



## **Deliverable No. 7.2**

### **First release hypermodelling framework deployed on test nodes**

Grant Agreement No.: 600841  
Deliverable No.: D7.2  
Deliverable Name: First release hypermodelling framework deployed on test nodes  
Contractual Submission Date: 31/03/2015  
Actual Submission Date: 30/05/2015

Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	<b>X</b>
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.:	D7.2
Document name:	First release hypermodelling framework deployed on test nodes
Nature (R, P, D, O) <sup>1</sup>	P
Dissemination Level (PU, PP, RE, CO) <sup>2</sup>	RE
Version:	1
Actual Submission Date:	30/5/2015
Editor: Institution: E-Mail:	Simone Bnà CINECA simone.bna@cineca.it

#### ABSTRACT:

This deliverable describes the first release of the hypermodelling framework technology. The hypermodelling framework design, which started from the VPHOP project original concept, has been highly refactored to be generalised and compatible to the CHIC needs. Even if further developments are foreseen in the next project years, all hypermodelling framework components are now in place and the connection with the other CHIC service has been analysed and when possible already implemented in a prototype version. In this document, the hypermodelling framework is described from an architectural point of view and for each VPH-HF component implementation details and APIs are provided.

#### KEYWORD LIST:

Hypermodelling framework, architecture, APIs, hypermodel, hypomodel, generic stub, model wrapper, director, workflow management service, authentication service, storage management service, registry service, annotation services,

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

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

<sup>1</sup> R=Report, P=Prototype, D=Demonstrator, O=Other

<sup>2</sup> 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)

<b>MODIFICATION CONTROL</b>			
<b>Version</b>	<b>Date</b>	<b>Status</b>	<b>Author</b>
1.0	15/05/2015	Draft	Simone Bnà, CINECA
2.0	25/05/15	Revision	N. Touser, G. Stamatakis
3.0	26/05/15	Revision	D. Tartarini, K. Duan, D. Walker
4.0	30/05/2015	Revision	M. Viceconti
5.0	05/06/2015	Revision (Finalized)	Simone Bnà, CINECA

#### List of contributors

- Debora Testi, CINECA
- Simone Bnà, CINECA
- Samuel Alexander, UCL
- Daniele Tartarini, USFD
- Kewei Duan, USFD
- Dawn Walker, USFD
- Marco Viceconti, USFD
- Feng Dong, BED
- Stelios G. Sfakianakis, FORTH
- Nikolaos Touser, ICCS
- Eleni Kolokotroni
- Eleftherios Ouzounoglou
- Nikolaos Christodoulou
- Georgios Stamatakis, ICCS
- Dimitra Dionysiou, ICCS

## Contents

CONTENTS .....	4
1 EXECUTIVE SUMMARY .....	5
2 INTRODUCTION .....	6
2.1 PURPOSE OF THIS DOCUMENT .....	6
2.2 DEFINITIONS .....	6
3 HYPERMODELLING FRAMEWORK ARCHITECTURE AND DEPLOYMENT SCENARIOS .....	10
3.1 HYPERMODELLING FRAMEWORK .....	10
3.2 DEPLOYMENT SCENARIOS .....	11
3.3 NEW VPH-HF ARCHITECTURE .....	13
3.4 VPH-HF SUPPORT FOR STRONGLY COUPLED HYPERMODELS .....	15
4 CHIC USER SCENARIO (A DAY IN THE LIFE OF A MODEL) .....	19
4.1 UNDER THE HOOD .....	20
5 GENERIC STUB REVISION .....	21
5.1 CHIC GENERIC STUB .....	21
5.2 ANALYSIS OF THE GENERIC METAMODEL STUB .....	21
5.3 ANALYSIS OF THE GENERIC COMPUTATIONAL MODEL STUB .....	21
6 HYPERMODELLING FRAMEWORK IMPLEMENTATION .....	24
6.1 CHIC-HF CLIENT – THE USER INTERFACE .....	24
6.2 CHIC-HF SERVER .....	29
6.3 CHIC-HF ANNOTATION AND SEARCH SERVICES .....	39
6.4 TEST NODES DEPLOYMENT .....	56
7 CONCLUSION .....	59
8 REFERENCES .....	60

## 1 Executive Summary

During the CHIC project second year, Work Package 7 (WP7) has been focused on the release of the first prototype of the hypermodelling framework (VPH-HF) and its deployment. Two different abstraction layers are identified in the deployment of the framework: the VPH-HF Orchestration Layer and the Computational/Model layer. The former comprises all the VPH-HF components, libraries and tools to enable the orchestration of workflows, staging and retrieval of data. The framework components are designed to be modular and exposing a web service interface for reusability. In particular the following components have been developed and deployed:

- CHIC-HF client (web interface)
- CHIC-HF server comprising all the components of the orchestration and computational/model layer (authentication, director, registry, storage management service, metadata repository)
- CHIC-HF annotation and search services (metadata schema, free tags annotation, semantic metadata)

The Computational/Model layer comprises the computational instances of the WP6 partners' models with their dependencies, tools and libraries, required for execution.

The two layers, Orchestration and Computational/Model, can be easily decoupled and deployed on different machines. This flexibility allows the deployment of the computational/model layer on HPC facilities where models have specific needs.

The framework has been deployed on test nodes (CINECA and USFD). It is tested using hypomodels and hypermodels developed in WP6 .

## 2 Introduction

### 2.1 Purpose of this document

During the second year of the CHIC project, WP7 main goal was the release of the first prototype of the hypermodelling framework, including development, deployment, and testing on the hypo- and hyper-models made available by WP6.

This document will provide a description of the first release of the hypermodelling framework prototype and in particular addresses the following aspects:

- Refactoring of the hypermodelling framework
- Revision of the CHIC generic stub as a consequence of the architectural update
- Description of the framework use from a modeller point of view
- Description of the hypermodelling framework components in terms of functionalities provided, APIs and implementation choices

Many CHIC technical WPs were in parallel developing prototypes for their respective tools; while an integration of the services was not foreseen at the end of the second year, a positive collaborative work among the different teams has allowed providing already in this document a description of the integration path. In some cases where early prototypes were available a preliminary integration was also performed so to already spot potential technical issues or to define requirements for next framework releases.

In the conclusion session, the next steps for the development and completion of the hypermodelling framework implementation will be provided.

### 2.2 Definitions

An agreed set of definitions has been put into place in the internal document D7.101 and deliverable D7.1 in order to increase the efficacy in the communication and reduce misinterpretations. In this section, the relevant definitions are reported so to allow the reader to have an early understanding of the terms that will be then used in the rest of the document.

Term	Definition
<b>Adaptor or Relational Model</b>	We define as an adaptor (or relational model) a piece of software that adapts/transforms the predicted output of a hypomodel so as to enable its provision as input to another hypomodel.
<b>Orchestration or Choreography</b>	In the context of Service-Oriented Architectures (SOA) the two terms indicate the coordinated execution of multiple services. Different authors use the two terms differently [12]. However, in our cases where the execution of the services is coordinate centrally (autocratic), the term <i>orchestration</i> is preferred.
<b>Component model</b>	An alias for hypomodel.

<b>Composite model</b>	An alias for hypermodel.
<b>Computer model</b>	We define a computer model as a computer program that implements a scientific model, so that, when executed according to a given set of control instructions (control inputs), it computes certain quantities (data outputs) on the basis of a set of initial quantities (data inputs) and a set of execution logs (control outputs).
<b>Data</b>	<p>We define data as factual information, whether observed or predicted.</p> <ul style="list-style-type: none"> <li>• Observed: generated through observation, measurement, etc.</li> <li>• Predicted: generated through speculative reasoning informed by existing knowledge.</li> </ul>
<b>Digital resource</b>	May refer to either Data or Models.
<b>Elementary model</b>	We define as an elementary model a model that, from a particular standpoint, appears (subjectively) not amenable to decomposition into meaningful simpler models, mainly because the current scientific knowledge and technological status cannot support such a decomposition. It is almost certain that future scientific discoveries and technological advantages will allow models that presently are considered as elementary models to be further decomposed into new elementary models.
<b>Folksonomy</b>	In information science, a folksonomy is a system of classification derived from the practice and method of collaboratively creating and translating tags to annotate and categorize content. This practice is also known as collaborative tagging, social classification, social indexing, and social tagging.
<b>Generic Stub</b>	The “Component Model Generic Stub”, or Generic Stub for short, is a template that all the models that participate in the CHIC system should comply with in order to be effectively integrated with the rest of the platform.
<b>Hypermodel</b>	We define as a hypermodel a model that emerges from the composition and orchestration of multiple hypomodels, each one of which is capable of simulating a specific entity or phenomenon. The hypermodel can simulate an entity or phenomenon that may be more complex than the ones simulated by each separate simpler model.
<b>Hypermodel editor</b>	A portal that provides appropriate tools for the design of new hypermodels.

<b>Hypermodelling</b>	We define as hypermodelling the process of developing hypermodels.
<b>Hypermodelling Framework</b>	We define as hypermodelling framework the software layer that facilitates the development of hypermodels and allows their execution. The various models and the supplementary components (adaptors, mergers, linkers, etc.) are not considered to be part of the framework themselves, but they are retrieved or invoked upon request.
<b>Hypermodelling infrastructure</b>	We define as hypermodelling infrastructure the set of technological components that facilitates the development of hypermodels and allows their execution. This includes both software and hardware.
<b>Hypomodel</b>	We define as a hypomodel a model that captures the existing knowledge about a portion of the process, typically at a characteristic space-time scale, and simulates a simpler entity or phenomenon compared to a model or a hypermodel.
<b>Integrative model</b>	An alias for hypermodel.
<b>Linker</b>	We define as a linker a piece of software inserted between the outputs of two hypomodels and the input of a third hypomodel so as to allow the merged and adapted output of the two hypomodels to be fed into the third hypomodel. The linker consists generally of adaptors and a merger. The linkers and the participating hypomodels are the constructive elements of a hypermodel.
<b>Merger</b>	We define as a merger a piece of software that merges the adequately adapted outputs of two hypomodels.
<b>Metadata</b>	<p>Metadata is “data about data”. The term is ambiguous as it used for two fundamentally different concepts (types) [11].</p> <ul style="list-style-type: none"> <li>• Structural metadata refers to the design and specification of data structures and is more properly called “data about the containers of data”.</li> <li>• Descriptive metadata, on the other hand, refers to individual instances of application data, i.e. the data content.</li> </ul>
<b>Meta-hypermodel</b>	The semantic description of a hypermodel.
<b>Meta-hypomodel</b>	The semantic description of a hypomodel.



<b>Meta-model</b>	We define as a meta-model the semantic description of a model. The meta-model can be considered as an abstract representation of a model, as it highlights certain properties of the model itself. Consequently, a meta-hypomodel and a meta-hypermodel is the semantic description of a hypomodel and a hypermodel respectively.
<b>Model</b>	In the CHIC project we use the general term “model” in order to define a mathematical or computational construct incorporating speculative information that represents the existing knowledge. Computational implementation of such a model is capable of virtually regenerating an entity or phenomenon.
<b>Ontology</b>	In computer science and information science, an ontology formally represents knowledge as a set of concepts within a domain, using a shared vocabulary to denote the types, properties and interrelationships of those concepts.
<b>Scientific Model</b>	We can define a scientific model as a finalized cognitive construct of finite complexity that idealizes an infinitely complex portion of reality through idealizations that contribute to the achievement of knowledge on that portion of reality that is objective, shareable, reliable and verifiable [10].
<b>Wrapper</b>	Software layer that “wraps” an existing model implementation and provides the integration layer so that the model can become one hypomodel of a hypermodel. This implies translation of the control flow, and management of the data flow.

**Table 1 – CHIC project definitions**

## 3 Hypermodelling framework architecture and deployment scenarios

### 3.1 Hypermodelling framework

The VPH Hypermodelling Framework (VPH-HF) was designed and developed as a technology with the aim to provide services and tools to allow:

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

The general use case that drove our initial development was that where a user, after appropriate authentication, has uploaded to a central repository patient-specific data that have been pre-processed so as to be suitable inputs for predictive models. From a web interface the user select an existing hypermodel, or compose available hypomodels into a new hypermodel, and then request its execution with a certain input set. From the same interface the user monitors the execution of the hypermodel, and can retrieve the hypermodels output set, which has also been stored centrally.

The starting point for VPH-HF was the implementation carried out during the VPHOP project and described in D7.1 [1] and presented in the scheme below (Figure 1). This was partially composed by C++ and python code for the components implementation.

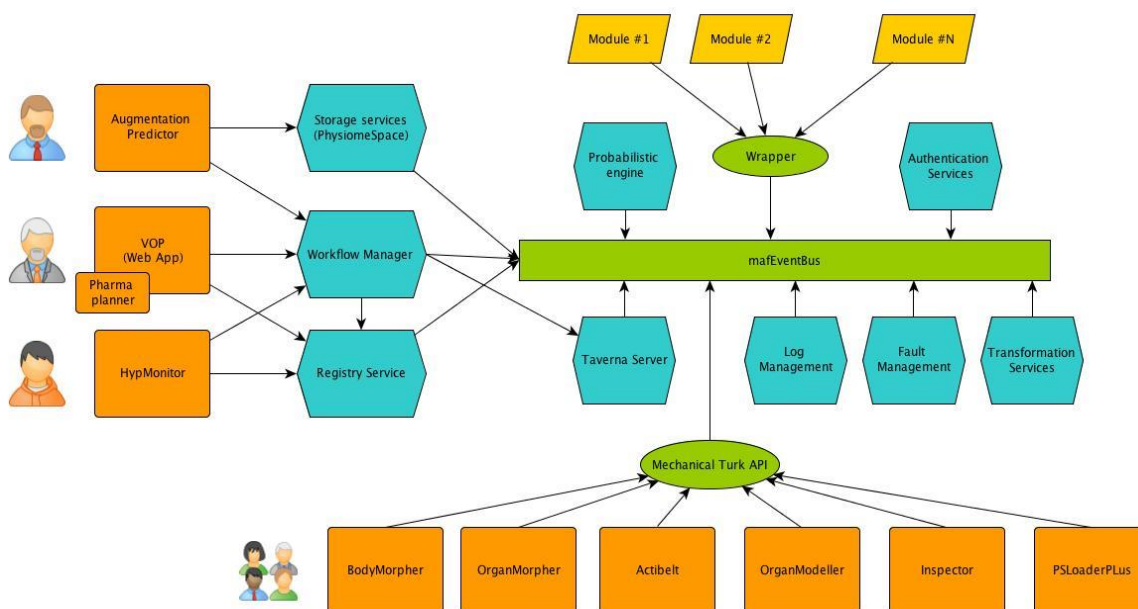


Figure 2 – VPHOP hypermodelling ICT architecture

In Year 1, the VPH-HF original software stack has been deployed on both USFD server and FORTH private cloud and two exemplar workflows have been deployed and executed. Based on this first deployment and analysis, the original VPH-HF architecture has been refactored so to better comply with the CHIC project needs.

