



Deliverable No. 8.3

Implementation of the interfaces of the CHIC repositories

Grant Agreement No.: 600841
 Deliverable No.: D8.3
 Deliverable Name: Implementation of the interfaces of the CHIC repositories
 Contractual Submission Date: 30/09/2015
 Actual Submission Date: 30/09/2015

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.:	D8.3
Document name:	Implementation of the interfaces of the CHIC repositories
Nature (R, P, D, O) ¹	R
Dissemination Level	PU
Version:	1
Actual Submission Date:	30/09/2015
Editor:	Bernard de Bono
Institution:	UCL
E-Mail:	b.bono@ucl.ac.uk

ABSTRACT:

The key aim of the CHIC infrastructure is to ensure that distinct resources are able to communicate effectively in support of hypermodelling studies. To that end, this document is focused on providing a technical account of the interfaces for four these resources, namely: the hypermodel repository, the clinical data repository, the metadata repository and the in silico trial repository.

KEYWORD LIST:

RESTful web service, SAML token, repository application programming interfaces, model repository interfaces, *in silico* trial repository interfaces, repositories, clinical data, models, in silico trial, semantics, metadata, triple store; semantic reasoning; RICORDO

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 600841.

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

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

MODIFICATION CONTROL			
Version	Date	Status	Author
0.3	19/09/2015	Draft	Bernard de Bono, Nikolaos Tousert, Roman Niklaus, Samuel Alexander
0.4	20/09/2015	Draft	Nikolaos Tousert, ICCS-NTUA
0.7	21/09/2015	Draft	Bernard de Bono, Nikolaos Tousert, Roman Niklaus, Samuel Alexander
0.8	24/09/2015	Pre-final draft	Nikolaos Tousert ICCS-NTUA
0.9	25/09/2015	Revision	Georgios Stamatakis ICCS-NTUA, Dimitra Dionysiou ICCS-NTUA
1.0	27/09/2015	Revision	Bernard de Bono

List of contributors

- Bernard de Bono [UCL],
- Georgios Stamatakis [ICCS],
- Dimitra Dionysiou [ICCS],
- Nikolaos Tousert [ICCS],
- Roman Niklaus [UBern],
- Samuel Alexander [UCL].

Contents

1	EXECUTIVE SUMMARY	8
2	INTRODUCTION	9
2.1	PURPOSE OF THIS DOCUMENT.....	9
3	RESTFUL WEB SERVICES AND AUTHORIZATION	12
3.1	REPOSITORIES MAKE USE OF RESTFUL APPLICATION PROGRAMMING INTERFACES	12
3.2	AUTHENTICATION IN REPOSITORY WEB SERVICES	13
4	INTERFACES OF THE CLINICAL DATA REPOSITORY.....	16
4.1	INTRODUCTION.....	16
4.2	AUTHENTICATION	16
4.3	DOMAIN MODEL	16
4.4	WEB-BASED USER INTERFACE	17
4.5	RESTFUL APPLICATION PROGRAMMING INTERFACES	18
4.6	SUMMARY.....	44
5	IMPLEMENTATION OF THE INTERFACES OF MODEL/TOOL AND <i>IN SILICO TRIAL</i> REPOSITORIES	46
5.1	MODEL/TOOL REPOSITORY SCHEMA	46
5.2	MODEL/TOOL REPOSITORY RESTFUL APPLICATION PROGRAMMING INTERFACES.....	48
5.3	IN SILICO TRIAL REPOSITORY SCHEMA	75
5.4	IN SILICO TRIAL REPOSITORY RESTFUL APPLICATION PROGRAMMING INTERFACES.....	77
6	RDF STORAGE SOLUTION FOR SEMANTIC METADATA	112
6.1	INTRODUCTION.....	112
6.2	TOOLS	112
6.3	THE CHIC SEMANTIC METADATA LIFECYCLE	116
6.4	WHAT CAN BE ANNOTATED?	117
7	CONCLUSIONS	118
8	REFERENCES.....	119

Figures

Figure 1: Brokered authentication for CHIC repository web services	13
Figure 2: Example of conditions element in SAML token	15
Figure 3: The domain model of the clinical data repository with domain classes (blue), domain enumerations (brown) and their relationships represented as connecting lines	17
Figure 4: The web-based user interface main view of the clinical data repository	18
Figure 5: Updated entity relationship (ER) diagram of model/tool repository	47
Figure 6: Updated entity relationship (ER) diagram of in silico trial repository	76
Figure 7: Prototype of RICORDO Template GUI.....	113
Figure 8: Selection of templates and search function	114
Figure 9: Interaction with OWLKB	115

Tables

Table 1: HTTP methods supported by the clinical data repository REST API.....	18
Table 2: The pagination concept applied to large result sets returned by the clinical data repository	19
Table 3: The includable attribute demonstrated on the basis of the groups resource implemented by the clinical data repository	20
Table 4: The possible return states used by the clinical data repository to indicate a successful completion of a request	21
Table 5: The possible return states used by the clinical data repository to indicate an unsuccessful completion of a request	21
Table 6: The template used to describe the API endpoint resources of the clinical data repository...	22
Table 7: Information for calling storeTool web service	48
Table 8: Information for calling getAllTools web service.....	49
Table 9: Information for calling getToolById web service	50
Table 10: Information for calling getLatestToolByToolName web service	51
Table 11: Information for calling getPreviousVersions web service.....	52
Table 12: Information for calling deleteToolById web service	53
Table 13: Information for calling storeParameter web service	54
Table 14: Information for calling deleteParameter web service	55
Table 15: Information for calling getParametersByToolId web service.....	56
Table 16: Information for calling getMandatoryParametersByToolId web service.....	57
Table 17: Information for calling getInputParametersByToolId web service	58
Table 18: Information for calling getOutputParametersByToolId web service	59
Table 19: Information for calling storeProperty web service	60
Table 20: Information for calling getAllProperties web service	61
Table 21: Information for calling getPropertyById web service	62
Table 22: Information for calling storePropertyValue web service	63
Table 23: Information for calling deletePropertyValue web service	64
Table 24: Information for calling getPropertyValuesByToolId web service.....	65
Table 25: Information for calling deletePropertyById web service	65
Table 26: Information for calling storeReference web service.....	66
Table 27: Information for calling deleteReferenceById web service.....	68
Table 28: Information for calling getReferencesByToolId web service	69
Table 29: Information for calling getAxes web service.....	70
Table 30: Information for calling storeFile web service.....	71

Table 31: Information for calling deleteFile web service.....	72
Table 32: Information for calling getFileById web service.....	73
Table 33: Information for calling getFilesOfKind web service	74
Table 34: Information for calling storeTrial web service	77
Table 35: Information for calling getAllTrials web service.....	78
Table 36: Information for calling getTrialById web service	79
Table 37: Information for calling getTrialByModelId web service.....	80
Table 38: Information for calling deleteTrialById web service	81
Table 39: Information for calling storeExperiment web service.....	82
Table 40: Information for calling getAllExperimentsByTrialId web service	83
Table 41: Information for calling getExperimentById web service.....	84
Table 42: Information for calling getExperimentStatusById web service.....	85
Table 43: Information for calling getExperimentsByStatus web service	86
Table 44: Information for calling updateExperimentStatus web service	87
Table 45: Information for calling deleteExperimentById web service.....	88
Table 46: Information for calling storeMiscellaneousParameter web service	89
Table 47: Information for calling getAllMiscellaneousParameters web service.....	90
Table 48: Information for calling getAllMiscellaneousParametersByExperimentId web service	91
Table 49: Information for calling getMiscellaneousParameterById web service	92
Table 50: Information for calling deleteMiscellaneousParameterById web service	93
Table 51: Information for calling storeSubject web service	93
Table 52: Information for calling deleteSubjectById web service	95
Table 53: Information for calling getAllSubjects web service.....	95
Table 54: Information for calling getSubjectById web service	96
Table 55: Information for calling storeTrReference web service	97
Table 56: Information for calling getAllTrReferences web service	99
Table 57: Information for calling getTrReferencesByTrialId web service	100
Table 58: Information for calling getTrReferencesByExperimentId web service	101
Table 59: Information for calling deleteTrReferenceById web service	102
Table 60: Information for calling storeLinkToReference web service	102
Table 61: Information for calling deleteReferenceLinkById web service	103
Table 62: Information for calling getTrAxes web service	104
Table 63: Information for calling storeTrFile web service	105
Table 64: Information for calling deleteTrFile web service	107
Table 65: Information for calling getTrFileById web service	107

Table 66: Information for calling getTrLatestFilesBySubjectId web service	108
Table 67: Information for calling getTrFilesOfKind web service	109
Table 68: Information for calling getTrPreviousVersions web service	110

1 Executive Summary

The co-ordinated management of data and models is crucial for cancer studies in CHIC, as this project is developing clinical trial driven tools, services and secure infrastructure that support the creation of multiscale cancer hyper-models. The latter are defined as choreographies of component models, each one describing a biological process at a characteristic spatiotemporal scale, and of relation models/metamodels defining the relations across scales.

The development of a secure and function-rich hypermodelling infrastructure consisting primarily of a hypermodelling editor and a hypermodelling execution environment is a central generic VPH geared objective of CHIC. In order to render models developed by different modellers semantically interoperable, an infrastructure for semantic metadata management must interoperate with repositories for clinical and in-silico trial data, as well as models and tools. Facilitated operations will range from automated dataset matching to model merging and managing complex simulation workflows.

The following entities have now been developed: a hypermodel repository, a hypermodel-driven clinical data repository, a distributed metadata repository and an in silico trial repository for the storage of executed simulation scenarios, an image processing toolkit, a visualization toolkit and cloud and virtualization services. The key aim of the CHIC infrastructure is to ensure that these distinct entities are able to communicate effectively in support of hypermodelling studies. To that end, this document is focused on providing a technical account of the interfaces for four of these resources, namely: the hypermodel repository, the clinical data repository, the metadata repository and the in silico trial repository. In particular:

Clinical data repository

The fully implemented API is presented, including the HTTP methods used, the applied pagination concept, resource addresses, accepted parameters, possible requests, responses and errors. Additionally, a first version of the external timeline tool developed by BED has been integrated into the data repository interface. Currently, the clinical data repository stores nephroblastoma and lung data but is fully prepared for brain and other data.

Model & tool repository

The model & tool repository is already able to store models and tools such as linkers and data transformation tools which are needed for the construction of hyper-models. Apart from models and tools, this repository is also able to store descriptive information of models, parameters (input and output), properties, references and files. This deliverable provides a concrete documentation for all CHIC partners so as to be able to consume model/tool repository's web services in order to view, save and delete repository's content.

In-silico trial repository

This section discusses how to access information about the conducted *in silico* trial, such as hyper-model's input, original input (medical data without any processing), hyper-model's output and information about the hyper-model used in the trial. Specifically, the authentication mechanism for repository web services, as well as relevant API calls are illustrated and documented.

Semantic metadata repository

The use of W3C approved formats for metadata triples provides a semantically meaningful and highly standardised way of storing fact about CHIC resources. In this deliverable, we describe the interfaces to the RICORDO-based metadata management suite, which includes: the RDFStore Templating Engine, the Web Ontology Language Knowledge Base (OWLKB), as well as the Local Ontology Lookup Service (LOLS).

2 Introduction

2.1 Purpose of this document

The purpose of this document is to provide users and developers of the CHIC hypermodeling infrastructure with a detailed overview of the methods available for interaction with the four key repositories of the CHIC framework, namely: those for clinical data, models & tools, in-silico trial data, as well as semantic metadata. An introductory note for the interface functionalities associated with these four resources is provided below.

CLINICAL DATA

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, before being accepted and stored by the clinical data repository. In its prototype implementation, which has been described in Deliverable “8.2 – Prototype implementation of the CHIC repositories”, the clinical data repository is already able to import and export data with the help of the simple and user-friendly web-based user interface. 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. In the end, the clinical data repository will contain for each patient all the relevant medical data including imaging data, clinical data, histological data and genetic data. To achieve a loosely coupled exchange of data between applications, the clinical data repository makes use of the REST (Representational State Transfer) architectural principle. 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 allows other services of the CHIC environment to programmatically access all the relevant medical data including imaging data, clinical data, histological data and genetic data.

In this deliverable we present the fully implemented API based on the general concepts introduced in the first deliverable “8.1 – Design of the CHIC repositories”. The description of the API includes the HTTP methods used, the applied pagination concept, resource addresses, accepted parameters, possible requests, responses and errors. Ultimately, the functionalities offered by the API match the ones offered by the web-based user interface. Additionally, a first version of the external timeline tool developed by BED has been integrated into the data repository interface. The timeline tool itself leverages the functionalities provided by the clinical data repository REST API. All objects can be displayed within the graphical environment and the datasets can be directly downloaded from the timeline interface. All components of the clinical data repository have been successfully deployed to the private cloud infrastructure provided by FORTH. The API documentation can be accessed by the following URL <https://cdr-chic.ics.forth.gr/api/help>. Furthermore, continued data exchange between the clinicians and researchers through the Trusted Third Party could be successfully conducted. Currently, the clinical data repository stores nephroblastoma and lung data but is fully prepared for brain and other data.

