# Deliverable No. 8.4

# Report on the final system

| | |
|---|---|
| Grant Agreement No.: | 600841 |
| Deliverable No.: | D8.4 |
| Deliverable Name: | Report on the final system |
| Contractual Submission Date: | 30/09/2016 |
| Actual Submission Date: | 30/09/2016 |

| Dissemination Level | | |
|---|---|---|
| PU | Public | X |
| 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) | |

| 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.: | D8.4 |
| Document name: | Report on the final system |
| Nature (R, P, D, O)[1] | R |
| Dissemination Level (PU, PP, RE, CO)[2] | PU |
| Version: | 1.0 |
| Actual Submission Date: | 30/09/2016 |
| Editor:<br>Institution:<br>E-Mail: | Nikolaos Tousert<br>ICCS-NTUA<br>tousertn@mail.ntua.gr |

**ABSTRACT:**

This document presents the final report of the CHIC Repositories. More specifically, this deliverable outlines the current status of the Model and Tool, the Clinical Data, the *In Silico* Trial and the RDF CHIC Repositories. Both the user interface and the web services that expose the content of the aforementioned Repositories to the other CHIC components are being documented. Apart from the interface of the aforementioned Repositories, their architecture which conforms to the integrated platform guidelines is also disclosed. Furthermore, as described in this document, the implementation of the appropriate policies and the security mechanisms ensure that all the Repositories operate within the defined legal and ethical framework of CHIC. Finally, some guidelines are being provided for the illustration of some common workflows related to the Repositories, such as the storage of a new model, the retrieval of all the data concerning a complete *in silico* trial, the interactions with the Local Ontology Lookup Service and the data flow.

**KEYWORD LIST:**

Model Repository, Clinical Data Repository, *In Silico* Trial Repository, semantics, hypermodel, hypomodel, model, single sign-on, RESTful web services, medical data, imaging data, clinical data, genetic data, data upload workflow, trial center, data object, annotation, auditing, domain model, Local Ontology Lookup Service, RDF store, OWLKB, pseudonymization, resources, Knowledge Database, Resource Description Framework, Human Phenotype Ontology

[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 research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n$^{o}$ 600841.*

| *MODIFICATION CONTROL* | | | |
|---|---|---|---|
| **Version** | **Date** | **Status** | **Author** |
| 0.1 | 15/08/2016 | Index Draft | Nikolaos Tousert, ICCS-NTUA |
| 0.2 | 18/08/2016 | Draft | Roman Niklaus, UBERN |
| 0.3 | 24/08/2016 | Draft | Philippe Büchler, UBERN |
| 0.4 | 08/09/2016 | Draft | Nikolaos Tousert, ICCS-NTUA |
| 0.5 | 12/09/2016 | Draft | Nikolaos Tousert, ICCS-NTUA |
| 0.6 | 15/09/2016 | Draft | Pierre Grenon, UCL |
| 0.7 | 18/09/2016 | Revision | Dimitra Dionysiou, ICCS-NTUA |
| 0.8 | 22/09/2016 | Draft | Nikolaos Tousert, ICCS-NTUA |
| 0.9 | 26/09/2016 | Draft | Nikolaos Tousert, ICCS-NTUA |
| 1.0 | 30/09/2016 | Final | Georgios Stamatakos, ICCS-NTUA |

**List of contributors**

- Georgios Stamatakos, ICCS-NTUA
- Dimitra Dionysiou, ICCS-NTUA
- Nikolaos Tousert, ICCS-NTUA
- Roman Niklaus, UBERN
- Philippe Büchler, UBERN
- Pierre Grenon, UCL

## Contents

## Figures

## Tables

# 1 Executive Summary

Deliverable 8.4 outlines the final report of the Repositories developed by the CHIC project, concerning the Model and Tool Repository, the Clinical Data Repository, the *In Silico* Trial Repository and the Metadata Repository. The aforementioned Repositories are tailored to the needs and the clinical scenarios of the CHIC project and aim to provide the community with a collaborative interface for exchanging knowledge and sharing work in an effective and standardized way. As documented in this deliverable, the utilization of a number of open source features and tools is expected to enhance usability and accessibility. The design and the development of all the CHIC Repositories have been driven by the guidelines produced by the other work packages, especially WP2 (user needs and requirements) and wp5 (IT Architecture). Since both the clinical data and the models stored in the CHIC Repository environment must conform to the legal and ethical framework developed in WP4, special attention has been given to the development of appropriate authentication and authorization mechanisms so as to ensure that only authorized persons have access to the content of the Repositories. To this end, as presented in chapter 3, the Repositories make use of the brokered authentication mechanism that has been proposed by WP 5. Both the design and the implementation of the aforementioned Repositories are covered in this document.

In more detail, documentation regarding the design and the architecture of the Model and Tool Repository is included in this document. The aforementioned architecture is general enough to support the storage of multiscale cancer models, linkers and data transformation tools along with all the related information (perspectives, parameters, etc.). The documentation of the web user interface which is built upon the aforementioned architecture serves as a reference point for all researchers who might want to interact with the Model Repository, while the documentation of the web services aims to provide the guidelines for the developers of the other CHIC components (CRAF, Hypermodelling Execution Framework, Hypermodelling Editor) that interact with the Repository through the web interfaces. Moreover, in order for the information related to the categorization of the models based on the 13 perspectives to be stored in the CHIC semantic infrastructure in the form of triples, the Model and Tool Repository has been integrated with the CHIC triplestore.

Information related to the implementation of the Clinical Data Repository which has been available to the CHIC users and is running on the CHIC cloud infrastructure is available through this deliverable. More specifically, the Clinical Data Repository includes all the features required to store the different types of data produced during the clinical workflow, which not only includes patient and treatment information, but also medical images, generic examination and histology. Just like the Model Repository, the web services that provide access to the content of the Clinical Data Repository are also presented in this document, as well as its user interface which has been developed using modern web technologies. Furthermore, reference is being made to the auditing system and the integration of the Clinical Data Repository with the CHIC RICORDO framework so as to retrieve information from the data uploaded on the system and storing the corresponding semantic triples back into the CHIC triplestore.

Regarding the *In Silico* Trial Repository, detailed documentation is being provided concerning the workflows related to the persistent storage, the retrieval or the updating of the simulation scenarios and the *in silico* predictions, either through the user interface or through the web services. In addition to this, the architecture of the aforementioned Repository which is based on a relational database, as well as its integration into the whole CHIC platform are also outlined.

Finally, reference is being made to the CHIC Metadata Repository which constitutes the semantic layer that handles the annotation of the CHIC resources (models, clinical data) encoded in RDF. The Metadata Repository, which consists of the Annotations Store and the Knowledge Database, along with some controlled vocabularies, facilitates the creation of machine-processable metadata descriptions of the CHIC resources. As described in this deliverable, the semantically integrated set of the aforementioned descriptions can be interrogated by the other CHIC components through the RDF and OWL ontology services in order to produce comparisons and elicit relationships. For instance, this might be applicable to the Hypermodelling Editor in order to elicit relationships of consistency and correspondence between the parameters or between the models.

# 2 Introduction

The documentation is an important part of software engineering, and it includes the following types:

1. Requirements – Statements that identify attributes, capabilities, characteristics or qualities of a system.
2. Architecture/Design – Overview of software. Includes relations to an environment and construction principles to be used in design of software components.
3. Technical – Documentation of code, algorithms, interfaces and APIs.
4. End user – Manuals for the end-user, system administrators and support staff.
5. Marketing – How to market the product and analysis of the market demand.

Since the Requirements are addressed by WP2, entitled “User Needs and Requirements”, the Architecture is addressed by WP5 “IT Architecture” and the Marketing is addressed by WP12 “Dissemination and Exploitation”, this document mostly addresses the Technical and the End user types of documentation.

Thereafter, this deliverable includes some manuals for the end-user which describe some features of the CHIC Repositories and aim at assisting the user in realizing them. Great effort has been made for the user documentation to be simple, consistent, up to date and not confusing. The Model and Tool Repository user documentation, which is included in chapter 4.3, intends to give assistance to researchers for storing their new models, updating their parameters or even filtering the existing models based on their perspective categorization. The web-based user interface presented in chapter 5.7 constitutes a comprehensive manual that helps authorized users to master the Clinical Data Repository and easily access the stored medical data (imaging, clinical, histological and genetic). Some user guidelines are also being given in chapter 6.3, concerning the interaction with the *In Silico* Trial Repository (storage and retrieval of all the data concerning a complete *in silico* trial).

In addition to the documentation related to the user interface of the CHIC Repositories, this deliverable also includes user guides for accessing the interfaces (APIs). More specifically, chapters 4.4, 5.8, 6.4 and 7.4 constitute a complete information guide for the developers of the other CHIC components in order to programmatically connect to the CHIC Repositories. The web services that have been documented in the aforementioned chapters expose the contents of the Model, Clinical Data, *In Silico* Trial and Metadata Repositories to the other CHIC components (Hypermodelling Editor, CRAF, Hypermodelling Execution Framework, etc.).

Since both the data and the models stored in the CHIC Repository environment must conform to the legal and ethical framework developed by WP4, the authentication and authorization mechanism which is presented in chapter 3 ensures that only authorized people have access to the content of the Repositories.

Even though the IT Architecture of the CHIC platform is addressed by WP5, the authors consider the design and the internal parts of the Repositories relevant to this deliverable, and thereafter, chapter 4.2 outlines the software architecture of the Model and Tool Repository, chapters 5.3, 5.4 and 5.6 address the domain model, the data types and some general concepts of the Clinical Data Repository, chapter 6.2 analyzes the design and the internal components of the *In Silico* Trial Repository, and finally, chapter 7.3 presents the semantic components and the RICORDO architecture.

The deliverable is also complemented with presentation related to the semantic integration with RICORDO, which is addressed in chapters 4.5 and 5.9 concerning the Model and the Clinical Data Repository respectively, whereas the publication of events from the Model Repository and the auditing mechanism of the Clinical Data Repository are described in chapters 4.6 and 5.5 respectively.

Finally, chapter 7 introduces, among others, the annotation of the CHIC resources which make their interpretation explicit, the ontology of them and the annotation vocabulary included in CHICRO schema.

# 3 Authentication

The CHIC Repositories make use of the security framework introduced in Deliverable "D5.2 - Security guidelines and initial version of security tools". Therefore, the users are not directly authenticated by the Repositories (Service Providers) themselves 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 CHIC Repositories provide complete access to the features of their databases 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 Repositories, 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 a CHIC 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 web sites of the CHIC Repositories 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 web site of the corresponding Repository, 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 1 depicts the brokered authentication flow with the CHIC Repositories.

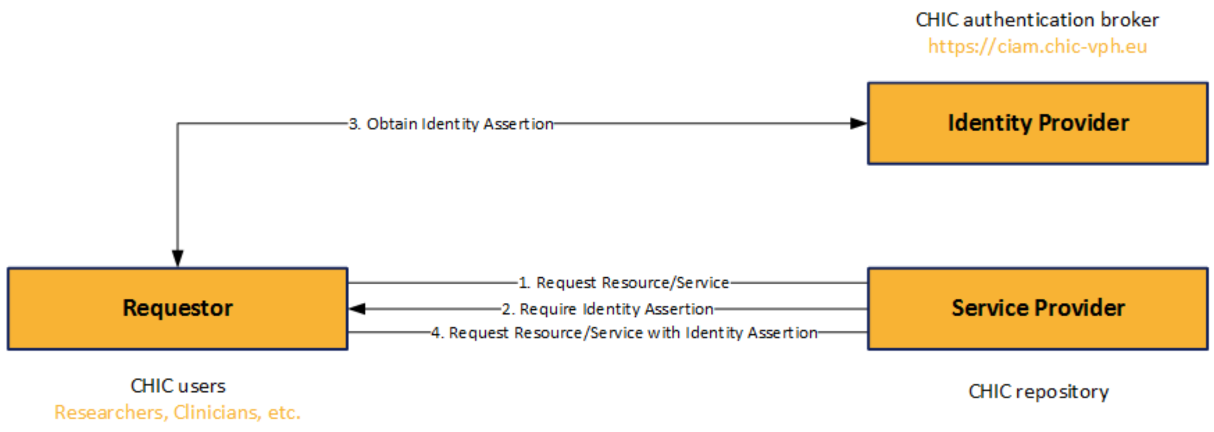

CHIC authentication broker
https://ciam.chic-vph.eu
Identity Provider
3. Obtain Identity Assertion
Requestor
1. Request Resource/Service
2. Require Identity Assertion
4. Request Resource/Service with Identity Assertion
Service Provider
CHIC users
Researchers, Clinicians, etc.
CHIC repository

**Figure 1: Brokered Authentication Flow with the CHIC Repositories**

# 4 Model and Tool Repository

## 4.1 *Introduction*

The CHIC Model and Tool 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 Repository (https://mr.chic-vph.eu ), 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. Based on the new requirements that came into effect during the CHIC project, the Model Repository has evolved from a simple and functional storage component, to a fully integrated and user-friendly web application that supports the execution of complex workflows. Even if the current status of the Model Repository conforms to the user needs and requirements (WP2), to the legal and ethical framework (WP4), to the IT Architecture (WP5) and to the integrated platform guidelines (WP10), the Repository is expected to be constantly updated throughout the remaining period of the CHIC project.

## 4.2 *Architecture of Model and Tool Repository*

As shown in figure 2, the Model and Tool Repository consists of four main entities, the models/tools, the properties, the parameters and the files.

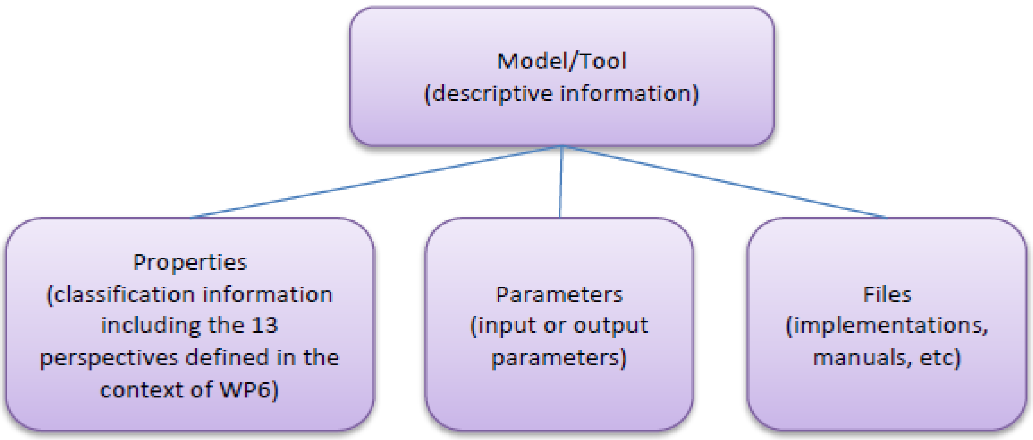

Model/Tool
(descriptive information)
Properties
(classification information including the 13 perspectives defined in the context of WP6)
Parameters
(input or output parameters)
Files
(implementations, manuals, etc)

**Figure 2: Main entities of Model and Tool Repository**

The basic principles of the Model Repository are [1]:

- Each model/tool has basic descriptive information, stored in the entity "mr_tool". This information uniquely defines the model/tool and differentiates it from the other models/tools.
- Each model is categorized based on the perspective from which it is viewed in the basic science context. This metamodeling description of each hypomodel based on the CHIC 13 perspective approach facilitates its technology mediated linking [2]. A detailed documentation of the 13 perspectives and their categories is presented in "D6.2 CHIC cancer component models: initial tested versions".
- The descriptive information of the perspectives is stored in the entity "mr_property". This entity does not contain the value of the perspective (related to a specific model/tool), but only the description of the perspective. The value that a perspective takes in case of a specific model/tool is stored in the entity "mr_tool_property".
- The models are treated as generic stubs, as described in "D7.1: Hypermodelling specifications", which have entry and exit points. Consequently, each model/tool has various parameters, serving as input parameters or output parameters, which are stored in entity "mr_parameter". This entity facilitates the transition from an abstract representation to a concrete one. Logical compatibilities between connected parameters must be taken into account along with the aspect of units, in order to avoid inconsistencies between the connected models/tools.
- Each model/tool may be associated with a set of references, stored in the entity "mr_reference", which provides direct or indirect links to additional material, extending in this way the knowledge base related to the specific model/tool.
- Every model/tool can be accompanied by a set of files. The information concerning the aforementioned files is stored in the entity "mr_file". The entity "mr_file" only holds the metadata of the file and not its data. The data of the files are stored internally in a file based repository. If a file is an implementation or a computational representation of a model/tool, then a suitable engine is specified for running the file.

According to the aforementioned principles of the Model Repository, the Entity-Relationship (ER) diagram of model/tool repository is presented in figure 3:

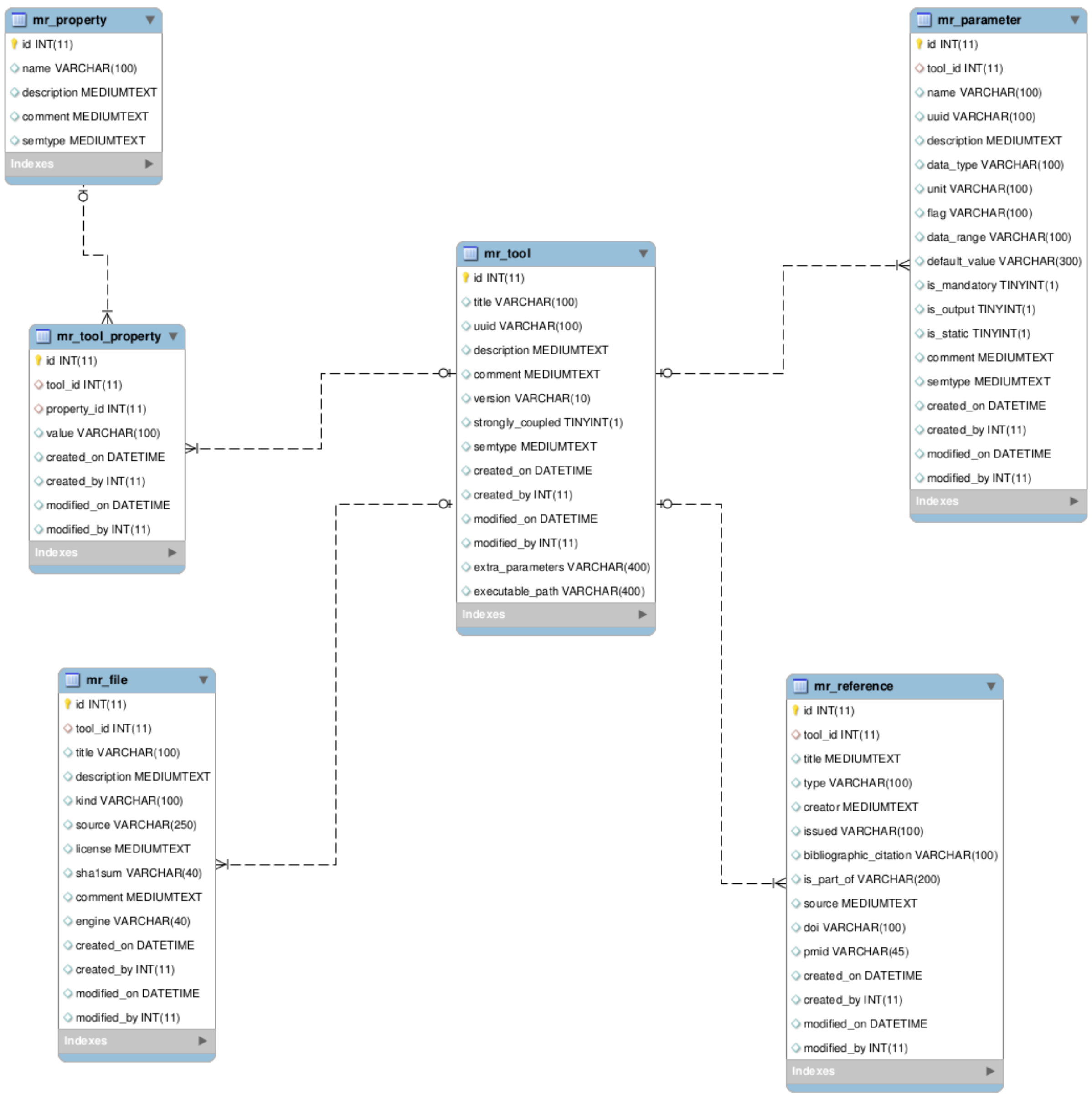


**Figure 3: Entity Relationship (ER) diagram of Model and Tool Repository**

As shown in figure 3, the MySQL database of the Model Repository consists of six entities (tables), which are named “mr_tool”, “mr_parameter”, “mr_reference”, “mr_file”, “mr_property” and “mr_reference”. The attributes (table columns) of the aforementioned entities are presented below:

## Entity mr_tool

- id: Primary key. Used to uniquely identify each table row.
- title: The name of the model/tool. Each model/tool should have a unique name.
- uuid:  The universally unique identifier of each model.
- description: The (short) textual description of the model/tool.
- comment: Any comment that the creator/modifier of the model/tool wants to include.

- version: The version of the model/tool.
- strongly_coupled: A flag that is being used in order to characterize the model as a strongly coupled model or as a non-strongly coupled model. Strongly coupled models are the models that dynamically exchange information (messages) during their execution. Non-strongly coupled models exchange information only before or after their execution. Since the strongly coupled models depend on a multiscale coupling library and environment in order to run, such as the Muscle 2 library, this flag has been created based on the new WP6 requirements.
- semtype: A url representing semantic information about this model.
- extra_parameters: This column holds the string that will be appended in the model's command line argument list. This string is consisted of flag-value pairs, it remains the same for every simulation of this model and it is considered to be essential for each running.
- executable_path: Name of the executable (including relative path) inside the compressed archive. The value of this column is essential in order for VPH-HF component to be able to allocate the executable of the model.
- created_on: The date and time when this model has been created.
- created_by: The identification of the creator of this model.
- modified_on: The date and time when this model has been modified.
- modified_by: The identification of the modifier of this model.

**Entity mr_parameter**

- id: Primary key. Used to uniquely identify each table row.
- tool_id: The id of the model/tool that this parameter is associated with. Linked to the entity "mr_tool".
- name: The name of the parameter. Parameters that belong to the same model should have unique names.
- description: The (short) textual description of what this parameter represents.
- uuid: The universally unique identifier of each parameter.
- data_type: The type of the parameter. Possible values can be number, string and file.
- unit: The units in which the parameter is represented. Only applicable if a parameter is a number.
- data_range: The range of the parameter values separated by "-". Only applicable if the parameter is a number.
- flag: The flag used in the command line argument list with which this parameter is provided through command line. Only applicable if the parameter is a static input parameter.
- default_value: The value that will be used if a parameter value is not provided to the tool.
- is_mandatory: True if this is a mandatory parameter.
- is_output: True if this parameter is an output parameter.
- is_static: True if this parameter is a static parameter. Static parameters are those parameters that their values are being exchanged before or after the execution of the

models. Dynamic parameters are those parameters that their values are being exchanged during the execution of the models.

- comment: Any additional comment concerning this parameter.
- semtype: A url representing semantic information about this parameter.
- created_on: The date and time when this parameter has been created.
- created_by: The identification of the creator of this parameter.
- modified_on: The date and time when this parameter has been modified.
- modified_by: The identification of the modifier of this parameter.

**mr_reference**

- id: Primary key. Used to uniquely identify each table row.
- tool_id: Linked to the model/tool that this resource refers to. Linked to the entity "mr_tool".
- title: The name given to the resource.
- type: The type of the resource. Example values: "book", "journal article", etc.
- creator: The creator(s) of the resource (e.g. authors, etc.).
- issued: The date of formal issuance (e.g. publication) of the resource.
- bibliographic_citation: The bibliographic citation of the resource.
- is_part_of: The related resource that this resource is part of.
- source: The related resource from which the described resource is derived from.
- doi: The DOI (Digital Object Identifier) of the resource. This field is empty if the resource doesn't have a DOI.

**mr_property**

- id: Primary key. Used to uniquely identify each table row.
- name: The name of the perspective.
- description: The (short) textual description of what this property represents.
- comment: Any comment that the creator of the perspective wants to include.
- semtype: A url representing semantic information about this perspective

**mr_tool_property**

- id: Primary key. Used to uniquely identify each table row.
- tool_id: The id of the model/tool. Linked to the entity "mr_tool".
- property_id: The id of the perspective. Linked to the entity mr_property.
- value: The value that the perspective takes in case of a specific model/tool.
- created_on: The date and time when the specific model has been assigned with this perspective value.

- created_by: The identification of the creator who has assigned this perspective value to a specific model.
- modified_on: The date and time when this perspective value has been modified for a specific model.
- modified_by: The identification of the modifier of this perspective value

**mr_file**

- id: Primary key. Used to uniquely identify each table row.
- tool_id: Linked to the model/tool that this file is associated with. Linked to the entity “mr_tool”.
- title: The name of the file.
- description: The (short) textual description of what this file represents.
- kind: Defines what this file is. Example values: “document”, “source code”, “muscle configuration file”, “t2flow”, “compressed package with binary and dependencies”, “xmml description”.
- source: The location where this file is internally stored.
- license: The license associated with this file. It can be the name of a well-known license (Apache, MIT, etc.) or the detailed description of the license.
- sha1sum: The sha1 checksum of this file (data). It is used in order to check the consistency of the file.
- comment: Any additional comment.
- engine: The engine that is suitable for executing this file. Only applicable in case that the file can be executed/run.
- created_on:  The date and time when this file has been uploaded.
- created_by: The identification of the creator who has uploaded this file.
- modified_on: The date and time when this file has been changed (Actually the metadata of this file have been changed)
- modified_by: The identification of the modifier of this file.

The entity Relationship diagram (ER) which has been depicted in figure 3, represents the design of the relational database of the Model and Tool Repository. This design has been documented in deliverable “D8.1: Design of the CHIC Repositories”, but during the CHIC project it has undergone many changes based on the new requirements that came from the other work packages. Most of the changes have been applied to the entities “mr_tool”, which is the entity that holds the basic descriptive information of the model, and “mr_parameter”, which is the entity that holds information about the parameter. The aforementioned changes concerning the aforementioned entities are presented below:

- **mr_tool:** Four more attributes (columns) have been added to this entity.
    - The “uuid” attribute which is the universally unique identifier for each model has been added taking into account the corresponding WP7 requirement.

- The “strongly_coupled” attribute which is a flag that indicates whether the model exchanges dynamically information during its execution, or it exchanges information only before or after its execution, has been added taking into account the corresponding WP6 requirement for being able to store in the Model Repository also strongly coupled models.
- The “extra_parameters” attribute which is a string that consists of flag-value pairs, appended in the model’s command line argument list,has been added taking into account the corresponding WP6 and WP7 requirements. Since some models always include in their command line list some predefined constant arguments, the storage of the aforementioned constant arguments in the relational database a necessity.
- The “executable_path” attribute which is a string that holds the relative path of the model executable inside the compressed archive, has been added taking into account the corresponding WP7 requirement. The value of this attribute is retrieved from the VPH-HF component through a web service in order to allocate the executable of the model inside the compressed folder.

- **mr_parameter:** Two more attributes have been added to this entity.
    - The “uuid” attribute which is the universally unique identifier for each parameter has been added taking into account the corresponding WP7 requirement.
    - The “flag” attribute, has been added taking into account the corresponding WP7 requirement. With the adoption of the “flag” attribute, the provision of input to the models is handled through flags. Consequently the position of the argument values in the command line argument list of the model does not play any role, since each argument value is always accompanied and recognized by the corresponding flag.

The schema of the relational database of the Model Repository that has been just reported in this chapter, has been designed in order to be able to efficiently store within the CHIC platform all the persistent data that are related to the models. Both the metadata description of the models (parameters, perspective values, references, basic descriptive information) and the files that are related to the models (executables, documentation, configuration files, source code) are stored in the MySQL database of the Repository. Since the MySQL database server is the component which is responsible for the persistent storage of the models, it is considered the most sensitive, critical and vital part of the Model Repository. Nevertheless, the Model Repository consists of many other components, such as the Apache Application Server, the Django Web Framework and some back-end and front-end libraries and dependencies. Moreover some security libraries are also being used in order to assure the conformity of the Model Repository with the CHIC legal framework and with internationally recognized security standards. All the aforementioned components were incorporated in order to build not just a local relational database, but a web platform fully integrated into the CHIC platform through the corresponding web services, and also available to the users through its user interface. Table 1 presents all the components, external libraries, applications or dependencies that are being used in the Model Repository along with their licenses.

**Table 1: External components (dependencies, libraries, applications) of the Model and Tool Repository**

| External Component (Dependency – Library – Application) | License | Usage |
|---|---|---|
| Apache HTTP Server | Apache license | A secure, efficient, and extensible server that provides HTTP services in sync with the current HTTP standards. |
| MySQL community edition | GPL license | The relational database server responsible for persistently storing information related to models. |
| Django Rest Framework | Copyright (c) 2011-2016, Tom Christie All rights reserved | A powerful and flexible toolkit for building web APIs |
| djangosaml2 | Apache2 license | A Django application that integrates the PySAML2 library into the Model Repository project in order to be able to incorporate the SAML front-end authentication mechanism. |
| dm.xmlsec.binding | BSD license | XML security library used to authenticate web service requests. |
| XML security library | MIT license | A C library that supports XML security standards (XML signature, XML encryption, etc.). It is being used by djangosaml2 and dm.xmlsec.binding. |
| Django | BSD license | The Python Web Framework that has been used for the development of the Model Repository |
| jQuery library | MIT license | A javascript library which is being used by the Model Repository for event handling, animation, and Ajax calls. |

| | | |
|---|---|---|
| Bootstrap framework | MIT license | HTML, CSS and JS framework for developing part of the front-end of the Model Repository. |

As shown in table 1, nine major external components are used in the Model Repository web application. Some of these components are related to the security (djangosaml2, dm.xmlsec.binding, XML security library), some are related to the back-end of the application (Apache HTTP Server, MySQL Database Server, Django, Django Rest Framework) and finally some are related to the front-end (jQuery library, Bootstrap framework).

Figure 4 depicts the system from a system engineer's point of view. Thereafter, the topology of the software components that have been described in table 1, as well as the physical connections between those components are all presented in figure 4.

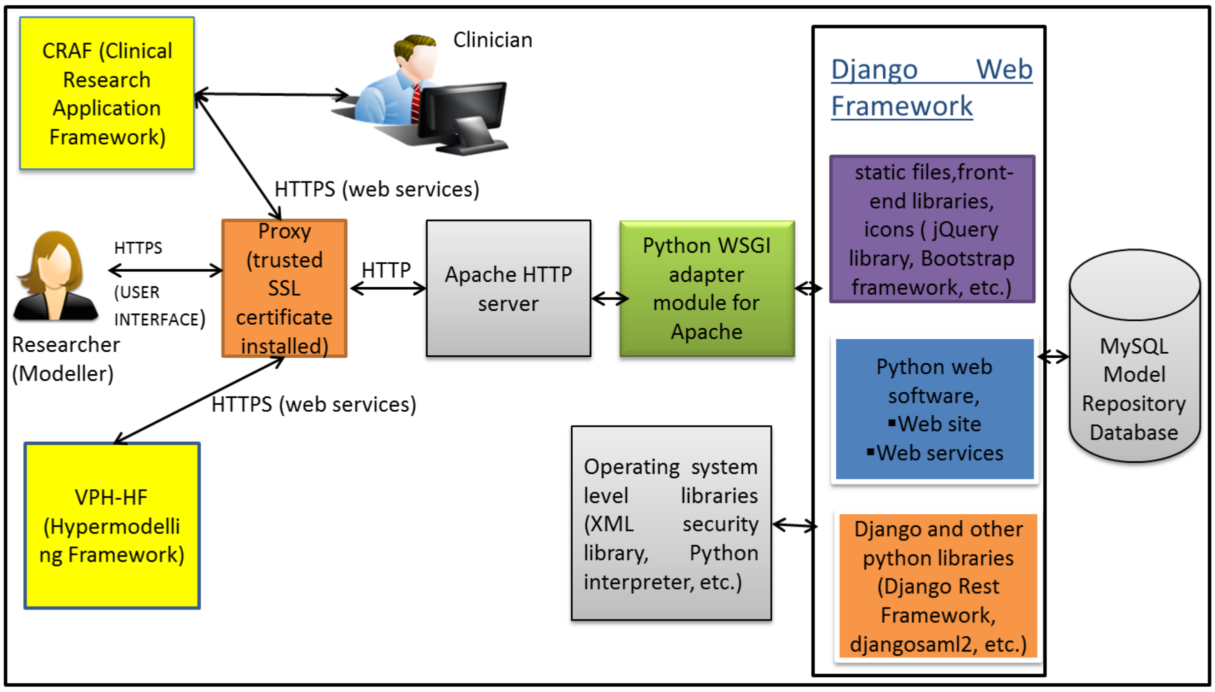

CRAF (Clinical Research Application Framework)
Clinician
HTTPS (web services)
HTTPS
(USER INTERFACE)
Researcher (Modeller)
Proxy (trusted SSL certificate installed)
HTTP
Apache HTTP server
Python WSGI adapter module for Apache
Django Web Framework
static files,front-end libraries, icons ( jQuery library, Bootstrap framework, etc.)
Python web software,
▪Web site
▪Web services
MySQL Model Repository Database
HTTPS (web services)
VPH-HF (Hypermodelling Framework)
Operating system level libraries (XML security library, Python interpreter, etc.)
Django and other python libraries (Django Rest Framework, djangosaml2, etc.)

**Figure 4: Topology of the components of the Model Repository**

Based on the topology depicted in figure 4, some of the components are wrapped by the Django Web Framework (Django Rest Framework, djangosaml2 security library, etc.) and some others have been installed in an operating system level (XML security library, Python interpreter, MySQL driver, etc.). Moreover, the contents of the Model Repository can be exposed through web services to other CHIC components such as the CRAF (Clinical Research Application Framework) and the VPH-HF (Hypermodelling Framework), or they can be exposed through the browser directly to the user. The protection of the privacy and integrity of the exchanged data is ensured by a proxy server which makes use of HTTPS protocol for outbound connections. This is guaranteed by a SSL certificate that has been installed by partner CUSTODIX in the virtual machine that accommodated the proxy. Moreover this certificate has been issued from a trusted Certificate Authority.

Even though most of the components that have been presented in table 1 have undergone some basic configuration based on the needs of the Model Repository web application, most of the work regarding the software development has been done through the Django web framework. The business logic, the presentation layer, the URL dispatching, the object relational mapping and the web services are all handled by the Django framework.

The Django framework is a free and open source web application framework, written in Python, which follows the model-view-controller (MVC) architectural pattern. It encourages rapid development and clean, pragmatic design. It allows high-performing, elegant Web application building. Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself (DRY). Python is used throughout, even for settings, files and data models.

The components and their connections that reside inside the Model Repository Django application are presented in Figure 5:

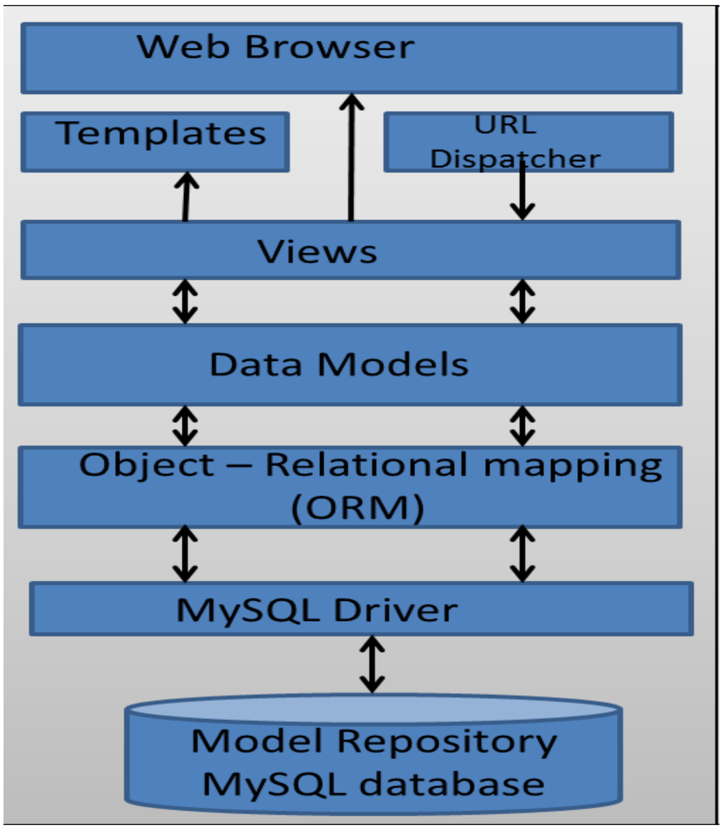

Web Browser
Templates
URL Dispatcher
Views
Data Models
Object – Relational mapping (ORM)
MySQL Driver
Model Repository MySQL database

**Figure 5: Components of the Model Repository Django Application**

As shown in figure 5, the Model Repository Django Application consists of the following components:

- **Object – Relational mapping (ORM):** The object-relational mapping is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language.
- **Data Models:** The data model defines the data in Python and interacts with it.
- **Views:** The view component consists of many view functions. The view function performs the requested function, which typically involves reading or writing to the Model Repository MySQL database. It may include other tasks as well. The business logic of the application is mostly included in this component. After performing any requested tasks, the view returns an HTTP response object (usually after passing the data through a template) to the web

browser. Optionally, the view can save a version of the HTTP response object in the caching system for a specific length of time.

- **Templates:** Templates typically return HTML pages. The Django template language offers HTML authors a simple-to-learn syntax while providing all the power needed for presentation logic. The Model Repository front-end static files (images) and the front-end libraries (jQuery, Bootstrap framework) are included in the templates.
- **URL Dispatcher:** The URL dispatcher maps the requested URL to a view function and calls it. If caching is enabled, the view function can check to see if a cached version of the page exists and bypass all further steps returning the cached version instead.