One of the main components of the original VPH-HF was the “Template wrapper”, which allowed exposing different models to the system as a XML-RPC web service. However, from the initial deployment and requirement specifications carried out for the CHIC project it came out that this implementation has a number of drawbacks:

- the wrapping of the models needs knowledge of C++ programming and its compilation
- the C++ code wrapping was a long process in cases, as in CHIC, where a big number of models has to be made available to build hypermodels
- the code was not ready to be exposed as a REST or SOAP web service
- the framework required each hypomodel to be wrapped twice, one in the workflow execution manager (Taverna) and one in the VPH-HF itself.

Considering the above, a refactoring of the hypermodelling framework was agreed in WP7. The aim of the refactoring was to simplify the wrapping process, having a modular architecture and services exposed as REST web services. As a result of the refactoring the VPH-HF architecture would be drastically simplified, and reduced to five modules: a Director module, which would coordinate the other components, provides a web service interface, and some registry services; a storage management module, to deal with the data flow; a workflow execution module, that would run the orchestrations defined by the hypermodels; an authentication module; and a local metadata repository. In addition, consistently with the CHIC Description of Work, VPH-HF will be extended to cope with incomplete inputs and efficiency bottlenecks by implementing stochastic execution, execution caching, and support for surrogate models replacement.

### 3.2 Deployment scenarios

As part of the refactoring process, VPH-HF was re-designed to be flexible and modular, so that in the future it can be easily extended to other scenarios. We have identified the following candidate architectural scenarios:

- *Scenario #1: cluster, local storage*

The hypermodel is executed on a single computer, or on a cluster where nodes are connected with very high-speed bus (e.g. Infiniband), and all data required in the hypermodel execution can be accessed from a high-speed local file system.

- *Scenario #2: cluster + HPC, local storage*

Same as scenario #1, but one or more hypomodels require being executed on High Performance Computing (HPC) hardware.

- *Scenario #3: cluster, remote storage*

Same as scenario #1, but some of the input and/or output file needs to be transferred from/to a remote storage, for instance the CHIC repositories.

- *Scenario #4: distributed computing, limited data flow*

The hypomodels execute on various machines over the Internet but the amount of data exchanged between hypomodels is small.

- *Scenario #5: distributed computing, large data flow*

The hypomodels execute on various machines over the Internet, and the amount of data exchanged between hypomodels is large.

The CHIC project specifications require the implementation of scenario #1, which is fully supported in the current deployed version of VPH-HF.

### 3.2.1 From scenarios to components

The refactoring of VPH-HF to meet the requirements of CHIC begun by analysing what the existing components implemented in VPHOP could provide out-of-the box and what would need to be newly developed. In particular:

*Scenario #1:* can be fully addressed by the Workflow Management Service, with the addition of the VPH-HF Director (which extends the functionalities previously offered by Workflow Manager) and the Storage Management Service.

*Scenario #2:* requires a mediation layer; one option could be the integration of the UCL Application Hosting Environment [6].

*Scenario #3:* requires to access RPCs, typically web services to download-upload data. While in theory such functionality could be added to the Director, it would be better to take advantage of the refactoring to create another module (Storage Management Service) that could supervise all data related activities.

*Scenario #4:* is probably best served using a cloud computing approach; the one adopted in VPH-Share project ([www.vph-share.eu](http://www.vph-share.eu)) seems a good candidate; the approach is to add a Cloud Management module, and to use a distributed file system with WebDAV mount points, to ensure that every hypomodel executing in the cloud sees the data locally.

*Scenario #5:* could be addressed as #4 by replacing the WebDAV with a parallel FTP or similar, or by further developing the MAF3 Bus module as the VPH-HF Transport bus, that is segmented over multiple memory spaces, connect with multiple protocols (XML-RPC, SOAP, or custom high-bandwidth solutions like parallel sockets).

The CHIC project requirements are satisfied addressing scenario #1; when all CHIC Repositories are deployed, it may be necessary to support also Scenario #3, to provide some additional flexibility.

Therefore the following development strategy was adopted with the following versioned releases:

1. VPH-HF 1.0: a portable software stack addressing *scenario #1*, and includes the Director, the Storage Management Service and the Workflow Management Service. Most of the work would be in making it clean and easy to install, document it, and provide end-user services such as the Registry, plus a robust core metadata model.
2. VPH-HF 1.1: addition of support for HPC resources.
3. VPH-HF 1.2: development of a fully-fledged data management system that takes control of data staging, and replications, computational sandboxes, and interfaces to remote storage services, addressing Scenario #3.

### 3.2.2 VPH-HF components main functionalities

1. The Director exposes all hypomodels and hypermodels installed in the VPH-HF instance as a registry, returning for each one the core metadata. This can be accessed as a single Web Service that dynamically lists all available models.
2. A service authenticates a remote user, and manages the relative permissions relative to data access and models execution.

3. The Director receives and manages execution requests for single hypomodels or for hypermodels, fetches the model description files, updates the requestor of the execution status, and returns the output when the execution is completed.
4. The Director supervises the Storage Management Service that replicates the input data in the Workflow Management Server as needed by the hypomodel/hypermodel and stores the output in a repository.
5. The Director invokes the Workflow Management System to execute a given hypomodel or hypermodel with a given input.
6. If the input is stochastic, the Director invokes the Stochastic Engine, receives the input sampling, and returns the output sampling; at the end it receives the stochastic output.
7. Every time a hypomodel runs, the Storage Management System replicates the input-output pair in an execution cache, which is used by the Director for identical runs, but also by the Reduced Order Models (ROM) Engine. When a reduced order model is available, the Director can request an estimate of the output and uses for various purposes.

### 3.3 New VPH-HF architecture

The scheme below (Figure 2) represents the updated VPH-HF architecture. Briefly, the functionalities previously provided by the so-called “Template wrapper” have been de-coupled into a Storage Management Service and an improved connection between the workflow manager, now called the Director, and the Workflow Management Service. Thus, the VPHOP “Template wrapper” disappears from the VPH-HF architecture. The functionalities provided have not changed from a modeller/user perspective but they will be provided by different software components.

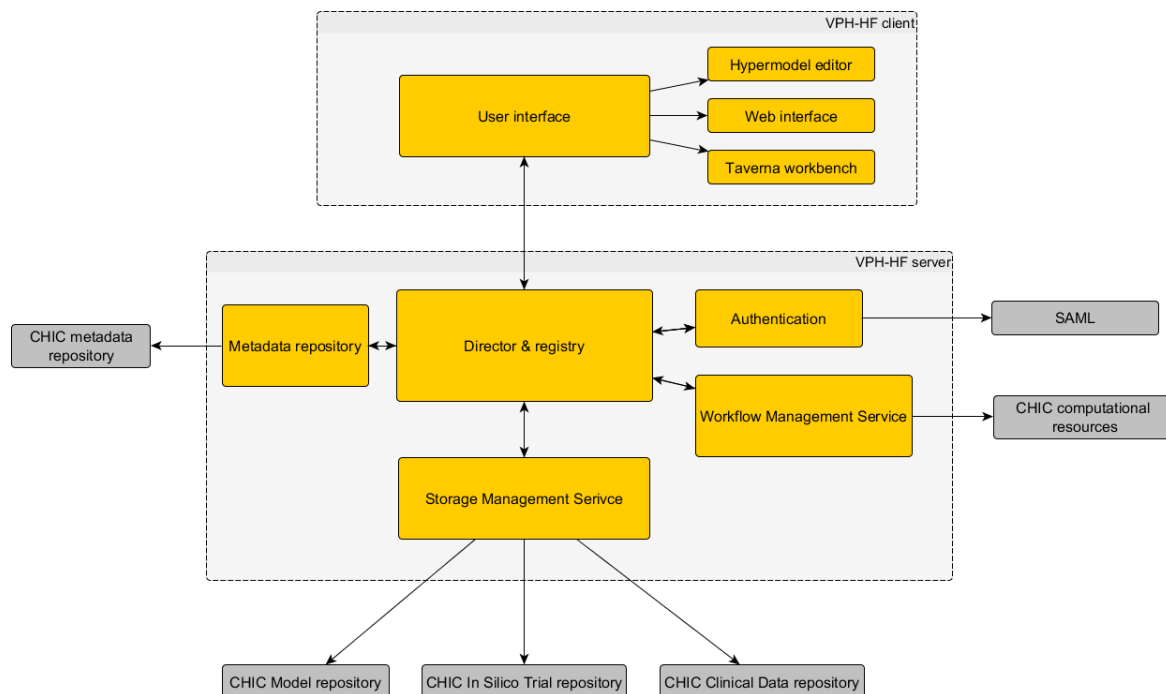


Figure 2 – VPH-HF hypermodelling ICT architecture, CHIC deployment scenario

Besides this, the VPH-HF architecture has been updated so VPH-HF components (in yellow in the scheme) can be connected to different CHIC or third party services (in grey in the scheme) depending on the deployment scenarios. This design would allow other third parties libraries/services to be supported in the future without having to modify the VPH-HF, as the interface towards the other components would not change.

The VPH-HF architecture has been thus refactored to be fully modular and to be easily extensible to different deployment scenarios.

Each VPH-HF component is exposed as a web service with a set of defined REST APIs that can be used by other VPH-HF services or also from external applications/tools. An end-user on the contrary can access the provided functionalities via different end-user applications like the CHIC Hypermodel Editor or the VPH-HF dedicated web interface.

This is a brief description of all the services functionalities:

1. **User Interfaces:** these are the components which are used by the end-users to interact with the back-end VPH-HF architecture to allow the creation, submission, and monitoring of a hypermodel execution. We are currently providing a basic web interface, designed to execute existing hypermodels on available patient's data; we are also developing the CHIC hypermodel editor, which will be used to create or edit hypermodels. Hypermodels to be executed with the Taverna workflow Management Service can also be edited using the dedicated Taverna Workflow Editor.
2. **Authentication services:** this component takes care of confirming the identity of the user and providing access according to the associated permissions. In the CHIC deployment, a SAML token will be exchanged for authentication and authorization.
3. **Director:** this was previously called *workflow manager*; it is the interface between the end-user application and the backend hypermodel technology and exposes the APIs to access the different VPH-HF functionalities. It aims to provide the functionalities to manage the request of a hypo/hypermodel execution. In particular, it would submit the execution to the workflow management server passing it the needed inputs/parameters, it would access the workflow management server to get information on the status and other parameters of the hypo/hypermodel execution and it would call the SMS to store back the output in a storage repository.  
**Registry:** this component is in the current implementation part of the Director, although functionally distinct. It provides in real time, a listing of all the resources accessible within that VPH-HF instance, including valid input sets, output sets, all available hypomodels, and all available hypermodels; each of these resources will expose a minimum set of metadata that are required before a resource is accessible within VPH-HF.
4. **Metadata Repository:** this component manages the essential metadata information necessary to the registry to provide information on which hypomodels are available to be used for composition and any information necessary for the execution. This module will not replace the metadata repository in place in CHIC but it will connect to this and store the minimum set of information necessary for the hypermodel to execute.
5. **Storage Management Service:** this component allows VPH-HF to interface with different storage solutions. It would allow the Director to retrieve the necessary inputs for the execution and to store back the outputs. This will be the interface also to the CHIC repositories.
6. **Workflow Management Service (WMS):** In the current implementation as WMS we use the Taverna Server, which orchestrates the hypermodel execution by calling the necessary hypomodels or hypermodels as specified by the Director. This module is also responsible to return to the director the status of the running hypermodels.

Depending on the deployment scenarios the different VPH-HF components can be also distributed onto different physical machines. In the case of the CHIC deployment (called also CHIC-HF) where we address the deployment scenario #3 previously described, the architecture will become the one depicted in Figure 2.

### **3.4 VPH-HF support for strongly coupled hypermodels**

The refactoring of VPH-HF allowed to deploy a modular and extensible software architecture that simplifies its sustainability in the medium/long term and candidates this platform to support future scenarios and research needs.

The core requirement for the original design of VPH-HF was to guarantee the best performance for the execution of hypermodels. This supported the assumption that a tightly coupled hypermodel needs to be designed targeting performance, which in general leads to develop models in highly optimized and/or parallelized code. This assumption reduced the number of workflow topologies to be supported by the workflow orchestrator to only the Directed Acyclic Graphs (DAG). Taverna is considered the most suitable workflow orchestrator for this scenario.

The CHIC project promotes the idea that hypomodels should be easily reusable as components in more sophisticated hypermodels, enforcing the reusability of any software artefact at any level in the project. Limiting the VPH-HF to execute only hypermodel with a DAG topology ensured such level of reusability, while providing the necessary computational efficiency. However, as the work of WP6 developed, it became evident that in some cases this limitation was excessive. In some cases, the violation of the DAG was limited to conditional branches or loops, features that the most recent versions of Taverna do support, and that we were confident could be included without breaking the general design paradigm of VPH-HF.

But in other cases, what was required was the parallel execution of two or more codes that would need to exchange not only a data flow but also control flow while executing. This broke the fundamental atomic execution nature of Taverna.

As a first attempt to address this new, unforeseen requirement, we developed in Taverna a custom component (called Taverna Coupler) that could be included in an otherwise DAG-compliant workflow to allow two models to execute in parallel while exchanging control messages, in addition to data. While the implementation was successfully completed, it became evident that the violation of such an essential design assumption would produce a number of undesired side effects. More important, it became clear that the modellers were not keen to embrace a massive refactoring of models legacy code to support the check-pointing mechanism required by this approach.

Even though the DAG approach based on Taverna is considered the most efficient, USFD and CINECA are experimenting the extension of VPH-HF with the support for a message-passing paradigm where the models are allowed to communicate each other. It will, potentially, reduce the amount of refactoring to legacy code to make it suitable to be run on VPH-HF. The message-passing paradigm can be implemented in different ways mainly depending on the computational architecture to be supported or restrictions on the communication protocols available and respective libraries. In CHIC the support should be provided for a private Cloud infrastructure where communications between models can take place on the same machine through files, sockets or similar solutions based on interprocess/queuing libraries (e.g. Boost.Interprocess<sup>3</sup>, QDB<sup>4</sup>) or TCP/IP connections that would support also instances running on different machines in the CHIC Cloud infrastructure.

---

<sup>3</sup> [http://www.boost.org/doc/libs/1\\_49\\_0/doc/html/interprocess.html](http://www.boost.org/doc/libs/1_49_0/doc/html/interprocess.html)

<sup>4</sup> <http://qdb.io>



The message-passing paradigm differs substantially from the dataflow paradigm supported by VPH-HF (representable by a DAG). In fact in the dataflow paradigm a model starts its execution as soon as the model preceding it in the DAG graph generates its input data. In a message passing paradigm all the models needing to communicate each other are launched in parallel at the beginning and are free to exchange data among them until they all terminate execution. Usually a model plays implicitly the role of orchestrator.

