



Deliverable No. 5.1.1
The CHIC technical architecture
– initial version

Grant Agreement No.:	600841
Deliverable No.:	D5.1.1
Deliverable Name:	The CHIC technical architecture –initial version
Contractual Submission Date:	31/03/2014
Actual Submission Date:	12/06/2014



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.:	D5.1.1
Document name:	The CHIC technical architecture
Nature (R, P, D, O) ¹	R
Dissemination Level (PU, PP, RE, CO) ²	PU
Version:	1.0
Actual Submission Date:	12/06/2014
Editor:	Giorgos Zacharioudakis
Institution:	FORTH
E-Mail:	gzaxar@ics.forth.gr

ABSTRACT:

This deliverable documents the initial version of the CHIC technical architecture. Based on the IEEE 1471 standard “Recommended practice for Architecture Description of Software-Intensive Systems” we describe the most important views of the architecture. In the Functional view we describe the functionality of the CHIC platform and its functional components, in the Information View we describe the type of information handled and in the Deployment view we present an initial status of the technical details behind the implementation of this architecture.

KEYWORD LIST:

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

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

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

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

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

MODIFICATION CONTROL			
Version	Date	Status	Author
0.1	10/04/2014	Draft	Giorgos Zacharioudakis
0.2	25/04/2014	Draft	Giorgos Zacharioudakis
0.3	05/05/2014	Draft	Giorgos Zacharioudakis
0.4	23/05/2014	Draft	Giorgos Zacharioudakis
0.5	30/05/2014	Draft	Giorgos Zacharioudakis
0.6	04/06/2014	Pre-final draft	Giorgos Zacharioudakis
1.0	12/06/2014	Final version submitted to EC	Giorgos Zacharioudakis

List of contributors

- Georgios Stamatakos, ICCS
- Fay Misichroni, ICCS
- Giorgos Zacharioudakis, FORTH
- Stelios Sfakianakis, FORTH
- Manolis Tsiknakis, TEI-C & FORTH
- Elias Neri, Custodix
- Marco Viceconti, USFD
- Debora Testi, CINECA
- Philippe Buechler, UBERN
- Roman Niklaus, UBERN
- Bernard de Bono, UCL
- Norbert Graf, USAAR
- Feng Dong, BED
- Anca Bucur, PHILIPS
- Georgios Aravanis, PHILIPS

Contents

1	EXECUTIVE SUMMARY.....	7
2	INTRODUCTION	8
3	ARCHITECTURAL APPROACH	10
3.1	ARCHITECTURAL DESCRIPTION - THE IEEE 1471 STANDARD.....	10
3.2	VIEWS OF THE ARCHITECTURE.....	11
3.3	VIEWS OF THE CHIC ARCHITECTURE	13
4	ARCHITECTURAL DRIVERS IN CHIC	14
4.1	STAKEHOLDERS	14
4.2	GOALS, CONSTRAINTS, PRINCIPLES, CONCERNS OF THE CHIC STAKEHOLDERS.....	15
5	FUNCTIONAL VIEW	18
5.1	INTRODUCTION - HIGH LEVEL VIEW	18
5.2	USER INTERFACE LAYER.....	20
5.3	BUSINESS LOGIC LAYER	21
5.4	DATA INTEGRATION LAYER.....	27
5.5	SECURITY LAYER	36
5.6	SEMANTICS LAYER.....	42
5.7	INFRASTRUCTURE LAYER.....	45
6	INFORMATION VIEW	47
6.1	INTRODUCTION – HIGH LEVEL VIEW.....	47
6.2	DATA FLOW AND INTERACTION OF THE REPOSITORIES WITH THE FUNCTIONAL COMPONENTS.....	48
6.3	CLINICAL DATA	49
6.4	USER REPOSITORY.....	51
7	DEPLOYMENT VIEW.....	54
7.1	INTRODUCTION – HIGH LEVEL VIEW	54
7.2	USER INTERFACE LAYER.....	54
7.3	BUSINESS LOGIC LAYER	55
7.4	DATA INTEGRATION LAYER.....	58
7.5	SECURITY LAYER	61
7.6	SEMANTICS LAYER.....	67
7.7	INFRASTRUCTURE LAYER.....	68
8	SECURITY VIEW.....	70
8.1	BROKERED AUTHENTICATION.....	70
8.2	SECURITY COMPONENTS	78
8.3	SECURITY VOCABULARY	80
8.4	SECURITY FRAMEWORK GUIDELINES.....	80
9	REFERENCES	84
	APPENDIX 1 – ABBREVIATIONS AND ACRONYMS	86

1 Executive Summary

A software architecture specification provides the necessary information to understand, build and deploy a complex software system. In this document we specify the initial CHIC technical architecture, following a systematic approach, as documented in the IEEE 1471 - ISO/IEC 42010:2007 standard and we select a set of views to describe the architecture, the Functional, the Information, the Deployment and the Security view. These views provide the necessary information to understand what the system does, its main functional components, their programming interfaces, what information is stored or exchanged in the system, the technical details of its execution environment and how all these modules are connected through a common security framework. This is the initial version of the CHIC architecture, which will be refined and updated on an iterative approach.

2 Introduction

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

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

The tools and facilities described above comprise a complex software system, which calls for a systematic approach to design, implement and document its architecture, in order to achieve its goals. In this document, we give an overview of a standardized approach to describe a software architecture, based on the ISO/IEC 42010:2007 standard. The key concept in this standard is that although a complex system can be seen and described from a high level view, in order to understand its internal workings, its specificities and its details we have to introduce views that are closer to its surface, by identifying viewpoints that relate to the concerns of the stakeholders involved and describe the architecture based on specific views. Following this approach, we select the key set of views to describe the CHIC architecture: the Functional view, the Information view, the Deployment view and the Security view.

The rest of this document is organized as following: In chapter 3 we present our architectural approach. In chapter 4 we present the architectural drivers behind the CHIC platform, its stakeholders, their concerns and the existing restrictions. In chapter 5 we elaborate on the Functional view of the CHIC architecture and its functional components. In chapter 6 we describe the Information view and the way information is stored or exchanged inside this system. In chapter 7 we present the Deployment view of the architecture, the technical details and the execution environment of the platform. Finally in chapter 8 we present the Security view of the architecture and how the platform will meet the security concerns of the CHIC stakeholders.

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

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

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

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

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

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

3.2 Views of the architecture

For the selection of specific views and viewpoints of the architecture, there are a number of different architectural frameworks supporting and proposing different views and viewpoints, such as the 4+1 views model [2], the Reference Model of Open Distributed Processing (RM-ODP) [3], the Zachman framework [4], the Department of Defense Architecture Framework (DoDAF) [5], etc.

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

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

Rozanski and Woods [6] prescribed a useful set of viewpoints to be used in documenting software architectures. They have essentially extended the 4+1 model by providing the **Information** viewpoint to deal with data related concerns, like structure, ownership, distribution, etc. and the **Operational** viewpoint in order to describe how the system is installed, monitored etc. Their six viewpoints are the following:

- The **functional** view documents the system’s functional elements, their responsibilities, interfaces, and primary interactions. A functional view is the cornerstone of most architecture documents and is often the first part of the documentation that stakeholders try to read. It drives the shape of other system structures such as the information structure, concurrency structure, deployment structure, and so on. It also has a significant impact on the system’s quality properties, such as its ability to change, its ability to be secured, and its runtime performance.
- The **information** view documents the way that the architecture stores, manipulates, manages, and distributes information. The ultimate purpose of virtually any computer system is to manipulate information in some form, and this viewpoint develops a complete but broad view of static data structure and information flow. The objective of this analysis is to answer the important questions around content, structure, ownership, latency, references, and data migration.
- The **concurrency** view describes the concurrency structure of the system and maps functional elements to concurrency units to clearly identify the parts of the system that can execute concurrently and how this is coordinated and controlled. This entails the creation of models that show the process and thread structures that the system will use and the inter-process communication mechanisms used to coordinate their operation.
- The **development** view describes the architecture that supports the software development process. Development views communicate the aspects of the architecture of interest to those stakeholders involved in building, testing, maintaining, and enhancing the system.

- The **deployment** view describes the environment into which the system will be deployed, including capturing the dependencies the system has on its runtime environment. This view captures the hardware environment that the system needs, the technical environment requirements for each element, and the mapping of the software elements to the runtime environment that will execute them.
- The **operational** view describes how the system will be operated, administered, and supported when it is running in its production environment. For all but the simplest systems, installing, managing, and operating the system is a significant task that must be considered and planned at design time. The aim of the operational view is to identify system-wide strategies for addressing the operational concerns of the system's stakeholders and to identify solutions that address these.

3.3 Views of the CHIC architecture

Based on the architectural drivers of the CHIC project, the stakeholders, the requirements, the constraints and the general principles that are given in detail in chapter 4, we have selected some of the views of the Rozanski and Woods model for the architecture description and we concluded in the following set of views:

- The **Functional view**, which is essential to describe the functionality and the functional elements of the architecture
- The **Information view**, which is necessary to describe the type of information, its flow and its management in the system
- The **Deployment view**, which is very important due to the highly technical nature of the project

In addition, due to the highly importance of the security aspects in the CHIC project, we have added another view which aims to describe specifically the security issues of the architecture:

- The **Security view**, which is critical due to the nature of the handled data and the stakeholder's principles and constraints, describes the security framework employed in CHIC.

In later updates of the architecture, should the need arise we may add information also regarding other views of the architecture, such as the Operational view, but in the current state of the architecture neither we see such a need nor do we have available the corresponding information.

In the following sections we describe in more detail the architectural drivers on which our architecture is based, as well as the architecture views and their detailed description.

4 Architectural drivers in CHIC

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

4.1 Stakeholders

The general groups of stakeholders in the CHIC project are the CHIC consortium partners, the patients, the external users and the European Commission.

The consortium partners can be further categorized into distinct stakeholders; the clinical partners, such as clinicians, clinical researchers and data providers, and the technical partners such as the modellers, researchers (in various domains, especially in the clinical, biological or bioinformatics domain), the software developers and the administrators of the final platform.

While each stakeholder might have different concerns or requirements over the CHIC platform, some of them might share in parallel more than one role. For example a clinical partner might both be a clinical data provider and a clinical researcher; a technical partner may both be a model provider and a software developer. For this reason, for the sake of clarity and better understanding of each stakeholder category, the requirements and the functionality they need, the categorization given below is based on the distinct *virtual* roles participating in the project, than the actual end users or stakeholders. Due to simplicity of the development neither do we enforce any strict separation of the roles in the functionality of the system at this stage of the architecture. The separation of functionality, or access control, is achieved in the implementation level of the architecture by introducing relevant mechanisms and policies (described in more detail in chapter 5.5 and chapter 8).

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

Stakeholder	Remarks
Data providers	The data providers are the end users who will be providing data into the CHIC platform. The main tasks of those users will be to de-identify, upload and annotate data for usage by other users or components of the platform.
Model providers	The model providers upload, annotate and execute already available models. They also develop and execute hypermodels (integrative models) using the CHIC platform.
Researchers	This is a general user group which contains clinicians, bioinformaticians,

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

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

4.2.1 Data providers

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

4.2.2 Model providers

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

property rights (IPR) management. An important goal also for CHIC is to ensure the technical compatibility between the technologies that the modellers use and the ones supported by CHIC.

4.2.3 Researchers

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

4.2.4 Clinicians

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

4.2.5 Legal partners

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

4.2.6 Software developers

The software developers, the IT partners of the consortium have as primary goal to deliver the CHIC platform with all the necessary functionality, with the time and budget constraints that are imposed by their contractual obligations with the rest of the consortium partners and the EC. Their primary concerns are to overcome the IT engineering difficulties and deliver a state of the art infrastructure.

4.2.7 Administrators

The administrators can be considered as a subset of the IT partners whose main task is to run the CHIC infrastructure when it enters a production phase. The primary concern of those users is having

an infrastructure that works robustly and the availability of tools that will make their job easier for the management of the whole platform.

4.2.8 Patients

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

4.2.9 External users

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

4.2.10 European Commission

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

5 Functional view

5.1 Introduction - High level view

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

In order to describe the functional view of the system, we need a list of its functional elements, their responsibilities, their interactions and their interfaces, either user interfaces or programmatic interfaces. An initial description of the CHIC functionality from the end users point of view has also been roughly described in the deliverable D10.1 “The CHIC Portal” since the user portal is the functional element which aggregates much of the functionality targeted to end users.

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

- On top is the **user interface layer** which includes the user portal, the main point of interaction of the end users with the rest of the platform, and some standalone applications. The portal can further be decomposed into distinct functional elements, the portlets, which act as the UI of corresponding functional components of the platform.
- Below is located the **business logic layer**, which includes the hypermodelling infrastructure and the visualization and imaging toolkit.
- The **data integration layer** sits below, and includes the logical repositories of the system. The functionality of these components is described in this section but a more detailed description of the information they store can be found in the Information View, in section 6.
- The **infrastructure layer** is located on the bottom, providing support in the other functional components in terms of computational and storage needs.
- Two vertical layers that are crossing all the other layers described above, the **semantics layer** and the **security layer**.
 - The semantics layer provides the functionality of a logical integration element of the system, by linking data and functional components via a common, semantic ‘glue’.
 - The security layer serves a similar goal, yet on another aspect, by linking data and functional components with a security framework which guarantees the legal and ethical concerns of the stakeholders.

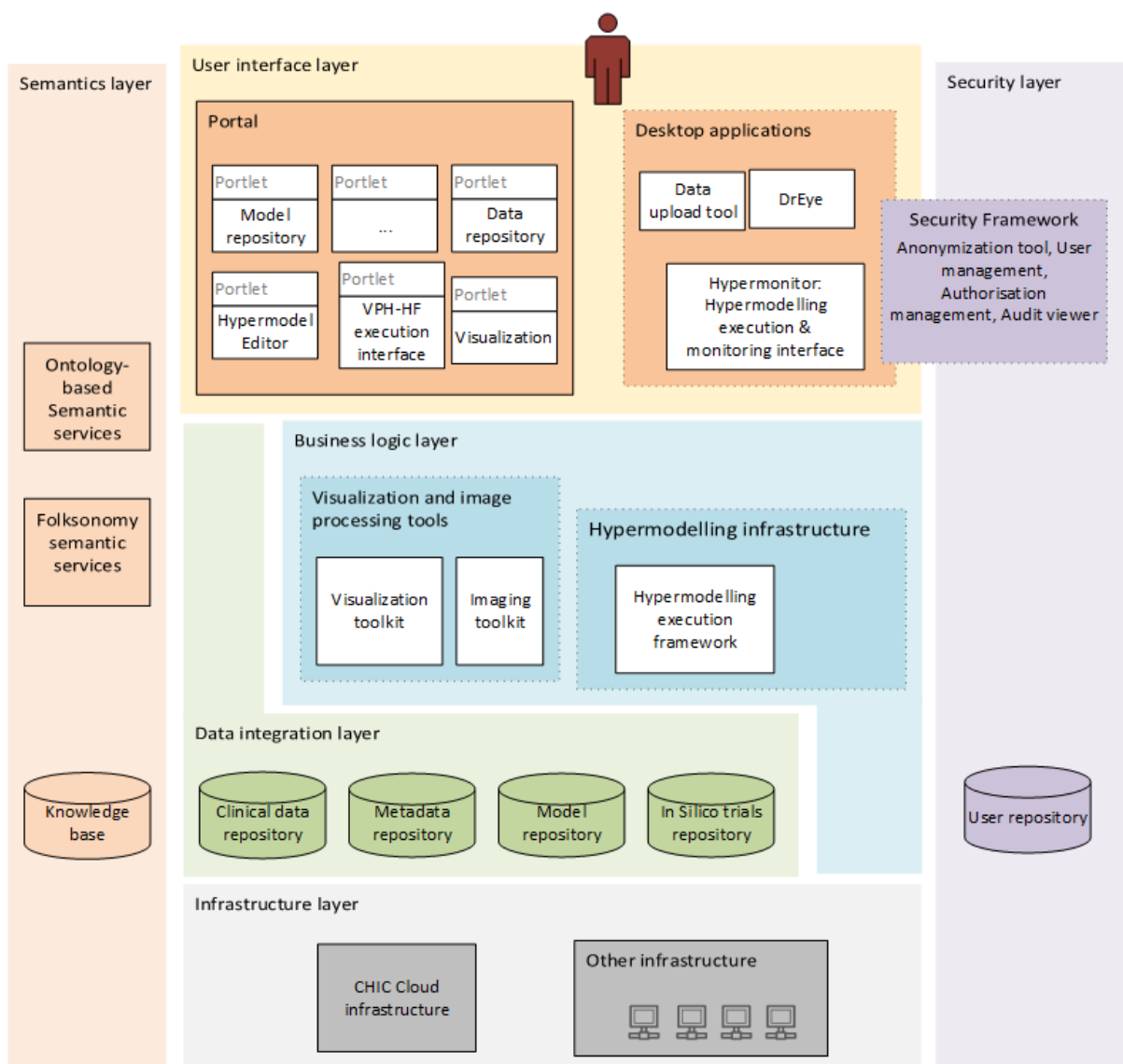


Figure 2 Functional view: A high level view of the CHIC functional components

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

ID	Functional groups	WP(-s)
1	Integrated platform (Portal)	WP10
2	Image processing and visualisation toolkits	WP9
3	Hypermodelling infrastructure	WP7

4	Data management system (data and model repositories)	WP8
5	Cloud infrastructure (storage and computation)	WP5, WP10
6	Semantics framework (annotation of data and models)	WP7, WP8
7	Security framework	WP5

As we mentioned above, in order to describe the functional view of the system we need a list of its functional elements and the functionality they provide, their interactions with other components and their interfaces, either user interfaces (UI) or programmatic interfaces (API). Below we have a list of all the functional components and the information that is currently available for each of them, up to this initial version of the CHIC architecture. A more detailed description of each functional component is given in the sections below.