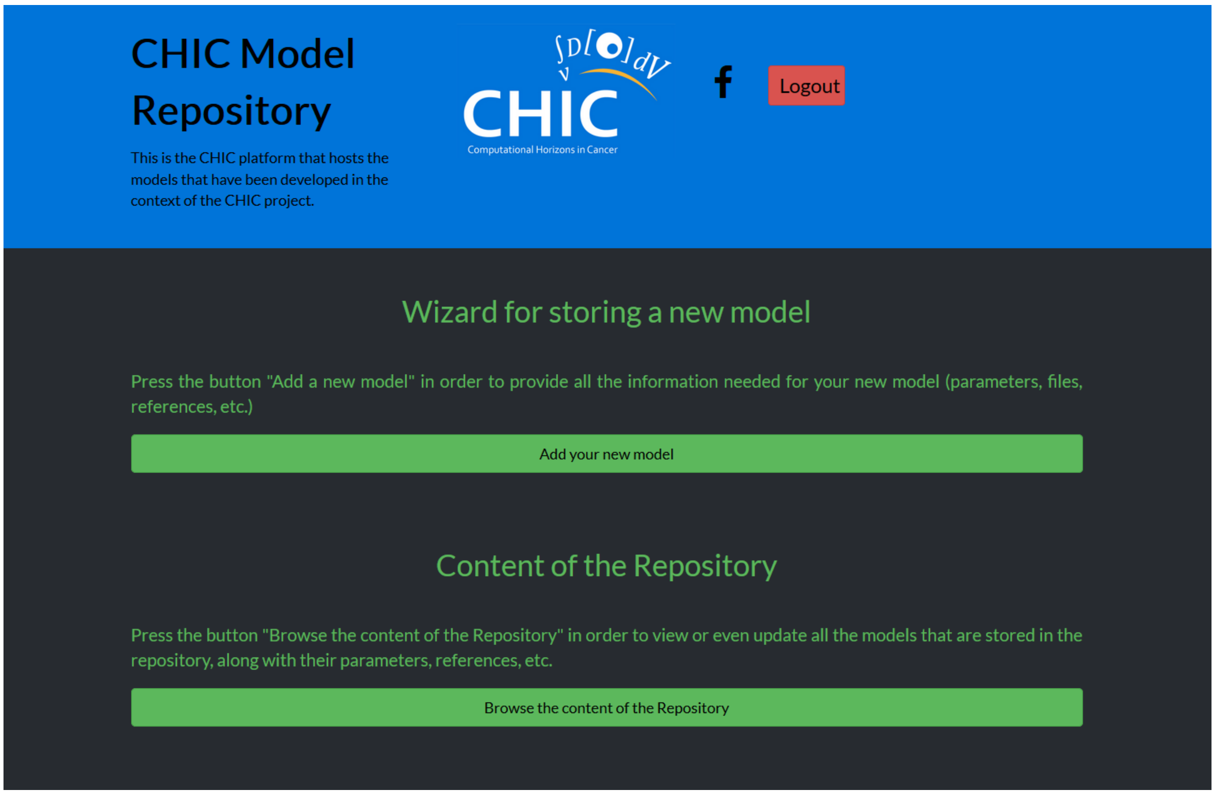
## *4.3 The user interface of Model and Tool Repository*

The principles of user interface design are intended to improve the quality of user interface design. According to Larry Constantine and Lucy Lockwood in their usage-centered design, these principles are [15]:

- **The structure principle:** Design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.
- **The simplicity principle:** The design should make simple, common tasks easy, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.
- **The visibility principle:** The design should make all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs don't overwhelm users with alternatives or confuse with unneeded information.
- **The feedback principle:** The design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.
- **The tolerance principle:** The design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.
- **The reuse principle:** The design should reuse internal and external components and behaviours, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.

The Model Repository, as well as the other CHIC Repositories, makes use of the aforementioned principles in order to produce a user interface which makes the interaction with the user (researcher, clinician, modeller) 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 6 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. The workflows for the storage of a new model and the browsing of the content of the Repository are being described in the next chapters.


CHIC Model Repository

This is the CHIC platform that hosts the models that have been developed in the context of the CHIC project.

Wizard for storing a new model

Press the button "Add a new model" in order to provide all the information needed for your new model (parameters, files, references, etc.)

Content of the Repository

Press the button "Browse the content of the Repository" in order to view or even update all the models that are stored in the repository, along with their parameters, references, etc.

**Figure 6: The main page of the Model Repository**

### 4.3.1 Wizard for storing new models

A wizard has been created for the Model and Tool Repository in order for the user to be able 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:

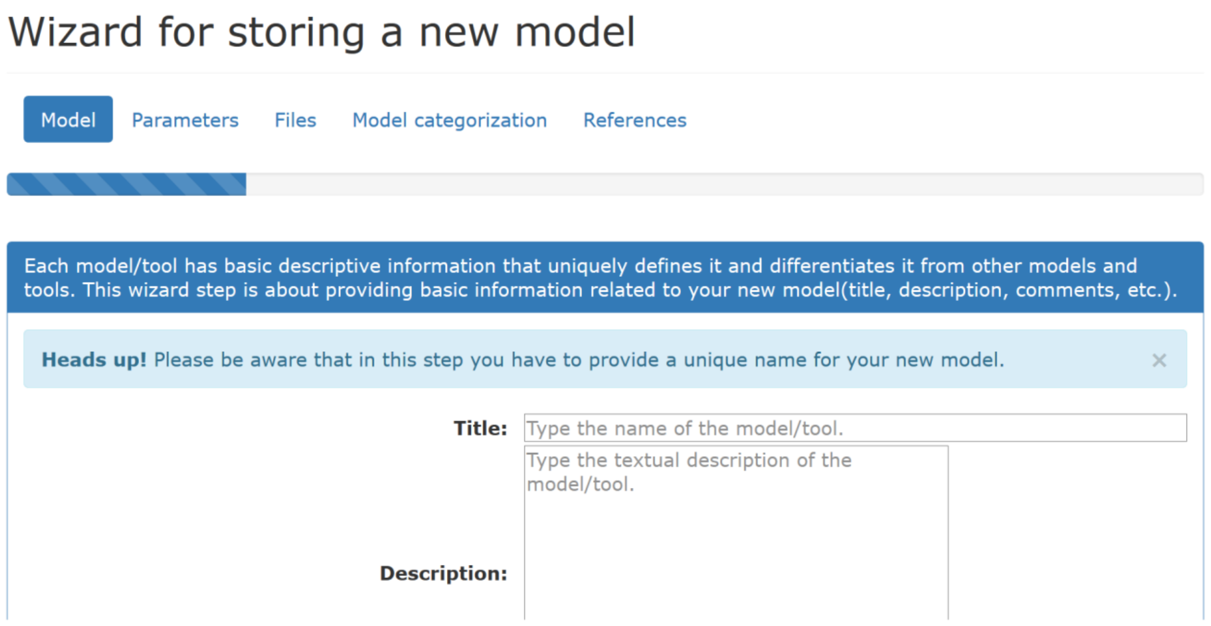
- Basic information of the new model (title, description, additional comments, etc.).
- Definition of the input and output parameters of the new model.
- Categorization of the new model based on the 13 Perspectives that have been designed within CHIC [2].
- References related to the new model (journal articles, conference proceedings, etc.)

This wizard consists of five steps, and in order for the information of the new model to be valid, the user has to:

- Provide a unique title for the new model.
- Provide unique names for all the parameters of the new model.
- Provide unique titles regarding the metadata file names. The actual names of the files that belong to the same model should also be unique.
- Provide unique titles for all the references of the new model.

It should be noted that the user is able to skip the 2 last steps of this wizard for later. More specifically, since the definition of the references of the model and the information about the categorization are not so critical for the execution of the model, the aforementioned steps could be skipped.

The screenshots of the different steps regarding the aforementioned wizard are presented in Figures 7-11.



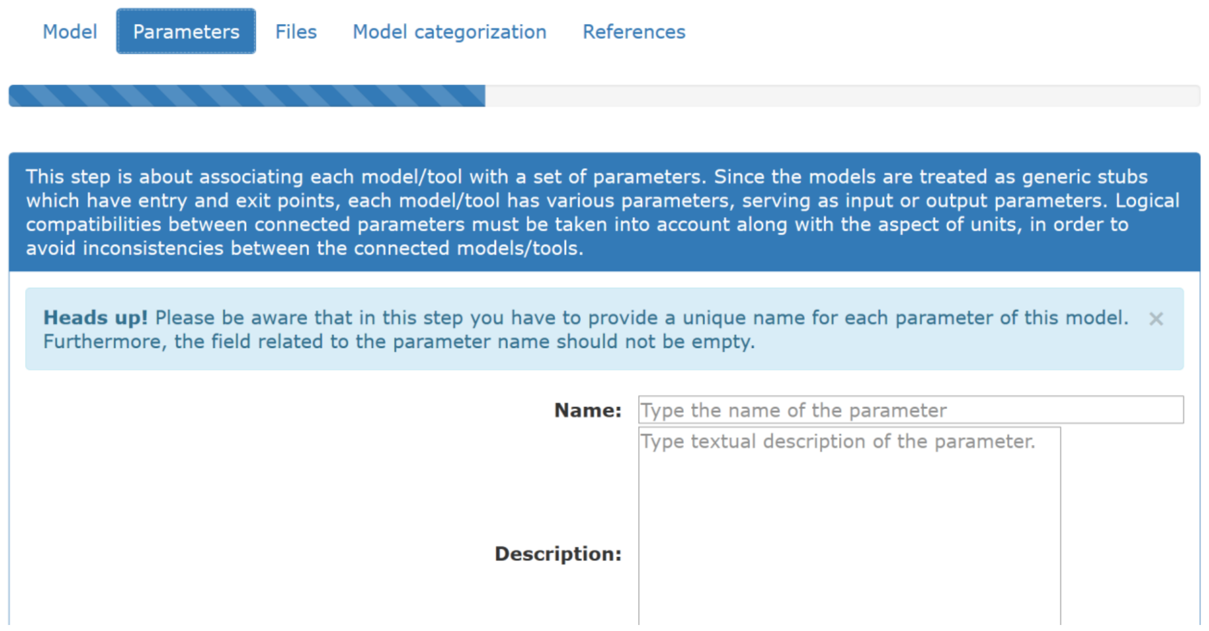
Each model/tool has basic descriptive information that uniquely defines it and differentiates it from other models and tools. This wizard step is about providing basic information related to your new model(title, description, comments, etc.).

**Heads up!** Please be aware that in this step you have to provide a unique name for your new model.

**Title:** Type the name of the model/tool.

**Description:** Type the textual description of the model/tool.

**Figure 7: The first step of the wizard. The user provides the basic information of the new model (title, description, etc.)**



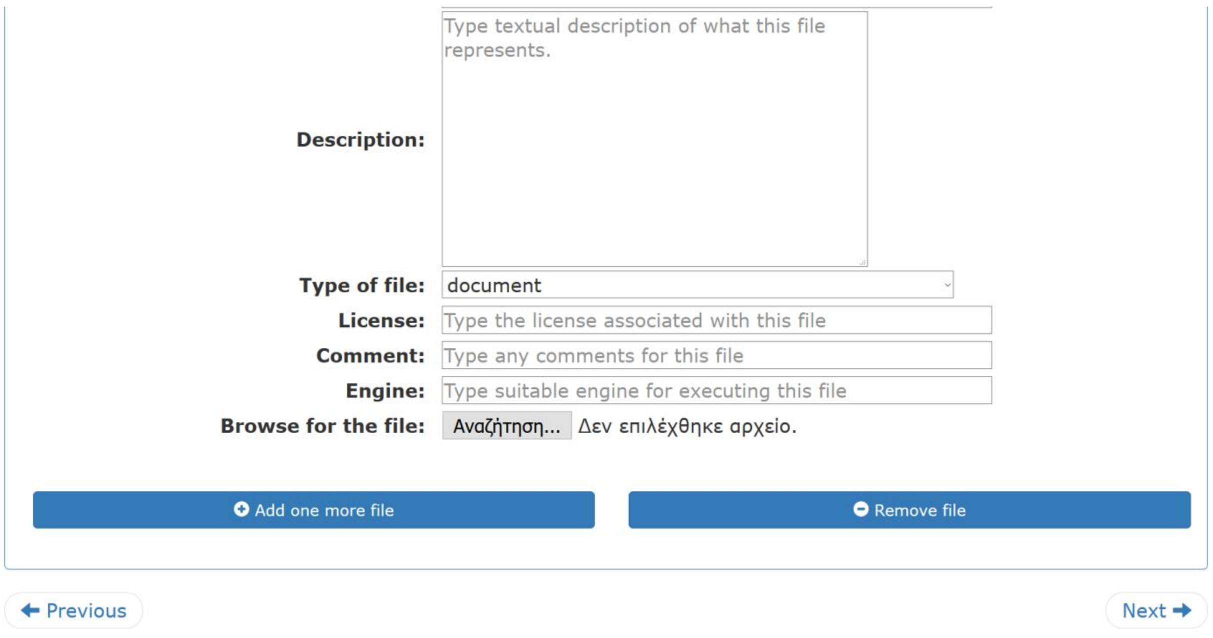
This step is about associating each model/tool with a set of parameters. Since the models are treated as generic stubs which have entry and exit points, each model/tool has various parameters, serving as input or output parameters. Logical compatibilities between connected parameters must be taken into account along with the aspect of units, in order to avoid inconsistencies between the connected models/tools.

**Heads up!** Please be aware that in this step you have to provide a unique name for each parameter of this model. Furthermore, the field related to the parameter name should not be empty.

**Name:** Type the name of the parameter

**Description:** Type textual description of the parameter.

**Figure 8: The second step of the wizard. The user provides information regarding the parameters of the new model (title, description, units, data range, etc.)**



Description: Type textual description of what this file represents.

Type of file: document

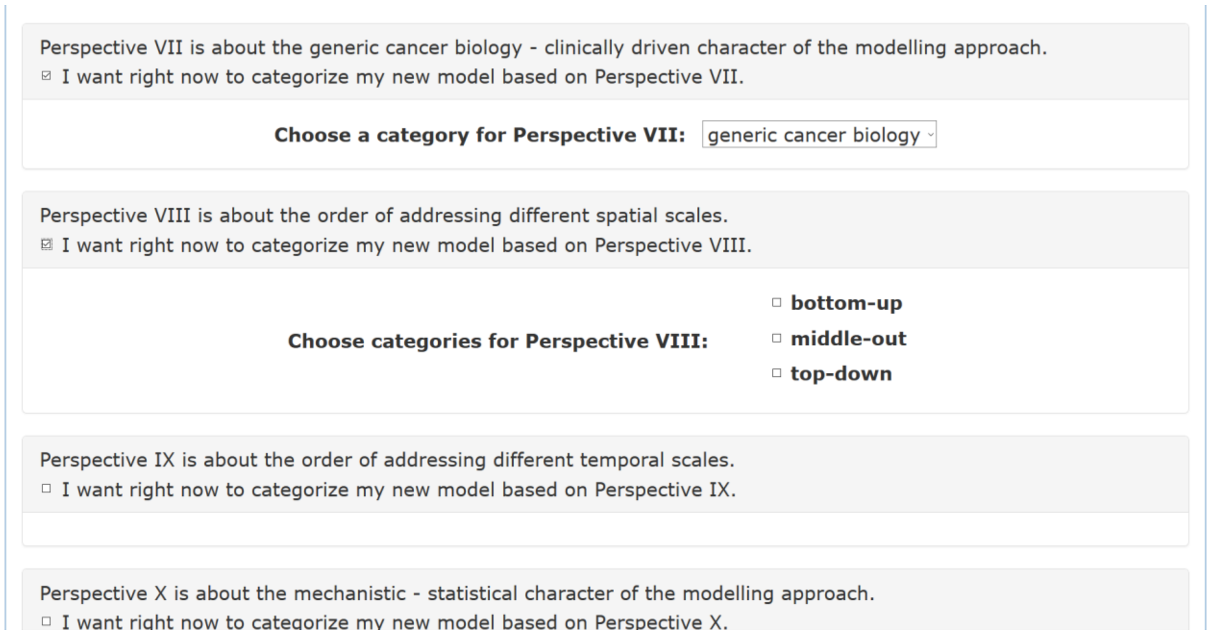
License: Type the license associated with this file

Comment: Type any comments for this file

Engine: Type suitable engine for executing this file

Browse for the file: Αναζήτηση... Δεν επιλέχθηκε αρχείο.

**Figure 9: The third step of the wizard. The user uploads files related to the new model (source code, executables, documentation, etc.)**



Perspective VII is about the generic cancer biology - clinically driven character of the modelling approach.

☑ I want right now to categorize my new model based on Perspective VII.

Choose a category for Perspective VII: generic cancer biology

Perspective VIII is about the order of addressing different spatial scales.

☑ I want right now to categorize my new model based on Perspective VIII.

Choose categories for Perspective VIII:

- ☐ bottom-up
- ☐ middle-out
- ☐ top-down

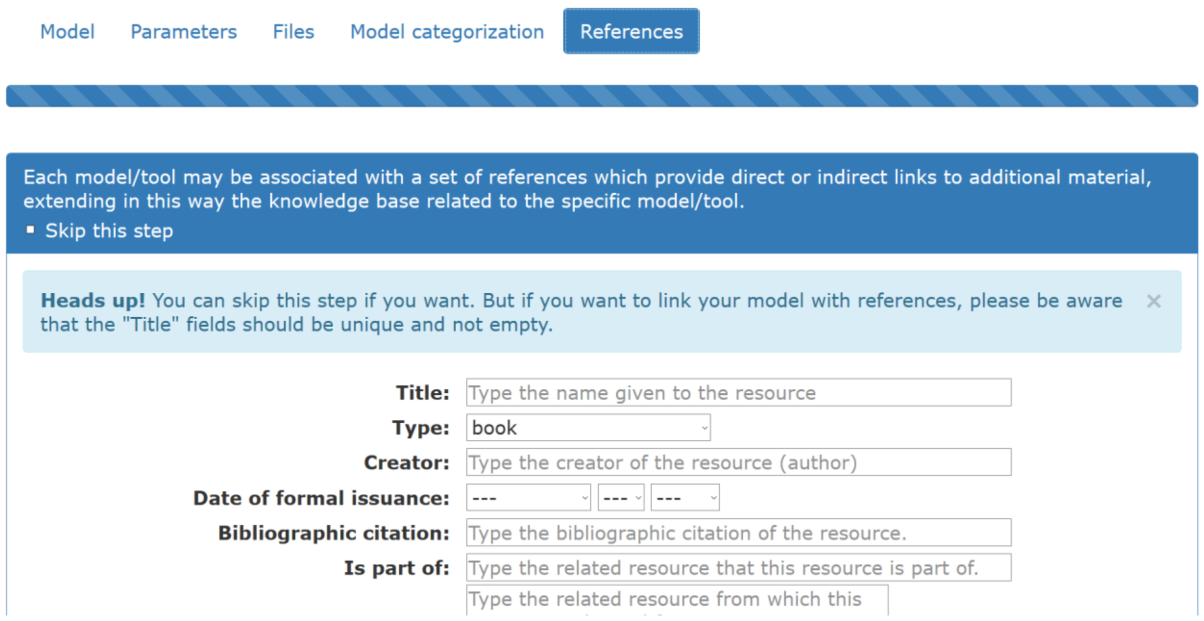
Perspective IX is about the order of addressing different temporal scales.

☐ I want right now to categorize my new model based on Perspective IX.

Perspective X is about the mechanistic - statistical character of the modelling approach.

☐ I want right now to categorize my new model based on Perspective X.

**Figure 10: The fourth step of the wizard. The user categorizes the new model based on the 13 Perspectives that have been defined within CHIC**



Each model/tool may be associated with a set of references which provide direct or indirect links to additional material, extending in this way the knowledge base related to the specific model/tool.

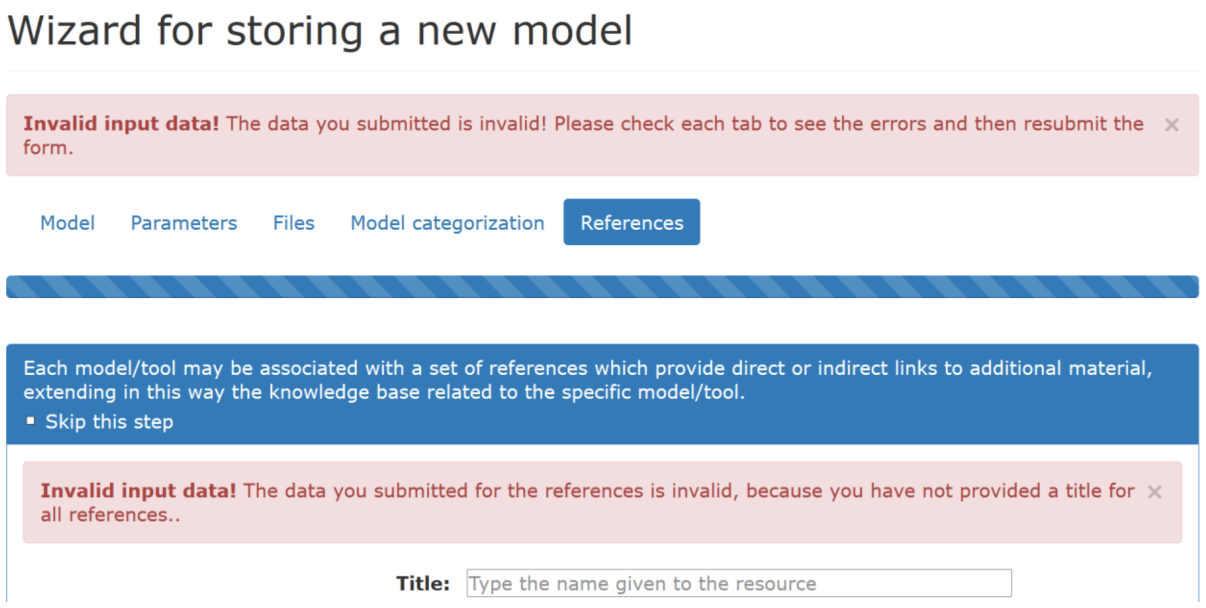
▪ Skip this step

**Heads up!** You can skip this step if you want. But if you want to link your model with references, please be aware that the "Title" fields should be unique and not empty. ×

| | |
|---|---|
| **Title:** | Type the name given to the resource |
| **Type:** | book |
| **Creator:** | Type the creator of the resource (author) |
| **Date of formal issuance:** | --- --- --- |
| **Bibliographic citation:** | Type the bibliographic citation of the resource. |
| **Is part of:** | Type the related resource that this resource is part of. |
| | Type the related resource from which this |

**Figure 11: The fifth step of the wizard: The user provides information about the references that are related to the new model (journal articles, conference proceedings, etc.)**

As shown in figures 7-11, all the information regarding the new model can be provided through a single page which consists of different tabs (one tab for each wizard step). After the provision of all the data of the new model, the corresponding information will be stored in the MySQL database of the Model Repository. The information related to the categorization of the new model will be stored both in the relational MySQL database and to the CHIC triplestore in the form of triples, as described in chapter 4.5. As shown in figure 12, in case of invalidity concerning the input data of the user, the Model Repository notifies the user accordingly with error messages in the corresponding tabs of the page.


Wizard for storing a new model

**Invalid input data!** The data you submitted is invalid! Please check each tab to see the errors and then resubmit the form. ×

Each model/tool may be associated with a set of references which provide direct or indirect links to additional material, extending in this way the knowledge base related to the specific model/tool.

▪ Skip this step

**Invalid input data!** The data you submitted for the references is invalid, because you have not provided a title for all references.. ×

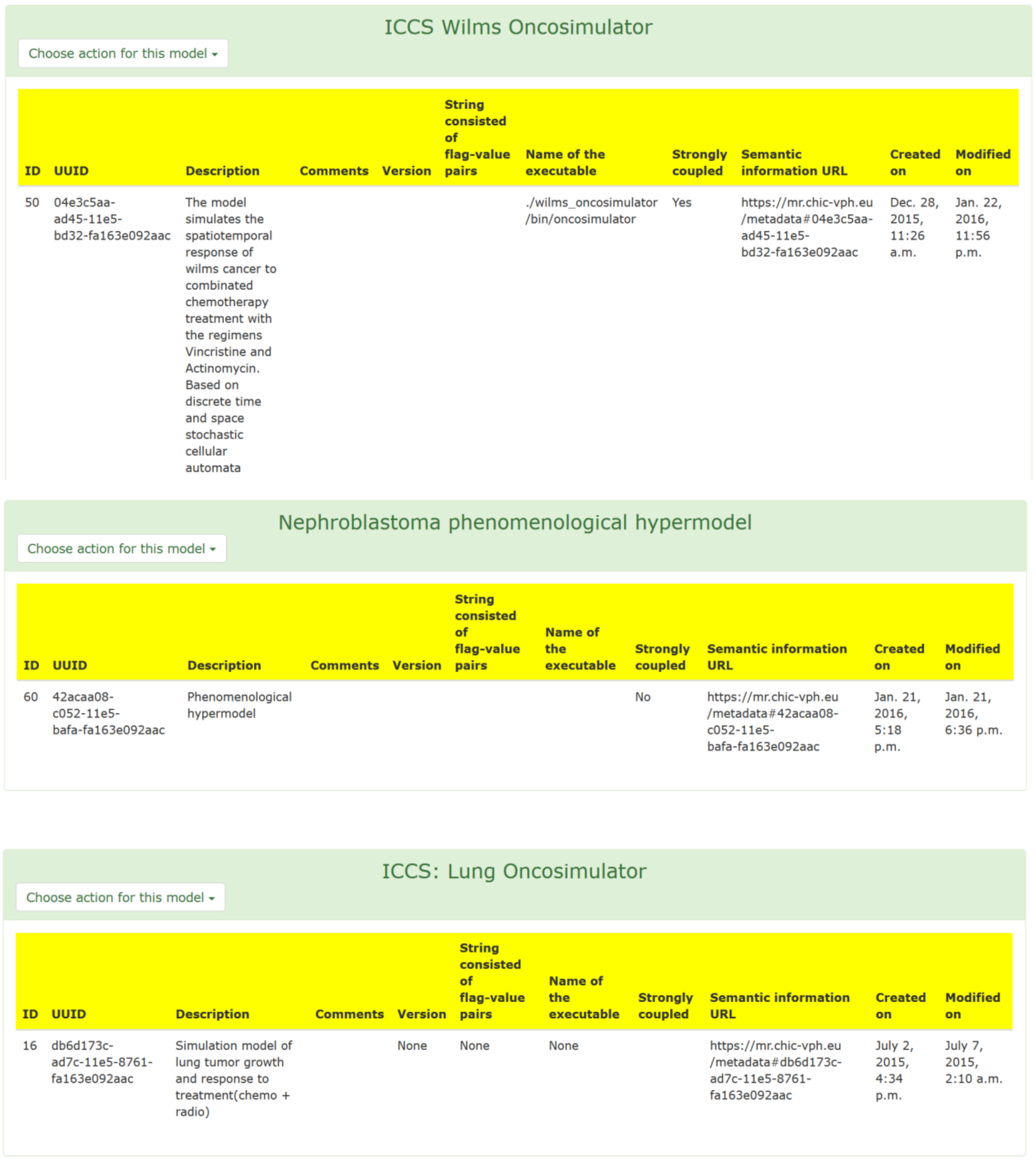
**Title:** Type the name given to the resource

**Figure 12: The wizard informs the user about the invalidity of the data when submitting the form**

### 4.3.2 Browsing and filtering the content of Model and Tool Repository

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 13. 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 13, the Model Repository displays the descriptive information of the models named Wilms Oncosimulator, Nephroblastoma phenomenological hypermodel and Lung Oncosimulator. Detailed information concerning the aforementioned models can be found in the deliverable D6.3 “Initial Standardized Cancer Hypermodels” [30].



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 combinated 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

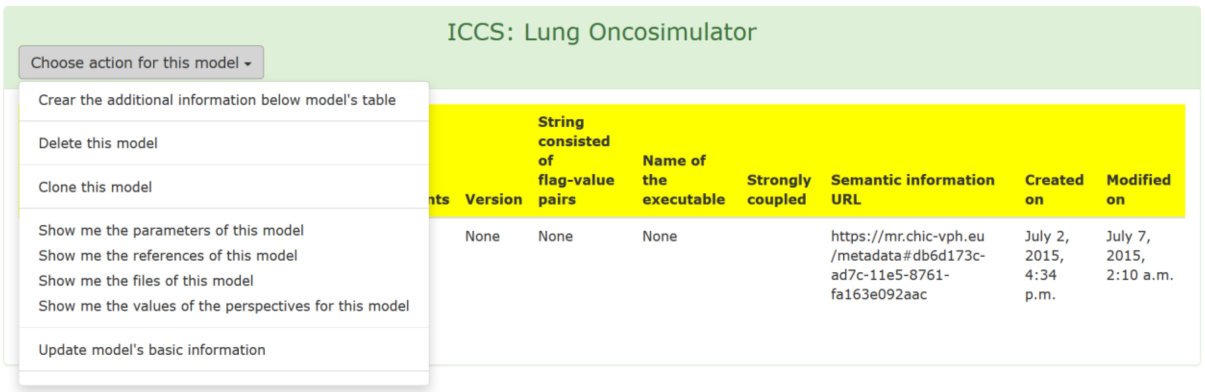
| ID | UUID | Description | Comments | Version | String consisted of flag-value pairs | Name of the executable | Strongly coupled | Semantic information URL | Created on | Modified on |
|---|---|---|---|---|---|---|---|---|---|---|
| 60 | 42acaa08-c052-11e5-bafa-fa163e092aac | Phenomenological hypermodel | | | | | 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. |

ICCS: Lung 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 |
|---|---|---|---|---|---|---|---|---|---|---|
| 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 13: 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 14 presents all the available actions that can be applied to a model, after pressing the button “Choose action for this model”, and figure 15 presents how information related to the parameters of the model named “Vasculature Model” can be rendered in the same screen after the corresponding user request. Just like the parameters of a model, different types of information (Perspective values, files, model basic description, etc.) may be displayed below the table of the model.

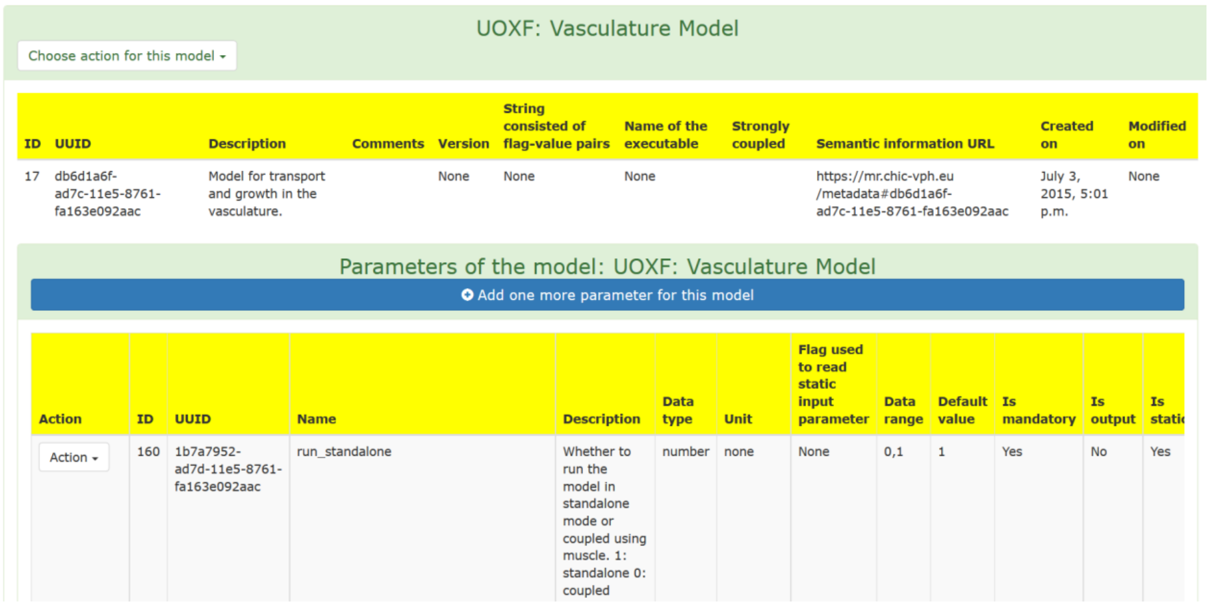


ICCS: Lung Oncosimulator

Choose action for this model
Crear the additional information below model's table
Delete this model
Clone this model
Show me the parameters of this model
Show me the references of this model
Show me the files of this model
Show me the values of the perspectives for this model
Update model's basic information

| nts | Version | String consisted of flag-value pairs | Name of the executable | Strongly coupled | Semantic information URL | Created on | Modified on |
|---|---|---|---|---|---|---|---|
| | 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 14: The user can apply many actions to a model (view the parameters, view the files, etc.)



UOXF: Vasculature Model

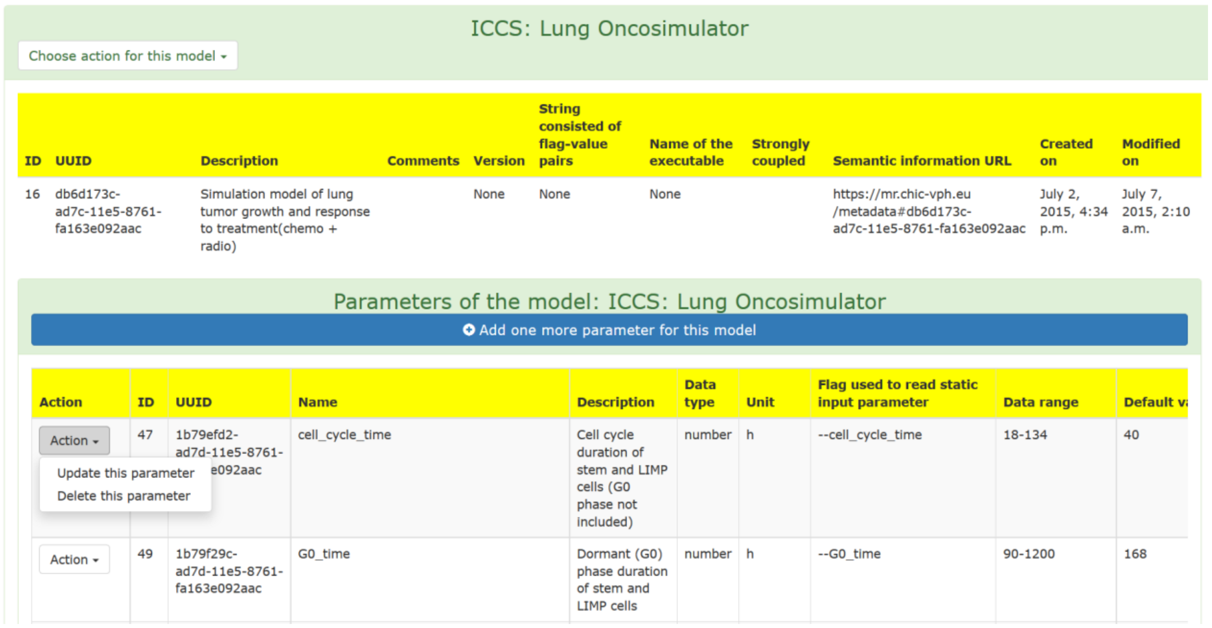
| ID | UUID | Description | Comments | Version | String consisted of flag-value pairs | Name of the executable | Strongly coupled | Semantic information URL | Created on | Modified on |
|---|---|---|---|---|---|---|---|---|---|---|
| 17 | db6d1a6f-ad7c-11e5-8761-fa163e092aac | Model for transport and growth in the vasculature. | | None | None | None | | https://mr.chic-vph.eu /metadata#db6d1a6f-ad7c-11e5-8761-fa163e092aac | July 3, 2015, 5:01 p.m. | None |

Parameters of the model: UOXF: Vasculature Model

| Action | ID | UUID | Name | Description | Data type | Unit | Flag used to read static input parameter | Data range | Default value | Is mandatory | Is output | Is static |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Action | 160 | 1b7a7952-ad7d-11e5-8761-fa163e092aac | run_standalone | Whether to run the model in standalone mode or coupled using muscle. 1: standalone 0: coupled | number | none | None | 0,1 | 1 | Yes | No | Yes |

Figure 15: The parameters of the model named "UOXFL: Vasculature Model" are being displayed below the model, after the corresponding user request

In addition to the depiction of the database content in a user-friendly way, the Model Repository also facilitates the updating process of the information related to models. This workflow is depicted in figures 16 and 17. More specifically, as shown in figure 16, the user is able to update a parameter of a model by pressing the button "Update this parameter", and as a result they will be redirected to the corresponding submission page, which is depicted in figure 17. Of course, the exact same procedure can be applied for the updating of the information related to Perspective values, references, files, etc.