MODELS & TOOLS

The model & tool CHIC repository stores cancer models, spanning from models of generic fundamental biomechanisms involved in cancer progression and treatment response, such as cell cycle and cell metabolism, to complex multiscale models of various types of cancer. In its prototype implementation, which has been described in Deliverable “8.2 – Prototype implementation of the CHIC repositories”, the model & tool repository is already able to store models and tools such as linkers and data transformation tools which are needed for the construction of hyper-models. Apart

from models and tools, this repository is also able to store descriptive information of models, parameters (input and output), properties, references and files. Since the back-end and the graphical user interface of the repositories has been described in Deliverable 8.2, this deliverable focuses on the implementation of the web services which are needed in order for the contents of model/tool repository to be exposed to other CHIC components, such as the hypermodelling editor and the hypermodelling execution framework. Consequently, this deliverable aims to be a concrete documentation for all CHIC partners so as to be able to consume model/tool repository's web services in order to view, save and delete repository's content.

IN-SILICO TRIAL DATA

Apart from the web services of model/tool repository, this deliverable presents the implementation and documentation of *in silico* trial repository's web services. *In silico* trial repository is used for the persistent storage of the simulation scenarios and the *in silico* predictions. *In silico* trial repository's web services are needed so that the *in silico* trial repository will successfully interact with the hypermodelling executional environment, in order to automatically store the outcome of the simulation and all the related data that constitute the *in silico* trial. Moreover, information about the conducted *in silico* trial, such as hyper-model's input, original input (medical data without any processing), hyper-model's output and information about the hyper-model used in the trial, can be automatically accessed, updated and saved by other CHIC components through *in silico* trial repository's web services.

Implementation of interfaces of model/tool and *in silico* trial repositories are presented in this deliverable in the same chapter (chapter 4) as both repositories reside in the same virtual machine, use the same web application servers and have been both developed using the same web application framework (Django). Chapter 4.1 explains the reason why the usage of web services, and especially the usage of RESTful application programming interfaces, provides major benefits to CHIC technical architecture. Chapter 4.2 presents the brokered authentication mechanism for model/tool and *in silico* trial repository web services. More specifically, chapter 4.2 aims to being the reference point for authentication procedure in the aforementioned repositories. The updated entity relationship diagrams of model/tool and *in silico* trial repositories on which the application programming interfaces are based, are presented in chapters 4.3 and 4.5 respectively. Last but not least, the chapters 4.4 and 4.6 aim at presenting all the necessary information and documentation, which is essential in order for the client to call model/tool, and *in silico* trial repository web services respectively. The description of the web service, the HTTP method used, the parameters, the returned object and the url of the service are all described in chapter 4.4 for model/tool repository and in chapter 4.6 for *in silico* trial repository.

SEMANTIC METADATA

In addition to the data itself, CHIC also stores and generates metadata: data about data. In past years, an *ad hoc* approach has often been taken to metadata, and this has resulted in a lack of interoperability. CHIC will take a different approach. By using standard W3C approved formats for storing and transmitting metadata, CHIC achieves a high level of interoperability, enabling future research to make use of CHIC's metadata with greater ease. Metadata is stored in the form of so-called triples, the building block of RDF (the Resource Description Framework), a semantically meaningful and highly standardised way of storing facts, which forms the backbone of the semantic web. Each triple has three components, which mimic the basic structure of the prototypical English sentence: the first component is the 'subject'; the second component is the 'relation' (analogous to the English verb); the third component is the 'object'. These so-called triples play the role of atoms in the physics of knowledge representation: arbitrarily sophisticated fragments of knowledge can be

broken down into these triples, and yet a machine of fixed sophistication suffices to parse and understand them. In the future, when other researchers want to make usage of CHIC metadata, it will not be necessary to spend time and money reverse-engineering CHIC-specific data storage formats. Rather, any of a number of off-the-shelf programs for dealing with triples can be put straight to use.

A suite of software has been developed for CHIC to facilitate creation, storage, and querying of CHIC's metadata. This suite, consisting of three components, is collectively known as RICORDO. Its three components are:

- 1) The RDFStore Templating Engine, which facilitates creation of query templates so that the end-user doesn't need to know the complicated querying languages that are ordinarily required in order to query RDF triple databases;
- 2) The Web Ontology Language Knowledge Base (OWLKB), which provides a convenient API for performing sophisticated semantic reasoning queries over the ontologies behind CHIC's metadata; and
- 3) The Local Ontology Lookup Service (LOLS) which provides lightning-fast translation between standardised (but not human readable) identifier strings used for triple stores, and human-readable labels describing them.

These three components, collectively comprising the RICORDO suite, each provide their services via APIs, engineered to facilitate integration into the various other components of CHIC. The servers, which serve the RICORDO APIs, have been installed on CHIC machinery and they are running smoothly, ready to handle whatever metadata tasks are needed.

3 RESTful web services and authorization

3.1 *Repositories make use of RESTful application programming interfaces*

CHIC repositories, are already capable of allowing users (clinicians, researchers, etc.) to interact with them through the user interface. Even if the user interface is one of the most important parts of the repositories, as it determines how easily the user can interact with them, it does not on its own meet the needs of the CHIC project, of making use of a service oriented architecture (SOA). SOA is an approach used to create an architecture based upon the use of services. Services (such as RESTful web services) carry out some small function, such as producing data, validating a client, or providing simple analytical services.

The use of web services in CHIC provides major benefits:

- In contrast to the use of large applications, which tend to be “information silos”, that cannot readily exchange information with each other, the use of finer-grained software services gives freer information flow within the CHIC project. Integrating major applications is often expensive. SOA can save integration costs.
- Organizing internal software as services makes it easier to expose its functionality externally. This leads to increased visibility that can have business value as, for example, when the model repository makes the tracking of a new model’s storage visible to the users (researchers, clinicians, etc.), increasing researcher’s satisfaction and reducing the costly overhead of status enquiries.
- The processes within the CHIC project are mostly dependent on the supporting software. It can be hard to change large, monolithic programs. This can make it difficult to change.
- Business processes are often dependent on their supporting software. It can be hard to change large, monolithic programs. This can make it difficult to make changes to clinical scenarios in order to meet new requirements (arising, for example, from changes in legislation) or to take advantage of new opportunities in *in Silico* Oncology. A service-based software architecture is easier to change – it has greater organizational flexibility, enabling it to avoid penalties and reap commercial advantage. (This is one of the ways in which SOA can make a project more “agile”).

In this context, CHIC repositories make use of RESTful web services. An overview of the main entities and the design of the standardized interfaces of the repositories has been described in deliverable “D10.2 - Design of the orchestration platform, related components and interfaces”. While the aforementioned deliverable describes the design of the interfaces, this deliverable aims at describing the implementation of interfaces by using RESTful web services.

REST (Representational State Transfer) is an architectural style and an approach to communications that is often used in the development of web services. By using a decoupled architecture, REST has become a popular building style for cloud-based APIs, such as those provided by Amazon, Microsoft and Google. When web services use REST architecture, like in CHIC project, they are called RESTful APIs (Application Programming Interfaces). REST typically runs over HTTP (Hypertext Transfer Protocol) and emphasizes that interaction between clients and services is enhanced by having a limited number of operations. Flexibility is provided by assigning resources their own unique URIs (Universal Resource Identifiers). As each operation uses a specific HTTP method (POST, GET, PUT, DELETE), REST avoids ambiguity. CHIC Repositories make use of the 4 HTTP methods, POST, GET, PUT, DELETE in order to perform create ,read, update and delete operations respectively.

It has been decided that CHIC repositories should make use of RESTful application programming interfaces, after taking into account the following:

- RESTful web services are easy to leverage by most tools, including those that are free and inexpensive. (No expensive tools require to interact with the web service).
- RESTful web services are easy to scale. Thus, REST is often chosen as the architecture for services that are exposed via the Internet (like Facebook, Twitter, and most public cloud providers).
- REST uses a small message format and as a result it provides better performance, as well as lower costs over time. Moreover there is no intensive processing required.
- REST is designed for use over the Open Internet/Web, making it a better choice for Web scale applications, and certainly for cloud-based platforms.

3.2 Authentication in repository web services

CHIC repository web services are consumed by non browser clients, like the hypermodelling execution framework, or the editor. As web services are typically stateless and a different security context can be required for each message, message level security is preferred. For REST web services, CHIC makes use of WS-* security components. More specifically, in order for the client to access resources from the repositories, it is essential to call the WS-Trust Secure Token Service to request a SAML security token and pass it through the HTTP Authorization header to repository REST services. More specifically, the client needs to send a SOAP (Simple Object Access Protocol) request containing an RST (Request Security Token) to the STS. The STS then returns the identity assertion as a SAML token embedded in a RSTR (Request Security Token Response). The base 64 encoded compressed SAML token can then be passed to the rest service, through the http authorization header. The sequential diagram which depicts the brokered authentication mechanism for CHIC repository web services is presented in figure 1.

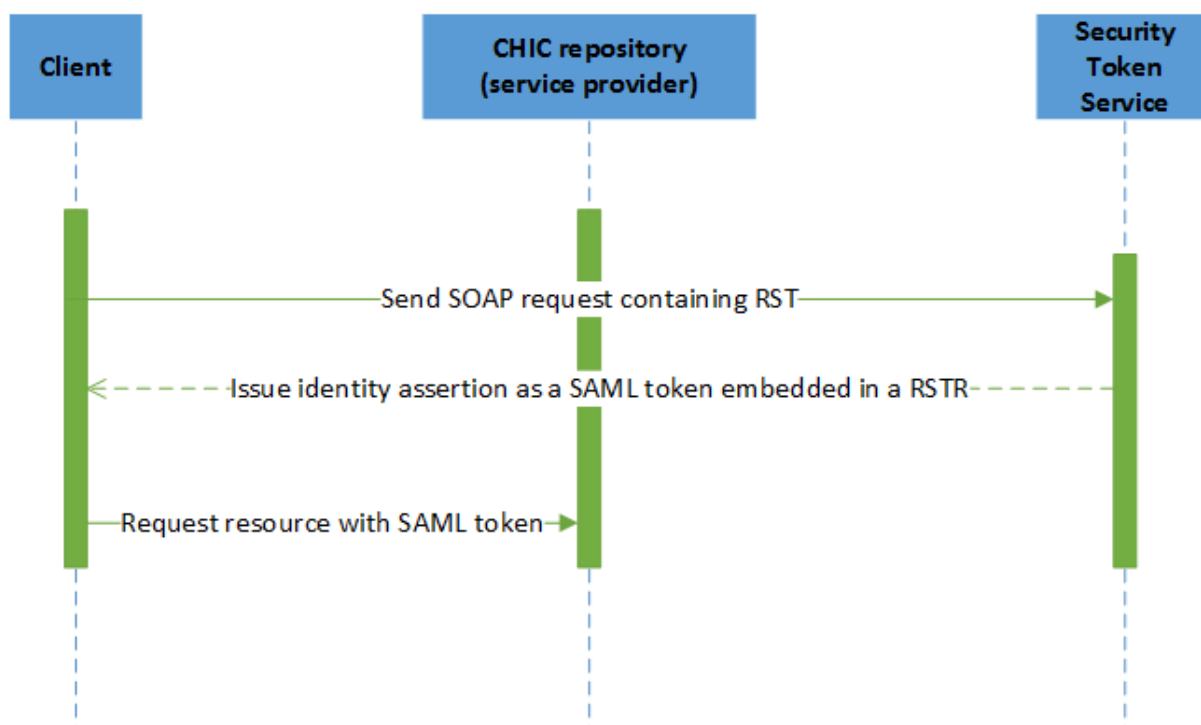


Figure 1: Brokered authentication for CHIC repository web services

As shown in figure 1, CHIC repositories are known as service providers (SP) because both of them are entities that provide web services. Consequently, three different components are engaged in the process of authorization concerning the repositories. These components are the client, the service provider (model/tool, *in silico* trial, clinical data, and semantic metadata repository) and the security token service. A brief description of these components is presented below:

- **Service Provider (SP)**

A service provider is an entity that provides web services. The service provider relies on a trusted security token service for authentication and authorization. In SAML, the XML-standard for exchanging data, the security domains that information is passed between are a service provider (SP) and an identity provider (IdP). SAML's service provider (SP) depends on receiving assertions from a SAML authority, or asserting party, a SAML identity provider.

- **Identity Provider (IdP)**

An identity provider (IdP), sometimes called an identity service provider or identity assertion provider, is an online service or website that authenticates users on the internet by means of security tokens, one of which is SAML 2.0. In the WS-Federation model an identity provider is a security token service (STS). Service providers depend on an identity provider or security token service to do the user authentication.