We investigated different strategies to improve the design of VPH-HF and different solutions were considered:

- Taverna Coupler: ad hoc component developed for Taverna to extend the WMS to support the message-passing paradigm.
- Muscle2 Coupler: a component acting as a second WMS in VPH-HF exposing the functionalities provided by the Muscle2 framework [9] (developed as part of the EC-FP7 project MAPPER<sup>5</sup>) to execute non-DAG workflows.

### 3.4.1 Taverna Coupler

The coupler component is a mini orchestrator that in principle allows the support of any kind of Directed Graphs topology. It implements in a Taverna component the logic to orchestrate the execution of an hypermodel composed of different hypomodels that cannot be easily described with DAG. Therefore it encodes a state machine to control the models and uses a message-based paradigm to issues control commands to the hypomodels. The coupler and the hypomodels can communicate through files and a mechanism to synchronize concurrent access to them has been implemented. The allowed communication pattern and the orchestration logic is hypermodel specific and have to be decided a priori. Relying on files for communication, the coupler needs to poll continuously the hypomodels to check their execution status, if they are communicating and issue new commands to them accordingly. The coupler synchronizes the access to files to send messages.

In order to allow this orchestration the code of the hypomodels has to be modified and a specific wrapper has been implemented. In fact the hypomodels orchestrated by the coupler have to be asynchronous and non-blocking, namely they are started by the coupler and return control immediately. The hypomodel continues to run in background waiting for new commands from the Coupler, sending data to another model or waiting to receive. Therefore a model wrapper based on Taverna components has been developed with the capability to start a model and return, provide the status of the model when polled by the Coupler.

Models code has to be modified to read synchronization messages and command/control instructions from the Coupler via files. The Coupler instead uses a polling loop mechanism to check the execution status of the models and acts accordingly. When all the models finish their execution the Coupler can terminate.

The early test on this approach revealed some drawbacks. The performance is limited by the speed of the Taverna component implementation of a loop. Using files to exchange messages introduces an overhead in tightly coupled execution scenarios where there is a frequent exchange of messages. Last, modellers have to modify their code to allow this file based message based communications. Nevertheless, this could be simpler then adopting other libraries.

---

<sup>5</sup> <http://www.mapper-project.eu/>

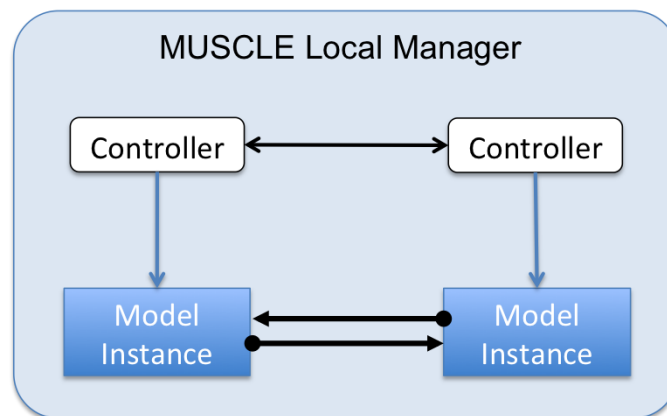


### 3.4.2 MUSCLE as Coupler and WMS

MUSCLE2 is a platform to execute time-driven multi scale simulations that was developed in the EC FP7 MAPPER<sup>6</sup> project. It requires that the sub-models can be considered as single component in a component-based simulation. The models communicate exchanging messages and they need to track their local simulation time. Furthermore a strict separation of sub-models is assumed in the design of MUSCLE 2 and a sub-model does not dictate how it should be coupled to other one.

MUSCLE2 is designed to satisfy the need to execute multiscale models on distributed High Performance Computing (HPC) platforms where models cannot communicate directly because of strict firewall policies. Moreover coupled models needs to run concurrently and this represents an issue on distributed HPC facilities that typically use job-scheduling systems. In order to solve all this problems MUSCLE developed the MUSCLE Transport Overlay (MTO) and integrated QosCosGrid<sup>7</sup> to make advanced reservation of resources, submission of cross-cluster jobs and staging in/out of simulation files. This support for the execution on distributed HPC architectures is an interesting asset for MUSCLE but for the scope of this experimentation in CHIC we are interested in the simpler scenario of a single private Cloud architecture (single site HPC). In the following we limit the analysis to this scenario.

MUSCLE2 is both a library (with API in C++, Java, Python, etc.) and an execution environment. The library allows the models to setup communication with other models without worrying about their respective location. The execution environment sets up a single Simulation Manager (per computational resource) and one or more Local managers to whom it assigns model instances (3). The Simulation Manager tracks which instances are executing and from where. The local managers start the model instances in separate threads and listen for communications for them. Model instances behave similarly to the models in the Taverna Coupler described above. They start computation immediately but they will block on message exchange with unready instances.



**Figure 3 The muscle Local Manager starts the model instances in separate threads. A controller control and dispatches communications for the instances**

The execution environment uses a network configuration topology file to setup the communications between models and Managers, but the actual coupling through the communications channels is setup at a later stage.

<sup>6</sup> <http://www.mapper-project.eu/>

<sup>7</sup> <http://www.qoscosgrid.org/trac/qcg>

The configuration topology file can be generated from a domain specific language, Multiscale Modelling Language (MML), which can elegantly describe through a formalism based on UML the multiscale models structure and their relations. This is a key feature of MUSCLE that makes it standing out from other solutions.

The described MUSCLE execution environment can be integrated in the VPH-HF architecture in two different ways not mutually exclusive and both under evaluation. It can be integrated as a second WMS in which case the Director will decide runtime depending on the workflow topology whether to use Taverna or MUSCLE. It is a straightforward solution when the workflow is only a DAG or a non-DAG.

In reality a hypermodel (workflow) could be hybrid, for example a DAG workflow embedding a Cyclic Directed Graph. One approach to address this scenario is to treat the non-DAG part of the workflow as a special configurable model called Muscle Coupler. It would take as input a topology configuration file and use MUSCLE to create the communication topology and execute the models instances.

In all the proposed approaches the modellers have to refactor their code to account for a standardised message-passing paradigm.

While we have identified two viable solutions to cope with these new requirements, and we are optimist that the VPH-HF can be extended to handle them, the potential issues of the performance and of the reusability of the hypomodels remains. At the end of this experimentation we will have to evaluate how reusable are hypomodels that require coupled message passing, and how efficient is their execution when compared to monolithic codes such as the OncoSimulator.

## 4 CHIC User scenario (A day in the life of a model)

In this section we provide a description of the process required to deploy and publish a model so that it can be exposed as VPH-HF compliant hypomodel:

- I) Prepare the test input set
  - a. Upload a set of files that form a valid test input set for your hypomodel
  - b. Create in the registry an entry for the new test input set, including the minimum annotation metadata set
- II) Prepare hypomodels
  - a. Verify that your hypomodel can be executed in batch mode on a Linux host
  - b. Do not invoke any graphical and interactive interface within your model
  - c. Ensure your hypomodel produces meaningful log files and returns correct success and error codes
  - d. Use standardized input file formats and command line arguments compliant with POSIX.1 2008 standard<sup>8</sup>
  - e. Remove all warning messages
  - f. Provide documentation and required library dependencies
- III) Install the hypomodels
  - a. Install the hypomodel code on the target cluster
  - b. Wrap the hypomodel so that it exposes as a Taverna component
  - c. Copy on the data repository an example of the input data objects
  - d. Test the execution of the hypomodel, and profile it over the test data
  - e. Provide a way to automatically test model correctness
  - f. Create in the registry an entry for the new hypomodel, including the minimum annotation metadata set
- IV) Create Hypermodel
  - a. Check that your control flow can be represented as a Directed Acyclic Graph. In this case it will be wrapped using Taverna, if not other method will be explored (Taverna or Muscle)
  - b. If there is a mismatch in the data flow of the new hypermodel
    - i. in term of data formats, install the necessary translation services
    - ii. in term of data flow logic, install the necessary relation hypomodels
  - c. Connect the hypermodel editor to a CHIC-HF server
  - d. Compose the available hypomodels into an hypermodel using the hypermodel editor which is connected to a VPH-HF instance, and lists all hypomodels available in it
  - e. Test the execution of the hypermodel, and profile it over the test data

<sup>8</sup> [http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1\\_chap12.html#tag\\_12\\_01](http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap12.html#tag_12_01)

- f. Create in the registry an entry for the new hypermodel, including the minimum annotation metadata set
  - g. Create in the model repository an entry for the new hypermodel
- V) Use the models
  - a. From the web interface connect to CHIC-HF server
  - b. Create a new input set with the data of a patient in the in silico trial repository
  - c. From the web interface, or form the Hypermodel Editor select one model and execute it with that new set as input
  - d. Receive a message that the execution is completed
  - e. Inspect the resulting output set.

## 4.1 Under the hood

Inside the CHIC-HF the execution process can be described as follows:

- 1) Director receives through its web service a request to execute the hypermodel associated with the unique id *ModelID* with the input data set with the unique id *InputSetID*;
- 2) Director queries the storage management services (SMS) to check if the model has already been run with this specific data, in which case SMS returns the URI of the relevant output data set with the unique id *OutputSetID* to the Director. The Director will send this information to the caller and terminate the hypermodel execution;
- 3) If the SMS does not find it, the Director queries the Registry and recovers the model description files associated with *ModelID* and send them to WMS, which returns a unique session with the id *sessionID*;
- 4) The Director passes to the SMS the WMS session id (*sessionID*), the model id (*ModelID*), and the input data set id (*InputSetID*);
- 5) SMS replicates the files of the input set in the execution sandbox, applying all necessary file format translation and decryption that might be required;
- 6) Director asks the WMS to execute *sessionID*, and poll the WMS until the execution is completed;
- 7) Director instructs SMS to execute the post-processing tasks, typically replicating all input-output set pairs in the cache, and the final output set back in the repository, eventually applying the necessary file format translation and encryption, and SMS returns to the Director the URI of the output data set which will have the unique id *OutputSetID*;
- 8) Director returns to the caller the URI of output data set.

## 5 Generic stub revision

### 5.1 CHIC Generic Stub

CHIC generic stub is composed of two parts: Generic Metamodel Stub and Generic Computational Model Stub.

*Generic Metamodel Stub* - All of the metamodel information should be made available from the CHIC Metadata Repositories using the programmatic interfaces (APIs) and data formats chosen by the consortium in the corresponding work packages.

*Generic Computational Model Stub* – All the models provided by WP6 should be able to be wrapped under a unified interface for the model execution. The models are categorized as three types: configurable, static and migrating.

### 5.2 Analysis of the Generic Metamodel Stub

The work on Generic Metamodel Stub focuses on the publication and reusability of the metamodel description of the models. With the support of an abstract layer for metadata repository, the VPH-HF architecture enables the reusability of metamodel description. The APIs can be included in the metadata repository component. It also has the capability to provide various presentations (data formats) of metamodel description.

### 5.3 Analysis of the Generic Computational Model Stub

The component model Generic stub is an abstraction of the models and a template for all the models of CHIC so that they can be effectively integrated into the VPH-HF and rest of the CHIC platform. The Model wrapper is an implemented instance of the Generic Stub.

In the current development stage, three types of models have been identified: *Static* models that represent a single procedural code that can be executed locally or remotely; *Configurable*, where the procedural code that describes the model is passed at run time together with the input set, and it is executed by an interpreter; *Migrating*, where the model is a process virtual machine that at run time is launched and execute a given procedural code.

As specified before, the previous “Template wrapper” functionalities have been moved to the Director and to the Storage Management Service. Then, for execution all hypomodels will be invoked directly from the WMS (Taverna or other).

First, we will analyse the accessibility of models, meaning their execution from the WMS, in its current Taverna-based implementation. Then we will look into the compatibility with respect to the model wrapper in the VPH-HF.

### 5.3.1 Accessibility of models

**Executables** - They can be accessed by the Tool service provided by Taverna. It requires the host machine of executables also work as SSH server. Examples have been shown in the demonstration of a breast cancer hypermodel using Matlab scripts.

**Web services** - Taverna provides natively the functions to access both SOAP and REST web services. In the previous development and in the ICCS hypermodel example, the access to XMLRPC services from Taverna was used by the definition of a specific beanshell (a template for future use if available). So this is also possible, though it needs a certain level of JAVA coding and the XMLRPC library support.

**VMs** – Taverna does not currently have the function to manage VMs execution. In the VPH-share project, a Taverna plug-in was developed to provide the execution of VMs from Taverna. So even if not natively from Taverna, this type of models can also be executed.

### 5.3.2 The compatibility with Model Wrapper

In D7.1, the model wrapper is defined as to support a certain number of operations mostly derived from the VPHOP template wrapper. In the table below we specify in the old and new design of the VPH-HF where these functionalities are provided.

Model wrapper functionality	Old VPHOP HF	New VPH-HF
Connection to the storage service to retrieve and push data	Template wrapper	Storage management service
Decrypt/encrypt data passed between the storage and the hypomodel	Template wrapper	Storage management service
Communication with the registry and the log management	Workflow manager	Director
Launches the script to be executed	Template wrapper	WMS
Provides functionalities to access the control flow	None	Director/WMS
Manages the creation of the log database	Template wrapper	Director/WMS

Table 2 – Comparison between the old and new design

The functional requirements (as listed in D7.1) will be exposed as APIs of the Director, which will take care, case by case, of interrogating the other VPH-HF components to collect the necessary information. The description of the Director APIs is reported in the next section.

## 6 Hypermodelling framework implementation

### 6.1 CHIC-HF client – the user interface

With respect to the original version it was decided to drop the development of the HyperMonitor application, as this was a stand-alone application based on MAF3<sup>9</sup> and was difficult to maintain with respect to the rest of the system also in relation to the cross-platform requirement. On the contrary, a web interface can make the future integration with the CHIC portal easier and it does not need specific deployment at the user site.

As presented in the architecture section, most of the VPH-HF functionalities can be accessed also directly from the Taverna client application, Taverna Workbench. However, as this is completely developed outside CHIC, it will not be presented in this section where we will focus only on the web interface and on the hypermodel editor interfaces.

#### 6.1.1 Web interface

Figure 2 shows that three VPH-HF clients are expected to interact with the VPH-HF framework, in particular with the director which has the function to orchestrate the submission and execution of a workflow: one of them is the web interface.

As we have pointed out above, a web technology has the advantage of an easier and better integration with the CHIC portal and no cross-platform issues as for a desktop GUI application. This component is in course of implementation.

As the VPH-HF services are based on the Django REST framework which uses the bootstrap engine<sup>10</sup> to create web browsable APIs, the web application is being developed relying on the same technology.