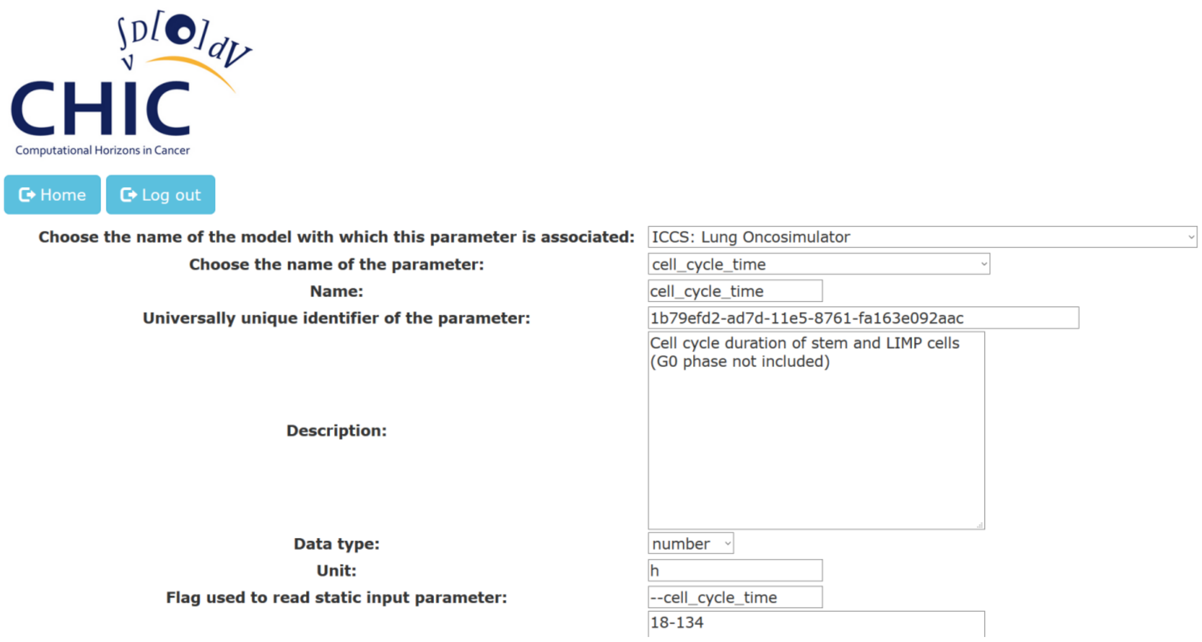

ICCS: Lung Oncosimulator

| ID | UUID | Description | Comments | Version | String consisted of flag-value pairs | Name of the executable | Strongly coupled | Semantic information URL | Created on | Modified on |
|---|---|---|---|---|---|---|---|---|---|---|
| 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. |

Parameters of the model: ICCS: Lung Oncosimulator

| Action | ID | UUID | Name | Description | Data type | Unit | Flag used to read static input parameter | Data range | Default va |
|---|---|---|---|---|---|---|---|---|---|
| Action ▾ (Update this parameter / Delete this parameter) | 47 | 1b79efd2-ad7d-11e5-8761-…e092aac | cell_cycle_time | Cell cycle duration of stem and LIMP cells (G0 phase not included) | number | h | --cell_cycle_time | 18-134 | 40 |
| Action ▾ | 49 | 1b79f29c-ad7d-11e5-8761-fa163e092aac | G0_time | Dormant (G0) phase duration of stem and LIMP cells | number | h | --G0_time | 90-1200 | 168 |

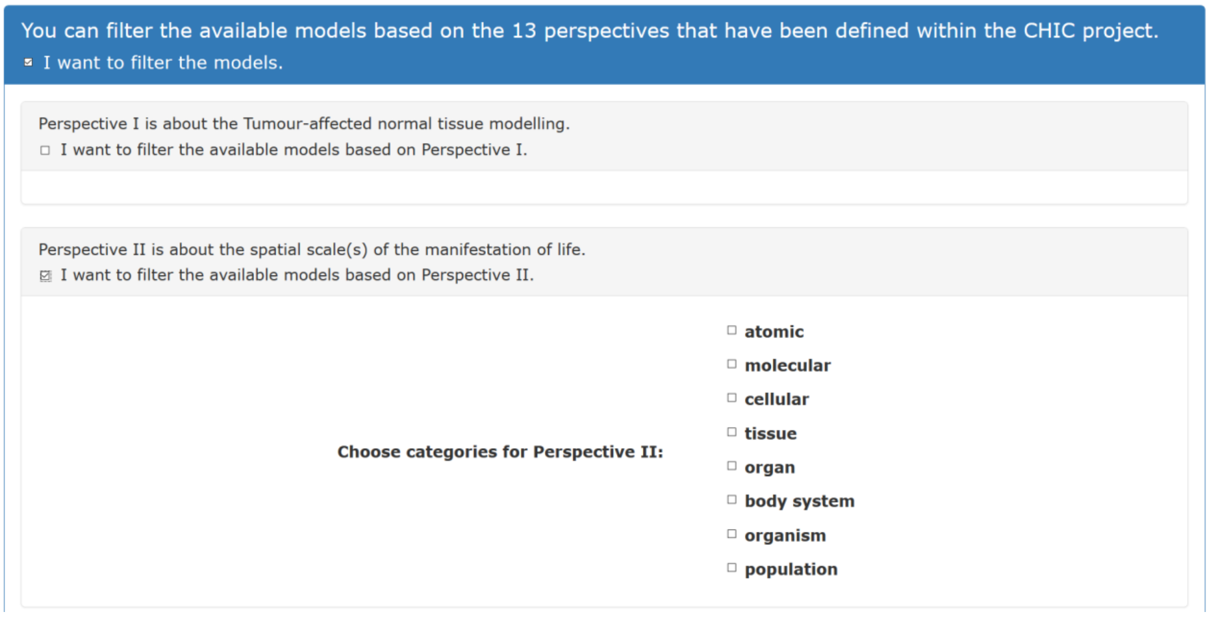
**Figure 16: The user is going to update the parameter "cell_cycle_time" which belongs to the model "Lung Oncosimulator"**



| | |
|---|---|
| Choose the name of the model with which this parameter is associated: | ICCS: Lung Oncosimulator |
| Choose the name of the parameter: | cell_cycle_time |
| Name: | cell_cycle_time |
| Universally unique identifier of the parameter: | 1b79efd2-ad7d-11e5-8761-fa163e092aac |
| Description: | Cell cycle duration of stem and LIMP cells (G0 phase not included) |
| Data type: | number |
| Unit: | h |
| Flag used to read static input parameter: | --cell_cycle_time |
| | 18-134 |

**Figure 17: The user is redirected to a submission page, in order to update the information related to the parameter "cell_cycle_time"**

Finally, since the number of the available models stored in the Model Repository may span from tens to thousands, the filtering of the models is a necessity. In respect to this, figure 18 presents a screenshot of the page related to the filtering of the models based on their categorization. As described in the deliverable “D6.2: CHIC Cancer Component Models: Initial Tested Versions”, the aforementioned categorization constitutes a metamodeling description of the corresponding model based on the CHIC 13 perspective approach. As shown in 18, the filtering of the models is based on the different categories selected by the user for each Perspective. For instance, the selection of the category “atomic” for Perspective II will result in displaying only the models whose spatial scale of the manifestation of life is of kind atomic.


You can filter the available models based on the 13 perspectives that have been defined within the CHIC project.
☑ I want to filter the models.

Perspective I is about the Tumour-affected normal tissue modelling.
☐ I want to filter the available models based on Perspective I.

Perspective II is about the spatial scale(s) of the manifestation of life.
☑ I want to filter the available models based on Perspective II.

Choose categories for Perspective II:
☐ atomic
☐ molecular
☐ cellular
☐ tissue
☐ organ
☐ body system
☐ organism
☐ population

**Figure 18: The user is able to filter the available models based on their categorization and their Perspective values**

## *4.4 Model and Tool Repository web services*

The Model and Tool Repository makes use of RESTful web services which are based on the entity relationship diagram depicted in figure 3 (chapter 3.2). The web services of the Model Repository are mainly 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 model/tool repository’s web services. 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. Even though the documentation of most of these web services has already been presented in “D8.3: Implementation of the interfaces of the CHIC Repositories”, it has been decided to also include them in this deliverable, since this document is considered to be the final report of the CHIC repositories. Moreover, this chapter incorporates all the changes and the updates that took place regarding the Model Repository web services, since the submission of the deliverable D8.3. The most significant change that took place is the replacement of external IP of the Model Repository (139.91.210.27) with the domain name “mr.chic-vph.eu” that has been reserved from partner Eurice. It must be noted that the progress of the CHIC project, the evolution of its individual components as well as the eventual overall expansion of the CHIC platform may pose new requirements for the creation of supplementary web services that will make the Model Repository more comprehensive.

**Model/Tool**

**The following web services (tables 2 - 7) should be used whenever the client needs to store, retrieve or delete descriptive information (title, description, comments) of the model/tool.**

**Table 2: 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 3: 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 | | |
| 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 | | |

**Table 4: 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 5: 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 6: Information for calling getToolByUuid web service**

| getToolByUuid | | 🔒 |
|---|---|---|
| Description | This method returns the descriptive information stored under the uuid (id, title, description, comment, version, semtype, strongly_coupled, executable_path, extra_parameters) 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) | uuid= | Required – uuid 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 getToolByUuid has thirteen keys named id, title, 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 7: 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> |

**Parameter**

**The following web services (tables 8-13) should be used whenever the client needs to store, retrieve or delete information related to parameters (name, description, data_type, data_range, etc.).**

Table 8: 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<br>• Discrete values example: value1,value2,value3<br>• Min value example: 3-<br>• Max value example: -10<br>• 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 – 1if 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 9: 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 10: 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 | | |
| The keys of the JSON object returned by method getParametersByToolId 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 mr_parameter entity (see figure 3) and each value of the nested JSON object represents the information of the corresponding column. | | |

| getParameterById | | 🔒 |
|---|---|---|
| Description | This method returns the descriptive information of the parameter stored under the given id (mr_parameter 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 | | |
| The Json object returned by method getParameterById has eighteen keys named tool, name, description, data_type, unit, flag, uuid, data_range, default_value, is_mandatory, is_output, is_static, comment, semtype, created_on, created_by, modified_on, modified_by and eighteen values associated with those keys. | | |

**Table 11: 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 |
|---|
| 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. |

**Table 12: 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 should be passed through the URL – query string parameter | tool_id= | Required – the id of the tool to 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 13: 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. | | |

**Property**

**The following web services (tables 14-20) 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 14: 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 15: 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 | | |
| 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. | | |

**Table 16: 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 17: 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 18: 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 19: 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 | | |
| The keys of the JSON object returned by method getPropertyValuesByToolId 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 name, description, comment, value, semtype. | | |

**Table 20: 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> |

**Reference**

**The following web services (tables 21-23) should be used whenever the client needs to store, retrieve or delete information related to references (reference title, reference authors, reference type, etc.).**

Table 21: 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 22: 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 23: 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 | | |
| The keys of the JSON object returned by method getReferencesByToolId 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 mr_reference entity (see figure 3) and each value of the nested JSON object represents the information of the corresponding column. | | |

**File**

**The following web services (tables 24-29) 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 24: 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 25: 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 26: 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 | 200 OK & attachment | |

| | | |
|---|---|---|
| (Content-Type: application/force-download Content-Disposition: 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 27: 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: 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 28: 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 | | |
| 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. |

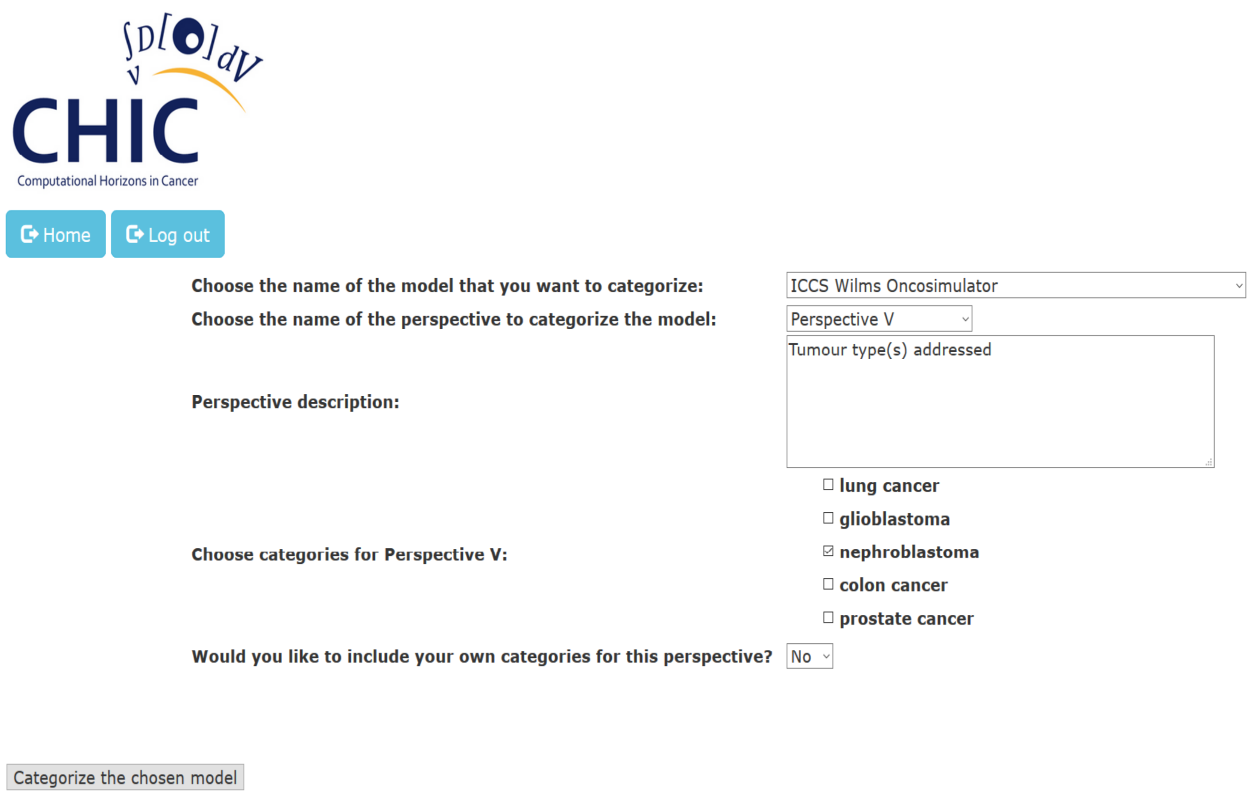
**Table 29: Information for calling getFilesByToolId web service**

| 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. | | |

## 4.5 Integration of Model and Tool Repository with CHIC triplestore

As discussed in chapter 4.2 "Architecture of Model and Tool Repository", the Model Repository makes use of a MySQL relational database for the persistent storage of information related to models. Even though there are many reasons for using a relational database, some of the meta-information related to models and tools will be converted to RDF triples so as to be stored in the CHIC triplestore. RDF triples can be applied equally to all structured, semi-structured and unstructured content. By defining new types and predicates, it is possible to create more expressive vocabularies within RDF in order to describe information related to models. This expressiveness enables RDF to define controlled vocabularies with exact semantics. Furthermore, triplestores have the ability to ingest diverse data, providing flexibility with respect to schema changes and mappings. They also allow for greater freedom, efficient handling of powerful queries and serving unforeseen information needs. Moreover, they employ intelligent data management solutions which combine full text search with graph analytics and logical reasoning to produce deep, rich results. The cost for data integration, management and query definition is much lower than other approaches. It must be also noted that these databases (also known as RDF, OWL, or Graph databases) are now widely used to manage unstructured and structured data in media and publishing, life sciences and financial services.

Consequently, the Model and Tool repository has been updated in order to be able to automatically store into the CHIC RDF triplestore, information related to the categorization of the models. As stated in the Deliverable 6.1 "Cancer hypomodelling and hypermodelling strategies and initial component models", mathematical and computational cancer models can be categorized depending on the perspective from which they are viewed in the basic science context. The definition of the thirteen perspectives and their indicative values is included in the aforementioned deliverable. Consequently, the Model and Tool repository and the CHIC semantics infrastructure make use of a common RDF mapping configuration file so as to produce a model (a set of RDF triples) based on the already locally stored relational data. The aforementioned configuration file maps some of the Model Repository's database tables and columns to CHIC RDF vocabularies and OWL ontologies. This mapping defines the virtual RDF graph that contains some of the information from the Model and Tool repository's MySQL database which is related to the categorization of the models. With this kind of integration between the Model Repository and the CHIC triplestore, the user is able to categorize their model by visiting only a single CHIC component. After the submission of the user's data, it is the Model Repository's responsibility to store the information related to the categorization of the model both to the repository's relational database and to the CHIC triplestore. The page where the user categorizes their model is shown in figure 19.



| | |
|---|---|
| Choose the name of the model that you want to categorize: | ICCS Wilms Oncosimulator |
| Choose the name of the perspective to categorize the model: | Perspective V |
| Perspective description: | Tumour type(s) addressed |
| Choose categories for Perspective V: | ☐ lung cancer<br>☐ glioblastoma<br>☑ nephroblastoma<br>☐ colon cancer<br>☐ prostate cancer |
| Would you like to include your own categories for this perspective? | No |

**Figure 19: The web page where the user categorizes their model**

As shown in figure 19, the user categorizes the model "ICCS Wilms Oncosimulator" for perspective V, named "Tumour type(s) addressed". Since the "ICCS Wilms Oncosimulator" is an integrated cancer treatment support system modelling the growth of nephroblastoma tumours, the user checks the box "nephroblastoma". After pushing the button "Categorize the chosen model", all the corresponding information of this categorization is going to be stored both in the Model Repository and the CHIC semantics infrastructure. The topology of the CHIC components that handle the semantic annotation of the categorization of the models based on the 13 perspectives that have been defined within CHIC, is shown in figure 20.

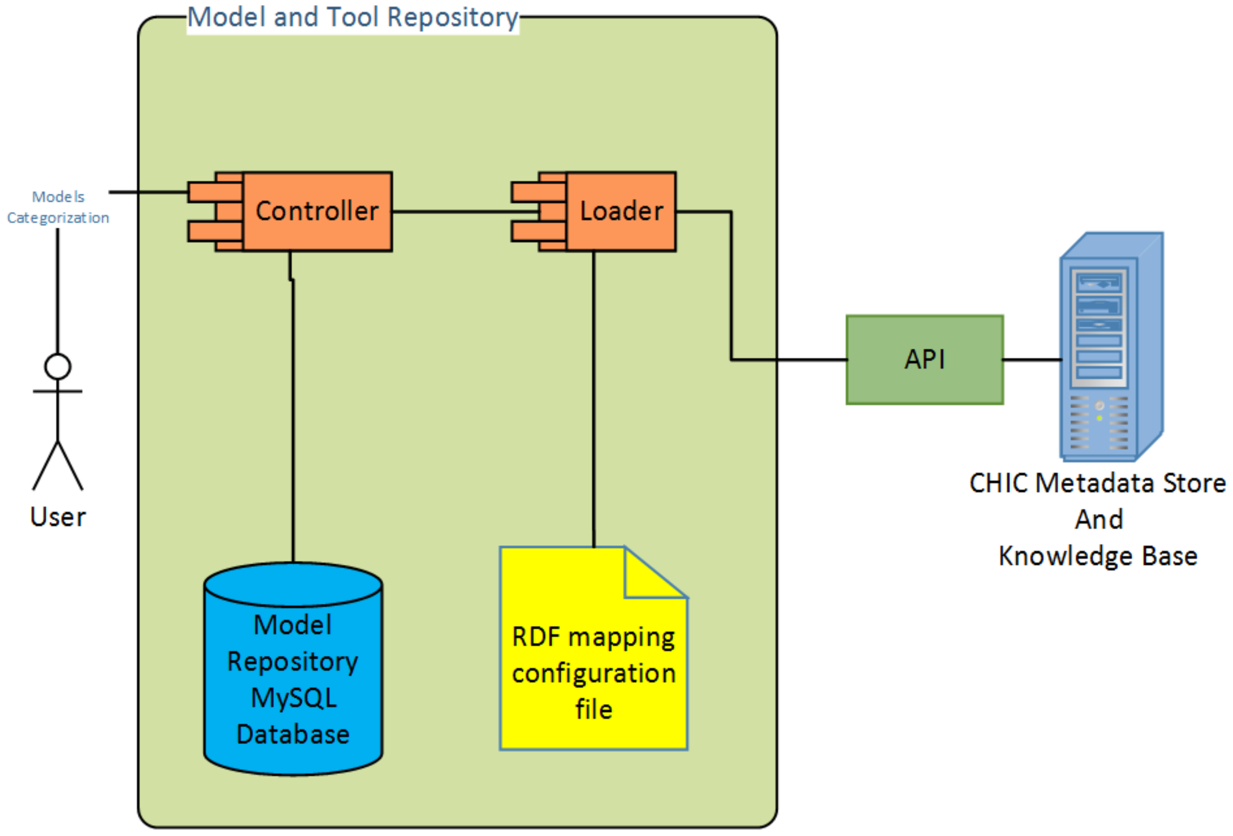

Model and Tool Repository
Models Categorization
Controller
Loader
User
API
CHIC Metadata Store And Knowledge Base
Model Repository MySQL Database
RDF mapping configuration file

**Figure 20: Topology of the CHIC components that handle the semantic annotation of the models**

As shown in figure 20, the following modules are used for the use case of the semantic annotation of the models categorization:

- **Controller:** The controller is the central module of the model repository that consists of many other submodules. It opens the local relational database connection and it handles web requests and presentation details that the user will see. It also calls the Loader module.
- **Loader:** The Loader is in charge of converting MySQL data into RDF property values that will be provided to the CHIC semantics infrastructure web services. It also loads the RDF mapping configuration file and calls the application programming interfaces of the CHIC metadata store.
- **RDF mapping configuration file:** This file includes the necessary information for mapping MySQL table and columns of the CHIC model repository to RDF properties, vocabularies and OWL ontologies of the CHIC metadata store.
- **API:** This module consists of all the web annotation services that are exposed from the CHIC metadata store and are being used, among others, for the semantic annotation of the models' categorization.

Table 30 presents the result of the semantic annotation of the categorization of the model named "ICCS Wilms Oncosimulator" for perspective V, in the form of subject-predicate-object expressions. The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. The RDF statements that are included in table 30 represent the following knowledge base:

- The CHIC resource with the URI https://mr.chic-vph.eu/metadata#04e3c5aa-ad45-11e5-bd32-fa163e092aac, represents a CHIC hypomodel.
- The aforementoned CHIC hypomodel has the name “ICCS Wilms Oncosimulator”
- The aforementioned CHIC hypomodel has the unique identifier "04e3c5aa-ad45-11e5-bd32-fa163e092aac"
- The aforementioned CHIC hypomodel addresses the tumour type named “Nephroblastoma”. As stated in the fifth row and third column of Table 30, the “Nephroblastoma” term has the URI “http://purl.obolibrary.org/obo/HP_0002667“ which has been derived from the human phenotype ontology.

**Table 30: The RDF statements that represent the semantic annotation of the categorization of the model named ”ICCS Wilms Oncosimulator” for perspective V**

| Subject | Predicate | Object |
|---|---|---|
| <https://mr.chic-vph.eu/metadata#04e3c5aa-ad45-11e5-bd32-fa163e092aac> | <http://www.chic-vph.eu/ontologies/resource#hasCHICuuid> | "04e3c5aa-ad45-11e5-bd32-fa163e092aac" |
| <https://mr.chic-vph.eu/metadata#04e3c5aa-ad45-11e5-bd32-fa163e092aac> | <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> | <http://www.chic-vph.eu/ontologies/resource#Model-ChicHypomodel> |
| <https://mr.chic-vph.eu/metadata#04e3c5aa-ad45-11e5-bd32-fa163e092aac> | <http://www.chic-vph.eu/ontologies/resource#hasName> | "ICCS Wilms Oncosimulator" |
| <https://mr.chic-vph.eu/metadata#04e3c5aa-ad45-11e5-bd32-fa163e092aac> | <http://www.chic-vph.eu/ontologies/resource#hasPositionIn-5> | <http://purl.obolibrary.org/obo/HP_0002667> |

The semantic knowledge base that has been described in table 30, has been produced from the Model Repository and has been stored in the CHIC triplestore in order for the other CHIC client components, like the editor or the CRAF (Clinical Research Application Framework), to be able to recognize the model named “ICCS: Wilms Oncosimulator” as a model that simulates the growth of nephroblastoma tumour, and more specifically, according to the human phenotype ontology, the growth of the neoplasm of the kidney that primarily affects children.

The same procedure can be applied for the semantic annotation of the categorization of any new model, through the model repository. Nonetheless, since new CHIC RDF vocabularies and OWL ontologies may be incorporated in the future in order to represent new perspective values, the upgrade and the maintenance of the Model Repository and the CHIC semantics infrastructure is essential in order to ensure the correct mapping between the Model Repository's relational database and the CHIC RDF schema.

## *4.6 The Model Repository publishes events to other CHIC components through AMQP Protocol*

The Model Repository web services that have been presented in chapter 4.4 can be invoked over the HTTP protocol by the other CHIC components in order for the Repository content to be accessible by them. But the aforementioned web services are not the only way of interaction between the Model Repository and the other CHIC components. Apart from waiting for client requests, the Model Repository publishes events to the other CHIC components whenever its content changes. Since the modification of the Model Repository database may have a huge impact on the workflows and the processes of the other CHIC components (Hypermodelling Editor, CRAF, VPH-HF), it has been decided to always publish events related to Model Repository database changes.

For this communication, the Model Repository makes use of "Pika", which is a pure-Python implementation of the AMQP protocol (Advanced Message Queuing Protocol). The AMQP protocol developed by the oasis open standards consortium, provides a platform-agnostic method for ensuring information is safely transported between applications, among organizations, within mobile infrastructures, and across the Cloud. AMQP is used in areas as varied as financial, front-office trading, ocean observation, transportation, smart-grid, computer-generated animation and online gaming. Many operating systems include AMQP implementations, and many application frameworks are AMQP-aware. AMQP can also be embedded in virtualization infrastructure [14].

In order for the CHIC components to asynchronously connect to each other through the AMQP protocol, CHIC makes use of RabbitMQ. With RabbitMQ, messages in CHIC are routed through exchanges before arriving at queues. In order for the Model Repository to interact with RabbitMQ, Pika library provides a wrapper which implements the methods and behaviors for an AMQP Channel. After the construction of the channel, the Model Repository publishes to the channel with the given "exchange", "routing_key" and "body". The "exchange" should specify the name of the exchange that the message was originally published to, the routing_key is used for routing messages depending on the exchange configuration and finally, the "body" should specify the body of the message. Table 31 presents the "exchange","routing_key" and "body" values for every event that is published whenever the content of the Model Repository changes.

**Table 31: Exchange, Routing Key and Body values for events published by the Model Repository**

| Repository Change | Exchange Value | Rouring Key | Body |
|---|---|---|---|
| Addition of new model/tool | “mr” | “models.new” | uuid of the new model/tool |
| Addition/change/deletion of model/tool parameter | “mr” | “models.changed” | uuid of the model to which the parameter belongs |
| Addition/change/deletion of model/tool reference | “mr” | “models.changed” | uuid of the model to which the reference is linked |
| Addition/deletion of model/tool file | “mr” | “models.changed” | uuid of the model to which the file belongs |
| Addition/change/deletion of model perspective value | “mr” | “models.changed” | uuid of the model with which the given perspective is associated |
| Deletion of model/tool | “mr” | “models.deleted” | uuid of the model that has been deleted |

# 5 The Clinical data Repository

## 5.1 Introduction

The clinical data repository will permanently host all the medical data produced or collected by the CHIC project. The data provided by the clinical environment will pass through de-identification and (pseudo)-anonymization processes, as described in the following chapter. Additionally, interfaces that will allow to import and export the contents of the clinical data repository will be developed. 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 export services that will be created will also assist in this direction, as many of the data sets to be gathered by the CHIC project will be reusable by future projects. The clinical data repository will contain for each patient all the relevant medical data including imaging data, clinical data, histological data and genetic data.

## 5.2 Data flow and interaction

The intention of the data flow described (Figure 21) 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 (hyper)models and the 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 21.

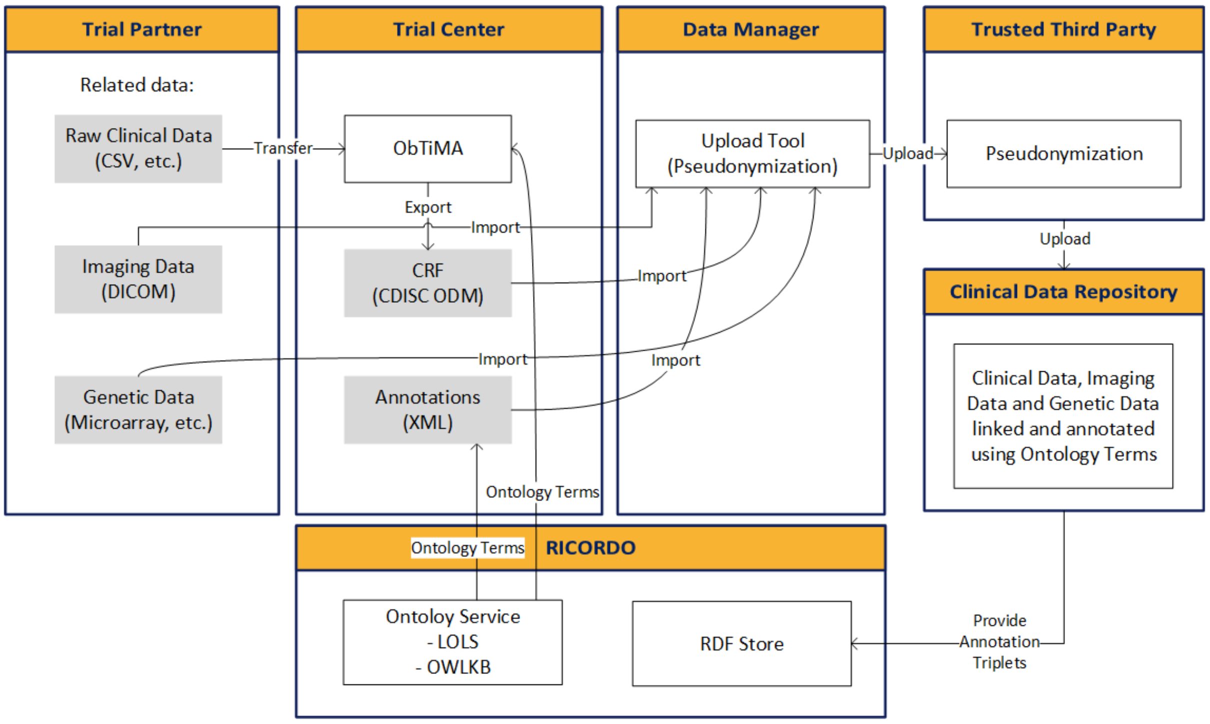


**Figure 21: The general workflow for data upload**

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 | RICRODO provides services to search ontology terms, to store annotation triples, to conduct semantically driven search queries and to perform automated semantic reasoning. |

## *5.3 Data types and standards*

In order to provide efficient anonymization/pseudonymization of the data, standard file format should be used. This standardization is also important to enable proper extraction of the metadata information from the files by the clinical data repository and storage of this information in searchable tables. In deliverable “D8.1 – Design of the CHIC repositories” most data types were already briefly described. In Table 33 the various types of clinical data are listed in combination with the standard file format used during the general workflow for data upload. The agreement on the file format used for data exchange clinical information will simplify the development process for all involved parties.

**Table 33: Data types and standards**

| Data type | Standard |
|---|---|
| Clinical data (pathological and also outcome) | CDISC ODM |
| Imaging data (post-processed, segmented, etc.) | DICOM, MetaImage, Nifti, Analyze |

| | |
|---|---|
| Genetic / Molecular data | MINiML |

For better understanding it is also important to know, where the data comes from. In Table 34 the different sources of the data types are listed. This insight is especially useful for parties not involved in the workflow prior to the data upload.

**Table 34: The different sources of the data types**

| Data type | Source |
|---|---|
| Clinical data | ObTiMA (ontology-based clinical trial management system) |
| Imaging data | Local PACS (Picture Archiving and Communication System) |
| Genetic / Molecular data | Platform specific (e.g. Affymetrix) files as generated by the appropriate equipment. |

## 5.4 *General Concepts*

The clinical data repository is built around the concept of data objects (*ObjectVersion*), which constitute the basic component of the system. These data objects can be any type of image file, processed data, study data etc. This approach provides a large flexibility to the system in terms of data formats, data organization and data exchange [5].

The system has been designed to support versioning. Data uploaded to the system are never deleted, but multiple versions of an object can be stored in the database. This approach limits problems associated with accidental deletion of data, while maintaining the flexibility to keep updating data files. For example, the initial data of the clinical study concerning a patient can be uploaded before the final examinations. Once the last examination has been performed, a new version of the file is uploaded to the system, which enables modellers to have access to the latest information while keeping the ability to see the history of the modifications.

### 5.4.1 Linking

Each new dataset can be linked with any object already present in the repository. For example anatomical structures can be segmented out of one or multiple medical images. Linking mechanisms ensure that an uploaded segmentation file is not only associated with the correct patient's data, but also that the original images used to perform the segmentation task can be identified by the users of the system. In the case of multimodal image segmentation, this implies that multiple links are created to relate the segmentation file with each of the multi-modal original images. If available, the system makes use of the meta-information stored in the files to automatically generate this linking. Manual linking is supported as well.

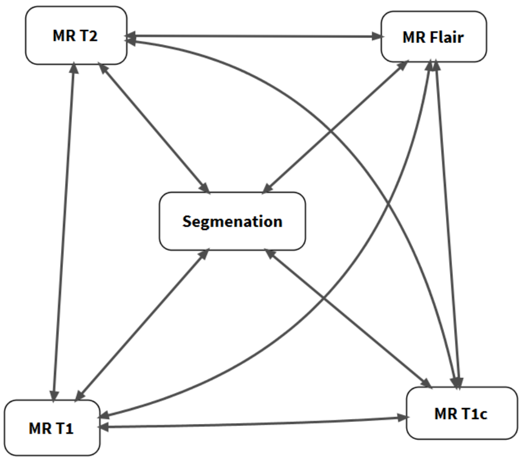


**Figure 22: Example of the linking to relate data objects in the clinical data repository for a multi-modal brain segmentation. In this case, four different MRI image datasets are used for the segmentation of brain tumours**

### 5.4.2 Annotation and Search

In addition to the imaging and clinical data, each data object can be annotated with multiple ontology terms. Initial investigations have been made to integrate an anatomical ontology; the Foundational Model of Anatomy (FMA) [6]. The FMA is a symbolic representation of the canonical, phenotypic structure of an organism; a spatial-structural ontology of anatomical entities and relations which form the physical organization of an organism at all salient levels of granularity. The ontology relies on a triplestore storage system and not in relations or tables. Therefore, a separate system is used to store the semantic information. Web based queries based on SPARQL [7] are used to retrieve the information from the ontology for annotation and semantic search. The approach is very flexible and allows to easily include multiple ontologies. In addition to the FMA, additional ontologies can be included with the help of the RICORDO system. Based on these annotations it is possible to conduct semantically driven search queries to find datasets containing the required anatomical structures or other properties. A detailed description of the semantic annotation is provided in section 5.9 of this deliverable.

### 5.4.3 Validation and Versioning

To ensure a high level of quality to the data stored in the repository, the system supports a multi-step validation process. During the validation process the user can review the metadata extracted from the data, include additional relevant information and finally publish the data object. Once published, the new data object is accessible by the other users of the system having the appropriate permissions.

The objects stored in the database cannot be changed or modified once the validation step has been completed. However, the modifications to the objects are shown as a new version of the data. Versioning makes use of a *GenericObject*; a new version of a dataset will have the same *GenericObject* as the ancestor. An example where a new version of the object will be generated corresponds to the modification in the definition of a clinical study. In case new questions or time points have been added to the study, a new version of the file will be generated. The clinical study will have the same unique identifier as the original object, and based on this identifier a new version can be created. With this approach, the users will still have access to the original data if needed by their models, but the new version will be shown as the current version of the data.

### 5.4.4 Data Organization

The system also allows the user to freely organize the data that they are using. To this aim, the user interface allows the creation of virtual folders to organize the datasets. This functionality provides the user with the flexibility to freely create and organize their personal workspace. Hereby, data objects are not physically moved or duplicated, but the system creates a reference to the data object, retaining the original file permission and ownership. For collaborations or other purposes, the user can share parts of his workspace by changing the permission of his folder accordingly. To simplify the collaboration within a group, the system provides a default shared group folder, which is accessible and manageable by all members of the group.

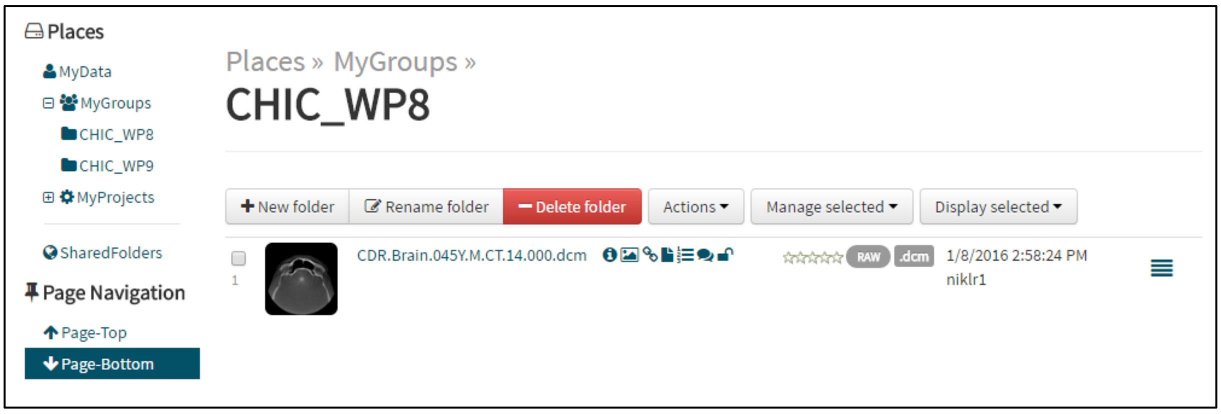


**Figure 23: The clinical data repository allows each user to freely organize the data into his desired folder structure for easy access to the data needed for his research. The structure created by one user can be directly shared to other co-workers. Modifications made by one user are immediately visible in the folder of the other collaborators. The mechanism should allow efficient collaboration between modellers working on the same tumour model.**

## *5.5 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.

### 5.5.1 Data Model

The clinical data repository makes use of the updated audit data model called XDASv2 [8] introduced in deliverable D5.2. An implementation in C# of the audit data model has been published as open-source on the GitHub platform under the MIT license [9].

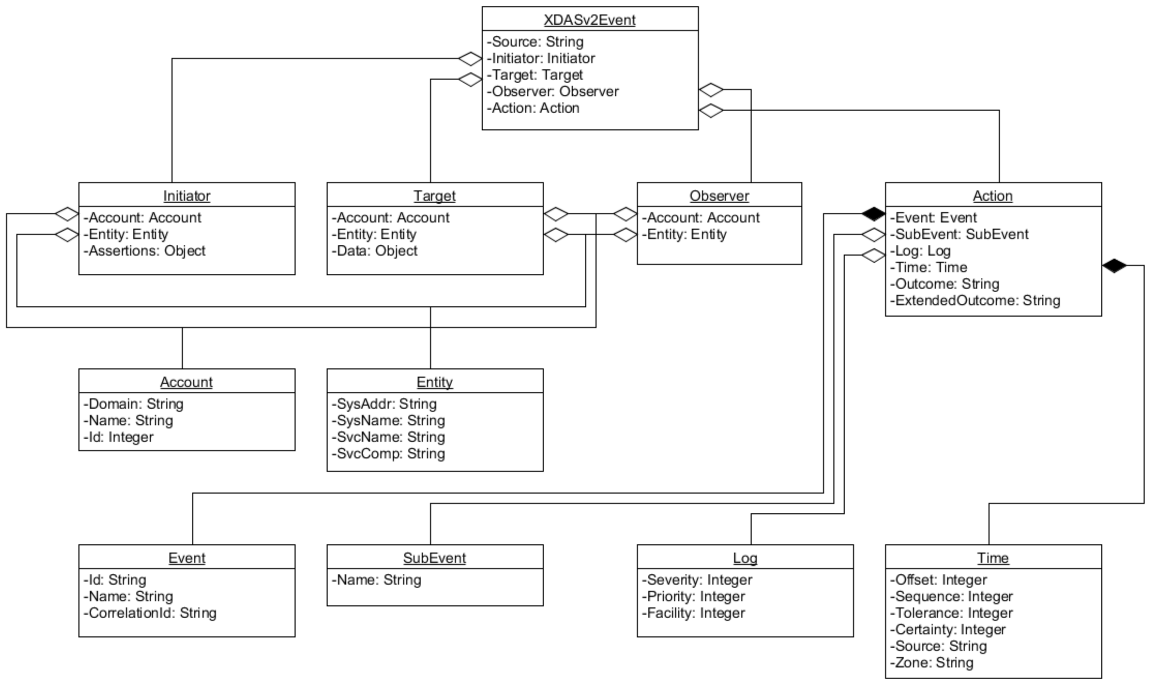


**Figure 24: The audit data model XDASv2 used by the clinical data repository for auditing.**

### 5.5.2 Architecture

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 25, 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.