- **Security Token Service (STS)**

Security tokens, sometimes called identity tokens, authentication tokens or even software tokens, play a major role in identity management as they are the device of choice for authenticating and authorizing a user's identity or "digital identity". A security token service (STS) is the web service that issues security tokens. In essence, an STS is a WS-Trust identity provider and a SAML assertion in WS-Trust is a kind of security token.

Before CHIC repositories serve the non browser client (client calling web services), it has to be ensured that the client is authorized to access the content of the repositories. Consequently the repositories should proceed to the following steps:

- **CHIC repositories should verify XML signature**

As already reported, the client will authenticate to the security token service which generates the SAML authentication assertion (SAML token) to prove that it has authenticated the client. The security token service will sign the assertion as proof that only it could have signed the assertion, and also to guarantee the integrity of the assertion. Subsequently CHIC repositories verify xml signature by using an xml security library in order to ensure that that the SAML token has been provided by CHIC security token service component. Model and *in silico* trial repositories make use of an xml security library named dm.xmlsec.binding1.3.2.

- **CHIC repositories should check audience element of SAML token.** Audience element is a validity condition for an assertion. In particular it declares that the assertion's semantics are only valid for the relying party named by URI in that element. Consequently, the client is authorized to CHIC repository web services if and only if the xml audience element has the correct value which is usually the urn of the server in which the repository is stored. Audience element is depicted in figure 2.

- **CHIC repositories should check NotOnOrAfter and NotBefore attributes of SAML token conditions element.** These attributes declare that the assertion's semantics are only valid for a time period which starts according to NotBefore value and ends according to NotOnOrAfter

value. In other words, if the date defined in the value of NotBefore attribute is after the current date of server or the date defined in the value of NotOnOrAfter attribute is before the current date of server, then the client is not authorized to consume repository web services. SAML token conditions element is depicted in figure 2.

```
<saml2:Conditions NotOnOrAfter="2015-07-18T08:25:06.711Z" NotBefore="2015-07-17T08:25:06.711Z">
  - <saml2:AudienceRestriction>
    <saml2:Audience>https://139.91.210.27</saml2:Audience>
  </saml2:AudienceRestriction>
</saml2:Conditions>
```

↑ NotOnOrAfter attribute ↑ NotBefore attribute
 ↑
 ↑

In order for the client to consume repository web services, the value of xml audience element should be the same with the urn of the server in which the repository is stored.

Figure 2: Example of conditions element in SAML token

After the verification of the XML signature and the checking of the conditions element of SAML token, the client is ready to consume CHIC repository web services which are described in the following chapters of this document.

4 Interfaces of the Clinical Data Repository

4.1 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 chapter 3 of deliverable D8.2. 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.

The focus in this deliverable is on the interfaces of the clinical data repository. Foremost, the authentication mechanism is explained in chapter 4.2. All general concepts introduced in deliverable D8.1 have been translated to the use case format in deliverable D8.2 and are now transformed into interfaces in chapter 4.4.

4.2 Authentication

The clinical data repository makes 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 clinical data repository (Service Provider) itself but rather by the CHIC authentication broker (Identity Provider) to support Single Sign-On (SSO). This procedure is called brokered authentication.

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

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

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

4.3 Domain model

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

- The RawImage and SegmentationImage entities have been replaced by the PreviewImage entity. It contains the information about the location of an extracted thumbnail image stored on the clinical data repository file system.
- The snapshot information about a patient has been extracted from the Patient entity to a standalone entity called PatientSnapshot. All information about the patient such as age, height, weight, sex, etc. is stored by this entity. The Patient entity is responsible for the system-wide unique pseudonym/identifier only.
- GenomicSeries, GenomicSample and GenomicPlatform entities have been added to support genomic data.

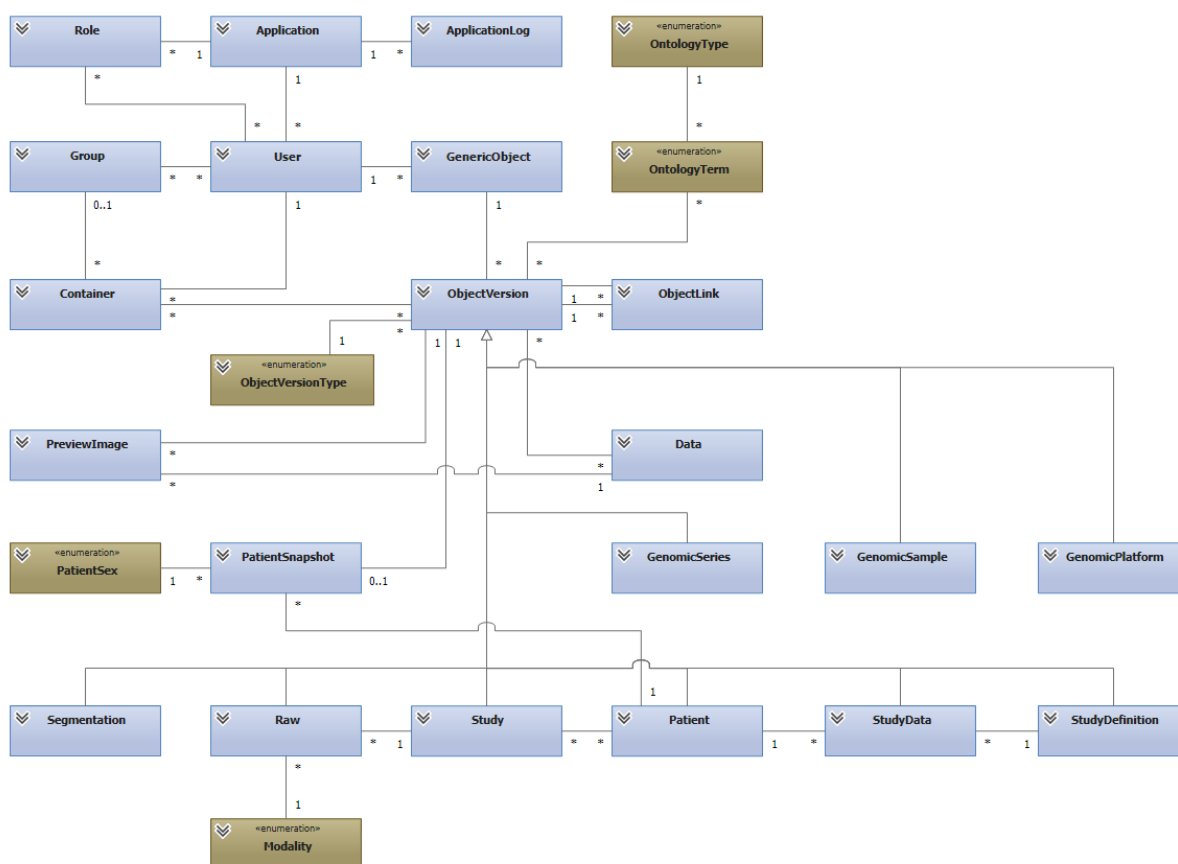


Figure 3: The domain model of the clinical data repository with domain classes (blue), domain enumerations (brown) and their relationships represented as connecting lines

4.4 Web-based user interface

The prototype implementation of the web-based user interface offers a main view illustrated in figure 4 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.

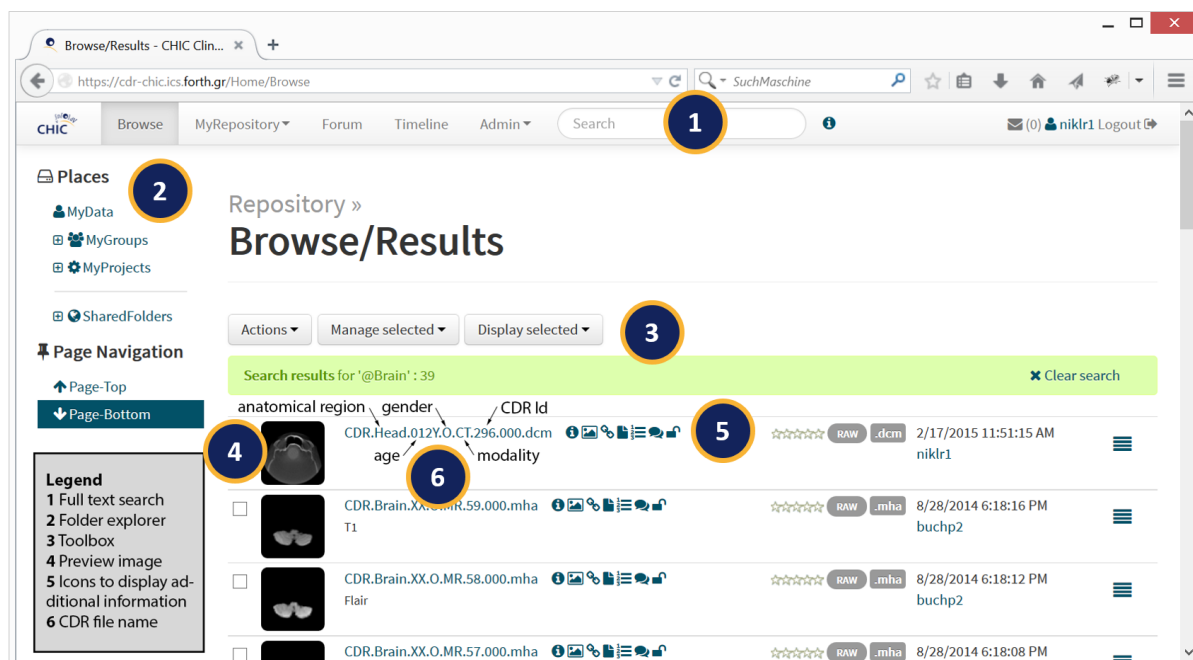


Figure 4: The web-based user interface main view of the clinical data repository

4.5 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.

4.5.1 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 1: 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). 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. It is described more detailed in chapter 4.5.5 Annotation & Search.

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. Only the UTF-8 character encoding is supported for both requests and responses.

4.5.2 Pagination

Pagination is the process of dividing a document into discrete pages in order to keep the loading time on 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 2: The pagination concept applied to large result sets returned by the clinical data repository

Parameter name	Value type	Default value	Description
rpp	int	25	Defines the amount of included results per page. Allowed values: 10, 25, 50, 100, 250, 500
page	int	0	Defines the current page index. Allowed values: 0, 1, 2, ...
Example Request			
GET https://cdr-chic.ics.forth.gr/api/objects?rpp=25&page=3			
Example Response			
<pre>{ "totalCount": 99, "pagination": { "rpp": 25, "page": 3 }, "items": [...], "nextPageUrl": "https://cdr-chic.ics.forth.gr/api/objects?rpp=25&page=4" }</pre>			

4.5.3 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 3: 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.ics.forth.gr/api/groups/1			
Example Response without include			
<pre>{ "id": 1, "name": "Test group", "chief": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/users/2" }, "selfUrl": "https://cdr-chic.ics.forth.gr/api/groups/1" }</pre>			
Example Request with include			
GET https://cdr-chic.ics.forth.gr/api/groups/1?include=chief			
Example Response with include			
<pre>{ "id": 1, "name": "Test group", "chief": { "id": 2, "username": "niklr1", "selfUrl": "https://cdr-chic.ics.forth.gr/api/users/2" }, "selfUrl": "https://cdr-chic.ics.forth.gr/api/groups/1" }</pre>			

4.5.1 Requests, Responses, and Errors

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

Table 4: 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 5: 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 user is not authorized to get a requested object or to do the request updated operation.
404 Not Found	The URL is correct, but the requested object does not (or no longer) exist.
405 Method Not Allowed	Different action methods may be restricted to one or more of the HTTP methods (GET, PUT, or POST). The received request uses one that is not allowed with the action method specified in the URL. In this case, other parts of the URL are not validated.
500 Internal Server Error	When a method causes an exception that has no adequate handling in the method itself. Developers of client systems are kindly requested to report these response states to the developing team and to transmit information about the respective request and the response objects.
501 Not Implemented	May occur during development. The requested action has been specified and documented, but not yet implemented.

4.5.2 Resource description template

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

Table 6: 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.		

4.5.3 Upload & Versioning


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.

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.

As described in chapter “3 General workflow for data upload” of deliverable D8.2 data providers do not directly upload to the clinical data repository. In a first step the data is uploaded from the hospital to the Trusted Third Party (TTP). In a second step the data is uploaded from the TTP to the clinical data repository.