In its final version the web interface should allow the user to:

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

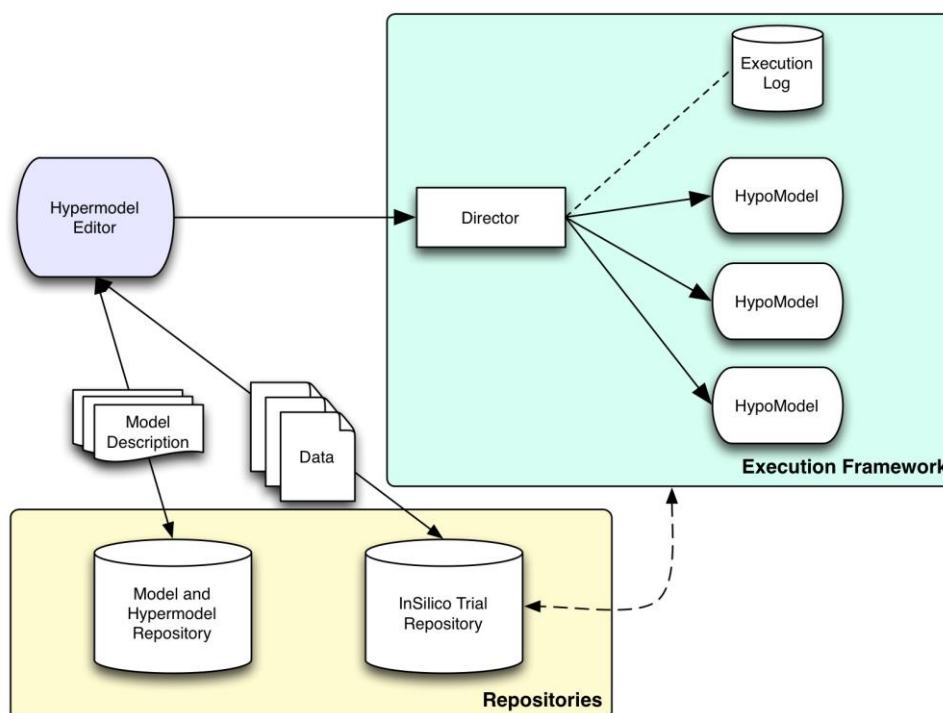
#### 6.1.2 The CHIC Hypermodelling editor

The Hypermodelling Editor of CHIC is the end user software component for designing, triggering and monitoring the execution, and retrieving the results of the CHIC integrative hypermodels. A simplified view of the architecture and its collaborators can be seen in Figure 4.

<sup>9</sup> <https://github.com/openmaf/MAF3>

<sup>10</sup> <http://getbootstrap.com/>





**Figure 4 - A simplified view of the CHIC orchestrated platform**

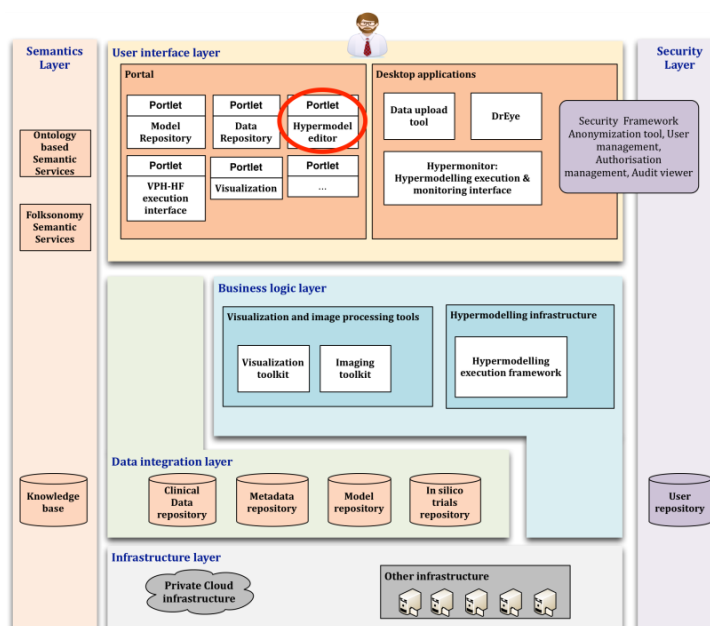
The Execution Framework provides the engine for the actual execution of the hypermodels in the CHIC computational cloud. This is where the hypomodels run exchange data and produce the final results of the hypermodels. The Hypermodelling editor is the front-end to this infrastructure. The editor allows the end users (computational biologists, researchers, etc.) to design new hypermodels and then submit them to the execution framework in order to test their research hypotheses or validate their assumptions. On the other hand, the model and data repositories are the core components of the CHIC data and model management layer. They provide query-based retrieval of the model and data descriptions in compliance with the core metadata schemas adopted in CHIC.

In this document we focus more in the Execution Framework instead of the general functionality of the Editor. More details about the Editor and its user interface can be found in deliverable 10.2. In the following we describe the interactions between the Editor and the CHIC HF that includes: a) the submission of a newly constructed hypermodel to the HF for the actual execution, and b) the monitoring and retrieval of the results of the execution.

#### 6.1.2.1 Hypermodel execution

The Editor is an end-user facing application, accessible through the CHIC portal (Figure 5). It allows the single sign on (SSO) of the CHIC users, which means that the users that have already logged into the CHIC portal are immediately identified by the Editor and do not need to enter their credentials (passwords) again. Therefore the user identity and their roles are immediately available and subject to any authorization decision made by the Editor or any of the downstream architectural components, such as the CHIC HF, which are reached through the Editor<sup>11</sup>.

<sup>11</sup> This is supported by the SAML machinery that is introduced in the CHIC security framework.



**Figure 5 - The Hypermodelling editor in the context of the CHIC architecture**

The primary goal of the Editor, that is to design new hypermodels, requires accessing the Model Repository in order for the user to browse, search, and retrieve the available hypo- and hypermodels based on their semantics and metadata based descriptions. At the level of the Hypermodelling Editor no implementation specific information is required for the models that the users have access to. On the other hand such information is obviously required in the execution framework in order for the designed hypermodels to run. The matching of the metamodel and the execution information of a corresponding realization of the same model is achieved by the use of common identifiers (URIs) shared between the model repositories and the Registry.

When the hypermodel is ready, the Editor builds the representation of the newly constructed hypermodel in the hypermodel language adopted by the CHIC project, stores it in the CHIC model repository as a new hypermodel, and then sends it to the **Director** component of the CHIC HF. In addition to this high level hypermodel description, any additional information required for the preparation of the execution such as the input parameter values that the user provides, the patient data to run the simulations with, etc. would be sent in the same request. The response from the Director will include an indication about the successful submission and an execution identifier that will allow the monitoring of the execution of the hypermodel later on. In the case of any missing information or other error, the Editor notifies the user and allows him to change some of the parameters or parts of the hypermodel.

#### 6.1.2.2 Monitoring and presentation of results

An additional functionality, after the successful submission of the hypermodel in the CHIC execution framework, is the monitoring of the execution progress. During the execution of the hypermodel there are visual indications of the execution progress while, since the models are run on the CHIC computational platform, the user can logged out and reconnect in the future, without affecting the execution. To support this functionality, the Editor cooperates with the Director, providing the execution identifier for the previous submission and retrieving the running status of the running hypermodels. Since there may be multiple hyper models executed at the same time for the same user, the Editor keeps a log with all the previously submitted hypermodels and their execution identifiers.

Additionally the user is able to retrieve results of current and past executions. When the workflow execution of a given hypermodel ends, the users are allowed to retrieve the results and in some cases, e.g. for plots, they can even preview those results inside their browser without downloading them. Each execution trace is stored centrally and kept for historical reasons, reproducibility, and provenance.

In the following we present the hypermodel execution and monitoring use case that shows the concrete interactions in a detailed sequence diagram. This is slightly adapted from the corresponding use case in Deliverable 10.2.

### 6.1.2.3 Hypermodel execution detailed scenario

The execution of a hypermodel is a complex use case. The first complexity is the setup of a new *trial* and new *experiment* that is linked with the selected hypermodel. These concepts are described in Deliverable 8.1 but basically a trial binds together a specific hypermodel and a set of patient or other data (“subjects”) that are either used as input or produced by the execution(s) of this hypermodel. Please refer to Figure 6 unterhalb for a view of this design. An experiment in this design corresponds to a single execution of the hypermodel using and producing some of the linked “subjects”.

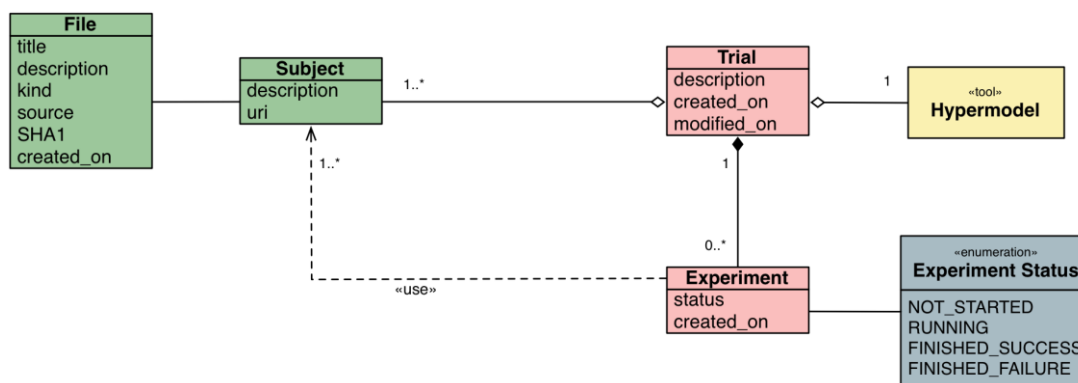


Figure 6 - A simplified view of the trials-experiments design of the InSilico Trial repository

The second “complexity” (or interesting point) in this use case relates to the actual execution of the hypermodel. This includes the communication with the Hypermodelling Execution Framework and more specifically the Director, the retrieval and storage of “Experiment” related data in the InSilico Trial repository, the monitoring of the execution, and the presentation of the final results.

In more details, Figure 7 presents the components and their interactions during the submission of a hypermodel for execution. More specifically, this sequence diagram includes the following activities:

- **Login:** The user logs into the Hypermodelling editor and the editor authenticates him/her by contacting the CHIC user authentication service. This is identical to the authentication process described in the previous paragraph.
- **Hypermodel selection:** The Editor then retrieves the available models from the model repository and filters them so that it presents only the hypermodels<sup>12</sup> to the user. The user selects one of those hypermodels and the selected one is subsequently loaded.
- **Trial and experiment creation:** The user is then guided to select one or more “subjects” to be used for the creation of new “trial” for the selected hypermodel. The subjects are retrieved using the “getAllSubjects” operation and the new trial is saved using the “storeTrial” method.

<sup>12</sup> In the case where we have a large number of models the Editor can use the “searchByClassAttribute” advanced query interface so that only the models that are truly hypermodel are returned to the Editor.

After this, the user can create a new “*experiment*” for this trial, which means that it provides additional input parameters if needed, and the editor uses the “*storeExperiment*” operation to save it in the inSilico Trial repository.

- **Hypermodel submission for execution:** On the press of a button, the user submits the hypermodel (in the format of the CHIC Hypermodelling language) for execution in the Hypermodel Execution Framework of CHIC. The editor invokes a “*runExperiment*” operation and the Execution Framework starts gathering the needed information. The Editor also registers its “interest” in monitoring the status of the execution so that the user has an overview of the running progress.
- **Hypermodel execution:** The execution framework retrieves the implementation details for each hypomodel included in the hypermodel by invoking the “*getTool*” and the related operations of the Model repository; for configurable models an actual model description file is transferred to the VPH-HF, while for static models only the URI of the hypomodel is passed, as it is already installed in the VPH-HF. Additionally, it updates the status of the experiment to “*RUNNING*” and finally to “*FINISHED\_SUCCESS*” when all the hypomodels have completed their execution. The final results of the execution are then saved in the InSilico Trial repository using the “*storeExperimentResult*” operation.
- **Presentation of results:** The Editor is notified about the successful termination of the hypermodel. It subsequently uses the “*getExperimentById*” to retrieve the complete information set for the specific experiment (i.e. hypermodel execution), which includes the produced files and results. Finally these results are shown to the user.

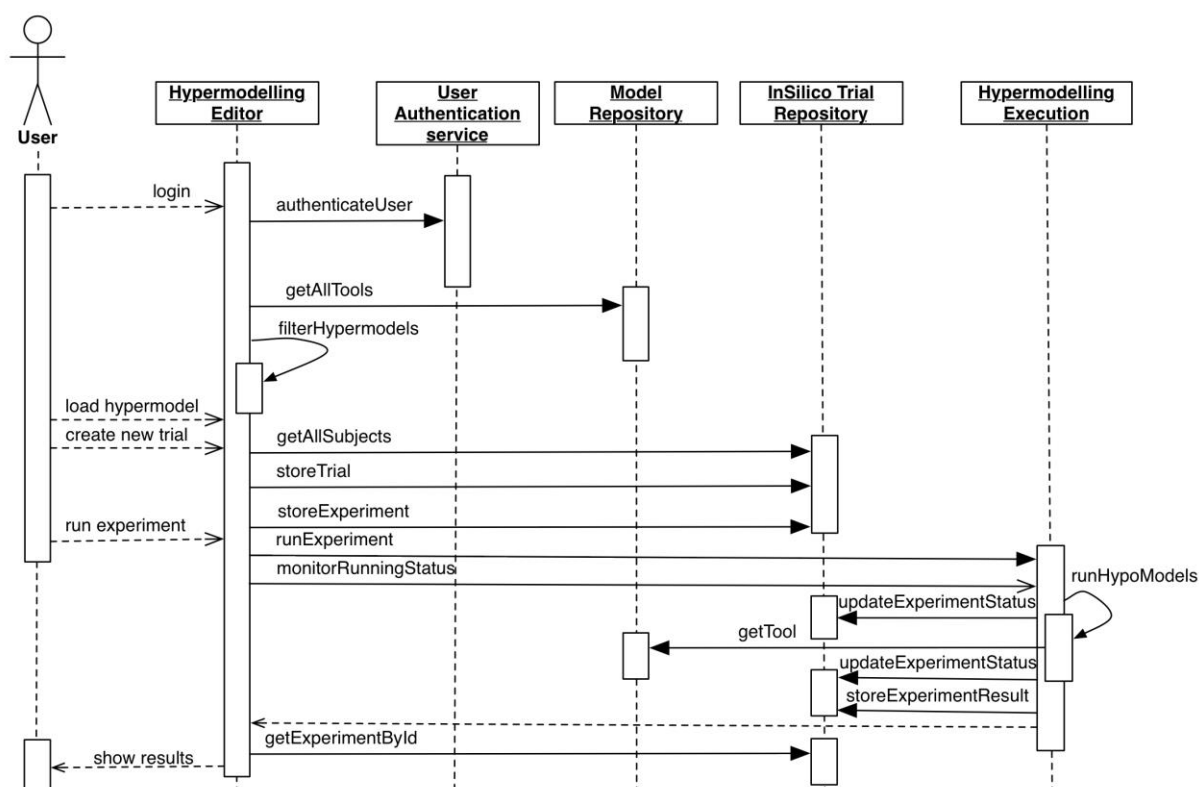


Figure 7 - Hypermodel execution sequence diagram

## 6.2 CHIC-HF server

### 6.2.1 Installation and configuration

The hypermodelling framework has been designed and implemented to be cross-platform compatible; however, as the final production environment has Linux as operative system most of the testing and documentation is provided for the deployment on Linux OS-based systems.

