfstore, two libraries have been developed and published as open-source on the GitHub platform under the MIT license. RdfMapperNet¹⁸ is a .NET library to map classes to RDF triples and RdfstoreNet¹⁹ is a .NET library for the Rdfstore API.

¹⁸ RdfMapperNet, “A .NET library to map classes to RDF triples.”, [Online]. Available: <https://github.com/niklr/RdfMapperNet>. [Accessed 9 August 2016].

¹⁹ RdfstoreNet, “A .NET library for the Open Physiology Rdfstore API.”, [Online]. Available: <https://github.com/niklr/RdfstoreNet>. [Accessed 9 August 2016].

5.4.2.2 User interface

The implementation of the web-based user interface offers a main view illustrated in Figure 34 which serves as entry point for almost all functionalities described throughout the user guide introduced in the deliverable “D8.2 – Prototype implementation of the CHIC repositories”. On top, an input field enables the end-user to search for datasets (1). On the left side, the folder explorer enables the user to organize data (2). MyData is the location of the user’s data; MyGroups is the default collaboration folder accessible to all group members; MyProjects are folders to organize data into personal projects; SharedFolder are folders of others which are shared to the user. In the middle of the main view, the toolbox enables the user to initiate batch commands for multiple objects or folders (3). A preview image assists the user to identify datasets (4). Several icons enable the user to display additional information about the corresponding dataset as requested (5). The file name introduced by the clinical data repository is based on a constructed template (6). The template is updated if the information is available otherwise XX is used as a placeholder.

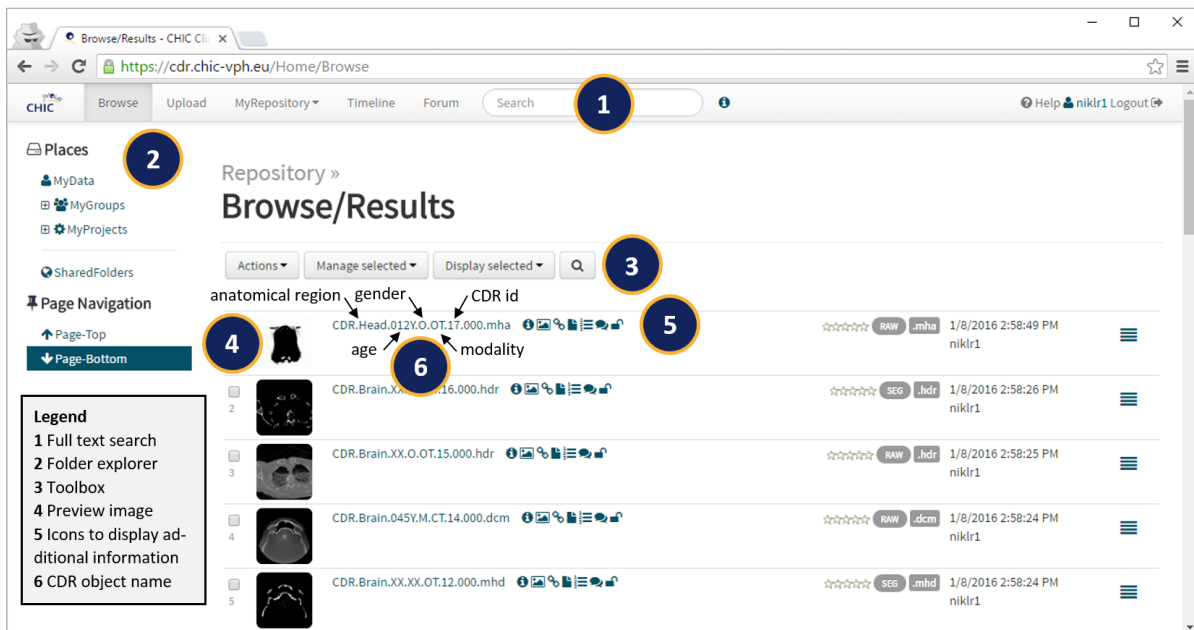


Figure 34: The web-based user interface main view of the clinical data repository

Since deliverable D8.3 a new functionality has been integrated to conduct sophisticated search queries. The query builder can be accessed by hitting the button with the magnifier symbol in the browse view (3).

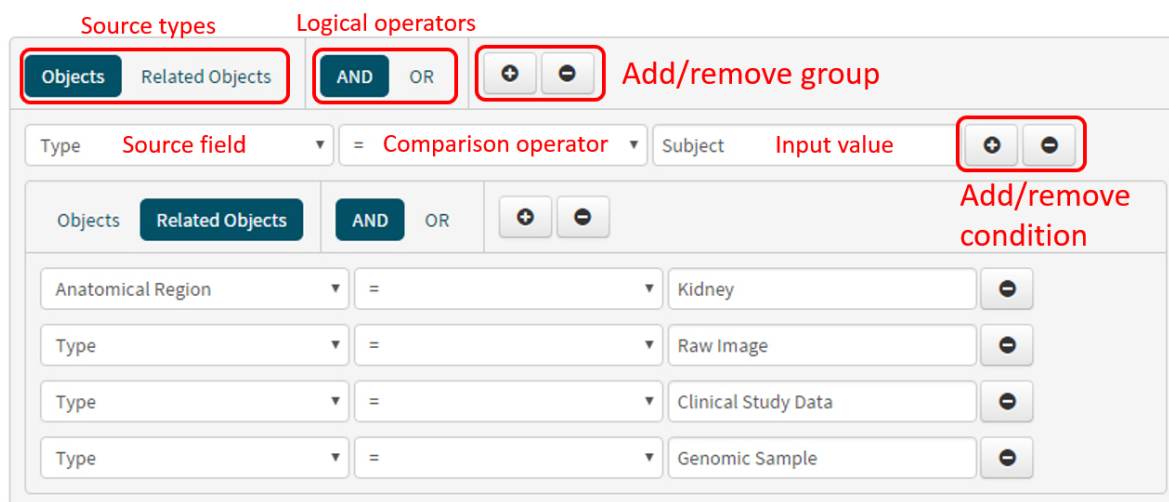


Figure 35: The dynamic search query builder integrated in the web-based user interface

The search case covered in Figure 35 has the purpose to find patients for whom we have imaging, clinical and miRNA nephroblastoma data. On the top left it is possible to select if the objects or the related objects should be searched. In this case, objects of type subject should be returned. Next to the source selection are the two supported logical operators. Next to the logical operators it is possible to add or remove groups. Inside a group multiple conditions are allowed. A condition consists of a source field, a comparison operator and input value of the user.

The query builder itself makes use of AngularJS, Bootstrap, jQuery and Underscore. It has been published²⁰ as open-source on the GitHub platform under the MIT license.

5.4.2.3 API interface

The clinical data repository makes use of the REST (Representational State Transfer) architectural principle to exchange data between applications in a loosely coupled way. Consumers of the REST API only need to know the resource address and how to make a request to that resource. How the resource actually gets its data is completely hidden from the consumer. This chapter describes the HTTP methods used, the applied pagination concept, resource addresses, accepted parameters, possible requests, responses and errors. For more information reference is made to deliverable “D8.3 – Implementation of the interfaces of the CHIC repositories”.

5.4.2.3.1 HTTP method definitions

A method refers to HTTP methods (sometimes referred to as verbs) which indicate the desired action to be performed on the identified resource. The clinical data repository interprets the received HTTP methods as follows:

²⁰ Angular-query-builder, “Dynamic query building UI written in Angular and Bootstrap.”, [Online]. Available: <https://github.com/niklr/angular-query-builder>. [Accessed 9 August 2016].

Table 33 HTTP methods supported by the clinical data repository REST API

HTTP method	Description
GET	Getting a resource. (idempotent)
POST	Creating a resource. (not idempotent)
PUT	Updating a resource. (idempotent)
DELETE	Deleting a resource. (idempotent)
OPTIONS	Getting information about the options available on the specific resource.

An idempotent HTTP method can be called many times without different outcomes.

Additionally, the REST API embraces the Open Data protocol (OData²¹). OData offers many different query options but the current implementation of the clinical data repository using ASP.NET Web API makes use of \$filter only. This query option is very powerful when it comes to filtering large result sets based on multiple conditions.

Although ASP.NET Web API²² supports JavaScript Object Notation (JSON) and Extensible Markup Language (XML) by default, the implemented and tested REST API makes use of JSON only to send and receive data. Only the UTF-8 character encoding is supported for both requests and responses.

5.4.2.3.2 Pagination

Pagination is the process of dividing a document into discrete pages in order to keep the loading time at a predictable level. Requests with large result sets may timeout or be truncated, therefore most resources returning a large result set are paginated by default.

Table 34 The pagination concept applied to large result sets returned by the clinical data repository.

Parameter name	Value type	Default value	Description
rpp	int	25	Defines the amount of included results per page. Allowed values: 10, 25, 50, 100, 250, 500
page	int	0	Defines the current page index. Allowed values: 0, 1, 2, ...
Example Request			
GET https://cdr.chic-vph.eu/api/objects?rpp=25&page=3			
Example Response			
{			

²¹ OData, “An open protocol to allow the creation and consumption of queryable and interoperable RESTful APIs in a simple and standard way.”, [Online]. Available: <http://www.odata.org>. [Accessed 9 August 2016].

²² ASP.NET Web API, “ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices.”, [Online]. Available: <http://www.asp.net/web-api>. [Accessed 9 August 2016].


```

"totalCount": 99,
"pagination": {
  "rpp": 25,
  "page": 3
},
"items": [
  ...
],
"nextPageUrl": "https://cdr.chic-vph.eu/api/objects?rpp=25&page=4"
}

```

5.4.2.3.3 Include

Include is a special parameter supported by several resources. It enables the caller to define which properties should be included in the response. This will reduce the amount of calls needed to get all information. Includable properties are marked under additional information of the resource response description. It is possible to include multiple properties at the same time by delimiting the property names by a comma.

Table 35 The includable attribute demonstrated on the basis of the groups resource implemented by the clinical data repository

Name	Description	Type	Additional information
Id	The identifier of the group	integer	None.
Name	The name of the group.	string	Filterable
Chief	The chief of the group.	BaseViewModel	Includable
SelfUrl	The URL to the resource.	string	None.
Example Request without include			
GET https://cdr.chic-vph.eu/api/groups/1			
Example Response without include			
<pre> { "id": 1, "name": "Test group", "chief": { "selfUrl": "https://cdr.chic-vph.eu/api/users/2" }, "selfUrl": "https://cdr.chic-vph.eu/api/groups/1" } </pre>			
Example Request with include			
GET https://cdr.chic-vph.eu/api/groups/1?include=chief			
Example Response with include			
<pre> { "id": 1, "name": "Test group", "chief": { "id": 2, "username": "niklr1", "selfUrl": "https://cdr.chic-vph.eu/api/users/2" } } </pre>			

```

},
"selfUrl": "https://cdr.chic-vph.eu/api/groups/1"
}

```

5.4.2.3.4 Requests, Responses, and Errors

A successful completion of a request returns one of three possible states:

Table 36 The possible return states used by the clinical data repository to indicate a successful completion of a request.

HTTP status code	Description
200 OK	The default state. On GET requests, the response contains all the requested objects. On PUT and POST requests, the requested updates have been done correctly on the persistence layer.
201 Created	Returned on successful POST requests when one or more new objects have been created. The response contains information on the newly created objects, e.g. identification values.
204 No Content	Returned on successful DELETE requests.

An unsuccessful completion of a request returns one of six possible states:

HTTP status code	Description
400 Bad Request	The format of the URL and/or of values in the parameter list is not valid. Or the URL indicates a non-existing action.
401 Unauthorized	Either the request does not contain required authentication information or the authenticated user is not authorized to get a requested object or to do the request updated operation.
404 Not Found	The URL is correct, but the requested object does not (or no longer) exist.
405 Method Not Allowed	Different action methods may be restricted to one or more of the HTTP methods (GET, PUT, or POST). The received request uses one that is not allowed with the action method specified in the URL. In this case, other parts of the URL are not validated.
500 Internal Server Error	When a method causes an exception that has no adequate handling in the method itself. Developers of client systems are kindly requested to report these response states to the developing team and to transmit information about the respective request and the response objects.
501 Not Implemented	May occur during development. The requested action has been specified and documented, but not yet implemented.

Table 37: The possible return states used by the clinical data repository to indicate an unsuccessful completion of a request.

5.4.2.3.5 Resource description template

In order to describe the input and output of the API endpoint resources the following template is used.




HTTP Method	Resource name	Requires Authentication?  Yes /  No
Description	A short text describing the resource.	
Content-Type	The Content-Type entity-header field indicates the media type of the entity-body sent to the recipient or, in the case of the HEAD method, the media type that would have been sent had the request been a GET.	
Parameters	A list of all parameters accepted by the resource.	
Example Request		
An example request which can be sent to the resource.		
Example Response		
An example response returned by the resource.		

Table 38: The template used to describe the API endpoint resources of the clinical data repository

5.4.2.3.6 Dynamic search


The sophisticated search queries introduced in the web-based user interface chapter are supported by the API as well. For this purpose the API has been extended with two new endpoints listed consecutively.

OPTIONS	dynamic_search	
Description	Returns the available options for this resource.	
Example Request		
OPTIONS https://cdr.chic-vph.eu/api/dynamic_search HTTP/1.1		
Example Response		
<pre>{ "logicalOperators": [{ "name": "And", "displayName": "AND", "position": 1 }, { "name": "Or", "displayName": "OR", "position": 2 }], "sourceTypes": [{ "name": "Objects", "displayName": "Objects", "position": 1, "sourceFields": [...] }, { "name": "RelatedObjects", "displayName": "Related Objects", </pre>		

```

    "position": 2,
    "sourceFields": [
      ...
    ]
  }
]
}

```

POST	dynamic_search?include={include}	
Description	Returns the result of the sophisticated search query.	
Content-Type	application/json	
Parameters	include (string) Allowed properties to be included: <ul style="list-style-type: none"> • See GET objects (D8.3) 	

Example Request

```

{
  "sourceType": {
    "name": "Objects"
  },
  "logicalOperator": {
    "name": "And"
  },
  "conditions": [
    {
      "sourceField": {
        "name": "ObjectType",
        "displayName": "Type"
      },
      "comparisonOperator": {
        "name": "Equals",
        "displayName": "="
      },
      "inputItem": {
        "data": "Subject",
        "displayName": "Subject",
        "isTypeahead": true
      }
    }
  ],
  "groups": [
    {
      "sourceType": {
        "name": "RelatedObjects"
      },
      "logicalOperator": {
        "name": "And"
      },
      "conditions": [
        {
          "sourceField": {
            "name": "AnatomicalRegion",
            "displayName": "Anatomical Region"
          },
          "comparisonOperator": {

```



```
{
  "totalCount": 7,
  "pagination": {
    "rpp": 25,
    "page": 0
  },
  "items": [
    ...
  ],
  "nextPageUrl": null
}
```

5.4.3 In Silico trial repository

5.4.3.1 Description

Since biological simulations require many computational resources, especially when the simulations involve multiscale imaging data, the *In Silico* Trial Repository is a critical component. The *In Silico* Trial Repository has been designed and developed in order to be able to persistently store all the simulation scenarios and the *in silico* predictions. The input data (the original state of the patient), the simulation scenario (the *in silico* treatment) and the output data (the state of the patient after the *in silico* treatment) are store persistently after the completion of the simulation scenario. The aforementioned data are readily available for evaluation, comparison, and validation without the need for executing the same simulation again. More specifically, the *In Silico* Trial Repository contains for each *in silico* trial all the related information including:

- model input (processed medical data that can be used as input to the specific model or hypermodel used in the simulation).
- model or hypermodel (not the actual model/hypermodel code used in the simulation but information about it).
- model output

The content of the *In Silico* Trial Repository is available to the users (researchers, modellers, clinicians) through the user interface that has been developed (<https://istr.chic-vph.eu>), and to the other CHIC components through the corresponding web services. Consequently, the user is now able either through the user interface of the Repository, or through other CHIC components, to easily store and retrieve all the data concerning a complete *in silico* trial (i.e. a set of simulation runs) that they or someone else has run.