5.2 User Interface layer

5.2.1 User Portal

5.2.1.1 Description

The user portal is the main user interface of the CHIC platform, which aggregates much of the functionality of all the underlying components to be usable by the end users. It consists of smaller implementation components, the portlets, which correspond to other functional components on the back end of the platform, such as the storage repositories and the hypermodelling execution framework.

The portal itself provides general purpose functionality, such as access control in the platform through the functionality of the security framework, tool management and collaboration tools, as well as integration of the CHIC specific functionality via its CHIC components. A preliminary description of the CHIC portal has also been documented in D10.1.

5.2.1.2 Functionality

ID	Functionality
1	Unified design and “look ‘n feel” of the user interface.
2	Integration of the user interfaces of various tools in a single, integrated web site.
3	Integrated user interface for access control (e.g. single sign on) to the CHIC platform, based on the underlying access control of the security framework tools.
4	User collaboration tools (calendar, user forum)
5	Tools (portlets) management.

	(Registration, publishing and execution of portlets)
--	------------------------------------------------------

5.2.1.1 Programmatic/User Interface

The Portal is the primary user-interfacing component of the CHIC platform. It features a web based graphical user interface with no programmatic interface. Screenshots or generic designs of the portal are not available yet in this initial version of the architecture, as it will very much be based and depended on the implementation of other components (portlets).

5.3 Business Logic layer

5.3.1 Hypermodelling editor

5.3.1.1 Description

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

5.3.1.2 Functionality

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

5.3.1.1 Programmatic/User Interface

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

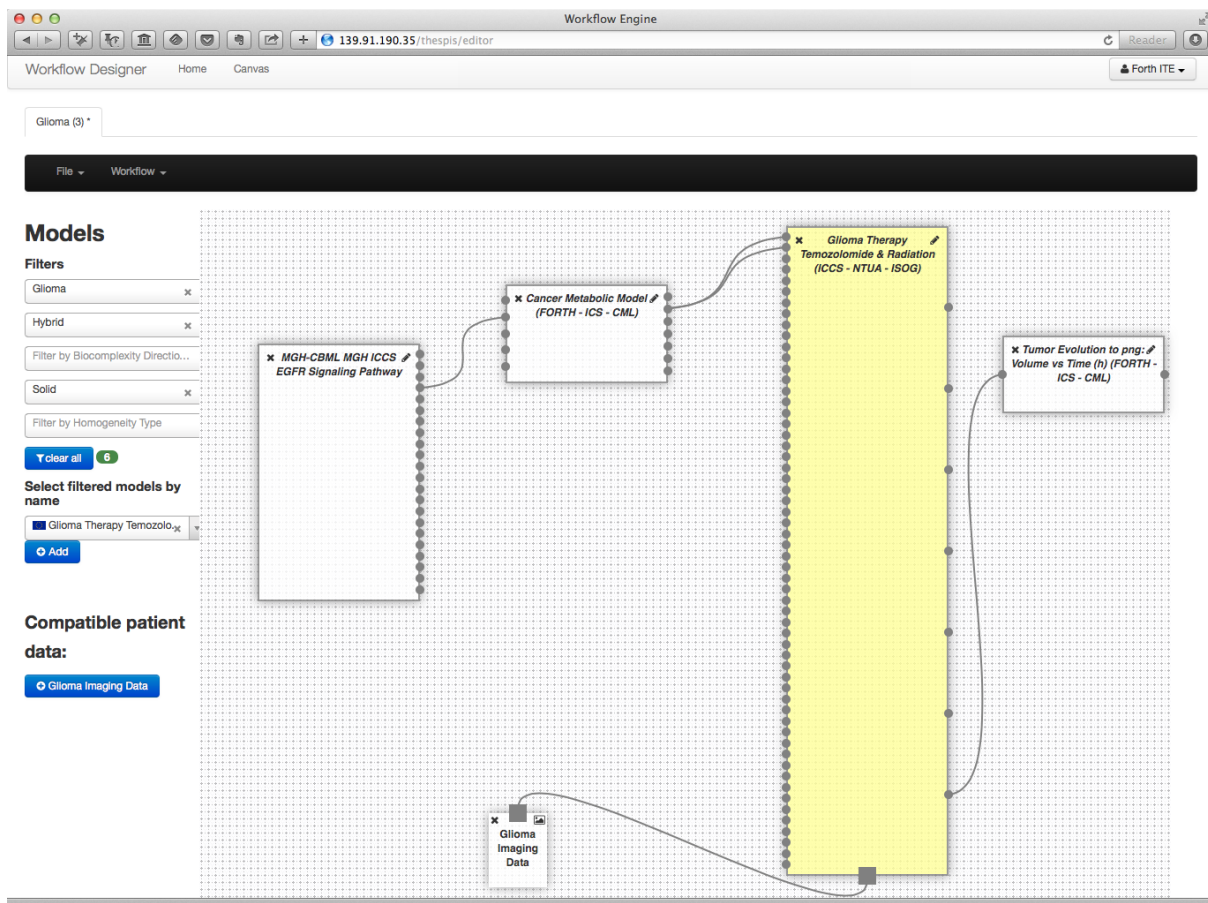


Figure 3 User interface of the hypermodelling editor.

5.3.2 Hypermonitor: hypermodels execution and monitoring

5.3.2.1 Description

Once a hypermodel has been defined, a user can execute it with given input data, and monitor its execution, using the hypermonitor component.

5.3.2.2 Functionality

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

1	Provide real-time status of all hypomodel implementations available in the CHIC cloud that the hypermodel to be executed requires; if multiple instances are available, the user can choose the one to be used.
2	Provide an execution interface, where the user selects the input data and submits the hypermodel for execution.
3	A global log monitor, where the execution of the hypermodel and of all hypomodels that compose it is reported in detail.
4	A command-line interface for the VPH-HF, where specific commands can be issued, primarily for debugging purposes.

5.3.2.1 Programmatic/User Interface

The first version of the Hypermonitor will be a MAF3 C++ multi-platform implementation with QT-based user-interface. The module must connect to the repositories and to the hypermodelling infrastructure (essentially the VPH Hypermodelling Framework, or VPH-HF).

It should also be possible from the Hypermodelling Editor to invoke the Hypermonitor with the hypermodel to be run passed as a parameter, and to connect to the VPH Hypermodelling Framework in order to run immediately a newly edited hypermodel.

The Hypermonitor is an end-user module and thus does not expose an API. It can be integrated into the portal with local execution on the client; it can link to the repositories via web services, given appropriate WSDL interfaces; and it connects to the hypermodelling infrastructure via a MAF3 proprietary communication layer (currently implemented in XML-RPC for efficiency).

The figure below shows how the user interface of the Hypermonitor is implemented in the VPHOP project. The CHIC version of the hypermonitor will require massive refactoring, due to the different specifications and to the heavy refactoring and extension of the VPH-HF.

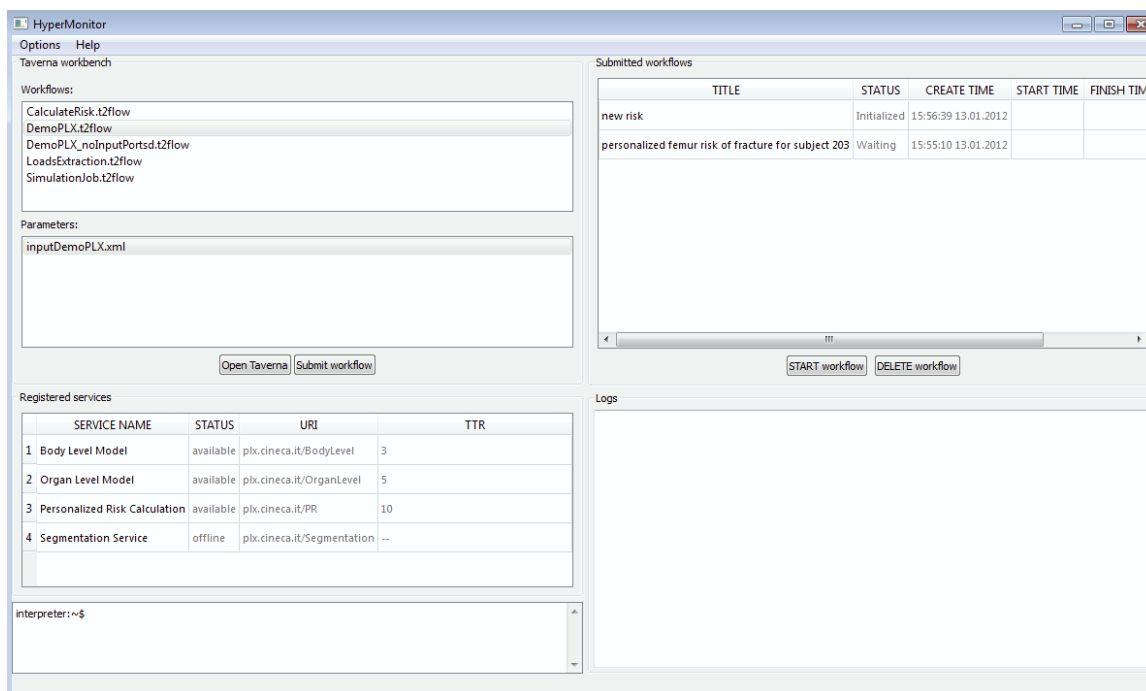


Figure 4 Screenshot of the Hypermonitor user interface.

5.3.3 Hypermodelling execution framework

5.3.3.1 Description

5.3.3.2 Functionality

ID	Functionality
1	Expose a real-time status update on all hypomodels available, in their respective implementations (in principle the same hypomodel could be available on multiple nodes of the cloud).
2	Authorise the execution requests and the access to the necessary input data using the CHIC authentication services.
3	Orchestrate the execution of a hypermodel with a given input set, managing both the control flow and the data flow among the hypomodels that compose it.
4	Maintain a global log of all hypomodel executions and of their orchestration, to allow debugging and execution monitoring.
5	Replicate in the in silico trials repository the results of the hypermodel execution for a given input set.

5.3.3.1 Programmatic/User Interface

The hypermodelling execution services are implemented with the VPH Hypermodelling Framework (VPH-HF), a MAF3 application that makes possible the deployment of distributed execution services for complex orchestrations of numerical models developed with heterogeneous modelling software. The VPH-HF, originally developed as part of the VPHOP project, is now being aggressively refactored to fulfil the complex needs of the CHIC project.

At the centre of VPH-HF there is a proprietary bus that manages the control flow across different modules; the bus can be segmented across multiple memory spaces / computers allowing a distributed orchestration. The modules are service components also written with the MAF3 software framework, or models developed with any software tools, whose execution component is wrapped with thin software layer that “translates” the control flow syntax of each modelling tool into the generic one supported by the VPH-HF bus. The data flow is currently managed by the wrapper, which creates on a local file system a sandbox, replicates locally all input data required by that hypomodel, and copies back to the data repository the output data.

In the current version all the communications between external services, modules, and the various bus segments are managed using a low-overhead XML-RPC protocol that was chosen in VPHOP for efficiency reasons. We are now exploring the possibility to add a SOAP communication protocol, where this does not significantly reduce the efficiency.

VPH-HF acts a service provider for the status update of the hypomodels, for the execution of given hypermodels with given inputs data, and for the global log of such execution. Currently, the only consumer of these services is the Hypermonitor application, also written in MAF3; the API is proprietary to MAF3.

VPH-HF acts as a service consumer for the authentication services, the storage services, and the semantic services; we expect standard WSDL interfaces for each of these services.

VPH-HF has no user interface, but APIs can be provided to access specific VPH-HF services by other CHIC architectural components: in particular to submit workflows, to get information on a workflow via its ID, to delete or stop a workflow execution.

5.3.4 Visualization toolkit

5.3.4.1 Description

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

5.3.4.2 Functionality

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

5.3.4.1 Programmatic/User Interface

Data visualisation is implemented with JQuery, D3.js and three.js. JQuery is a fast, small, and feature-rich JavaScript library. It makes HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, JQuery has changed the way of JavaScript programming.

D3.js is a JavaScript library for web-based information visualisation using HTML, SVG and CSS. D3 gives the full capabilities of modern browsers without tying to a proprietary framework, combining powerful visualisation components and a data-driven approach to DOM manipulation. For example the timeline visualisation will be based on D3.js.

Three.js is a WebGL-based lightweight cross-browser JavaScript library/API for web-based 3D graphics and visualisation in web browsers. It simplifies WebGL programming by providing high-level programming interfaces which are similar to traditional OpenGL APIs. Three.js scripts may be used in conjunction with the HTML5 canvas element, SVG or WebGL. For example visualisation of 3D segmentation surfaces can be achieved with three.js.

5.3.5 Image processing toolkit

5.3.5.1 Description

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

5.3.5.2 Functionality

ID	Functionality
1	Loading and Writing of medical image data (.dcm format).
2	Visualization of medical image data.
3	General image processing capabilities (such as image smoothing, resampling, etc.) provided by plugins.

4	Offers a variety of tools for definition and refinement of manual annotations (Magic wand, Contouring tool, histogram tool).
---	------------------------------------------------------------------------------------------------------------------------------

5.3.5.1 Programmatic/User Interface

Doctor Eye requires .NET framework (3.5. or higher). The integrated image processing routines rely on the Insight Registration and Segmentation Toolkit (ITK) library.