As mentioned before, each of the VPH-HF services can be configured so to be connected to different third parties services. The configuration can be executed manually by adapting some configuration files as described in the documentation or by using a local script, which is in course of development. In this and next sections, we will provide information on how to configure each of the services manually.

After some preliminary analysis, for the implementation of most of the VPH-HF components we decided to use python as the main programming language.

In particular, among different frameworks and technologies analysed, we have chosen Django REST framework<sup>13</sup> as base technology to be used to develop the services for the following reasons:

- It is a powerful and flexible toolkit built on top of Django that makes it easy to build and test REST Web APIs;
- Take advantage of the RESTful architectures such as statelessness, good decoupling, etc.;
- Built on top of Django, which is a mature framework;
- The Web browsable API to easy test the APIs during the development;
- Possibility to interface to the S3 amazon services;
- Possibility to combine the logic for a set of related views in a single class, called a ViewSet;
- Support of OAuth2.0 Authorization;
- Well documented and with an active developer community.

These are the software or libraries needed to use the VPH-HF services:

- *git tool* to clone the project, <http://git-scm.com/>
- *python 2.7*, <https://www.python.org/download/releases/2.7/>
- *virtualenv*, <https://virtualenv.pypa.io/en/latest/>
- *pip*, <https://pypi.python.org/pypi/pip>

On windows, after installing python, you need to install:

- *setuptools* (for python 2.7, <https://pypi.python.org/packages/2.7/s/setuptools/setuptools-0.6c11.win32-py2.7.exe#md5=57e1e64f6b7c7f1d2eddfc9746bbaf20>) or *distribute*
- instead of using the pip installer, download *get-pip.py* (<https://bootstrap.pypa.io/get-pip.py>), and run it.
- install virtual env (<https://pypi.python.org/pypi/virtualenv>)

On Mac using MacPorts utility<sup>14</sup>:

- *sudo port install python27* (or the version of interest and changing the packages version below accordingly)
- *sudo port install py27-pip py27-django py27-virtualenv*

Installation guidelines are provided for Linux (some steps might be slightly different under Windows and some notes are provided when needed on the differences).

<sup>13</sup> <http://www.django-rest-framework.org>

<sup>14</sup> <http://www.macports.org/>

Clone the project:

*git clone https://github.com/openmaf/VPH-HF.git*

Create a *virtualenv* to isolate our package dependencies locally

*virtualenv env*

and then to activate the virtual environment

*source env/bin/activate*

---

Note: On Windows use: *env\Scripts\activate*

---

Install the following packages into the *virtualenv*:

*pip install django*

*pip install django-rest-framework*

*pip install jsonschema*

*pip install django-downloadview*

*pip install pycrypto*

*pip install requests*

*pip install celery*

*pip install pyev*

*pip install xmltodict*

*pip install django-rest-swagger (to create the APIs documentation)*

Then, synchronize the database

*cd ./VPH-HF/web*

*python <appname>\_manage.py migrate*

The last step is to launch the server from the command line:

*python <appname>\_manage.py runserver host:port*

To verify that everything is up and running, open the browser and go to:

*http://host:port/\**

The default local configuration, as specified in the *config.py* configuration file, is:

*http://127.0.0.1:8000/ (SMS)*

*http://127.0.0.1:8001/ (DIRECTOR)*

*http://127.0.0.1:8080/taverna-server/ (TAVERNA)*

The VPH-HF framework needs also a Taverna server instance in order to orchestrate the submission of workflows. Follow the instructions in the Taverna website on how to install the latest version of Taverna<sup>15</sup>. Finally, in order to submit tasks to a specific celery queue a RabbitMQ server<sup>16</sup> must be up and running and some scripts must be edited and/or installed. More information on the setup process can be found in the README file of the project.

---

<sup>15</sup> <http://dev.mygrid.org.uk/wiki/display/tav250/A+Beginner%27s+Installation+Guide+to+Taverna+Server>

<sup>16</sup> <http://celery.readthedocs.org/en/latest/getting-started/brokers/rabbitmq.html>

## 6.2.2 Authentication

This section presents the VPH-HF component that can be configured to allow the access only to a set of users after their credentials verification. The user authorisation is then passed to the director, which then deals with the connection to all other services.

As from the VPH-HF deployment scenarios, this service should allow to be configured so that VPH-HF can work with different authentication protocols like SAML for CHIC. Django, the framework used to develop the Director and the Storage Management Service components, provides a mechanism to implement a custom authentication scheme. In this context the CHIC SAML authentication service has been implemented as an extension of the Django native mechanism.

### 6.2.2.1 SAML authentication

SAML is a Single Sign-On protocol that provides an authentication and authorization mechanism between the Identity Provider (idp), which provides the information about the user, and the service provider (who provides a set of services to the user).

The idp provides a token to the service provider that could be used to validate the identity and to extract basic information about the user.

For the CHIC project, the VPH-HF has to support this protocol integrating the CHIC idp provided by CUSTODIX (see D5.2 [4] for complete reference to the CHIC security layer).

- CHIC idp service descriptor: <https://sts-fp7.custodix.com/sts/services/STS?wsdl>
- CHIC user profile manager: <https://ciam-dev-chic.custodix.com/idp/Authn/UserPassword>
- CHIC idp registration: <https://ciam-dev-chic.custodix.com/idm/register.xhtml?domainName=CHIC%20Development>

CHIC idp provides three methods:

- Issue: Given the username and the password it generates a valid token.
- Renew: Given a valid token it generates a new one, where the expire time is extended.
- Validate: Given a token it checks if it is valid or not.

In our scenarios the Issue and the Validate are the most useful methods.

For the first one, the VPH-HF service extends the functionality of the *base* authentication mechanism, where the user sends the credentials in each request to the service and if they generate a valid token, the access is granted.

For the second one, it is needed to send a SAML standard request as described in the CHIC Security guideline deliverable [4]; this case represents a user who comes from another service in the CHIC infrastructure from which he/she already got a valid token and needs access to the VPH-HF.

The token is set in the header of the request as shown below:

```
Authorization=[SAML auth=<compressed_b64_token>]
```

In order to improve the efficiency of each request the token has to be compressed in advance in gzip format and base64 encoded.

## 6.2.3 Director

### 6.2.3.1 Introduction

The Director is the component of the VPH-HF framework, which refactors the old “workflow manager” and plays the central role in the hypermodelling framework. It manages the entire process for the execution of an hypermodel (i.e. workflow) and handles requests from the end-user applications of submission of workflow: authentication, preparing the computational platform, stage in- out data, submit execution to the WMS, check the execution status and retrieving outputs.

### 6.2.3.2 Functionalities

In this paragraph, the functionalities of the director will be described via the available APIs.

Method	URL	Argument	Description
GET	http://baseurl/vphhf/director/workflowlist/		Retrieve all the workflow objects stored in the database (see Figure 8)
POST	http://baseurl/vphhf/director/workflowlist/	<ul style="list-style-type: none"> <li>Workflow_title (String)</li> <li>Description (String)</li> <li>Comment (String)</li> <li>Model_id (Int)</li> <li>Inputset_id (Int)</li> <li>Experiment_id (Int)</li> </ul>	To submit a new workflow on the WMS
GET	http:// baseurl/vphhf/director/workflowlist/{id}	<ul style="list-style-type: none"> <li>Id (Int)</li> </ul>	To retrieve the <id> workflow object stored in the database
DELETE	http:// baseurl/vphhf/director/workflowlist/{id}	<ul style="list-style-type: none"> <li>Id (Int)</li> </ul>	To delete the <id> workflow object stored in the database (and abort)
GET	http:// baseurl/vphhf/director/workflowlist/{id}/status	<ul style="list-style-type: none"> <li>Id (Int)</li> </ul>	To get the status of the workflow
PUT	http:// baseurl/vphhf/director/workflowlist/{id}/status	<ul style="list-style-type: none"> <li>Id (Int)</li> <li>Status (String)</li> </ul>	To start the <id> workflow stored in the database (see Figure 9)



GET	http:// baseurl/vphhf/director/workflowlist/{id}/create time	<ul style="list-style-type: none"> <li>• Id (Int)</li> </ul>	To get the time when the workflow has been created
GET	http:// baseurl/vphhf/director/workflowlist/{id}/startti me	<ul style="list-style-type: none"> <li>• Id (Int)</li> </ul>	To get the time when the workflow has been started
GET	http:// baseurl/vphhf/director/workflowlist/{id}/finishti me	<ul style="list-style-type: none"> <li>• Id (Int)</li> </ul>	To get the time when the workflow has finished
GET	http:// baseurl/vphhf/director/workflowlist/{id}/expiry	<ul style="list-style-type: none"> <li>• Id (Int)</li> </ul>	To get the time when the workflow has expired
GET	http:// baseurl/vphhf/director/workflowlist/{id}/log	<ul style="list-style-type: none"> <li>• Id (Int)</li> </ul>	To get the log of the workflow execution
GET	http:// baseurl/vphhf/director/workflowlist/<id>/stderr	<ul style="list-style-type: none"> <li>• Id (Int)</li> </ul>	To get the standard error of the workflow execution
GET	http:// baseurl/vphhf/director/workflowlist/<id>/stdou t	<ul style="list-style-type: none"> <li>• Id (Int)</li> </ul>	To get the standard output of the workflow execution
GET	http:// baseurl/vphhf/director/workflowlist/<id>/urlou tput	<ul style="list-style-type: none"> <li>• Id (Int)</li> </ul>	To get the URL of the workflow output
PUT	http:// baseurl/vphhf/director/workflowlist/<id>/urlou tput	<ul style="list-style-type: none"> <li>• Id (Int)</li> <li>• Output_url (URL)</li> </ul>	To set the URL of the workflow output

**Table 3 - The Director APIs description. Baseurl indicate the URL that host the service, such as chic-vph.eu.**

Django REST framework v3.1.2 eefa7516-0c87-49e1-a992-5807e2004200

Api Root / Workflow List

## Workflow List

OPTIONS GET

GET /workflowlist/

```

HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, POST, HEAD, OPTIONS

[
  {
    "id": 1,
    "owner": "eefa7516-0c87-49e1-a992-5807e2004200",
    "workflow_title": "prova hello world",
    "workflow_description": "",
    "workflow_comment": "",
    "model_id": 2,
    "inputset_id": 1,
    "workflow_id": "66028296-ceae-4f80-8b70-81acc610c15c",
    "url_output": "",
    "created_on": "2015-03-13T12:00:58.721518Z",
    "status": "http://cse.shef.ac.uk:8001/workflowlist/1/status/",
    "create_time": "http://cse.shef.ac.uk:8001/workflowlist/1/createtime/",
    "start_time": "http://cse.shef.ac.uk:8001/workflowlist/1/starttime/",
    "finish_time": "http://cse.shef.ac.uk:8001/workflowlist/1/finishtime/",
    "expiry_time": "http://cse.shef.ac.uk:8001/workflowlist/1/expiry/",
    "log": "http://cse.shef.ac.uk:8001/workflowlist/1/log/",
    "stdout": "http://cse.shef.ac.uk:8001/workflowlist/1/stdout/",
    "stderr": "http://cse.shef.ac.uk:8001/workflowlist/1/stderr/",
    "exitcode": "http://cse.shef.ac.uk:8001/workflowlist/1/exitcode/"
  }
],

```

Figure 8 - The Director view for retrieval all the workflow objects stored in the DB

Django REST framework v3.1.2 eefa7516-0c87-49e1-a992-5807e2004200

Api Root / Workflow List / Workflow List Detail / Workflow Status

## Workflow Status

OPTIONS GET

GET /workflowlist/8/status/

```

HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, PUT, HEAD, OPTIONS

{
  "workflow_status": "Finished"
}

```

Raw data HTML form

Workflow status

The status of the workflow run

PUT

Figure 9 - The Director view for starting a workflow

### 6.2.3.3 Asynchronous task queue

The director is not instructed to upload directly the workflow outputs on the storage repository but to call the Storage Management Service post-processing module, which moves the final outputs to the final destination (the In Silico Trial repository). When the user creates a new workflow, an asynchronous task is submitted to a task queue in order to check periodically when the workflow run

has finished. When the execution of the workflow is completed, the SMS post-processing operations are performed and the output URL of the workflow is updated. The implementation is based on Celery and pyev. Celery<sup>17</sup> is an asynchronous task queue/job queue based on distributed message passing. RabbitMQ<sup>18</sup> has been chosen as Message dispatcher server.

## 6.2.4 Registry

This component is in the current implementation based on the Django-REST framework part of the Director, although functionally distinct. It provides in real time, a listing of all the resources accessible within that VPH-HF instance, including valid input sets, output sets, all available hypomodels, and all available hypermodels; each of these resources will expose a minimum set of metadata that are required before a resource is accessible within VPH-HF, including description, location, and license.

## 6.2.5 Workflow Management Service

The VPH-HF hypermodel framework is based on the client-server paradigm in which a Workflow Management Server represents the orchestrator of the execution of hypermodels and the Director the actor that submits/starts/stops the execution.

As explained in section 3.4 the core requirement for the original design of VPH-HF was to support only Directed Acyclic Graphs (DAG) topology in order to guarantee the best performance for the execution of hypermodels. Taverna<sup>19</sup> is considered the most suitable workflow orchestrator for this scenario. Taverna is an open source and domain-independent Workflow Management System and consists of:

- the Taverna server that acts as a remote workflow execution service that enables a dedicated server to be set up for executing workflows remotely;
- the Taverna Workbench that enables the graphically creation, editing and running of workflows locally.

The Taverna server has been installed and configured on CINECA and USFD test nodes for managing the execution of workflows started by the director (see section 6.4 on the test node deployment for more information). However, as the work of WP6 developed, it becomes evident that the violation of the DAG topology was unavoidable. In some cases, the violation was limited to conditional branches or loops, features that the most recent versions of Taverna do support; in other cases, the violation happens when the parallel execution of two or more codes exchange not only a data flow but also control flow while executing. This broke the fundamental atomic execution nature of Taverna. We investigated different strategies to improve the design of VPH-HF and different solutions were considered:

- Taverna Coupler: ad hoc component developed for Taverna to extend the WMS to support the message-passing paradigm in addition to the dataflow paradigm.
- Muscle2 Coupler: a component acting as a second WMS in VPH-HF exposing the functionalities provided by the Muscle2 framework [9] (developed as part of the EC-FP7 project MAPPER) to execute non-DAG workflows.