5.4.3.2 Functionality

ID	Functionality
1	Adding a new instance of an in-silico trial. The instance can be partially created (input data and/or hypermodel) or can be fully created (input data, hypermodel and results). The functionality is provided to the end user through a graphical user interface and to other components through web services.

2	Updating an instance of an in-silico trial. The functionality is provided to the end user through a graphical user interface and to other components through web services.
3	Deleting any information stored in the repository (<i>in silico</i> trials, <i>in silico</i> experiments, results, etc.) The functionality is provided to the end user through a graphical user interface and to the other CHIC components through web services.
4	Filter the instances of an in-silico trial according to specific criteria. For example the instances related to a specific hypermodel or to a specific patient may be presented. The functionality is provided to the end user through a graphical user interface.
5	Access control to the in-silico trials repository (Single sign on)

5.4.3.1 CHIC components interaction

The *In Silico* Trial Repository exposes its content to the other CHIC components through the already developed web services which are presented in chapter 2.5. Thereafter, the Hypermodelling Framework, the CRAF (Clinical Research Application Framework) and the Hypermodelling Editor are able to retrieve, update and delete the content of the *In Silico* Trial Repository. Table 39 briefly outlines the interaction of the *In Silico* Trial Repository with the rest of the CHIC components and Figure 36 presents the interconnections.

Table 39 Interactions of the In Silico Trial Repository with the other CHIC components

Interaction of the <i>In Silico</i> Trial Repository with the other CHIC components	
Interaction with CRAF	CRAF creates in the <i>In Silico</i> Trial Repository new <i>in silico</i> trials and new <i>in silico</i> experiments
	CRAF retrieves from the <i>In Silico</i> Trial Repository information related to the completed <i>in silico</i> experiments and their results
Interaction with the Hypermodelling Editor	The Hypermodelling Editor creates in the <i>In Silico</i> Trial Repository new <i>in silico</i> trials and new <i>in silico</i> experiments
Interaction with the Hypermodelling Framework	The Hypermodelling Framework stores in the <i>In Silico</i> Trial Repository the results of the simulation (output files of the hypermodel, PDF report, etc.)
	The Hypermodelling Framework notifies the <i>In Silico</i> Trial Repository about the status of the pending <i>in silico</i> experiment (FINISHED SUCCESSFULLY, FINISHED ERRONEOUSLY, etc.)

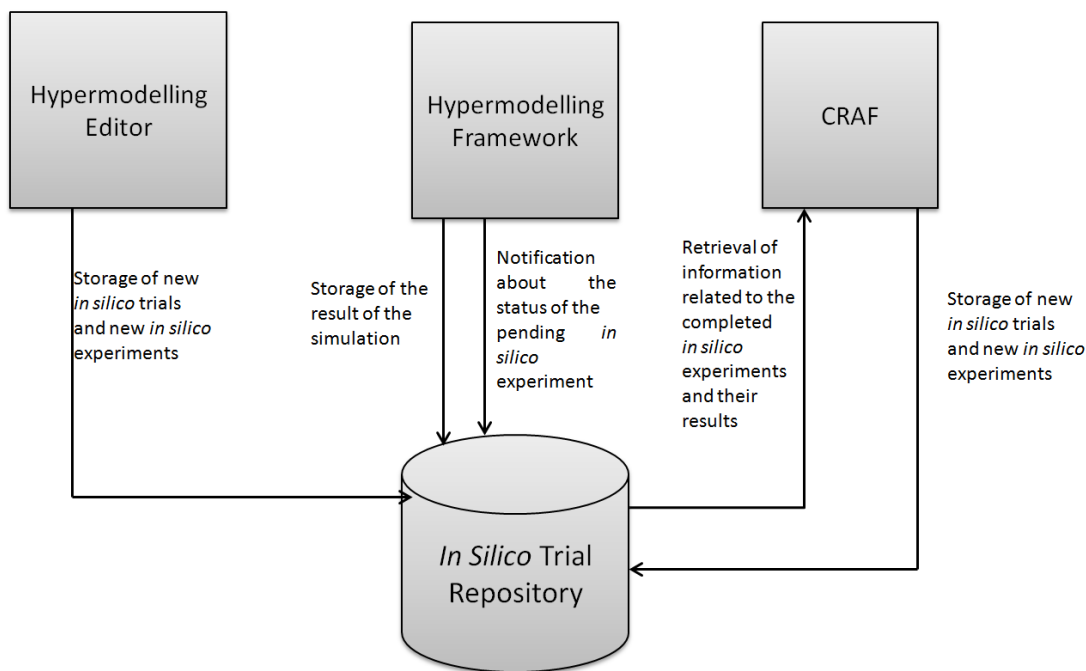


Figure 36 Interactions of the In Silico Trial Repository with the other CHIC components

5.4.3.2 User Interface

A web-based user interface has been designed and implemented in order to allow users to interact with the *In Silico* Trial Repository. The user has the ability to easily store and retrieve all the data concerning a complete *in silico* trial (i.e. a set of simulation runs) that they or someone else has run. The user interface of the *In Silico* Trial Repository has been designed with the intention of improving the experience of the user when interacting with the Repository. Special emphasis has been given during the development of the Repository to provide a user interface where the user will need to provide minimal input for inspecting and evaluating the results of the *in silico* experiments.

After the authentication of the user, the user is redirected to the main page of the *In Silico* Trial Repository which is depicted in Figure 37. As shown in this figure, the user is able to store a new *in silico* experiment through a wizard, or browse the content of the Repository in order to view or even update the available simulations and their status.



Figure 37 The main page of the In Silico Trial Repository

The wizard that has been created for the *In Silico* Trial Repository allows the user to store the simulation scenarios and the *in silico* predictions. Although the persistent storage of the input and output simulation data can be performed by the other CHIC components (CRAF, Hypermodelling framework) through the corresponding web services of the *In Silico* Trial Repository, this wizard provides an alternative way for saving the results through the user interface of the Repository. More specifically, the user is able through this wizard to store all the related information of the new *in silico* experiment, including:

- Description of the *in silico* trial
- Input and output files of the new *in silico* experiment
- Description of the *in silico* experiment
- References related to the new experiment and the corresponding *in silico* trial
- Description related to the initial state of the patient and the final simulated state of the patient

Figure 38 presents the first step of the wizard where the user is able to provide information related to the *in silico* trial to which the new *in silico* experiment belongs, while Figure 39 presents the fifth step of the wizard. As shown in Figure 39, the user is able through the fifth step to upload one or more output files related to the simulation.

Wizard for storing a new experiment

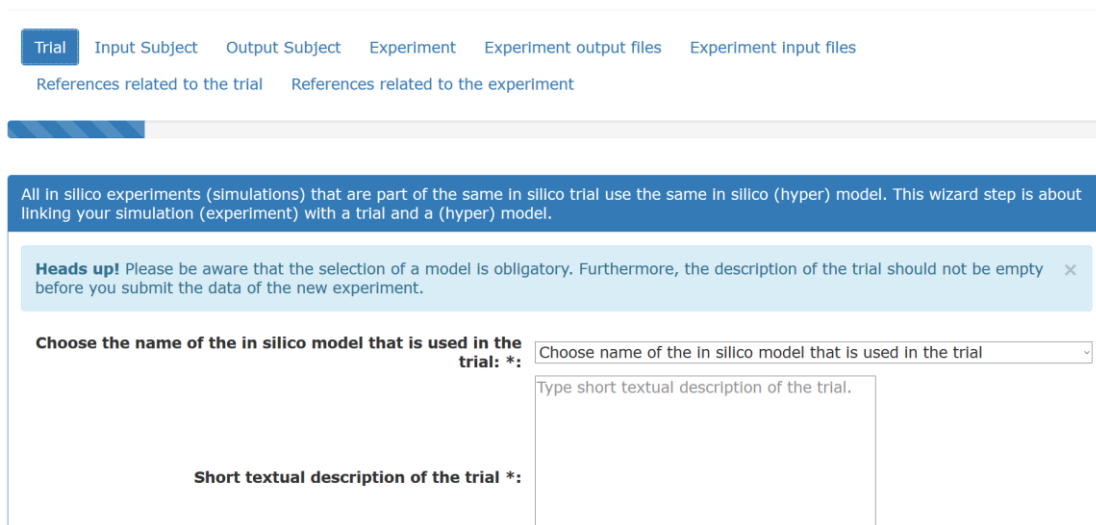


Figure 38 The first step of the wizard. The user provides information related to the *in silico* trial to which the new *in silico* experiment belongs

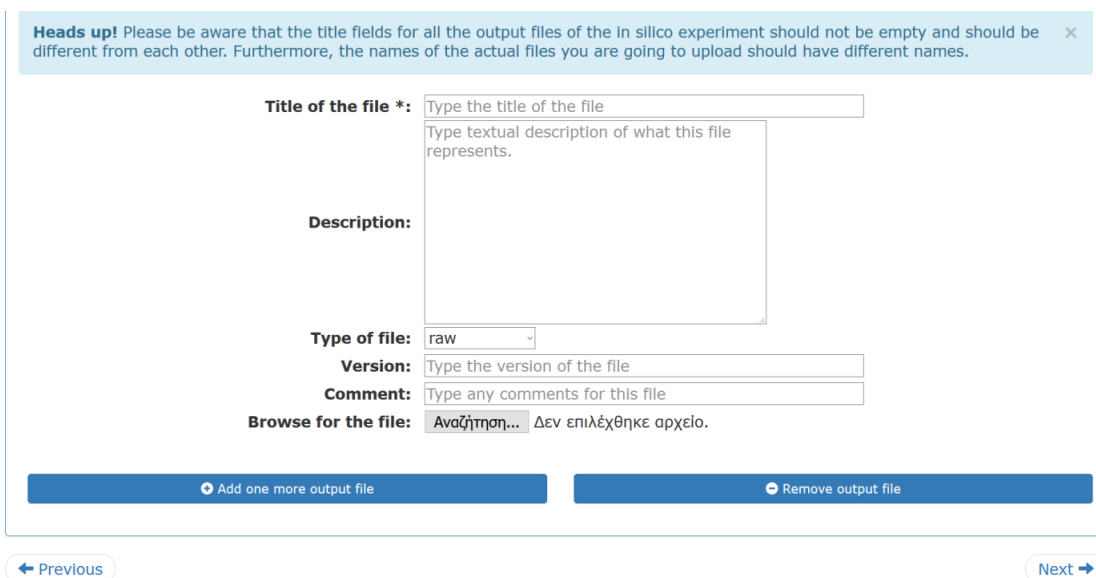


Figure 39 The fifth step of the wizard. The user uploads one or more output files related to the simulation

Apart from the wizard for storing a new *in silico* experiment, the user is able through the user interface of the *In Silico* Trial Repository to browse all the available simulations. Based on the design of the Repository, the basic principles of the *in silico* trial database are the subject, the *in silico* trial, and the *in silico* experiment. All the *in silico* experiments are organized in *in silico* trials and all the *in silico* experiments that are part of the same *in silico* trial use the same (hyper)model. Consequently, the (hyper)model that is being used for a specific experiment is defined in the *in silico* trial entity. This means that no more than one trial can be assigned to a single (hyper)model.

Based on this design, the first step for browsing the content of the *In Silico* Trial Repository is for the user to examine the available trials, and therefore, to inspect the different (hyper)models for which

there are available finished simulations. Figure 40 presents a screenshot of a part of the page which is related with the presentation of the available *in silico* trials. As shown in the aforementioned figure, the description, the ID and the date of the creation of the *in silico* trial, as well as the name of the corresponding model, are all available in the same page.

Trials stored in the Repository

Trial related to model Nephroblastoma phenomenological hypermodel							
Choose action for this trial ▾							
Trial ID	Trial Description	Model ID used in trial	Model name used in trial	The url where the in silico model is located	Comments related to the trial	Trial was created on	Trial was modified on
28	Trial for Nephroblastoma	60	Nephroblastoma phenomenological hypermodel	None	None	Jan. 21, 2016, 9:59 p.m.	None

Trial related to model FORTH: Preprocessing tool							
Choose action for this trial ▾							
Trial ID	Trial Description	Model ID used in trial	Model name used in trial	The url where the in silico model is located	Comments related to the trial	Trial was created on	Trial was modified on
29	Trial for Nephroblastoma	45	FORTH: Preprocessing tool	None	None	Jan. 26, 2016, 3:39 p.m.	None

Trial related to model Nephroblastoma MUSCLE multimodeller hypermodel							
Choose action for this trial ▾							
Trial ID	Trial Description	Model ID used in trial	Model name used in trial	The url where the in silico model is located	Comments related to the trial	Trial was created on	Trial was modified on
30	Trial for Nephroblastoma	61	Nephroblastoma MUSCLE multimodeller hypermodel	None	None	Jan. 28, 2016, 1:51 a.m.	None

Trial related to model Lung MUSCLE multimodeller hypermodel							
Choose action for this trial ▾							
Trial ID	Trial Description	Model ID used in trial	Model name used in trial	The url where the in silico model is located	Comments related to the trial	Trial was created on	Trial was modified on
31	Trial for Lung Cancer	89	Lung MUSCLE multimodeller hypermodel	None	None	May 25, 2016, 7:52 p.m.	None

Trial related to model Nephroblastoma MUSCLE multimodeller hypermodel

Choose action for this trial ▾

Trial ID	Trial Description	Model ID used in trial	Model name used in trial	The url where the in silico model is located	Comments related to the trial	Trial was created on	Trial was modified on
30	Trial for Nephroblastoma	61	Nephroblastoma MUSCLE multimodeller hypermodel	None	None	Jan. 28, 2016, 1:51 a.m.	None

Trial related to model Lung MUSCLE multimodeller hypermodel

Choose action for this trial ▾

Trial ID	Trial Description	Model ID used in trial	Model name used in trial	The url where the in silico model is located	Comments related to the trial	Trial was created on	Trial was modified on
31	Trial for Lung Cancer	89	Lung MUSCLE multimodeller hypermodel	None	None	May 25, 2016, 7:52 p.m.	None

Figure 40 Part of the page of the *In Silico* Trial Repository which indicates the available in silico trials

After choosing the *in silico* trial of their interest, the user may view the content of all the simulations that belong to the aforementioned trial, such as the description, the status and the unique identifier of the experiment. For instance, as shown in Figure 41, the user is able to view information related to the last four executions of Nephroblastoma multimodeller hypermodel. Moreover, the pseudonymized identification of the patient used in the experiment and the input and output files of the simulation can all be provided by the user interface of the *In Silico* Trial Repository.