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

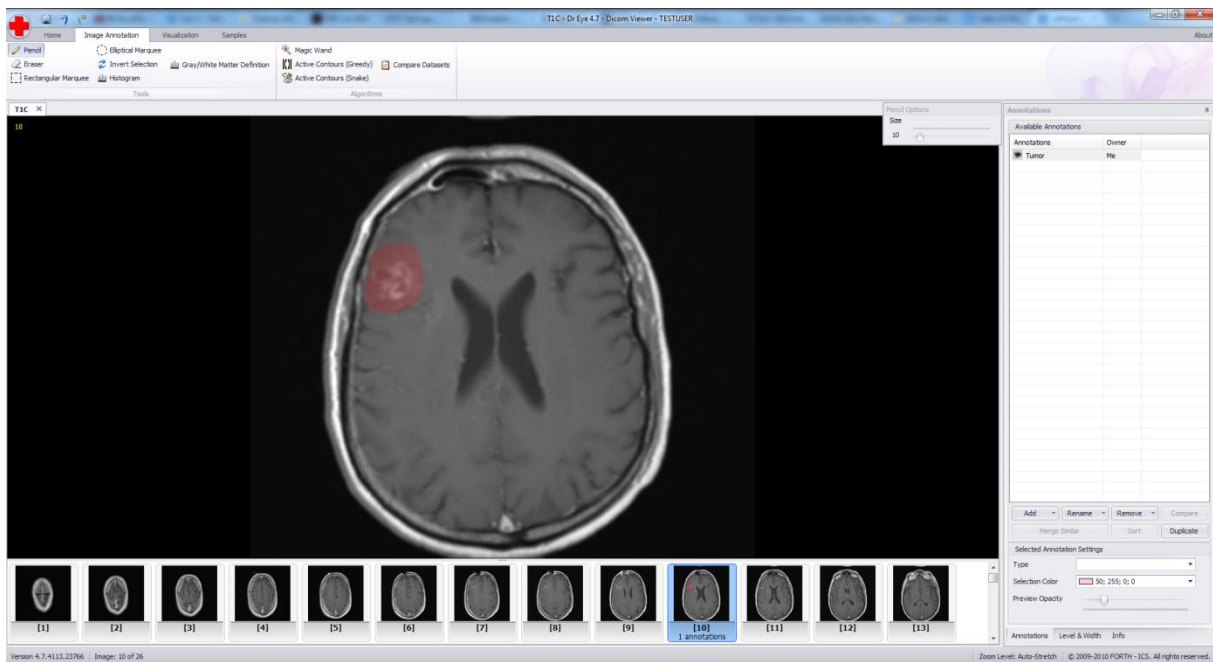


Figure 5 Screenshot of DrEye

5.4 Data Integration layer

5.4.1 Model repository

5.4.1.1 Description

The model repository is the physical entity where the modules used in construction and execution of hypermodels are stored. The aforementioned modules can be hypomodels, hypermodels, linkers, data transformation tools and other tools. The model repository also provides the necessary interfaces for accessing stored modules or for storing new modules, by both end users and other components of the CHIC architecture.

5.4.1.2 Functionality

ID	Functionality
1	Adding a new module, including all its related information (descriptive information, input and output parameters, executable and documentation files etc). The functionality is provided to the end user through a graphical user interface.
2	Updating any information stored in the repository. The functionality is available to either owners of the module that the information is related to or to administrators. The functionality is provided to the end user through a graphical user interface.
3	Deleting a module. The functionality is available to either owners of the module that the information is related to or to administrators. The functionality is provided to the end user through a graphical user interface.
4	Filtering stored modules according to specific criteria. The functionality is provided to the end user through a graphical user interface and to other components through web services.
5	Full text search of the repository. The functionality is provided to the end user through a graphical user interface and to other components through web services.
6	Storing hypermodels constructed by the hypermodeling editor using web services.
7	Updating the metadata repositories if necessary.
8	Access control to the model repository (Single sign on)

5.4.1.1 Programmatic/User Interface

The primary user interface is web based. Additionally the model repository will be accessible through a “portlet” integrated into the CHIC portal.

Furthermore, the model repository will incorporate a REST read-write API for searching and retrieving modules (hypomodels, hypermodels, linkers, data transformation tools and other tools) and for storing constructed hypermodels. This API will be secured (authentication and authorization) according to the adopted security framework in CHIC.

More information about the aforementioned API would be provided in deliverable “D10.2 Design of the orchestration platform, related components and interfaces”.

5.4.2 Clinical data repository

5.4.2.1 Description

The clinical data repository is used by the data upload tool and OBTIMA to write (store) clinical data. Modellers, researchers, and other experts have access to read (view) the clinical data stored in the CHIC portal. The hypermodelling infrastructure and the visualization and image processing toolkit have access to read the clinical data stored for processing purposes.

5.4.2.2 Functionality

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

5.4.2.1 Programmatic/User Interface

The clinical data repository is accessible through a web user interface integrated into the CHIC portal. Additionally, it supports a REST API for uploading, annotating, organizing, searching and downloading clinical data. This API will be secured (authenticated and authorized) according to the adopted security framework in CHIC. The following pseudo-code illustrates how a single file can be uploaded to the clinical data repository using the REST API.

```
POST https://virtualskeleton.ch/api/files/upload HTTP/1.1
Content-Type: multipart/form-data; boundary="--09335213-878a-416b-b469-35551259de44"
Authorization: Basic ZGVtb0B2aXJ0dWFsc2t1bGV0b24uY2g6ZGVtbw==
Content-Length: 161300

--09335213-878a-416b-b469-35551259de44
Content-Disposition: form-data; filename=sample.dcm

"content of sample.dcm"

--09335213-878a-416b-b469-35551259de44--
```

Figure 6 Example of a file upload to the clinical data repository over a RESTful API.

The primary user interface to the repository is done using a web interface. This interface has been designed to allow the users to upload datasets, browse/search personal and shared data, organize objects in virtual project folders and download available objects for local processing. The concept of virtual personal project folders will allow the user to freely organize objects into a folder structure. Additionally, shared group folders will be implemented. These folders will be accessible and modifiable by defined groups of users, which enable collaboration within research teams on organized set of objects. In addition, the virtual folder structure – both private and group folders – will be mirrored on the REST interface, making access to the file straightforward. In order to consistently annotate uploaded data for later search, a comprehensive anatomical ontology for the annotation of anatomical regions will be used.

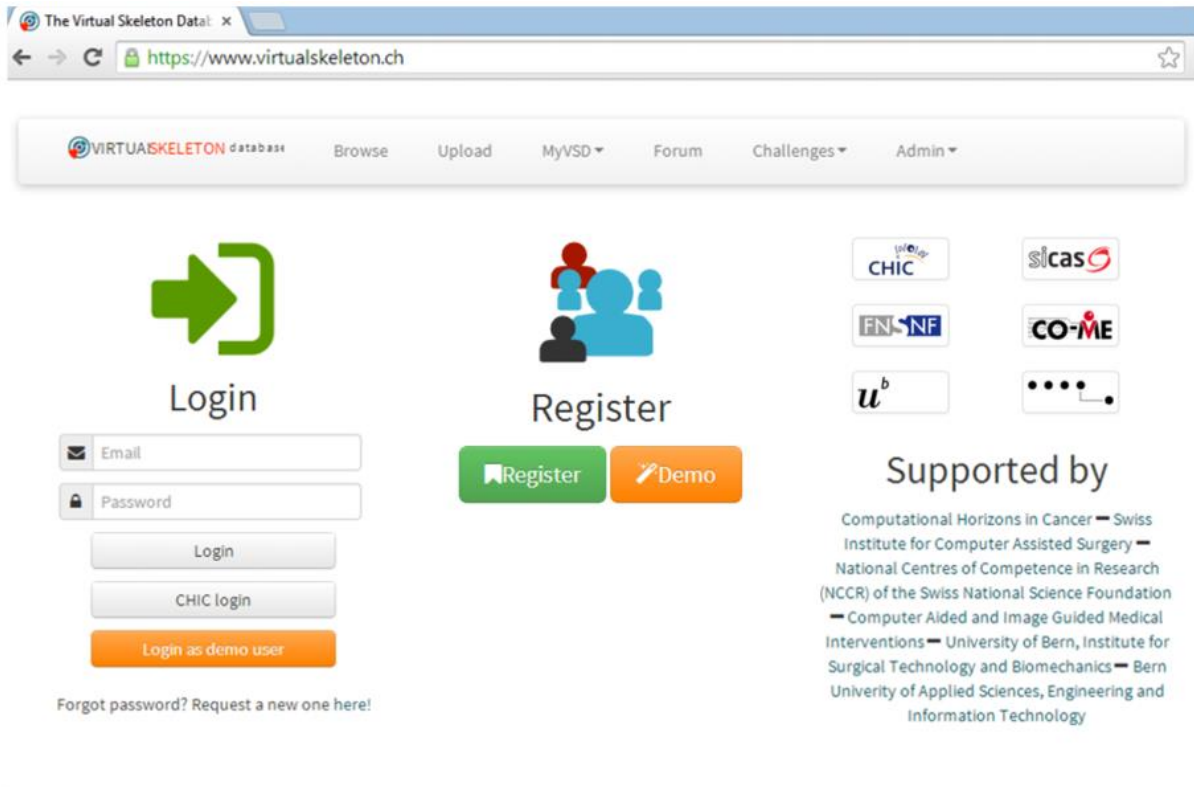


Figure 7 Screenshot of the Clinical data repository interface (login screen)

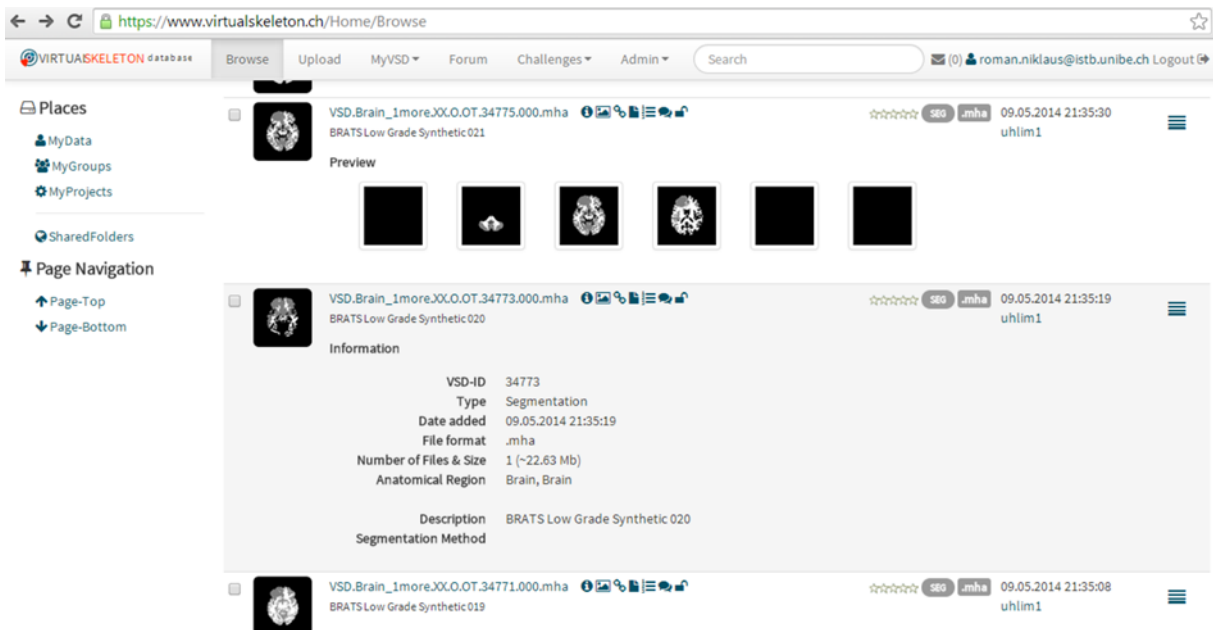


Figure 8 Screenshot of the Clinical data repository interface

5.4.3 In Silico trial repository

5.4.3.1 Description

The in-silico trial repository is the physical entity where in-silico trial are stored. Every “instance” of the in-silico trial consists of three parts:

- Its specification (the hypermodel to be used),
- The patient participating in the in-silico trial and its corresponding data (the input data set to be used in the execution of the hypermodel) and
- The results produced when the in-silico trial is completed (the output of the hypermodel execution).

The in-silico trial repository also provides the necessary interfaces for accessing stored information or for storing additional one, by both end users and other components of CHIC architecture.

5.4.3.2 Functionality

ID	Functionality
1	Adding a new instance of an in-silico trial. The instance can be partially created (input data and/or hypermodel) or can be fully created (input data, hypermodel and results). The functionality is provided to the end user through a graphical user interface and to other components through web services.
2	Updating an instance of an in-silico trial. The functionality is provided to the end user through a graphical user interface and to other components through web services.
3	Deleting any information stored in the repository. The functionality is available only to administrators. The functionality is provided to the end user through a graphical user interface.
4	Filter the instances of an in-silico trial according to specific criteria. For example the instances related to a specific hypermodel or to a specific patient may be presented. The functionality is provided to the end user through a graphical user interface and to other components through web services.
5	Access control to the in-silico trials repository (Single sign on)

5.4.3.1 Programmatic/User Interface

The primary user interface is web based. Additionally the in-silico trial repository will be accessible through a “portlet” integrated into the CHIC portal.

Furthermore, the in-silico trial repository will incorporate a REST read-write API for searching, retrieving and storing of in-silico trial instances. The aforementioned instances:

- Can be created with partial information (specifying only the input data and/or the hypermodel to be used).
- Can be updated (addition of hypermodels and/or results).

- Can be created with complete information (input data, hypermodel, results).

This API will be secured (authentication and authorization) according to the adopted security framework in CHIC.

More information about the aforementioned API will be provided in deliverable “D10.2 Design of the orchestration platform, related components and interfaces”.

5.4.4 Metadata repository

5.4.4.1 Description

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

5.4.4.2 Functionality

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

5.4.4.1 Programmatic/User Interface

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

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

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

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

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

5.4.5 OBTiMA data push services

5.4.5.1 Description

The push services enable the syntactic and semantic integration of hospital information system (HIS) data via ObTiMA with data from other heterogeneous biomedical data sources. They support data owners to retrieve data from HIS and other biomedical databases and import them into ObTiMA. They integrate the data semantically and therefore, enable the seamless reuse of the data by the various analysis tools and services of the CHIC platform. This Push service was developed within the p-medicine project by Fraunhofer IBMT and can be exploited within CHIC. A complete description of this push service is provided in deliverable D8.5 (Release of services for the interaction with Hospital Information Systems) of the p-medicine project. This deliverable is restricted to p-medicine, but it is agreed that parts of the deliverable can be made public for the CHIC project.

5.4.5.2 Functionality

ID	Functionality
1	Provide an Interface to reuse data from ObTiMA in the clinical data repository of CHIC
2	Allowing the reuse of data from ObTiMA in the CHIC project

5.4.5.3 Programmatic/User Interface

The Data Upload tool is part of the ObTiMA application for “pushing” the data through the security/pseudonymization framework to their final destination, which is the clinical data repository of CHIC. It provides a task in ObTiMA for the push services and strives to be a user-friendly tool hiding a lot of complexity behind. Its main functionality is to allow a user to load files containing patient data after pseudonymisation into the data repository. The trial management system ObTiMA allows collecting data on Case Record Forms (CRFs). Questions on the CRFs can be annotated with the HDOT ontology of p-medicine by using an annotation module integrated in ObTiMA. Therefore in ObTiMA the data are already ontology-annotated, when the CRFs are created accordingly. The transfer can be conducted fully automatically from the ObTiMA user interface.

The developed push service described in p-medicine needs to be adapted to the requirements of the CHIC project. In the p-medicine design we have two data warehouses, the local and the central. The data originating from a hospital information system (HIS) go through a 1st round of pseudonymization to be stored in a local repository and then imported in OBTiMA. After that these data go then through a 2nd round of pseudonymization and are imported into the central data warehouse of p-medicine (see Figure 9).