---

<sup>17</sup> <http://www.celeryproject.org/>

<sup>18</sup> <http://www.rabbitmq.com/>

<sup>19</sup> <http://www.taverna.org.uk/>

## 6.2.6 Storage Management Service

### 6.2.6.1 Introduction

This paragraph summarises the implementation of the Storage Management Service (SMS) as from the new design of the VPH-HF framework.

SMS is a software component capable to store and retrieve files from different type of file systems either local or remote. Currently for the CHIC needs it supports local file systems but can be easily extended to support virtually any storage system: Amazon S3, Lustre, HSF and so on. It exposes a REST interface to access virtually any supported storage system to manage files in a transparent and standardised way. This component is used by the Director to manage input and output files necessary for the execution of workflows in the WMS.

SMS is also the component responsible for the pre-processing of the input data so that they are retrieved from the storage and prepared as needed by the workflow management service (WMS); at the same time, the SMS (after the workflow execution) retrieves the output and stores it in a storage system. In CHIC, the output files are stored in the In Silico Trial repository.

### 6.2.6.2 Pre-post processing tasks

The minimum annotation metadata dictionary that annotates all VPH-HF resources includes also, in addition to individual data objects, also data sets. Thus, it is possible to refer to the set of inputs files required to initialise a certain model with a single URI, the maps to an *input set*.

As anticipated in the introduction, the SMS component can be used to execute pre-process and post-process tasks.

In particular, as pre-process task, it creates a storage sandbox on the Workflow Management Server and replicates the file in the specified input set in that sandbox; then it translates the input set, described as a set of data object URI and relative metadata, into a valid input instruction for the WMS. The Taverna format for the definition of the input list parameters uses a XML format called *Baclava*. In case multiple WMS have to be supported the Director module will instruct the SMS module on which input definition format needs to be provided.

The post-process task allows downloading the workflow output as a zip file and storing it in the In Silico Trial repository, and create an output set with the necessary semantic data, to be registered in the metadata repository.

Method	URL	Argument	Description
POST	http://baseurl/vphhf/sms/preprocess/	<ul style="list-style-type: none"> <li>Workflow_id</li> <li>Inputlist_id</li> </ul>	It creates the sandbox folder on the WMS and replicates the set of input files as specified in the input list file in that folder.
POST	http://baseurl/vphhf/sms/postprocess/	<ul style="list-style-type: none"> <li>workflow_id</li> </ul>	It allows downloading the workflow output as a zip file and store it in the In Silico Trial repository

**Table 4 - description of the APIs for the pre-post processing tasks**

### 6.2.6.3 File System Storage Service

In order to address the *Scenario #1: cluster, local storage* described in section 3.2, a file system storage service has been developed inside the SMS. The functionalities can be accessed via the following web service REST APIs described below.

Method	URL	Argument	Description
GET	http://baseurl/vphhf/sms/storagelist/filesystem/filelist/		To retrieve all the file objects stored in the database (see Figure 10)
POST	http://baseurl/vphhf/sms/storagelist/filesystem/filelist/	<ul style="list-style-type: none"> <li>Blob</li> </ul>	To upload a file and create a file object to be stored in the database (see Figure 10)
GET	http://baseurl/vphhf/sms/storagelist/filesystem/filelist/{id}	<ul style="list-style-type: none"> <li>Id (Int)</li> </ul>	To retrieve the <id> file object stored in the database
DELETE	http://baseurl/vphhf/sms/storagelist/filesystem/filelist/{id}	<ul style="list-style-type: none"> <li>Id (Int)</li> </ul>	To delete the <id> file object stored in the database
GET	http://baseurl/vphhf/sms/storagelist/filesystem/filelist/{id}/content	<ul style="list-style-type: none"> <li>Id (Int)</li> </ul>	To download the file associated to the <id> file object stored in the database

Table 5 - description of the APIs for the FileSystem Storage Service

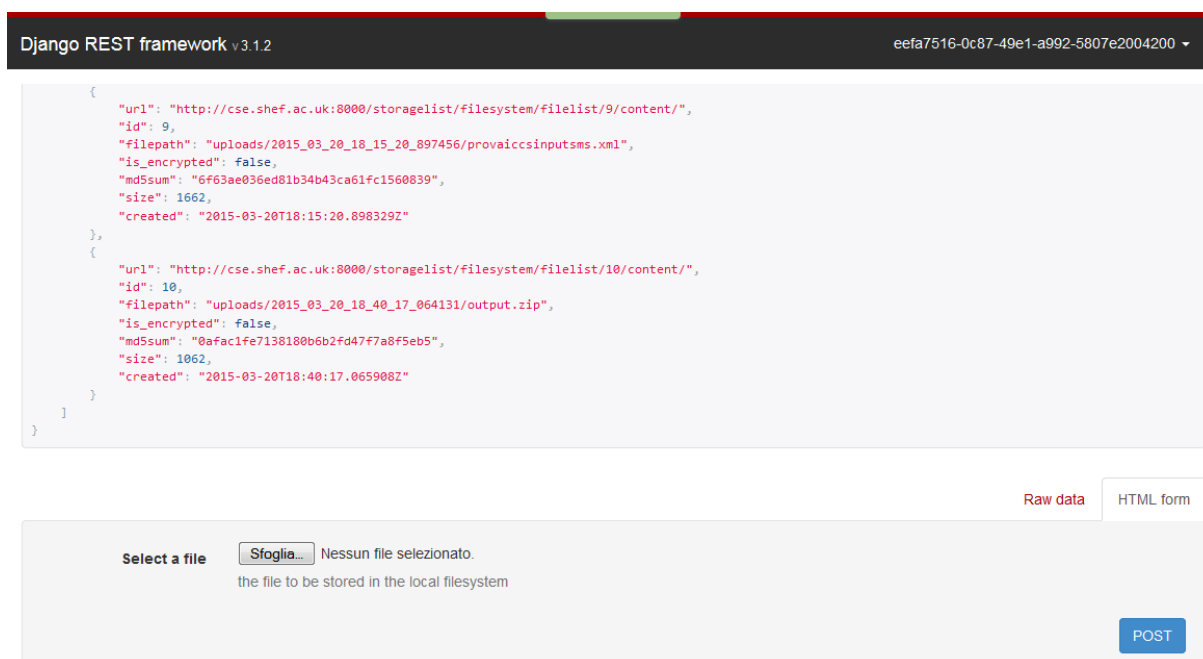


Figure 10 - The SMS view for retrieval the list of files stored in the filesystem

#### 6.2.6.4 Encryption

The necessity to protect data from unauthorised users is a requirement in the CHIC project. The solution adopted in CHIC consists of using the AES-CBC-256 **symmetric-key** algorithm to encrypt the resources. Symmetric-key algorithms are algorithms for cryptography that use the same cryptographic key for both encryption of plaintext and decryption of ciphertext. This implies that the key must be stored in a secure place and preferably on another location then where the encrypted data are stored, as we mentioned in the section above. For this reason, Custodix<sup>20</sup> has provided, as part of the CHIC security framework, a service to store the key.

The encryption services have been integrated into the VPH-HF framework through the Storage Management System. The SMS can be configured to enable/disable the encryption services according to the needs during the deployment phase. The SMS encryption module generates a 256 bit random key for every new file we want to encrypt. After the file is encrypted, the key is stored on the Custodix remote web server. To enhance the security of the system, the key is encrypted and 64-based encoded before being uploaded to the server. We use an RSA public key encryption protocol according to PKCS#1 OAEP (RFC 3447) [8]. This scheme is more properly called RSAES-OAEP [7].

More information on the encryption functionality and its implementation can be found in D10.3 [2].

#### 6.2.6.5 Remote Repository Interface

In the CHIC project, the SMS must be able to dialog with different remote repositories at the same time: the CHIC Model Repository, the CHIC In Silico Trial Repository and the CHIC Clinical Data Repository (see D8.1 [5] for more details). The CHIC Model repository contains resources associated to hypomodels and hypermodels, the CHIC In Silico Trial Repository stores the input parameters and the output data associated to a hypermodel run and the CHIC Clinical Data repository collects clinical data in the formats: DICOM (Digital Imaging and Communications in Medicine), MetaImage, Analyze, Niftii, and HDF5 (Hierarchical Data Format). An interface to interact with these repositories has been

<sup>20</sup> Custodix is a private company member of the consortium of the CHIC project

implemented inside the SMS. Only the minimum set of functionalities needed by the HF has been added to this interface.

### 6.2.7 Metadata repository

A local resource metadata repository is developed to store metadata according to the predefined schema. The development of the metadata repository is divided into two parts: metadata repository API in RESTful style and a web application client to allow users to access and add metadata, both of which are deployed on the same host.

The metadata repository API has the following endpoints. JSON is used as request and response data format.

- 1) [GET] `http://baseurl/vphhf/metadata/all`
- 2) [GET] `http://baseurl/vphhf/metadata?uri={uri}`
- 3) [POST] `http://baseurl/vphhf/metadata`
- 4) [DELETE] `http://baseurl/vphhf/metadata?uri={uri}`

Method 1) returns a list of all available metadata for all resources. The structure of the metadata record follows the definition of the metadata schema (see next section for details). Method 2) returns a filtered list of metadata. In the current implementation, the metadata can be filtered by the metadata URI. Method 3) allows creating or updating a metadata record. Similar metadata schema is used when submitted the request data to the POST method. Method 4) allows deleting a metadata record by providing the metadata URI.

The metadata repository web application client is deployed at `http://baseurl/vphhf/metadataapp`. It invokes the metadata repository API and has the following functions:

- Add metadata: a form based entry system to allow users to enter all metadata information according to the metadata schema, then submit to the server for adding the record into the metadata repository.
- View metadata: user can view a list of metadata by their URIs. A linked to the detail metadata information is provided in JSON format. A filter function is available to allow users to view a list of metadata records by the metadata URI.
- Edit metadata: when a metadata record is viewed in a detailed view, the update metadata function is enabled to allow users to modify and submit the updated metadata record to the metadata repository.
- Delete metadata: user can delete a metadata record while viewing the metadata list, the selected metadata URIs is submitted to the server to delete the record from the metadata repository.

## 6.3 CHIC-HF annotation and search services

### 6.3.1 Metadata schema

A complete set of tags and schemas with associated services for annotation and search by the users are being made available by the different CHIC partners and activities. However, to proceed with the VPH-HF implementation it was important to define from the very beginning the set of information that are needed for a hypermodel to be executed into the system. For this reason a core or minimal metadata schema for VPH-HF has been proposed and agreed with the other CHIC WPs so to make

sure that no information needed to execute the hypermodel are missing and that the mandatory ones are provided by the CHIC services.

CINECA drafted the VPH-HF metadata schema relying on the previous experience in the annotation of biomedical resources of the PhysiomeSpace service and it was adapted it to the CHIC VPH-HF specific needs.

In CHIC there are currently three types of resources: *data*, *hypomodels*, *hypermodels*. All resources share a common set of metadata, organised under:

- CommonMetadata

These include a GlobalID URI that points to the resource, name, type, description, folksonomy tags, the author of the resource, the publicity level, the ontology tags, quality indices, the documentation, the provenance history, and the first six perspective of the CHIC characterisation, which apply also to data. The CHIC perspectives are reported in details in D6.1 and D6.2 [13-14].

Then each type of resource has its own metadata, organised under:

- SpecificMetadata

With a sub-tag for each resource type. The hypomodel resource includes the CHIC perspectives 7, 10, 11, 12, and 13 (which apply only to an hypomodel), inputs, outputs, benchmarks, audit logs, etc. The dataset resource include the file size, format, checksum, endianness, whether is encrypted, etc. The hypermodel type includes CHIC perspectives 8 and 9, inputs, outputs, benchmarks, audit logs, etc.

Each metadata is described by its hierarchical position with respect to the other metadata, its name, its type (i.e. string, URI etc.), its multiplicity, its default value set if any, if it is a mandatory field or not, and a textual description of its meaning.

Below the current VPH-HF metadata schema is reported (in bold the mandatory fields); the schema is not meant to be in its final form as changes or additions might be needed in the future to have a complete and efficient integration among all the CHIC services.



Root	Tags	Sub tags	Fields	Sub-fields	Sub-Sub-fields	type	Multiplicity	ValueList	Mandatory (M=Miriam)	Description
ResourceID						tag	1		y	Root of the metadata schema
	SchemaURI					URI	1		y	URI to the schema used to annotate the resource
	SchemaVersion					date	1		y	date of the version of the used schema
	CommonMetadata					tag	1		y	metadata common to all resource types
		GlobalID				URI	1		y	unique globalID of the resource on the CHIC infrastructure
		LocalID				string	1		n	ID internal to the resource provider
		Name				string	1		y, M	title given by the author to the resource
		Type				list	1	dataset, hypomodel, hypermodel	y	type of resource
		Description				string	1		y, M	textual human readable description of



							the resource
		Tags		list	n		tags to be added by the users for folksonomies
		ResouceAuthor		string	n	y, M	creator of the resource (Miriam require name and contact)
		Access		list	1	private, shared among private groups, public access	level of visibility of the resource
		SemanticRepresentation		tag	n		semantic annotation sub-tags
			OntologyURI	URI	1		URI of the ontology used for the semantic annotation
			SemanticConcepts	tag	1		root of the associated semantic concepts. It it the subtree (eventually debora see with Bernard)
		Quality		tag	1		Quality indicators (more to be eventually added in the future)
			CurationState	list	1	Ingested, Annotated, Curated	Represents the state of curation for the object

			Rating	float	1		n	average rating of the resource by users
			Views	int	1		n	number of accesses/views of the resource
		Documentation		tag	1		n	Documentation sub-section
			ResourceCopyright	string	1		y, M	copyright of the resource (Miriam: requires terms of distribution)
			LicenseType	list	1	GPL, LGPL, EULA, OLP, TLP, VLP, BSD, MIT, proprietary, Creative Commons	y	type of license associated to the resource
			LicenceURL	URI	1		n	file or url description of the license
			References	string	n		n, M	references, papers, documentation associated to the resource (Miriam: authorship and authors' contacts must be present in the reference)
		Provenance		tag	n		n	traceability sub-section; one for each event