POST	upload	
------	--------	---

Description	<p>Uploads a supported file to the clinical data repository (e.g. DICOM, Metalmage, Analyze, Nifti, CDISC ODM, MINiML, etc.)</p> <p>Exactly one file can be uploaded per request. This resource should be used to upload files smaller than 200MB.</p>
Content-Type	multipart/form-data
Example Request	
<pre>POST https://cdr-chic.ics.forth.gr/api/upload HTTP/1.1 Content-Type: multipart/form-data; boundary="--1234" Content-Length: 161300 --1234 Content-Disposition: form-data; filename=sample.dcm "content of sample.dcm" --1234--</pre>	
Example Response	
<pre>{ "file": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/files/1" }, "relatedObject": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1" } }</pre>	


POST	chunked_upload?chunk={chunk} chunked_upload/commit?filename={filename}		
Description	The chunked upload resources can be used in order to upload large files. Chunks can be any size up to 100 MB. A typical chunk is 4 MB. Using large chunks will mean fewer calls and faster overall throughput. However, whenever a transfer is interrupted, you will have to resume at the beginning of the last chunk, so it is often safer to use smaller chunks.		
Content-Type	multipart/form-data		
Parameters	chunk (int)	The number of the current chunk.	
	filename (string)	The filename of the chunked upload to be committed.	
Example Request			

1. Send a POST request equivalent to the one described in the upload resource to **https://cdr-chic.ics.forth.gr/api/chunked_upload?chunk=1** for the first chunk.
2. Repeatedly POST subsequent chunks by incrementing the number in the URL identifying the current chunk.
3. After the last chunk, POST to **https://cdr-chic.ics.forth.gr/api/chunked_upload/commit?filename=sample.dcm** to complete the upload.


Example Response

```
{
  "file": {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/files/1"
  },
  "relatedObject": {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1"
  }
}
```

The clinical data repository uses *GenericObjects* to extend the generic *ObjectVersion* concept. A new version of a dataset (*ObjectVersion*) will have the same *GenericObject* as the ancestor. If, for example, a clinical trial form has been update with new questions in the clinical trial software, a new version will be generated keeping the same unique identifier as the original object. Based on this identifier, a new version can be created in the clinical data repository. With this approach, the users will still have access to the initial data version (through the *GenericObject*), but the new version will be shown as the current version. Versioning is handled automatically by the upload resources. Once uploaded the objects can be accessed by the resources listed consecutively.

GET	objects objects?rpp={rpp}&page={page}&include={include}	
Description	Gets a paginated list of objects, which constitute the basic component of the system.	
Parameters	include (string) Allowed properties to be included: <ul style="list-style-type: none"> • License • Files • LinkedObjectRelations • OntologyItems • OntologyItemRelations • ObjectPreviews • ObjectGroupRights 	


	<ul style="list-style-type: none"> ObjectUserRights
Example Request	
GET https://cdr-chic.ics.forth.gr/api/objects?rpp=25&page=1 HTTP/1.1	
Example Response	
<pre>{ "totalCount": 26, "pagination": { "rpp": 25, "page": 1 }, "items": [{ ... see GET objects/{id} }], "nextPageUrl": null }</pre>	

OPTIONS	objects	
Description	Gets the available options for this resource. It includes information about enum types.	
Example Request		
OPTIONS https://cdr-chic.ics.forth.gr/api/objects HTTP/1.1		
Example Response		
<pre>{ "types": [{ "key": 0, "value": "None" }, { "key": 1, "value": "RawImage" }, { "key": 2, "value": "SegmentationImage" }, { "key": 3, "value": "StatisticalModel" }, {</pre>		

```

    "key": 4,
    "value": "StudyDefinition"
  },
  {
    "key": 5,
    "value": "StudyData"
  },
  {
    "key": 6,
    "value": "SurfaceModel"
  }
  ...
],
...
}

```

GET	objects/{id}?include={include}		
Description	Gets the information of the object with the specified identifier.		
Parameters	id (integer)	The identifier of the object.	
	include (string)	Allowed properties to be included: <ul style="list-style-type: none">• See GET objects	
Example Request			
GET https://cdr-chic.ics.forth.gr/api/objects/1 HTTP/1.1			
Example Response			
<pre>{ "sliceThickness": 0.4, "spaceBetweenSlices": null, "kilovoltPeak": 120.0, "modality": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/modalities/3" }, "id": 1, "createdDate": "2015-08-24T12:41:03.9570000Z", "name": "CDR.Abdominal_aorta.050Y.M.CT.1.000.dcm", "description": null, "ontologyCount": 1, "type": 1, "downloadUrl": "https://cdr-chic.ics.forth.gr/api/objects/1/download", "license": null, "files": { "totalCount": 3, "pagination": { "rpp": 25, "page": 0 } }, }</pre>			

```


"items": [
  {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/files/2"
  },
  {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/files/3"
  },
  {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/files/4"
  }
],
"nextPageUrl": null
},
"linkedObjects": {
  "totalCount": 1,
  "pagination": {
    "rpp": 25,
    "page": 0
  },
  "items": [
    {
      "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/3"
    }
  ],
  "nextPageUrl": null
},
"linkedObjectRelations": {
  "totalCount": 1,
  "pagination": {
    "rpp": 25,
    "page": 0
  },
  "items": [
    {
      "selfUrl": "https://cdr-chic.ics.forth.gr/api/object-links/2"
    }
  ],
  "nextPageUrl": null
},
"ontologyItems": {
  "totalCount": 1,
  "pagination": {
    "rpp": 25,
    "page": 0
  },
  "items": [
    {
      "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/3789"
    }
  ],
  "nextPageUrl": null
},
"ontologyItemRelations": {
  "totalCount": 1,
  "pagination": {
    "rpp": 25,
    "page": 0
  },
  "items": [
    {

```


```

    "selfUrl": "https://cdr-chic.ics.forth.gr/api/object-
ontologies/0/13"
  },
  "nextPageUrl": null
},
"objectPreviews": [
  {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1/previews/0"
  },
  {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1/previews/1"
  },
  {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1/previews/2"
  }
],
"objectGroupRights": [
  {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/object-group-rights/49"
  },
  {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/object-group-rights/50"
  }
],
"objectUserRights": null,
"selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1"
}

```


PUT	objects/{id}	
Description	Updates the object with the specified identifier.	
Content-Type	application/json	
Parameters	id (integer)	The identifier of the object.
Notes	<p>Properties annotated with the “Settable” attribute can be updated.</p> <ul style="list-style-type: none">• Description• License• Modality• SegmentationMethod	
<pre>{ "sliceThickness": 0.4, "spaceBetweenSlices": null, "kilovoltPeak": 120.0, "modality": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/modalities/3" }, }</pre>		

```
"id": 1,
"createdDate": "2015-08-24T12:41:03.9570000Z",
"name": "CDR.Abdominal_aorta.050Y.M.CT.1.000.dcm",
"description": "CHIC test description",
...
"selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1"
}
```

DELETE	objects/{id}	
Description	Deletes the unpublished object with the specified identifier.	
Parameters	id (integer)	The identifier of the object.
Example Request		
DELETE https://cdr-chic.ics.forth.gr/api/objects/1 HTTP/1.1		
Example Response		
HTTP/1.1 204 No Content		


4.5.4 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 will make use of the meta-information stored in the files to automatically generate this linking.

GET	object-links/{id}	
Description	Gets the relation representing a link between two objects.	
Parameters	id (integer)	The identifier of the relation.
Example Request		
GET https://cdr-chic.ics.forth.gr/api/object-links/1 HTTP/1.1		

Example Response

```
{
  "id": 1,
  "description": "Link created based on patient information.",
  "object1": {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1"
  },
  "object2": {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/2"
  },
  "selfUrl": "https://cdr-chic.ics.forth.gr/api/object-links/1"
}
```


POST	object-links	
Description	Creates a link between the provided objects.	
Content-Type	application/json	
Example Request		
<pre>{ "object1": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1" }, "object2": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/2" } }</pre>		
Example Response		
<pre>{ "id": 1, "description": "Link created manually by demo", "object1": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1" }, "object2": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/2" }, "selfUrl": "https://cdr-chic.ics.forth.gr/api/object-links/1" }</pre>		


DELETE	object-links/{id}	
Description	Deletes the relation with the specified identifier representing a link between	


	two objects.
Parameters	id (integer) The identifier of the relation.
Example Request	
DELETE https://cdr-chic.ics.forth.gr/api/object-links/1 HTTP/1.1	
Example Response	
HTTP/1.1 204 No Content	

4.5.5 Annotation & 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). 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 will be used to store the semantic information. Web based queries based on SPARQL will be 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 will be included in a second step by integrating the RICORDO system. Based on these annotations it will be possible to conduct semantically driven search queries to find datasets containing the required anatomical structures or other properties.


OPTIONS	ontologies	
Description	Gets the available options for this resource. It includes information about enum types.	
Example Request		
OPTIONS https://cdr-chic.ics.forth.gr/api/ontologies HTTP/1.1		
Example Response		
<pre>{ "types": [{ "key": 0, "value": "FMA" }] }</pre>		

GET	ontologies/{type}?rpp={rpp}&page={page}		
Description	Gets a list of ontology items of the specified ontology.		
Parameters	type (integer)	The ontology type.	
Example Request			
GET https://cdr-chic.ics.forth.gr/api/ontologies/0 HTTP/1.1			
Example Response			
<pre>{ "totalCount": 1002, "pagination": { "rpp": 25, "page": 0 }, "items": [{ "id": 20394, "term": "Body", "type": 0, "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/20394" }, { "id": 7197, "term": "Liver", "type": 0, "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/7197" }, ...], "nextPageUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0?rpp=25&page=1" }</pre>			


GET	ontologies/{type}/{id}		
Description	Gets the information of the ontology item with the specified identifier.		
Parameters	type (integer)	The ontology type.	
	id (integer)	The identifier of the ontology item.	
Example Request			
GET https://cdr-chic.ics.forth.gr/api/ontologies/0/7197 HTTP/1.1			
Example Response			



```
{
  "id": 7197,
  "term": "Liver",
  "type": 0,
  "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/7197"
}
```

GET	ontologies/{type}/{id}		
Description	Gets the information of the ontology item with the specified identifier.		
Parameters	type (integer)	The ontology type.	
	id (integer)	The identifier of the ontology item.	
Example Request			
GET https://cdr-chic.ics.forth.gr/api/ontologies/0/7197 HTTP/1.1			
Example Response			
<pre>{ "id": 7197, "term": "Liver", "type": 0, "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/7197" }</pre>			

GET	object-ontologies/{type}/{id}		
Description	Gets the relation between an object and an ontology item.		
Parameters	type (integer)	The ontology type.	
	id (integer)	The identifier of the object ontology relation.	
Example Request			
GET https://cdr-chic.ics.forth.gr/api/object-ontologies/0/2 HTTP/1.1			
Example Response			
<pre>{ "id": 2, "position": 0, "type": 0, "object": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1"</pre>			

```
{,
  "ontologyItem": {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/7197"
  },
  "selfUrl": "https://cdr-chic.ics.forth.gr/api/object-ontologies/0/2"
}
```

PUT	object-ontologies/{type}/{id}		
Description	Updates the relation between the object and the ontology item.		
Parameters	type (integer)	The ontology type.	
	id (integer)	The identifier of the object ontology relation.	
Content-Type	application/json		
Example Request			
<pre>{ "id": 2, "position": 0, "type": 0, "object": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1" }, "ontologyItem": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/20394" }, "selfUrl": "https://cdr-chic.ics.forth.gr/api/object-ontologies/0/2" }</pre>			
Example Response			
<pre>{ "id": 2, "position": 0, "type": 0, "object": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1" }, "ontologyItem": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/20394" }, "selfUrl": "https://cdr-chic.ics.forth.gr/api/object-ontologies/0/2" }</pre>			


POST	object-ontologies/{type}	
------	--------------------------	---

Description	Creates a relation between the object and the ontology item.
Parameters	type (integer) The ontology type.
Content-Type	application/json
Example Request	
<pre>{ "position": 0, "type": 0, "object": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/2" }, "ontologyItem": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/7197" } }</pre>	
Example Response	
<pre>{ "id": 3, "position": 0, "type": 0, "object": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/2" }, "ontologyItem": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/7197" }, "selfUrl": "https://cdr-chic.ics.forth.gr/api/object-ontologies/0/3" }</pre>	

A URI with a \$filter System Query Option identifies a subset of the entries from the collection of entries identified by the resource path section of the URI. The subset is determined by selecting only the entries that satisfy the predicate expression specified by the query option.

The expression language that is used in \$filter operators supports references to properties and literals. The literal values can be strings enclosed in single quotes, numbers and boolean values (true or false) or any of the additional literal representations.


An example of filtering the ontology terms is given below and more examples can be found on the official OData documentation.

GET	ontologies/{type}?\$filter={filter}	
Description	Gets a filtered list of ontology items of the specified ontology.	

Parameters	type (integer) The ontology type. filter (string) The predicate expression used to filter the list.
Example Request	
GET https://cdr-chic.ics.forth.gr/api/ontologies/0? \$filter=startswith(Term, 'Right') HTTP/1.1	
Example Response	
<pre>{ "totalCount": 72, "pagination": { "rpp": 25, "page": 0 }, "items": [{ "id": 5832, "term": "Right lymphatic duct", "type": 0, "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/5832" }, { "id": 5831, "term": "Right bronchomediastinal lymphatic trunk", "type": 0, "selfUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0/5831" }, ...], "nextPageUrl": "https://cdr-chic.ics.forth.gr/api/ontologies/0?\$filter=startswith(Term, 'Right') &rpp=25&page=1" }</pre>	

4.5.6 Validation

To ensure a high level of quality to the data stored in the repository, the system will support 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.