Experiments that belong to trial with ID 25. The aforementioned trial is related to Nephroblastoma multimodeller hypermodel

You can filter the available experiments based on the id of the patient:

- I want to filter the experiments.

This in silico experiment has been conducted with Nephroblastoma multimodeller hypermodel

Choose action for this experiment ▾

Experiment ID	Experiment UUID	Experiment Description	ID of the trial to which this experiment belongs	ID of input subject entity	ID of output subject entity	In the in silico experiment the "placebo model" must be used (yes/no)	Status of the in silico experiment	Comments	Experiment was created on	Experiment was modified on
661	502ba118-690b-11e6-888d-fa163e099cf4	Nephroblastoma	25	1297	1298		FINISHED SUCCESSFULLY	None	Aug. 23, 2016, 11:26 a.m.	Aug. 23, 2016, 11:41 a.m.



Figure 41 Information related to the last four simulations (in silico experiments) of Nephroblastoma multimodeller hypermodel

As shown in Figure 41, the user is also able to apply many actions with respect to a specific experiment. For instance they can download the input or the output files of the experiment, update the experiment’s information view the pseudonymized patient identification, etc.

For a complete guide regarding the user interface of the *In Silico* Trial Repository, the reader may refer to the deliverable “D8.4 Report on the final system”.

5.4.3.3 API Interface

The *In Silico* Trial Repository makes use of RESTful web services. The aforementioned web services are based on the interfaces described in deliverable “D10.2 – Design of the orchestration platform, related components and interfaces”. This chapter aims at presenting all the necessary information which is essential in order for the client to access the Repository’s content. The description of the web service, the HTTP method used, the parameters of the service, the URL and the returned object of the service are all described in the following tables. Each table is related to a specific RESTful web service.

5.4.3.3.1 Trial

The following web services (Table 40 - Table 45) should be used whenever the client needs to store, retrieve or delete information related to trials (description of trial, model used in the trial, comments on the trial, etc.).

Table 40 Information for calling storeTrial web service


storeTrial		
Description	This method stores the basic descriptive information of the trial, the model, the placebo model, etc. It returns the id of the trial	
URL	https://istr.chic-vph.eu/trial_app/storeTrial	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	description=	Required – the description of the trial
	model_id=	Required – the id of the in silico model that is used in the trial
	model_url=	Required – the url where the in silico model is located
	placebo_model_id=	Not required – the id of the in silico model that is used as a placebo
	placebo_model_url=	Not required – the url where the placebo in silico model is located
	comment=	Not required – comments on the trial
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method storeTrial has one key, named id, and one value which is associated with this key.		

Table 41 Information for calling getAllTrials web service



getAllTrials		
Description	This method returns the corresponding descriptive information of all the trials stored in <i>in silico</i> trial repository (trial ids, description of the trial, comments, etc.).	
URL	https://istr.chic-vph.eu/trial_app/getAllTrials	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS	No parameters required	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getAllTrials</code> are as many as the different trials stored in the <i>in silico</i> trial repository. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the <code>tr_trial</code> entity (see figure 39) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 42 Information for calling getUserTrials web service

getUserTrials		
Description	This method returns information for all the trials that have been created by the user with which the saml token is associated.	
URL	https://istr.chic-vph.eu/trial_app/getUserTrials	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER	Only the SAML token is required.	
Returns	200 OK & JSON object	
	400 http status code if bad request	

	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getUserTrials</code> are as many as the different trials that have been created by/for that user. Each value associated with a specific key is represented by a nested JSON object. The keys of the aforementioned nested JSON object are named <code>id</code>, <code>description</code>, <code>model_id</code>, <code>model_url</code>, <code>placebo_model_id</code>, <code>comment</code>, <code>created_on</code>, <code>created_by</code>, <code>modified_on</code>, <code>modified_by</code>.</p>		

Table 43 Information for calling `getTrialById` web service


getTrialById		
Description	This method returns the descriptive information (description of the trial, comments, etc.), of the given trial.	
URL	https://istr.chic-vph.eu/trial_app/getTrialById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the trial
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The JSON object returned by method <code>getTrialById</code> has eleven keys named <code>id</code>, <code>description</code>, <code>model_id</code>, <code>model_url</code>, <code>placebo_model_id</code>, <code>placebo_model_url</code>, <code>comment</code>, <code>created_on</code>, <code>created_by</code>, <code>modified_on</code> and <code>modified_by</code>, and eleven values associated with those keys.</p>		

Table 44 Information for calling getTrialByModelId web service



getTrialByModelId		
Description	This method returns the information related to the trial in which the given model is used (trial id, description of the trial, comments, etc.). The argument is the id of the tool used in the model repository	
URL	https://istr.chic-vph.eu/trial_app/getTrialByModelId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the model which is used in the trial
Returns	200 OK & JSON object 400 http status code if bad request 401 http status code if no SAML token inside HTTP header 403 http status code if SAML token not verified 500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method getTrialByModelId has eleven keys named id, description, model_id, model_url, placebo_model_id, placebo_model_url, comment, created_on, created_by, modified_on and modified_by, and eleven values associated with those keys.		

Table 45 Information for calling deleteTrialById web service


deleteTrialById		
Description	This method deletes the trial, the experiments included in the trial, the reference links and everything else that is associated with this trial	
URL	https://istr.chic-vph.eu/trial_app/deleteTrialById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL –	id=	Required – the id of the trial

query string parameter)		
Returns	200 OK if trial has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

5.4.3.3.2 Experiment


The following web services (Table 46 - Table 55) should be used whenever the client needs to store, retrieve or delete information related to experiments (description of experiment, link to the trial to which this experiment belongs, comments on the experiment, etc.).

Table 46 Information for calling storeExperiment web service

storeExperiment		
Description	This method stores the necessary and descriptive information of an experiment. It returns the id of the stored experiment	
URL	https://istr.chic-vph.eu/trial_app/storeExperiment	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	trial_id=	Required – the id of the trial with which the new experiment is associated
	description=	Required – the description of the new experiment
	subject_id_in=	Required – the id of the subject that is used as an input to the new <i>in silico</i> experiment
	subject_id_out=	Required – the id of the subject that is produced after the execution of the new <i>in silico</i> experiment

	placebo=	Required – true if in the in silico experiment the placebo model must be used, otherwise false
	status=	Not required – the status of the in silico experiment (NOT STARTED, ON PROGRESS, FINISHED SUCCESSFULLY, FINISHED ERRONEOUSLY)
	comment=	Not required – Comments related to the experiment
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method storeExperiment has one key, named id, and one value which is associated with this key.		

Table 47 Information for calling getUserExperiments web service

getUserExperiments		
Description	This method returns information for all the experiments that have been created by the user with which the saml token is associated	
URL	https://istr.chic-vph.eu/trial_app/getUserExperiments	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER	Only the SAML token is required.	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML

	token>
Json Response	
<p>The keys of the JSON object returned by method <code>getUserExperiments</code> are as many as the different experiments that have been created by/for that user. Each value associated with a specific key is represented by a nested JSON object. The keys of the aforementioned nested JSON object are named <code>id</code>, <code>trial</code>, <code>description</code>, <code>subject_id_in</code>, <code>subject_id_out</code>, <code>placebo</code>, <code>status</code>, <code>comment</code>, <code>uuid</code>, <code>created_on</code>, <code>created_by</code>, <code>modified_on</code>, <code>modified_by</code>. The value which corresponds to the key <code>trial</code> is another json object with keys <code>id</code> and <code>model_id</code>. The value which corresponds to the key <code>subject_id_out</code> is another json object with keys <code>id</code>, <code>subject_external_id</code> and <code>description</code>. The value which corresponds to the key <code>subject_id_in</code> is another json object with keys <code>id</code>, <code>subject_external_id</code> and <code>description</code>.</p>	

Table 48 Information for calling `getUserPendingExperiments` web service


getUserPendingExperiments		
Description	This method returns information for all the experiments with status "either "NOT STARTED" or "ON PROGRESS" that belong to the user associated with the SAML token.	
URL	https://istr.chic-vph.eu/trial_app/getUserPendingExperiments	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER	Only the SAML token is required.	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getUserPendingExperiments</code> are as many as the different "NOT STARTED" or "ON PROGRESS" experiments that have been created by/for that user . Each value associated with a specific key is represented by a nested JSON object. The keys of the aforementioned nested JSON object are named <code>id</code>, <code>trial</code>, <code>description</code>, <code>subject_id_in</code>, <code>subject_id_out</code>, <code>placebo</code>, <code>status</code>, <code>comment</code>, <code>uuid</code>, <code>created_on</code>, <code>created_by</code>, <code>modified_on</code>, <code>modified_by</code>. The value which corresponds to the key <code>trial</code> is another json object with keys <code>id</code> and <code>model_id</code>. The value which corresponds to the key <code>subject_id_out</code> is another json object with keys <code>id</code>, <code>subject_external_id</code> and <code>description</code>. The value which corresponds to the key <code>subject_id_in</code> is another json object with keys <code>id</code>, <code>subject_external_id</code> and <code>description</code>.</p>		

Table 49 Information for calling getAllExperimentsByTrialId web service



getAllExperimentsByTrialId		
Description	This method returns information of all the experiments which belong to a given trial	
URL	https://istr.chic-vph.eu/trial_app/getAllExperimentsByTrialId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	trial_id=	Required – the id of the trial
Returns	200 OK & JSON object 400 http status code if bad request 401 http status code if no SAML token inside HTTP header 403 http status code if SAML token not verified 500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getAllExperimentsByTrialId are as many as the different experiments which belong to the given trial. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the tr_experiment entity (see figure 39) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 50 Information for calling getExperimentById web service

getExperimentById		
Description	This method returns the experiment and the related information stored under the id (description, subject_id_in, subject_id_out, placebo, status, comment, etc.)	
URL	https://istr.chic-vph.eu/trial_app/getExperimentById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL –	id=	Required – the id of the experiment

query string parameter)		
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method <code>getExperimentById</code> has thirteen keys named <code>id</code> , <code>uuid</code> , <code>trial_id</code> , <code>description</code> , <code>subject_id_in</code> , <code>subject_id_out</code> , <code>placebo</code> , <code>status</code> , <code>comment</code> , <code>created_on</code> , <code>created_by</code> , <code>modified_on</code> and <code>modified_by</code> , and twelve values associated with those keys.		

Table 51 Information for calling `getExperimentByUuid` web service


getExperimentByUuid		
Description	This method returns the experiment and the related information stored under the <code>uuid</code> (<code>description</code> , <code>subject_id_in</code> , <code>subject_id_out</code> , <code>placebo</code> , <code>status</code> , <code>comment</code> , etc.)	
URL	https://istr.chic-vph.eu/trial_app/getExperimentByUuid	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	<code>uuid=</code>	Required – the <code>uuid</code> of the experiment
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method <code>getExperimentByUuid</code> has thirteen keys named <code>id</code> , <code>uuid</code> , <code>trial_id</code> , <code>description</code> , <code>subject_id_in</code> , <code>subject_id_out</code> , <code>placebo</code> , <code>status</code> , <code>comment</code> , <code>created_on</code> , <code>created_by</code> , <code>modified_on</code> and <code>modified_by</code> , and thirteen values associated with those keys.		

Table 52 Information for calling getExperimentStatusById web service



getExperimentStatusById		
Description	This method returns the status of the experiment	
URL	https://istr.chic-vph.eu/trial_app/getExperimentStatusById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the experiment
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method getExperimentStatusById has one key named status, and one value associated with this key.		

Table 53 Information for calling getExperimentsByStatus web service

getExperimentsByStatus		
Description	This method returns all the experiments that are on a given status	
URL	https://istr.chic-vph.eu/trial_app/getExperimentsByStatus	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	status=	Required – the status of the in silico experiment (NOT STARTED, ON PROGRESS, FINISHED SUCCESSFULLY, FINISHED ERRONEOUSLY)
Returns	200 OK & JSON object	

	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getExperimentsByStatus</code> are as many as the different experiments that are on a given status. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the <code>tr_experiment</code> entity (see figure 39) and each value of the nested JSON object represents the information of the column.</p>		

Table 54 Information for calling `updateExperimentStatus` web service



updateExperimentStatus		
Description	This method updates the status of a given experiment	
URL	https://istr.chic-vph.eu/trial_app/updateExperimentStatus	
Encoding	application/x-www-form-urlencoded	
HTTP Method	PUT	
PARAMETERS (parameters passed through request body)	id=	Required – the id of the experiment
	status=	Required - the status of the in silico experiment (NOT STARTED, ON PROGRESS, FINISHED SUCCESSFULLY, FINISHED ERRONEOUSLY)
Returns	200 OK if the status of the experiment has been updated	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>


Table 55 Information for calling deleteExperimentById web service

deleteExperimentById		
Description	This method deletes the experiment and the corresponding experiment references (links)	
URL	https://istr.chic-vph.eu/trial_app/deleteExperimentById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the experiment
Returns	200 OK if experiment has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

5.4.3.3.3 Miscellaneous parameter


The following web services (Table 56 - Table 61) should be used whenever the client needs to store, retrieve or delete information related to miscellaneous parameters (value assigned to miscellaneous parameter, link to the experiment with which the miscellaneous parameter is associated, etc.).