			Event	list	1	Create, update, delete	y, M	list of events to be extended (create mandatory as a minimum)
			EventDate	date	1		y, M	date of event (Miriam: creation and last change date. Version history and checksum would be welcome but not compulsory)
			OperatorID	string	1		n	user performing the action on the resource
			Parameters	string	n		n	parameters eventually associated with the event
			Version	string	n		n	Identficator of the resource version
		PrimaryCharacterisation		tag	1		n	Characterisation perspectives as from D6.1
			Perspective1	list	1	tumour, normal, treatment_response	n	Tumor-affected/normal tissue modelling
			Perspective2	list	1	molecule, cell, tissue, organ, organism	n	Spatial scales of manifestation of life
			Perspective3	list	1	nsec, msec, sec, min,	n	Temporal scale of the

						h,d, y		manifestation of life
			Perspective4	string	1	mutation, apoptosis, metabolism, cell_adhesion	mitosis, necrosis, n	biomechanisms addressed
			Perspective5	string	1	nephroblastoma, glioblastoma, lung_tumour	n	tumor types addressed
			Perspective6	string	1	chemotherapy, radiotherapy, targeted_therapy	n	treatment modalities addressed
	SpecificMetadata			tag	1		y	information specific for each resource type
		Hypomodel		tag	1		n	models subsection
			SecondaryCharacterisation	tag	1		n	
			Perspective7	string	1	generic_cancer_biology, clinically_driven	n	generic cancer biology / clinically driven character of modelling
			Perspective10	string	1	mechanistic, statistical	n	mechanistic-statistical character of the modelling approach
			Perspective11	string	1	deterministic, stochastic	n	deterministic-stochastic character of the modelling approach

				Perspective12	string	1	continous, discrete	n	continous, finite, discrete character of the mathematics involved	
				Perspective13	string	1	closed_form, algorithmic	n	closed form solution or algorithmic	
			Inputs		list	n		y	inputs to the model -> to be evaluated if we need to expand this as a sub-section	
			Outputs		list	n		y	outputs to the model -> to be evaluated if we need to expand this as a sub-section	
			ModelType		tag	1		y	three type of models considered according to the generic stub definition - fields are not individually mandatory but at least one must be selected.	
			HypomodelURI		URI	1		y	where the hypomodel can be phyiscally accessed	
				Static	tag	1		n	webservice-like subsection	
					ProtocolType	list	1	REST, SOAP, xmlrpc	n	protocol to be used for

										the communication
					EndPoint	URI	1		n	service endpoint
				Migrating		tag	1		n	virtual machines like sub-section
					TargetPort	int	1		n	port number at which the model answers
					TransportProtocol	bool	1	TCP, UDP	n	protocol to be used for the communication
					ApplicationProtocol	list	1	HTTP, HTTPS, FTP, SSH	n	list to be extended
				Configurable		tag	1		n	script/executables like subsection
					OS	list	n	Windows, Linux, Mac	n	
					Environment	list	1	Matlab, C, Python	n	list to be completed
					DependingLibraries	string	n		n	libraries needed to be installed for the model to run
			BenchmarkRuntime			string	1		n	Typical execution time
			BenchmarkInput			URI	n		n	Input fot benchmark test

			AuditLog	URI	1		n	link to the audit log
			Dataset	tag	1		n	data subsection
			StorageURI	URI	1		y	where the data file is located
			FileSize	int	1		y	dimension of the file
			FileChecksum	string	1		y	checksum to be used to control upload/download integrity
			FileFormat	list	1	DICOM, zip, rar, jpg, tiff, raw, CSV .....	y	list to be completed
			FileType	bool	1	ascii, binary	y	
			Endianity	bool	1	little, big	y	
			Encryption	bool	1	yes, no	y	
			Hypermodel	tag	1		n	hypermodel subsection
			inputs	list	n		y	inputs to the model -> to be evaluated if we need to expand this as a sub-section
			outputs	list	n		y	outputs to the model -> to be evaluated if we



								need to expand this as a sub-section
			HypermodelURI	URI	1		y	where the hypermodel description file is located
			TertiaryCharacterisation	tag	1		n	
			Perspective8	string	1	bottom_up, top_down, middle_out	n	order of addressing different spatial scales
			Perspective9	string	1	bottom_up, top_down, middle_out	n	order of addressing different temporal scales
			BenchmarkRuntime	string	1		n	Typical execution time
			BenchmarkInput	URI	n		n	Input fot benchmark test
			AuditLog	URI	1		n	link to the audit log
		Trial		tag	1		n	trials subsection
		Experiment		tag	n		y	
			Description	string	1		n	
			Dataset	tag	1..2		y	

			Description	string	1		n	
			<b>HypermodelURI</b>	<b>URI</b>	<b>1</b>		<b>Y</b>	
		Reference		Tag	1		n	
			<b>Type</b>	<b>list</b>	<b>1</b>	<b>book, journal article, conference article, ...</b>	<b>Y</b>	
			Doi	String	1		n	
			PubMed identifier	String	1		n	
			Date of formal issuance	Date	1		n	
			Bibliographic citation	String	n		n	
			Origin resource	String	1		n	

**Table 6 - The CHIC metadata schema.**

### 6.3.2 Free tags annotation

The free tags annotation service allows modellers to add annotation tags to resources such as data, models (including hypomodels and hypermodels). Two types of tags are implemented: individual user tags and community tags. Individual modellers add individual user tags freely, and community tags are generated according to the frequency of the individual user tags for each resource.

The development of the annotation service is divided into two parts: tagging API in RESTful style and a test web application client, both of which are deployed on the same host.

The annotation tagging API has the following endpoints. JSON is used as request and response data format.

- 1) [GET] `http://baseurl/vphhf/taggingservice/tags`
- 2) [GET] `http://baseurl/vphhf/taggingservice/tags/{label}`
- 3) [GET] `http://baseurl/vphhf/taggingservice/tags/people?id={userId}`
- 4) [GET] `http://baseurl/vphhf/taggingservice/tags/resources?uri={resourceUri}`
- 5) [GET] `http://baseurl/vphhf/taggingservice/tags/community`
- 6) [POST] `http://baseurl/vphhf/taggingservice/tags`
- 7) [DELETE] `http://baseurl/vphhf/taggingservice/tags?ids={id1,id2}`

Method 1) returns a list of all available tags on all resources. Each tag includes modeller user id – *userID*, the actual tag term – *label*, the URI of the resource being tagged – *resourceURI*, the reference of the tag term if available – *reference* and the id of this tag record – *id*. Methods 2) – 4) return a list of filtered results by the actual tag term, the modeller's user id, the resource URI, respectively. Method 5) returns a list of community tags according to the tag term frequency. Method 6) allows the modellers to add tags to a resource. The id of the tag record is automated generated. Information including modeller user id – *userID*, the actual tag term – *label*, the URI of the resource being tagged – *resourceURI*, and the reference of the tag term if available – *reference*, are required when submitting the request. Method 6) allows tags to be deleted by providing the id of the tag records.

The tagging web application client, which invokes the annotation tagging API, is deployed at `http://baseurl/vphhf/taggingapp/`. There are three main functions: add tags, view tags and view community tags.

Figure 12 and 13 shows the snapshot of the adding tags page in the annotation tagging client. Modellers have to provide a user id, being tagged resource URI and tag terms when adding tags. After resource URI is provided, if a resource description exists, it will be extracted and display to the user. When entering tag terms, modellers can either enter their free terms or use a term suggested by ontologies. Three sets of ontologies are used in this implementation, which are Gene Ontology (GO) including biological process ontology, cellular component ontology, molecular function ontology, Cell Ontology (CL) and Foundational Model of Anatomy Ontology (FMA). Figure 13 shows a sample list of suggestions for auto complete the tag terms.

Once the *add tag* form is submitted, the tags are added into the resource metadata. Figure 14 shows a sample resource metadata file in XML format with user added tags.

Figure 15 shows a snapshot of the viewing tags page in the annotation tagging client. Users are allowed to delete selected tags, view tags by filtering by user id, resource URI or tag terms. If the reference of a specific tag term (e.g. *biological\_process*) if available, a link to the reference is provided.



Tagging Service Client
Add Tags
View Tags
View Community Tags

User ID

Resource URI

Tags

Add Tag

**Resource Description**  
Matlab based implementation of the ODE model  
<http://www.ncbi.nlm.nih.gov/pubmed/17382967>

Figure 11 - Snapshot of annotation tagging web application client – add tags.

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 12 - Snapshot of annotation tagging web application client – add tags with ontology terms suggestion.

← → cse.insigneo.org:8090/chic/resource/1

Click entity for XPath. Double-click to collapse/expand. Enter XPath or XML string then click XPath/Render for results or to XML Tree-ify, respectively.

Source Options  
XPath/Render XML

```
<Resource mandatory="true" resourceid="1" type="tag">
  <SchemaURI mandatory="true" multiplicity="1" type="uri"></SchemaURI>
  <SchemaVersion mandatory="true" multiplicity="1" type="date"></SchemaVersion>
  <CommonMetadata mandatory="true" multiplicity="1" type="tag">
    <GlobalID mandatory="true" multiplicity="1" type="URI">
      http://cse.insigneo.org:8090/chic/resource/1 </GlobalID>
    <LocalID mandatory="true" multiplicity="1" type="string">
      1 </LocalID>
    <Name mandatory="true" multiplicity="1" type="string">
      WNT_Bcatenin_model </Name>
    <Type mandatory="true" multiplicity="1" type="list">
      Hypomodel </Type>
  </CommonMetadata>
  <Description mandatory="true" multiplicity="1" type="string">
    Matlab based implementation of the ODE model http://www.ncbi.nlm.nih.gov/pubmed/17382967 </Description>
  </Resource>
  <Tags>
    <tag>
      <label>ODE</label>
      <userID>xia</userID>
    </tag>
    <tag>
      <label>model</label>
      <userID>xia</userID>
    </tag>
    <tag>
      <label>protein</label>
      <userID>xia</userID>
    </tag>
    <tag>
      <label>cell_cycling</label>
      <userID>xia</userID>
    </tag>
    <tag>
      <label>hypomodel</label>
      <userID>Kewei</userID>
    </tag>
    <tag>
      <label>hypomodel</label>
      <userID>Kewei</userID>
    </tag>
    <tag>
      <label>hypomodel</label>
      <userID>Kewei</userID>
    </tag>
  </Tags>
```

Figure 13 - A sample resource metadata with added user tags.

Tagging Service Client   Add Tags   **View Tags**   View Community Tags

filter by UserID ▼  Submit

Delete Selected

Tag	Resource	Provider
<input type="checkbox"/> test	<a href="#">prova</a>	testi
<input type="checkbox"/> workflow	<a href="#">prova</a>	testi
<input type="checkbox"/> edit	<a href="#">prova</a>	testi
<input type="checkbox"/> hypomodel	<a href="#">prova</a>	testi
<input type="checkbox"/> workflow	<a href="http://cse.insigneo.org:8090/chic/resource/2">http://cse.insigneo.org:8090/chic/resource/2</a>	testi
<input type="checkbox"/> normal	<a href="http://cse.insigneo.org:8090/chic/resource/2">http://cse.insigneo.org:8090/chic/resource/2</a>	testi
<input type="checkbox"/> hypomodel	<a href="http://cse.insigneo.org:8090/chic/resource/2">http://cse.insigneo.org:8090/chic/resource/2</a>	xia
<input type="checkbox"/> biological_process	<a href="http://api.ccgiv.org.uk/taggingservice/tags">http://api.ccgiv.org.uk/taggingservice/tags</a>	xia
<input type="checkbox"/> cellular_component	<a href="http://api.ccgiv.org.uk/taggingservice/tags">http://api.ccgiv.org.uk/taggingservice/tags</a>	xia
<input type="checkbox"/> molecular_function	<a href="http://api.ccgiv.org.uk/taggingservice/tags">http://api.ccgiv.org.uk/taggingservice/tags</a>	xia
<input type="checkbox"/> hypomodel	<a href="http://api.ccgiv.org.uk/taggingservice/tags">http://api.ccgiv.org.uk/taggingservice/tags</a>	xia
<input type="checkbox"/> hypomodel	<a href="http://chic-project.eu/">http://chic-project.eu/</a>	xia
<input type="checkbox"/> hypomodel	<a href="http://chic-project.eu/category/about">http://chic-project.eu/category/about</a>	xia

Figure 14 - Snapshot of annotation tagging web application client – view tags.

### 6.3.3 Semantic metadata

#### 6.3.3.1 Introduction

Semantic metadata refers to: facts about a data set—that is, meta-facts, or metadata—which facts are specially formatted for greater machine readability, interoperability, and automated semantic reasoning. In the context of CHIC, models and their parts are given various annotations, and these annotations are stored as metadata. For the sake of interoperability and automated reasoning, these annotations refer to standard reference ontologies. For example, if we have an image of a lung, rather than annotate it with a bare string ("lung"), we might annotate it with an ontology term, such as for example FMA\_7195, the "lung" entry in the Foundational Model of Anatomy [<http://sig.biostr.washington.edu/projects/fm/>], a widely-used anatomy ontology. This makes it possible to use the metadata easily, without having to program ad hoc machinery to do so. This also reduces ambiguity and avoids over-dependence on a particular spoken language (English).

#### 6.3.3.2 Tools

CHIC uses RICORDO [<http://www.ricordo.eu/>], a collection of APIs that facilitate the creation, storage, and reasoning over of metadata. Individual annotations are stored in the form of RDF triples. RDF is the Resource Description Framework [<http://www.w3.org/RDF/>], a W3C recommendation which forms the backbone of the semantic web. An RDF triple consists of a subject, a predicate, and an object. For a concrete example, imagine a triple whose subject is (a formal name of) an image; whose object is (an ontology term for) "lung"; and whose predicate is (a formal name of) "is an image of". This triple means that the image in question is an image of a lung. Crucially, by storing the fact in this form, the fact becomes amenable to automated reasoning, in an interoperable way.

Triples are stored in a so-called "triple store". A triple-store is a database specially designed to contain metadata in triples form (i.e., in the form of a subject-predicate-object factoid). Currently, the most appropriate triple store is Virtuoso [[http://www.w3.org/2001/sw/wiki/OpenLink\\_Virtuoso](http://www.w3.org/2001/sw/wiki/OpenLink_Virtuoso)], an open-source triple store with demonstrated performance and scalability. RICORDO provides (via its RDFSTORE program) a template API intended to facilitate querying a triple store with user-friendly forms. To create annotations using ontology terms, it is necessary to search an ontology and find which term to use. RICORDO provides tools for doing this. RICORDO's Local Ontology Lookup Service provides an API for quickly searching for ontology terms. RICORDO further provides, through its OWLKB program, an API for automated semantic reasoning, and can generate so-called "composite terms", semantically meaningful combinations of existing ontology terms. For example, suppose we wish to annotate a procedure with a date of occurrence, and suppose we have reference ontologies with terms for "date" and for "occurrence", but not for "date of occurrence". OWLKB can be used to generate a composite term for "date of occurrence" from the constituent terms "date" and "occurrence", and most importantly, the composite term is semantically meaningful: automated reasoners can understand what it means.

