



## Deliverable No. 7.4

### Final Hypermodelling framework deployed on test nodes

Grant Agreement No.: 600841  
Deliverable No.: D7.4  
Deliverable Name: Final Hypermodelling framework deployed on test nodes  
Contractual Submission Date: 30/09/2016  
Actual Submission Date: 30/09/2016

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



# COVER AND CONTROL PAGE OF DOCUMENT

Project Acronym:	CHIC
Project Full Name:	Computational Horizons In Cancer (CHIC): Developing Meta- and Hyper-Multiscale Models and Repositories for In Silico Oncology
Deliverable No.:	D7.4
Document name:	Final hypermodelling framework deployed on test nodes
Nature (R, P, D, O)	O
Dissemination Level (PU, PP, RE, CO)	RE
Version:	1.2
Actual Submission Date:	30/9/2016
Editor: Institution: E-Mail:	Simone Bnà CINECA simone.bna@cenea.it

## ABSTRACT:

This deliverable describes the final release of the hypermodelling framework (VPH-HF) technology and its deployment on the CHIC production and test nodes. 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. Minor updates and bug fixes are expected to take place during the remainder of the project, but all hypermodelling framework components are now in place and the connection with the other CHIC services has been implemented. In this document, the hypermodelling framework final version is described from an architectural point of view; for each VPH-HF component implementation details and APIs are provided together with instructions for the framework deployment.

## 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.*

<b>MODIFICATION CONTROL</b>			
<b>Version</b>	<b>Date</b>	<b>Status</b>	<b>Author</b>
0.1	20/07/2016	Draft	Simone Bnà
0.2	04/08/2016	Draft	Debora Testi
1.0	12/09/2016	Draft	Simone Bnà
1.1	19/09/2016	Final	Debora Testi
1.2	30/09/2016	Final	Comments from internal reviewer included

#### List of contributors

- Debora Testi, CINECA
- Simone Bnà, CINECA
- Stelios Sfakianakis, FORTH-ICS
- Ioannis Karatzanis, FORTH-ICS
- Eleni Georgiadi, ICCS
- Eleni Kolokotroni, ICCS
- Georgios Stamatakis, ICCS
- Dawn Walker, USFD
- Daniele Tartarini, USFD
- Sam Gamble, USFD
- Marco Viceconti, USFD
- James Grogan, UOXF
- Alok Ghosh, UPENN
- Ioannis Karatzanis, FORTH

## Contents

1	EXECUTIVE SUMMARY .....	5
2	INTRODUCTION .....	6
2.1	PURPOSE OF THIS DOCUMENT.....	6
2.2	DEFINITIONS .....	6
3	CHIC ARCHITECTURE OVERVIEW .....	9
4	VPH-HF HYPERMODELLING FRAMEWORK .....	13
4.1	HYPERMODELLING FRAMEWORK ARCHITECTURE .....	14
4.2	HYPERMODELLING FRAMEWORK COMPONENTS AND IMPLEMENTATION .....	16
4.2.1	<i>User Interfaces</i> .....	17
4.2.2	<i>Director</i> .....	21
4.2.3	<i>Storage Management Service</i> .....	22
4.2.4	<i>Authentication Service</i> .....	22
4.2.5	<i>Registry</i> .....	23
4.2.6	<i>Message Broker</i> .....	23
4.2.7	<i>Workflow Management System - Orchestrator</i> .....	24
4.2.8	<i>Caching Service</i> .....	24
4.2.8.1	<i>Cache Integration</i> .....	25
4.2.9	<i>Surrogate Modelling Service (SUMOD)</i> .....	26
4.3	CHIC GENERIC STUB .....	27
4.3.1	<i>Analysis of the Generic Metamodel Stub</i> .....	27
4.3.2	<i>Analysis of the Generic Computational Model Stub</i> .....	27
4.3.3	<i>Hypermodel Multiscale modelling</i> .....	30
4.4	HYPERMODELLING FRAMEWORK INTEGRATION INTO CHIC PLATFORM.....	37
4.4.1	<i>VPH-HF and the Hypermodelling editor</i> .....	37
4.4.2	<i>VPH-HF and CRAF</i> .....	41
4.4.3	<i>VPH-HF and Authentication/authorisation</i> .....	41
4.4.4	<i>VPH-HF and the CHIC repositories</i> .....	42
5	HYPERMODEL EXECUTION .....	43
5.1	HYPERMODEL EXECUTION SCENARIO .....	43
5.2	DATA FLOW .....	45
6	CHIC-HF DEPLOYMENT .....	46
6.1	VPH-HF ORCHESTRATION LAYER DEPLOYMENT .....	47
6.1.1	<i>Presentation Layer</i> .....	48
6.1.2	<i>Logic Layer</i> .....	49
6.1.3	<i>Data Layer</i> .....	51
6.2	COMPUTATIONAL/MODEL LAYER DEPLOYMENT .....	52
6.3	THE CHIC WILMS ONCOSIMULATOR HYPERMODEL EXAMPLE .....	53
6.3.1	<i>Introduction</i> .....	53
6.3.2	<i>Building and deployment in the hypermodelling framework</i> .....	54
7	CONCLUSION .....	56
8	REFERENCES .....	57
	APPENDIX 1 – ABBREVIATIONS AND ACRONYMS .....	60
	APPENDIX 2 – API DESCRIPTION .....	61
2.1	WORKFLOW DATA MODEL .....	61
2.2	DIRECTOR API DESCRIPTION .....	62
2.3	FILE OBJECT DATA MODEL .....	68
2.4	SMS API DESCRIPTION .....	69