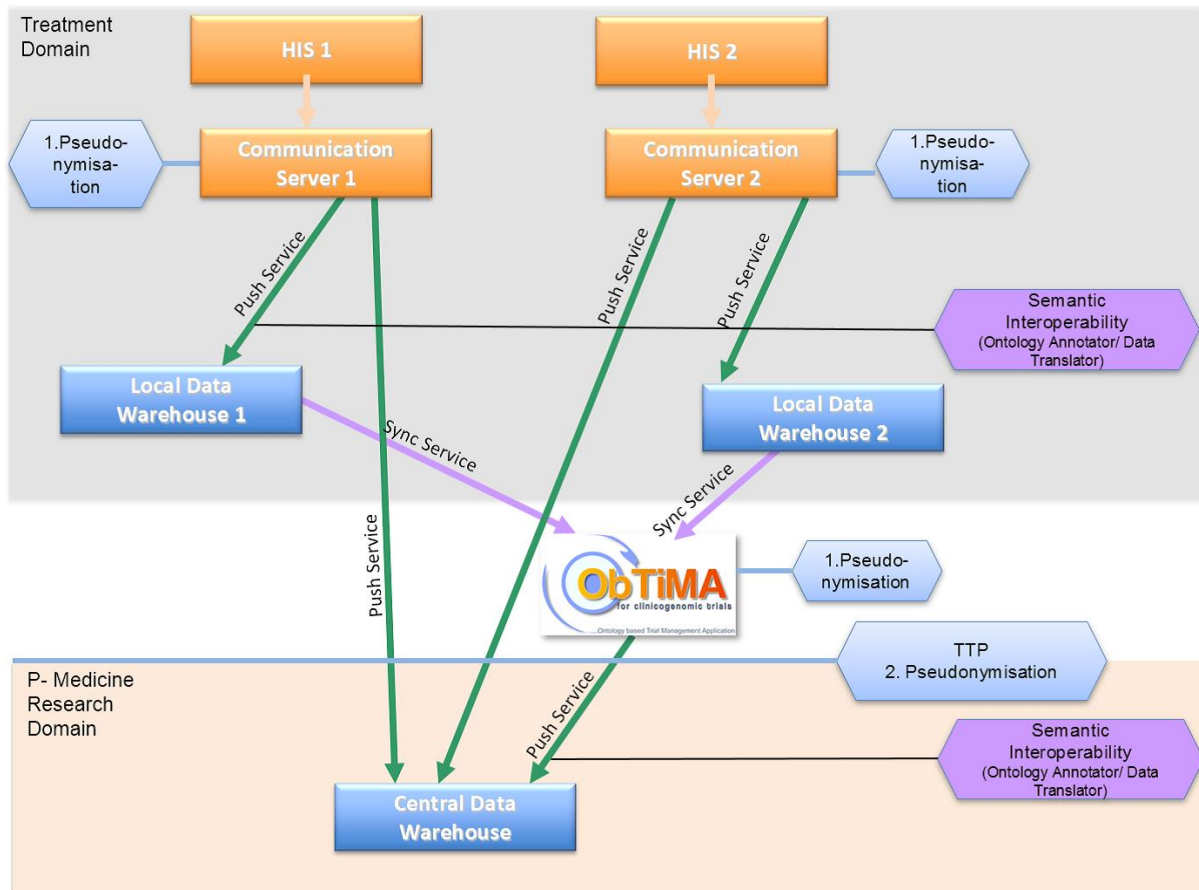


Figure 9 The push and sync services as developed in the p-medicine project.

In CHIC we will have the data flow as shown in Figure 13 in chapter 6.2. In this figure, we just describe that data from OBTIMA will pass through the security/pseudonymization framework and will be imported into the Clinical data repository of CHIC. It is subject to be decided whether the architecture will follow the same approach as in the p-medicine project, having also a 1st round of pseudonymization and a local (located at the hospital) data warehouse, something which would greatly help achieve interaction and interoperability between the two projects.

5.4.6 Interoperable interfaces

5.4.6.1 Description

This functional component provides a standardized programmatic interface for retrieving input parameters for a certain patient, loaded from the Clinical Data Repository and evaluating input parameters on a selected model, retrieved from the underlying Model Repository. The evaluation is realized through interoperating with the corresponding to the model, execution engines.

5.4.6.2 Functionality

ID	Functionality
1	Receive Parameter Values
2	Retrieve the corresponded model and model input parameters from the Model Repository
3	Retrieve the Patient Data from the Clinical Data Repository
4	In case value interpretation is needed into an appropriate format, an interpretation model is executed.
5	Evaluate the input parameters with the retrieved model on the corresponding execution engine.
6	Return the results of the evaluation

5.4.6.1 Programmatic/User Interface

This functional component will provide an XML Web Service interface to:

- Retrieve values of a certain model input parameter for a given patient;
- Execute a model for a given set of input parameters, returning a set of values of output parameters;

This functionality will be exposed through the following methods:

- For the value retrieval, the following method will be available:

Method	Description
<pre>List<ParameterValue> getInputParamaterValues(int modelId, String patientId, String queryEngineUrl, String modelRepositoryUrl)</pre>	Retrieves values of all input parameters for a model with a given id for a given patient. This call requires URLs of actual query engine and models repository web services;

Method	Description
<pre>List<ParameterValue> executeModel(int modelId, List<ParameterValue> inputValues, String modelRepositoryUrl)</pre>	Evaluates model with a given id for given set of input parameters. URL of models repository web service is required;

- For the model execution, the following method will be available:

This component will be secured according to the adopted security framework in CHIC. This component will not include any User Interface.

5.5 Security layer

5.5.1 Description

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

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

The security framework contains the following tools & components:

1. Authentication and Identity Management Components
 - a. Identity and Access Management Site (IAM)
User enrolment and management site and services through which users can be enrolled, revoked, edited ...
 - b. Identity Provider (IdP)
A service which is responsible for authentication within CHIC. It provides identity assertions to web sites accessed from a browser.
 - c. Secure Token Service (STS)
A service which is responsible for authentication within CHIC. It provides identity assertions to services accessed from a non-browser client.
2. Authorization components
 - a. Policy Decision Point (PDP)
A PDP is the entity which takes authorization decisions. A PDP accepts authorization requests.
 - b. Policy Administration Point (PAP)
A PAP is the endpoint which manages policies. The PAP provides the PDP with all policies required to produce an authorization decision. The PAP also consists of management services where authorization rules can be configured generating authorization policies.
 - c. Policy Information Point (PIP)
Policies evaluated by the PDP need information (attributes) about the actors to be able to take an authorization decision. Most attributes are typically provided through the authorization requests. A PIP is responsible for providing the missing attributes to the PDP.
 - d. Policy Enforcement Point (PEP)
A PEP is the component which integrates with application code. The PEP is responsible for creating the authorization request and sends it to the PDP.
3. Audit Service
4. Security gateway/proxy

5. Integration modules and extensions
 - a. Various integration modules are available within CHIC to integrate JAVA, PHP and .NET applications into the security framework.
 - b. Extensions
 - c. Liferay Extension
6. De-identification tool (CATS)

5.5.2 IAM (Identity and Access Management site)

5.5.2.1 Description

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

5.5.2.2 Functionality

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

5.5.2.3 Programmatic/User Interface

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

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

5.5.3 IdP (Identity Provider)

5.5.3.1 Description

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

5.5.3.2 Functionality

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

5.5.3.3 Programmatic/User Interface

HTTP interfaces as defined in the SAML Web Browser SSO Profile. More detailed information on security profiles will be available in deliverable “D5.2: Security Guidelines and initial version of security tools” due on M18.

5.5.4 STS (Secure Token Service)

5.5.4.1 Description

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

5.5.4.2 Functionality

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

5.5.4.3 Programmatic/User Interface

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

5.5.5 PDP (Policy Decision Points)

5.5.5.1 Description

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

5.5.5.2 Functionality

ID	Functionality
1	Take authorization decisions: permit or deny

5.5.5.3 Programmatic/User Interface

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

5.5.6 PAP (Policy Administration Point)

5.5.6.1 Description

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

5.5.6.2 Functionality

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

5.5.6.3 Programmatic/User Interface

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

5.5.7 PIP (Policy Information Point)

5.5.7.1 Description

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

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

5.5.7.2 Functionality

ID	Functionality
1	Query a subject or resource attribute

5.5.7.3 Programmatic/User Interface

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

5.5.8 PEP (Policy Enforcement Point)

5.5.8.1 Description

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

5.5.8.2 Functionality

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

5.5.8.3 Programmatic/User Interface

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

5.5.9 Audit Services

5.5.9.1 Description

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

5.5.9.2 Functionality

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

5.5.9.3 Programmatic/User Interface

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

5.5.10 Security Gateway

5.5.10.1 Description

Not every service provider can easily integrate with the CHIC security framework. To enable these services to be integrated with the security framework a security gateway is available which is responsible for handling all security protocols. Identity information is then passed by the gateway to backend service providers through HTTP headers.

5.5.10.2 Functionality

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

5.5.10.3 Programmatic/User Interface

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

5.5.11 Security Integration Modules and extensions

5.5.11.1 Description

Modules and libraries are available within CHIC which integrate the security framework in various programming languages (e.g. Java, PHP, .NET), web server implementations (e.g. Apache HTTP, Apache Tomcat) and applications (e.g. Liferay). These modules will be described in more detail in deliverable “D5.2: Security Guidelines and initial version of security tools” due on M18.

5.5.12 De-Identification

5.5.12.1 Description

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

5.5.12.2 Functionality

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

5.5.12.3 Programmatic/User Interface

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

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

5.6 Semantics layer

5.6.1 Ontology-based semantic services

5.6.1.1 Description

Ontology-based semantic services include:

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

5.6.1.2 Functionality

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

5.6.1.1 Programmatic/User Interface

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

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

REST services are used to access the Metadata Repository.

REST services are used to access the Knowledge Base.

5.6.2 Folksonomy semantic services

5.6.2.1 Description

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

5.6.2.2 Functionality

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

5.6.2.1 Programmatic/User Interface

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

- to add tags to a resource with suggestions from existing tags and ontology terms (e.g. Gene Ontology, Cell Ontology), see Figure 10 for a snapshot of the initial version;
- to view tags, edit or delete them, see Figure 11 for a snapshot of the initial version;
- to view community tags, see Figure 12 for a snapshot of the initial version.

The RESTful APIs provide the following functions:

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

Tagging Service Client
Add Tags
View Tags
View Community Tags

User ID

Resource URI

Tags

- Big toe<FMA><FMA_25047>
- binding<CL><GO_0005488>
- binding<molecular_function><GO:0005488>
- Biglycan<FMA><FMA_82825>
- Bile duct<FMA><FMA_9706>
- biconcave<CL><PATO_0002039>
- bile duct<CL><UBERON_0002394>
- binucleate<CL><PATO_0001406>
- biliary bud<CL><UBERON_0004912>
- Biliary tree<FMA><FMA_14665>
- biliary tree<CL><UBERON_0001173>
- biotin import<biological_process><GO:1901689>
- Biceps brachii<FMA><FMA_37670>
- Biceps femoris<FMA><FMA_22356>
- Biliary system<FMA><FMA_79646>
- Biogenic amine<FMA><FMA_67161>
- Bipolar neuron<FMA><FMA_67282>
- bilaminar disc<CL><UBERON_0000091>
- biliary system<CL><UBERON_0002294>
- biofilm matrix<cellular_component><GO:0097311>

Figure 10 Folksonomy service snapshot – add a tag to a resource

Tagging Service Client
Add Tags
View Tags
View Community Tags

filter by

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

Figure 11 Folksonomy service snapshot – view/edit/delete tags

Tagging Service Client	
Tag	Occurance
hypomodel	9

Figure 12 Folksonomy service snapshot – view community tags

5.6.3 Knowledge base

5.6.3.1 Description

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

5.6.3.2 Functionality

ID	Functionality
1	Storage of ontologies
2	Inferencing

5.6.3.1 Programmatic/User Interface

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

5.7 Infrastructure layer

5.7.1 Private Cloud

5.7.1.1 Description

The CHIC private cloud is an infrastructural component having as its main purpose to provide computational and storage facilities to other components. This component is described in more detail in the Deployment View, in section 7.

5.7.1.2 Functionality

ID	Functionality
	Provides computational power to other components, such as the hypermodelling execution framework.
	Provides storage space to other components, such as the storage repositories.

5.7.1.1 Programmatic/User Interface

The CHIC private cloud is based on the OpenStack cloud infrastructure software platform.

The OpenStack platform offers a variety of tools for accessing the underlying services, such as command line interface (CLI) and tools, software development kits (SDKs) for many programming languages such as Java, Python, .NET, Ruby, Node.js and PHP. It also offers an application programming interface (API) for direct programmatic access of its services, based on RESTful web services, such as Block Storage, Object Storage, Compute, Identity, Orchestration etc.

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

6 Information view

In this section we present the Information View, which describes the way that the system stores, manipulates, manages, and distributes information. This view answers the questions around content, structure, ownership, references and flow of information. In the following paragraphs we present a broad view of the static data structure and the information flow between the information repositories and the other functional components.

6.1 Introduction – High level view

The most important information of a system is the one that we choose to store in non-volatile, or long-term, memory. This is the information that we cannot afford to lose and is stored in various persistent repositories of the platform. Besides the core data sets there is also additional information flowing around the functional components of the platform, but this information is only necessary during the runtime of a system, so we don't focus on it in the Information View; this runtime and communication information is presented in the corresponding sections of each functional component.

The essential functionality that each repository offers is to read and write information in it, so in order to describe the information view we need to capture the schema of the stored information and the list of components that each repository interacts with.

The core data sets in CHIC are:

- **Clinical data**

Clinical data sets originating from Hospital Information Systems (HIS) or ongoing clinical trials (e.g. from OBTIMA).

- **Models and other tools**

- Hypomodels, hypermodels, linkers, data transformation tools and other tools developed in the context of CHIC and other projects.

- **In Silico trial data**

Data participating in an in-silico trial. This data includes the input data set, the hypermodel participating in the in-silico trial and the resulting data set produced by the execution of the hypermodel.

- **Metadata**

Data collected or generated by the CHIC platform, in order to describe all the other data sets above; primarily the clinical data and models.

- **Ontologies**

Ontologies which specify the theory of a domain that is relevant to the annotation of CHIC resources.

- **CHIC Users info**

Data regarding the CHIC end users, their permissions and other security related information.

Although we store each core data set in its own **logical** repository, in terms of implementation parts or all of this information may reside to the same **physical** repository. This depends on the implementation of the architecture, so further information will be provided in the section where the Deployment View is analysed.

6.2 Data flow and interaction of the repositories with the functional components

In Figure 13 we present the major building blocks of the information view of the CHIC platform.