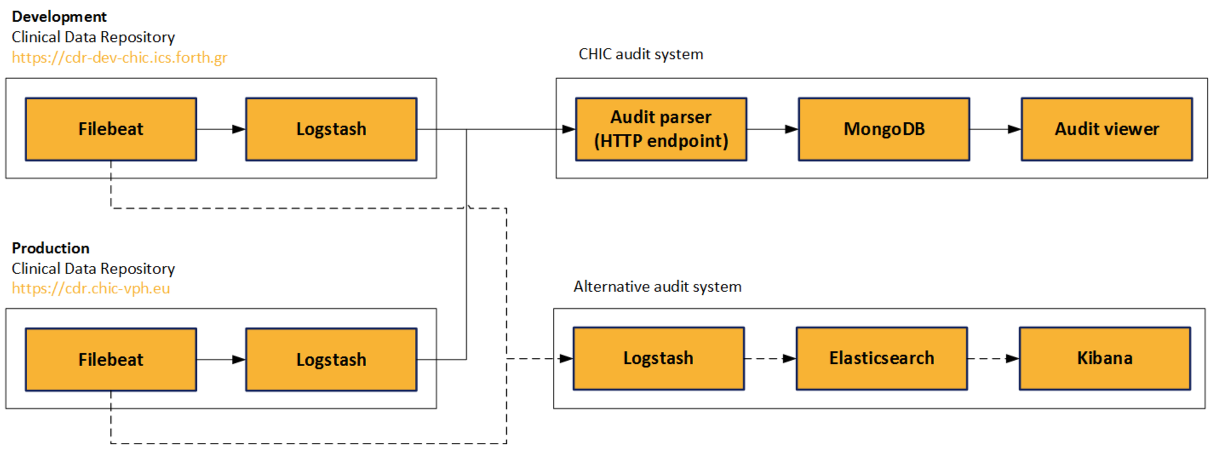


**Figure 25: The components of the audit systems and the interactions with the clinical data repository**

**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.

### 5.5.3 Setup on the CDR

The website and API of the clinical data repository can be configured independently. For this purpose the log4net.config files located in the respective bin folder illustrated in Figure 26 can be modified according to given requirements and conditions. It is possible to define the location of the log file which will be processed by Filebeat. Furthermore, the configuration file offers the possibility to specify the maximum file size and the amount of file backups before starting rotating the log files.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />
  </configSections>
  <log4net>
    <appender name="XDASv2RollingFileAppender" type="log4net.Appender.RollingFileAppender">
      <threshold value="XDASv2" />
      <file value="C:\www\VSD\Data\Logs\Audit\xdas.txt" />
      <appendToFile value="true" />
      <rollingStyle value="Size" />
      <maxSizeRollBackups value="10" />
      <maximumFileSize value="250KB" />
      <staticLogFileName value="true" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%message%newline" />
      </layout>
    </appender>
    <level>
      <name value="XDASv2" />
      <value value="50000" />
    </level>
    <root>
      <level value="ALL" />
      <appender-ref ref="XDASv2RollingFileAppender" />
    </root>
  </log4net>
</configuration>
```

**Figure 26: The log4net.config file to configure the clinical data repository auditing**

Filebeat can be downloaded from the official website of elastic [10]. After extracting the content to a permanent location on the filesystem, Filebeat needs to be configured and installed as a Windows service. Filebeat is configured with the help of the filebeat.yml file partly listed in Figure 27. The PowerShell script to install Filebeat as a Windows service is listed in Figure 28.

```
############################# Filebeat ######################################
filebeat:
  # List of prospectors to fetch data.
  prospectors:
    # Each - is a prospector. Below are the prospector specific configurations
    -
      paths:
        - C:\www\VSD\Data\Logs\Audit\*
      document_type: XDASv2
      force_close_files: true
############################# Output ##########################################
# Configure what outputs to use when sending the data collected by the beat.
# Multiple outputs may be used.
output:
  logstash:
    hosts: ["localhost:5044"]
```

**Figure 27: An excerpt of the file filebeat.yml to configure Filebeat on the clinical data repository**

```
# delete service if it already exists
if (Get-Service filebeat_audit_xdasv2 -ErrorAction SilentlyContinue) {
  $service = Get-WmiObject -Class Win32_Service -Filter "name='filebeat_audit_xdasv2'"
  $service.StopService()
  Start-Sleep -s 1
  $service.delete()
}

$workdir = Split-Path $MyInvocation.MyCommand.Path

# create new service
New-Service -name filebeat_audit_xdasv2 `
  -displayName filebeat_audit_xdasv2 `
  -binaryPathName "`"$workdir\filebeat.exe`" -c `"$workdir\filebeat.yml`""
```

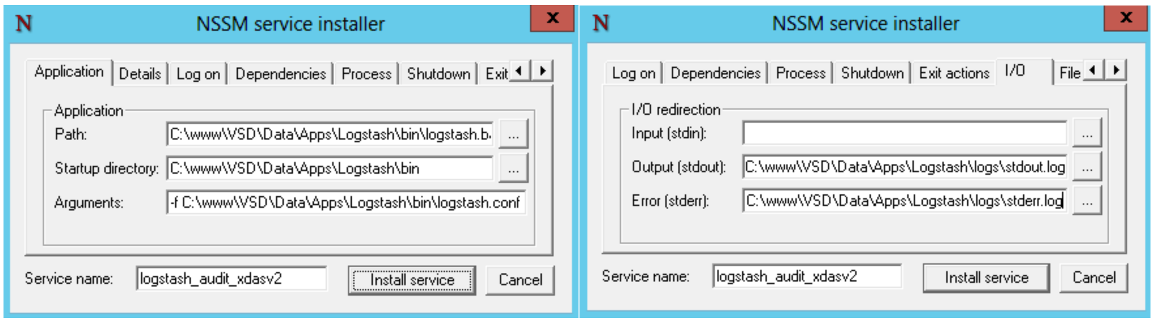
**Figure 28: The PowerShell script to install Filebeat as a Windows service on the clinical data repository**

Logstash can be downloaded from the official website of elastic [10]. After extracting the content to a permanent location on the filesystem, Logstash needs to be configured and installed as a Windows service. Logstash is configured with the help of the logstash.conf file listed in Figure 29.

```
input {
  beats {
    port => 5044
    ssl => false
    codec => "json"
  }
}
filter {
  if [type] == "XDASv2" {
    json {
      source => "message"
    }
  }
}
output {
  http {
    http_method => "post"
    url => "..."
  }
}
```

**Figure 29: The logstash.conf file to configure Logstash on the clinical data repository**

In order to install Logstash as a Windows service an additional tool called NSSM is needed. NSSM is a service helper which handles failure of the application running as a service. It monitors the running service and will restart it if it dies. NSSM can be configured to absolve all responsibility for restarting an application and let Windows take care of recovery actions. The progress is listed in the system Event Log to help understand why an application isn't behaving as it should.

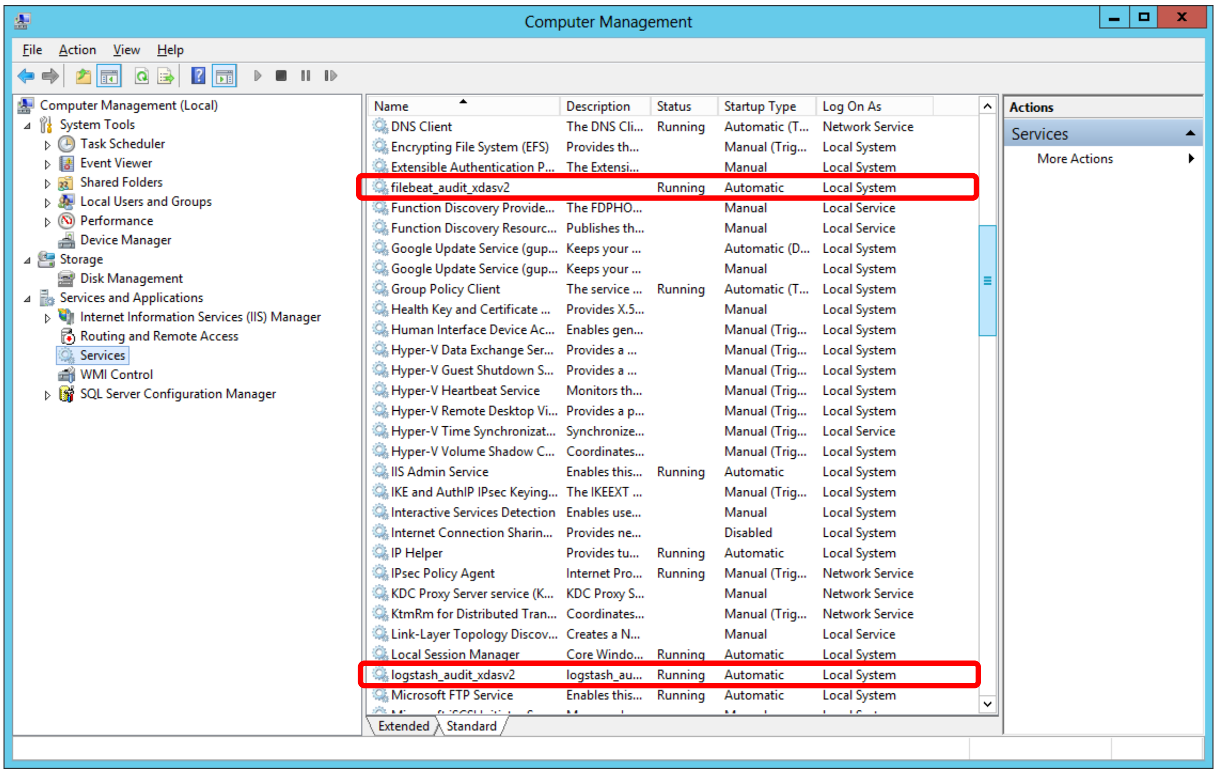

NSSM service installer
Application | Details | Log on | Dependencies | Process | Shutdown | Exit
Log on | Dependencies | Process | Shutdown | Exit actions | I/O | File
Install service
Cancel

Application
Path: C:\www\VSD\Data\Apps\Logstash\bin\logstash.b
Startup directory: C:\www\VSD\Data\Apps\Logstash\bin
Arguments: -f C:\www\VSD\Data\Apps\Logstash\bin\logstash.conf
Service name: logstash_audit_xdasv2

I/O redirection
Input (stdin):
Output (stdout): C:\www\VSD\Data\Apps\Logstash\logs\stdout.log
Error (stderr): C:\www\VSD\Data\Apps\Logstash\logs\stderr.log
Service name: logstash_audit_xdasv2

**Figure 30: Installing Logstash as a Windows service on the clinical data repository using NSSM**

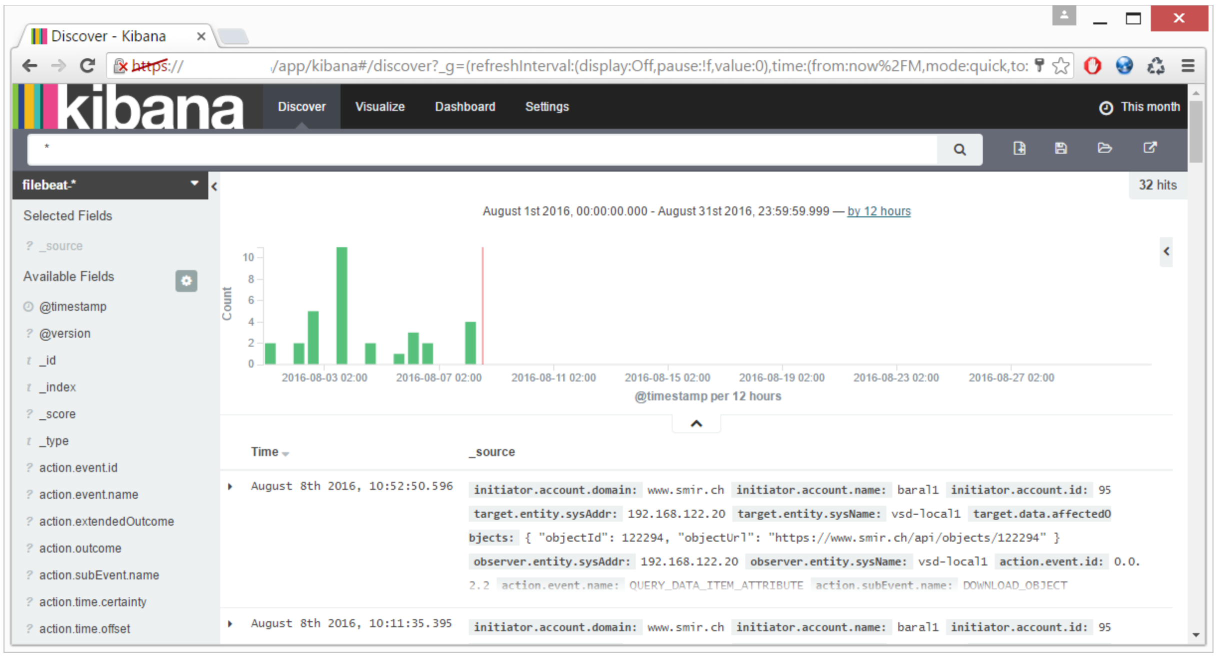
Both services Filebeat and Logstash should now appear in the Services tab of the Computer Management tool as illustrated in Figure 31.



| Name | Description | Status | Startup Type | Log On As |
|---|---|---|---|---|
| DNS Client | The DNS Cli... | Running | Automatic (T... | Network Service |
| Encrypting File System (EFS) | Provides th... | | Manual (Trig... | Local System |
| Extensible Authentication P... | The Extensi... | | Manual | Local System |
| filebeat_audit_xdasv2 | | Running | Automatic | Local System |
| Function Discovery Provide... | The FDPHO... | | Manual | Local Service |
| Function Discovery Resourc... | Publishes th... | | Manual | Local Service |
| Google Update Service (gup... | Keeps your ... | | Automatic (D... | Local System |
| Google Update Service (gup... | Keeps your ... | | Manual | Local System |
| Group Policy Client | The service ... | Running | Automatic (T... | Local System |
| Health Key and Certificate ... | Provides X.5... | | Manual | Local System |
| Human Interface Device Ac... | Enables gen... | | Manual (Trig... | Local System |
| Hyper-V Data Exchange Ser... | Provides a ... | | Manual (Trig... | Local System |
| Hyper-V Guest Shutdown S... | Provides a ... | | Manual (Trig... | Local System |
| Hyper-V Heartbeat Service | Monitors th... | | Manual (Trig... | Local System |
| Hyper-V Remote Desktop Vi... | Provides a p... | | Manual (Trig... | Local System |
| Hyper-V Time Synchronizat... | Synchronize... | | Manual (Trig... | Local Service |
| Hyper-V Volume Shadow C... | Coordinates... | | Manual (Trig... | Local System |
| IIS Admin Service | Enables this... | Running | Automatic | Local System |
| IKE and AuthIP IPsec Keying... | The IKEEXT ... | | Manual (Trig... | Local System |
| Interactive Services Detection | Enables use... | | Manual | Local System |
| Internet Connection Sharin... | Provides ne... | | Disabled | Local System |
| IP Helper | Provides tu... | Running | Automatic | Local System |
| IPsec Policy Agent | Internet Pro... | Running | Manual (Trig... | Network Service |
| KDC Proxy Server service (K... | KDC Proxy S... | | Manual | Network Service |
| KtmRm for Distributed Tran... | Coordinates... | | Manual (Trig... | Network Service |
| Link-Layer Topology Discov... | Creates a N... | | Manual | Local Service |
| Local Session Manager | Core Windo... | Running | Automatic | Local System |
| logstash_audit_xdasv2 | logstash_au... | Running | Automatic | Local System |
| Microsoft FTP Service | Enables this... | Running | Automatic | Local System |

**Figure 31: Filebeat and Logstash running as Windows services on the clinical data repository**

The setup of the CHIC audit system (audit parser, MongoDB, audit viewer) and the alternative audit system (Elasticsearch and Kibana) is not described in detail here (see D5.2.2) but an example of the Kibana user interface is illustrated in Figure 32. A sample audit record using the XDASv2 data model is listed in Figure 33.



```
August 8th 2016, 10:52:50.596   initiator.account.domain: www.smir.ch initiator.account.name: baral1 initiator.account.id: 95
                                target.entity.sysAddr: 192.168.122.20 target.entity.sysName: vsd-local1 target.data.affectedO
                                bjects: { "objectId": 122294, "objectUrl": "https://www.smir.ch/api/objects/122294" }
                                observer.entity.sysAddr: 192.168.122.20 observer.entity.sysName: vsd-local1 action.event.id: 0.0.
                                2.2 action.event.name: QUERY_DATA_ITEM_ATTRIBUTE action.subEvent.name: DOWNLOAD_OBJECT

August 8th 2016, 10:11:35.395   initiator.account.domain: www.smir.ch initiator.account.name: baral1 initiator.account.id: 95
```

**Figure 32: An example of the Kibana user interface displaying audit records using XDASv2**

```
{
  "_index": "filebeat-2016.08.08",
  "_type": "XDASv2",
  "_id": "AVZpWuwxUt2j4qrrTnU1",
  "_score": null,
  "_source": {
    "initiator": {
      "account": {
        "domain": "cdr-dev-chic.ics.forth.gr",
        "name": "niklr1",
        "id": 7
      }
    },
    "target": {
      "entity": {
        "sysAddr": "10.1.2.57",
        "sysName": "cdr-dev-chic"
      },
      "data": {
        "affectedObjects": [
          {
            "objectId": 12,
            "objectUrl": "https://cdr-dev-chic.ics.forth.gr/api/objects/12"
          }
        ]
      }
    },
    "observer": {
      "entity": {
        "sysAddr": "10.1.2.57",
        "sysName": "cdr-dev-chic"
      }
    },
    "action": {
      "event": {
        "id": "0.0.2.2",
        "name": "QUERY_DATA_ITEM_ATTRIBUTE"
      },
      "subEvent": {
        "name": "DOWNLOAD_OBJECT"
      },
      "time": {
        "offset": 1470646363,
        "certainty": 100
      },
      "outcome": "0",
      "extendedOutcome": "0"
    },
    "@version": "1",
    "@timestamp": "2016-08-08T08:52:50.596Z",
    "type": "XDASv2",
    "input_type": "log",
    "count": 1,
    "beat": {
      "hostname": "cdr-dev-chic",
      "name": "cdr-dev-chic"
    },
    "source": "C:\\www\\VSD\\Data\\Logs\\Audit\\xdas.txt",
    "offset": 41179,
    "fields": null,
    "host": "cdr-dev-chic",
    "tags": [
      "beats_input_codec_json_applied"
    ]
  },
  "fields": {
    "@timestamp": [
      1470646370596
    ]
  },
  "sort": [
    1470646370596
  ]
}
```

**Figure 33: A sample audit record in JSON format using the XDASv2 data model generated by the clinical data repository and processed by Filebeat and Logstash**

## *5.6 Domain Model*

The domain model of the clinical data repository, which has been introduced in deliverable D8.1, is illustrated in Figure 34 for the sake of completeness. Apart from the following modifications, the domain model stays the same as described in deliverables D8.1 and D8.3:

- The Triplestore entity has been added in order to support multiple triplestores in the export process of triples.
- AnnotationTripleLog has been added to track the progress of the triple export process. It stores information about the triple to be exported, the action to be performed (insert or delete), the status of the export process, the retry count, etc.
- TripleSubjectType has been added in order to support different types of triple subjects such as object, user, file, etc.
- The AnnotationTriple entity has been added to track modifications of a specific subject having multiple triples. Only the differences will be processed by the internal export mechanisms.

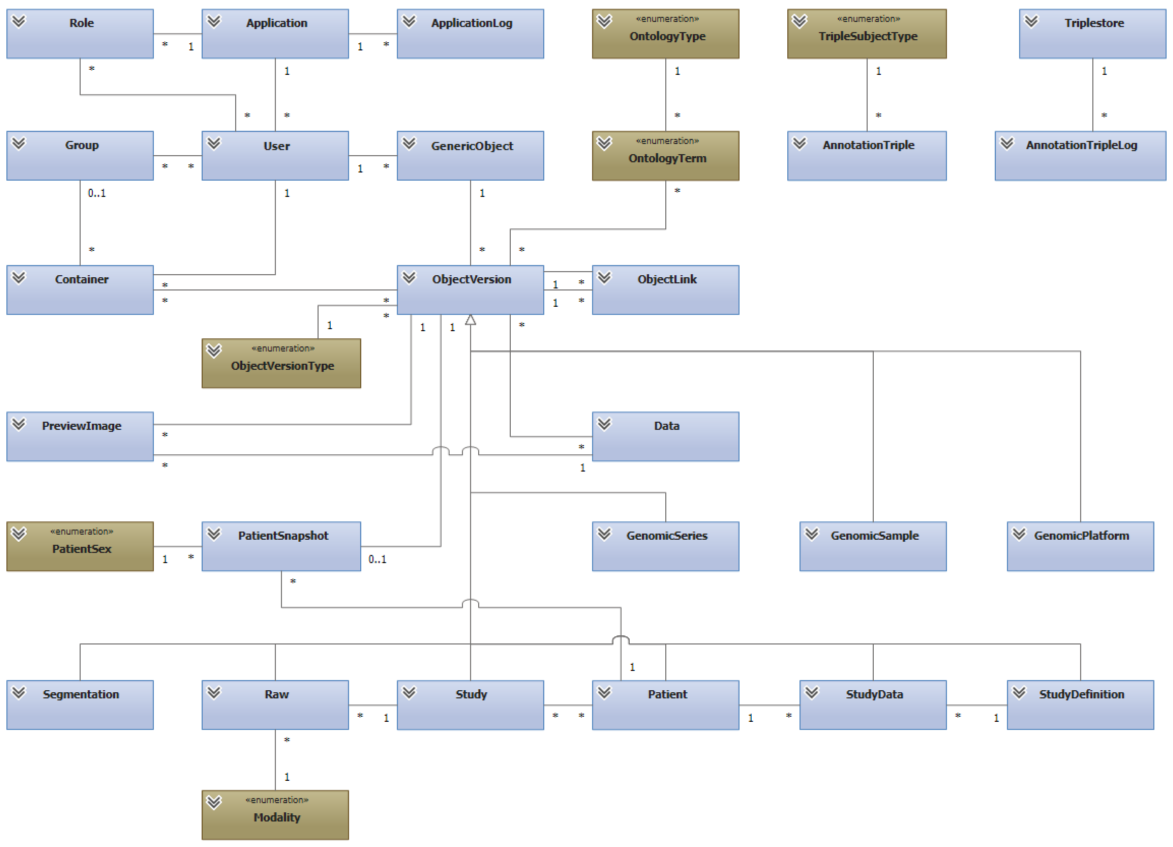
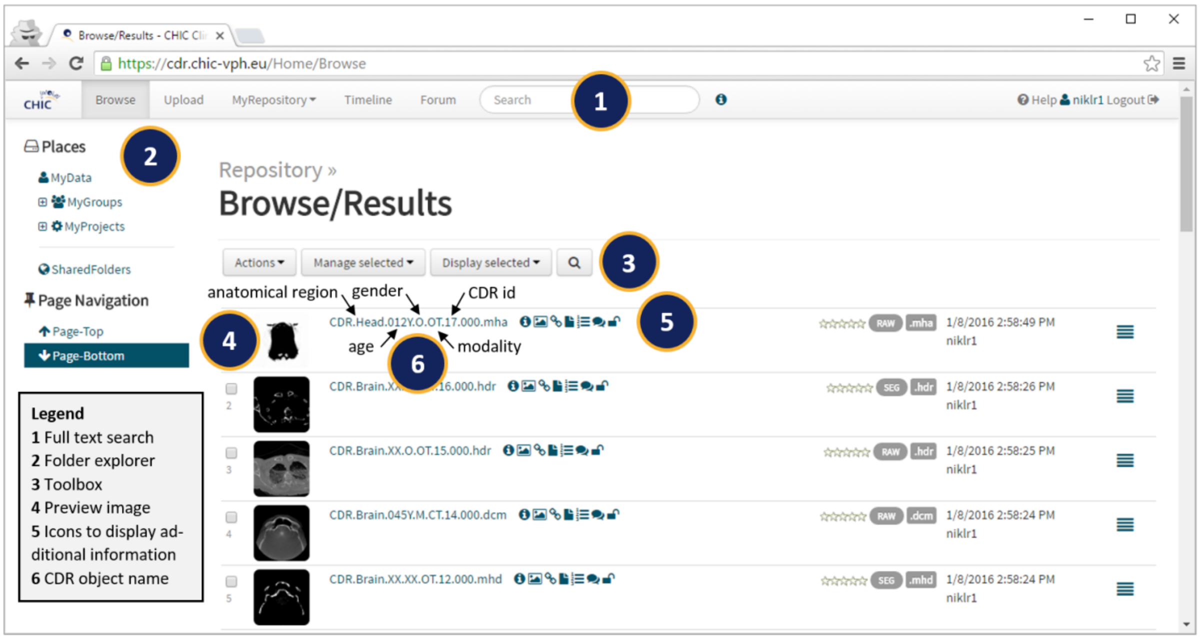


**Figure 34: The domain model of the clinical data repository with domain classes (blue), domain enumerations (brown) and their relationships represented as connecting lines.**

## 5.7 *Web-based user interface*

The implementation of the web-based user interface offers a main view illustrated in Figure 35 which serves as entry point for almost all functionalities described throughout the user guide introduced in the previous deliverable D8.2. 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.



Legend
**1** Full text search
**2** Folder explorer
**3** Toolbox
**4** Preview image
**5** Icons to display additional information
**6** CDR object name

| | | | | |
|---|---|---|---|---|
| | CDR.Head.012Y.O.OT.17.000.mha | RAW .mha | 1/8/2016 2:58:49 PM | niklr1 |
| 2 | CDR.Brain.XX.16.000.hdr | SEG .hdr | 1/8/2016 2:58:26 PM | niklr1 |
| 3 | CDR.Brain.XX.O.OT.15.000.hdr | RAW .hdr | 1/8/2016 2:58:25 PM | niklr1 |
| 4 | CDR.Brain.045Y.M.CT.14.000.dcm | RAW .dcm | 1/8/2016 2:58:24 PM | niklr1 |
| 5 | CDR.Brain.XX.XX.OT.12.000.mhd | SEG .mhd | 1/8/2016 2:58:24 PM | niklr1 |

**Figure 35: 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).

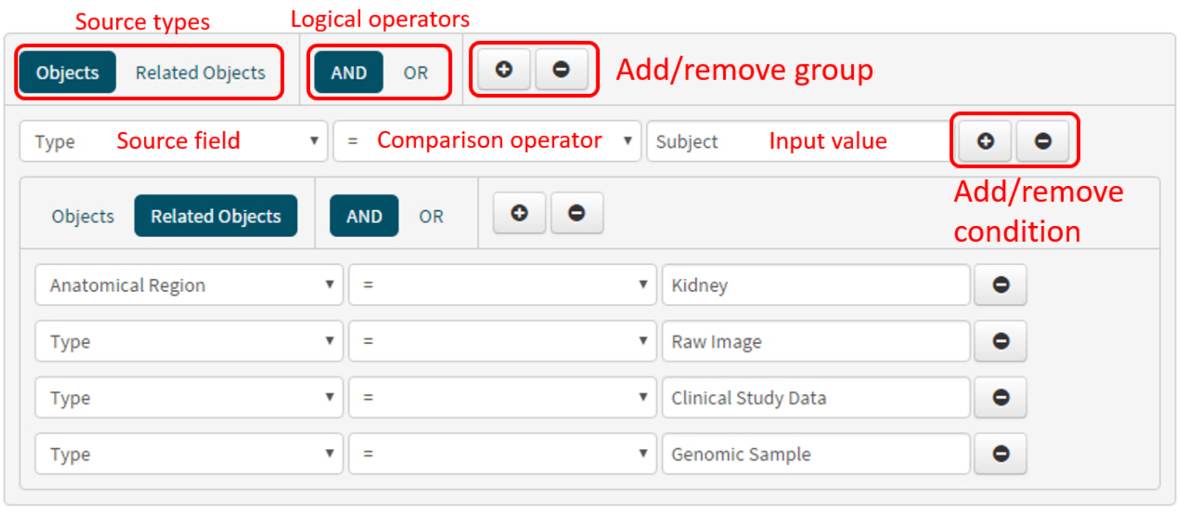


**Figure 36: The dynamic search query builder integrated in the web-based user interface**

The search case covered in Figure 36 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 published as open-source on the GitHub platform under the MIT license [11].

## *5.8 RESTful application programming interfaces*

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.

### 5.8.1 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:

**Table 35: 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 (OData) [3]. 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 API 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 [4]. Only the UTF-8 character encoding is supported for both requests and responses.

### 5.8.2 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 36: 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.<br>Allowed values: 10, 25, 50, 100, 250, 500 |
| page | int | 0 | Defines the current page index.<br>Allowed values: 0, 1, 2, ... |

**Example Request**

```
GET https://cdr.chic-vph.eu/api/objects?rpp=25&page=3
```

**Example Response**

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

### 5.8.3 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 37: 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** | | | |
| `{ "id": 1, "name": "Test group", "chief": { "selfUrl": "https://cdr.chic-vph.eu/api/users/2" }, "selfUrl": "https://cdr.chic-vph.eu/api/groups/1" }` | | | |
| **Example Request with include** | | | |
| `GET https://cdr.chic-vph.eu/api/groups/1?include=chief` | | | |
| **Example Response with include** | | | |
| `{ "id": 1, "name": "Test group", "chief": { "id": 2, "username": "niklr1", "selfUrl": "https://cdr.chic-vph.eu/api/users/2" },` | | | |

```
GET https://cdr.chic-vph.eu/api/groups/1
```

```
{
  "id": 1,
  "name": "Test group",
  "chief": {
    "selfUrl": "https://cdr.chic-vph.eu/api/users/2"
  },
  "selfUrl": "https://cdr.chic-vph.eu/api/groups/1"
}
```

```
GET https://cdr.chic-vph.eu/api/groups/1?include=chief
```

```
{
  "id": 1,
  "name": "Test group",
  "chief": {
    "id": 2,
    "username": "niklr1",
    "selfUrl": "https://cdr.chic-vph.eu/api/users/2"
  },
  "selfUrl": "https://cdr.chic-vph.eu/api/groups/1"
}
```

### 5.8.4 Requests, Responses and Errors

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

**Table 38: 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:

**Table 39: The possible return states used by the clinical data repository to indicate an unsuccessful completion of a request**

| 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 used 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** | When a method causes an exception that has no adequate handling in the |

| | |
|---|---|
| **Error** | 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. |

### 5.8.5 Resource Description Template

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

**Table 40: The template used to describe the API endpoint resources of the clinical data repository**

| **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. | | |

### 5.8.6 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**