Table 56 Information for calling storeMiscellaneousParameter web service

storeMiscellaneousParameter		
Description	This method stores information related to a miscellaneous parameter. It returns the id of the created record.	
URL	https://istr.chic-vph.eu/trial_app/storeMiscellaneousParameter	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	experiment_id=	Required – the id of the experiment with which the


		miscellaneous parameter is associated
	hypomodel_parameter_id=	Required – the id of hypomodel’s parameter stored in model/tool repository (mr_parameter entity) with which the miscellaneous parameter is associated
	hypermodel_parameter_id=	Not required – the id of hypermodel’s parameter stored in model/tool repository (mr_parameter entity) with which the miscellaneous parameter is associated
	value=	Required – the value that has been assigned to miscellaneous parameter for a given experiment
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method storeMiscellaneousParameter has one key, named id, and one value which is associated with this key.		

Table 57 Information for calling getAllMiscellaneousParameters web service

getAllMiscellaneousParameters		
Description	This method returns information of all miscellaneous parameters	
URL	https://istr.chic-vph.eu/trial_app/getAllMiscellaneousParameters	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS	No parameters required	
Returns	200 OK & JSON object	

	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getAllMiscellaneousParameters</code> are as many as the different miscellaneous parameters that are stored in the <i>in silico</i> trial repository. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the <code>tr_miscellaneous_parameter</code> entity (see figure 39) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 58 Information for calling `getUserMiscellaneousParameters` web service

getUserMiscellaneousParameters		
Description	This method returns information for all the miscellaneous parameters that have been created by the user with which the saml token is associated.	
URL	https://istr.chic-vph.eu/trial_app/getUserMiscellaneousParameters	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER	Only the SAML token is required.	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getUserMiscellaneousParameters</code> are as many as the different miscellaneous parameters that have been created by/for that user. Each value associated with a specific key is represented by a nested JSON object. The keys of the aforementioned nested JSON object are named <code>id</code>, <code>experiment_id</code>, <code>hypomodel_parameter_id</code></p>		

,hypermodel_parameter_id , value, created_on, created_by, modified_on, modified_by.

Table 59 Information for calling getAllMiscellaneousParametersByExperimentId web service




getAllMiscellaneousParametersByExperimentId		
Description	This method returns information of all miscellaneous parameters which are associated with a given experiment	
URL	https://istr.chic-vph.eu/trial_app/getAllMiscellaneousParametersByExperimentId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	experiment_id=	Required – the id of the experiment
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getAllMiscellaneousParametersByExperimentId are as many as the different miscellaneous parameters which are associated with the given experiment. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the tr_miscellaneous_parameter entity (see figure 39) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 60 Information for calling getMiscellaneousParameterById web service

getMiscellaneousParameterById		
Description	This method returns information of the miscellaneous parameter stored under the id (experiment_id, hypomodel_parameter_id, hypermodel_parameter_id, value, etc)	

URL	https://istr.chic-vph.eu/trial_app/getMiscellaneousParameterById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the miscellaneous parameter
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method getMiscellaneousParameterById has eight keys named experiment_id, hypomodel_parameter_id, hypermodel_parameter_id, value, created_on, created_by, modified_on and modified_by, and eight values associated with those keys.		

Table 61 Information for calling deleteMiscellaneousParameterById web service

deleteMiscellaneousParameterById		
Description	This method deletes the miscellaneous parameter	
URL	https://istr.chic-vph.eu/trial_app/deleteMiscellaneousParameterById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the miscellaneous parameter
Returns	200 OK if miscellaneous parameter has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64

		encoded compressed SAML token>
--	--	--------------------------------

5.4.3.3.4 Subject

The following web services (Table 62 - Table 66) should be used whenever the client needs to store, retrieve or delete information related to the subject (description of the subject, comments on the subject, etc.).

Table 62 Information for calling storeSubject web service


storeSubject		
Description	This method stores information related to a subject. The method returns the id of the created subject	
URL	https://istr.chic-vph.eu/trial_app/storeSubject	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	description=	Required – the description of the state of the subject
	subject_external_id=	Not required – the external id of the subject
	external_url=	Not required – the url of the external repository
	comment=	Not required – comments on the subject
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method storeSubject has one key named id and one value associated with this key.		

Table 63 Information for calling deleteSubjectById web service



deleteSubjectById		
Description	This method deletes a subject (and the linked files) stored under the provided subject_id	
URL	https://istr.chic-vph.eu/trial_app/deleteSubjectById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the subject
Returns	200 OK if subject has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

Table 64 Information for calling getAllSubjects web service

getAllSubjects		
Description	This method returns all the subjects that are stored in the Repository	
URL	https://istr.chic-vph.eu/trial_app/getAllSubjects	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS	No parameters required	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64

		encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getAllSubjects</code> are as many as the different subjects that are stored in the <i>in silico</i> trial repository. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the <code>tr_subject</code> entity (see figure 39) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 65 Information for calling `getUserSubjects` web service


getUserSubjects		
Description	This method returns information for all the subjects that have been created by the user with which the saml token is associated.	
URL	https://istr.chic-vph.eu/trial_app/getUserSubjects	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER	Only the SAML token is required.	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getUserSubjects</code> are as many as the different subjects that have been created by/for that user. Each value associated with a specific key is represented by a nested JSON object. The keys of the aforementioned nested JSON object are named <code>id</code>, <code>description</code>, <code>subject_external_id</code>, <code>external_url</code>, <code>comment</code>, <code>created_on</code>, <code>created_by</code>, <code>modified_on</code>, <code>modified_by</code>.</p>		

Table 66 Information for calling `getSubjectById` web service


getSubjectById		
----------------	--	---

Description	This method returns the subject and the related information stored under the id (description, subject_external_id, external_url, comments, etc.)	
URL	https://istr.chic-vph.eu/trial_app/getSubjectById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the subject
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method getSubjectById has nine keys named id, description, subject_external_id, external_url, comment, created_on, created_by, modified_on and modified_by, and nine values associated with those keys.		

5.4.3.3.5 Reference


The following web services (Table 67 - Table 73) should be used whenever the client needs to store, retrieve or delete information related to experiment’s/trial’s references (title of reference, reference authors, link to the experiment/trial with which this reference is associated, etc.).

Table 67 Information for calling storeTrReference web service

storeTrReference		
Description	This method stores the information of a reference and returns the id	
URL	https://istr.chic-vph.eu/trial_app/storeTrReference	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	title=	Required – the title of the reference
	type=	Not required – the type of the

		reference (book, journal article, etc.)
	creator=	Not required – the creator(s) of the resource
	issued=	Not required – the date of formal issuance
	bibliographic_citation=	Not required – bibliographic citation of the resource
	is_part_of=	Not required – the related resource that this resource is part of
	source=	Not required – the related resource from which the described resource is derived from
	doi=	Not required – digital object identifier of the resource
	pmid=	Not required – pubmed identifier
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method storeTrReference has one key named id, and one value associated with this key.		

Table 68 Information for calling getAllTrReferences web service

getAllTrReferences		
Description	This method returns all the references and the related information	
URL	https://istr.chic-vph.eu/trial_app/getAllTrReferences	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS	No parameters required	

Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getAllTrReferences</code> are as many as the different references that are stored in the <i>in silico</i> trial repository. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the <code>tr_reference</code> entity (see figure 39) and each value of the nested JSON object represents the corresponding information of the column.</p>		

Table 69 Information for calling `getTrReferencesByTrialId` web service


getTrReferencesByTrialId		
Description	This method returns the related information of all references which are associated with the given trial.	
URL	https://istr.chic-vph.eu/trial_app/getTrReferencesByTrialId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	trial_id=	Required – the id of the trial
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getTrReferencesByTrialId</code> are as many as the different references that are associated with the given trial. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the <code>tr_reference</code> entity (see figure 39) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 70 Information for calling getTrReferencesByExperimentId web service



getTrReferencesByExperimentId		
Description	This method returns the related information of all the references which are associated with the given experiment.	
URL	https://istr.chic-vph.eu/trial_app/getTrReferencesByExperimentId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	experiment_id=	Required – the id of the experiment
Returns	200 OK & JSON object 400 http status code if bad request 401 http status code if no SAML token inside HTTP header 403 http status code if SAML token not verified 500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getTrReferencesByExperimentId are as many as the different references that are associated with the given experiment. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the tr_reference entity (see figure 39) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 71 Information for calling deleteTrReferenceById web service

deleteTrReferenceById		
Description	This method deletes a reference and the corresponding links to trials or experiments	
URL	https://istr.chic-vph.eu/trial_app/deleteTrReferenceById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the reference
Returns	200 OK if reference (along with the links) has been deleted	

	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

Table 72 Information for calling storeLinkToReference web service



storeLinkToReference		
Description	This method creates a link from a trial or an experiment to a reference. Returns the id of the link	
URL	https://istr.chic-vph.eu/trial_app/storeLinkToReference	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	reference_id=	Required – the id of the reference
	option=	Required – the type link (trial/experiment)
	id=	Required – the id of the experiment/trial
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method storeLinkToReference has one key named id (the id of the created link), and one value associated with this key.		

Table 73 Information for calling deleteReferenceLinkById web service


deleteReferenceLinkById		
Description	This method deletes the reference link (trial or experiment link)	

	depending of the provided argument	
URL	https://istr.chic-vph.eu/trial_app/deleteReferenceLinkById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETERS (parameters should be passed through the URL – query string parameter)	id=	Required – the id of the link
	option=	Required – type of the link (trial/experiment)
Returns	200 OK if reference link has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

5.4.3.3.6 File


The following web services (Table 74 - Table 79) should be used whenever the client needs to store, retrieve or delete information related to files containing experiment data (title of file, description of file, file version, etc.).

Table 74 Information for calling storeTrFile web service

storeTrFile		
Description	This method stores the file information and returns the id	
URL	https://istr.chic-vph.eu/trial_app/storeTrFile	
Encoding	Multipart/form-data	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	subject_id=	Required – the id of the subject with which the file is associated
	title=	Required – the title of the file
	description=	Not required – description of the file
	kind=	Not required – defines what this file is (document, spreadsheet, csv, etc.)
	version=	Required – the version of the

		file (should be in the format X.X for example 1.2)
	sha1sum=	Not required – the sha1 checksum of the file
	comment=	Not required – comments on the file
	file=	Required – the actual file (blob)
	engine=	Not required – The engine that is suitable for executing this file
	license=	Not required – The license associated with this file
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The JSON object returned by method storeTrFile has one key, named id, and one value which is associated with this key.		

Table 75 Information for calling deleteTrFile web service

deleteTrFile		
Description	This method deletes a certain file	
URL	https://istr.chic-vph.eu/trial_app/deleteTrFile	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the file
Returns	200 OK if file has been deleted	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	

HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
-------------	---------------------	--

Table 76 Information for calling getTrFileById web service



getTrFileById		
Description	This method returns the file (which is associated with a subject)	
URL	https://istr.chic-vph.eu/trial_app/getTrFileById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the file
Returns (Content-Type: application/force-download Content-Disposition: attachment)	200 OK & attachment	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>

Table 77 Information for calling getTrFilesOfKind web service

getTrFilesOfKind		
Description	This method returns the information of all the files of a specific kind of a given subject	
URL	https://istr.chic-vph.eu/trial_app/getTrFilesOfKind	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS (parameters should be passed through the URL – query string parameter)	subject_id=	Required – the id of the subject
	kind=	Required - kind of file (document, spreadsheet, csv, etc.)
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	

	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getTrFilesOfKind are as many as the different latest version files of a specific kind which are associated with the given subject. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the tr_file entity (see figure 39) and each value of the nested JSON object represents the information of the column.</p>		

Table 78 Information for calling getTrFilesBySubjectId web service



getTrFilesBySubjectId		
Description	This method returns information (only metadata, not attachment) for all the files that are associated with the given subject	
URL	https://istr.chic-vph.eu/trial_app/getTrFilesBySubjectId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS (parameters should be passed through the URL – query string parameter)	id=	Required – the id of the subject
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getTrFilesBySubjectId are as many as the different files that are associated with the given subject. Each value associated with a specific key is represented by a nested JSON object. The keys of the aforementioned nested JSON object are named id, title, description, kind, version, sha1sum, comment, engine, license, created_on, created_by, modified_on, modified_by.</p>		

Table 79 Information for calling getUserTrFiles web service

getUserTrFiles		
Description	This method returns information (only metadata, not attachment) for all the files (stored in in silico trial repository) that have been created by the user with which the saml token is associated	
URL	https://istr.chic-vph.eu/trial_app/getUserTrFiles	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER	Only the SAML token is required.	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getUserTrFiles are as many as the different files that have been created by/for that user. Each value associated with a specific key is represented by a nested JSON object. The keys of the aforementioned nested JSON object are named id, subject, title, description, kind, version, sha1sum, comment, engine, license, created_on, created_by, modified_on, modified_by.</p>		

5.4.4 Metadata repository

5.4.4.1 Description

The metadata repository is the repository for metadata about CHIC resources, namely elementary process models (EPMs) and associated data. This metadata provides descriptions of the foregoing resources according to a number of facets, including multiscale anatomy, units of measurement, biophysical qualities and physiological and pathological aspects. The aim of the repository is to make available annotations of CHIC resources to facilitate, for example, the semantic search of resources but also to warrant the semantic coherence and integration of CHIC resources.

5.4.4.2 Functionality

ID	Functionality
1	Storage of metadata statements
2	Search: both simple look up and inference (to extents yet to be determined)

5.4.4.1 Programmatic/User Interface

The baseline metadata storage is adapted from the output of the VPH project RICORDO. Metadata recorded in RDF format is stored in an RDF database.

An RDF database deployed on a server provides a SPARQL endpoint where the metadata repository may be accessed.

A REST web service interface provides programmatic search access to the metadata for predefined queries (RICORDO RDF Webservice).

A reference user interface using the above Web service could be made available to search the metadata repository.

Search of the metadata repository may be used to power the semantic search of the CHIC resource repositories (clinical data and models).

5.5 Infrastructure layer

5.5.1 Private Cloud

5.5.1.1 Description

The CHIC private cloud is an infrastructural component that provides computational, storage and network resources to other functional components. It provides an abstraction and virtualization of the underlying computational resources and it helps to scale-up, manage, clone, migrate or replace the needed resources without affecting the rest of the architecture since it provides a de-coupling and virtualization layer.

5.5.1.2 Functionality

ID	Functionality
1	Provides computational resources to other components, such as the hypermodelling execution framework.
2	Provides storage space to other components, such as the storage repositories.
3	Provides network connectivity to other components, such as the security services.

5.5.1.1 User Interface

The CHIC private cloud is based on the Openstack cloud infrastructure platform. Openstack has a web interface, called Dashboard, for management and administration of the most common tasks of the cloud infrastructure. Below we present some screenshots of the most common administrative tasks that can be performed through Dashboard.