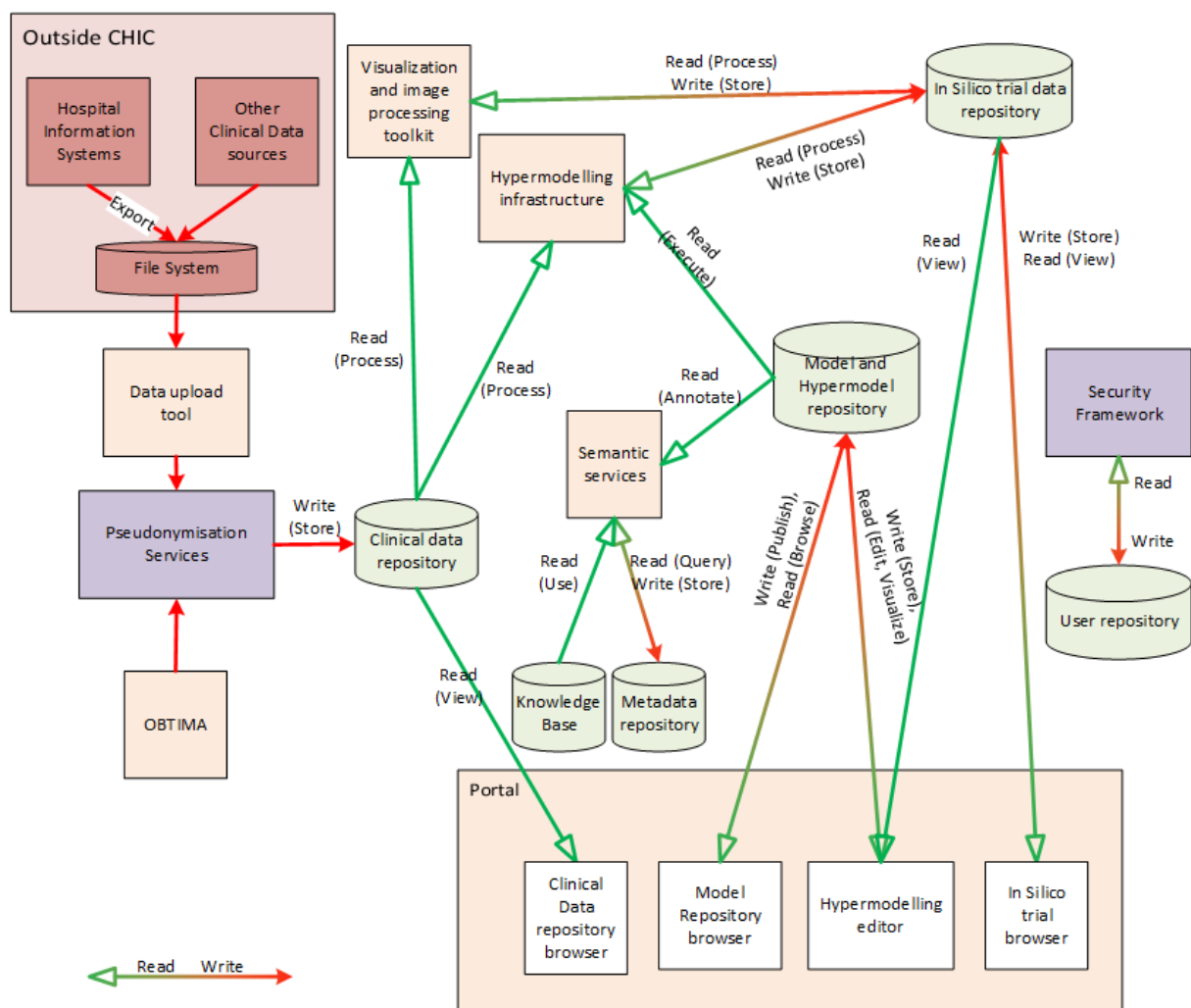


Figure 13 Information view: Repositories and interacting components

Here we can see the major repositories where crucial information is stored:

- The Clinical data repository, which stores de-identified clinical data originating from hospitals or clinical trials (through OBTIMA).
- The Model and Hypermodel repository, which store models or integrative models (hypermodels) respectively.
- The In-Silico trial repository, which stores the input data set, the hypermodel participating in the in-silico trial and the resulting data set produced by the execution of the hypermodel.
- The Metadata repository, which stores meta-information describing the clinical data and the models.
- The Knowledge Base, which stores the ontologies used by the semantic services.
- The User repository, which stores the CHIC user information, such as authorized users, credentials etc.

Figure 13 depicts which functional components interact with the repositories and for what purpose, either to read or write information in them. We depict only the major components and the major interaction purposes since, all functional components use or process information of those repositories, although sometimes indirectly through other components.

Elaborate details on the design and implementation of all the repositories, such as the schema and format of the stored information, will be provided in the deliverable D8.1 “Design of the CHIC repositories”. Below we give two indicative examples, describing the information stored in the Clinical data repository and in the User repository.

6.3 Clinical data

Several types of clinical data will be stored in this repository. Clinical image datasets of the patients will be stored. These datasets represent 2D or 3D stack of images stored in the standard DICOM (Digital Imaging and Communications) file format. The images will be provided at different time point of the patients’ treatment. In addition, information on the clinical trial will be provided and stored in the repository. These data will be exported from the clinical trial management system (such as Obtima) and stored in the repository in a Comma Separated (csv) or xml file format. Appropriate internal relational link will enable to retrieve all information relative to a specific patient. This connection will be performed based on the pseudonymization provided by the Trusted Third Party. The clinical database will also be able to store images processed by the clinician as part of the treatment procedure. Typically, manual segmentation of the tumour will be stored as binary DICOM image in the system.

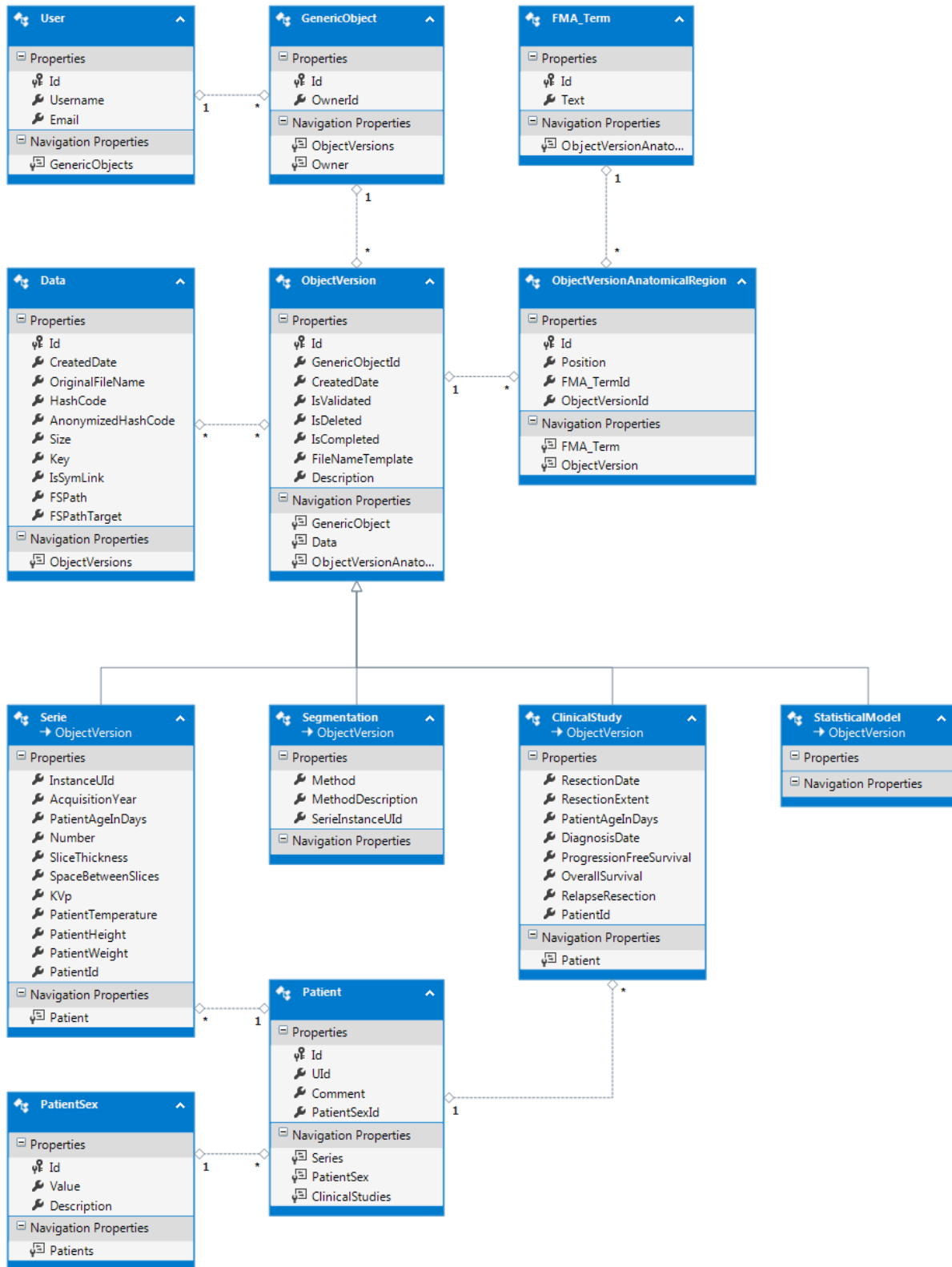



Figure 14 Schema of the clinical data repository database.

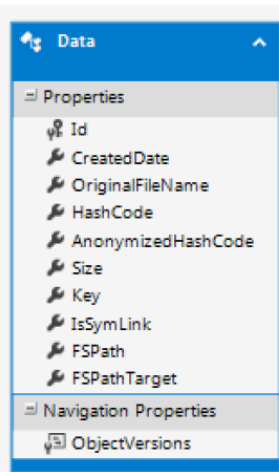
The database schema illustrated above is not complete yet. The purpose of this schema is to provide an approximate overview about how data will be stored within the clinical data repository. All

uploaded files are represented in the Data table. One of the most important design aspects is the fact, that the files itself are not stored within the database. This can be best explained with a sample DICOM (Digital Imaging and Communications in Medicine) file.

User filesystem:


Name	Änderungsdat...	Typ	Größe
 sample.dcm	08.03.2010 18:17	DCM-Datei	550 KB

Database:



Id	1
CreatedDate	2014-05-14 16:56:29.770
OriginalFileName	sample.dcm
HashCode	6F7533F66B7C6783CFA7391CB5439A48522B8712
AnonymizedHashCode	A3E3EE841FE0311CD20C92F3315791C7123D2D68
Size	563200
Key	0
IsSymLink	False
FSPath	Dicom\0000afc0b2c94eeda1e0e54a690c47c3.vsd
FSPathTarget	NULL

Clinical data repository filesystem:

Name	Date modified	Type	Size
 0000afc0b2c94eeda1e0e54a690c47c3.vsd	10.06.2011 09:23	VSD File	550 KB

During the upload process it is possible to extract information contained within the file. For a DICOM this could be for example the acquisition year, slice thickness, space between slices, etc. For a clinical study file this could be for example the date of resection, extent of resection, progression free survival and so on. *The information that should be extracted still needs to be defined.*

6.4 User repository

In Figure 15 we present in brief the schema of the User repository database. Below we describe some indicative attributes of this schema in order to give an exemplar overview of the stored information.

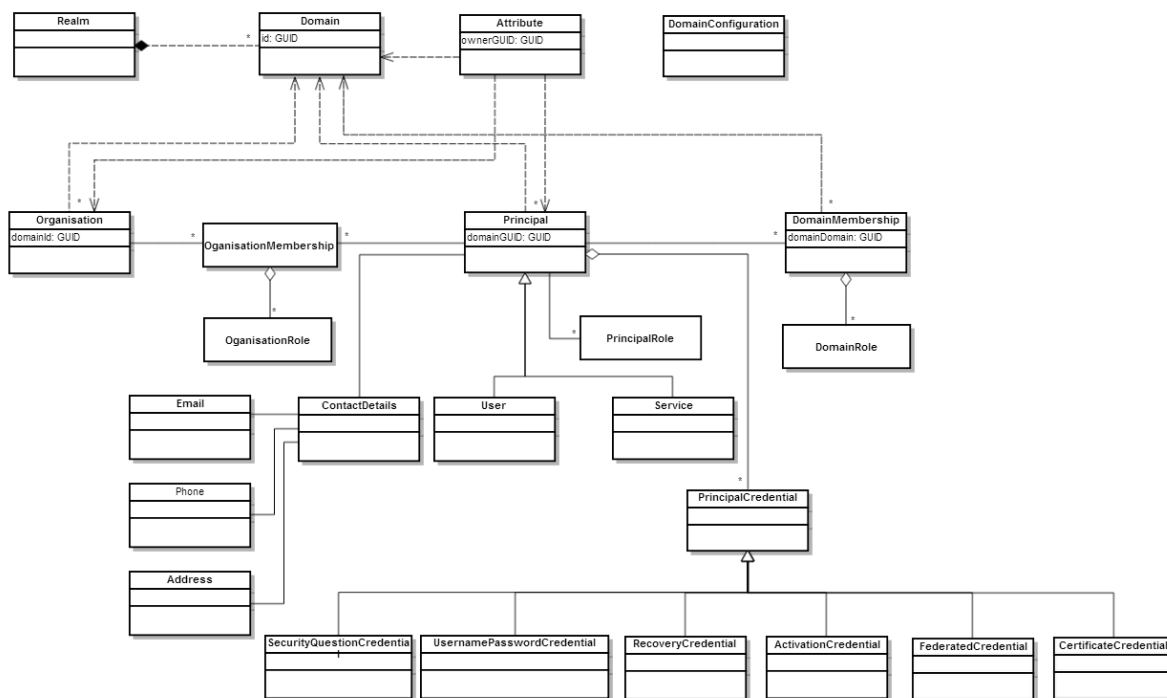


Figure 15 Schema of the user repository database.

- The **Realm** is the root node of the hierarchical tree. There can be only one realm in the user repository. All domains are thus part of the same realm.
- A **Domain** is identified by its GUID. This GUID will be used by principals and organisations to reference their domain participation weakly. This way there is loose coupling between domains and Principal/organisation. This is needed because in future versions each Domain will have its own data source (database, LDAP,...) .
- **DomainRole**. By default users have the member role in a domain. Additionally users can be assigned other membership roles.
- **OrganisationRole**. Users assigned to an organisation by default have the member role. Additionally users can be made organisation administrator.
- **User**. A user is a human principal.
- **Service**. A service is a non-human principal.
- **ActivationCredential**. This credential is used when a principal logs in through the activation page.
- **FederatedCredential**. This credential uniquely identifies a principal and is inserted in a user’s SAML token.
- **RecoveryCredential**. This credential is used when a principal logs in through the recovery page.
- **UsernamePasswordCredential**. A credential used to log in on IdP/STS using username and password.

- **DomainMembership.** Domain membership indicates a principal's role in a particular domain.
- **OrganisationMembership.** Organisation membership indicates that a principal is member of the organisation and its organisation role.

7 Deployment view

7.1 Introduction – high level view

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

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

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

7.2 User Interface layer

7.2.1 User Portal

Resource	Mapping
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Web application/framework
Operating system	Windows/Linux (currently installed on Windows but expected to be installed on Linux, once in production mode)
Programming environment	Java programming language, Tomcat application server
Communication technologies or protocols with other CHIC components	HTTP/XML/JSON/REST services
Data types that it operates on	N/A
Technical dependencies or requirements (libraries, tools)	Liferay/Tomcat/MySQL
Computational needs (estimation of hardware needs)	Server machine running a web server. Depends on the expected load of the portal.
Storage needs (estimation of hardware needs)	1 GB of storage space

URL (if applicable)	
---------------------	--

7.3 Business Logic layer

7.3.1 Hypermodelling editor

Resource	Mapping
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Web based
Operating system	Cross platform, but Linux is the preferred host Operating System
Programming environment	Node.js/Javascript
Communication technologies or protocols with other CHIC components	HTTP/REST based API with JSON and XML payload
Data types that it operates on	Hypermodel descriptions
Technical dependencies or requirements (libraries, tools)	Node.js/Express application server, PostgreSQL DBMS for persistence, Redis as a caching layer, Nginx web server as a web reverse proxy
Computational needs (estimation of hardware needs)	Moderate requirements, a server machine with plenty of RAM is always preferred.
Storage needs (estimation of hardware needs)	1 GB of storage but depending on usage
URL (if applicable)	N/A

7.3.2 Hypermonitor

Resource	Mapping
Developer	USFD - CINECA
Software type (Web application, standalone application, service, library, framework etc.)	Standalone application
Operating system	Cross-platform but MAC and Windows the most tested release.
Programming environment	C++, python
Communication technologies or protocols with other CHIC components	It connects via HTTP/REST APIs to the hypermodelling framework components functionalities it does not provide any communication by itself.
Data types that it operates on	Workflow description file.