```
{
   "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",
          "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:<br>• 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":{
                   "name":"Equals",
                   "displayName":"="
                },
                "inputItem":{
                   "data":7203,
                   "displayName":"Kidney",
                   "isTypeahead":true
                }
             },
             {
                "sourceField":{
                   "name":"ObjectType",
                   "displayName":"Type"
                },
                "comparisonOperator":{
                   "name":"Equals",
                   "displayName":"="
                },
                "inputItem":{
                   "data":"RawImage",
                   "displayName":"Raw Image",
                   "isTypeahead":true
                }
             },
             {
                "sourceField":{
                   "name":"ObjectType",
                   "displayName":"Type"
                },
                "comparisonOperator":{
                   "name":"Equals",
                   "displayName":"="
                },
                "inputItem":{
                   "data":"ClinicalStudyData",
                   "displayName":"Clinical Study Data",
                   "isTypeahead":true
                }
             },
             {
                "sourceField":{
                   "name":"ObjectType",
                   "displayName":"Type"
                },
                "comparisonOperator":{
                   "name":"Equals",
                   "displayName":"="
                },
                "inputItem":{
                   "data":"GenomicSample",
                   "displayName":"Genomic Sample",
                   "isTypeahead":true
                }
             }
          ]
       }
    ]
 }
```

**Example Response**

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

## 5.9 *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.

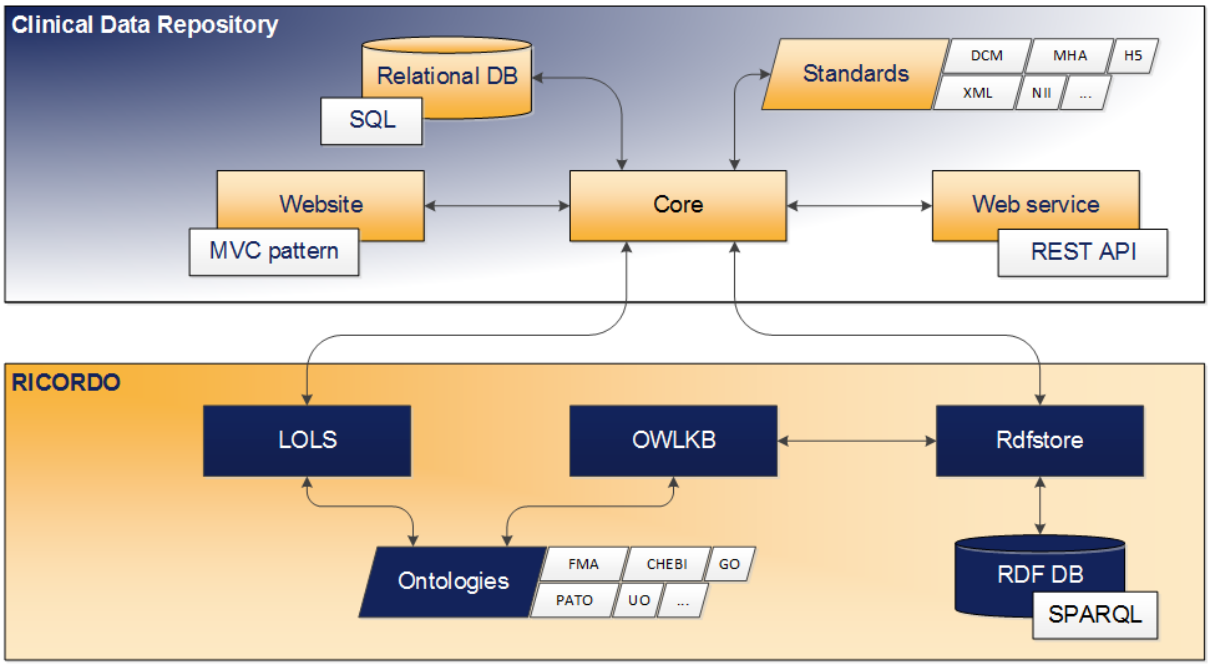
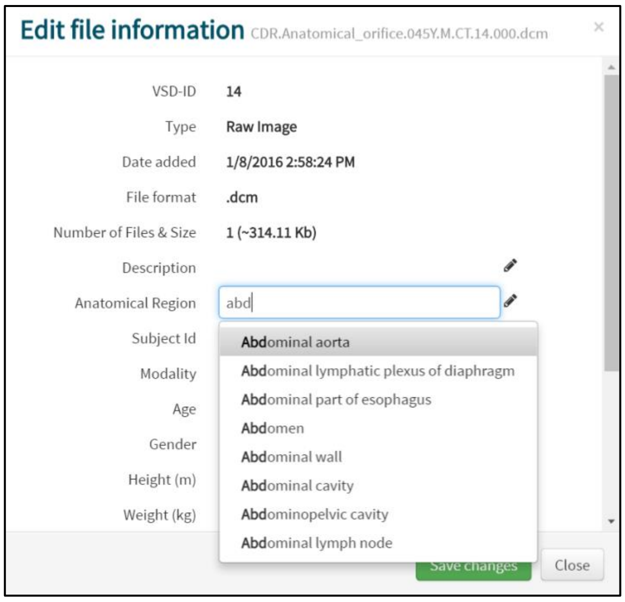


**Figure 37: A visual representation of interactions between clinical data repository and RICORDO components.**

RICORDO offers three components called LOLS, RDF store 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. RDF store 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.

### 5.9.1 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 38, 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.


Edit file information CDR.Anatomical_orifice.045Y.M.CT.14.000.dcm

|  |  |
|---|---|
| VSD-ID | 14 |
| Type | Raw Image |
| Date added | 1/8/2016 2:58:24 PM |
| File format | .dcm |
| Number of Files & Size | 1 (~314.11 Kb) |
| Description |  |
| Anatomical Region | abd |
| Subject Id |  |
| Modality |  |
| Age |  |
| Gender |  |
| Height (m) |  |
| Weight (kg) |  |

**Figure 38: User dialog to annotate an object with anatomical regions using the autocomplete function offered by the Local Ontology Lookup Service (LOLS).**

### 5.9.2 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 RDF store. Therefore, the metadata must be processed to triples before being exported to the RDF store. 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 RDF store every time. The RDF store 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 RDF store 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 RDF store 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 RDF store directly. Once the query involves information stored in an ontology such as “get all objects which are part of FMA Head" the RDF store 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 RDF store, two libraries have been developed and published as open-source on the GitHub platform under the MIT license. RdfMapperNet [12] is a .NET library to map classes to RDF triples and RdfstoreNet [13] is a .NET library for the Rdfstore API.

## 5.10 Summary

The implementation of the clinical data repository has been made available to the CHIC users and is running on the CHIC cloud infrastructure. The system includes all the features required to store the different types of data produced during the clinical workflow, which not only includes patient and treatment information, but also medical images, generic examination, histology. Due to its architecture the system can handle virtually any kind of data, but the concept within CHIC was to rely on standard formats.

The recommended brokered authentication mechanism introduced in deliverable "D5.2 - Security guidelines and initial version of security tools" has been fully integrated into the clinical data repository to support Single Sign-On. A single point controls the access to all the infrastructure, including the CDR. Several interfaces have been developed to store, browse, search and retrieve data from the CDR. First a complete website has been developed using modern web technologies. In addition to the website access, a specific REST API has been developed to provide access to the CDR content for integration with other software components. The features and functionalities are identical between both interfaces.

Semantic annotation of the clinical data has been integrated within the CHIC RICORDO framework. The CDR automatically retrieves the information from the files uploaded on the system and generates semantic triples. These triples are initially stored locally and a synchronization mechanism has been developed to synchronize the local triples with remote storage locations (in our case, RICORDO). Manual annotation of specific field is also possible and relies on pre-selected ontologies.

Finally, a system to track the activities by individual users on the system has put in place. This system tracks the access to the system and to the data to build a chronological audit trail for each user. In this deliverable we have presented the data model used for audit records and the architecture of the auditing within the clinical data repository which supports different and multiple audit systems at the same time.

All components of the clinical data repository have been successfully deployed to the private cloud infrastructure provided by FORTH allowing great flexibility in terms of compute, storage, and networking resources. The deployed services can be accessed by the following URLs:

- Production
  - Website: https://cdr.chic-vph.eu
  - API: https://cdr.chic-vph.eu/api
- Development
  - Website: https://cdr-dev-chic.ics.forth.gr
  - API: https://cdr-dev-chic.ics.forth.gr/api

# 6 *In Silico* Trial Repository

## 6.1 *Introduction*

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. The two CHIC components that usually interact with the *In Silico* Trial Repository are the Hypermodelling Execution Framework, which stores the outcome of a simulation back to the Repository, and the CRAF which retrieves the results. Even if the current status of the *In Silico* Trial Repository conforms to the user needs and requirements (WP2), to the legal and ethical framework (WP4), to the IT Architecture (WP5) and to the integrated platform guidelines (WP10), The Repository is expected to be constantly updated throughout the remaining period of the CHIC project.

## 6.2 *Architecture of the In SIlico Trial Repository*

One of the main purposes of the *In Silico* Trial Repository is to test the repeatability and reproducibility of the experiments conducted in the context of *in silico* cancer domain.

Repeatability is the ability for an individual to show that an experiment, repeated using the same material and equipment, yields the same result. In *in silico* medicine this means that if we run the same module multiple times on the same computer using the same software the same result would be yielded.

Reproducibility is the ability for different individuals to show that an experiment repeated using different but similar material and different equipment yields the same statistical result. In *in silico* medicine this means that we are able to recreate a simulation without necessarily using the same software or computer that was used in the original simulation. Reproducing an experiment is one important approach that scientists use to gain confidence in their conclusions [1].

The *In Silico* Trial Repository can serve perfectly the aforementioned initiatives. By storing in one place the complete information concerning the input data, the output data, and the modules which participate in the *in silico* experiments and the *in silico* trials, the *In Silico* Trial Repository can advance *in silico* medicine in general, by facilitating the validation of the current *in silico* medicine discoveries.

The *In Silico Trial Repository* consists of three main entities, the subjects, the experiments and the trials. The basic principles of the *In Silico* Trial Repository are the following:

- The subject entity represents an instance of a subject. The subject may be a person, healthy or not, an animal, etc. The subject can be linked to another data repository, such as the CHIC clinical data repository, a clinical trial management system (ObTiMA, OpenClinica, etc.), a hospital record management system, etc. Every instance of a subject can be accompanied by a set of files.
- The *in silico* experiment entity consists of triples of "initial state of the subject" – "*in silico (*hyper)model" – "final state of the subject". The *in silico* (hyper)model that is used in an *in silico* experiment is not stated in the experiment entity, but in the *in silico* trial entity in which the experiment belongs.
- The *in silico* experiments are organized in *in silico* trials. All *in silico* experiments that are part of the same *in silico* trial use the same *in silico* (hyper)model.
- The (hyper)model that is being used (and the location where it is stored) is defined in the *in silico* trial entity.
- To be in alliance with the real clinical trial the term "placebo model" is introduced. In case of cancer disease the placebo model can be a free growth model.
- Each *in silico* experiment and each *in silico* trial may be linked to external references (journal articles, conference proceedings, etc.)
- Apart from the input or output files and the parameters that are patient specific, the *In Silico* Trial Repository is able to also store the values that have been assigned to input miscellaneous parameters of the corresponding hypermodel.

Based on the aforementioned principles, Figure 39 presents the Entity Relationship (ER) diagram of the *In Silico* Trial Repository.

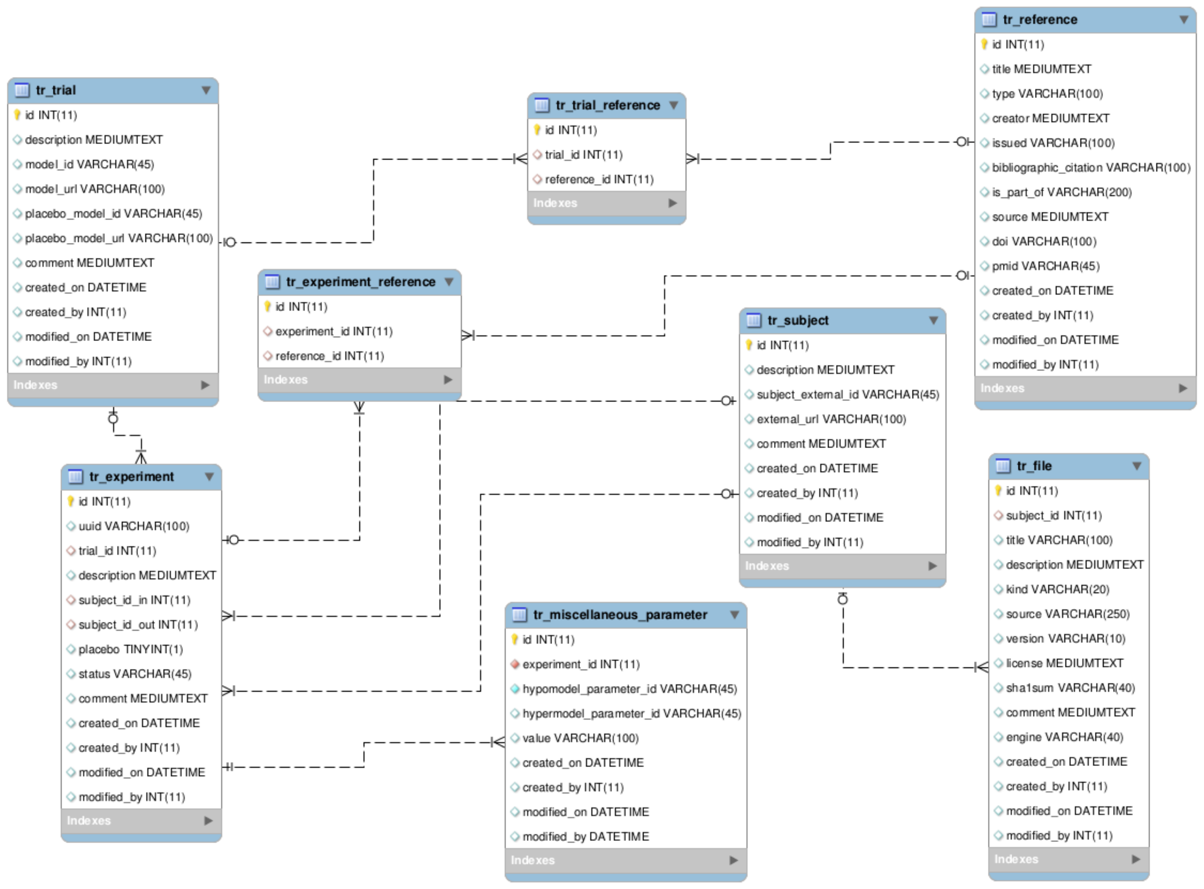


**Figure 39: Entity Relationship (ER) diagram of the In Silico Trial Repository**

As shown in Figure 39, the main entities used in the Repository are named “tr_trial”, “tr_experiment”, “tr_reference”, “tr_experiment_reference”, “tr_trial_reference”, “tr_subject”, “tr_file” and “tr_miscellaneous_parameter”. A description of the aforementioned entities along with their attributes (MySQL columns) is given below:

### Entity tr_subject

- id: Primary key. Used to uniquely identify each row table row.
- description: The (short) textual description of the state of the subject. The subject may be a person (healthy or patient), an animal, etc.
- subject_external_id: The external id of the subject. This field is used only in the case in which this subject’s case (real or virtual) is directly or indirectly linked to a subject stored in an external repository.
- external_url: The URL of the external repository mentioned above. Such external repositories can be the CHIC clinical data repository, a clinical trial management system (ObTiMA, OpenClinica, etc.), a hospital record management system, etc.
- comment: Any additional comment
- created_on: The date and time when this subject has been created.
- created_by: The id of the creator of this subject
- modified_on: The date and time when this subject has been modified

- modified_by: The id of the modifier of this subject

**Entity tr_trial**

- id: Primary key. Used to uniquely identify each row of the table
- description: The (short) textual description of the trial
- model_id: The id of the *in silico* model that is used in the trial
- model_url: The URL where the *in silico* model is located. This URL may point to the CHIC model/tool Repository or to an external model repository (e.g. the biomodels repository)
- placebo_model_id: The *in silico* model that is used as a placebo. Usually in cancer disease this is a free growth model.
- placebo_model_url: The URL where the placebo *in silico* model is located.
- comment: Any additional comment
- created_on: The date and time when this trial has been created
- created_by: The id of the creator of this trial
- modified_on: The date and time when this trial has been modified
- modified_by: The id of the modifier of this trial

**Entity tr_experiment**

- id: Primary key. Used to uniquely identify each row of the table
- uuid: Universally unique identifier of the experiment. (This attribute has been created after request from WP7)
- trial_id: The id of the trial to which this experiment belongs
- description: The (short) textual description of the *in silico* experiment
- subject_id_in: The id of (the state of) the subject that is used an input to the *in silico* experiment
- subject_id_out: The id of (the state of) the subject that is produced after the execution of the *in silico* experiment
- placebo: True if in the *in silico* experiment the “placebo” model must be used
- status: The status of the *in silico* experiment. It can be “NOT STARTED”, “ON PROGRESS”, “FINISHED SUCCESSFULLY” and “FINISHED ERRONEOUSLY”
- comment: Any additional comment
- created_on: The date and time when this experiment has been created
- created_by: The id of the creator of this experiment
- modified_on: The date and time when this experiment has been modified
- modified_by: The id of the modifier of this experiment

**Entity tr_reference**

- id: Primary key. Used to uniquely identify each row of the table
- title: The name given to the resource
- type: The type of the resource. Example values: "book", "journal article", "conference proceedings"
- creator: The creator(s) of the resource (e.g. authors, etc.)
- issued: The date of formal issuance (e.g. publication) of the resource
- bibliographic_citation: The bibliographic citation of the resource
- is_part_of: The related resource that this resource is part of
- source: The related resource from which the described resource is derived from
- doi: The DOI (Digital Object Identifier) of the resource. This field is empty if the resource doesn't have a DOI.
- pmid: The PubMed identifier. This field is empty if the resource is not included in the PubMed database.
- created_on: The date and time when this reference has been created
- created_by: The id of the creator of this reference
- modified_on: The date and time when this reference has been modified
- modified_by: The id of the modifier of this reference

**Entity tr_trial_reference**

- id: Primary key. Used to uniquely identify each row of the table
- trial_id: The id of the trial. Linked to the table "tr_trial"
- reference_id: The id of the reference. Linked to the table "tr_reference"
- created_on: The date and time when this record has been created
- created_by: The id of the creator of this record
- modified_on: The date and time when this record has been modified
- modified_by: The id of the modifier of this record

**Entity tr_experiment_reference**

- id: Primary key. Used to uniquely identify each row of the table
- experiment_id: The id of the experiment. Linked to the table "tr_experiment"
- reference_id: The id of the reference. Linked to the table "tr_reference"
- created_on: The date and time when this record has been created
- created_by: The id of the creator of this record
- modified_on: The date and time when this record has been modified
- modified_by: The id of the modifier of this record

**Entity tr_file**

- id: Primary key. Used to uniquely identify each table row.
- subject_id: The id of the subject to which this file is linked.
- title: The name of the file
- description: The (short) textual description of what this file represents
- kind: Defines what this file is. Example values: “document”, “raw”, “log”,”dat”,”report”, etc.
- source: The location where this file is internally stored
- version: The version of the file
- sha1sum: The sha1 checksum of this file (data). It is used in order to check the consistency of the file
- comment: Any additional comment
- created_on: The date and time when this record has been created
- created_by: The id of the creator of this record
- modified_on: The date and time when this record has been modified
- modified_by: The id of the modifier of this record

**Entity tr_miscellaneous_parameter**

- id: Primary key. Used to uniquely identify each table row
- experiment_id: The id of the experiment to which the value of this miscellaneous parameter is linked
- hypomodel_parameter_id: The id of the hypomodel parameter to which the value of this miscellaneous parameter is linked. It is a reference link to “mr_parameter” entity from the Model Repository
- hypermodel_parameter_id: The id of the hypermodel parameter to which the value of this miscellaneous parameter is linked. It is a reference link to “mr_parameter” from the Model Repository
- value: The value that has been assigned to this miscellaneous parameter during the corresponding *in silico* experiment
- created_on: The date and time when this record has been created
- created_by: The id of the creator of this record
- modified_on: The date and time when this record has been modified
- modified_by: The id of the modifier of this record.

The entity Relationship diagram (ER) which has been depicted in Figure 39, represents the design of the relational database of the *In Silico* Trial Repository. This design has been documented in “D8.1: Design of the CHIC Repositories”, but during the CHIC project it has undergone some changes based on the new requirements that came from the other work packages. For instance, the “tr_experiment” has now the attribute (table column) uuid which holds the universally unique identifier for each *in silico* experiment. Moreover, the “tr_miscellaneous_parameter” entity may be used in order to hold the values that have been assigned to the model miscellaneous parameters during the corresponding *in silico* experiment.

The schema of the relational database of the *In Silico* Trial Repository that has been just reported in this chapter, has been designed in order to be able to efficiently store within the CHIC platform all the persistent data that are related to simulations. The input of the models, the identification of the patient used in the experiment, the values that have been assigned to the model parameters during the simulation, the information of the (hyper)model used in the *in silico* trial and the PDF report that is generated by the CRAF component are all stored in the MySQL database of the *In Silico* Trial Repository. Apart from the MySQL database server which is responsible for the persistent storage of the simulation data, the *In Silico* Trial Repository consists of many other components, such as the Apache Application Server, the Django Web Framework, some back-end and front-end libraries and dependencies and some security libraries. All the aforementioned components have been utilized in order to build a fully integrated web application which not only stores the simulation data in a local relational database, but also takes part in all the complex research and clinical workflows within the CHIC platform through the web services that have been developed according to the legal and ethical framework of CHIC. Table 41 presents all the components, external libraries, applications and dependencies that are being used in the *In Silico* Trial Repository along with their licenses.

**Table 41: External components (dependencies, libraries, applications) of the In Silico Trial Repository**

| **External Component (Dependency – Library – Application)** | **License** | **Usage** |
|---|---|---|
| Apache HTTP Server | Apache license | A secure, efficient, and extensible server that provides HTTP services in sync with the current HTTP standards. |
| MySQL community edition | GPL license | The relational database server responsible for persistently storing information related to models. |
| Django Rest Framework | Copyright (c) 2011-2016, Tom Christie All rights reserved | A powerful and flexible toolkit for building web APIs |
| djangosaml2 | Apache2 license | A Django application that integrates the PySAML2 library into the Model Repository project in order to be able to incorporate the SAML front-end authentication mechanism. |

| | | |
|---|---|---|
| dm.xmlsec.binding | BSD license | XML security library used to authenticate web service requests. |
| XML security library | MIT license | A C library that supports XML security standards (XML signature, XML encryption, etc.). It is being used by djangosaml2 and dm.xmlsec.binding. |
| Django | BSD license | The Python Web Framework that has been used for the development of the Model Repository |
| jQuery library | MIT license | A javascript library which is being used by the Model Repository for event handling, animation, and Ajax calls. |
| Bootstrap framework | MIT license | HTML, CSS and JS framework for developing part of the front-end of the Model Repository. |

As shown in Table 41, nine major external components are used in the *In Silico* Trial Repository web application. Some of these components are related to the security (djangosaml2, dm.xmlsec.binding, XML security library), some are related to the back-end of the application (Apache HTTP Server, MySQL Database Server, Django, Django Rest Framework) and finally some are related to the front-end (jQuery library, Bootstrap framework).

Just like the Model Repository, the main component of the *In Silico* Trial Repository is the Django web framework which has been utilized in order to develop the major part of the aforementioned Repository. The business logic, the presentation layer, the URL dispatching, the object relational mapping and the web services are all handled by the Django framework. Moreover, the Django web framework has been properly configured in order to integrate all the external libraries (security dependencies, front-end tools, etc.), to analyze the URLs of the incoming requests, to perform the business logic, to develop the web services, to handle the HTTP requests, and to connect to the local relational database of the *In Silico* Trial Repository. Figure 40 depicts how the Django web framework accommodates the *In Silico* Trial Application.

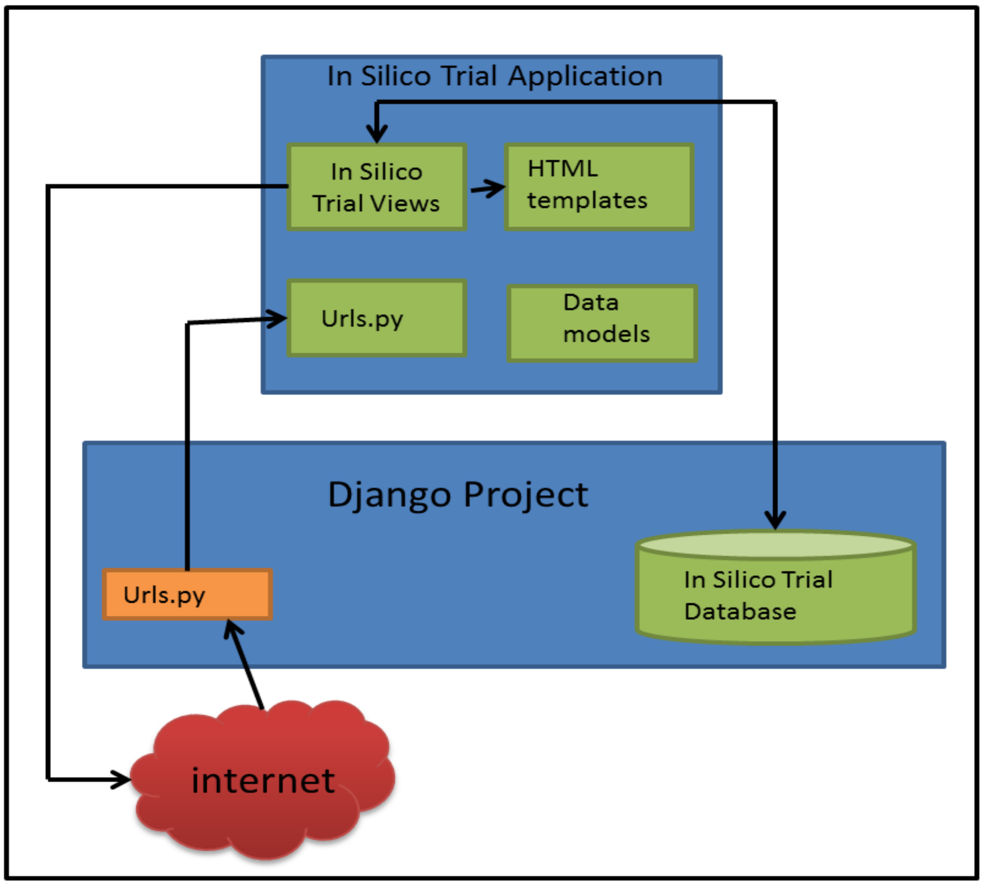


**Figure 40: The In Silico Trial Repository has been integrated into the Django Web Framework**

As shown in Figure 40:

- The *In Silico* Trial Application includes some combination of models, views, templates, template tags, static files, URLs, middleware, etc. The *In Silico* Trial Repository application has been wired into the Django framework with the INSTALLED_APPS setting.
- The *In Silico* Trial Application uses data models in Python, in order to create a virtual object for the MySQL relational database of the *In Silico* Trial Repository.
- The Python file urls.py analyzes the URL of the incoming HTTP request and decides which Python function that resides inside the view.py file should be called. Then the aforementioned Python function will either prepare the data to be presented in the HTML template, or it will connect to the local MySQL database to perform changes based on the user requirements.
- The "data model" box (inside the *In Silico* Trial Application), contains many classes which describe in a more high level the schema of the Repository. It should be noted that the *In Silico* Trial Repository data model needs to be always synchronized with the MySQL *In Silico* Trial Repository database.

Due to the configuration of the components of the most critical part of the *In Silico* Trial Repository which is the Django web framework presented in Figure 40 and due to the external libraries that have been incorporated and presented in Table 41, the contents of the *In Silico* Trial Repository can be exposed through web services to other CHIC components such as the CRAF and the VPH-HF, or they can be rendered through the browser directly to the user. Just like in the case of the Model Repository, the protection of the privacy and integrity of the exchanged data (which may include sensitive information) is ensured by a proxy server which makes use of HTTPS protocol for outbound connections. This is guaranteed by a SSL certificate that has been installed by partner CUSTODIX in the virtual machine that accommodated the proxy.

Figure 41 presents the integration of the *In Silico* Trial Repository into the CHIC platform.

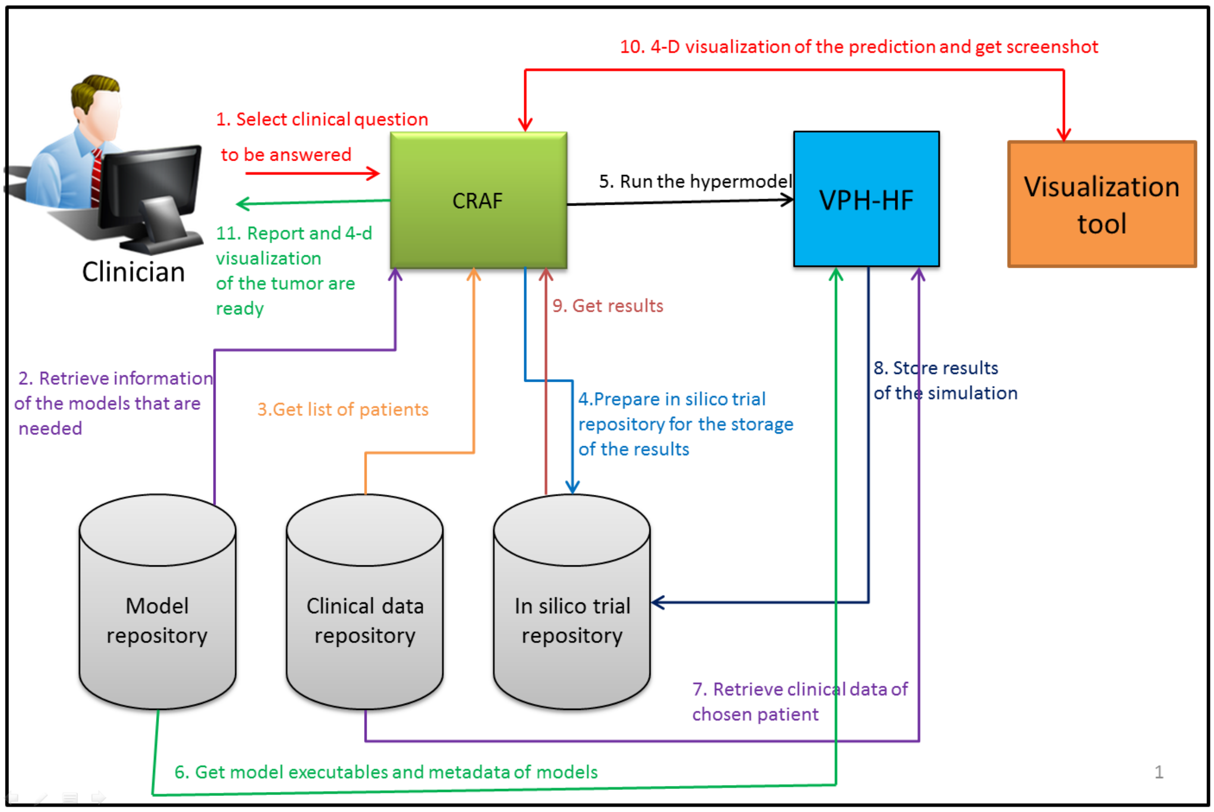

10. 4-D visualization of the prediction and get screenshot
1. Select clinical question to be answered
Clinician
11. Report and 4-d visualization of the tumor are ready
CRAF
5. Run the hypermodel
VPH-HF
Visualization tool
9. Get results
2. Retrieve information of the models that are needed
3.Get list of patients
4.Prepare in silico trial repository for the storage of the results
8. Store results of the simulation
Model repository
Clinical data repository
In silico trial repository
7. Retrieve clinical data of chosen patient
6. Get model executables and metadata of models
1

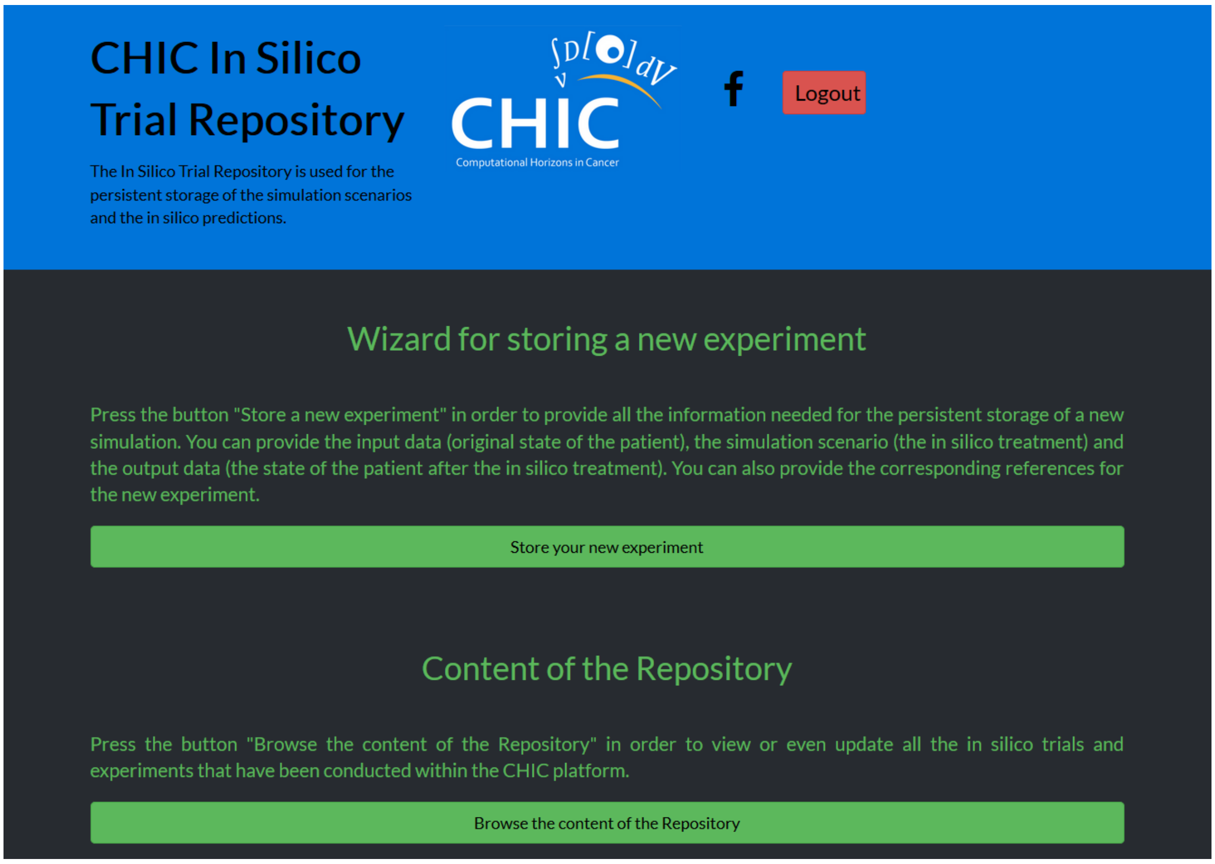
**Figure 41: Integration of the In SIlico Trial Repository into the CHIC Platform**

As shown in Figure 41, the *In Silico* Trial Repository communicates with the CRAF and the VPH-HF components. More specifically, before the execution of the hypermodel, CRAF prepares the *In Silico* Trial Repository for the storage of the results (creation of subjects, creation of the new experiment, etc.), and then the Hypermodelling Framework stores the results of the simulation back to the *In Silico* Trial Repository. Finally, CRAF component retrieves the results from the *In Silico* Trial Repository in order to visualize the results for the user.

## *6.3 The user interface of the In Silico Trial Repository*

The *In Silico* Trial Repository makes use of the principles of user interface design that have been mentioned in section 4.3 so as to improve the experience of the user when interacting with the Repository. Special emphasis has been given during the development of the *In Silico* Trial 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 (see chapter 3), the user is redirected to the main page of the *In Silico* Trial Repository which is depicted in Figure 42. As shown in the aforementioned 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. The workflows for the storage of a new *in silico* experiment and the browsing of the content of the Repository are being described in the next chapters.



CHIC In Silico Trial Repository

The In Silico Trial Repository is used for the persistent storage of the simulation scenarios and the in silico predictions.

Wizard for storing a new experiment

Press the button "Store a new experiment" in order to provide all the information needed for the persistent storage of a new simulation. You can provide the input data (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). You can also provide the corresponding references for the new experiment.

Content of the Repository

Press the button "Browse the content of the Repository" in order to view or even update all the in silico trials and experiments that have been conducted within the CHIC platform.

Logout
Store your new experiment
Browse the content of the Repository

**Figure 42: The main page of the In Silico Trial Repository**

### 6.3.1 Wizard for storing a new experiment

A wizard has been created for the *In Silico* Trial Repository in order for the user to be able 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

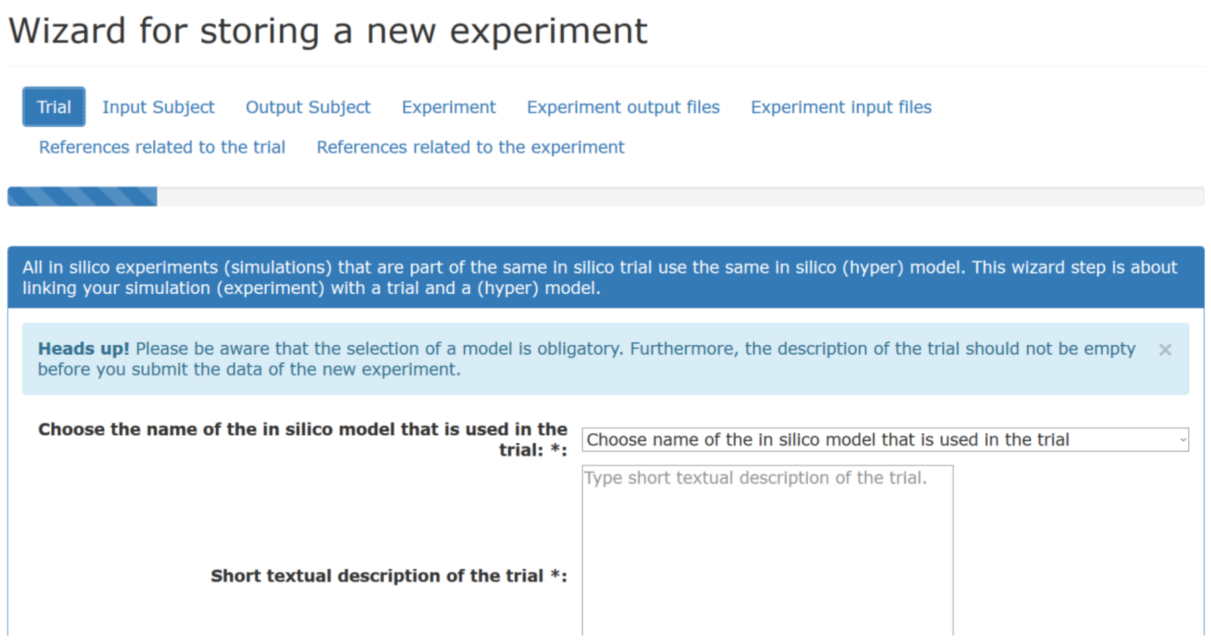
This wizard consists of seven steps, and in order for the information of the new *in silico* experiment to be valid, the user has to:

- Provide a description of the *in silico* trial
- Provide a description of the state of the input subject (initial state of the patient)
- Provide a description of the state of the output subject (final, simulated state of the patient)

- Provide a description for the new *in silico* experiment
- Provide at least one output file of the simulation. In case of many output files, the names of the output files should be unique (both the actual file names and the metadata titles)
- Provide input files of the simulation that have different names. The same applies for the references related to the experiments and the trials (they should have unique titles).

It should be noted that the user is able to skip the three last steps of this wizard for later. More specifically, the user may not provide any input files, or references related to the simulation.

The screenshots of the different steps regarding the aforementioned wizard are presented in Figures 43-50.


Wizard for storing a new experiment

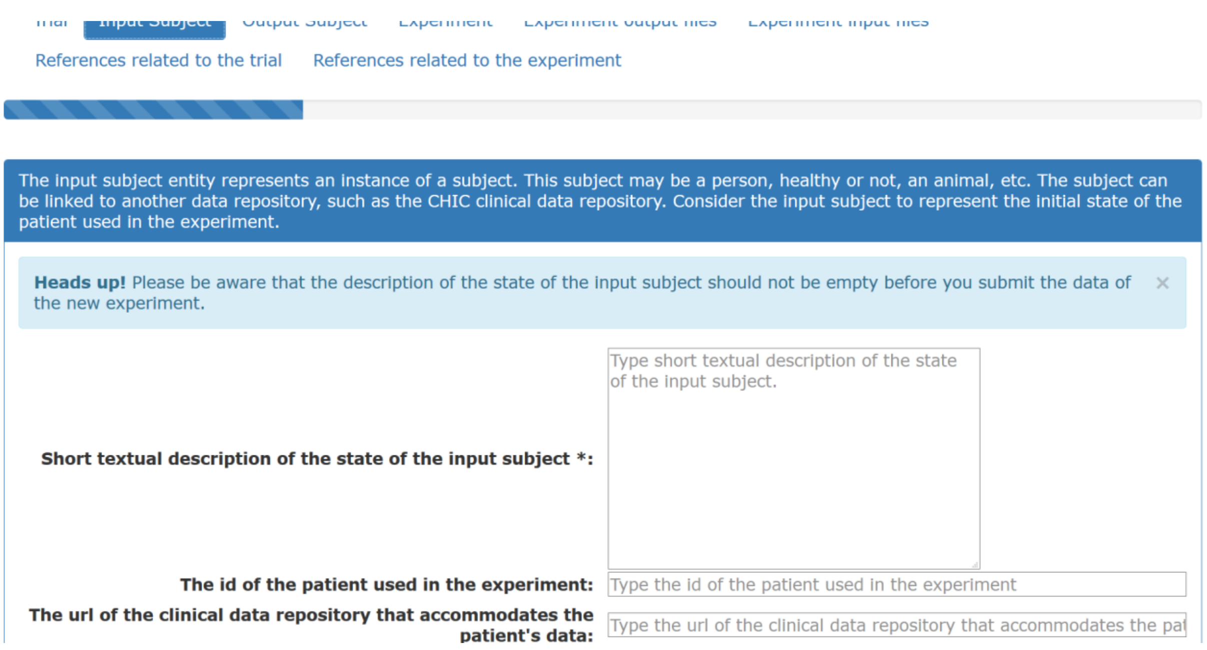
All in silico experiments (simulations) that are part of the same in silico trial use the same in silico (hyper) model. This wizard step is about linking your simulation (experiment) with a trial and a (hyper) model.

**Heads up!** Please be aware that the selection of a model is obligatory. Furthermore, the description of the trial should not be empty before you submit the data of the new experiment.

**Choose the name of the in silico model that is used in the trial: *:** Choose name of the in silico model that is used in the trial

**Short textual description of the trial *:** Type short textual description of the trial.

**Figure 43: The first step of the wizard. The user provides information related to the in silico trial to which the new in silico experiment belongs**



The input subject entity represents an instance of a subject. This subject may be a person, healthy or not, an animal, etc. The subject can be linked to another data repository, such as the CHIC clinical data repository. Consider the input subject to represent the initial state of the patient used in the experiment.

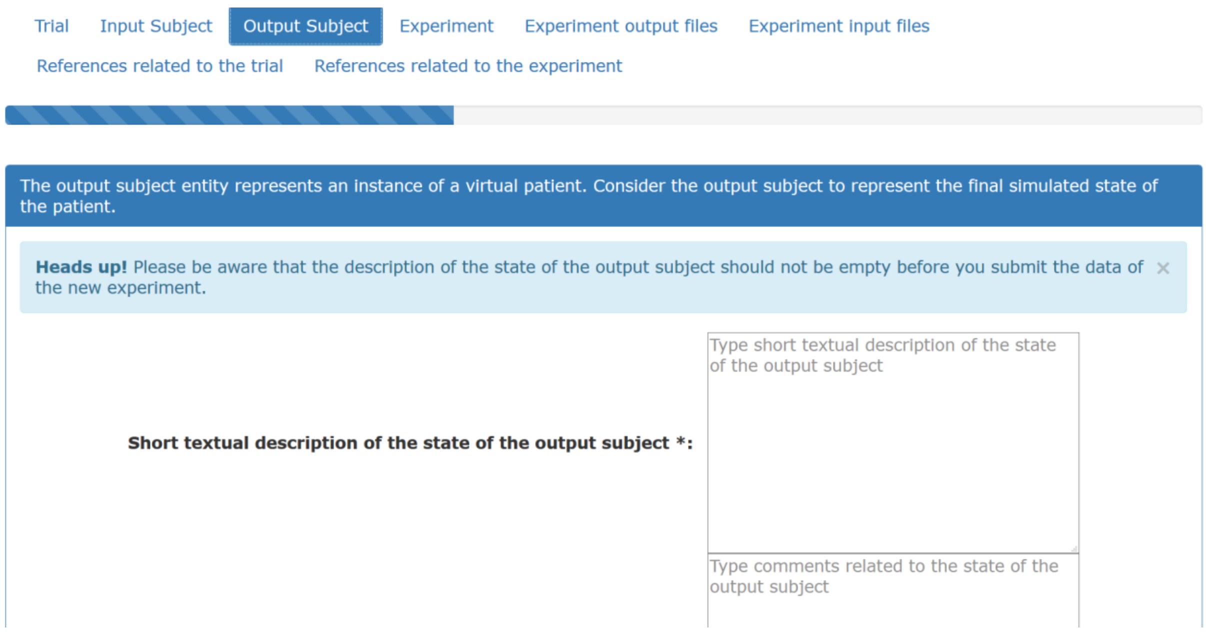
**Heads up!** Please be aware that the description of the state of the input subject should not be empty before you submit the data of the new experiment.

**Short textual description of the state of the input subject *:** Type short textual description of the state of the input subject.

**The id of the patient used in the experiment:** Type the id of the patient used in the experiment

**The url of the clinical data repository that accommodates the patient's data:** Type the url of the clinical data repository that accommodates the pat

**Figure 44: The second step of the wizard. The user provides information related to the initial state of the patient**



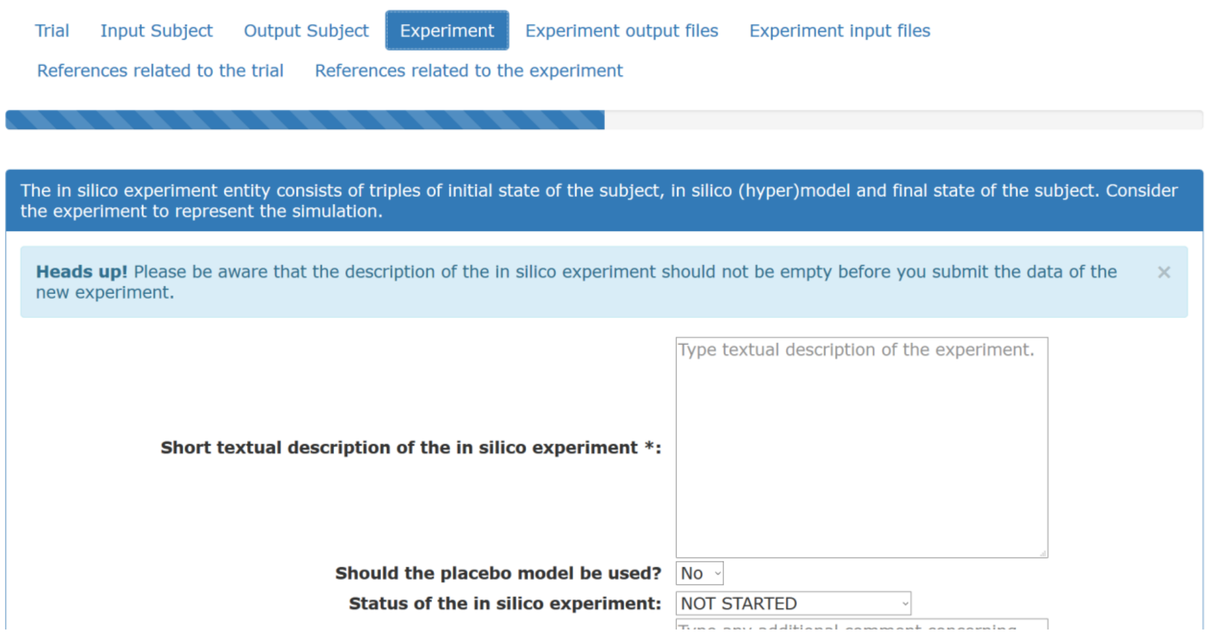
The output subject entity represents an instance of a virtual patient. Consider the output subject to represent the final simulated state of the patient.

**Heads up!** Please be aware that the description of the state of the output subject should not be empty before you submit the data of the new experiment.

**Short textual description of the state of the output subject *:** Type short textual description of the state of the output subject

Type comments related to the state of the output subject

**Figure 45: The third step of the wizard. The user provides information related to the final - simulated state of the patient**


Trial | Input Subject | Output Subject | Experiment | Experiment output files | Experiment input files | References related to the trial | References related to the experiment

The in silico experiment entity consists of triples of initial state of the subject, in silico (hyper)model and final state of the subject. Consider the experiment to represent the simulation.

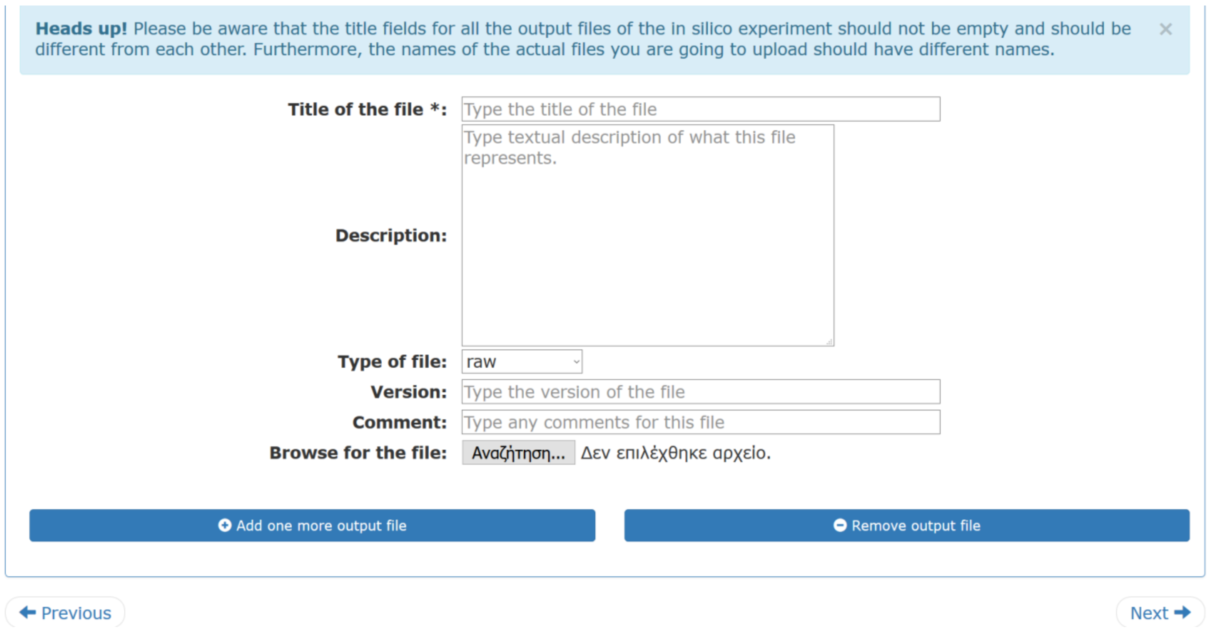
**Heads up!** Please be aware that the description of the in silico experiment should not be empty before you submit the data of the new experiment.

**Short textual description of the in silico experiment *:** Type textual description of the experiment.

**Should the placebo model be used?** No

**Status of the in silico experiment:** NOT STARTED

Figure 46: The fourth step of the wizard. The user provides information related to the new in silico experiment


**Heads up!** Please be aware that the title fields for all the output files of the in silico experiment should not be empty and should be different from each other. Furthermore, the names of the actual files you are going to upload should have different names.

**Title of the file *:** Type the title of the file

**Description:** Type textual description of what this file represents.

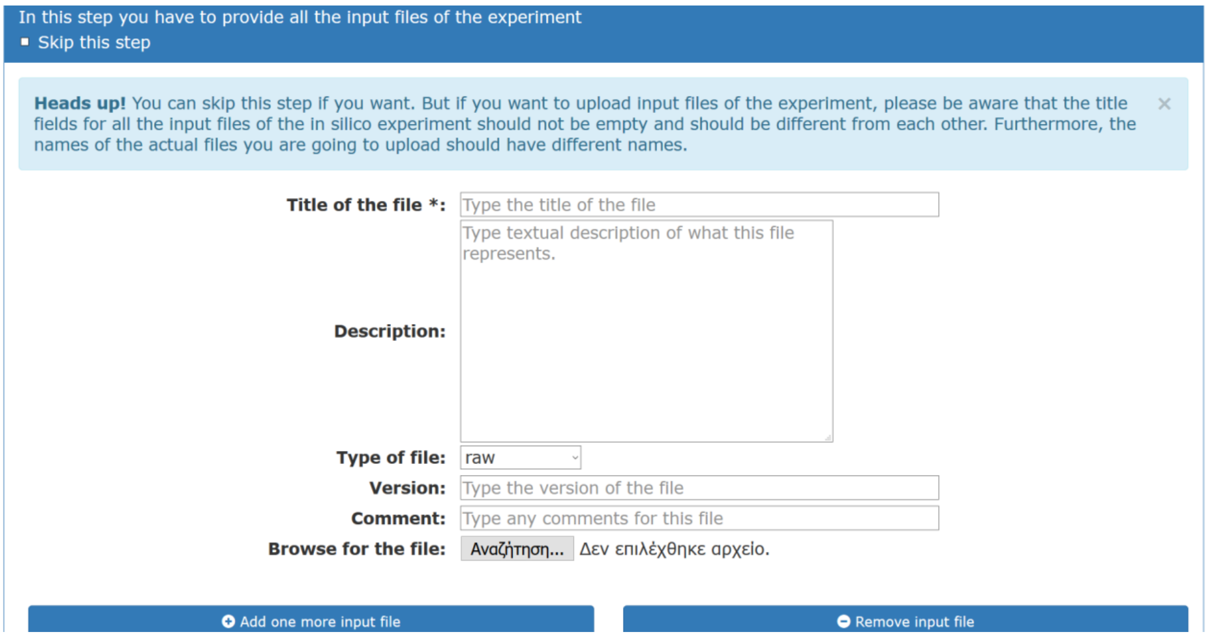
**Type of file:** raw

**Version:** Type the version of the file

**Comment:** Type any comments for this file

**Browse for the file:** Αναζήτηση... Δεν επιλέχθηκε αρχείο.

Figure 47: The fifth step of the wizard. The user uploads one or more output files related to the simulation

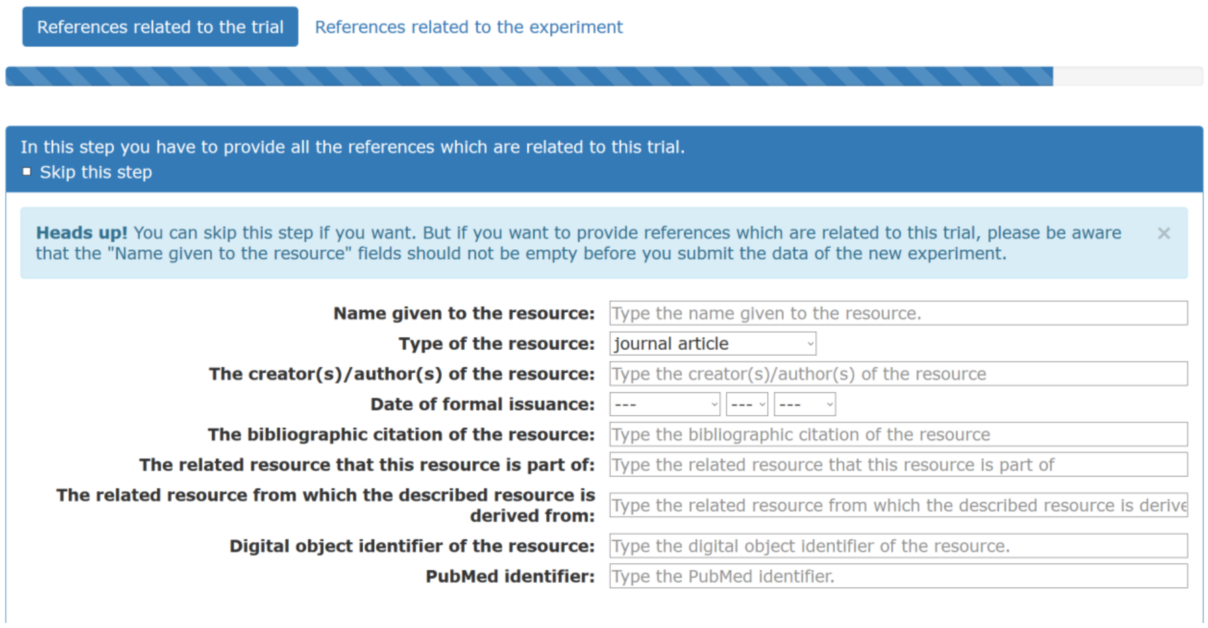

In this step you have to provide all the input files of the experiment

▪ Skip this step

**Heads up!** You can skip this step if you want. But if you want to upload input files of the experiment, please be aware that the title fields for all the input files of the in silico experiment should not be empty and should be different from each other. Furthermore, the names of the actual files you are going to upload should have different names. ×

| | |
|---|---|
| **Title of the file *:** | Type the title of the file |
| **Description:** | Type textual description of what this file represents. |
| **Type of file:** | raw |
| **Version:** | Type the version of the file |
| **Comment:** | Type any comments for this file |
| **Browse for the file:** | Αναζήτηση... Δεν επιλέχθηκε αρχείο. |

Figure 48: The sixth step of the wizard. The user uploads one or more input files related to the simulation


References related to the trial
References related to the experiment

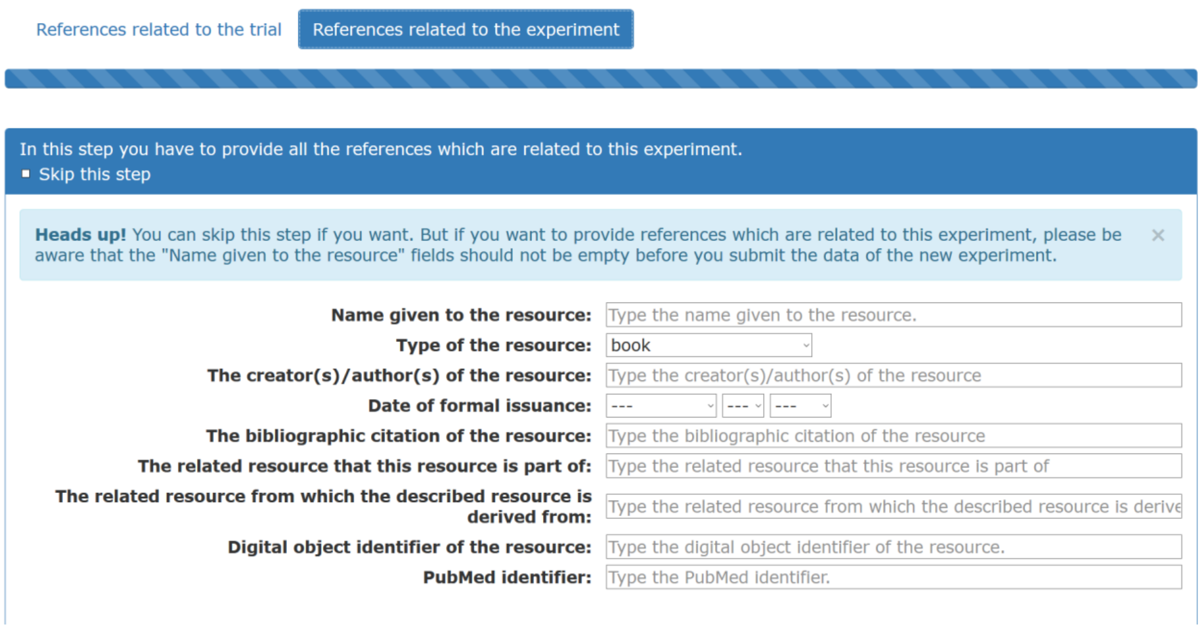
In this step you have to provide all the references which are related to this trial.

▪ Skip this step

**Heads up!** You can skip this step if you want. But if you want to provide references which are related to this trial, please be aware that the "Name given to the resource" fields should not be empty before you submit the data of the new experiment. ×

| | |
|---|---|
| **Name given to the resource:** | Type the name given to the resource. |
| **Type of the resource:** | journal article |
| **The creator(s)/author(s) of the resource:** | Type the creator(s)/author(s) of the resource |
| **Date of formal issuance:** | --- --- --- |