GET	objects/unpublished objects/unpublished?rpp={rpp}&page={page}&include={include}	
Description	Gets a paginated list of unpublished / not validated objects of the current user.	

Example Request

GET <https://cdr-chic.ics.forth.gr/api/objects/unpublished> HTTP/1.1

Example Response

```
{
  "totalCount": 3,
  "pagination": {
    "rpp": 25,
    "page": 1
  },
  "items": [
    {
      ...
      see GET objects/{id}
    }
  ],
  "nextPageUrl": null
}
```


GET	objects/published objects/published?rpp={rpp}&page={page}&include={include}	
Description	Gets a paginated list of published / validated objects of the current user.	
Example Request		
GET https://cdr-chic.ics.forth.gr/api/objects/published HTTP/1.1		
Example Response		
<pre>{ "totalCount": 3, "pagination": { "rpp": 25, "page": 1 }, "items": [{ ... see GET objects/{id} }], "nextPageUrl": null }</pre>		

PUT	objects/{id}/publish	
-----	----------------------	---

Description	Publishes / validates the object with the specified identifier.
Example Request	
PUT https://cdr-chic.ics.forth.gr/api/objects/1/publish HTTP/1.1	
Example Response	
<pre>{ "id": 1, ... "selfUrl": "https://demo.virtualskeleton.ch/api/objects/1" }</pre>	

4.5.7 Data organization


The clinical data repository allows each user to freely organize the data into his/her desired folder structure for easy access to the data needed for his/her research. 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. A folder structure created by one user can be directly shared to others. Modifications made by one user will immediately be visible in the folder of the other collaborators. The mechanism should allow efficient collaboration between modellers working on the same tumour model. 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.

GET	folders folders?rpp={rpp}&page={page}&include={include}	
Description	Gets a paginated list of folders.	
Parameters	include (string) Allowed properties to be included: <ul style="list-style-type: none">• ParentFolder• ChildFolders• ContainedObjects• ContainedObjectRelations• FolderGroupRights• FolderUserRights	
Example Request		
GET https://cdr-chic.ics.forth.gr/api/folders HTTP/1.1		
Example Response		
{		

```

"totalCount": 6,
"pagination": {
  "rpp": 25,
  "page": 0
},
"items": [
  {
    ...
    see GET folders/{id}
  }
],
"nextPageUrl": null
}


```

GET	folders/{id}?include={include}		
Description	Gets the information of the folder with the specified identifier.		
Parameters	id (integer)	The identifier of the folder.	
	include (string)	Allowed properties to be included: <ul style="list-style-type: none">• See GET folders	
Example Request			
GET https://cdr-chic.ics.forth.gr/api/folders/3 HTTP/1.1			
Example Response			
<pre>{ "id": 3, "name": "Test1", "level": 2, "parentFolder": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/2" }, "childFolders": { "totalCount": 1, "pagination": { "rpp": 25, "page": 0 }, "items": [{ "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/4" }], "nextPageUrl": null }, "containedObjects": { "totalCount": 1, "pagination": { "rpp": 25,</pre>			

```


    "page": 0
  },
  "items": [
    {
      "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1"
    }
  ],
  "nextPageUrl": null
},
"containedObjectRelations": {
  "totalCount": 1,
  "pagination": {
    "rpp": 25,
    "page": 0
  },
  "items": [
    {
      "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/3/objects/1"
    }
  ],
  "nextPageUrl": null
},
"folderGroupRights": null,
"folderUserRights": null,
"selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/3"
}


```

PUT	folders/{id}	
Description	Updates or moves the folder with the specified identifier.	
Parameters	id (integer)	The identifier of the folder.
Content-Type	application/json	
Notes	Properties annotated with the “Settable” attribute can be updated. <ul style="list-style-type: none">• Name• ParentFolder	
Example Request		
<pre>{ "id": 3, "name": "Test2", "parentFolder": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/2" }, "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/3" }</pre>		


Example Response


```
{
  "id": 3,
  "name": "Test2",
  "level": 2,
  "parentFolder": {
    "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/2"
  },
  ...
  "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/3"
}
```

POST	folders	
Description	Creates a new folder.	
Content-Type	application/json	
Example Request		
<pre>{ "name": "Test3", "parentFolder": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/2" } }</pre>		
Example Response		
<pre>{ "id": 6, "name": "Test3", "level": 2, "parentFolder": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/2" }, ... "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/6" }</pre>		


DELETE	folders/{id}	
Description	Deletes the folder with the specified identifier.	
Parameters	id (integer)	The identifier of the folder.

Example Request
DELETE https://cdr-chic.ics.forth.gr/api/folders/6 HTTP/1.1
Example Response
HTTP/1.1 204 No Content

GET	folder-objects/{id} folder-objects/{id}?include={include}	
Description	Gets the relation between a folder and an object.	
Parameters	<div><div>id (integer)</div><div>The identifier of the relation.</div></div> <div><div>include (string)</div><div>Allowed properties to be included:</div><div><ul style="list-style-type: none">• RelatedFolder• RelatedObject</div></div>	
Example Request		
GET https://cdr-chic.ics.forth.gr/api/folder-objects/1 HTTP/1.1		
Example Response		
<pre>{ "id": 1, "relatedFolder": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/1" }, "relatedObject": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/1" } "selfUrl": "https://cdr-chic.ics.forth.gr/api/folder-objects/1" }</pre>		


POST	folder-objects	
Description	Creates a relation between a folder and an object.	
Content-Type	application/json	
Example Request		
<pre>{ "relatedFolder": {</pre>		

<pre>"selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/1" }, "relatedObject": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/2" } }</pre>
Example Response
<pre>{ "id": 2, "relatedFolder": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/folders/1" }, "relatedObject": { "selfUrl": "https://cdr-chic.ics.forth.gr/api/objects/2" } "selfUrl": "https://cdr-chic.ics.forth.gr/api/folder-objects/2" }</pre>

DELETE	folder-objects/{id}	
Description	Deletes the relation between a folder and an object.	
Parameters	id (integer) The identifier of the relation.	
Example Request		
DELETE https://cdr-chic.ics.forth.gr/api/folder-objects/1 HTTP/1.1		
Example Response		
HTTP/1.1 204 No Content		

4.5.8 Download

The system provides download functionality for available datasets for further processing. A single object or multiple objects can be downloaded simultaneously. Even a whole folder structure can be downloaded in a compressed container file, preserving the folder structure. To save bandwidth and time the files represented by the objects or folders are compressed on the fly for the download.


GET	objects/{id}/download	
Description	Downloads the object with the specified identifier.	
Parameters	id (integer) The identifier of the object.	

Example Request

```
GET https://cdr-chic.ics.forth.gr/api/objects/1 HTTP/1.1
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/zip
Content-Length: 209327
Content-Disposition: attachment; filename=cdr_object_1_20150913_010452.zip
...
```

GET	folders/{id}/download	
Description	Downloads the folder with the specified identifier.	
Parameters	id (integer)	The identifier of the folder.
Example Request		
GET https://cdr-chic.ics.forth.gr/api/folders/1 HTTP/1.1		
Example Response		
HTTP/1.1 200 OK Content-Type: application/zip Content-Length: 209327 Content-Disposition: attachment; filename=cdr_folder_1_20150913_010452.zip ...		

4.6 Summary

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 allows other services of the CHIC environment to programmatically access all the relevant medical data including imaging data, clinical data, histological data and genetic data.

Additional data exchange between the clinicians and researchers through the Trusted Third Party could be successfully conducted. At this point the clinical data repository stores nephroblastoma and lung data but is fully prepared for brain and other data.

In this deliverable we have presented the fully implemented API based on the general concepts introduced in the first deliverable. The functionalities offered by the API match the ones offered by the web interface. Additionally, a first version of the external timeline tool developed by BED has been integrated into the data repository interface. The timeline tool itself leverages the

functionalities provided by the data repository REST API. All objects can be displayed within the graphical environment and the datasets can be directly downloaded from the timeline interface.

All components of the clinical data repository have been successfully deployed to the private cloud infrastructure provided by FORTH. The API documentation can be accessed by the following URL <https://cdr-chic.ics.forth.gr/api/help>

5 Implementation of the interfaces of model/tool and *in silico trial* repositories

5.1 Model/tool repository schema

The main entities of the model/tool repository are the tool, the parameter, and the property. The tool entity includes all the descriptive information of a tool, the parameter entity contains all the information regarding the input and output parameters of a tool and the property entity contains the properties that could characterize a tool. The actual value of a property for a specific tool is stored in the tool_property entity.

The entity relational diagram of the model/tool repository, which has been retrieved from deliverable “D8.1 – Design of the CHIC repositories”, is depicted, in the sake of completeness in figure 5:

As shown in the following figure, the entity relational diagram of model/tool repository remains the same as in deliverable D8.1, apart from the following modifications:

- Entity mr_parameter has been modified in order to be able to arrange the parameters of the models into 2 groups. Each parameter which belongs to the first group named “static” should get input before the execution of the model, or should provide output after the execution of the model. Each parameter which belongs to the second group named “dynamic” should get input or provide output during the execution of the model.
- Entity mr_tool has been modified in order to be able to store the url representing semantic information about the model.
- Column “version” has been moved from entity mr_file to entity mr_tool. This change was essential in order for the system to be able to associate an experiment stored in *in silico trial* repository with the specific model version which has been used for that experiment.

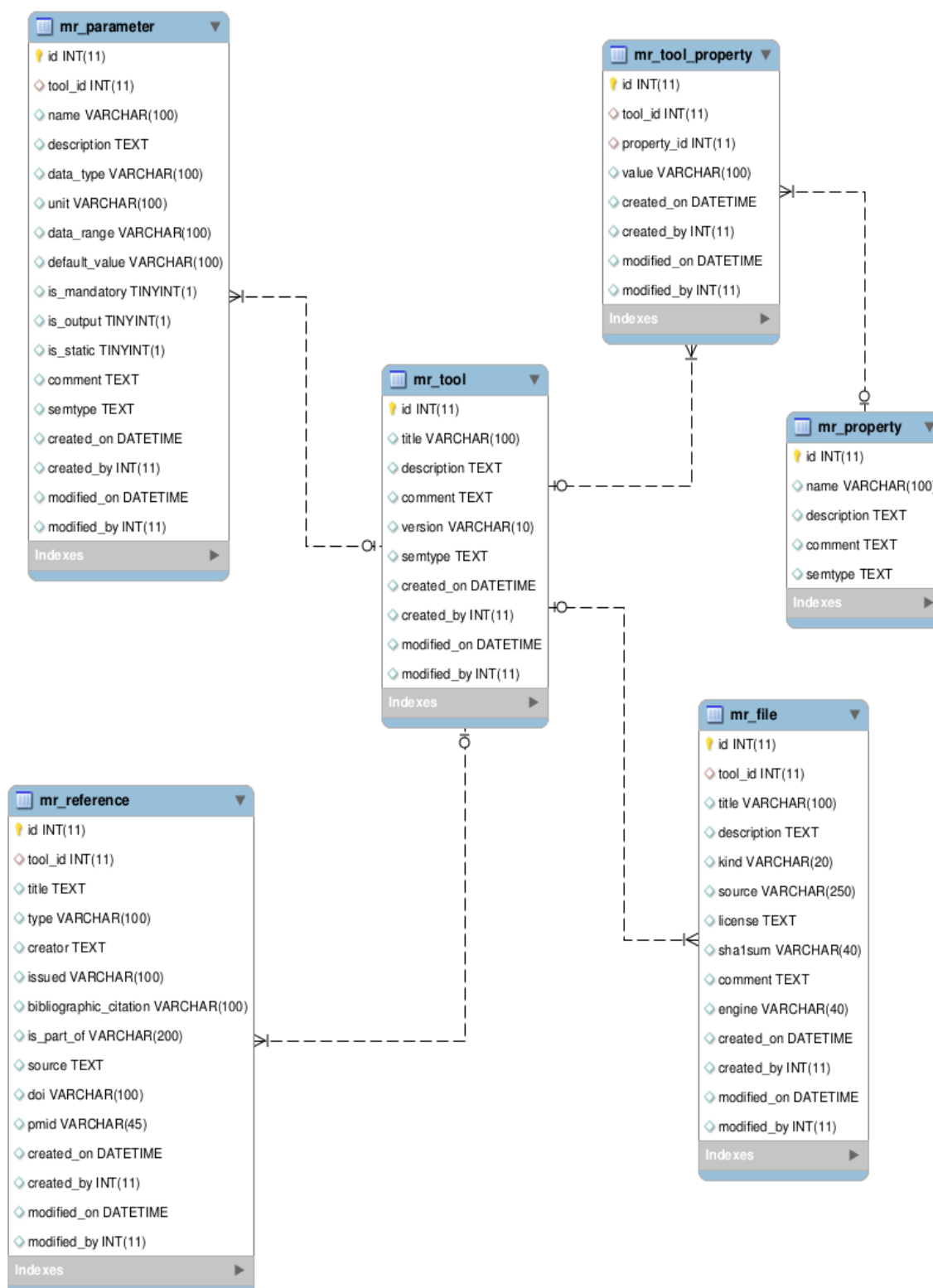


Figure 5: Updated entity relationship (ER) diagram of model/tool repository


5.2 Model/tool repository RESTful application programming interfaces

As described in chapter 5.3, model/tool repository makes use of RESTful web services which are based on the entity relationship diagram depicted in figure 5. Model/tool repository's RESTful web services are mainly based on and the interfaces described in deliverable "D5.3 – 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.