Technical dependencies or requirements (libraries, tools)	MAF3 is the library at the base of the VPH-HF implementation. MAF3 is based principally on Qt, and on a set of pre-compiled libraries that represents the "Foundation Libraries" including last stable versions of ITK, and VTK.
Computational needs (estimation of hardware needs)	Hypermonitor is a very light application so it has minimum computation needs (1GB RAM).
Storage needs (estimation of hardware needs)	No data are stored locally by the application but are managed by the hypermodelling execution framework.
URL (if applicable)	N/A

7.3.3 Hypermodelling execution framework

Resource	Mapping
Developer	USFD - CINECA
Software type (Web application, standalone application, service, library, framework etc.)	Software framework
Operating system	Cross-platform but Unix/Linux the most tested release.
Programming environment	C++, python Taverna workflow orchestrator is based on Java: it is being used but not developed during CHIC.
Communication technologies or protocols with other CHIC components	HTTP/REST APIs to access the hypermodelling framework components functionalities. Supported protocol is XML-RPC but SOAP is being considered for inclusion.
Data types that it operates on	Workflow description file. The hypermodelling framework will have also to access models, data, and any other information necessary to the workflow execution.
Technical dependencies or requirements (libraries, tools)	MAF3 is the library at the base of the VPH-HF implementation. MAF3 is based principally on Qt, and on a set of pre-compiled libraries that represents the "Foundation Libraries" including last stable versions of ITK, and VTK. The Workflow Manager in VPH-HF is implemented based on the Flask Python Microframework. Postgres SQL database is used as well.
Computational needs (estimation of hardware needs)	The VPH-HF is an orchestration layer. Itself it needs a couple of cores, and maybe 1GB of RAM;

	to this it has to be added all the cores and memory that the hypomodels to be orchestrated require to execute.
Storage needs (estimation of hardware needs)	The VPH-HF orchestrates multiple hypomodels: the workflow manager and each hypomodel must be able to access a local file system where a sandbox is created. The needs in term of storage will be the same as the hypomodels to be executed.
URL (if applicable)	https://sourceforge.net/projects/vphhf/

7.3.4 Visualization toolkit

Resource	Mapping
Developer	BED
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	Follow the requirement from system architecture
Programming environment	Java
Communication technologies or protocols with other CHIC components	Follow the system architecture
Data types that it operates on	Clinical data and data from model simulation
Technical dependencies or requirements (libraries, tools)	A range of open sources, such as d3.js and 3.js
Computational needs (estimation of hardware needs)	From low to high depending on the demands
Storage needs (estimation of hardware needs)	From low to high depending on the demands
URL (if applicable)	

7.3.5 Image processing toolkit

Resource	Mapping
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Standalone application
Operating system	Windows
Programming environment	C++/Visual Basic .NET
Communication technologies or protocols with	It interacts indirectly via user interaction with the

other CHIC components	data repository. Currently, there is no direct communication with the hypermodeling framework planned.
Data types that it operates on	Medical images (.dcm or .mha format)
Technical dependencies or requirements (libraries, tools)	Dr. Eye and its plugin-environment serve as a basis for the toolkit. Moreover, it requires the Insight Registration and Segmentation ToolKit (ITK) library and .NET framework (3.5 or higher).
Computational needs (estimation of hardware needs)	No special requirements. A standard personal computer is sufficient to run the application.
Storage needs (estimation of hardware needs)	N/A
URL (if applicable)	http://biomodeling.ics.forth.gr/?page_id=8

7.4 Data Integration layer

7.4.1 Model repository

Resource	Mapping
Developer	ICCS
Software type (Web application, standalone application, service, library, framework etc.)	Web based (the User Interface part)
Operating system	Cross-platform
Programming environment	Programming language: Python, XML, Javascript Database management system: MySQL and/or MongoDB Web application framework: Django Web server: Apache HTTP server
Communication technologies or protocols with other CHIC components	REST services, XML, JSON
Data types that it operates on	Models (structured information and files)
Technical dependencies or requirements (libraries, tools)	The ones resulting from the programming environment (i.e. Python, MySQL, Apache etc.)
Computational needs (estimation of hardware needs)	N/A
Storage needs (estimation of hardware needs)	Estimated on 20GB.
URL (if applicable)	N/A

7.4.2 Clinical data repository

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

Developer	UBERN
Software type (Web application, standalone application, service, library, framework etc.)	Web based
Operating system	The Web application is running on a Microsoft Windows Server 2012 R2.
Programming environment	<p>Programming languages: C#, Javascript, HTML, CSS</p> <p>Database management system: Microsoft SQL Server 2012 R2.</p> <p>Web application framework: Microsoft ASP.NET Framework 4.5, ASP.NET MVC 4 and ASP.NET Web API</p> <p>Web server: Microsoft Internet Information Services (IIS)</p>
Communication technologies or protocols with other CHIC components	REST API, JSON
Data types that it operates on	Mostly clinical data (Dicom, MetalImage, Analyze, Niftii, HDF5)
Technical dependencies or requirements (libraries, tools)	<p>Insight Segmentation and Registration Toolkit (ITK) HDF5DotNet (http://hdf5.net/) DLL's must be copied to a folder which must be part of the PATH environment variable.</p> <p>Fuseki Server (http://jena.apache.org/documentation/serving_data/) is used to initiate SPARQL queries against an external resource (http://bio2rdf.org/FMA:9611).</p>
Computational needs (estimation of hardware needs)	3 nodes, 64 GB RAM, 8 cores (2 sockets and each has 4 cores)
Storage needs (estimation of hardware needs)	10-100 TB (depending on the number of patients)
URL (if applicable)	https://www.virtualskeleton.ch

7.4.3 In silico trial repository

Resource	Mapping
Developer	ICCS
Software type (Web application, standalone application, service, library, framework etc.)	Web based (the User Interface part)
Operating system	Cross-platform
Programming environment	<p>Programming language: Python, XML, Javascript</p> <p>Database management system: MySQL and/or MongoDB</p> <p>Web application framework: Django</p> <p>Web server: Apache HTTP server</p>

Communication technologies or protocols with other CHIC components	REST services, XML, JSON
Data types that it operates on	Input data, links to hypermodels and results of model execution (structured information and files)
Technical dependencies or requirements (libraries, tools)	The ones resulting from the programming environment (i.e. Python, MySQL, Apache etc.)
Computational needs (estimation of hardware needs)	N/A
Storage needs (estimation of hardware needs)	Depends on the number and volume of results. Initial estimation of 100GB.
URL (if applicable)	N/A

7.4.4 Metadata repository

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

7.4.5 OBTIMA data push services

Resource	Mapping
Developer	Fraunhofer IBMT / USAAR
Software type (Web application, standalone application, service, library, framework etc.)	Functional module within the ObTiMA web application
Operating system	N/A
Programming environment	Java

Communication technologies or protocols with other CHIC components	HTTP, REST
Data types that it operates on	Strings, (Typed) Numerical Values, Dates
Technical dependencies or requirements (libraries, tools)	Spring Framework
Computational needs (estimation of hardware needs)	N/A
Storage needs (estimation of hardware needs)	N/A
URL (if applicable)	http://obtima.org

7.4.6 Interoperable interfaces of model repository

Resource	Mapping
Developer	PHILIPS
Software type (Web application, standalone application, service, library, framework etc.)	Web Service
Operating system	Red Hat Enterprise Linux
Programming environment	Java
Communication technologies or protocols with other CHIC components	XML Web Services, Rest API (Model Repository, Clinical Data Repository)
Data types that it operates on	Models, Model input parameters
Technical dependencies or requirements (libraries, tools)	RedHat JBOSS Application Server, Script Execution Engines depending on models (URL required-not dependent), Model Repository Service, Clinical Data Repository, Query Engine
Computational needs (estimation of hardware needs)	N/A
Storage needs (estimation of hardware needs)	N/A
URL (if applicable)	-

7.5 Security layer

The security framework is comprised of many different components, each one with its one technical details and configuration, so, besides the technical deployment, we describe it in more detail in chapter 8.

7.5.1.1 IAM (Identity and Access Management site)

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

7.5.1.2 Idp (Identity Provider)

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

	Live: https://ciam-chic.custodix.com/idp
--	----------------------------------------------------------------------------------------------

7.5.1.3 STS (Secure Token Service)

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

7.5.1.4 PDP (Policy Decision Point)

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

URL (if applicable)	To be defined
----------------------------	---------------

7.5.1.5 PAP (Policy Administration Point)

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

7.5.1.6 PIP (Policy Information Point)

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

7.5.1.7 PEP (Policy Enforcement Point)

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

7.5.1.8 Audit Services

Resource	Mapping
Developer	Custodix
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	Web Service
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	50 GB (for application code and log files)
URL (if applicable)	To be defined

7.5.1.9 Security Gateway

Resource	Mapping
Developer	Custodix

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

7.5.1.10 De-identification

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

7.6 Semantics layer

7.6.1 Ontology-based semantic services

Resource	Mapping
Developer	UCL
Software type (Web application, standalone application, service, library, framework etc.)	Web services
Operating system	Cross-platform
Programming environment	JAVA
Communication technologies or protocols with other CHIC components	REST
Data types that it operates on	N/A
Technical dependencies or requirements (libraries, tools)	Metadata repository and Knowledge Base
Computational needs (estimation of hardware needs)	N/A
Storage needs (estimation of hardware needs)	Deployed as Tomcat WAR
URL (if applicable)	https://github.com/sarala/ricordo-owlkb-ws https://github.com/sarala/ricordo-rdfstore-ws

7.6.2 Folksonomy semantic services

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

	http://api.ccgv.org.uk/taggingservice/tags
--	-----------------------------------------------------------------------------------------------------

7.6.3 Knowledge base

Resource	Mapping
Developer	UCL
Software type (Web application, standalone application, service, library, framework etc.)	Web accessible database
Operating system	Cross-platform
Programming environment	Java
Communication technologies or protocols with other CHIC components	Ontology-based services (REST) and Java API
Data types that it operates on	Baseline option: OWL (Ontology Web Language)
Technical dependencies or requirements (libraries, tools)	Baseline option: OWLAPI, Reasoners
Computational needs (estimation of hardware needs)	Intensive (Prototype needs dedicated server and 64Gb RAM)
Storage needs (estimation of hardware needs)	Dedicated server
URL (if applicable)	N/A

7.7 Infrastructure layer

7.7.1 Private Cloud

Resource	Mapping
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	OpenStack cloud platform. Framework of tools.
Operating system	Linux
Programming environment	N/A (The software is used off-the-shelf, we are not developing it)
Communication technologies or protocols with other CHIC components	REST services: JSON/XML format, Command Line Interface (CLI) tools, Java SDK
Data types that it operates on	N/A
Technical dependencies or requirements (libraries, tools)	Operating System, network configuration, specific hardware resources.
Computational needs (estimation of hardware)	Many computer nodes (>5)

needs)	
Storage needs (estimation of hardware needs)	Many TBs of storage (>5)
URL (if applicable)	https://www.openstack.org/

8 Security view

Due to the nature of the handled data in CHIC, especially personal clinical data, and the stakeholder's principles and constraints that require a coherent and secure framework to handle this information, it is essential to describe in detail the security view of the architecture, in order to clearly present how the stakeholder's requirements will be met.

CHIC will not reinvent but instead base its security architecture on the security framework developed in past and ongoing EU projects such as ACGT, P-Medicine, EURECA, INTEGRATE. This security architecture makes use of modular re-usable components dealing with e.g. authentication, authorisation, auditing and de-identification. These components are based on widely used industry standards such as SAML, WS-*, XACML.

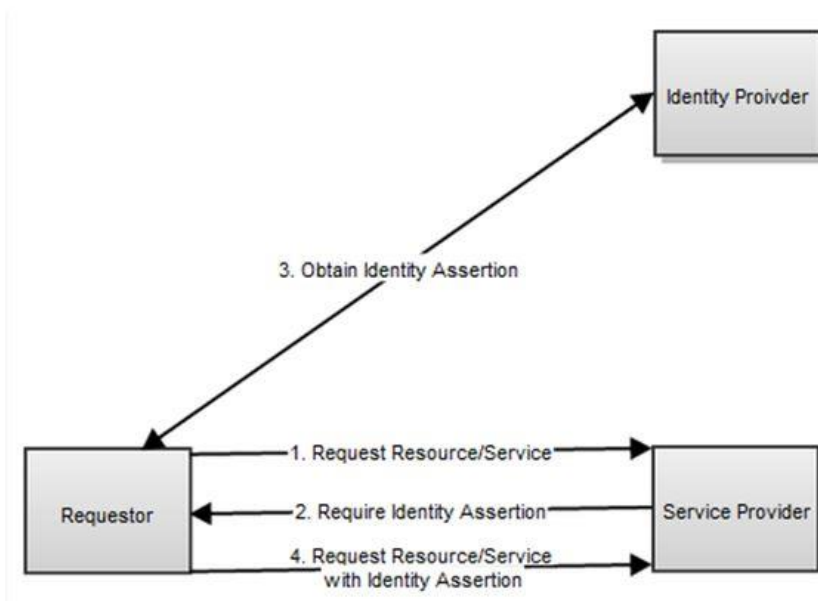
Typical components amongst others within security architectures are:

- An Identity Provider (IdP) which is responsible for authentication and provides identity assertion to other services called service providers.
- A service provider is an accessible service within an architecture that serves protected resources.
- A policy decision point is an entity that takes authorisation decisions.

This chapter aims at providing an initial description of the security architecture. Integration tutorials and an updated description will be available on M18 as deliverable "D5.2: Security Guidelines and initial version of security tools".

8.1 Brokered Authentication

Direct authentication of the user on each service (service provider) is not practical or advisable (e.g. authentication would have to be implemented on each service or a user would have to authenticate on each service separately). CHIC will therefore provide brokered authentication in which an authentication broker (such as an IdP) is responsible for authenticating the users and issuing identity tokens. The user can then use such identity tokens to access the project's services. To avoid that the user needs to provide his credentials each time he accesses a different service, the authentication broker can keep the authenticated session open. This results in new identity tokens being issued automatically (Single Sign-on) for each service the user accesses as long as the authentication broker's authenticated (SSO) session is still active.



Brokered authentication is initiated when a user (the requestor) requests a protected resource from a service provider (1). The Service provider replies that he needs an identity assertion (2) to be able to verify whether the user is allowed to access the requested resource. The requestor authenticates on a supported identity provider and requests an identity assertion (3). The requestor sends the assertion to the service provider as claim who he is and, if he gets access, receives the requested resource (4).

Enhanced clients (Non browser clients) such as desktops and server-side HTTP clients typically have the capability to directly contact identity providers without requiring a redirection from the service provider. Browsers on the other hand do not have this capability and will require a redirection from the service provider to the selected identity provider. This will have impact on the communication flow and thus different communication protocols are available depending on whether the client is a browser or not.

8.1.1 Identity Token Format

The format of the framework's assertion is defined by the SAML (Security Assertion Markup Language [7][8]) standard. SAML is an OASIS standard that defines signed identity assertions in an XML-based format. Such an identity assertion contains the user's authentication and authorisation information. Next to the format, SAML also defines the protocols for browsers and enhanced clients on how to exchange the assertions between identity and service providers. Next to SAML there are also other commonly used standards, such as WS-Federation, that define protocols on how to exchange a SAML assertion.

A SAML 2.0 assertion is structured in three parts:

1. Authentication statements which assert that the user did indeed authenticate with the Identity Provider at a given time using a particular method of authentication.
2. Attribute statements which assert that the user is associated with certain attributes.

3. Authorization decision statements which assert the a user is permitted to perform action on a resource with given evidence.

Example SAML Token

```

<saml2:Assertion ID="..." IssueInstant="..." Version="2.0" xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
  <saml2:Issuer>CHIC IdP</saml2:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    ...<!--XML Signature-->
  </ds:Signature>
  <saml2:Subject>
    <saml2:NameID>...<!--User's UUID--></saml2:NameID>
  </saml2:Subject>
  <saml2:Conditions NotBefore="..." NotOnOrAfter="...">
    <saml2:AudienceRestriction>
      <saml2:Audience>https://service-chic-test.custodix.com<!--Target service provider for which this token is valid--></saml2:Audience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>
  <saml2:AttributeStatement>
    <saml2:Attribute FriendlyName="mail" Name="urn:oid:0.9.2342.19200300.100.1.3">
      <saml2:AttributeValue><!--User's email address-->y</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="uuid" Name="urn:oid:0.9.2342.19200300.100.1.1">
      <saml2:AttributeValue><!--User's UUID--></saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="givenName" Name="urn:oid:2.5.4.42">
      <saml2:AttributeValue>Homer</saml2:AttributeValue>
    </saml2:Attribute>
    ...
  </saml2:AttributeStatement>
</saml2:Assertion>

```

The big advantages of SAML are:

1. Open commonly used & implemented standard
2. Exchange of identity information both on client-server as server-server communication.
3. Support for delegation by allowing the delegation chain to be inserted in the SAML token. A server can thus act through a SAML token in name of another user.