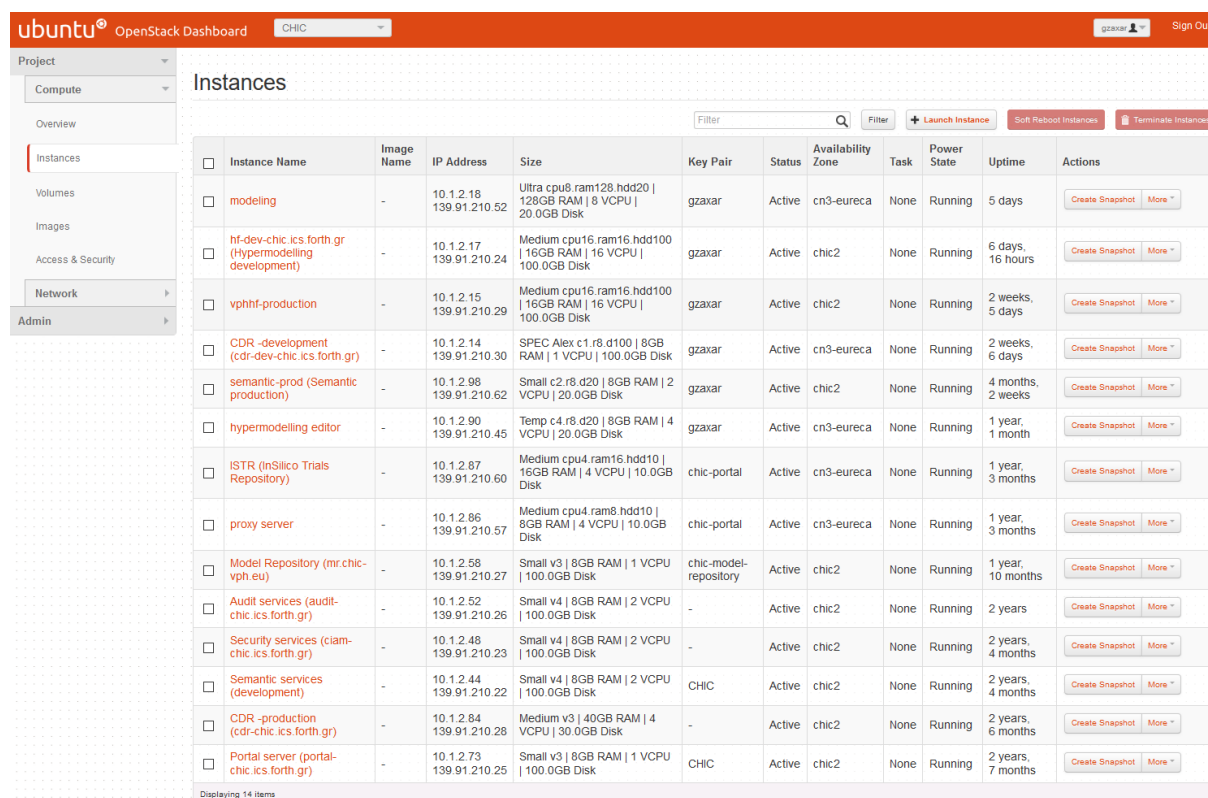
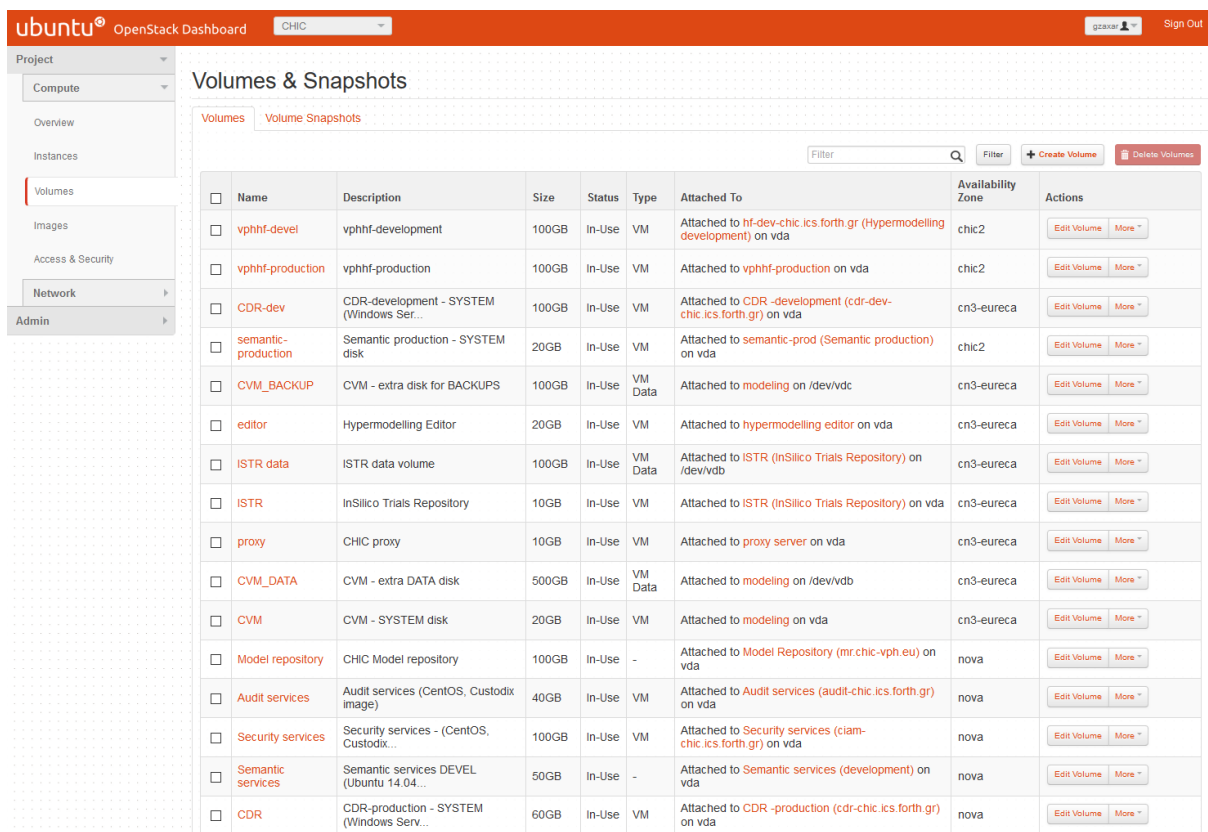


Figure 42 Openstack Dashboard: Instance management

In Figure 42 we see the web interface for the management of VM instance management. With this menu and its submenus an administrator can see an overview of the running details of a VM, can perform power management options (power on, shutdown, reboot, suspend), view logs, view the VMs console, take snapshots or launch new instances.



Name	Description	Size	Status	Type	Attached To	Availability Zone	Actions
vphhf-devel	vphhf-development	100GB	In-Use	VM	Attached to hf-dev-chic.ics.forth.gr (Hypermodelling development) on vda	chic2	Edit Volume More
vphhf-production	vphhf-production	100GB	In-Use	VM	Attached to vphhf-production on vda	chic2	Edit Volume More
CDR-dev	CDR-development - SYSTEM (Windows Ser...	100GB	In-Use	VM	Attached to CDR -development (cdr-dev-chic.ics.forth.gr) on vda	cn3-eureca	Edit Volume More
semantic-production	Semantic production - SYSTEM disk	20GB	In-Use	VM	Attached to semantic-prod (Semantic production) on vda	chic2	Edit Volume More
CVM_BACKUP	CVM - extra disk for BACKUPS	100GB	In-Use	VM Data	Attached to modeling on /dev/vdc	cn3-eureca	Edit Volume More
editor	Hypermodelling Editor	20GB	In-Use	VM	Attached to hypermodelling editor on vda	cn3-eureca	Edit Volume More
ISTR_data	ISTR data volume	100GB	In-Use	VM Data	Attached to ISTR (InSilico Trials Repository) on /dev/vdb	cn3-eureca	Edit Volume More
ISTR	InSilico Trials Repository	10GB	In-Use	VM	Attached to ISTR (InSilico Trials Repository) on vda	cn3-eureca	Edit Volume More
proxy	CHIC proxy	10GB	In-Use	VM	Attached to proxy server on vda	cn3-eureca	Edit Volume More
CVM_DATA	CVM - extra DATA disk	500GB	In-Use	VM Data	Attached to modeling on /dev/vdb	cn3-eureca	Edit Volume More
CVM	CVM - SYSTEM disk	20GB	In-Use	VM	Attached to modeling on vda	cn3-eureca	Edit Volume More
Model repository	CHIC Model repository	100GB	In-Use	-	Attached to Model Repository (mr.chic-vph.eu) on vda	nova	Edit Volume More
Audit services	Audit services (CentOS, Custodix image)	40GB	In-Use	VM	Attached to Audit services (audit-chic.ics.forth.gr) on vda	nova	Edit Volume More
Security services	Security services - (CentOS, Custodix...	100GB	In-Use	VM	Attached to Security services (clam-chic.ics.forth.gr) on vda	nova	Edit Volume More
Semantic services	Semantic services DEVEL (Ubuntu 14.04...	50GB	In-Use	-	Attached to Semantic services (development) on vda	nova	Edit Volume More
CDR	CDR-production - SYSTEM (Windows Serv...	60GB	In-Use	VM	Attached to CDR -production (cdr-chic.ics.forth.gr) on vda	nova	Edit Volume More

Figure 43 Openstack Dashboard: Volume management

In Figure 43 we see the web interface used for the management of volumes (virtual hard disks) and snapshots. Through this menu and submenus an administrator can create volumes, attach or detach them to VM instances, resize, delete, take snapshots or create new volumes from snapshots of other volumes (cloning).

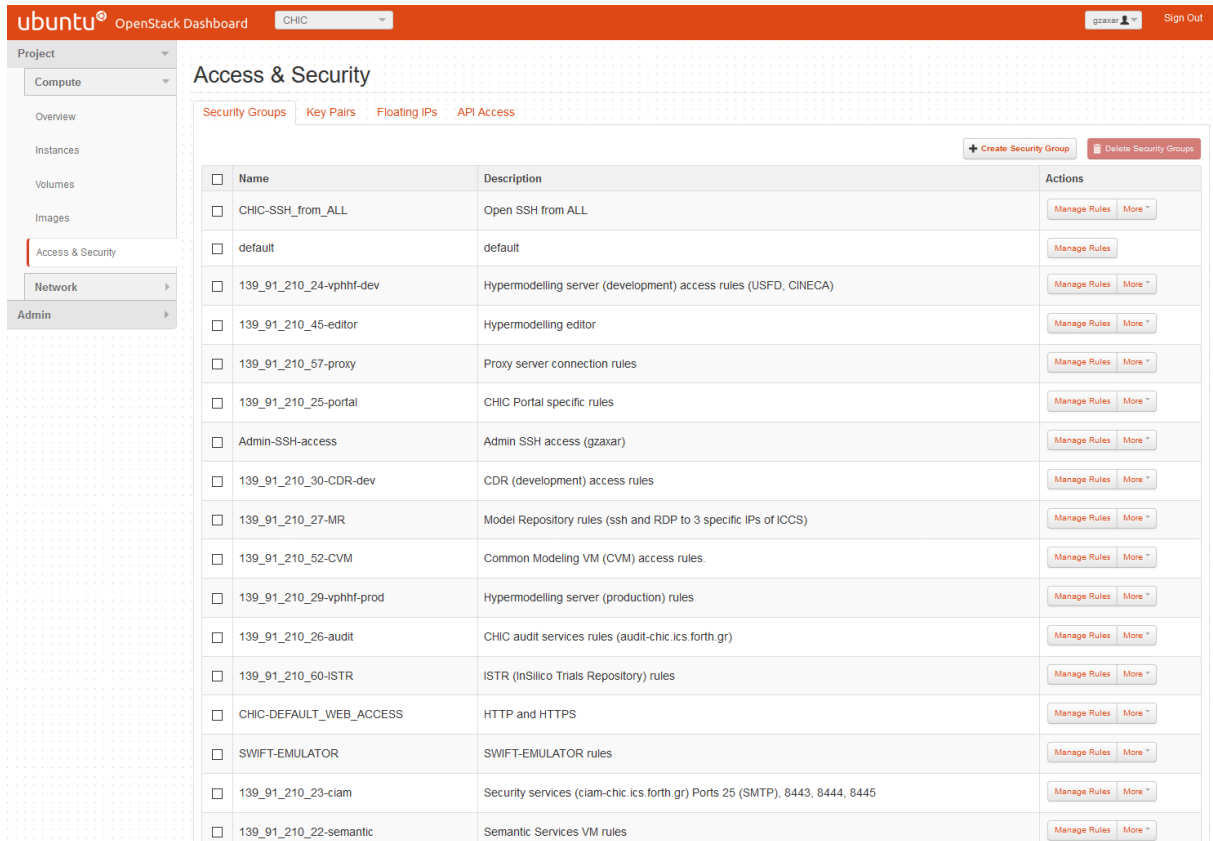


Figure 44 Openstack Dashboard: Access & Security management

In Figure 44 we see the web interface for managing access and security rules of an instance. Firewall rules management, ssh key pairs, floating IP management.