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

##### 6.3.3.2.1 Input and output of RICORDO tools

RICORDO tools accept input through an API over the HTTP protocol, and send their responses as HTTP responses. For specific details about the API, and specific format of the input to RICORDO, see RICORDO's documentation [<http://open-physiology.org/owlkb/doc.html>], [<http://open-physiology.org/rdfstore/doc.html>], [<http://open-physiology.org/LOLS/doc.html>].

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

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

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

### 6.3.3.3 The CHIC semantic metadata lifecycle

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

#### 1. Creation

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

#### 2. Insertion into triple store

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

#### 3. Querying

Once stored in the triplestore, triples are accessed by querying. The triplestore can be queried directly, using the SPARQL query language. This, however, will be invisible to the end-user: such queries will mostly be done programmatically "under the hood" by other programs in CHIC. (As an alternative to SPARQL, such other programs can also use RICORDO's RDFSTORE template system; this decision is up to the developers of the other components of CHIC.). In particular, our intention is that the ultimate end-user be able to query and use the metadata via friendly graphical front-ends requiring no knowledge of the underlying technology.

### 6.3.3.4 What can be annotated?

Flexibility is one of the key virtues of the RDF data format. Rather than referring to objects with machine-specific designators (filenames, memory locations, database indices, etc.), objects are referred to by IRI ([Internationalized Resource Identifiers, another W3C standard](https://www.ietf.org/rfc/rfc3987.txt) [https://www.ietf.org/rfc/rfc3987.txt]), making RDF agnostic about technical implementation details. In short, anything that can be given a stable IRI, can be annotated.

## 6.4 Test nodes deployment

### 6.4.1 USFD node deployment

The test node at the USFD is configured consistently with the technical specification of the final production node, which will be hosted by FORTH, and to comply with the other software components specifications highlighted in deliverable D5.1 [3]. The assumption underlying the configuration choice is that the CHIC Hypermodelling Framework will be deployed in a private Cloud in a scenario where the computational resources are located in the same institution computational facility. Even though this scenario could easily be replicated on a single machine, two test nodes with different hardware configurations are used at USFD to test more complex scenarios and allow a better insight in terms of analysing the framework performance.

Two different abstraction layers are identified in the deployment of the framework: the VPH-HF Orchestration Layer and the Computational/Model layer (Figure 6). The former comprises all the VPH-HF components, libraries and tools to enable the orchestration of workflows, storage and retrieval of data. The latter, Computational/Model layer, comprises the suite of models with the tools and libraries they require for their execution. The two layers can easily be decoupled and deployed on different machines. Specifically, the Computational/Model layer may require substantial computational resources available only on HPC clusters. Therefore it is envisaged that some models could be installed on a dedicated machine and the user will have the chance to select the most appropriate target machine for the execution of his model workflow.

The VPH-HF design is based on modularity. Its components can be installed and run from different machines provided the necessary security measures are in place. Namely, the components belonging to the Orchestration layer (Figure 6) can be installed both on a single machine or different distributed

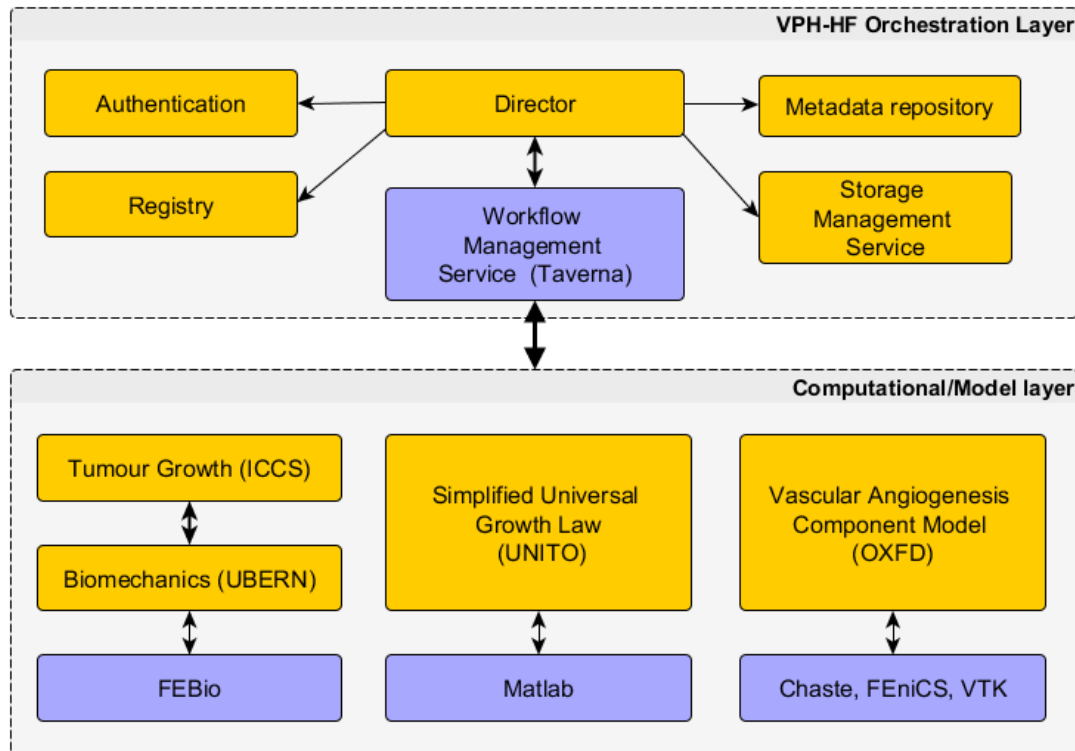


Figure 15 - The software layers on the USDF test node. The orchestration layer on top comprises components developed by USFD and CINECA using Python and Django Framework; the Workflow Management Service is provided by a third party engine, the Taverna Server. The computational/Model Layer is composed by the models currently provided by WP6 partners. Components in orange background are developed within CHIC project, in light blue the third-party components. The arrows show the computational dependencies between models and libraries.



ones. The same logic applies to the Computational Layer. An alternative scenario would be to deploy the VPH-HF Orchestration Layer on an application server while the Computational/Model Layer on a HPC resource and it is considered in the VPH-HF design and development for future extensions. The VPH-HF components (Director, Storage services, Local repositories, etc) are developed in Python relying on Django<sup>21</sup>, Django Rest Framework<sup>22</sup> and SQLite<sup>23</sup>. As explained in section 6.2, VPH-HF uses Taverna Server<sup>24</sup> as workflow management engine, which requires Tomcat<sup>25</sup> server to run. The VPH-HF components interact through secure encrypted connections and X.509 certificates.

#### 6.4.1.1 WP6 model requirements

The test node in USFD has been configured in order to run the models provided by partners of the WP6 and described in deliverable D6.2. The models were first provided as monolithic hypermodels whose hypomodels cannot be isolated and run independently. WP7 and WP6 teams are collaborating to decouple the hypomodels composing each hypermodel and verifying implications in performance. Some technical solutions have been provided and analysed to solve the problem. One of these is to abstract the hypo models as “*communicating processes*”: they are not passive any more, they are reactive, sending and receiving information without the intervention of the central orchestrator. The orchestrator’s primary responsibility would then be to start them, let them exchange data independently, and possibly stop them when the results of the hyper model have been produced.

The models are implemented in different programming languages and use particular interpreters or external libraries:

- **The (Pure) Tumour Growth and Response to Treatment**  
Component Model (ICCS) / Biomechanics component model (UBERN).  
Format: Binary executable file coupled with external software FEBio<sup>26</sup>.  
Third-party software/libraries: FEBio<sup>26</sup>.
- **Vascular Angiogenesis Component Model (UOXFD)**  
Format: Python interpreted code.  
Third-party software/libraries: Chaste<sup>27</sup>, FEniCS<sup>28</sup> (version 1.4), VTK (version 5.10), Python.
- **Gross Component Models Based on Simplified Universal Growth Laws (UNITO)**  
Format: Matlab® scripting language.  
Third-party software/libraries: Matlab®

<sup>21</sup> <http://www.djangoproject.com>

<sup>22</sup> <http://www.django-rest-framework.org/>

<sup>23</sup> <https://www.sqlite.org/>

<sup>24</sup> <http://www.taverna.org.uk>

<sup>25</sup> <http://tomcat.apache.org/>

<sup>26</sup> <http://febio.org/>

<sup>27</sup> <http://chaste.cs.ox.ac.uk>

<sup>28</sup> [www.fenicsproject.org](http://www.fenicsproject.org)

#### 6.4.1.2 Software and Libraries installed

The USFD test node is configured to support the VPH-HF components and the models execution. Therefore the above-mentioned frameworks, the external software and libraries are installed and configured to run in a secure environment. The machine runs Ubuntu 14.04 LTS 64 bit operating system on a dual Intel Xeon 12 core (E5-2695 v2 @ 2.40GHz), 256GB of RAM memory, 14TB storage space in RAID 5.

Software installed and relative versions are reported below:

- Python 3.4 and Python 2.7
- Django 1.7.5
- Django Rest Framework 3.1
- MySQL 5.6
- FEniCS 1.4
- VTK 5.10
- Matlab R2014b
- FEBio
- Taverna Server 2.5
- Apache Tomcat 7.0.53

#### 6.4.2 CINECA node deployment

The test node at CINECA is configured to host in a secure environment the new VPH-HF framework for development purposes. As we said before, the VPH-HF design is based on modularity, which means that its components can be installed and run from different machines provided the necessary security measures are in place. In the CINECA node deployment, the components belonging to the Orchestration layer (Figure 6) are all installed on the CINECA test machine. This machine runs a Ubuntu 14.04 LTS 64 bit operating system installed in a Virtual Machine in a Cloud environment with an Intel Xeon 2 cores (E-312xx @ 2.5GHz (Sandy Bridge)), 4 GB of RAM memory and 10 GB storage space. The Computational Model/layer is not deployed in the CINECA test machine but the Computational Model/layer service hosted on the USFD test node is reused. This is legitimated by the fact that the Computational Model/layer is not affected by the development process of the VPH-HF. Finally, a set of workflow tests have been created and used for regression testing during the development phase.

Software installed and relative versions are reported below:

- Python 2.7
- Django 1.7
- Django Rest Framework 3.1
- SQLite 3
- Taverna Server 2.5.4
- Apache Tomcat 7.0.52

## 7 Conclusion

In this deliverable, the architecture of the new VPH-HF framework is described both from a modeller point of view and a developer point of view, providing a description of the current implementation of all its components, functionalities, APIs and technological choices. The transition from the previous version of the framework to the new one is also briefly reported. As a consequence of this architectural refactoring, the CHIC generic stub has been revised.

The framework installation and the execution of some hypermodels provided by partners of WP6 have been deployed and tested on USFD node. The framework installation and testing have been also performed on CINECA test node.

The next steps concern the integration of the CHIC hypermodelling editor with the CHIC-HF with particular attention to the communication protocol. Even though the two services interact via a Web Service interface, the editor has to instruct the VPH-HF framework to execute any newly designed workflow. A Hypermodelling Language based on XML is under definition by partner CINECA with collaboration from USFD and FORTH starting from the Multiscale Modelling Language developed in the EC MAPPER project.

## 8 References

- [1] CHIC\_600841 D7.1 – Hypermodelling specifications
- [2] CHIC\_600841 D10.3 - The CHIC encryption services
- [3] CHIC\_600841 D5.1-1 – The CHIC technical architecture – initial version
- [4] CHIC\_600841 D5.2 – Security guidelines and initial version of security tools
- [5] CHIC\_600841 D8.1 – Design of the CHIC repositories
- [6] Coveney, PV; Saksena, RS; Zasada, SJ; McKeown, M; Pickles, S; (2007) The application hosting environment: Lightweight middleware for grid-based computational science. COMPUT PHYS COMMUN , 176 (6) 406 - 418. [10.1016/j.cpc.2006.11.011](https://doi.org/10.1016/j.cpc.2006.11.011).
- [7] <https://www.dlitz.net/software/pycrypto/>
- [8] <http://www.ietf.org/rfc/rfc3447.txt>
- [9] J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. Ben Belgacem, D. Groen, and A. G. Hoekstra, “Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment,” *Journal of Computational Science*, vol. 5, no. 5, pp. 719–731, Sep. 2014.
- [10] Viceconti M 2011 A tentative taxonomy for predictive models in relation to their falsifiability. *Philos Transact A Math Phys Eng Sci* 369(1954):4149-61.
- [11] <http://en.wikipedia.org/wiki/Metadata>
- [12] <http://www.infoq.com/news/2008/09/Orchestration>
- [13] CHIC\_600841 D6.1 – Cancer hypomodelling and hypermodelling strategies and initial component
- [14] CHIC\_600841 D6.2 – CHIC cancer component models: initial tested versions

## Appendix 1 – Abbreviations and acronyms

<i>VPH-HF</i>	Virtual Physiological Human – Hypermodelling Framework
<i>VPHOP</i>	Virtual Physiological Human – OsteoPorotic
<i>XML</i>	eXtensible Markup Language
<i>RPC</i>	Remote Procedural Call
<i>REST</i>	REpresentational State Transfer
<i>SOAP</i>	Simple Object Access Protocol
<i>HPC</i>	High Performance Computing
<i>WMS</i>	Workflow Management System
<i>WEBDAV</i>	WEB Distributed Authoring and Versioning
<i>FTP</i>	File Transfer Protocol
<i>ROM</i>	Reduced Order Models
<i>AES-CBC</i>	Advanced Encryption Standard-Cipher Block Chaining
<i>OAEP</i>	Optimal Asymmetric Encryption Padding
<i>RSA</i>	Initials of the surnames Ron Rivest, Adi Shamir, and Leonard Adleman; it is a public-key cryptography algorithm
<i>RFC</i>	Request For Comment
<i>DAG</i>	Direct Acyclic Graph
<i>SMS</i>	Storage Management System
<i>VM</i>	Virtual Machine
<i>HTTP</i>	HyperText Transfer Protocol
<i>IDP</i>	IDentity Provider
<i>MUSCLE</i>	MUltiscale Coupling Library & Environment
<i>DICOM</i>	Digital Imaging and Communications in Medicine
<i>HDF</i>	Hierarchical Data Format
<i>GO</i>	Gene Ontology
<i>CL</i>	Cell Ontology
<i>FMA</i>	Foundational Model of Anatomy Ontology
<i>RDF</i>	Resource Description Framework
<i>W3C</i>	World Wide Web Consortium
<i>SPARQL</i>	Protocol and RDF Query Language
<i>RICORDO</i>	Researching Interoperability using Core Reference Datasets and Ontologies for the Virtual Physiological Human
<i>GUI</i>	Graphical User Interface
<i>SSO</i>	Single Sign On

*SAML*      Security Assertion Markup Language