8.1.2 Brokered authentication for web services

8.1.2.1 SOAP Web Services

The WS-* specifications are the most commonly used standards to secure SOAP Web Services. WS-* is a collective noun for a variety of specifications associated with web services. Together these specifications form the basic framework for web services build on the first-generation standards of SOAP and WSDL.

WS-Security [10] defines how web service messages can be exchanged in a secure way by guarding the integrity, confidentiality and the sender's identity of the messages. To enforce this, WS-Security uses XML signature (for integrity), XML encryption (for confidentiality) and various security token formats, such as SAML, Kerberos, X.509 (for sender authentication), to provide end-to-end security.

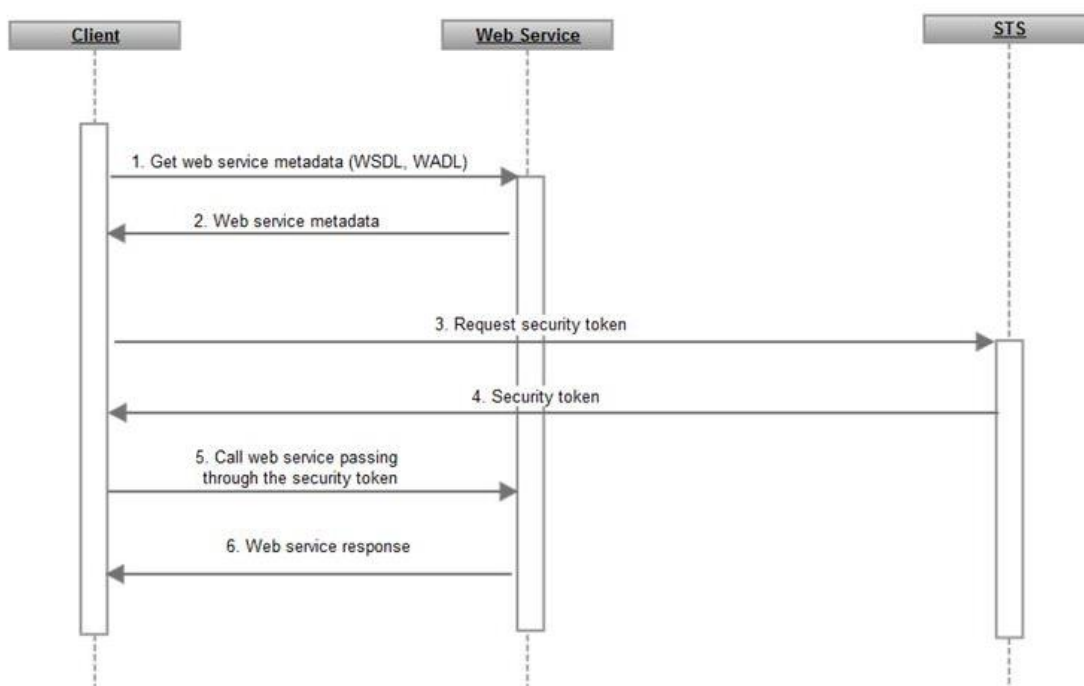
WS-Trust [11] is an extension of WS-Security providing methods for issuing, renewing and validating security tokens and providing ways to establish, assess the presence of, and broker trust relationships. Using the extensions defined in WS-Trust, applications can participate in secure communication designed to work within the web service framework. A main concept in WS-Trust is the Security Token Service (STS). This is a special web service that issues security tokens conforming to the WS-Security specification.

WS-SecureConversation [12] defines extensions that include session key derivation and security context establishment/sharing.

WS-SecurePolicy [13] defines XML based policies that are called security policy assertions. These policies allow web services to express their constraints and requirements. Policies can be used to drive development tools to generate code with certain capabilities, or may be used at runtime to negotiate the security aspects of web service communication. The intent is to provide enough information for compatibility and interoperability to be determined by web service participants.

WS-Federation [14] aims to simplify the development of federated services through cross-realm communication and management of federation services by re-using the WS-Trust Security Token Service model and protocol. WS-Federation defines two important profiles:

1. The Active Requestor Profile which defines how enhanced clients can request identity tokens from an identity provider (called Secure Token Service or STS in WS-*) and exchange it with a service provider.



2. The Passive Requestor Profile which defines a protocol for browsers to exchange identity token between identity and service providers. As the Passive Requestor Profile is usually not used for web service calls but rather for Web Site (browser) authentication, it is not further described in this chapter.

A SOAP client will thus request identity tokens and authenticate on a Secure Token Service (STS) as defined in WS-Trust. An STS is a SOAP Web Service with the following operations (non-exclusive):

1. **Issue**: this operation issues a new security token based on the credential provided or proven in the request. The operation accepts a request security token (RST) and returns a request security token response (RSTR).

```

<wst:RequestSecurityToken Context="..." xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestType>...</wst:RequestType>
  <wst:SecondaryParameters>...</wst:SecondaryParameters> ... <wsp:AppliesTo>...</wsp:AppliesTo>
  <wst:Claims Dialect="...">...</wst:Claims>
  <wst:Entropy>
    <wst:BinarySecret>...</wst:BinarySecret>
  </wst:Entropy>
  <wst:Lifetime>
    <wsu:Created>...</wsu:Created>
    <wsu:Expires>...</wsu:Expires>
  </wst:Lifetime>
</wst:RequestSecurityToken>

```

```

<wst:RequestSecurityTokenResponse Context="..." xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestedSecurityToken>...</wst:RequestedSecurityToken>
</wst:RequestSecurityTokenResponse>

```

2. **Cancel:** this operation cancels a token so that it cannot be used anymore when it is no longer needed. After cancellation the STS will not renew or validate the token anymore.

```

<wst:RequestSecurityToken>
  <wst:RequestType> http://docs.oasis-open.org/ws-sx/ws-trust/200512/Cancel </wst:RequestType>
  <wst:CancelTarget> ... </wst:CancelTarget>
</wst:RequestSecurityToken>

```

```

<wst:RequestSecurityTokenResponse>
  <wst:RequestedTokenCancelled/>
</wst:RequestSecurityTokenResponse>

```

3. **Renew:** a previously issued possible expired token is presented and the same token is returned with new expiration semantics. The requestor must either prove authorized use of the token or be trusted by STS to issue third-party renewal requests.

```

<wst:RequestSecurityToken xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestType> http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew </wst:RequestType>
  <wst:RenewTarget> ... reference to previously issued token ... </wst:RenewTarget>
  <wst:AllowPostdating/>
  <wst:Renewing Allow="..." OK="..."/>
</wst:RequestSecurityToken>

```

```

<wst:RequestSecurityTokenResponse xmlns:wst="...">
  <wst:TokenType> ... </wst:TokenType>
  <wst:RequestedSecurityToken>...</wst:RequestedSecurityToken>
  <wst:Lifetime>...</wst:Lifetime>
</wst:RequestSecurityTokenResponse>

```

4. **Validate:** this operation evaluates the specified token. The result can be a status, a new token or both.

```

<wst:RequestSecurityToken xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestType>...</wst:RequestType>
  <wst:ValidateTarget>... </wst:ValidateTarget>
</wst:RequestSecurityToken>

<wst:RequestSecurityTokenResponse xmlns:wst="..." >
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestedSecurityToken>...</wst:RequestedSecurityToken>
  <wst:Status>
    <wst:Code>...</wst:Code>
    <wst:Reason>...</wst:Reason>
  </wst:Status>
</wst:RequestSecurityTokenResponse>

```

WS-Trust does not specify or limit the format of the issued token. Within CHIC SAML 2.0 tokens will be issued by the STS. The WS-Security profile "SAML Token Profile 1.1." defines how a SAML token can be passed to SOAP web services through SOAP headers.

```

<!--SAML token issued by STS for "End User" targetted to "Service A"-->
<saml2:Assertion xmlns:saml2="..."
ID="uid-ec80226b-0a2e-43bd-a98b-54b071407edd" IssueInstant="2012-05-09T09:43:00.292Z" Version="2.0">
  <saml2:Issuer>STS</saml2:Issuer>
  <saml2:Subject>
    <saml2:NameID>End User</saml2:NameID>
  </saml2:Subject>
  <saml2:Conditions NotBefore="..." NotOnOrAfter="...">
    <saml2:AudienceRestriction>
      <saml2:Audience>http://serviceA.com/</saml2:Audience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>
  <saml2:AttributeStatement>
    ...
  </saml2:AttributeStatement>
</saml2:Assertion>

```

8.1.2.2 REST Web Services

As REST Web Services are also web services, a natural solutions to secure REST Web Services would be to look at the WS-* specifications. A Secure Token Service (STS) can then be used by REST clients to request a security token.

Before calling the REST service a client sends a SOAP request containing an RST (RequestSecurityToken) to the STS. The STS then returns the identity assertion as a SAML token, embedded in a RSTR (RequestSecurityTokenResponse). In SOAP this SAML token is passed to the calling service through a SOAP header. This is not possible in REST though as there are no SOAP headers. In REST, the HTTP headers and request line are part of the message. An HTTP header is therefore the REST equivalent of a SOAP header. A SAML token can then be passed to a REST service through the HTTP authorization header. The authorization header value should be formatted as follows: "SAML <Base 64 encoded compressed SAML token>". The SAML token is hereby compressed with the zlib compression algorithm. This ensures that the token fits in the typical 4kb header size limit.

To secure a REST service the same SOAP STS service is used as with SOAP services as defined in WS-Trust. REST clients are responsible for retrieving a SAML token from the STS and then passing it through a HTTP authorization header to the REST service. The client somehow needs to find out what

the REST service's security policies are (i.e what STS to call). In WS-* a web service publicises its security policies through WS-Policy annotations on the service's WSDL metadata. Such a WSDL file annotated with security policies (by using the WSDL HTTP binding) could actually also be used for REST services. As REST is resource based while SOAP and the WSDL metadata format are action based, this is not an exact match. There is a metadata format, called WADL, which is specifically designed to describe a REST service. No known specification that defines how to add security policies to a WADL metadata file, is available yet. Therefore research should be done on whether WS-Policy metadata annotations can be used to add security annotations to a WADL metadata file.

A drawback of using a SOAP STS is that clients are forced to use SOAP to request security tokens. Therefore the secure token service could be better integrated into REST services by REST-ifying it.

A next step is to guarantee the integrity and confidentiality of a REST message. Signing and possibly encrypting the message body is not sufficient as the HTTP request line and headers are also part of a REST message. Although no standards exist yet that define how to do message level security for REST services, big service providers such as Amazon already implement it.

8.1.3 Web Site (browser) Authentication

There are three big standards that define SSO and brokered authentication on the web.

1. OpenID
 1. The SAML Web Browser SSO Profile
 2. The WS-Federation Passive Requestor Profile

OpenID [8] is a lightweight HTTP-based profile for Single Sign-On and attribute exchange. OpenID is mainly oriented towards browsers, not really towards REST and SOAP web services called by enhanced clients such as server-side HTTP clients. A record standardisation effort "OpenID Connect" tries to solve this problem in OpenID Connect 1.0 (2014) [9]. OpenID Connect came to late though to be considered for this Security Framework. Within the course of CHIC it should be watched closely whether it gains any track and could as a result later on in the project be included in the framework. OpenID does not use SAML as token format. As SAML is the token format used to protect Web Services, it would be preferable stay with a SAML based standard for protecting web sites.

The **WS-Federation Passive Requestor Profile** [14] defines how passive requestors such as a browser can obtain a security token, this in contrast to the WS-Federation Active Requestor Profile as explained in 8.1.2.1. As CHIC already uses WS-Federation for active requestors it could also be a good choice for Web Browser Authentication. The WS-Federation Passive Requestor Profile is mainly implemented by Microsoft though and not as much in other frameworks.

The **SAML Web Browser SSO Profile** is part of the SAML (Security Assertion Markup Language) 2.0 standard [7][8]. SAML, as explained in 8.1.1, is an XML-based protocol making it possible to exchange authentication and authorization data within or between security domains. The exchange is done by using signed assertions containing identity information. The SAML standard mainly focusses on how to do this exchange through a browser (e.g. the SAML Web Browser Single Sign-On Profile).

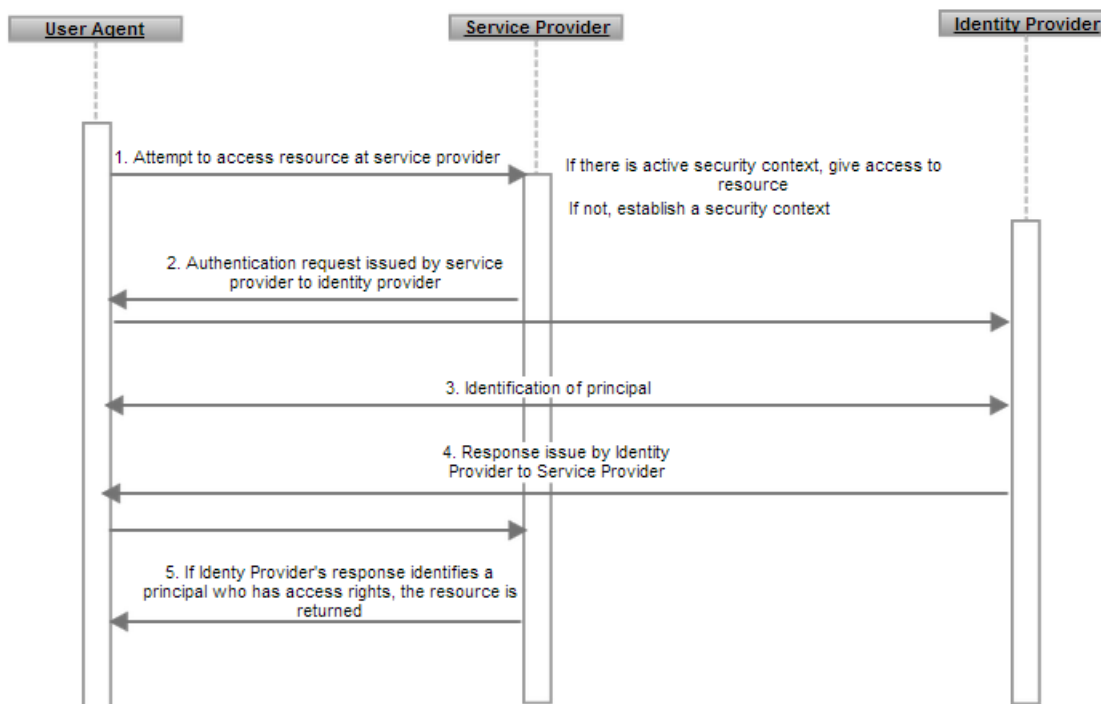


Figure 16 SAML Browser SSO Profile

The Web Browser SSO Profile is initiated by an end user who visits a protected web site, also called a relying party or Service Provider (SP). The SP redirects the user to the assertion provider (also called Identity Provider (IdP)) passing through an authentication request. The IdP will request the user to authenticate himself and upon successful authentication issue an identity assertion for the user containing all information on the user needed by the SP to authenticate and authorize the user. The assertion is send back to the SP that will use it to determine whether the user is allowed to access the requested resource.

CHIC makes use of the SAML Web Browser SSO Profile for browser based authentication.