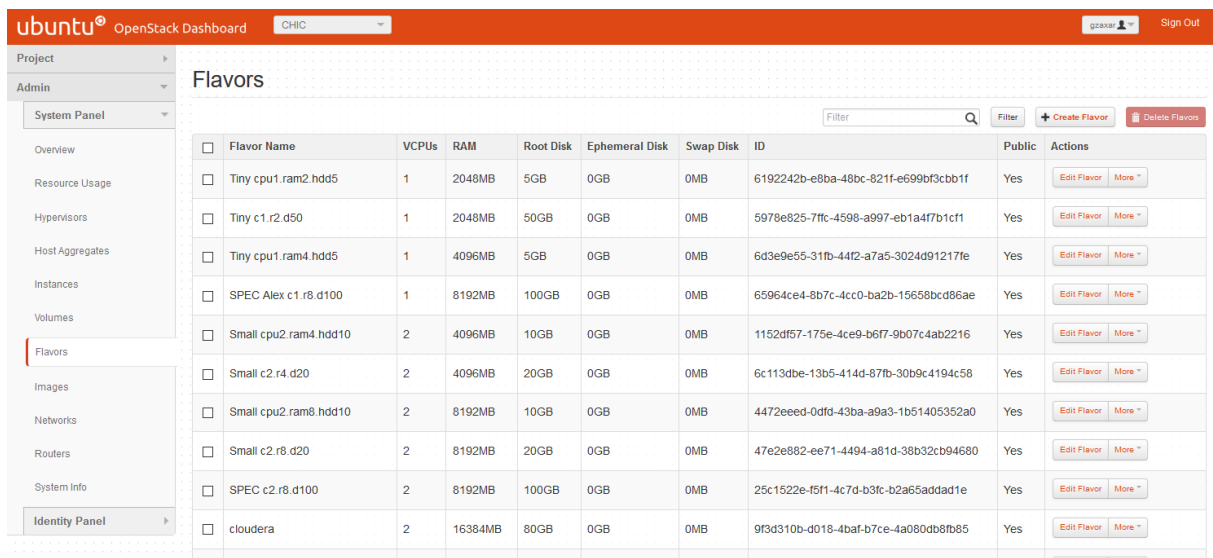
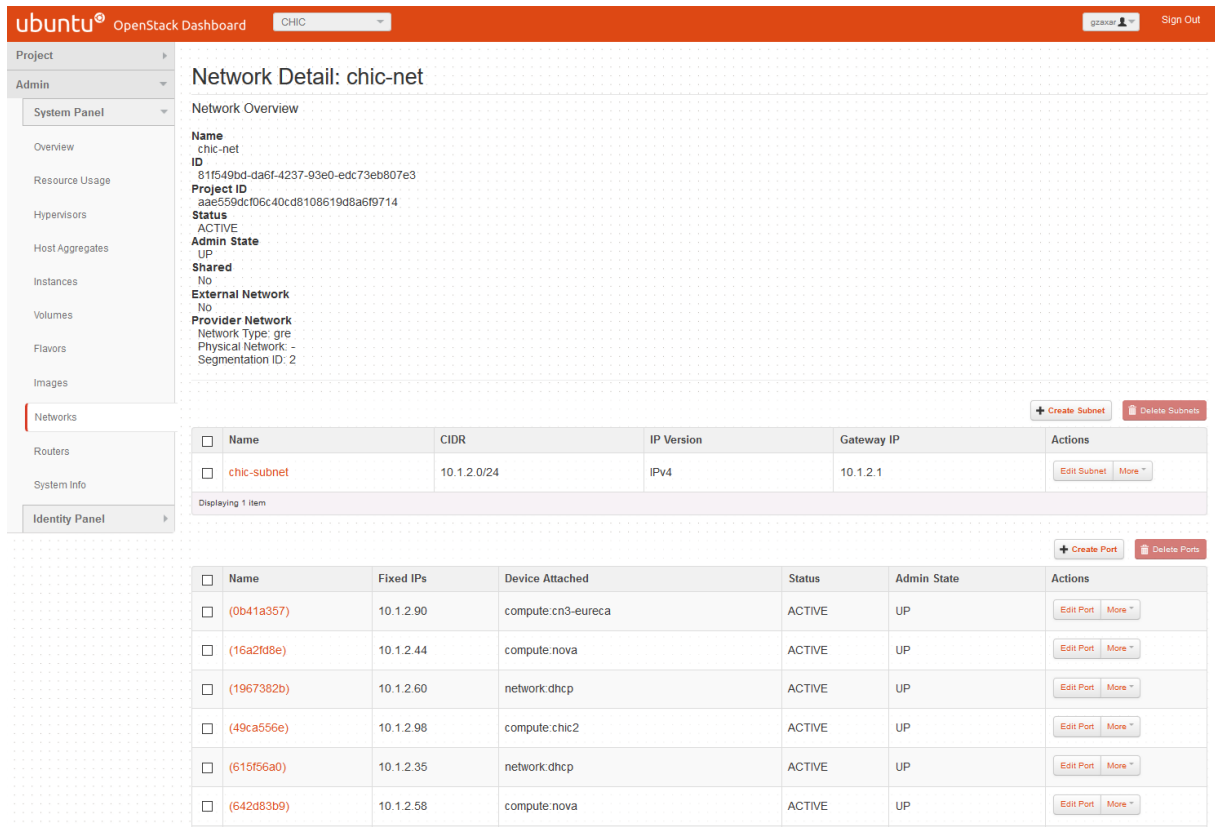


Figure 45 Openstack Dashboard: Flavor management

In Figure 45 we see the web interface for managing instance flavors; pre-configured groups of instance attributes such as RAM, vCPU, hard disk size that are used for launching new instances.



The screenshot shows the OpenStack Dashboard interface for network management. The main content area is titled "Network Detail: chic-net" and displays the following information:

- Network Overview:**
 - Name: chic-net
 - ID: 811549bd-da6f-4237-93e0-edc73eb807e3
 - Project ID: aae559dc106c40cd8108619d8a6f9714
 - Status: ACTIVE
 - Admin State: UP
 - Shared: No
 - External Network: No
 - Provider Network:
 - Network Type: gre
 - Physical Network: -
 - Segmentation ID: 2

Below the overview, there are two tables:

Subnets Table:

Name	CIDR	IP Version	Gateway IP	Actions
chic-subnet	10.1.2.0/24	IPv4	10.1.2.1	Edit Subnet More

Ports Table:

Name	Fixed IPs	Device Attached	Status	Admin State	Actions
(0b41a357)	10.1.2.90	compute:cn3-eureca	ACTIVE	UP	Edit Port More
(16a2fd8e)	10.1.2.44	compute:nova	ACTIVE	UP	Edit Port More
(1967382b)	10.1.2.60	network:dhcp	ACTIVE	UP	Edit Port More
(49ca556e)	10.1.2.98	compute:chic2	ACTIVE	UP	Edit Port More
(615f56a0)	10.1.2.35	network:dhcp	ACTIVE	UP	Edit Port More
(642d883b9)	10.1.2.58	compute:nova	ACTIVE	UP	Edit Port More

Figure 46 Openstack Dashboard: Network management

In Figure 46 we see the web interface for the management of network attributes. Through this menu and submenus an administrator can create subnets, routers, connect networks, DHCP servers.

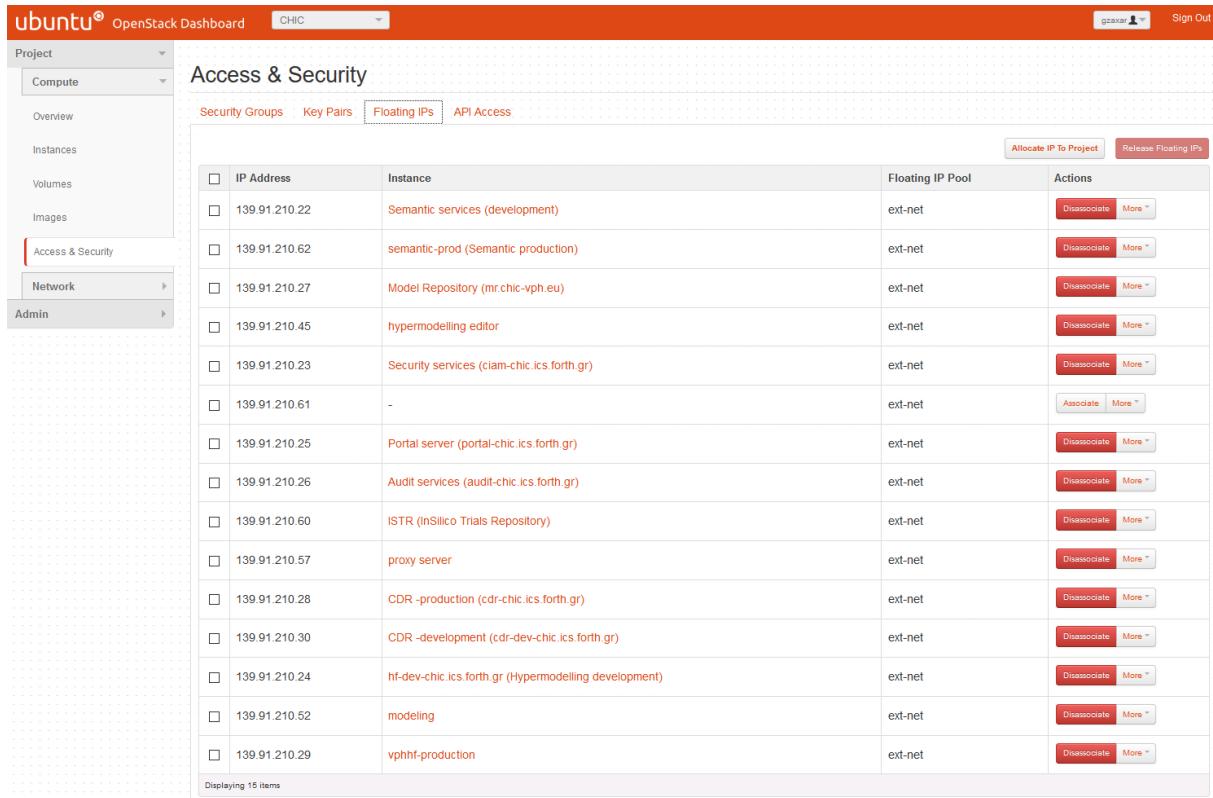


Figure 47 Openstack Dashboard: Floating IP management

In Figure 47 we see the web interface for the management of floating IPs. A floating IP can be allocated from a floating IP pool for a specific project, it can be dynamically allocated during a VM instantiation and it can be attached/detached to a VM instance and forward its network traffic from a network to another.

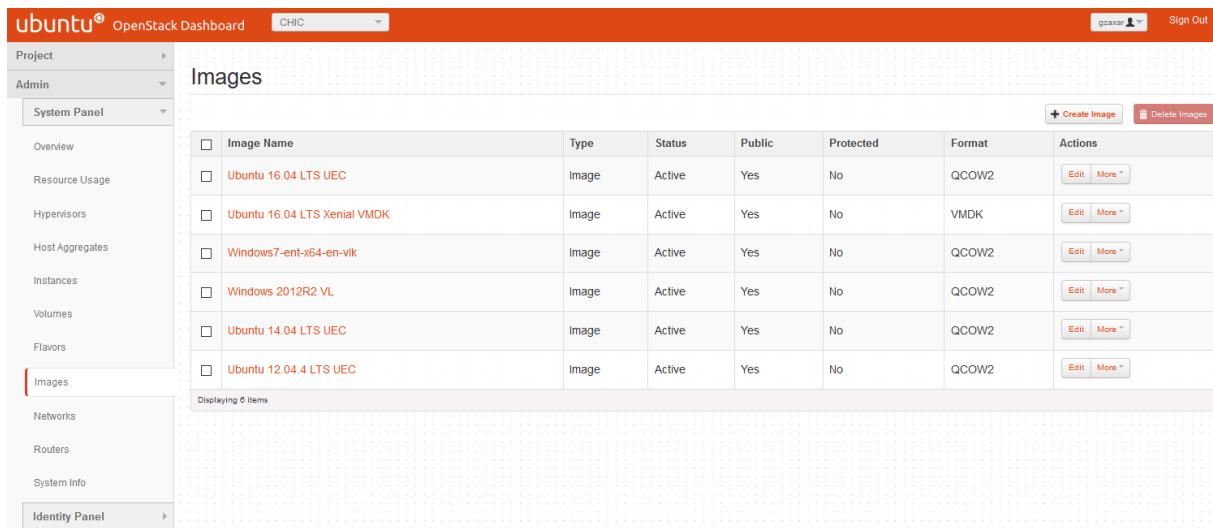


Figure 48 Openstack Dashboard: Image management

In Figure 48 we see the web interface for the management of VM images. These are pre-built images of various flavors of operating systems, libraries, pre-installed programs. During instantiation of a

new VM an administrator can select a VM image, a VM flavor, an existing or a new volume (hard disk) and various other attributes regarding network and security.

The screenshots presented above summarize the most common administrative tasks performed through the Openstack Dashboard. There are also other menus for more specific tasks and there are also command line tools for the fine-grained management tasks that need deeper knowledge of the cloud architecture.

5.5.1.2 API interface

The CHIC private cloud is based on the OpenStack cloud infrastructure software platform. Besides Dashboard, the web interface for managing the most common administrative tasks, the OpenStack platform offers a variety of tools for accessing the underlying services, such as

- Command line interface (CLI) and tools for cloud management.
- Software development kits (SDKs) for many programming languages such as Java, Python, .NET, Ruby, Node.js and PHP.
- Application programming interface (API) based on RESTful web services for direct programmatic access of its services such as Block Storage, Object Storage, Compute, Identity, Orchestration etc.

The complete OpenStack API can be found online at <http://api.openstack.org/> .

6 Deployment view

6.1 Introduction – high level view

In this section we describe the Deployment view, which describes the environment into which the system will be deployed, including the dependencies that the system has on its runtime environment. This view captures the hardware and software environment needs of the system, the technical environment requirements for each element, the mapping of the software elements to the runtime environment that will execute them and any other dependency to technical resources, such as hardware or software that is required.

The technical details around the implementation of a system are not usually the main concern of its architecture specification; these details, however, often govern and restrict many of the design decisions of the architecture while trying to provide a feasible and functional system. Thus, for a complex software system such as the CHIC platform we believe that it is necessary to describe in detail also its Deployment view and the running environment of the system, in terms of software or hardware resources.

Below we outline the technical details for each functional component of the architecture. However, the architecture is still under design into many aspects and many components are not yet implemented, so technical details are still under consideration and may change till the architecture is finalized.