Model/Tool


The following web services (tables 7 - 12) should be used whenever the client needs to store, retrieve or delete descriptive information (title, description, comments) of the model/tool.

Table 7: 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://139.91.210.27/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
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 8: Information for calling getAllTools web service

getAllTools		
Description	This method returns all the models/tools and the corresponding descriptive information stored (title, description, comment, version, semtype). It returns null when no model/tool stored in the repository.	
URL	https://139.91.210.27/model_app/getAllTools	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
Parameters	No parameters required	
Returns	200 OK & JSON object	
	400 http status code if bad request	


	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getAllTools are as many as the different models/tools stored in the model/tool repository. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of mr_tool entity (see figure 5) and each value of this nested object represents the information of the corresponding column</p>		

Table 9: Information for calling getToolById web service

getToolById		
Description	This method returns the descriptive information stored under the id (title, description, comment, version, semtype) and null when not existing	
URL	https://139.91.210.27/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 eleven keys named title, description, comment, version, semtype, created_on, created_by, modified_on and modified_by, and eleven values associated with those keys.		

Table 10: Information for calling getLatestToolByToolName web service

getLatestToolByToolName		
Description	This method returns the information of the latest version of a given model/tool name	
URL	https://139.91.210.27/model_app/getLatestToolByToolName	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	title=	Required – the title of the model/tool
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64


		encoded compressed SAML token>
Json Response		
The JSON object returned by method <code>getLatestToolByToolName</code> has eleven keys named title, description, comment, version, semtype, created_on, created_by, modified_on and modified_by, and eleven values associated with those keys.		

Table 11: Information for calling `getPreviousVersions` web service

getPreviousVersions		
Description	This method returns information of all the previous versions of a given model/tool	
URL	https://139.91.210.27/model_app/getPreviousVersions	
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 given 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 keys of the JSON object returned by method <code>getPreviousVersions</code> are as many as the different		

previous versions of a given model/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_tool entity (see figure 5) and each value of the nested JSON object represents the information of the corresponding column.


Table 12: 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://139.91.210.27/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 13-18) should be used whenever the client needs to store, retrieve or delete information related to parameters (name, description, data_type, data_range, etc.).

Table 13: Information for calling storeParameter web service

storeParameter		
Description	This method stores the parameter information of a tool and returns the id	
URL	https://139.91.210.27/model_app/storeParameter	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	tool_id=	Required - id of the tool to which the parameter belongs
	name=	Required – name of the parameter
	description=	Not Required – description of the parameter
	data_type=	Required – the type of the parameter (number, string, file)
	unit=	Not Required – the units in which the parameter is represented (only applicable if the parameter is a number)
	data_range=	Required – Data range of the parameter <ul style="list-style-type: none"> Discrete values example: value1,value2,value3 Min value example: 3- Max value example: -10 <ul style="list-style-type: none"> Min max values example: 3-5
	default_value=	Required – the value that will be used if a parameter value is not provided to the tool


	is_mandatory=	Required – 1 if the parameter is mandatory, 0 if it is optional
	is_output=	Required – 1 if the parameter is output, 0 if it is input
	is_static=	Required – 1 if the parameter is static, 0 if it is dynamic
	comment=	Not Required – comments on the parameter
	semtype=	Not required – url representing semantic information about this parameter
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 14: Information for calling deleteParameter web service

deleteParameter		
Description	This method deletes a certain parameter	
URL	https://139.91.210.27/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 15: Information for calling getParametersByToolId web service

getParametersByToolId		
Description	This method returns the information of all the parameters of a given tool	
URL	https://139.91.210.27/model_app/getParametersByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	tool_id=	Required – the id of the tool to which the parameters belong
Returns	200 OK & JSON object	


	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getParametersByToolId</code> are as many as the different parameters belonging to the tool. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the <code>mr_parameter</code> entity (see figure 5) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 16: Information for calling `getMandatoryParametersByToolId` web service

getMandatoryParametersByToolId		
Description	This method returns the information of the mandatory parameters of a given tool	
URL	https://139.91.210.27/model_app/getMandatoryParametersByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	tool_id=	Required - the id of the tool to which the mandatory parameters belong
Returns	200 OK & JSON object	
	400 http status code if bad request	


	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getMandatoryParametersByToolId</code> 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 <code>mr_parameter</code> entity (see figure 5) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 17: Information for calling `getInputParametersByToolId` web service

<code>getInputParametersByToolId</code>		
Description	This method returns the information of the input parameters of a given tool	
URL	https://139.91.210.27/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		
<p>The keys of the JSON object returned by method <code>getInputParametersByToolId</code> 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 <code>mr_parameter</code> entity (see figure 5) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 18: Information for calling `getOutputParametersByToolId` web service


<code>getOutputParametersByToolId</code>		
Description	This method returns the information of the output parameters of a given tool	
URL	https://139.91.210.27/model_app/getOutputParametersByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	<code>tool_id=</code>	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		
<p>The keys of the JSON object returned by method <code>getOutputParametersByToolId</code> 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 <code>mr_parameter</code> entity (see figure 5) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Property


The following web services (tables 19-25) should be used whenever the client needs to store, retrieve or delete information related to properties (property name, property value, property description, property comments).

Table 19: Information for calling storeProperty web service

storeProperty		
Description	This method stores the basic descriptive information of a property and returns the id	
URL	https://139.91.210.27/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
	semtyp=	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 20: Information for calling getAllProperties web service

getAllProperties		
Description	This method returns all the properties and the corresponding descriptive information stored (id, name, description, comment, semtype)	
URL	https://139.91.210.27/model_app/getAllProperties	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS	No parameters required	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	


	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getAllProperties</code> are as many as the different properties 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 <code>mr_property</code> entity (see figure 5) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 21: Information for calling `getPropertyById` web service

<code>getPropertyById</code>		
Description	This method returns the descriptive information stored under the property id (name, description, comment)	
URL	https://139.91.210.27/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 22: 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://139.91.210.27/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 23: Information for calling deletePropertyValue web service


deletePropertyValue		
Description	This method deletes the property value for a certain tool	
URL	https://139.91.210.27/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 24: Information for calling getPropertyValuesByToolId web service


getPropertyValuesByToolId		
Description	This method retrieves all the property – value pairs for a given tool	
URL	https://139.91.210.27/model_app/getPropertyValuesByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	tool_id=	Required – the id of the tool with which the property – value pairs are associated
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getPropertyValuesByToolId are as many as the different properties 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.</p>		

Table 25: Information for calling deletePropertyById web service


deletePropertyById		
Description	This method deletes the property of the given id and the	

	corresponding values	
URL	https://139.91.210.27/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 26-29) should be used whenever the client needs to store, retrieve or delete information related to references (reference title, reference authors, reference type, etc.).

Table 26: 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://139.91.210.27/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 27: Information for calling deleteReferenceById web service


deleteReferenceById		
Description	This method deletes a specific reference	
URL	https://139.91.210.27/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 28: Information for calling getReferencesByToolId web service



getReferencesByToolId		
Description	This method returns all the references of a given tool	
URL	https://139.91.210.27/model_app/getReferencesByToolId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	tool_id=	Required – the id of the tool with which the references are associated
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method 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 5) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 29: Information for calling getAxes web service

getAxes		
Description	This method returns all the references based on the arguments. It makes use of the is_part_of attribute and given the option and the level (if the option is different than that of sibling) returns the desired references along with their information.	
URL	https://139.91.210.27/model_app/getAxes	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the reference
	option=	Required – it takes one of the following string values: <ul style="list-style-type: none"> • Ancestors • Descendants • Siblings
	level=	Not required – this parameter is integer and it is required if the option is different than that of Siblings
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 `getAxes` are as many as the different references which are siblings, ancestors or descendants with the given reference. Each value is 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 5) and each value of the nested JSON object represents the information of the corresponding column.

File


The following web services (tables 30-33) 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 30: Information for calling storeFile web service

storeFile		
Description	This method stores the file information and returns the id	
URL	https://139.91.210.27/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 31: Information for calling deleteFile web service

deleteFile		
Description	This method deletes a certain file	
URL	https://139.91.210.27/model_app/deleteFile	
Encoding	application/x-www-form-urlencoded	
HTTP Method	DELETE	
PARAMETER (parameter should be passed through the URL –	id=	Required – the id of the file


query string parameter)		
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 32: Information for calling getFileById web service

getFileById		
Description	This method returns the given file (attachment)	
URL	https://139.91.210.27/model_app/getFileById	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	id=	Required – the id of the file
Returns (Content-Type: application/force-download Content-Disposition: attachment)	200 OK & attachment	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	

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

Table 33: 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://139.91.210.27/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 5) and each value of the nested JSON object represents the information of the corresponding column.

5.3 *In silico* trial repository schema

The three main entities of the in-silico trial repository are the trial, the experiment, and the subject. The trials are the digital equivalent of a clinical one, with models in place of the hypotheses and scientific methods, and placebo models in place of a placebo drug. For cancer cases a free growth model could be used as a placebo model.

A trial consists of multiple experiments. The experiments are performed with the same model each time, which is defined in the referred trial. Each experiment scheduled or ran in the CHIC framework is stored in this repository in the form of triples. A triple consists of the status of the subject of the experiment before the experiment execution, the model that is stored in the trial table and the status of the subject after the experiment.

Each experiment is performed on a subject that is on a given state and creates a new subject with the new state. The external ids and URLs are referring to external repositories that could be used such as the CHIC clinical data system, a clinical trial management system (ObTiMA, OpenClinica), etc.

The entity relational diagram of *in silico* trial repository, which has been retrieved from deliverable “D8.1 - Design of the CHIC repositories” is depicted, in the sake of completeness, in figure 5.

As shown in figure 5, entity `tr_miscellaneous_parameter` has been added to *in silico* trial repository. This entity aims at storing the values of miscellaneous parameters of the hypermodel that has been used in the experiment. Miscellaneous parameters are parameters which are independent of the patient, but the execution of the (hyper)model is affected by them. Such parameters are the time of simulation, the dimension of geometric cells, the margin percent, etc.

Since miscellaneous parameters cannot be correlated with subjects or patients, it became necessary to link directly the value of those parameters with the specific experiment in which those miscellaneous parameters have been used. More specifically `tr_miscellaneous_parameter` entity consists of the following columns: (attributes):

- `id`: Primary key. Used to uniquely identify each table row.
- `experiment_id`: The id of the experiment to which this parameter has been used. Linked to the table `tr_experiment`.
- `hypomodel_parameter_id`: The id of hypomodel’s parameter stored in model/tool repository with which this miscellaneous parameter is associated. Linked to the table `mr_parameter` of model/tool repository.
- `hypermodel_parameter_id`: The id of hypermodel’s parameter stored in model/tool repository with which this miscellaneous parameter is associated. Linked to the table `mr_parameter` of model/tool repository.
- `value`: The value that has been assigned to this miscellaneous parameter during the experiment.
- `created_on`: The date and time that this record has been created.
- `created_by`: The id of the creator of this record.
- `modified_on`: The date and time that this record has been modified.
- `modified_by`: The id of the modifier of this record.