| **The bibliographic citation of the resource:** | Type the bibliographic citation of the resource |
| **The related resource that this resource is part of:** | Type the related resource that this resource is part of |
| **The related resource from which the described resource is derived from:** | Type the related resource from which the described resource is derive |
| **Digital object identifier of the resource:** | Type the digital object identifier of the resource. |
| **PubMed identifier:** | Type the PubMed identifier. |

Figure 49: The seventh step of the wizard. The user provides one or more references related to the in silico trial


References related to the trial
References related to the experiment

In this step you have to provide all the references which are related to this experiment.
▪ Skip this step

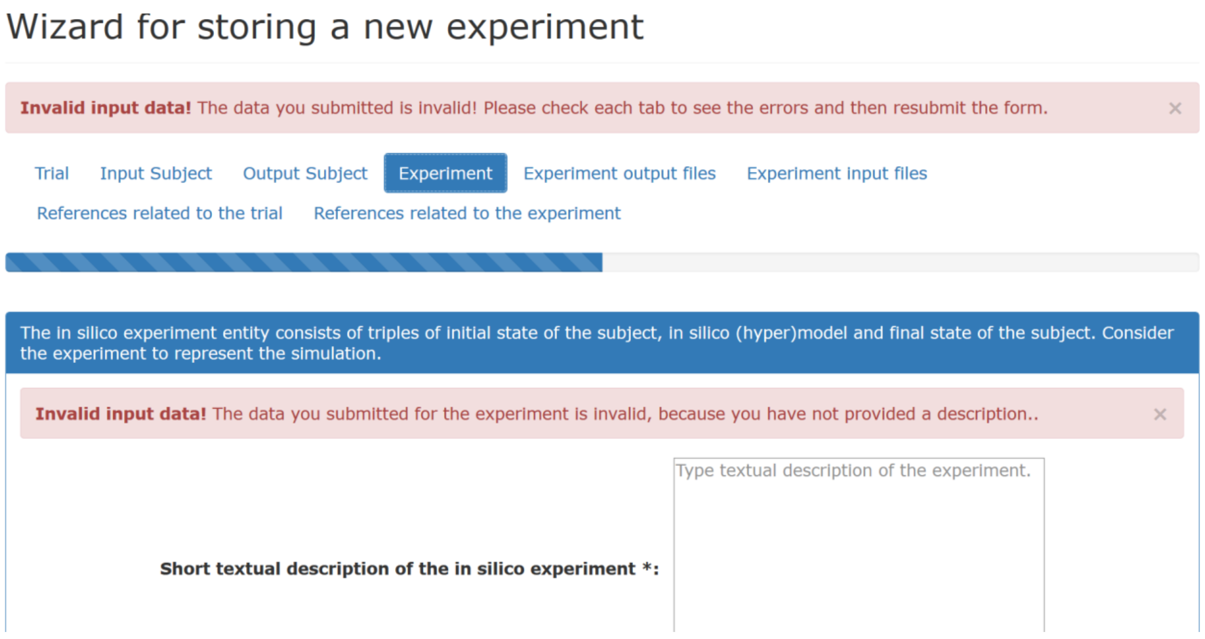
**Heads up!** You can skip this step if you want. But if you want to provide references which are related to this experiment, please be aware that the "Name given to the resource" fields should not be empty before you submit the data of the new experiment.

| | |
|---|---|
| **Name given to the resource:** | Type the name given to the resource. |
| **Type of the resource:** | book |
| **The creator(s)/author(s) of the resource:** | Type the creator(s)/author(s) of the resource |
| **Date of formal issuance:** | --- --- --- |
| **The bibliographic citation of the resource:** | Type the bibliographic citation of the resource |
| **The related resource that this resource is part of:** | Type the related resource that this resource is part of |
| **The related resource from which the described resource is derived from:** | Type the related resource from which the described resource is derive |
| **Digital object identifier of the resource:** | Type the digital object identifier of the resource. |
| **PubMed identifier:** | Type the PubMed identifier. |

**Figure 50: The eighth step of the wizard. The user provides one or more references related to the in silico experiment**

As shown in Figures 43-50, all the information regarding the newly created *in silico* experiment can be provided through a single page which consists of different tabs (one tab for each wizard step). After the provision of all the data of the experiment, the corresponding information will be stored in the MySQL database of the *In Silico* Trial Repository. It should be noted that the user is able to store dynamically in the same page variable number of references, input and output files.

As illustrated in Figure 51, in case of invalidity concerning the data of the new *in silico* experiment, the *In Silico* Trial Repository notifies the user accordingly with error messages in the corresponding tabs of the page.


Wizard for storing a new experiment

**Invalid input data!** The data you submitted is invalid! Please check each tab to see the errors and then resubmit the form.

Trial | Input Subject | Output Subject | Experiment | Experiment output files | Experiment input files | References related to the trial | References related to the experiment

The in silico experiment entity consists of triples of initial state of the subject, in silico (hyper)model and final state of the subject. Consider the experiment to represent the simulation.

**Invalid input data!** The data you submitted for the experiment is invalid, because you have not provided a description..

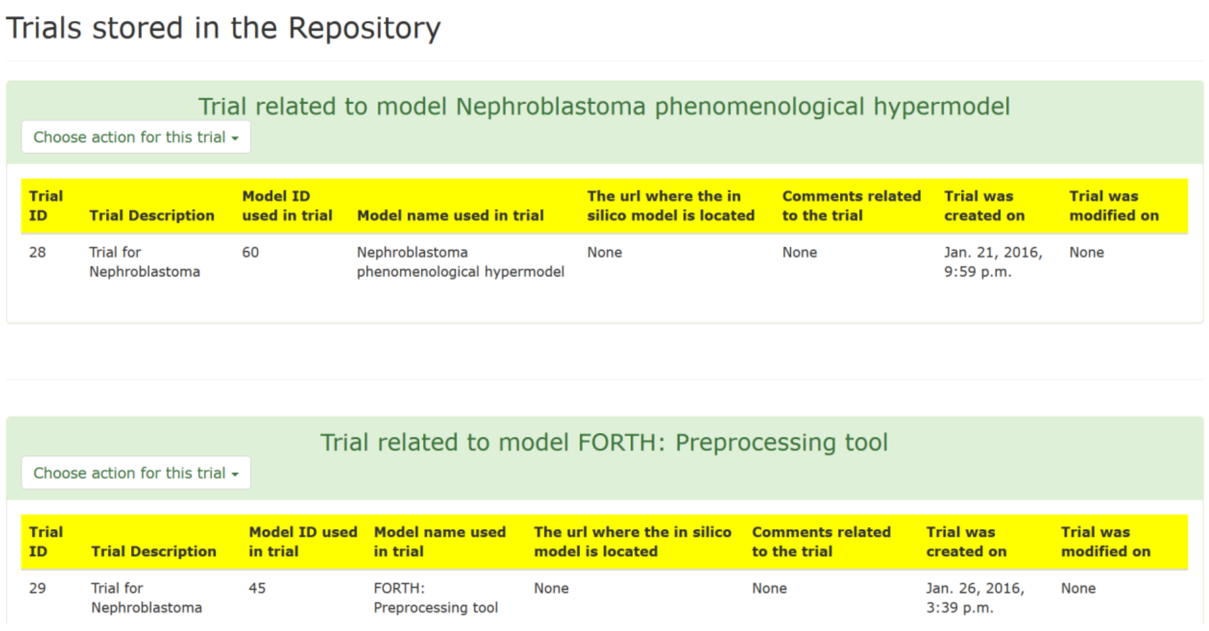
**Short textual description of the in silico experiment *:** Type textual description of the experiment.

**Figure 51: The wizard informs the user about the invalidity of the data when submitting the form**

### 6.3.2 Browsing the content of the *In Silico* Trial Repository

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. As discussed in chapter 6.2 "Architecture of the *In Silico* Trial 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 52 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.

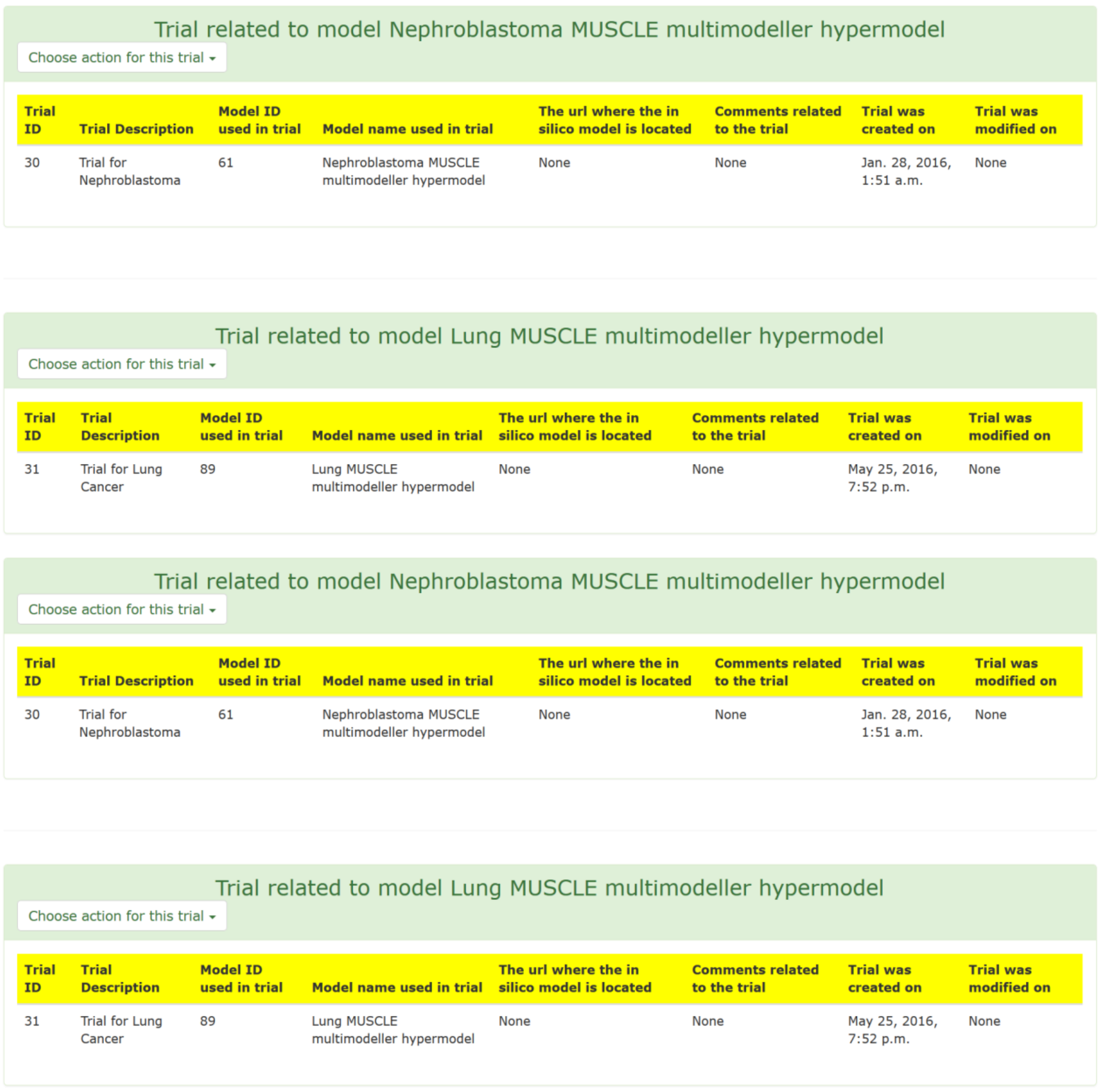

Trials stored in the Repository

Trial related to model Nephroblastoma phenomenological hypermodel

| 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

| 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

| 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

| 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

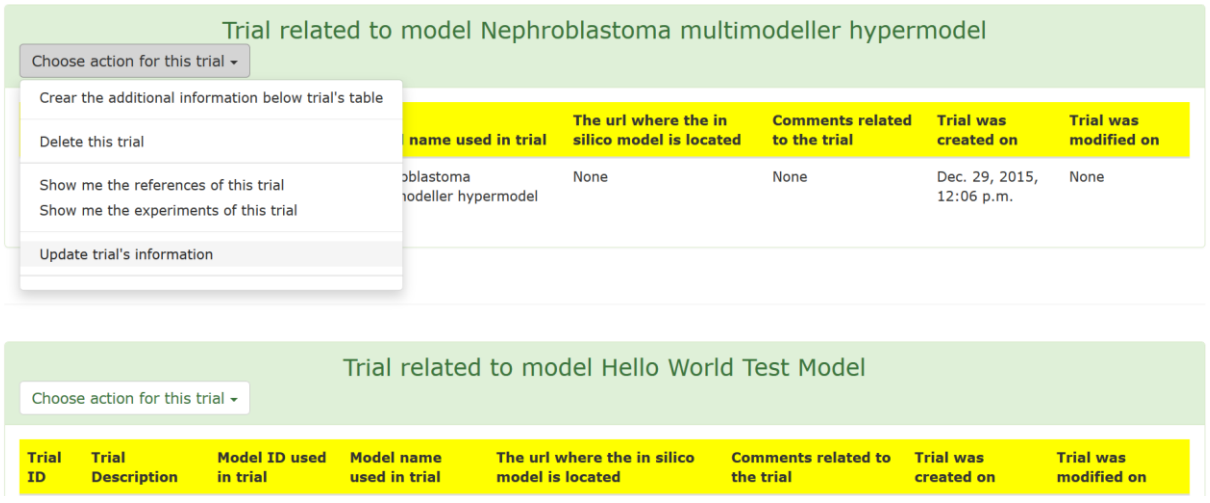
| Trial ID | Trial Description | Model ID used in trial | Model name used in trial | The url where the in silico model is located | Comments related to the trial | Trial was created on | Trial was modified on |
|---|---|---|---|---|---|---|---|
| 30 | Trial for Nephroblastoma | 61 | Nephroblastoma MUSCLE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

Trial related to model Lung MUSCLE multimodeller hypermodel

| 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 52: Part of the page of the In Silico Trial Repository which indicates the available in silico trials**

For each *in silico* trial depicted in Figure 52, the user may apply many actions, such as deleting the trial, updating the corresponding information, or even viewing the related references and *in silico* experiments. With respect to this, Figure 53 presents the available actions that can be applied and Figure 54 presents the page where the user is being redirected for updating the description of the trial which is related to Nephroblastoma Multimodeller Hypermodel.



Trial related to model Nephroblastoma multimodeller hypermodel

Choose action for this trial ▾

- Crear the additional information below trial's table
- Delete this trial
- Show me the references of this trial
- Show me the experiments of this trial
- Update trial's information

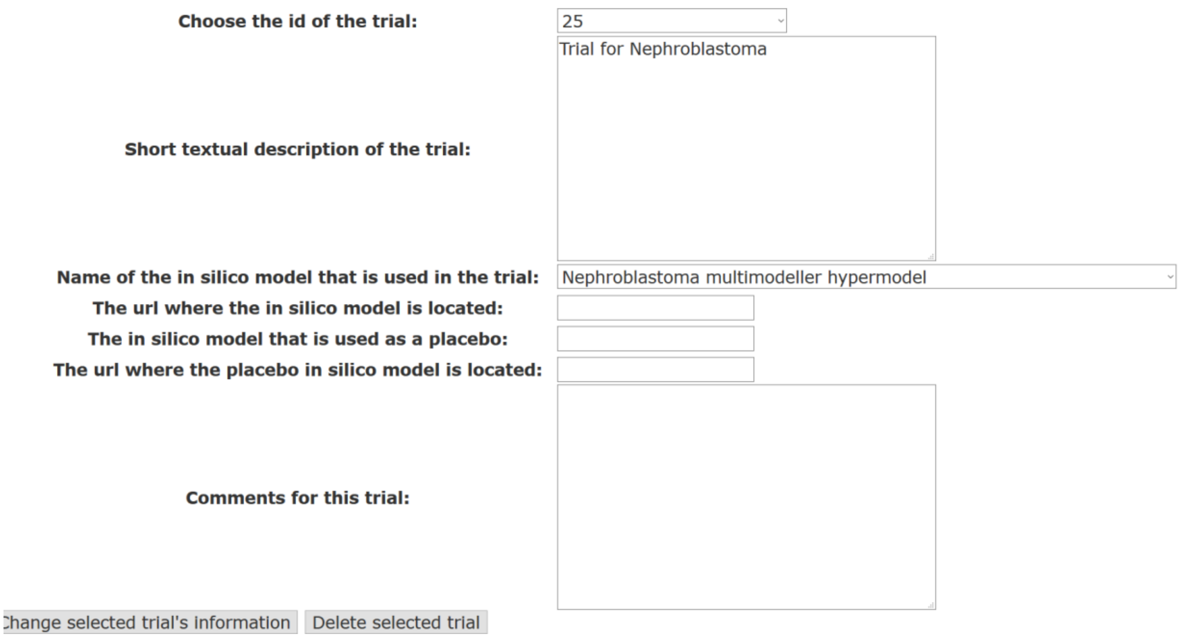
| name used in trial | The url where the in silico model is located | Comments related to the trial | Trial was created on | Trial was modified on |
|---|---|---|---|---|
| oblastoma odeller hypermodel | None | None | Dec. 29, 2015, 12:06 p.m. | None |

Trial related to model Hello World Test Model

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 |
|---|---|---|---|---|---|---|---|

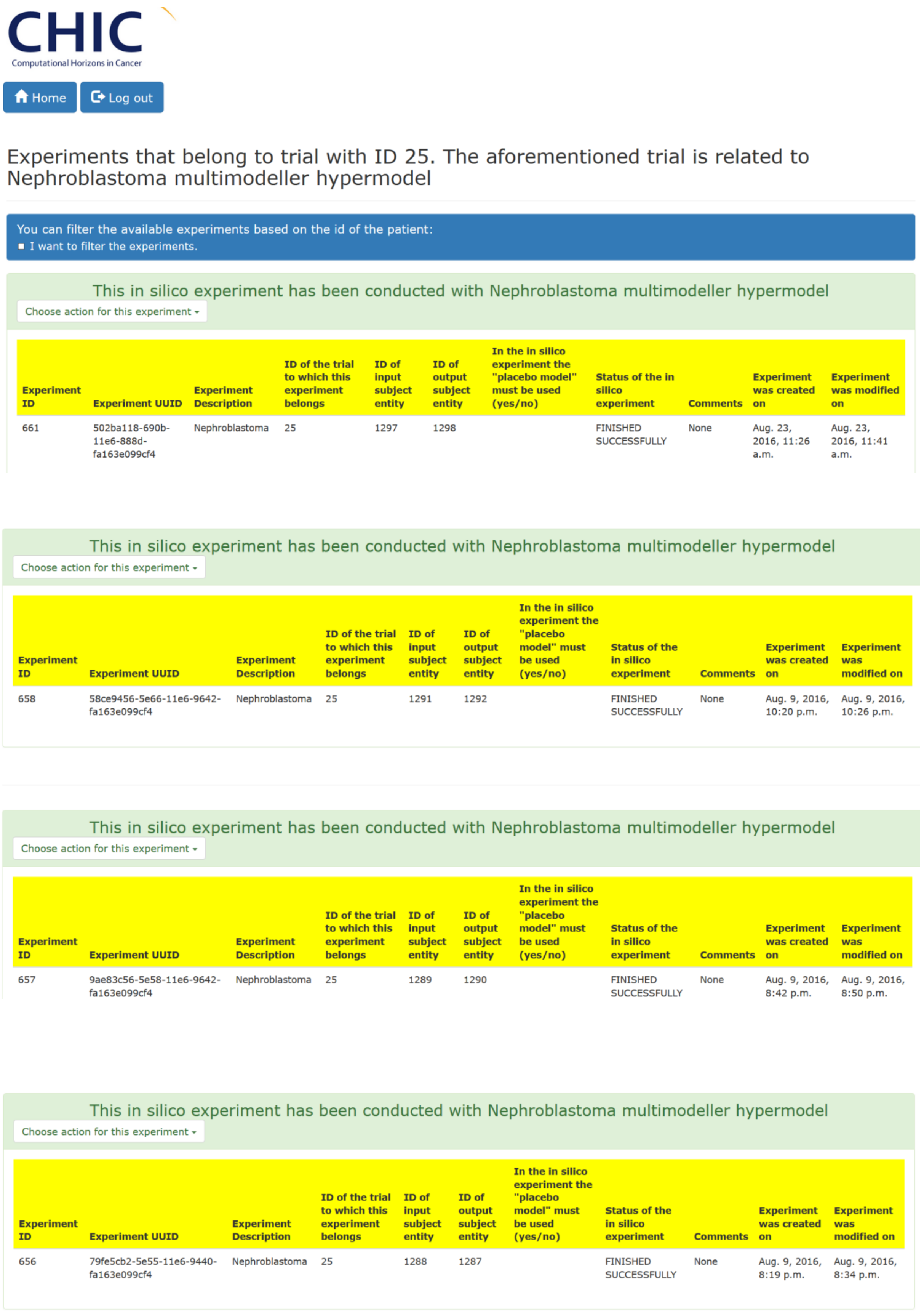
**Figure 53: The user is going to update the description of the in silico trial which is related to Nephroblastoma Multimodeller Hypermodel**



| | |
|---|---|
| Choose the id of the trial: | 25 |
| Short textual description of the trial: | Trial for Nephroblastoma |
| Name of the in silico model that is used in the trial: | Nephroblastoma multimodeller hypermodel |
| The url where the in silico model is located: | |
| The in silico model that is used as a placebo: | |
| The url where the placebo in silico model is located: | |
| Comments for this trial: | |

**Figure 54: The page where the user applies changes to the trial related to Nephroblastoma Multimodeller Hypermodel**

After choosing the *in silico* trial of his 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 55, 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. Figure 56 presents the most critical part of this workflow, in which the user downloads the output data of the last *in silico* experiment that has been conducted with Nephroblastoma Multimodeller Hypermodel (for detailed information concerning the Nephroblastoma Multimodeller Hypermodel (or Nephroblastoma Integrated Hypermodel) please refer to the deliverable D6.3 "Initial Standardized Cancer Hypermodels") [30].



Home Log out

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. |

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 |
|---|---|---|---|---|---|---|---|---|---|---|
| 658 | 58ce9456-5e66-11e6-9642-fa163e099cf4 | Nephroblastoma | 25 | 1291 | 1292 | | FINISHED SUCCESSFULLY | None | Aug. 9, 2016, 10:20 p.m. | Aug. 9, 2016, 10:26 p.m. |

This in silico experiment has been conducted with Nephroblastoma multimodeller hypermodel

Choose action for this experiment

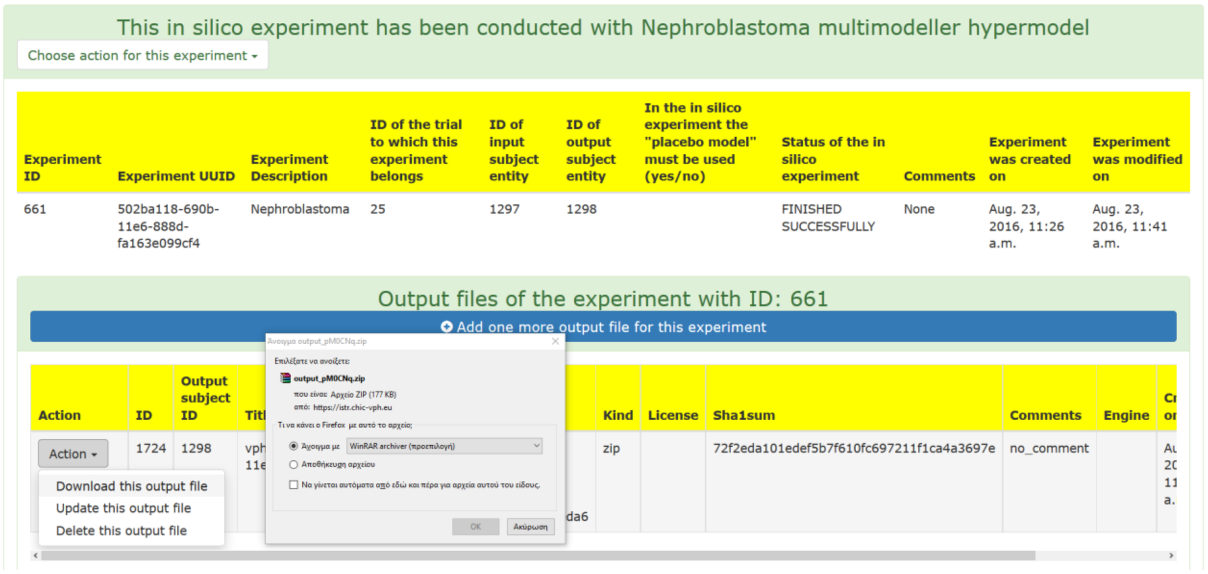
| Experiment ID | Experiment UUID | Experiment Description | ID of the trial to which this experiment belongs | ID of input subject entity | ID of output subject entity | In the in silico experiment the "placebo model" must be used (yes/no) | Status of the in silico experiment | Comments | Experiment was created on | Experiment was modified on |
|---|---|---|---|---|---|---|---|---|---|---|
| 657 | 9ae83c56-5e58-11e6-9642-fa163e099cf4 | Nephroblastoma | 25 | 1289 | 1290 | | FINISHED SUCCESSFULLY | None | Aug. 9, 2016, 8:42 p.m. | Aug. 9, 2016, 8:50 p.m. |

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 |
|---|---|---|---|---|---|---|---|---|---|---|
| 656 | 79fe5cb2-5e55-11e6-9440-fa163e099cf4 | Nephroblastoma | 25 | 1288 | 1287 | | FINISHED SUCCESSFULLY | None | Aug. 9, 2016, 8:19 p.m. | Aug. 9, 2016, 8:34 p.m. |

**Figure 55: Information related to the last four simulations (*in silico* experiments) of Nephroblastoma multimodeller hypermodel**


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. |

Output files of the experiment with ID: 661

Add one more output file for this experiment

| Action | ID | Output subject ID | Titl | | Kind | License | Sha1sum | Comments | Engine | Cr on |
|---|---|---|---|---|---|---|---|---|---|---|
| Action ▾ | 1724 | 1298 | vph 11e | da6 | zip | | 72f2eda101edef5b7f610fc697211f1ca4a3697e | no_comment | | Au 20 11 a. |

Άνοιγμα output_pM0CNq.zip

Επιλέξατε να ανοίξετε:

output_pM0CNq.zip

που είναι: Αρχείο ZIP (177 KB)

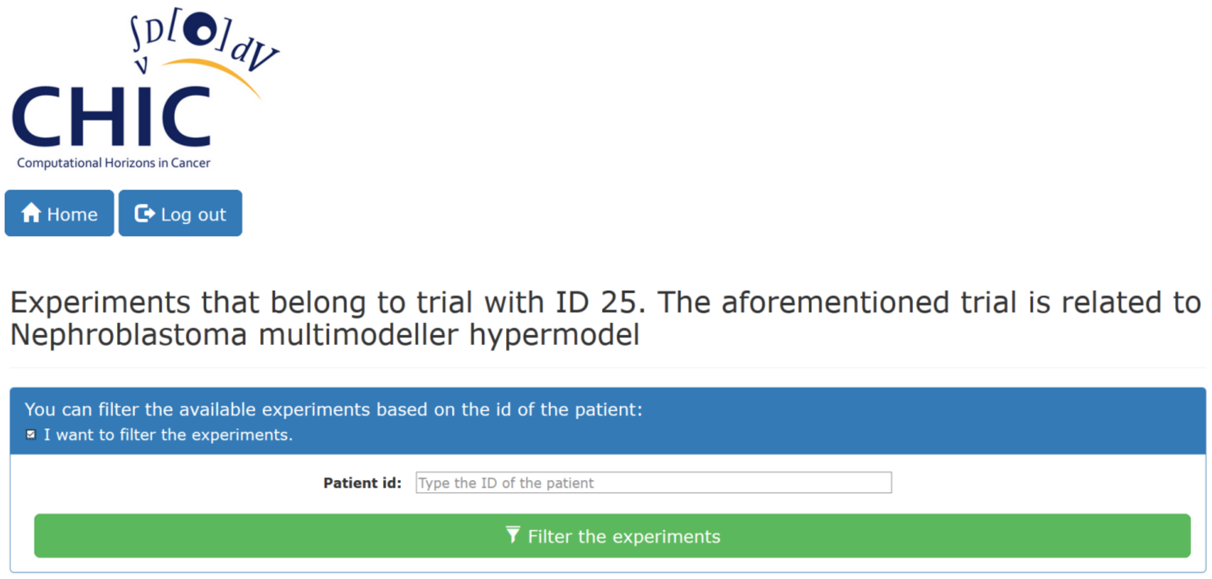
από: https://istr.chic-vph.eu

Τι να κάνει ο Firefox με αυτό το αρχείο;

- [✓ Άνοιγμα με] WinRAR archiver (προεπιλογή)
- Αποθήκευση αρχείου
- ☐ Να γίνεται αυτόματα από εδώ και πέρα για αρχεία αυτού του είδους.

**Figure 56: The user downloads the output data of the last in silico experiment which is related to Nephroblastoma Multimodeller Hypermodel**

Finally, since the number of the available *in silico* experiments stored in the Repository may span from tens to millions, the filtering of the executions is a necessity. In respect to this, Figure 57 presents a screenshot of the page related to the filtering of the experiments based on the patient used in the simulation. As shown in the aforementioned figure, in order for the user to filter the content of the Repository, they could just provide a pseudonymized identification of the patient used in the desired simulations.



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.

Patient id: Type the ID of the patient

**Figure 57: The user is able to filter the available in silico experiments by providing a pseudonymized identification of the patient used in the simulation**

## 6.4 *In Silico Trial Repository web services*

The *in silico* trial repository makes use of RESTful web services which are based on the entity relationship diagram depicted in Figure 39. *In silico* trial's repository RESTful 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 *in silico* trial repository's web services. 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. This chapter has been included in this deliverable with the aim of being a reference point for all the other CHIC components in order for them to be able to access and modify the content of the *In Silico* Trial Repository. Nonetheless, this documentation is not considered to be the final, since minor changes and updates are going to be applied to the schema of the Repository based on the new requirements that may arise till the end of the CHIC project.

**Trial**

**The following web services (tables 42-47) 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 42: 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 43: Information for calling getAllTrials web service**

| getAllTrials | 🔒 |
|---|---|
| Description | This method returns the corresponding descriptive information of all the trials stored in *in silico* 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 | | |
| The keys of the JSON object returned by method getAllTrials are as many as the different trials stored in the *in silico* 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 tr_trial entity (see figure 39) and each value of the nested JSON object represents the information of the corresponding column. | | |

**Table 44: 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 | | |
| The keys of the JSON object returned by method getUserTrials 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 id, description, model_id , model_url, placebo_model_id, comment, created_on, created_by, modified_on, modified_by. | | |

**Table 45: 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 | | |
| The JSON object returned by method getTrialById 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 46: 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 47: 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 – query string parameter) | id= | Required – the id of the trial |
| 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> |

**Experiment**

**The following web services (tables 48-57) 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 48: 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 in silico experiment |
| | subject_id_out= | Required – the id of the subject that is produced after the execution of the new *in silico* 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 49: 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 | | |
| The keys of the JSON object returned by method getUserExperiments 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 id, trial, description, subject_id_in , subject_id_out, placebo, status, comment, uuid, created_on, created_by, modified_on, modified_by. The value which corresponds to the key trial is another json object with keys id and model_id. The value which corresponds to the key subject_id_out is another json object with keys id, subject_external_id and description. The value which corresponds to the key subject_id_in is another json object with keys id, subject_external_id and description. | | |

**Table 50: 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 | | |
| The keys of the JSON object returned by method getUserPendingExperiments 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 id, trial, description, subject_id_in , subject_id_out, placebo, status, comment, uuid, created_on, created_by, modified_on, modified_by. The value which corresponds to the key trial is another json object with keys id and model_id. The value which corresponds to the key subject_id_out is another json object with keys id, subject_external_id and description. The value which corresponds to the key subject_id_in is another json object with keys id, subject_external_id and description. | | |

**Table 51: 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 | | |
| 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. | | |

**Table 52: 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 – 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 getExperimentById has thirteen keys named id, uuid, trial_id, description, subject_id_in, subject_id_out, placebo, status, comment, created_on, created_by, modified_on and modified_by, and twelve values associated with those keys. |

**Table 53: Information for calling getExperimentByUuid web service**

| getExperimentByUuid | | 🔒 |
|---|---|---|
| Description | This method returns the experiment and the related information stored under the uuid (description, subject_id_in, subject_id_out, placebo, status, comment, 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) | uuid= | Required – the uuid 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 getExperimentByUuid has thirteen keys named id, uuid, trial_id, description, subject_id_in, subject_id_out, placebo, status, comment, created_on, | | |

| |
|---|
| created_by, modified_on and modified_by, and thirteen values associated with those keys. |

**Table 54: 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 55: 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 | | |
| The keys of the JSON object returned by method getExperimentsByStatus 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 tr_experiment entity (see figure 39) and each value of the nested JSON object represents the information of the column. | | |

**Table 56: 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 57: 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> |

**Miscellaneous parameter**

**The following web services (tables 58-63) 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 58: 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 59: 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 | | |
| The keys of the JSON object returned by method getAllMiscellaneousParameters are as many as the different miscellaneous parameters that are stored in the *in silico* 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 tr_miscellaneous_parameter entity (see figure 39) and each value of the nested JSON object represents the information of the corresponding column. | | |

**Table 60: 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 | | |
| The keys of the JSON object returned by method getUserMiscellaneousParameters 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 id, experiment_id, hypomodel_parameter_id ,hypermodel_parameter_id , value, created_on, created_by, modified_on, modified_by. | | |

**Table 61: 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 | experiment_id= | Required – the id of the experiment |

| | | |
|---|---|---|
| the URL – 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 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. | | |

**Table 62: 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 63: 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> |

**Subject**

**The following web services (tables 64-68) 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 64: 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 65: 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 66: 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 | | |
| The keys of the JSON object returned by method getAllSubjects are as many as the different subjects that are stored in the *in silico* 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 tr_subject entity (see figure 39) and each value of the nested JSON object represents the information of the corresponding column. | | |

Table 67: 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 | | |
| The keys of the JSON object returned by method getUserSubjects 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 id, description, subject_external_id , external_url, comment, created_on, created_by, modified_on, modified_by. | | |

Table 68: 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. | | |

**Reference**

**The following web services (tables 69-75) 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 69: 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 70: 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 | | |
| The keys of the JSON object returned by method getAllTrReferences are as many as the different references that are stored in the *in silico* 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 tr_reference entity (see figure 39) and each value of the nested JSON object represents the corresponding information of the column. | | |

**Table 71: 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 |
|---|
| The keys of the JSON object returned by method getTrReferencesByTrialId 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 tr_reference entity (see figure 39) and each value of the nested JSON object represents the information of the corresponding column. |

**Table 72: 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 | | |
| 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. |

**Table 73: 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 74: 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 75: 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> |

**File**

**The following web services (tables 76-81) 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 76: 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<br><br>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 77: 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 78: 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 79: 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 | | |
| 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. | | |

**Table 80: 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 | | |
| 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. | | |

**Table 81: 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 | | |
| 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. | | |

# 7 Semantic Metadata Management

## 7.1 Relevant CHIC Resources

The semantic framework in CHIC is dedicated to the support of providing machine-processable descriptions of CHIC resources. CHIC resources are computational artefacts that are created, stored and managed by the varied components of the CHIC platform (in particular the Model Repository, the Hypermodel Editor, and the Clinical Data Repository).

CHIC resources include prominently encoded mathematical models of two different categories, the CHIC hypomodels and the CHIC hypermodels which result from the assembly of relevant hypomodels. A large motivation for the metadata description of these resources in CHIC is to facilitate the relevant and well informed combination of hypomodels. This is the reason why hypomodels are the primary CHIC resources handled in the metadata semantic framework. The CHIC semantic framework also intends to recording descriptions of the hypermodels and their parameters, consistently with combined hypomodels. To this end, such descriptions are also thought to be semantically interoperable with descriptions of data used in running the models in the CHIC platform and, in particular, clinical data.

Consequently, the following are the main types of CHIC resources that are relevant to the semantic treatment of their descriptions:

- Model
    - Hypomodel
    - Hypermodel
- Model parameter
    - Input and output
    - Biologically meaningful parameters
- Clinical data objects

## 7.2 *Annotation of Resources in order to make interpretation explicit and usage of Processable Reference Knowledge Models (Ontologies)*

The method and technology used in order to record the descriptions of the CHIC resources consists in using language independent conceptual structures so as to generate a formal vocabulary that will ensure the explicitness of a number of information elements. The primary types of encoded information are the domain specific interpretation of the objects and the record of information in the model code. Domain interpretation is usually not encoded explicitly in models as their encoding language is not designed to handle the required qualitative descriptions, however, the range of non-encoded information may be somewhat broader. For example, units of measurements for quantities ascribed to parameters may remain only implicit and thus not explicitly handled by the software required to process the model. In this case, many models come with parameters that have an assigned default value and take a range of numerical values standing for quantities. While the model is designed to expect an order of values that is consistent with an implicit unit or scale, this is not explicitly encoded and remains a hidden assumption which is in the best case documented as part of an informal description of the model.

In order to make informal descriptions of models explicit and formally encoded, the process of annotation consists in representing a CHIC resource as an object in its own rights and ascribing to it a range of attributes and characteristics using a formal knowledge representation language. The result is a simple description of a resource that can then be integrated within a larger range of resource descriptions. The overall result is a semantically integrated set of descriptions of CHIC resources that can then be interrogated to produce comparisons and elicit relationships based on the range of encoded information. Consequently, a use case for this sort of information is the construction of categorizations for the models and their parameters and another example is the use of such categorizations to elicit relationships of consistency and correspondence between the parameters or between the models. For example, the usage of controlled vocabularies and semantically encoded constraints for the units regarding the concentration of drugs, may facilitate the check of consistency when linking the parameters of different hypomodels.

In order to accomplish the aforementioned goal, the CHIC semantic framework uses formalized theories of domain specific descriptions, called 'ontologies' from which the CHIC framework may derive unique ways of referring to descriptive terms. Moreover, through consistent reuse and interlinking between the aforementioned terms, the semantic integration is ensured. In addition to this, the use of ontologies combined with the use of unique identifiers for CHIC resources results in a knowledge representation which is a form of data. Such semantic data, named 'metadata', from the standpoint of the CHIC resources description, is created, stored and retrieved within the semantic parts of the overall integrating CHIC infrastructure.

## *7.3 Semantic Components and RICORDO Architecture*

As shown in Figure 58, (image obtained from http://www.ricordo.eu), the core of the semantic infrastructure in CHIC rests on the reuse and adaptation of the RICORDO third-party solution. The semantic infrastructure provides layers of services between backend semantic data storage and ontology storage in order to interact with knowledge management tools in an application context. CHIC defines an application area in which the sources of data are the CHIC model and clinical data repositories and CHIC user interfaces take on the role of consumers and producers of metadata through the use of semantic services.

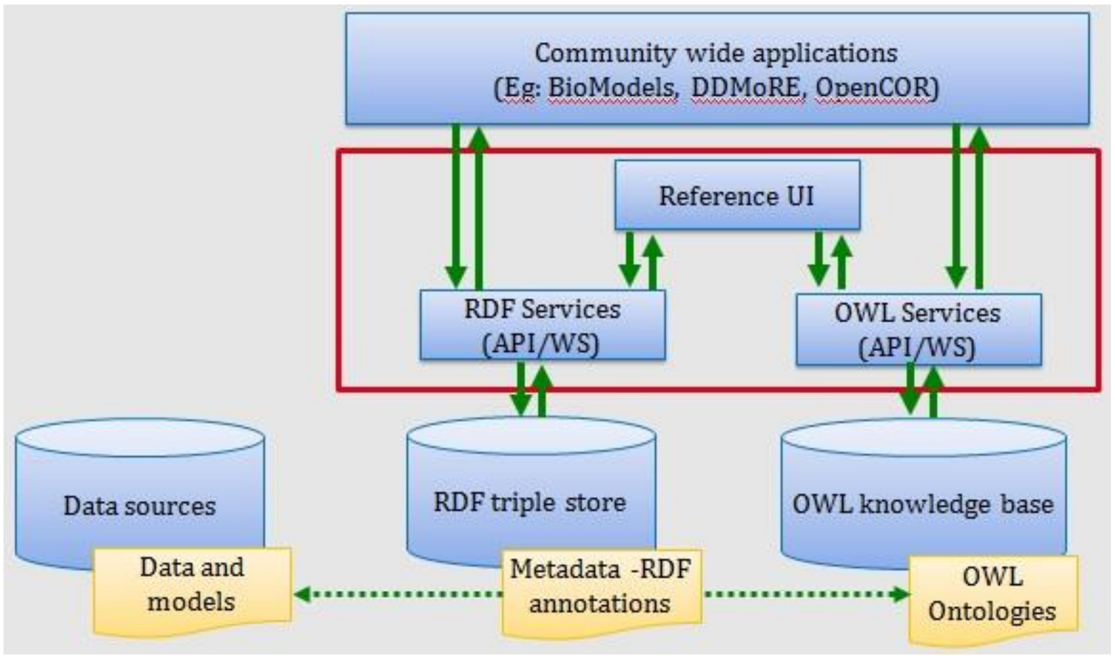


**Figure 58: (Image obtained from http://www.ricordo.eu.). The overall RICORDO architecture in relation to envisioned (non-CHIC related) application contexts**

The semantic part of the CHIC infrastructure has a modular architecture which consists of the two main components, the Annotations Store and the Knowledge Database. Both components are described in the following chapters.

### 7.3.1 Annotations Store

An annotation store is a database in which annotation statements are registered by using an encoded dedicated format. CHIC annotations are encoded in the Resource Description Framework [16], which is a W3C standard for the description of resources. RDF is serialised in a number of syntaxes but the choice of syntax does not affect the storage of annotation. In principle, an annotation is a triple (a 3-placeslist) such that:

- The first element is the 'subject' of annotation
- The second element is 'predicate', a binary relation that holds between the first element and the third
- The third element is the 'object' of annotation.

The objects in questions are unique identifiers in the form of URIs which most of the time are URLs. For example, in order to express the fact that a given model is a model addressing a modelling question in oncology, an annotation statement would require three objects:

- An object referring to the specific model, for example, <http:://example.com#ThisModel>
- An object representing the relation linking a model to a modelling question area, for example <http://example.com#modelsQuestionIn>
- An object represenating the oncology field, for example, <http://www.example.com#Oncology>

An annotation statement can be written as follows in order to represent the intended description:

<http:://example.com#ThisModel> http://example.com#modelsQuestionIn
<http://www.example.com#Oncology> .

The aforementioned statement would be stored and be retrieved in full or in part. For instance, the object could be easily retrieved, given the subject and the predicate.

An RDF database storing annotations is called a 'triple store'. There are various implemented systems providing triple storage which implement the standards for querying and manipulating RDF data called SPARQL [17]. These systems implement basic functionalities for the Creation, Retrieval, Update and Deletion of data.

### 7.3.2 Knowledge Database

A knowledge base is a form of storage for the knowledge contained in the formal definition of an ontology. The language used in specifying ontologies in CHIC is the Ontology Web Language (OWL) [18], which is a W3C standard. OWL is embedded by design in the RDF technology. It comes with built in language elements for describing kinds of objects and their relations. OWL is domain independent and used to formalize the ontology in a given domain. It provides the ability to define 'classes' of objects, their instances and to define further specific classes based on logical restrictions using domain relationships.

A particularity of knowledge bases is that they include an inference system in the form of reasoning. The main contribution of such inference system is to perform logical computation based on the equivalence and the subsumption between classes and to categorize instances; it is also a mechanism for checking the logical consistency of an ontology, i.e., that incompatible facts are not derivable from it. There is also a variety of reasoners that are available which differ in their performance and the complexity of logical fragments of OWL on which they perform.

### 7.3.3 Annotation Store (RDF Triplestore)

The CHIC semantic infrastructure uses the Fuseki2 server as RDF triple store [19]. Fuseki2 implements the SPARQL protocols for basic management operations. In CHIC, an instance of Fuseki2 is used as backend storage for annotations.

### 7.3.4 Knowledge Base (OWL Knowledge Base)

The CHIC semantic infrastructure uses the RICORDO OWLKB application [20] in order to instantiate and persist an OWL knowledge base. The RICORDO OWLKB uses a third party library (OWL API) [21] and provides an extra layer of services around the knowledge base. It is deployed in CHIC using the Elk reasoner [22].

## *7.4 Semantic Services*

### 7.4.1 RDF Services

In addition to direct SPARQL endpoint access to the triple store, semantic services are also being provided. By using the SPARQL language and protocol, the triple store can be accessed, queried and updated programmatically. The RICORDO RDF store application [23] is used as a middle man to simplify these operations by providing RESTfull web services that can be invoked through the use of simpler templates. Templates provide a predefined mechanism for performing selected operations under domain specific assumptions (for example, retrieval of models) and are accessed through a number of documented parameters.

There are both annotation management services to allow for the creation and addition of annotations to the CHIC RDF store and also query services that allow for the retrieval of annotation elements matching specified query criteria.

Rdfstore 2.0 is the so-called Ricordo metadata wrapper. It serves as a messenger between SPARQL endpoint and end-user. The motivation behind Rdfstore 2.0 is to alleviate the need to handle SPARQL syntax and make it simpler and more straightforward to deal with metadata. This is done with a system of templates, which are customized at the organizational level.

The end-user tells the appropriate systems team: ``I want a form that'll let me query the database for X.'' The team creates a template for that query. Now the end-user can select that template and query the database for X by filling out a simple form, no SPARQL required.

### 7.4.2 Installation Requirements

- Java and Java runtime: Rdfstore 2.0 requires Java Runtime and a Java compiler for its installation.
- RDF Triple Store: Rdfstore 2.0 assumes that a triple store is running and has exposed a SPAQRL endpoint. Rdfstore 2.0 has been used in combination with the following:
  - Virtuoso Open-source Edition (GNU GPL license)

    http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/
  - Fuseki and Fuseki2 (Apache 2.0 license)

    https://jena.apache.org/documentation/fuseki2/
  - Specific configuration information is provided below for these.

### 7.4.3 Installation Steps

1. Rdfstore 2.0 is available for download from:

http://github.org/semitrivial/rdfstore

Command "git clone" can be used, or any other means, to copy this repository locally.

2. Within the directory containing the copy of the repository, compile Rdfstore using "make" (or manually: "javac -g Rdfstore.java")

3. Within that directory, run "java Rdfstore -help" to get help on the command line arguments, or see further below.

In order for Rdfstore 2.0 to be useful, *templates* need to be present in the relevant subdirectory, as discussed below.

### 7.4.4 Running RDFstore 2.0

Rdfstore 2.0 can be run with the following command-line arguments:

**Table 82: RDFstore command-line options**

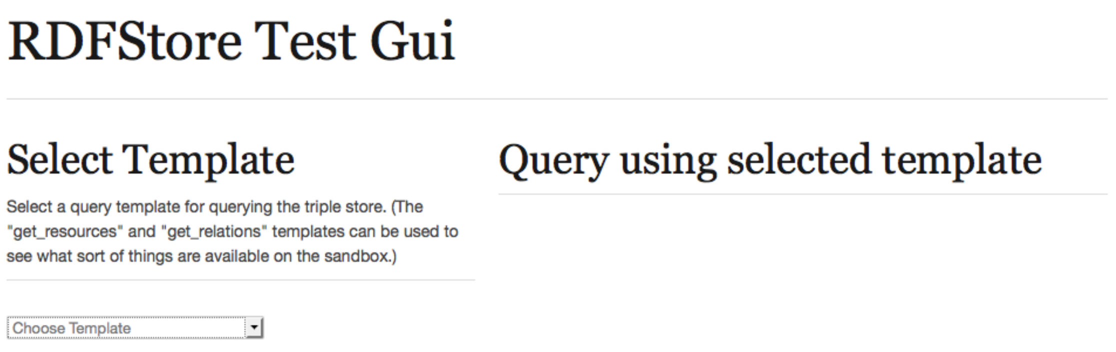
| Argument | Description | Example or default |
|---|---|---|
| -templates <directory> | Specifies the location of the directory containing Rdfstore templates. | Default: ./templates |
| -endpoint <URL> | Specifies the SPARQL query endpoint location for the coordinated triple store. | http://localhost:3030:/chic/query?force-accept=text%2Fplain&output=tsv&query=SELECT... http://localhost:3030/chic/query?force-accept=text%2Fplain&output=tsv&query= |
| -method <GET or POST> | Specifies which HTTP method your SPARQL endpoint uses. | Default: GET |
| -update <URL> | Specifies a separate address | Default: copies "endpoint" |

| | | |
|---|---|---|
| | for updates, when applicable. | |
| -updatemethod <GET or POST> | Specifies the HTTP method for a separate address for updates, when applicable. | Default: copies "method" |
| -format <format> | A string, containing "%s". The %s will be replaced by the query itself, useful for things like triplestore-specific preambles, etc. | Default: %s |
| -port <number> | Specifies which port Rdfstore will listen for connections on. | Default: 20060 |
| -help | Displays a help screen. | |

### 7.4.5 Simple GUI

While the goal of Rdfstore 2.0 is to deliver an API, it comes with a built-in simple GUI mainly for the purpose of illustration and education. When Rdfstore is running, the GUI can be accessed at http://HOST:PORT/gui/. For example, if HOST is "localhost" and PORT is "20060", the GUI would be accessible within a Web browser at:

http://localhost:20060/gui/


RDFStore Test Gui

Select Template

Select a query template for querying the triple store. (The "get_resources" and "get_relations" templates can be used to see what sort of things are available on the sandbox.)

Query using selected template

Figure 59: Simple Rdfstore 2.0 GUI

### 7.4.6 Template System

SPAQRL 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.

For instance, the SPARQL query to find all things which are "part-of" the class "acids" might look like so:

```
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. 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.

#### 7.4.6.1 Advanced Template Commands

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

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

The command,

# Preprocessor0 = http://open-physiology.org: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 Proprocessor0 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.

#### 7.4.6.2 RDFStore API

RDFStore has a dynamic API. The API is defined by the templates loaded when RDFStore is started. For each template, there is a corresponding API command. If the template is named X.txt, and depends on parameters [0], [1], and [2], then the API command looks like:

http://example.com:20060/X/?0=fill_this_in&1=also_fill_this&2=this_too

### 7.4.7 Low Level Services

We refer to services as low level services when they are mere syntactic variations on basic SPAQRL commands. There are three kinds: commands to add a record and commands to delete a record as well as commands to query records.

#### 7.4.7.1 Query

A low level command allows wrapping (URL encoded) SPARQL queries.

For example: select ?x ?y ?z where {?x ?y ?z} limit 10

Can be invoked as:

http://localhost.org:20060/Raw_SPARQL/?0=select%20%3Fx%20%3Fy%20%3Fz%20where%20{%3Fx%20%3Fy%20%3Fz}%20limit%2010

#### 7.4.7.2 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

#### 7.4.7.3 Deletion

A low level command allows inserting a triple (SPAQRL INSERT DATA):

http://open-physiology.org:20060/Delete_Triple_%28Fuseki%29/?0=a&1=b&2=c

### 7.4.8 Specific TRIPLEStores Documentation

Specific documentation for using RDFStore with individual triplestores: Virtuoso and Fuseki.

#### 7.4.8.1 Virtuoso

When your server is running Virtuoso, by default the SPARQL endpoint is on port 8890. In the following documentation, we'll assume you keep that default; if you change it to another port, then change everything accordingly.

##### 7.4.8.1.1 Queries

Depending on what format you'd like the results in, you can use one of the following strings as the "endpoint" when running Rdfstore.

*7.4.8.1.1.1 JSON format*

Endpoint string:

http://localhost:8890/sparql?default-graph-uri=&format=application%2Fsparql-results%2Bjson&timeout=0&debug=on&query=

Minimum working example commandline:

```
java Rdfstore -endpoint "http://localhost:8890/sparql?default-graph-
uri=&format=application%2Fsparql-results%2Bjson&timeout=0&debug=on&query="
```

###### *7.4.8.1.1.2 HTML Format*

Endpoint string:

http://localhost:8890/sparql?default-graph-uri=&format=text%2Fhtml&timeout=0&debug=on&query=

Minimum working example commandline:

```
java Rdfstore -endpoint "http://localhost:8890/sparql?default-graph-
uri=&format=text%2Fhtml&timeout=0&debug=on&query="
```

###### *7.4.8.1.1.3 Other formats*

Virtuoso makes a lot of other formats available. To see the list, go to this Virtuoso SPARQL documentation page and scroll down to "16.2.3.3.3. Response Format".

For each listed content type, the general formula for the endpoint string is:

http://localhost:8890/sparql?default-graph-uri=&format=(content type)&timeout=0&debug=on&query=

where (content type) is replaced by the url-encoded mimetype from the above link.

**Example:**

Suppose you want the format as "application/x-turtle".

Urlescape to get: "application%2Fx-turtle".

The endpoint string is:

http://localhost:8890/sparql?default-graph-uri=&format=application%2Fx-turtle&timeout=0&debug=on&query=

##### 7.4.8.1.2 Adding Triples

There are two things to know to set up triple-authoring via Rdfstore via Virtuoso.

###### *7.4.8.1.2.1 Must specify graph*

When adding a triple in Virtuoso, it is necessary to specify which graph it goes in. Here's an example "Insert_Triple.txt" template:

```
# 0 = Graph IRI
# 1 = Subject IRI
# 2 = Predicate IRI
# 3 = Object IRI
INSERT INTO <[0]>
{
  <[1]>
  <[2]>
  <[3]>
}
```

Note that the four comments at the beginning are just to tell the GUI what placeholder text to put in the blank fields; they aren't strictly necessary.

###### *7.4.8.1.2.2 Must grant permission*

By default, Virtuoso forbids triple-insertion via SPARQL endpoint. If triple-insertion is forbidden, then your triple-insert Rdfstore templates will fail.

Here's how to enable triple-insertion via SPARQL endpoint:

- Connect to Virtuoso's ISQL console. From the command line on the machine where Virtuoso is running, this is usually done with the "isql" command (or "isql-vt" on Ubuntu).
- Issue the command:

```
GRANT execute ON SPARQL_INSERT_DICT_CONTENT TO "SPARQL";
```

- You might be prompted for your Virtuoso credentials; if so, enter them.
- Issue the command:

```
GRANT execute ON SPARQL_INSERT_DICT_CONTENT TO SPARQL_UPDATE;
```

- If you also want to enable templates to delete triples, issues the following commands as well:

```
GRANT execute ON SPARQL_DELETE_DICT_CONTENT TO "SPARQL"
GRANT execute ON SPARQL_DELETE_DICT_CONTENT TO SPARQL_UPDATE;
```

Note: If you are worried about the security implications of allowing triple-insertion via SPARQL endpoint, our recommendation is as follows. You should configure your machine so that only localhost is permitted to connect to port 8890 (or whichever port Virtuoso is running on). Then, you can perform proper validation of user input in whatever program it is you're designing, before invoking the RDFStore API.

#### 7.4.8.2 Fuseki

By default, the Fuseki triplestore runs a SPARQL endpoint on port 3030. If you're running Fuseki on some other port, change everything accordingly.

When using Fuseki, one gives one's dataset a name, and that name has to be inserted into the SPARQL endpoint URL. For the documentation below, we will assume your dataset is named "dataset". If you use a different name, change everything accordingly.

##### 7.4.8.2.1 Queries

Depending on what format you like, you can run RDFStore with the following endpoint strings.

###### *7.4.8.2.1.1 JSON*

Endpoint string:

http://localhost:3030/dataset/query?force-accept=text%2Fplain&output=json&query=

Minimum working example command line (query only, no update support):

```
java Rdfstore -endpoint "http://localhost:3030/dataset/query?force-
accept=text%2Fplain&output=json&query="
```

Remember to change "dataset" to the actual name of your Fuseki dataset!

###### *7.4.8.2.1.2 Text*

Endpoint string:

http://localhost:3030/dataset/query?force-accept=text%2Fplain&output=text&query=

Minimum working example command line (query only, no update support):

```
java Rdfstore -endpoint "http://localhost:3030/dataset/query?force-
accept=text%2Fplain&output=text&query="
```

Remember to change "dataset" to the actual name of your Fuseki dataset!

###### *7.4.8.2.1.3 XML*

Endpoint string:

http://localhost:3030/dataset/query?force-accept=text%2Fplain&output=xml&query=

Minimum working example command line (query only, no update support):

```
java Rdfstore -endpoint "http://localhost:3030/dataset/query?force-
accept=text%2Fplain&output=xml&query="
```

Remember to change "dataset" to the actual name of your Fuseki dataset!

###### *7.4.8.2.1.4 Tab Separated Values (TSV)*

Endpoint string:

http://localhost:3030/dataset/query?force-accept=text%2Fplain&output=tsv&query=

Minimum working example command line (query only, no update support):

```
java Rdfstore -endpoint "http://localhost:3030/dataset/query?force-
accept=text%2Fplain&output=tsv&query="
```

Remember to change "dataset" to the actual name of your Fuseki dataset!

##### 7.4.8.2.2 Adding Triples

The Fuseki SPARQL endpoint uses different URLs for SPARQL queries and SPARQL updates. Furthermore, it only accepts SPARQL updates sent with an HTTP POST, it rejects updates sent with HTTP GET.

Fortunately, RDFStore allows you to specify a separate address/method for updates. Run RDFStore with command line options

```
-updatemethod POST
```

and

```
-update "http://localhost:3030/dataset/update"
```

(replace "dataset" with the actual name of your dataset).

**Minimal working example**

If you want to run RDFStore using Fuseki as the triplestore, returning query results in JSON format, and with a dataset named "models", you can run it as follows:

```
java Rdfstore -endpoint "http://localhost:3030/models/query?force-
accept=text%2Fplain&output=json&query=" -update "http://localhost:3030/models/update" -
updatemethod POST
```

### 7.4.9 OWL Ontology Services

As mentioned above, the RICORDO OWLKB [20] provides specific APIs for querying and adding to the ontologies that reside in its knowledge base. These operations are exposed in the form of RESTfull services with a simplified syntax. Moreover, a mechanism exists to create new terms in an ontology when no logically equivalent term has been found. The knowledge base also supports inferencing on its ontology.

The RDF store application on the other hand is dedicated to querying RDF annotations. However, it contains as part of its implementation regarding the template mechanism described above, a way of performing query expansion via a preliminary call to the knowledge base. Such a query includes a step in which a query is sent to the knowledge base in order to extend the search space on the annotation store.

OWLKB 2.0 is the Ricordo semantic reasoning server. It provides an API for querying semantic data which is loaded from an ontology. OWLKB is smart enough to know the semantic meanings of the terms in the ontology and to act accordingly.

As a simple example, suppose that the ontology says widget X was created at factory Y, and that factory Y only creates blue widgets. A query for "show all blue widgets" will show X even if the ontology does not explicitly say that X is blue: the reasoner is smart enough to deduce the blueness of X from the other two facts.

### 7.4.10 Installation

- Ensure a java runtime and java compiler are installed.
- Use "`git clone`", or any other means, to copy the repository from http://github.org/semitrivial/owlkb
- Within the directory containing the copy of the repository, expand OWLKB's dependencies using "`jar -xf dep.jar`"
- Within the directory containing the copy of the repository, compile OWLKB using "make" (or on Windows: "`javac -g Owlkb.java`")
- Within that directory, run "`java Owlkb.java -help`" to get help on the command line arguments, or see further below.

### 7.4.11 Loading an Ontology

OWLKB loads ontologies in .owl form; we assume the user has an owlfile on their system. When running OWLKB, one should specify the location of the desired .owl file. This is done using the -file command line argument.

For example:

```
java Owlkb -file /home/ontologies/ricordo.owl
```

#### 7.4.11.1 Command line arguments

OWLKB can be run with the following command line arguments.

| Argument | Description | Example or default |
|---|---|---|
| -file <location of file> | Specifies which ontology file OWLKB will load. | Default: ./templates |
| -port <port number> | Specifies which ontology file OWLKB will load. | Default: 20080 |
| -reasoner <elk or hermit> | Specifies which reasoner OWLKB will use. | Supported: Elk, HermiT |
| -namespace <base of iri> | Specifies the namespace to be used for classes created with OWLKB. | |
| -save <true or false> | Specifies whether or not OWLKB saves new classes to harddrive | Default: true |
| -help | Displays a help screen. | |

### 7.4.12Simple GUI

OWLKB comes with a simple GUI. When OWLKB is running, the GUI can be accessed at

http://localhost:20080/gui/

(replace "localhost" with whatever host you're running OWLKB on, and replace "20080" with whatever port your OWLKB is running on, if necessary).

For example, the demonstration instance of OWLKB is running on host open-physiology.org on port 20080, so the GUI is at http://open-physiology.org:20080/gui.

The built-in GUI is mainly just for demonstration purposes. We anticipate OWLKB will mainly be used directly via the API.

### 7.4.13 KBCaller Java Library

OWLKB is not designed as a library and is not. The reason for this is that OWLKB is rather resource-intensive when loaded with a non-trivial ontology. Thus it makes more sense as a separate process than as a library. Nevertheless, KBCaller is a Java mini-library which abstracts the act of sending API requests to OWLKB over HTTP and can be used in Java-based projects.

#### 7.4.13.1 Constructor

```
public KBCaller( String url )
```

Creates a KBCaller object. Specify the url of an OWLKB instance, including port.

For example, the OWLKB demo instance has the url http://open-physiology.org:20080. If OWLKB is running on the same machine as the Java application you're working on, and if OWLKB is running on its default port (20080), you can use the url http://localhost:20080

#### 7.4.13.2 API Methods

In all cases except for "addlabel", the methods return a list of results as a JSON list, e.g. something like:

['FMA_50801','CHEBI_999','RICORDO_56345634']

If you would prefer the results as an ArrayList<String> and you don't want to add a full JSON parser dependency to your project, we've included a bare-bones JSON-list-parser function in KBCaller:

```
public ArrayList<String> parse_json( String json ) throws IOException
```

You can compose this with any of the String-returning API methods (except "subhierarchy"), for example:

```
KBCaller kbcaller = new KBCaller( "http://open-physiology.org:20080" );
String subclasses_raw;
List<String> subclasses;
try
{
  String subclasses_raw = kbcaller.subterms( "part-of some FMA_50801" );
  subclasses = kbcaller.parse_json( kbcaller.subterms( "part-of some FMA_50801" ) );
}
catch( Exception e )
{
  e.printStackTrace();
}
```

If you want to parse the "subhierarchy" JSON, you'll probably want to use a full JSON parser for that, as it's not a flat list.

### 7.4.14 OWLKB API

OWLKB launches a server which listens for connections and responds 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.

#### 7.4.14.1 Subterms

Finds all subterms of the indicated term. For example, "amino acid" is a subterm of "acid".

Example:

http://localhost:20080/subterms/CHEBI_33709

#### 7.4.14.2 Parents

Finds all the direct parents (i.e., the direct superclasses) of the indicated term.

Example:

http://localhost:20080/parents/CHEBI_33709

#### 7.4.14.3 Children

Finds all the direct children (i.e., the direct subclasses) of the indicated term.

Example:

http://localhost:20080/children/CHEBI_33709

#### 7.4.14.4 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.

Example:

http://localhost:20080/siblings/CHEBI_33709

#### 7.4.14.5 Subhierarchy

Finds all subterms of the indicated term, and displays them in a hierarchical format (using JSON).

Example:

http://localhost:20080/subhierarchy/CHEBI_33709

#### 7.4.14.6 Eqterms

Finds all terms equivalent 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).

Example:

http://localhost:20080/eqterms/CL_0000548+and+(capable_of+some+GO_0022414)

#### 7.4.14.7 Terms

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.

Example:

http://localhost:20080/terms/CL_0000548+and+(capable_of+some+GO_0022414)

#### 7.4.14.8 Instances

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.)

Example:

http://localhost:20080/instances/TYPE-OF-CLINICAL-CONTEXT

#### 7.4.14.9 Labels

Finds all labels annotated to the indicated term (specifically, all rdfs:label's). For example, the label "Brain" is annotated to FMA_50801.

Example:

http://localhost:20080/labels/FMA_50801

#### 7.4.14.10 Search

Finds all classes in the ontology with the given label (specifically, the given rdfs:label). Note that this is an exact, case-sensitive search--a search for "Brai" or "brain" will not return "Brain" for instance.

Example:

http://localhost:20080/search/Brain

#### 7.4.14.11 Addlabel

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 <rdfs:label>. 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).