6.2 Presentation layer

6.2.1 Clinical Research Application Framework (CRAF)

Resource	Mapping
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Web application (two-tier, i.e. client side in Browser talking to the server side over HTTP APIs) but also a standalone application in Java exists
Operating system	Cross platform, but Linux is the preferred host Operating System for the server side.
Programming environment	Java 8 on the server side and the standalone application, Javascript on the client side of the web application
Communication technologies or protocols with other CHIC components	HTTP/REST based API with JSON payload, and message oriented communication over the RabbitMQ message broker
Data types that it operates on	Hypermodels as computational artifacts, patient data for invoking the hypermodels, results and visualization files for the outputs of the executions

Technical dependencies or requirements (libraries, tools)	<p>Server side: Undertow application server, PostgreSQL DBMS for persistence, Redis as a caching layer, Nginx web server as a web reverse proxy</p> <p>Client side: Javascript, Angular 2</p> <p>Standalone Application: Java, Apache PDFBox</p>
Computational needs (estimation of hardware needs)	<p>Server side: 8 GB of RAM</p> <p>Client side/ application: 4 GB RAM</p>
Storage needs (estimation of hardware needs)	<p>Server side: 15 GB of storage but depending on usage</p>
URL (if applicable)	N/A

6.2.2 DrEye - Image processing toolkit

Resource	Mapping
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Standalone application
Operating system	Windows (version 8.1 or higher)
Programming environment	C# / C++ / Visual Basic .NET
Communication technologies or protocols with other CHIC components	It interacts indirectly via user interaction with the data repository. Currently, there is no direct communication with the hypermodeling framework planned.
Data types that it operates on	Medical images (.dcm or .mha format)
Technical dependencies or requirements (libraries, tools)	Dr. Eye and its plugin-environment serve as a basis for the toolkit. Moreover, it requires the Insight Registration and Segmentation ToolKit (ITK) library and .NET framework (4.0 or higher).
Computational needs (estimation of hardware needs)	No special requirements. A standard personal computer is sufficient to run the application.
Storage needs (estimation of hardware needs)	N/A
URL (if applicable)	http://biomodeling.ics.forth.gr/dreye http://biomodeling.ics.forth.gr/?page_id=8

6.2.3 Hypermodelling editor

Resource	Mapping
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Web based , two-tier architecture
Operating system	Cross platform, but Linux is the preferred host Operating System for the server side.
Programming environment	Java in the server, Elm ²³ /Javascript on the client (browser)
Communication technologies or protocols with other CHIC components	HTTP/REST based API with JSON payload, and message oriented communication over the RabbitMQ message broker
Data types that it operates on	Hypermodel descriptions coming from the Model Repository, and their RDF annotations from the CHIC Semantic infrastructure
Technical dependencies or requirements (libraries, tools)	Undertow application server ²⁴ , PostgreSQL DBMS for persistence, Redis as a caching layer, Nginx web server as a web reverse proxy
Computational needs (estimation of hardware needs)	Moderate requirements, a server machine with plenty of RAM is always preferred.
Storage needs (estimation of hardware needs)	15 GB of storage but depending on usage
URL (if applicable)	N/A

6.2.4 User Portal

Resource	Mapping
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Web application/framework
Operating system	Windows/Linux (currently installed on Linux, but there's no strict dependency or requirement)

²³ <http://elm-lang.org/>

²⁴ <http://undertow.io/>

Programming environment	Java programming language, Tomcat application server
Communication technologies or protocols with other CHIC components	HTTP/XML/JSON/REST services
Data types that it operates on	N/A
Technical dependencies or requirements (libraries, tools)	Liferay/Tomcat/MySQL
Computational needs (estimation of hardware needs)	Server machine running a web server. Depends on the expected load of the portal.
Storage needs (estimation of hardware needs)	1 GB of storage space
URL (if applicable)	https://portal.chic-vph.eu/

6.3 Service layer

6.3.1 Hypermodelling execution framework

Resource	Mapping
Developers	USFD - CINECA
Software type (Web application, standalone application, service, library, framework etc.)	Web application - Software framework
Operating system	The VPH-HF software components are designed to be cross-platform but they have been tested on Unix/Linux platforms.
Programming environment	Python Taverna workflow server and Muscle coupling library are based on Java: they are used but not developed in CHIC.
Communication technologies or protocols with other CHIC components	REST/HTTP-S APIs to access the hypermodelling framework components functionalities. AMQP protocol is used to communicate asynchronously with a list of components following the publish-subscribe pattern. SSH protocol is used for data transfer and launch some remote execution between the execution machines in the private cloud.
Data types that it operates on	Taverna workflow description format (aka T2flow) and xMML model description format for the workflow description and Baclava XML format for the input set description. The hypermodelling framework will have also to access models, data, and any other information necessary to the workflow execution.

Technical dependencies or requirements (libraries, tools)	<p>The execution framework is mainly constituted by VPH-HF which is mainly developed in the Django web service framework with a MySQL database and few Python components belonging to the Django ecosystem. RabbitMQ/Celery are used for event and queue based interactions.</p> <p>Taverna server orchestrates the execution of the models and a series of scripts wrap every model to make it possible to launch via a standardised interface. Muscle library is necessary to execute strongly coupled models developed via message passing paradigm.</p>
Computational needs (estimation of hardware needs)	<p>The VPH-HF orchestration layer needs a couple of cores, and 1-2 GB of RAM; to this it has to be added all the cores and memory that the hypomodels require in order to be executed. Taverna server (based on Tomcat server) requires few more GiB of memory (1GiB min) and processors to monitor the workflow execution.</p> <p>The amount of memory and CPUs required by the VPH-HF computational layer increases linearly w.r.t the number of requests per second of workflow execution.</p>
Storage needs (estimation of hardware needs)	<p>The VPH-HF orchestrates multiple hypo/hypermodels execution that is performed in a sandbox in the local file system. The needs in term of storage will be the same as the hypomodels to be executed and is model dependent.</p>
URL (if applicable)	<p>https://github.com/INSIGNEO/VPH-HF https://github.com/INSIGNEO/VPH-HF-automation-tools</p>

6.3.2 Visualization toolkit

Resource	Mapping
Developer	BED
Software type (Web application, standalone application, service, library, framework etc.)	Standalone
Operating system	Windows
Programming environment	C++
Communication technologies or protocols with other CHIC components	Command line arguments allow tasks to be executed and data exchanged via files on local disk.
Data types that it operates on	Clinical data (dicom, mha, mhd nifti, json) and data from model simulation
Technical dependencies or requirements (libraries, tools)	Open source libraries: VTK, ITK, Qt and Qwt

Computational needs (estimation of hardware needs)	Min screen size: small laptop (about 28x18cm) Graphics and CPU: Low to high depending on demands.
Storage needs (estimation of hardware needs)	Low
URL (if applicable)	---

6.3.3 Security services

The security framework is comprised of many different components, each one with its one technical details and configuration. In this section we present in brief the technical deployment, but the full design and implementation details can be found in deliverable D5.2.2 “Final version of security tools and guidelines”.

6.3.3.1.1 IAM (Identity and Access Management site)

Resource	Mapping
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	REST and SOAP Web Services
Data types that it operates on	CHIC User Model
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	Development: https://ciam-dev-chic.custodix.com/idm Live: https://ciam.chic-vph.eu/idm

6.3.3.1.2 Idp (Identity Provider)

Resource	Mapping
----------	---------

Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	SAML v2 protocol
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	Development: https://ciam-dev-chic.custodix.com/idp Live: https://ciam.chic-vph.eu/idp

6.3.3.1.3 STS (Secure Token Service)

Resource	Mapping
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web service
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	REST and SOAP Web Services
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	Development: https://ciam-dev-chic.custodix.com/sts Live:

	https://ciam.chic-vph.eu/sts
--	---

6.3.3.1.4 PDP (Policy Decision Point)

Resource	Mapping
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web service
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	Web Service
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	

6.3.3.1.5 PAP (Policy Administration Point)

Resource	Mapping
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	SOAP and REST Web Service
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	

6.3.3.1.6 PIP (Policy Information Point)

Resource	Mapping
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web service
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	SOAP and REST Web Service
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	

6.3.3.1.7 PEP (Policy Enforcement Point)

Resource	Mapping
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Library
Operating system	
Programming environment	Java
Communication technologies or protocols with other CHIC components	Web Service
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	
Storage needs (estimation of hardware needs)	
URL (if applicable)	

6.3.3.1.8 Audit Services

Resource	Mapping
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	Web Service
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	50 GB (for application code and log files)
URL (if applicable)	Development: https://audit-dev-chic.custodix.com/audit-viewer https://audit-dev-chic.custodix.com/audit-parser Live: https://audit.chic-vph.eu/audit-viewer https://audit.chic-vph.eu/audit-parser

6.3.3.1.9 Security Gateway

Resource	Mapping
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web service
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	Web Service
Data types that it operates on	N/A
Technical dependencies or requirements (libraries, tools)	N/A
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware)	25 GB (for application code and log files)

needs)	
URL (if applicable)	

6.3.3.1.10 De-identification

Resource	Mapping
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	SOAP and REST Web Service
Data types that it operates on	N/A
Technical dependencies or requirements (libraries, tools)	N/A
Computational needs (estimation of hardware needs)	2 core
Storage needs (estimation of hardware needs)	50 GB (for application code and log files)
URL (if applicable)	Development: https://ttp-dev-chic.custodix.com Live: https://ttp-chic.custodix.com

6.3.4 Semantic services

6.3.4.1 Ontology-based semantic services

Resource	Mapping
Developer	UCL
Software type (Web application, standalone application, service, library, framework etc.)	Web services
Operating system	Cross-platform
Programming environment	JAVA
Communication technologies or protocols with other CHIC components	REST
Data types that it operates on	N/A

Technical dependencies or requirements (libraries, tools)	Metadata repository and Knowledge Base
Computational needs (estimation of hardware needs)	N/A
Storage needs (estimation of hardware needs)	Web server application
URL (if applicable)	https://github.com/open-physiology/chic https://github.com/open-physiology/owlkb https://github.com/open-physiology/rdfstore

6.3.4.2 Folksonomy semantic services

Resource	Mapping
Developer	BED
Software type (Web application, standalone application, service, library, framework etc.)	Web application, RESTful APIs
Operating system	Cross platform
Programming environment	Java, MongoDB, JavaScript, JQuery, Tomcat
Communication technologies or protocols with other CHIC components	RESTful APIs JSON
Data types that it operates on	Resource URI
Technical dependencies or requirements (libraries, tools)	Spring framework
Computational needs (estimation of hardware needs)	2 x CPU @ 2.6GHz, RAM 8 GB
Storage needs (estimation of hardware needs)	50GB for application, data, and logs
URL (if applicable)	Web app: http://api.ccgv.org.uk/taggingapp/ RESTful API: http://api.ccgv.org.uk/taggingservice/tags

6.3.4.3 Knowledge base

Resource	Mapping
Developer	UCL
Software type (Web application, standalone application, service, library, framework etc.)	Web accessible database
Operating system	Cross-platform
Programming environment	Java
Communication technologies or protocols with other CHIC components	Ontology-based services (REST) and Java API

Data types that it operates on	Baseline option: OWL (Ontology Web Language)
Technical dependencies or requirements (libraries, tools)	Baseline option: OWLAPI, Reasoners
Computational needs (estimation of hardware needs)	Intensive (Prototype needs dedicated server and 64Gb RAM)
Storage needs (estimation of hardware needs)	Web server application
URL (if applicable)	https://github.com/open-physiology/owlkb

6.4 Data layer

6.4.1 Model repository

Resource	Mapping
Developer	ICCS
Software type (Web application, standalone application, service, library, framework etc.)	Web based (the User Interface part)
Operating system	Cross-platform
Programming environment	Programming language: Python, XML, Javascript, HTML, CSS Database management system: MySQL community edition Web application framework: Django Web server: Apache HTTP server
Communication technologies or protocols with other CHIC components	<ul style="list-style-type: none"> • REST services through HTTP protocol and JSON • Passing of messages through AMQP protocol
Data types that it operates on	Models and tools (structured information and files)
Technical dependencies or requirements (libraries, tools)	<ol style="list-style-type: none"> 1) MySQL community edition (GPL license link: http://www.gnu.org/licenses/old-licenses/gpl-2.0.html) 2) Django Rest Framework (Copyright (c) 2011-2016, Tom Christie All rights reserved. Link: http://www.django-rest-framework.org/#license) 3).djangosaml2 (Apache 2 license, link: https://pypi.python.org/pypi/djangosaml2/) 4) dm.xmlsec.binding (BSD license, link: https://pypi.python.org/pypi/dm.xmlsec.binding/1.3.2) 5) Python 2.7 (Open Source, link: https://www.python.org/download/releases/2.7/license/) 6) XML security library (MIT license, link:

	https://www.aleksey.com/xmlsec/) 7) Django (BSD license, link: https://www.djangoproject.com/foundation/faq/) 8) jQuery library (MIT license, link: https://jquery.org/license/) 9) Bootstrap Framework (MIT license, link: http://getbootstrap.com/getting-started/)
Computational needs (estimation of hardware needs)	Intel Xeon E312xx (1 vCPU)
Storage needs (estimation of hardware needs)	60 GB
URL (if applicable)	https://mr.chic-vph.eu

6.4.2 Clinical data repository

Resource	Mapping
Developer	UBERN
Software type (Web application, standalone application, service, library, framework etc.)	Web based
Operating system	The Web application is running on a Microsoft Windows Server 2012 R2 64bit.
Programming environment	Programming languages: C#, Javascript, HTML, CSS Database management system: Microsoft SQL Server 2012 R2. Web application framework: Microsoft ASP.NET Framework, ASP.NET MVC and ASP.NET Web API Web server: Microsoft Internet Information Services (IIS)
Communication technologies or protocols with other CHIC components	REST API, JSON
Data types that it operates on	Clinical data (CDISC ODM) Imaging data (DICOM, MetalImage, Analyze, Nifti) Genetic / molecular data (MINiML) Histopathology data (JPEG, CSV, XML)
Technical dependencies or requirements (libraries, tools)	- ASP.NET (http://www.asp.net/) - Entity Framework (https://github.com/aspnet/EntityFramework) - SimpleITK (http://www.simpleitk.org/) - ReCaptcha (https://www.google.com/recaptcha) - Fuseki (http://jena.apache.org/)

	<ul style="list-style-type: none"> - Fellow Oak DICOM for .NET (https://github.com/fo-dicom/fo-dicom) - dotNetRDF (http://dotnetrdf.org/) - Newtonsoft Json (http://www.newtonsoft.com/json) - VDS.Common (https://www.nuget.org/packages/VDS.Common/) - jQuery (https://jquery.org/) - Bootstrap (http://www.getbootstrap.com) - HDF5DotNet (http://hdf5.net/) - Statismo (https://github.com/statismo/statismo) - FontAwesome (http://fontawesome.github.io/Font-Awesome) - Google Web Fonts (https://www.google.com/fonts) - ANTLR (http://www.antlr.org/) - Helix Toolkit (https://github.com/helix-toolkit) - Apache log4net (https://logging.apache.org/log4net/) - Modernizr (https://modernizr.com/) - OWIN (http://owin.org/) - RestSharp (http://restsharp.org/) - Web Grease (https://www.nuget.org/packages/WebGrease/) - AngularJS (https://angularjs.org/) - Elasticsearch, Logstash, Filebeat, Kibana (https://www.elastic.co/de/products) - Rdfstore, LOLS (http://open-physiology.org)
Computational needs (estimation of hardware needs)	<p>Production VM: 40 GB RAM, 2.9 GHz (4 sockets and 4 virtual processors)</p> <p>Development VM: 8 GB RAM, 2.9 GHz (1 socket and 1 virtual processor)</p>
Storage needs (estimation of hardware needs)	500 GB (depending on the number of patients)
URL (if applicable)	<p>Production</p> <p>Website: https://cdr.chic-vph.eu</p> <p>API: https://cdr.chic-vph.eu/api</p> <p>Development</p> <p>Website: https://cdr-dev-chic.ics.forth.gr</p> <p>API: https://cdr-dev-chic.ics.forth.gr/api</p>