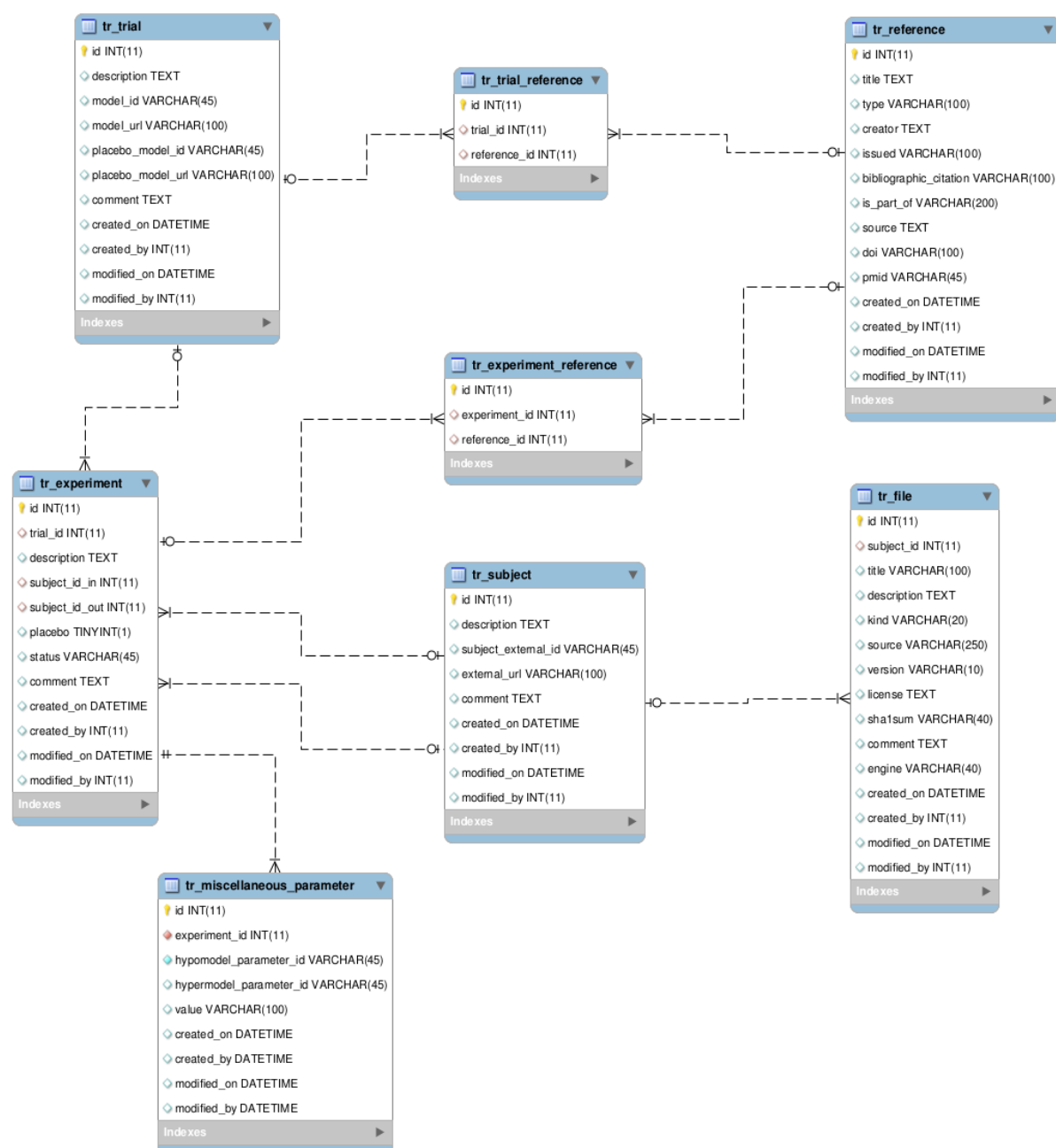


Figure 6: Updated entity relationship (ER) diagram of in silico trial repository


5.4 *In silico* trial repository RESTful application programming interfaces

As described in chapter *in silico* trial repository makes use of RESTful web services which are based on the entity relationship diagram depicted in figure 6. *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.

Trial


The following web services (tables 34-38) 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 34: 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://139.91.210.27/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 35: Information for calling getAllTrials web service

getAllTrials		
Description	This method returns the corresponding descriptive information of all the trials stored in <i>in silico</i> trial repository (trial ids, description of the trial, comments, etc.).	
URL	https://139.91.210.27/trial_app/getAllTrials	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS	No parameters required	
Returns	200 OK & JSON object	


	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getAllTrials are as many as the different trials stored in the <i>in silico</i> trial repository. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the tr_trial entity (see figure 6) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 36: 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://139.91.210.27/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 37: Information for calling getTrialById web service

getTrialById		
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://139.91.210.27/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 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 <code>getTrialById</code> has eleven keys named <code>id</code> , <code>description</code> , <code>model_id</code> , <code>model_url</code> , <code>placebo_model_id</code> , <code>placebo_model_url</code> , <code>comment</code> , <code>created_on</code> , <code>created_by</code> , <code>modified_on</code> and <code>modified_by</code> , and eleven values associated with those keys.		


Table 38: Information for calling `deleteTrialById` web service

deleteTrialById		
Description	This method deletes the trial, the experiments included in the trial and the reference links	
URL	https://139.91.210.27/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 39-45) 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 39: 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://139.91.210.27/trial_app/storeExperiment	
Encoding	application/x-www-form-urlencoded	
HTTP Method	POST	
PARAMETERS (parameters passed through request body)	trial_id=	Required – the id of the trial with which the new experiment is associated
	description=	Required – the description of the new experiment
	subject_id_in=	Required – the id of the subject that is used as an input to the new <i>in silico</i> experiment
	subject_id_out=	Required – the id of the subject that is produced after the execution of the new <i>in silico</i> experiment
	placebo=	Required – true if in the <i>in silico</i> experiment the placebo model must be used, otherwise false
	status=	Not required – the status of the <i>in silico</i> 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 JSON object returned by method storeExperiment has one key, named id, and one value which is associated with this key.		

Table 40: Information for calling getAllExperimentsByTrialId web service

getAllExperimentsByTrialId		
Description	This method returns information of all the experiments which belong to a given trial	
URL	https://139.91.210.27/trial_app/getAllExperimentsByTrialId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	trial_id=	Required – the id of the trial
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	


	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getAllExperimentsByTrialId</code> 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 <code>tr_experiment</code> entity (see figure 6) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 41: 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://139.91.210.27/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 <code>getExperimentById</code> has twelve keys named <code>id</code> , <code>trial_id</code> , <code>description</code> , <code>subject_id_in</code> , <code>subject_id_out</code> , <code>placebo</code> , <code>status</code> , <code>comment</code> , <code>created_on</code> , <code>created_by</code> , <code>modified_on</code> and <code>modified_by</code> , and twelve values associated with those keys.		


Table 42: Information for calling `getExperimentStatusById` web service

<code>getExperimentStatusById</code>		
Description	This method returns the status of the experiment	
URL	https://139.91.210.27/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 43: Information for calling `getExperimentsByStatus` web service

<code>getExperimentsByStatus</code>		
Description	This method returns all the experiments that are on a given status	
URL	https://139.91.210.27/trial_app/getExperimentsByStatus	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	<code>status=</code>	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 <code>getExperimentsByStatus</code> are as many as the different experiments that are on a given status. Each value associated with a specific key is		

represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the tr_experiment entity (see figure 6) and each value of the nested JSON object represents the information of the column.

Table 44: Information for calling updateExperimentStatus web service



updateExperimentStatus		
Description	This method updates the status of a given experiment	
URL	https://139.91.210.27/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 45: Information for calling deleteExperimentById web service

deleteExperimentById		
Description	This method deletes the experiment and the corresponding experiment references (links)	
URL	https://139.91.210.27/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 46-50) 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 46: 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://139.91.210.27/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 47: Information for calling getAllMiscellaneousParameters web service

getAllMiscellaneousParameters		
Description	This method returns information of all miscellaneous parameters	
URL	https://139.91.210.27/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 6) and each value of the nested JSON object represents the information of the corresponding column.

Table 48: Information for calling `getAllMiscellaneousParametersByExperimentId` web service

<code>getAllMiscellaneousParametersByExperimentId</code>		
Description	This method returns information of all miscellaneous parameters which are associated with a given experiment	
URL	https://139.91.210.27/trial_app/getAllMiscellaneousParametersByExperimentId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	experiment_id=	Required – the id of the experiment
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
The keys of the JSON object returned by method <code>getAllMiscellaneousParametersByExperimentId</code> 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 6) and each value of the nested JSON object represents the information of the corresponding column.

Table 49: 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://139.91.210.27/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 50: Information for calling deleteMiscellaneousParameterById web service

deleteMiscellaneousParameterById		
Description	This method deletes the miscellaneous parameter	
URL	https://139.91.210.27/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 51-54) 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 51: 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://139.91.210.27/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 52: 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://139.91.210.27/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 53: Information for calling getAllSubjects web service

getAllSubjects		
Description	This method returns all the subjects	
URL	https://139.91.210.27/trial_app/getAllSubjects	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	

PARAMETERS	No parameters required	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method <code>getAllSubjects</code> are as many as the different subjects that are stored in the <i>in silico</i> trial repository. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the <code>tr_subject</code> entity (see figure 6) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 54: Information for calling `getSubjectById` web service


<code>getSubjectById</code>		
Description	This method returns the subject and the related information stored under the id (description, subject_external_id, external_url, comments, etc.)	
URL	https://139.91.210.27/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 55-62) 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 55: Information for calling storeTrReference web service

storeTrReference		
Description	This method stores the information of a reference and returns the id	
URL	https://139.91.210.27/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

D8.3 – Implementation of the interfaces of the CHIC repositories

	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 56: Information for calling getAllTrReferences web service


getAllTrReferences		
Description	This method returns all the references and the related information	
URL	https://139.91.210.27/trial_app/getAllTrReferences	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS	No parameters required	
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getAllTrReferences are as many as the different references that are stored in the <i>in silico</i> trial repository. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the tr_reference entity (see figure 6) and each value of the nested JSON object represents the corresponding information of the column.</p>		

Table 57: 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://139.91.210.27/trial_app/getTrReferencesByTrialId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	trial_id=	Required – the id of the trial
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method 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 6) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 58: 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://139.91.210.27/trial_app/getTrReferencesByExperimentId	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETER (parameter should be passed through the URL – query string parameter)	experiment_id=	Required – the id of the experiment
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getTrReferencesByExperimentId are as many as the different references that are associated with the given experiment. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the tr_reference entity (see figure 6) and each value of the nested JSON object represents the information of the corresponding column.</p>		

Table 59: Information for calling deleteTrReferenceById web service




deleteTrReferenceById		
Description	This method deletes a reference and the corresponding links to trials or experiments	
URL	https://139.91.210.27/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 60: 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://139.91.210.27/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 61: 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://139.91.210.27/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>

Table 62: Information for calling getTrAxes web service


getTrAxes		
Description	This method returns all the references based on the arguments. It makes use of the is_part_of attribute and given the option and the level (if the option is different than that of sibling) returns the desired references with all the reference information	
URL	https://139.91.210.27/trial_app/getTrAxes	
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 reference
	option=	Required – it takes one of the following string values: <ul style="list-style-type: none"> • Ancestors • Descendants • Siblings

	level=	Not required – this parameter is integer and it is required if the option is different than that of Siblings
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getTrAxes are as many as the different references which are siblings, ancestors or descendants with the given reference. 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 6) and each value of the nested JSON object represents the information of the corresponding column.</p>		

File

The following web services (tables 63-68) 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 63: Information for calling storeTrFile web service

storeTrFile		
Description	This method stores the file information and returns the id	
URL	https://139.91.210.27/model_app/storeTrFile	
Encoding	Multipart/form-data	

HTTP Method	POST	
PARAMETERS (parameters passed through request body)	subject_id=	Required – the id of the subject with which the file is associated
	title=	Required – the title of the file
	description=	Not required – description of the file
	kind=	Not required – defines what this file is (document, spreadsheet, csv, etc.)
	version=	Required – the version of the file (should be in the format X.X for example 1.2)
	Sha1sum=	Not required – the sha1 checksum of the file
	comment=	Not required – comments on the file
	file=	Required – the actual file (blob)
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 64: Information for calling deleteTrFile web service




deleteTrFile		
Description	This method deletes a certain file	
URL	https://139.91.210.27/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 65: Information for calling getTrFileById web service

getTrFileById		
Description	This method returns the file (which is associated with a subject)	
URL	https://139.91.210.27/model_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 66: Information for calling getTrLatestFilesBySubjectId web service

getTrLatestFilesBySubjectId		
Description	This method returns information of all the latest version files of a given subject	
URL	https://139.91.210.27/trial_app/getTrLatestFilesBySubjectId	
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
Returns	200 OK & JSON object	
	400 http status code if bad request	


	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getTrLatestFilesBySubjectId are as many as the different latest version files 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 6) and each value of the nested JSON object represents the information of the column.</p>		

Table 67: Information for calling getTrFilesOfKind web service

getTrFilesOfKind		
Description	This method returns the information of the latest versions of all the files of a specific kind of a given subject	
URL	https://139.91.210.27/model_app/getTrFilesOfKind	
Encoding	application/x-www-form-urlencoded	
HTTP Method	GET	
PARAMETERS (parameters should be passed through the URL – query string parameter)	subject_id=	Required – the id of the subject
	kind=	Required - kind of file (document, spreadsheet, csv, etc.)
Returns	200 OK & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	

	403 http status code if SAML token not verified	
	500 http status code if internal server error	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getTrFilesOfKind are as many as the different latest version files of a specific kind which are associated with the given subject. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the tr_file entity (see figure 6) and each value of the nested JSON object represents the information of the column.</p>		

Table 68: Information for calling getTrPreviousVersions web service

getTrPreviousVersions		
Description	This method returns information of all the previous versions of a given file	
URL	https://139.91.210.27/model_app/getTrPreviousVersions	
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 & JSON object	
	400 http status code if bad request	
	401 http status code if no SAML token inside HTTP header	
	403 http status code if SAML token not verified	
	500 http status code if internal server error	

HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 encoded compressed SAML token>
Json Response		
<p>The keys of the JSON object returned by method getTrPreviousVersions are as many as the different previous version files of a given file. 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 6) and each value of the nested JSON object represents the information of the column.</p>		

6 RDF storage solution for semantic metadata

6.1 Introduction

Semantic metadata refers to data that has been specially formatted for greater machine readability, interoperability, and automated semantic reasoning. In the context of CHIC, models and their parts are given various standardised annotations, and these annotations are stored as metadata on a CHIC server. For the sake of interoperability and automated reasoning, these annotations make reference to a set of standard open-source reference ontologies created and maintained by the biomedical community at large. For example, if we have an image of a lung, rather than annotate it with a bare string ("lung"), which would not be semantically interoperable, we might instead annotate it with an ontology term (such as FMA_7195: the "lung" entry in the Foundational Model of Anatomy, a widely-used anatomy ontology). This makes it possible to use the metadata easily, without having to program ad hoc machinery on a use-case-by-use-case basis. This also reduces ambiguity and reduces dependence on a particular spoken language (such as English).