8.2 Security Components

8.2.1 Authentication Components

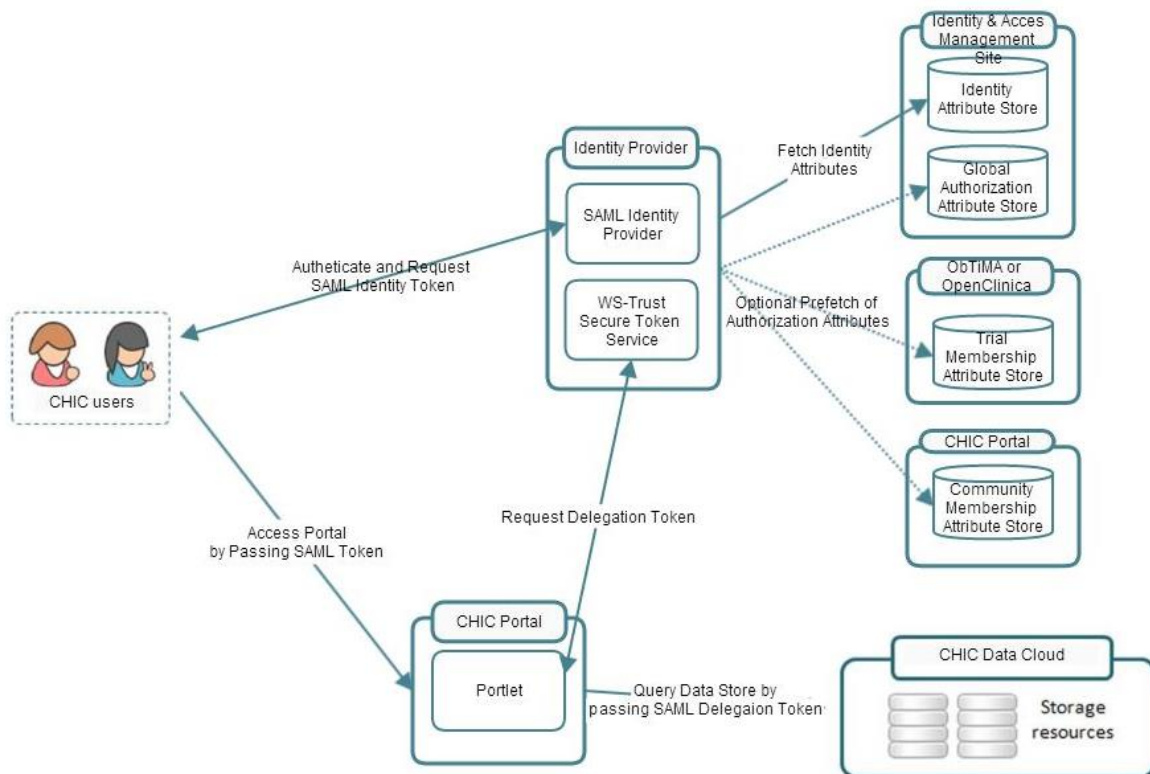


Figure 17 CHIC Security Components (Authentication)

Figure 17 shows the interaction between the different security components as explained in 5.5 and 8.

A user accesses CHIC by browsing to the CHIC portal. Through the CHIC portal the user can initiate the login procedure redirecting the user to the CHIC Identity Provider (IdP) passing through a SAML authentication request. The IdP then requests the user to authenticate. Upon successful authentication, the IdP will query all backend attribute providers (also called PIPs, Policy Information Points in the authorization context) for information on the authenticated user. The IdP issues a SAML identity assertion, including the queried information, and redirects the user back to the portal where the assertion is validated.

A rendered portlet is required to obtain a token from the CHIC Secure Token Service (STS) when it needs to call a backend service. Such a token should identify the original end user performing the action and the portlet or portal as intermediate service. This is also called a delegation token. A portlet obtains a delegation token by calling the STS passing through the end user’s identity assertion (issued by IdP as explained above). Upon successful authentication and after validation that the given portlet is indeed allowed to delegate in name of the user, a delegation token is issued to the portlet. By using the delegation token the portlet can call the backend service.

8.2.2 Authorization Components

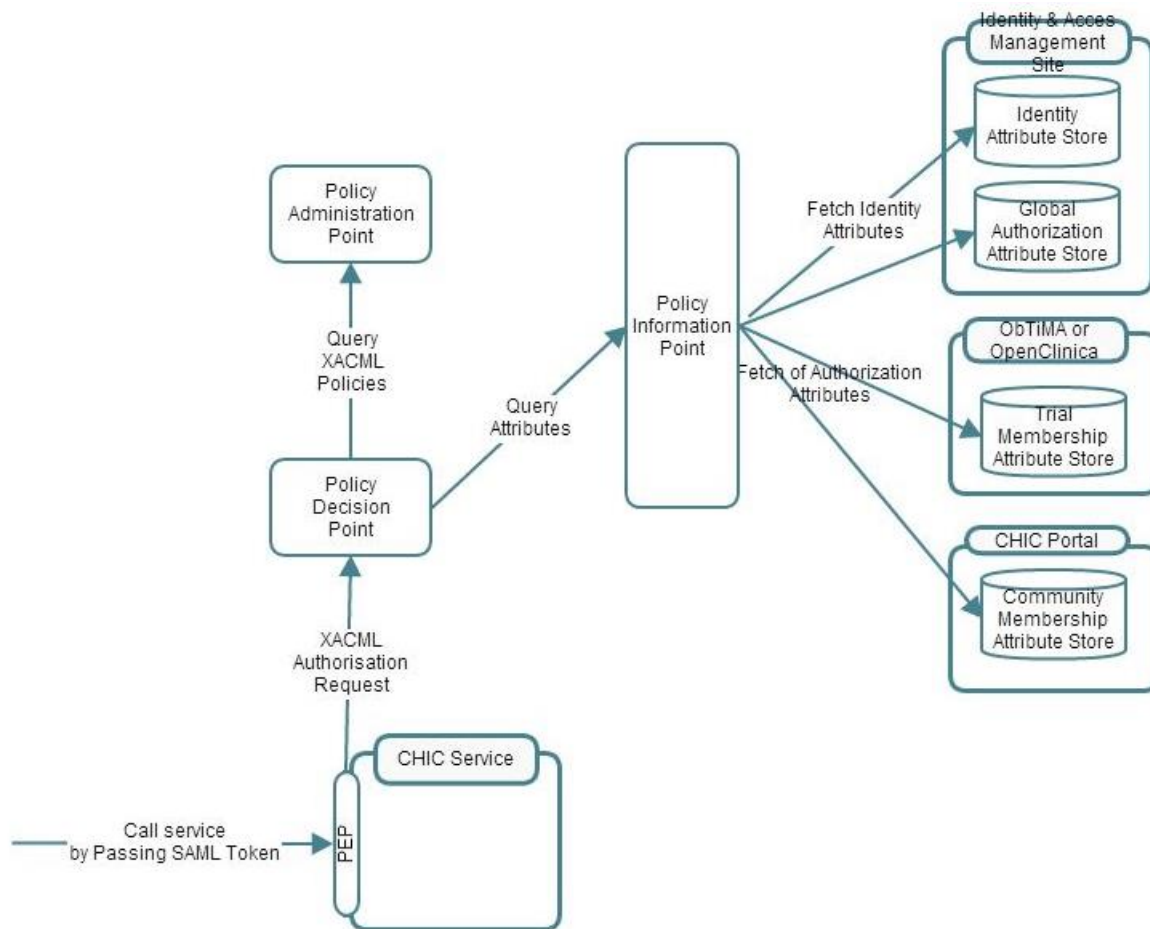


Figure 18 CHIC Security Components (Authorization)

Figure 18 shows the interaction between the different authorization components as explained in 5.5 and 8.

A Policy Enforcement Point (PEP) integrates a CHIC service in the authorization framework. When a service needs an authorization decision, the PEP will create an authorization request, authenticate on the CHIC Policy Decision Point (PDP) and parse the returned authorization decision. The PEP will populate the authorization request with all information available within the service that might be relevant concerning the subject performing the action, the resource on which the action is performed and the actual action.

Upon receiving an authorization request the PDP will query the Policy Administration Point (PAP) for relevant policies to be evaluated. During policy evaluation the PDP might need some additional information on the subject or resource not available in the authorization request. Such additional information is obtained by querying the CHIC Policy Information Points such as the CHIC user repository (IAM), the CHIC portal, clinical trial software used for trial memberships, the CHIC data repository.

8.3 Security Vocabulary

This section provides a preliminary definition of the security vocabulary. It will be further extended in the next security and architectural deliverables.

Attribute Name	URN	Description
UUID	urn:custodix:ciam:1.0:principal:uuid	The federated ID of the user
Alternative UUID	urn:custodix:ciam:1.0:principal:uuid:alias	The alternative federated ID's of the user (after account merging)
Uid (userid)	urn:oid:0.9.2342.19200300.100.1.1	Contains a human readable identifier: the username of the user.
Domain uuid	urn:custodix:ciam:1.0:domain:uuid	The security domains uuid this user is member of.
Domain name (unique)	urn:custodix:ciam:1.0:domain:name	The security domains name this user is member of.
Email	urn:custodix:ciam:1.0:principal:email urn:oid:0.9.2342.19200300.100.1.3	Registered email address
Title	urn:oid:2.5.4.12	Title of the user
Given name	urn:oid:2.5.4.42	First name of the user
Name	urn:oid:2.5.4.4	Last name of the user
Organisation name	urn:oid:2.5.4.10	Name of the organisation the user is part of
Role	urn:oid:2.5.6.8	CHIC role

8.4 Security Framework Guidelines

8.4.1 Transport Layer Security

All web service communication within CHIC MUST go over a secured HTTP connection by using the HTTP SSL/TLS protocol as specified in [15],[16],[17],[18]

- SSL version 1.0 or 2.0 MUST NOT be used.
- TLS 1.1/1.2 are RECOMMENDED
- SSL version 3.0 or TLS 1.0/1.1/1.2 MUST be used.

TLS and SSL are cryptographic protocols aimed to provide communication security over the Internet. They use X.509 certificates (asymmetric cryptography) to assure the counterparty with whom they are communicating, and to exchange a symmetric key. This allows for data/message confidentiality and message authentication codes for message integrity and as a by-product message authentication. In CHIC we limit the use of these protocols to SSL version 3.0 or TLS version 1.0/1.1/1.2. SSL version 1.0 was never made public and SSL version 2.0 is considered insecure (vulnerable to attacks). Transport Layer Security (TLS) MUST be enabled to protect web service

endpoints and secure communications between CHIC service providers. SSL/TLS only provide protection between adjacent HTTP Nodes. It therefore does not protect HTTP SOAP or REST messages when the message path contains intermediaries. SSL/TLS should thus always be in conjunction with message level security (e.g. encryption of SOAP and REST messages) when the message path contains intermediaries or when these security functions are required to be performed independently of the connection.

Administrators MUST follow the NHS [19] guidelines for choosing the SSL/TLS cryptographic algorithms.

In SSL and TLS, choices of algorithms are expressed as ciphersuites. The Approved Cryptographic Algorithms Good Practice Guideline specifies the ciphersuites that are required, recommended, discouraged and prohibited. The ciphersuites defined in the SSL and TLS specifications that use anonymous Diffie-Hellman (i.e. those that have DH_anon in their symbolic name) are vulnerable to man-in-the-middle attacks and MUST NOT be used. It is also RECOMMENDED that ciphersuites that include MD5 (i.e. those that have MD5 in their symbolic name) be avoided, due to known security weaknesses of the MD5 algorithm. It is RECOMMENDED that ciphersuites that use 40 or 56 bit keys be avoided, due to their relative ease of compromise through brute-force attack.

Following Ciphersuites are RECOMMENDED for secure communication between CHIC web components:

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA

The specified algorithm suites are considered to be widely-implemented, secure and interoperable.

The TLS/SSL authentication keys MUST be signed by a trusted certificate authority.

- Authentication keys MUST be equal or higher than 2048 bit

Self-signed keys are not permitted in the CHIC production environment. They should be signed by a Central Authority (CA) like Verisign, Comodo Group, Go Daddy, GlobalSign, etc. Keys less than 2048 bit are not considered secure anymore.

8.4.2 Message layer Security

Guidelines for Message Layer Security will be provided in “D5.2: Security Guidelines and initial version of security tools” together with the integration tutorials.

8.4.3 Authentication Protocols

Guidelines for the various authentication protocols (active and passive) will be provided in “D5.2: Security Guidelines and initial version of security tools” together with the integration tutorials.

9 References

- [1] IEEE 1471 - ISO/IEC 42010:2007 standard. <http://www.iso-architecture.org/ieee-1471>
- [2] Kruchten, Philippe. "Architectural Blueprints - The 4+1 View Model of Software Architecture." IEEE Software, 12(6):42–50, November 1995.
- [3] Reference Model of Open Distributed Processing. ITU-T Rec. X.901-X.904 / ISO/IEC 10746, <http://www.rm-odp.net/>
- [4] Zachman framework. <http://www.zachman.com/about-the-zachman-framework>
- [5] Department of Defense Architecture Framework (DoDAF). <http://dodcio.defense.gov/dodaf20.aspx>
- [6] Rozanski, Nick, and Woods, Eoin. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2nd Edition. Addison Wesley, 2011.
- [7] OASIS, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>
- [8] OpenID Foundation, OpenID Authentication, version 2.0, 2007, http://openid.net/specs/openid-authentication-2_0.html
- [9] OpenID Foundation, OpenID Connect Core, version 1.0, 2014, http://openid.net/specs/openid-connect-core-1_0.html
- [10] OASIS, WS-Security specification, version 1.1, 2004, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
- [11] OASIS, WS-Trust specification, version 1.4, 2009, <http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.pdf>
- [12] OASIS, WS-SecureConversation specification, version 1.4, 2009, <http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.pdf>
- [13] OASIS, WS-SecurePolicy, version 1.3, 2009, <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.pdf>
- [14] OASIS, WS-Federation specification, version 1.2 , 2009, <http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.pdf>
- [15] IETF, rfc 6101: The Secure Sockets Layer (SSL) Protocol, Version 3.0, 2011, <http://tools.ietf.org/html/rfc6101>
- [16] IETF, rfc 2246: The Transport Layer (TLS) Protocol, Version 1.0, 1999, <http://tools.ietf.org/html/rfc2246>
- [17] IETF, rfc 4346: The Transport Layer (TLS) Protocol, Version 1.1, 2006, <http://tools.ietf.org/html/rfc4346>
- [18] IETF, rfc 5246: The Transport Layer (TLS) Protocol, Version 1.2, 2011, <http://tools.ietf.org/html/rfc5246>
- [19] NHS, Approved Cryptographic Algorithms Good Practice Guideline, version 3.0, 2012, <http://systems.hscic.gov.uk/infogov/security/infrasec/gpg/acs.pdf>

Appendix 1 – Abbreviations and acronyms

<i>AD</i>	Architectural Description
<i>API</i>	Application Programming Interface
<i>CLI</i>	Command-Line Interface
<i>CRF</i>	Case Report Form
<i>CSV</i>	Comma Separated Values
<i>DICOM</i>	Digital Imaging and Communications in Medicine
<i>EC</i>	European Commission
<i>GUID</i>	Globally Unique Identifier
<i>HDOT</i>	Health Data Ontology Trunk
<i>HIS</i>	Hospital Information System
<i>HTTP</i>	Hypertext Transfer Protocol
<i>IAM</i>	Identity and Access Management
<i>IdP</i>	Identity Provider
<i>IPR</i>	Intellectual Property Rights
<i>IT</i>	Information Technology
<i>ITK</i>	Insight Segmentation and Registration Toolkit
<i>JSON</i>	JavaScript Object Notation
<i>LDAP</i>	Lightweight Directory Access Protocol
<i>MAF3</i>	Multimod Application Framework 3
<i>NHS</i>	National Health Service
<i>PAP</i>	Policy Administration Point
<i>PDP</i>	Policy Decision Point

<i>PEP</i>	Policy Enforcement Point
<i>PIP</i>	Policy Information Point
<i>REST</i>	Representational State Transfer
<i>RM-ODP</i>	Reference Model of Open Distributed Processing
<i>RPC</i>	Remote Procedure Call
<i>RST</i>	Request Security Token
<i>RSTR</i>	Request Security Token Response
<i>SAML</i>	Security Assertion Markup Language
<i>SOAP</i>	Simple Object Access Protocol
<i>SSL</i>	Secure Sockets Layer
<i>SSO</i>	Single Sign-On
<i>STS</i>	Secure Token Service
<i>TLS</i>	Transport Layer Security
<i>UUID</i>	Universally Unique Identifier
<i>VPH</i>	Virtual Physiological Human
<i>VPH-HF</i>	VPH Hypermodelling Framework
<i>WSDL</i>	Web Service Description Language
<i>XACML</i>	eXtensible Access Control Markup Language
<i>XML</i>	Extensible Markup Language