Example:

http://localhost:20080/addlabel/RICORDO12345=volume+of+blood+in+aorta

### 7.4.15 JSON

There are three ways to coerce data into JSON format:

1. Include an URL paramater `'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"
```

### 7.4.16 Verbose Results

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

### 7.4.17Manchester Syntax

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 urlencoded.

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)"

### 7.4.18 Terminology Services

Knowledge base caters for the logical content of term definitions. In addition, ontology terms can be endowed with lexical information that allows to provide a human readable label for them and also to offer a form of natural language interface to the knowledge base in order to find candidate terms by their labels rather than by their logical descriptions. This facilitates the accessibility to the end users by making use of the knowledge base.

While it is not the specialization of knowledge base technologies to handle natural language processing, the RICORDO OWLKB is complemented with a minimalist terminology service. This service provides lookup assistance and some degree of search based on the labels of the ontology. However, it is far less robust and rich than third party services available over the web such as the Ontology Lookup Service [24]. The rationale for having a smallest dedicated service is to provide basic terminological support for ontologies that are local to the deployed knowledge base rather than in the public domain, hence, the RICORDO addition of a "Local OLS" [25].

LOLS stands for Local Ontology Lookup Service. Its intended purpose: for a given set of ontologies, let people look up rdfs:labels from IRIs and IRIs from rdfs:labels. LOLS is lean and minimalist, allowing easy deployment on any machine, removing the need to refer to a centralized label lookup service which might be located on the other side of the world.

Technically, LOLS has two components:

i) A converter which turns an OWL file into a LOLS file. Written in Java to use the OWLAPI.

ii) The main engine, which loads a LOLS file and serves API requests in HTTP. Written in C.

### 7.4.19 Prerequisite for Installation

- java runtime and java compiler
- C compiler (gcc)

### 7.4.20 Installation Instructions (tested on Linux and Mac)

- Use "`git clone`", or any other means, to copy the repository from http://github.org/semitrivial/LOLS

Two subdirectories will be created: "converter" and "server"

- In the converter directory: expand dependencies with "`jar -xf dep.jar`"
- In the converter directory: "`make`" (or "`javac -g Convert.java`"). This creates a "Convert.class" java executable for converting OWL files to LOLS files.
- In the server directory: "`make`" (or "`gcc lols.c srv.c trie.c util.c -o lols`"). This creates an executable "`lols`" for running LOLS.

### 7.4.21 LOLS File Preparation

LOLS loads IRIs and rdfs:labels from an N-Triples file, which can be generated from an OWL ontology file by means of a converter written in java.

Navigate to the LOLS converter directory (created in "Installation" above).

Run the following command:

```
java Convert (OWLfile) >(outputfilename)
```

For example, if your OWL file is located at "/home/ontologies/fma.OWL", and if you want the LOLS file to be called "fma.LOLS", then you would run:

Example command to extract an N-Triples file, `fma.nt`, from an OWL file, /home/ontologies/fma.OWL:

```
java Convert /home/ontologies/fma.OWL >fma.nt
```

It might be necessary to manually edit the LOLS file to remove unrelated output from the top of it, which was placed there by the OWL reasoner. (In a future version of LOLS this step will not be necessary.)

#### 7.4.21.1 Multiple OWL Files

If you have multiple OWL files and you want a single LOLS file to cover all of them, what you should do is create a shell ontology (see example below) file which imports all the desired ontologies. Then run the converter on the shell ontology.

Example

For example, suppose you want your LOLS file to cover /home/fma.owl, /home/chebi.owl, and /home/go.owl. Then you can create the following shell ontology and run the converter on it:

```
    <?xml version="1.0"?>
         <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#">
           <owl:Ontology rdf:about="http://open-physiology.org/shell-ontology">
             <owl:imports rdf:resource="file:/home/ricordo/ontology/fma.owl"/>
             <owl:imports rdf:resource="file:/home/ricordo/ontology/chebi.owl"/>
             <owl:imports rdf:resource="file:/home/ricordo/ontology/go.owl"/>
           </owl:Ontology>
         </rdf:RDF>
```

By modifying the above example in the obvious way, you can write a shell ontology to cover whatever set of ontologies you like. Then run the converter on it to get the desired LOLS file. (Note: the url "http://open-physiology.org/shell-ontology" in the example is just a placeholder url, anything will work there and it won't effect the resulting LOLS file.)

### 7.4.22 Running the LOLS Server

Once you've created a LOLS file, you can launch the LOLS server by going to the "server" directory (created in "Installation" above) and running:

```
./lols (path to LOLSfile)
```

For example, if you created the LOLSfile "/home/ontologies/mylols.LOLS", then you can run:

```
./lols /home/ontologies/mylols.LOLS
```

By default, LOLS will open an HTTP server on port 5052. (You can change that in srv.c and re-compile, if you prefer another port.) See "API" (below) and "Built-in GUI" (below) for how to actually use that server.

### 7.4.23Simple GUI

LOLS comes with a simple built-in GUI. Assuming the LOLS server is running, you can access the GUI at http://(yourdomain):5052/gui

Example

If your domain is "example.com" then you can access the LOLS GUI at

http://example.com:5052/gui

Of course, if you don't have a domain, an IP address or "localhost" can be used instead.

### 7.4.24 LOLS API

LOLS launches a server which listens for connections and responds to the following types of requests.

In each case, the results are output in JSON format.

#### 7.4.24.1 IRI

Finds all rdfs:labels associated to the class with the specified IRI. The IRI can either be specified in full, as in the second example, or else abbreviated as in the first example.

Example (shortform):

http://localhost:5052/iri/FMA_50801

Example (longform):

http://localhost:5052/iri/http%3A%2F%2Fpurl.org%2Fobo%2Fowlapi%2Ffma%23FMA_50801

Note that "http%3A%2F%2Fpurl.org%2Fobo%2Fowlapi%2Ffma%23FMA_50801" is the urlencoded result of "http://purl.org/obo/owlapi/fma#FMA_50801".

#### 7.4.24.2 Label

Finds all IRIs of classes with the indicated rdfs:label (case sensitive). The IRIs are given in full.

Example:

http://localhost:5052/label/Brain

#### 7.4.24.3 Label Case Insensitive

Finds all IRIs of classes with the indicated rdfs:label (case insensitive). The IRIs are given in full.

Example:

http://localhost:5052/label/brain

#### 7.4.24.4 Label shortiri

Finds all IRIs of classes with the indicated rdfs:label (case sensitive). The IRIs are given in abbreviated form, if possible.

Example:

http://localhost:5052/label-shortiri/Brain

#### 7.4.24.5 Label shortiri Case Insensitive

Finds all IRIs of classes with the indicated rdfs:label (case insensitive). The IRIs are given in abbreviated form, if possible.

Example:

http://localhost:5052/label-shortiri-case-insensitive/brain

## 7.5 *CHIC Semantic Models*

The CHIC semantic infrastructure hosts semantic data that is made primarily of annotations of CHIC resources on the one hand and supporting data on the other hand. The supporting data is made on the side of the annotation store with schemas for the articulation of these descriptions and the querying of annotations. A schema, in that sense, is a less complex ontology of the CHIC resources themselves, an ontology we baptised CHICRO for CHIC Resource Ontology. CHICRO contains the main types of CHIC resources handled in the CHIC Model Repository and the Hypermodelling Editor. In addition, the Clinical Data Repository has its own schema and variant or extension of CHICRO.

### 7.5.1 Ontology of CHIC Resources (Main Concept in CHICRO)

CHICRO is a very small ontology intended to categorise and describe the relationships between CHIC resources at a high level of generality. It's main concepts are as follows [26]:

- **Modelling objects** – a class of objects. An instance of the class Modelling Object is a model or a part (logical or conceptual) of a model, in particular a parameter.
- **Mathematical models** – a subclass of the class of modelling objects. An instance of the class MathematicalModel is a model.
- **Hypermodels** – a subclass of the class of mathematical models. An instance of the class Hypermodel is a hypermodel in the sense of CHIC technical specifications.
- **Hypomodels** – a subclass of the class of mathematical models. An instance of the class Hypomodel is a hypomodel in the sense of CHIC technical specifications.
- **Parameter** of a mathematical model – a subclass of the class of modelling objects such that an instance of this class is a parameter of a mathematical model.
- **Input parameter** of a mathematical model – a subclass of the class of parameter of a mathematical model such that an instance of this class is an input parameter of some mathematical model.
- **Output parameter** of a mathematical model – a subclass of the class of parameter of a mathematical model such that an instance of this class is an output parameter of a mathematical model.

A schema has been implemented in RDF and is used in the main CHIC triple store. While the implementation is mature enough to handle the main cases, it is also open ended and extensible and therefore provisions are made for new refinement or corrective versions. At the time of this writing, the current version is 0.9.1 (Figure 60) and will be consolidated into 1.0 in the final stages of the project.

Selected versions may be accessed here: https://github.com/open-physiology/chic/tree/master/rdfschema

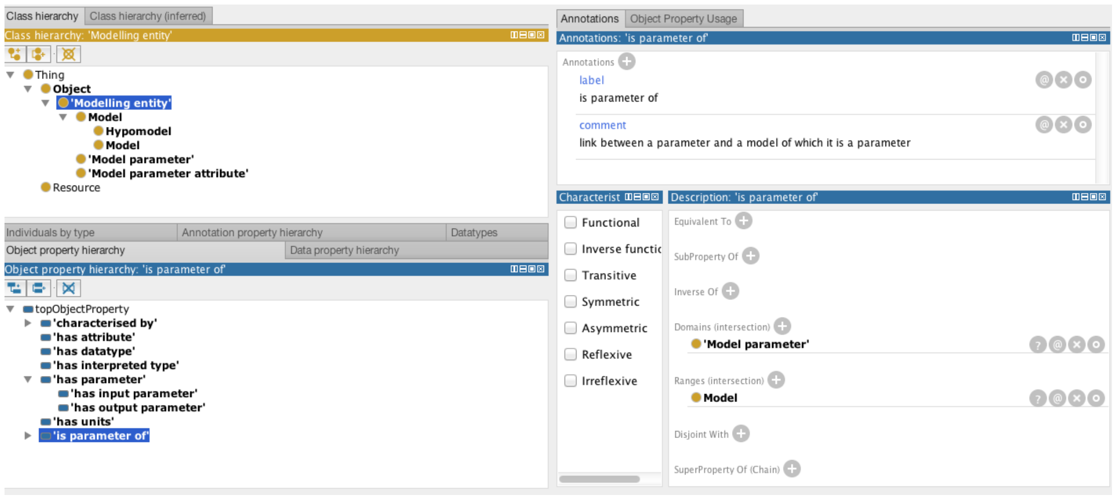


**Figure 60: A screenshot of CHCRO visualised in the Protégé ontology editor. The left top panel shows the class primitive hierarchy (there are no classified models nor inferred specialisations appearing)**

### 7.5.2 Annotation Vocabulary included in CHICRO Schema

The CHICRO RDF schema also allows the constraint of the vocabulary used in order to annotate selected types of resources. Table 83 shows annotation properties susceptible of being used in the annotation of models and parameters. The vocabulary is constrained to use entities of the intended kind in the subject and object positions of the annotation triples by using the listed properties as predicates [26].

The properties apply to a number of object of different kinds and levels of generality and can be broadly categorised as being intended for:

- All-objects annotation
- Model annotation
- Parameter annotation

**Table 83: Annotation properties applying to all objects**

| Name | Symbol in CHICRO | Domain | Range |
|---|---|---|---|
| Title or Name | hasName | chicro:Object | rdfs:Literal |
| Description | hasDescription | chicro:Object | rdfs:Literal |

We define 13 annotation relationships corresponding to each of the intended perspectives (D6.1)

**Table 84: Annotation properties applying to models**

| Name | Symbol in CHICRO | Domain | Range |
|---|---|---|---|
| Perspective 1 | hasPositionIn-01 | chicro:Model | Set of relevant URIs for Tumour-Affected Normal Tissue Modelling |
| Perspective 2 | hasPositionIn-02 | chicro:Model | Set of relevant URIs for Spatial Scale(s) of the Manifestation of Life |
| Perspective 3 | hasPositionIn-03 | chicro:Model | Set of relevant URIs for Temporal Scale(s) of the Manifestation of Life |
| Perspective 4 | hasPositionIn-04 | chicro:Model | Set of relevant URIs for Biomechanism(S) Addressed |
| Perspective 5 | hasPositionIn-05 | chicro:Model | Set of relevant URIs for Tumour Type(S) Addressed |
| Perspective 6 | hasPositionIn-06 | chicro:Model | Set of relevant URIs for Treatment Modality(-ies) Addressed |
| Perspective 7 | hasPositionIn-07 | chicro:Model | Set of relevant URIs for Generic Cancer Biology – Clinically Driven Character of the Modelling Approach |
| Perspective 8 | hasPositionIn-08 | chicro:Model | Set of relevant URIs for Order of Addressing Different Spatial Scales |
| Perspective 9 | hasPositionIn-09 | chicro:Model | Set of relevant URIs for Order of Addressing Different Temporal Scales |
| Perspective 10 | hasPositionIn-10 | chicro:Model | Set of relevant URIs for Mechanistic-Statistical |

| | | | |
|---|---|---|---|
| | | | Character of the Modelling Approach |
| Perspective 11 | hasPositionIn-11 | chicro:Model | Set of relevant URIs for Deterministic-Stochastic Character of the Modelling Approach |
| Perspective 12 | hasPositionIn-12 | chicro:Model | Set of relevant URIs for Continuous-Finite-Discrete Character of the Mathematics Involved |
| Perspective 13 | hasPositionIn-13 | chicro:Model | Set of relevant URIs for Closed Form Solution – Algorithmic Simulation Modelling Approach |

**Table 85: Annotation properties applying to model parameters**

| **Name** | **Symbol in CHICRO** | **Domain** | **Range** |
|---|---|---|---|
| datatype | parameterhasDatatype | chicro:Parameter | xsd datatypes (and possible additions) |
| unit | parameterHasUnit | chicro:Object | Set of relevant URIs |

In order to record the interpretation of a given object according to a domain specific definition, for example, that a given parameter is a rate of cell killing, we use a specific relationship rather than the rdf:type property.

**Table 86: Annotation property for making a semantic type explicit**

| **Name** | **Symbol in CHICRO** | **Domain** | **Range** |
|---|---|---|---|
| Semantic type | hasInterpretedType | chicro:Object | Set of relevant URIs |

In addition to the above, the CHICRO schema also contains information which intends on guiding the development of the corresponding Graphical User Interfaces that are either used to create or display annotations.

#### 7.5.2.1 Service operations using the annotation vocabulary included in CHICRO Schema

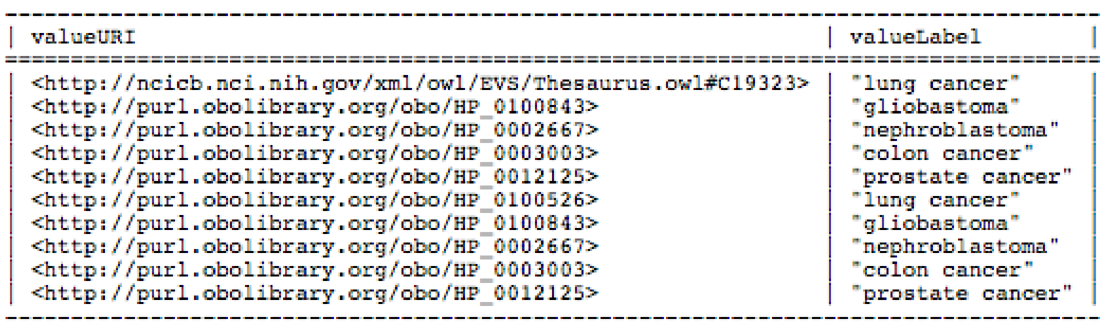
These can be retrieved by direct query to the RDF store. For example, given the model description guidelines [27], Figure 61 shows the SPARL query to retrieve the vocabulary related to the annotation of Perspective V.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

select ?valueURI ?valueLabel where {
<http://www.chic-vph.eu/ontologies/resource#hasPositionIn-5>            <http://www.chic-vph.eu/ontologies/resource#designatedValueWithPreferredLabel> ?pair .
?pair rdf:first ?valueURI .
?pair rdf:rest ?rest .
?rest rdf:first ?valueLabel
}
```

**Figure 61: Example of SPARQL query to retrieve information to be used in GUI**

Figure 62 shows the set of pairs of URIs and labels to be displayed to a user through a Graphical User Interface corresponding to the query. Such information may be used, for example in the Model Repository annotation interface.



```
--------------------------------------------------------------------------------
| valueURI                                                  | valueLabel        |
================================================================================
| <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C19323> | "lung cancer"     |
| <http://purl.obolibrary.org/obo/HP_0100843>               | "gliobastoma"     |
| <http://purl.obolibrary.org/obo/HP_0002667>               | "nephroblastoma"  |
| <http://purl.obolibrary.org/obo/HP_0003003>               | "colon cancer"    |
| <http://purl.obolibrary.org/obo/HP_0012125>               | "prostate cancer" |
| <http://purl.obolibrary.org/obo/HP_0100526>               | "lung cancer"     |
| <http://purl.obolibrary.org/obo/HP_0100843>               | "gliobastoma"     |
| <http://purl.obolibrary.org/obo/HP_0002667>               | "nephroblastoma"  |
| <http://purl.obolibrary.org/obo/HP_0003003>               | "colon cancer"    |
| <http://purl.obolibrary.org/obo/HP_0012125>               | "prostate cancer" |
--------------------------------------------------------------------------------
```

**Figure 62: Example of the mapping of URIs related to the annotation for Perspective IV with the labels corresponding to the concepts to be displayed in a GUI (Output retrieved from RDF Store)**

#### 7.5.2.2 Template-based service operations using the annotation vocabulary

Furthermore, a number of specialised service calls can be defined to suit specific use cases and requirements. In a manner of illustration we specify a few basic examples. The tools put in place allow for specifying any further needed specific requests.

##### 7.5.2.2.1 Get_Hypomodels

Arguments: none

Description: Returns a once column table of URIs for hypomodels.

| | |
|---|---|
| Method: | GET |
| URL: | http://<HOST>:<PORT>/Get_Hypomodels/ |
| Body: | [nil] |

Example request: Returns all the hypomodels URIs

```
http://localhost:20060/Get_Hypomodels
```

Example response:

```
<pre>
&#60;http://example/update-base/#model1&#62;
&#60;http://example/update-base/#model2&#62;
</pre>
```

Template .txt

```
select distinct ?model where {
?model <http://www.w3.org/2000/01/rdf-schema#type>
<http://www.chic.eu/ontologies/resource#Model-ChicHypomodel>
}
```

#### 7.5.2.2.2 Get_HypomodelInputParameter_ByInterpretation_exactMatch

Argument_0: URL encoded URI

Description: Returns a one column table of URIs of input parameters of any model such that the parameter is annotated with the URI provided as argument.

| | |
|---|---|
| Method: | GET |
| URL: | http://<HOST>:<PORT>/Get_HypomodelInputParameter_ByInterpretation_exactMatch/?0=<URL-ENCODED-URI> |
| Body: | [nil] |

Example request: Returns all the input parameters whose interpretation is http://www.chic.eu/ontologies/some-domain-ontology#CHIC_0001023 (a fictional concept for the sake of illustration)

```
http://localhost:20060/Get_HypomodelInputParameter_ByInterpretation_exactMatch/?0=http%3A
%2F%2Fwww.chic.eu%2Fontologies%2Fsome-domain-ontology%23CHIC_0001023
```

Example response:

```
<pre>
&#60;http://example/update-base/#parameter2a&#62;
&#60;http://example/update-base/#parameter1&#62;
</pre>
```

Template .txt

```
# 0 interpretation
select distinct ?parameter where {
?parameter  <http://www.chic.eu/ontologies/some-model-ontology#interpreted-type>  <[0]>
.
?parameter       <http://www.chic.eu/ontologies/some-model-ontology#input-parameter-of>
?model }
```

#### 7.5.2.2.3 Get_HypomodelOutputParameter_ByInterpretation_exactMatch

Argument_0: URL encoded URI

Returns a one column table of URIs of output parameters of any model such that the parameter is annotated with the URI provided as argument.

| | |
|---|---|
| Method: | GET |
| URL: | http://<HOST>:<PORT>/Get_HypomodelOutputParameter_ByInterpretation_exactMatch/?0=<URL-ENCODED-URI> |
| Body: | [nil] |

Example request: Returns all the output parameters whose interpretation is http://www.chic.eu/ontologies/some-domain-ontology#CHIC_0001023 (a fictional concept for the sake of illustration)

```
http://localhost:20060/Get_HypomodelOutputParameter_ByInterpretation_exactMatch/?0=http%3
A%2F%2Fwww.chic.eu%2Fontologies%2Fsome-domain-ontology%23CHIC_0001023
```

Example response:

```
<pre>
```

```
&#60;http://example/update-base/#parameter3a&#62;
</pre>
```

Template .txt

```
# 0 interpretation
select distinct ?parameter where {
?parameter <http://www.chic.eu/ontologies/resource#hasInterpretedType> <[0]> .
?parameter < http://www.chic.eu/ontologies/resource#output-parameter-of> ?model
}
```

#### 7.5.2.2.4 Get_Consistent_HypomodelOutputParameter_ByInterpretation_exactMatch

Argument_0: URL encoded URI

Returns a one column table of URIs of output parameters of any model such that the parameter is annotated with the URI provided as argument.

| | |
|---|---|
| Method: | GET |
| URL: | http://<HOST>:<PORT>/Get_Consistent_HypomodelOutputParameter_ByInterpretation_exactMatch/?0=<URL-ENCODED-URI> |
| Body: | [nil] |

Example request: Returns all the output parameters of any hypomodel whose interpretation is exactly the same as that of http://example/update-base/#parameter2a (a fictional parameter URI for the sake of illustration)

```
http://localhost:20060/Get_Consistent_HypomodelOutputParameter_ByInterpretation_exactMatch
/?0=http%3A%2F%2Fexample%2Fupdate-base%2F%23parameter2a
```

Example response:

```
<pre>
&#60;http://example/update-base/#parameter3a&#62;
</pre>
```

Template .txt

```
# 0 parameter
select distinct ?parameter2 where {
<[0]> < http://www.chic.eu/ontologies/resource#interpreted-type> ?i .
?parameter2 < http://www.chic.eu/ontologies/resource#interpreted-type> ?i .
?parameter2 < http://www.chic.eu/ontologies/resource#output-parameter-of> ?model2
}
```

### 7.5.3 Domain Ontologies

The domain ontology coverage in CHIC intends to support the annotation of models and their parameters. These are relatively well defined and short excerpts of third party ontologies may be used to pinpoint concepts requiring an externalizable reference. In some cases, the existing repertoire of ontologies presents gaps and does not cover the needs of specific annotation properties or incompletely does so. The methodological strategy adopted early in the project to reuse communally maintained publicly available ontologies and complete them on an ad hoc basis is still followed. It is possible that these ad hoc additions will be consolidated in ontology products in their own right if they prove significantly reusable and pass standard tests of quality assurance (following in particular the OBO Foundry principles of ontology development [28]).

Our strategy is to enable the use of semantic annotation and then to go through cycles of refinements. For this reason we will adopt the following:

| **Domain** | | |
|---|---|---|
| Units of measurement | Unit Ontology | http://obofoundry.org/ontology/uo.html |
| Perspectives | Existing Ontologies or Ad Hoc Ontology | Existing ontologies or parts of existing ontologies can be used for different perspectives (for example, the Disease Ontology for perspective 5 or the Unit Ontology for perspective 3) or ad hoc ontologies can be developed for others. |
| Semantic types | Combinations of ontologies, among which:<br>FMA<br>CHEBI<br>GO<br>PATO<br>Ad Hoc Extensions | The external ontologies are developed and maintained within the framework of the Open Biomedical Ontologies (OBO Foundry, http://obofoundry.org/).<br><br>Ad hoc extensions are the subject of further development and involve extensions and combinations of the above. |

The relevance of ontologies in the CHIC infrastructure is primarily perceived to be the knowledge base which supports the search of models. Ontologies are deposited in the knowledge base which as mentioned above supports reasoning. The storage of annotation supports lookup corresponding to exact match searches. By combining both, it becomes possible to use inferences in order to make more relevant or abstract queries.

For instance, as shown in Figure 62, models can be annotated according to Perspective V by using a number of selected terms. Many of these belong to the Human Phenotype Ontology [29]. Queries can be run in order to retrieve models annotated to these terms. In order to retrieve models while keying the query to a more general term that could subsume several terms, the knowledge of the way the terms relate in a hierarchy is needed. This knowledge is provided by the HPO which in turn is included in the CHIC knowledge base.

Figure 63 shows the knowledge in a hierarchical arrangement. One of the terms used in the annotation of CHIC model is that for the Wilms tumour. By the standards of HPO, this is a specialization of neoplasm.

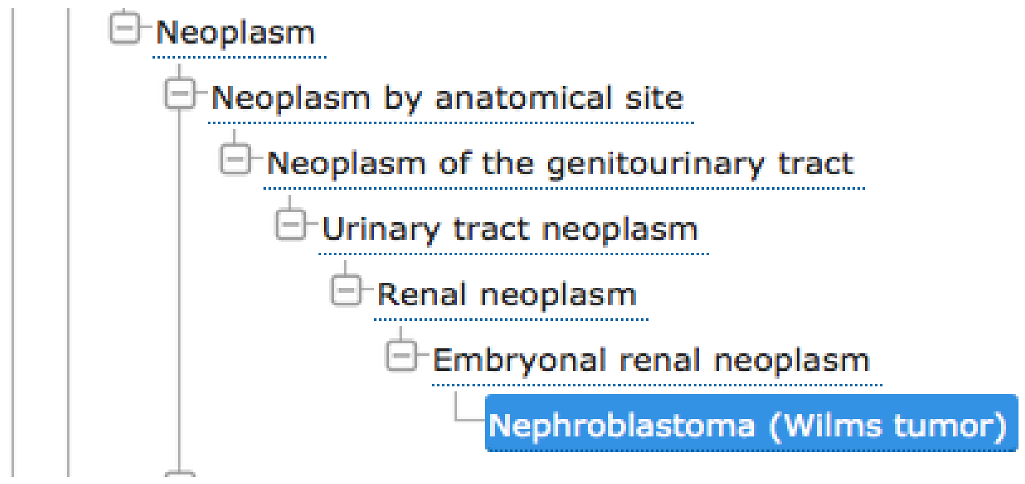
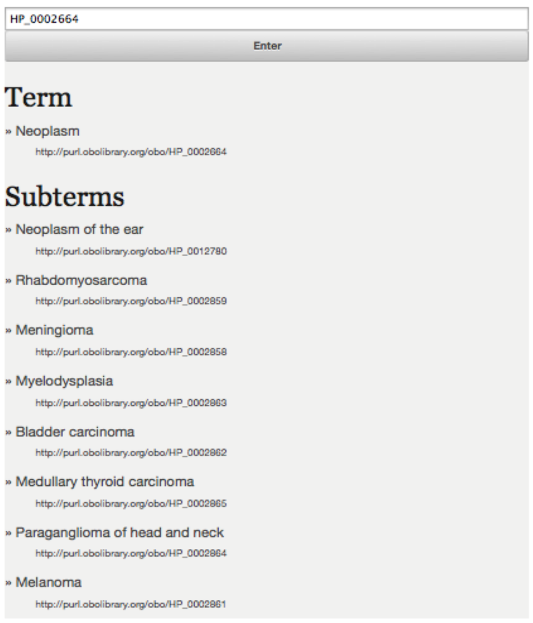
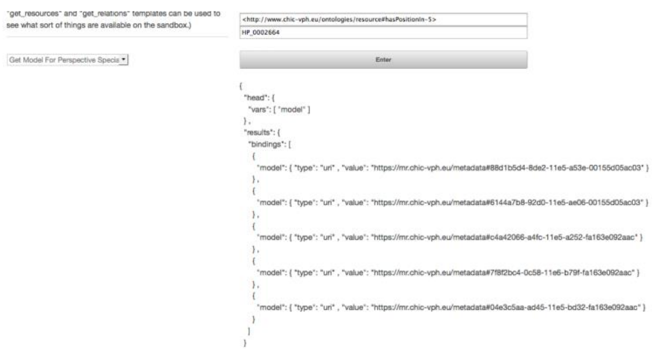

Neoplasm
Neoplasm by anatomical site
Neoplasm of the genitourinary tract
Urinary tract neoplasm
Renal neoplasm
Embryonal renal neoplasm
Nephroblastoma (Wilms tumor)

**Figure 63: Excerpt of the HPO hierarchy**

The RDF store application allows searching the CHIC triple store for models annotated with a specialization of neoplasm (Figure 64) and the lookup queries sent to the triple store are then applied to the list of specializations to produce the results (Figure 65).

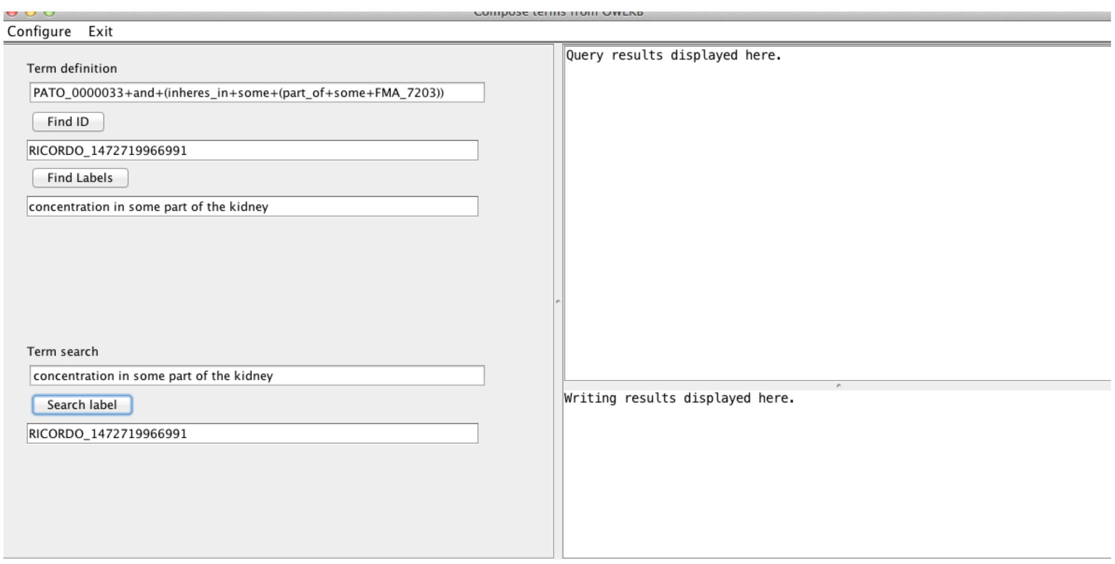
| | |
|---|---|
|  HP_0002664<br>Enter<br>**Term**<br>» Neoplasm<br>http://purl.obolibrary.org/obo/HP_0002664<br>**Subterms**<br>» Neoplasm of the ear<br>http://purl.obolibrary.org/obo/HP_0012780<br>» Rhabdomyosarcoma<br>http://purl.obolibrary.org/obo/HP_0002859<br>» Meningioma<br>http://purl.obolibrary.org/obo/HP_0002858<br>» Myelodysplasia<br>http://purl.obolibrary.org/obo/HP_0002863<br>» Bladder carcinoma<br>http://purl.obolibrary.org/obo/HP_0002862<br>» Medullary thyroid carcinoma<br>http://purl.obolibrary.org/obo/HP_0002865<br>» Paraganglioma of head and neck<br>http://purl.obolibrary.org/obo/HP_0002864<br>» Melanoma<br>http://purl.obolibrary.org/obo/HP_0002861 |  "get_resources" and "get_relations" templates can be used to see what sort of things are available on the sandbox.)<br>Get Model For Perspective Specia<br><http://www.chic-vph.eu/ontologies/resource#hasPositionIn-5><br>HP_0002664<br>Enter<br>{ "head": { "vars": [ "model" ] }, "results": { "bindings": [ { "model": { "type": "uri" , "value": "https://mr.chic-vph.eu/metadata#88d1b5d4-8de2-11e5-a53e-00155d05ac03" } }, { "model": { "type": "uri" , "value": "https://mr.chic-vph.eu/metadata#6144a7b8-92d0-11e5-ae06-00155d05ac03" } }, { "model": { "type": "uri" , "value": "https://mr.chic-vph.eu/metadata#c4a42066-a4fc-11e5-a252-fa163e092aac" } }, { "model": { "type": "uri" , "value": "https://mr.chic-vph.eu/metadata#7f8f2bc4-0c58-11e6-b79f-fa163e092aac" } }, { "model": { "type": "uri" , "value": "https://mr.chic-vph.eu/metadata#04e3c5aa-ad45-11e5-bd32-fa163e092aac" } } ] } } |
| **Figure 64: Specializations of neoplasm listed from inference in the knowledge base** | **Figure 65: Query results for models annotated with a specialization of neoplasm in the triple store** |

The mechanism described here illustrates the level of integration in the CHIC semantic infrastructure as it involves communication between the two primary components of the CHIC semantic infrastructure which are the triple store of annotations on the one hand and the knowledge base of ontologies on the other hand.

## 7.6 Ontology Authoring (Prototype)

The main challenge in the annotation strategy used in CHIC and in relation to the final infrastructure envisioned consists in improving the availability of terms. As mentioned earlier, most of the annotation properties are well defined and the range of intended annotations available can be listed with occasional additions on a discrete basis. There is a very important exception which is the annotation of the interpretation of model parameters, in particular their biological meaning when they have one. By nature, these relate to large classes of biological phenomena, measurements and quantities. Form experience, it is possible to sketch a generic vocabulary that may cover most cases (e.g., concentration, rate, etc.). But in order to gain such a scope, the generalization has to be high and the specificity of the parameter may remain inexplicit. This state of affairs is one of the impediments to the automation of parameter matching in the hyper-modelling editor.

A still experimental solution consists in allowing for the definition of a range of parameter interpretations, as they become needed. The primary technical obstacle is to resolve the gap between the annotating user (a modeller) and the format of additional ontology term definitions that the semantic infrastructure currently allows by making use of compositions.


Compose terms from OWLKB

Configure Exit

Term definition

PATO_0000033+and+(inheres_in+some+(part_of+some+FMA_7203))

Find ID

RICORDO_1472719966991

Find Labels

concentration in some part of the kidney

Term search

concentration in some part of the kidney

Search label

RICORDO_1472719966991

```
Query results displayed here.
```

```
Writing results displayed here.
```

**Figure 66: The knowledge base and its API support the creation of terms from their logical definition. Such terms can then be labelled with a human readable name and can then be made available to the user through their lexical information or by other means of sea**

Figure 66 illustrates a prototype desktop GUI showing how the functionality could be implemented in other parts of the user oriented CHIC components. This solution is not mature enough to gauge whether it can be robustly integrated as a final component in the CHIC semantic infrastructure.

# 8 Conclusion

The final report of the CHIC Repositories has been presented in this document. More specifically, the Model Repository which stores the multiscale models and the complementary tools and modules that are needed for the construction of hypermodels has been analysed in chapter 4. The Clinical Data Repository which stores the heterogeneous multiscale data coming from the clinical environment (clinical trials) and the *In Silico* Trial Repository which stores the input and output of the *in silico* simulations along with the complete profile of each simulation are described in chapters 5 and 6 respectively. Finally, the Metadata Repository which stores the machine-readable documentation material that represents models and data has been outlined in the last chapter. As illustrated in this document, all the aforementioned Repositories play a key role in the operation of the CHIC platform as a whole, since they constitute the software building blocks of the system. The integration of the aforementioned Repositories into the CHIC platform ensures the persistent, secure and efficient storage of all the CHIC resources and thus both models and data are readily available to authorized users, either through the user interface or through the corresponding web services. Nonetheless, since technology is never stagnant and new clinical and research requirements may be posed in the future, the Repositories will always have to be revised, updated and extended with the creation of supplementary modules and web services in order to make them even more comprehensive.

# 9 References

[1] D8.1 – Design of the CHIC repositories

[2] D6.2 – CHIC cancer component models: initial tested versions

[3] 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].

[4] 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].

[5] M. Kistler, S. Bonaretti, M. Pfahrer, R. Niklaus, P. Büchler, The Virtual Skeleton Database: An Open Access Repository for Biomedical Research and Collaboration. J. Med. Internet Res. 15:e245, 2013.

[6] C. Rosse, and J. Mejino. A reference ontology for biomedical informatics: the Foundational Model of Anatomy. J. Biomed. Inform. 36:478–500, 2003.

[7] SPARQL, “Query Language for Resource Description Framework (RDF)”, [Online]. Available: http://www.w3.org/TR/rdf-sparql-query. [Accessed 9 August 2016].

[8] 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].

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

[10] elastic, “The company behind the open source projects Elasticsearch, Logstash, Kibana, and Beats”, [Online]. Available: https://www.elastic.co/. [Accessed 9 August 2016].

[11] 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].

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

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

[14] https://www.oasis-open.org/news/pr/iso-and-iec-approve-oasis-amqp-advanced-message-queuing-protocol

[15] Constantine L., and Lockwood, L. *Software for Use: A Practical Guide to the Essential Models and Methods of Usage-Centered Design.* Reading, MA: Addison-Wesley, 1999. (Russian translation 2004, Chinese translation 2004, Japanese translation 2005.)

[16] https://www.w3.org/RDF/

[17] https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/

[18] https://www.w3.org/OWL/

[19] http://jena.apache.org/documentation/serving_data/index.html

[20] https://github.com/open-physiology/owlkb

[21] http://owlapi.sourceforge.net/

[22] https://www.cs.ox.ac.uk/isg/tools/ELK/

[23] https://github.com/open-physiology/rdfstore

[24] http://www.ebi.ac.uk/ols/index

[25] https://github.com/open-physiology/LOLS

[26] D7.3 – Hypermodels annotation services

[27] D6.1 – Cancer hypomodelling and hypermodelling strategies and initial component models

[28] http://www.obofoundry.org/

[29] http://human-phenotype-ontology.github.io/

[30] D6.3 – Initial Standardized Cancer Hypermodels

# Appendix – Abbreviations and acronyms

| | |
|---|---|
| *SOA* | Service Oriented Architecture |
| *SP* | Service Provider |
| *SSO* | Single Sign On |
| *STS* | Security Token Service |
| *REST* | Representational State Transfer |
| *RST* | Request Security Token |
| *RSTR* | Request Security Token Response |
| *SAML* | Security Assertion Markup Language |
| *HTTP* | HyperText Transfer Protocol |
| *RFC* | Request for Comments |
| *API* | Application Programming Interface |
| *JSON* | JavaScript Object Notation |
| *XML* | Extensible Markup Language |
| *UTF* | Unicode Transformation Format |
| *URL* | Uniform Resource Locator |
| *TTP* | Trusted Third Party |
| *MIT* | Massachusetts Institute of Technology |
| *CRAF* | Clinical Research Application Framework |
| *WP* | Work Package |
| *RDF* | Resource Description Framework |
| OWL | Web Ontology Language |
| HPO | Human Phenotype Ontology |
| ER | Entity Relationship |
| UUID | Universally Unique Identifier |
| ID | Identification |
| HF | Hypermodelling Framework |
| SSL | Secure Sockets Layer |
| URL | Uniform Resource Locator |
| ORM | Object Relational Mapping |
| MVC | Model View Controller |

| | |
|---|---|
| HTML | Hypertext Markup Language |
| AMQP | Advanced Message Queuing Protocol |