6.2 Tools

For storing, updating, maintaining, and querying metadata, CHIC makes use of the RICORDO software suite, a collection of APIs that facilitate the creation, storage, and reasoning over of metadata.

Individual annotations are stored in the form of RDF triples. RDF is the Resource Description Framework, a longstanding W3C recommendation that forms the backbone of the semantic web. An RDF triple consists of a subject IRI, a predicate IRI, and an object IRI (an *IRI* is an *Internationalised Resource Identifier*, a unique, language-agnostic name used to refer to an object in the semantic web). For a concrete example, imagine a triple whose subject term is (an IRI for a formal name of) a particular photograph; whose object is (an ontology term for) "lung"; and whose predicate is (an IRI for a formal name of) "image-of". This triple means that the image in question is an image of a lung. Crucially, by storing the fact in this form, the fact becomes amenable to automated reasoning, in an interoperable way. The fact that the image is an image of a lung can be understood and used generically by any kind of software with basic W3C-compliant semantic reasoning capabilities, without any sort of special ad-hoc programming about photographs etc.

Triples are stored in a so-called "triple-store". A triple-store is a database specially designed to contain linked metadata in triples form. Currently, the most appropriate triple store is Virtuoso, an open-source triple store with widely demonstrated performance and scalability. (Another popular triple store is Fuseki, which has less of a learning curve, but CHIC has chosen Virtuoso for its higher performance, in spite of Virtuoso's somewhat greater learning curve.) RICORDO provides (via its RDFSTORE program) a templating API intended to facilitate querying a triple store with user-friendly forms (see figures). A template curator can create a general template one time, to facilitate submission of such queries an infinite number of times. For example, if end-users express a wish to perform searches such as "find all resources related to drug X" or "find all resources related to drug Y", a template curator can create a general template for "find all resources related to drug ____" (the last word to be filled in dynamically). Once this template has been added to the CHIC instance of the RDFSTORE templating system, queries for resources related to arbitrary drugs are available

through a simple URL-based HTTP API. The intention is not that end-users interact with the RDFSTORE templating system directly, but rather that other partners in the CHIC process use the system to integrate metadata into the various other components of CHIC.

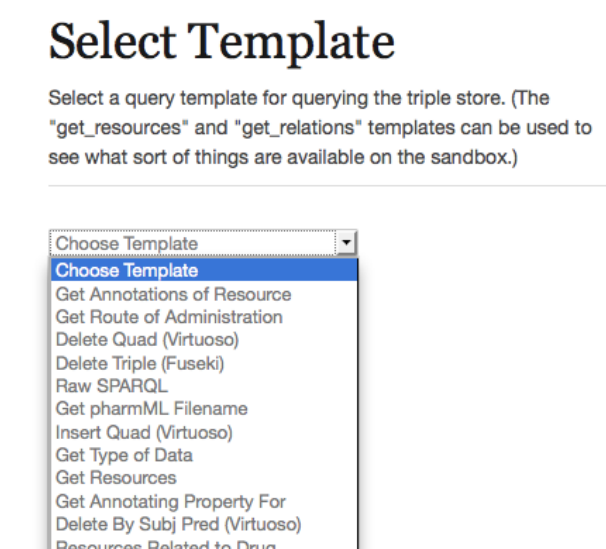


Figure 7: Prototype of RICORDO Template GUI

To create annotations using ontology terms, it is necessary to search an ontology and find which term to use. For example, if we want to annotate a photograph as being an image of a lung, we must first determine what is the correct, interoperable and semantically meaningful way to refer to lung (as opposed to writing the plain-text word “lung” which is not interoperable or semantically meaningful to a machine). This means querying an anatomy ontology for “lung”. RICORDO provides tools for doing this. RICORDO's Local Ontology Lookup Service (see figures) provides an API for quickly searching for ontology terms. The Local Ontology Lookup Service was designed for light-weight deployability, making it easy to deploy on the same machines where other CHIC infrastructure is run, freeing CHIC from being dependent on remote, typically very slow and unreliable, outside parties. The Local Ontology Lookup Service also includes an autocomplete API.

Control Panel

What kind of lookup do you want to perform?

Find full IRI from label (case sensitive)
Find full IRI from label (case sensitive)
Find short IRI from label (case sensitive)
Find full IRI from label (case insensitive)
Find short IRI from label (case insensitive)
Find label from IRI
Autocomplete (case sensitive)
Autocomplete (case insensitive)

Lookup

Enter label to look up
Enter

Results

Results will be displayed here.

Control Panel

What kind of lookup do you want to perform?

Find short IRI from label (case insensitive)

☒ Formatted Results
☐ Raw Results

Lookup

lung|
Enter

Results

FMA_7195

Figure 8: Selection of templates and search function

RICORDO further provides, through its OWLKB program, an API for automated semantic reasoning, and can generate so-called "composite terms", semantically meaningful combinations of existing ontology terms. For example, suppose we wish to annotate a procedure with a date of occurrence, and suppose we have reference ontologies with terms for "date" and for "occurrence", but not for "date of occurrence". OWLKB can be used to generate a composite term for "date of occurrence" from the constituent terms "date" and "occurrence", and most importantly, the composite term is semantically meaningful: automated reasoners can understand what it means.

Term

» (Unlabeled class)
http://www.ricordo.eu/ricordo.owl#RICORDO_1418055983401

Subterms

» (Unlabeled class)
http://www.ricordo.eu/ricordo.owl#RICORDO_1426802524429

» (Nothing -- the empty class)
<http://www.w3.org/2002/07/owl#Nothing>

» Forebrain
http://purl.org/obo/owlapi/fma#FMA_61992

» Midbrain
http://purl.org/obo/owlapi/fma#FMA_61993

Figure 9: Interaction with OWLKB

Official documentation for the different components of the RICORDO toolset are available at the following addresses:

RDFStore: <http://open-physiology.org/rdfstore/doc.html>

Local Ontology Lookup Service: <http://open-physiology.org/LOLS/doc.html>

OWLKB: <http://open-physiology.org/owlkb/doc.html>

The RICORDO toolset and the Virtuoso triplestore have been deployed to a VPS (Virtual Private Server) hosted by CHIC. They provide service to other CHIC components via APIs over the HTTP protocol.

6.2.1 Input and output of RICORDO tools

RICORDO tools accept input through an API over the HTTP protocol, and send their responses as HTTP responses. For specific details about the API, and specific format of the input to RICORDO, see RICORDO's documentation.

Roughly speaking, RICORDO sends its output in JSON format. JSON stands for JavaScript Object Notation and it is the most appropriate format for use by web-based applications. We say, "roughly speaking", because there are a couple of caveats.

1. For legacy reasons, RICORDO's OWLKB by default sends its responses in an HTML format; in order to coax OWLKB into sending responses in JSON it is necessary to send a specific header one's API request, specifically the header "Accept: application/json".
2. RICORDO's RDFStore acts as middleman between CHIC application and triple-store: in particular, its output is simply the triplestore output. Most triplestores, including Virtuoso, offer JSON output; Rdfstore's documentation explains how to configure RDFStore to request JSON output from Virtuoso.

So in summary: RICORDO's tools accept input over an API; documentation has been provided for that input; and, with a couple minor caveats, RICORDO's tools send their output in JSON format.

6.3 The CHIC semantic metadata lifecycle

Semantic metadata in CHIC has a lifecycle consisting of three stages.

6.3.1 Creation

Some annotations will be created manually, but most will be automatically generated by various CHIC components. In order to facilitate this, CHIC's developers need APIs for querying the background reference ontologies in order to find which terms to use; RICORDO provides such APIs.

6.3.2 Insertion into triple store

Once generated, an annotation will be entered into the triplestore for storage and subsequent querying. Individual triples can be inserted one-by-one via SPARQL (or via more user-friendly forms generated from triples). But most triples will probably be bulk-loaded: multiple (possibly very many) automatically generated triples are first written to a file, in RDF format, and the file is then bulk-loaded into the Virtuoso triple store using its bulk-loading features. This allows Virtuoso to take advantage of multiple CPU cores, as well as to loading algorithms designed for bulk-loading, in order to add large amounts of triples to the store very quickly.

6.3.3 Querying

Once stored in the triplestore, triples are accessed by querying. The triplestore can be queried directly, using the SPARQL query language. This, however, will be invisible to the end-user: such queries will mostly be done programmatically "under the hood" by other programs in CHIC. (As an alternative to SPARQL, such other programs can also use RICORDO's RDFSTORE template system; this decision is up to the developers of the other components of CHIC.) In particular, our intention is that the ultimate end-user be able to

query and use the metadata via friendly graphical front-ends requiring no knowledge of the underlying technology.

6.4 *What can be annotated?*

Flexibility is one of the key virtues of the RDF data format. Rather than referring to objects with machine-specific designators (filenames, memory locations, database indices, etc.), objects are referred to by IRI (Internationalized Resource Identifiers, another W3C standard), making RDF agnostic about technical implementation details. In short, anything that can be given a stable IRI, can be annotated.

7 Conclusions

The above technical description provides the basis for the development and maintenance of information flow between the core repositories of resources in CHIC. Given that the technical specification of these interfaces is in itself in a state of evolution, further detailed documentation (and, indeed, updates to this technical description) is provided via the up-to-date web links associated with the respective sections. A practical description of the application of these interfaces in the data and knowledge management of the CHIC hypermodelling infrastructure will be provided in the deliverable D7.3, on M36.

8 References

- 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>.
- 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>.
- 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.
- C. Rosse, and J. Mejino. A reference ontology for biomedical informatics: the Foundational Model of Anatomy. J. Biomed. Inform. 36:478–500, 2003.
- SPARQL, “Query Language for Resource Description Framework (RDF)”, [Online]. Available: <http://www.w3.org/TR/rdf-sparql-query>.
- <https://pypi.python.org/pypi/dm.xmlsec.binding/1.3.2>
- L. Richardson and S. Ruby, Restful Web Services, 1st ed. O’Reilly Media, May 2007.
- H. Lockhart et al, “Security Assertion Markup Language (SAML) V2.0 Technical Overview”, <http://www.oasis-open.org/committees/download.php/14361/sstc-saml-tech-overview-2.0-draft-08.pdf>
- open-physiology.org website
- RDFStore 2.0 Documentation: <http://open-physiology.org/rdfstore/doc.html>
- OWLKB 2.0 Documentation: <http://open-physiology.org/owlkb/doc.html>
- LOLS Documentation: <http://open-physiology.org/LOLS/doc.html>

Appendix 1 – Abbreviations and acronyms

<i>SOA</i>	Service Oriented Architecture
<i>REST</i>	Representational State Transfer
<i>API</i>	Application Programming Interface
<i>HTTP</i>	Hypertext Transfer Protocol
<i>URI</i>	Universal Resource Identifier
<i>STS</i>	Security Token Service
<i>SAML</i>	Security Assertion Markup Language
<i>SOAP</i>	Simple Object Access Protocol
<i>RST</i>	Request Security Token
<i>RSTR</i>	Request Security Token Response
<i>SP</i>	Service Provider
<i>IdP</i>	Identity Provider
<i>SP</i>	Service Provider
<i>SSO</i>	Single Sign-On
<i>RFC</i>	Request for Comments
<i>JSON</i>	JavaScript Object Notation
<i>XML</i>	Extensible Markup Language
<i>UTF</i>	Unicode Transformation Format
<i>URL</i>	Uniform Resource Locator
<i>TTP</i>	Trusted Third Party
<i>FMA</i>	Foundational Model of Anatomy
<i>RDF</i>	Resource Description Framework
<i>W3C</i>	World Wide Web Consortium
<i>IRI</i>	Internationalised Resource Identifier
<i>OWLKB</i>	Web Ontology Language KnowledgeBase
<i>LOLS</i>	Local Ontology Lookup Service
<i>VPS</i>	Virtual Private Server
<i>SPARQL</i>	SPARQL Protocol and RDF Query Language
<i>CPU</i>	Central Processing Unit