6.4.3 In silico trial repository

Resource	Mapping
Developer	ICCS
Software type (Web application, standalone)	Web based (the User Interface part)

application, service, library, framework etc.)	
Operating system	Cross-platform
Programming environment	Programming language: Python, XML, Javascript, HTML, CSS Database management system: MySQL community edition Web application framework: Django Web server: Apache HTTP server
Communication technologies or protocols with other CHIC components	<ul style="list-style-type: none"> • REST services through HTTP protocol and JSON
Data types that it operates on	Input data, links to hypermodels and results of model execution (structured information and files)
Technical dependencies or requirements (libraries, tools)	<ol style="list-style-type: none"> 1) MySQL community edition (GPL license link: http://www.gnu.org/licenses/old-licenses/gpl-2.0.html) 2) Django Rest Framework (Copyright (c) 2011-2016, Tom Christie All rights reserved. Link: http://www.django-rest-framework.org/#license) 3).djangosaml2 (Apache 2 license, link: https://pypi.python.org/pypi/djangosaml2/) 4) dm.xmlsec.binding (BSD license, link: https://pypi.python.org/pypi/dm.xmlsec.binding/1.3.2) 5) Python 2.7 (Open Source, link: https://www.python.org/download/releases/2.7/license/) 6) XML security library(MIT license, link: https://www.aleksey.com/xmlsec/) 7) Django (BSD license, link: https://www.djangoproject.com/foundation/faq/) 8) jQuery library (MIT license, link: https://jquery.org/license/) 9) Bootstrap Framework (MIT license, link: http://getbootstrap.com/getting-started/)
Computational needs (estimation of hardware needs)	Intel Xeon E312xx (4 vCPU)
Storage needs (estimation of hardware needs)	100 GB
URL (if applicable)	https://istr.chic-vph.eu

6.4.4 Metadata repository

Resource	Mapping
Developer	UCL
Software type (Web application, standalone application, service, library, framework etc.)	Web accessible database
Operating system	Cross-platform
Programming environment	N/A
Communication technologies or protocols with other CHIC components	SPARQL endpoint or Ontology-based services (REST)
Data types that it operates on	Metadata serialized in RDF
Technical dependencies or requirements (libraries, tools)	TBD
Computational needs (estimation of hardware needs)	TBD
Storage needs (estimation of hardware needs)	Web server application
URL (if applicable)	Third parties backend and https://github.com/open-physiology/rdfstore

6.5 Infrastructure layer

6.5.1 Private Cloud

Resource	Mapping
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	OpenStack cloud technology platform.
Operating system	Linux
Programming environment	N/A (The software is not being developed in CHIC. It is provided as free software under the Apache 2.0 license)
Communication technologies or protocols with other CHIC components	It supports but not directly communicate with other CHIC components. However, it does provide many different technologies for communication and management: <ul style="list-style-type: none"> • REST services: JSON/XML format, • Command Line Interface (CLI) tools, • Software development kits (SDKs): Java, Node.js, Python, Ruby, .NET, PHP
Data types that it operates on	N/A

Technical dependencies or requirements (libraries, tools)	Operating System, network configuration, hardware resources.
Computational needs (estimation of hardware needs)	Many computer nodes (>3)
Storage needs (estimation of hardware needs)	Many TBs of storage (>5)
URL (if applicable)	https://www.openstack.org/ https://developer.openstack.org/

Appendix 1 – Abbreviations and acronyms

<i>AD</i>	Architectural Description
<i>API</i>	Application Programming Interface
<i>CATS</i>	Custodix Anonymization Tool
<i>CLI</i>	Command-Line Interface
<i>CPU</i>	Central Processing Unit
<i>CRAF</i>	Clinical Research Application Framework
<i>CRF</i>	Case Report Form
<i>CDISC</i>	Clinical Data Interchange Standards Consortium
<i>CSV</i>	Comma Separated Values
<i>DHCP</i>	Dynamic Host Configuration Protocol
<i>DICOM</i>	Digital Imaging and Communications in Medicine
<i>EC</i>	European Commission
<i>FMA</i>	Foundational Model of Anatomy
<i>GUI</i>	Graphical User Interface
<i>GUID</i>	Globally Unique Identifier
<i>HDOT</i>	Health Data Ontology Trunk
<i>HIS</i>	Hospital Information System
<i>HTML</i>	HyperText Markup Language
<i>HTTP</i>	Hypertext Transfer Protocol
<i>IAM</i>	Identity and Access Management
<i>IdP</i>	Identity Provider
<i>IP</i>	Internet Protocol
<i>IPR</i>	Intellectual Property Rights
<i>IT</i>	Information Technology
<i>ITK</i>	Insight Segmentation and Registration Toolkit

<i>JSON</i>	JavaScript Object Notation
<i>LDAP</i>	Lightweight Directory Access Protocol
<i>LOLS</i>	Local Ontology Lookup Service
<i>MAF3</i>	Multimod Application Framework 3
<i>miRNA</i>	microRNA
<i>MRI</i>	Magnetic Resonance Imaging
<i>NHS</i>	National Health Service
<i>ODM</i>	Operational Data Model
<i>OWL</i>	Web Ontology Language
<i>OWLKB</i>	RICORDO semantic reasoner server
<i>PAP</i>	Policy Administration Point
<i>PDP</i>	Policy Decision Point
<i>PEP</i>	Policy Enforcement Point
<i>PIMS</i>	Personal Information Management Service
<i>PIP</i>	Policy Information Point
<i>PSE</i>	Problem Solving Environment
<i>RAM</i>	Random Access Memory
<i>RDF</i>	Resource Description Framework
<i>REST</i>	Representational State Transfer
<i>RM-ODP</i>	Reference Model of Open Distributed Processing
<i>RPC</i>	Remote Procedure Call
<i>RST</i>	Request Security Token
<i>RSTR</i>	Request Security Token Response
<i>SAML</i>	Security Assertion Markup Language
<i>SDK</i>	Software Development Kit
<i>SOAP</i>	Simple Object Access Protocol

<i>SPARQL</i>	Protocol and RDF Query Language
<i>SSL</i>	Secure Sockets Layer
<i>SSO</i>	Single Sign-On
<i>STS</i>	Secure Token Service
<i>TLS</i>	Transport Layer Security
<i>TTP</i>	Trusted Third Party
<i>UI</i>	User Interface
<i>URL</i>	Uniform Resource Locator
<i>UUID</i>	Universally Unique Identifier
<i>vCPU</i>	virtual CPU
<i>VM</i>	Virtual Machine
<i>VPH</i>	Virtual Physiological Human
<i>VPH-HF</i>	VPH Hypermodelling Framework
<i>VPH-OP</i>	Osteoporotic VPH
<i>WSDL</i>	Web Service Description Language
<i>XACML</i>	eXtensible Access Control Markup Language
<i>XDAS</i>	Distributed Audit Service
<i>XML</i>	Extensible Markup Language
<i>XMML</i>	eXploration and Mining Markup Language

Appendix 2 – List of figures

Figure 1 Conceptual model of architectural description from IEEE1471	10
Figure 2 : Functional view: A high level view of the CHIC functional components	21
Figure 3 The architecture of the CHIC clinical domain and CRAF as its core.....	23
Figure 4 The main CRAF window	24
Figure 5 Patient selection.....	24
Figure 6 Overview of the input/output parameters before the execution of the chosen hypermodel for the selected patient and the chosen question	25
Figure 7 Screenshot of DrEye	27
Figure 8 Side by side comparison of DICOM slides on two different time points, overview of the histograms and of the statistics magnitudes for the corresponding ROIs.....	27
Figure 9 User interface of the hypermodelling editor.	30
Figure 10 VPH-HF admin panel-dashboard	33
Figure 11 CCGVis user interface	42
Figure 12 CCGVis loading data.....	43
Figure 13 CCGVis task execution	44
Figure 14 CCGVis interactive 2D visualization	45
Figure 15 CCGVis interactive 3D visualization	45
Figure 16 CCGVis comparison of simulation using superimposed isosurfaces	46
Figure 17 CCGVis plot view.....	47
Figure 18 CCGVis saving a report.....	47
Figure 19 Folksonomy service snapshot – add a tag to a resource	59
Figure 20 Folksonomy service snapshot – view/edit/delete tags	60
Figure 21 Folksonomy service snapshot – view community tags.....	60
Figure 22 Interactions of the Model Repository with the other CHIC components.....	67
Figure 23 The main page of the Model and Tool Repository	68
Figure 24 The third step of the wizard. The user uploads a variable number of files related to the new model (source code, executables, documentation, etc.).....	69
Figure 25 The fourth step of the wizard. The user categorizes their new model based on the 13 perspectives that have been defined within CHIC.....	69
Figure 26 A screenshot of part of the content of the Model Repository.....	71
Figure 27 The user may apply many actions to a model (view the parameters, view the files, etc.) ...	71
Figure 28 Brokered Authentication Flow with the clinical data repository	95
Figure 29: The general workflow for data upload	96
Figure 30: The audit data model XDASv2 used by the clinical data repository for auditing	98
Figure 31: The components of the audit systems and the interactions with the clinical data repository	99

Figure 32: A visual representation of interactions between clinical data repository and RICORDO components.....100

Figure 33: User dialog to annotate an object with anatomical regions using the autocomplete function offered by the Local Ontology Lookup Service (LOLS).101

Figure 34: The web-based user interface main view of the clinical data repository.....102

Figure 35: The dynamic search query builder integrated in the web-based user interface103

Figure 36 Interactions of the In Silico Trial Repository with the other CHIC components112

Figure 37 The main page of the In Silico Trial Repository113

Figure 38 The first step of the wizard. The user provides information related to the in silico trial to which the new in silico experiment belongs114

Figure 39 The fifth step of the wizard. The user uploads one or more output files related to the simulation.....114

Figure 40 Part of the page of the In Silico Trial Repository which indicates the available in silico trials116

Figure 41 Information related to the last four simulations (in silico experiments) of Nephroblastoma multimodeller hypermodel117

Figure 42 Openstack Dashboard: Instance management148

Figure 43 Openstack Dashboard: Volume management149

Figure 44 Openstack Dashboard: Access & Security management150

Figure 45 Openstack Dashboard: Flavor management.....150

Figure 46 Openstack Dashboard: Network management.....151

Figure 47 Openstack Dashboard: Floating IP management152

Figure 48 Openstack Dashboard: Image management.....152

Appendix 3 – List of tables

Table 1 Stakeholders of CHIC	15
Table 2 Interactions of the Model Repository with the other CHIC components	66
Table 3 Information for calling storeTool web service	72
Table 4 Information for calling getAllTools web service	73
Table 5 Information for calling getToolById web service.....	73
Table 6 Information for calling getToolByParameterId web service	74
Table 7 Information for calling getToolByUuid web service	75
Table 8 Information for calling deleteToolById web service.....	75
Table 9 Information for calling storeParameter web service.....	76
Table 10 Information for calling deleteParameter web service.....	78
Table 11 Information for calling getParametersByToolId web service.....	78
Table 12 Information for calling getParameterById web service	79
Table 13 Information for calling getMandatoryParametersByToolId web service	80
Table 14 Information for calling getInputParametersByToolId web service	80
Table 15 Information for calling getOutputParametersByToolId web service.....	81
Table 16 Information for calling storeProperty web service.....	82
Table 17 Information for calling getAllProperties web service	83
Table 18 Information for calling getPropertyById web service	83
Table 19 Information for calling storePropertyValue web service.....	84
Table 20 Information for calling deletePropertyValue web service	85
Table 21 Information for calling getPropertyValuesByToolId web service.....	85
Table 22 Information for calling deletePropertyById web service.....	86
Table 23 Information for calling storeReference web service	87
Table 24 Information for calling deleteReferenceById web service.....	88
Table 25 Information for calling getReferencesByToolId web service	88
Table 26 Information for calling storeFile web service.....	89
Table 27 Information for calling deleteFile web service	90
Table 28 Information for calling getFileById web service	91
Table 29 Information for calling getPackageByToolId web service	91
Table 30 Information for calling getFilesOfKind web service.....	92
Table 31 Information for calling getFilesByToolId web service.....	93
Table 32 Parties involved in the general workflow for data upload.....	97
Table 33 HTTP methods supported by the clinical data repository REST API	104
Table 34 The pagination concept applied to large result sets returned by the clinical data repository.	104

Table 35 The includable attribute demonstrated on the basis of the groups resource implemented by the clinical data repository.....	105
Table 36 The possible return states used by the clinical data repository to indicate a successful completion of a request.....	106
Table 37: The possible return states used by the clinical data repository to indicate an unsuccessful completion of a request.....	106
Table 38: The template used to describe the API endpoint resources of the clinical data repository	107
Table 39 Interactions of the In Silico Trial Repository with the other CHIC components.....	111
Table 40 Information for calling storeTrial web service	118
Table 41 Information for calling getAllTrials web service	119
Table 42 Information for calling getUserTrials web service.....	119
Table 43 Information for calling getTrialById web service.....	120
Table 44 Information for calling getTrialByModelId web service.....	121
Table 45 Information for calling deleteTrialById web service.....	121
Table 46 Information for calling storeExperiment web service	122
Table 47 Information for calling getUserExperiments web service.....	123
Table 48 Information for calling getUserPendingExperiments web service	124
Table 49 Information for calling getAllExperimentsByTrialId web service	125
Table 50 Information for calling getExperimentById web service.....	125
Table 51 Information for calling getExperimentByUuid web service	126
Table 52 Information for calling getExperimentStatusById web service	127
Table 53 Information for calling getExperimentsByStatus web service	127
Table 54 Information for calling updateExperimentStatus web service.....	128
Table 55 Information for calling deleteExperimentById web service.....	129
Table 56 Information for calling storeMiscellaneousParameter web service.....	129
Table 57 Information for calling getAllMiscellaneousParameters web service	130
Table 58 Information for calling getUserMiscellaneousParameters web service	131
Table 59 Information for calling getAllMiscellaneousParametersByExperimentId web service	132
Table 60 Information for calling getMiscellaneousParameterById web service	132
Table 61 Information for calling deleteMiscellaneousParameterById web service	133
Table 62 Information for calling storeSubject web service.....	134
Table 63 Information for calling deleteSubjectById web service	135
Table 64 Information for calling getAllSubjects web service	135
Table 65 Information for calling getUserSubjects web service	136
Table 66 Information for calling getSubjectById web service	136
Table 67 Information for calling storeTrReference web service	137
Table 68 Information for calling getAllTrReferences web service.....	138

Table 69 Information for calling getTrReferencesByTrialId web service	139
Table 70 Information for calling getTrReferencesByExperimentId web service	140
Table 71 Information for calling deleteTrReferenceById web service.....	140
Table 72 Information for calling storeLinkToReference web service	141
Table 73 Information for calling deleteReferenceLinkById web service.....	141
Table 74 Information for calling storeTrFile web service.....	142
Table 75 Information for calling deleteTrFile web service.....	143
Table 76 Information for calling getTrFileById web service	144
Table 77 Information for calling getTrFilesOfKind web service.....	144
Table 78 Information for calling getTrFilesBySubjectId web service	145
Table 79 Information for calling getUserTrFiles web service	146