## 1 Executive Summary

During the last twelve months, Work Package 7 (WP7) has been focused on the release of the final prototype of the hypermodelling framework (VPH-HF) and its deployment on both the test nodes and the production environment. Two distinct layers are identified in the deployment of the framework: the VPH-HF Orchestration Layer, and the Computational/Model layer.

The VPH-HF Orchestration Layer 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 exposed via a web service interface to allow reusability. In particular, at the time of the deliverable submission all VPH-HF components have been developed and deployed. However, some minor changes or bug fixing might be still required before the end of the project to support the deployment of the final version of the CHIC hyper/hypo-models.

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

These two layers: VPH-HF Orchestration and Computational/Model, can be easily decoupled and deployed on different machines. This flexibility will allow the future deployment of the computational/model layer on HPC facilities if model execution has specific computational needs.

The VPH-HF framework has been deployed on test nodes (CINECA and USFD) as well as on the production environment (CHIC private cloud), and it is being tested using hypomodels and hypermodels developed as part of WP6 activities.

## 2 Introduction

### 2.1 Purpose of this document

During the third year of the CHIC project, the main goal of WP7 was the release of the final prototype of the hypermodelling framework (VPH-HF), including development, deployment, and testing of the hypo- and hyper-models made available by WP6.

This document provides a description of the hypermodelling framework final prototype and in particular addresses the description of the architecture together with the details on the specific components and their implementation.

At the same time, many CHIC technical WPs have been developing prototypes for their respective tools; thus, this document is also reporting on the positive collaborative work among the different CHIC teams, which has allowed efficient and successful integration of all the CHIC components.

The deliverable also describes from where the source code can be downloaded and how it can be installed on other systems.

In addition to the description of new components, some sections are repeated or updated from the previous deliverable (D7.2) to provide the reader with a complete picture of the hypermodelling framework.

### 2.2 Definitions

An agreed set of definitions has been put into place in an 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 as a reminder to 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. However, in our cases where the execution of the services is coordinated 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>	We define data as factual information, whether observed or predicted. <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>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 software component and user tool that provides appropriate mechanisms for the design of new hypermodels through the composition of multiple hypomodels.
<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).</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 are the semantic descriptions 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.
<b>Wrapper</b>	Software layer that “wraps” an existing model implementation and provides the integration layer so that the model can become a hypomodel of a hypermodel. This implies translation of the control flow, and management of the data flow.

**Table 1 – CHIC project definitions**



### 3 CHIC architecture overview

The IEEE 1471 standard “Recommended practice for Architecture Description of Software-Intensive Systems” [1] addresses the activities of the creation, analysis, and sustainment of architectures of software-intensive systems, and the recording of such architectures in terms of architectural descriptions. The **architectural description** (AD) is defined as “a collection of products to document an architecture”. The AD can be divided into several views each one covering one or more stakeholder concerns.

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

For the selection of specific views and viewpoints of the architecture, there are a number of different architectural frameworks supporting and proposing different views and viewpoints, such as the 4+1 views model [2], the Rozanski and Woods model [6], the Reference Model of Open Distributed Processing (RM-ODP) [3], the Zachman framework [4], the Department of Defense Architecture Framework (DoDAF) [5], and many more.

In this section, we use the **Functional View** [6], as this view is essential to describe the functionality and the functional elements of an architecture (for more details on the CHIC architecture please refer to D5.1.1 “The CHIC technical architecture – initial version”). The functionality of the system results from the functionality of its constituent functional components. Some functional components are meant to be used by end users, mostly the ones in the user interface layer, while other components are meant to be used or interact programmatically with other software components and consequently they will provide a programmatic interface (API).

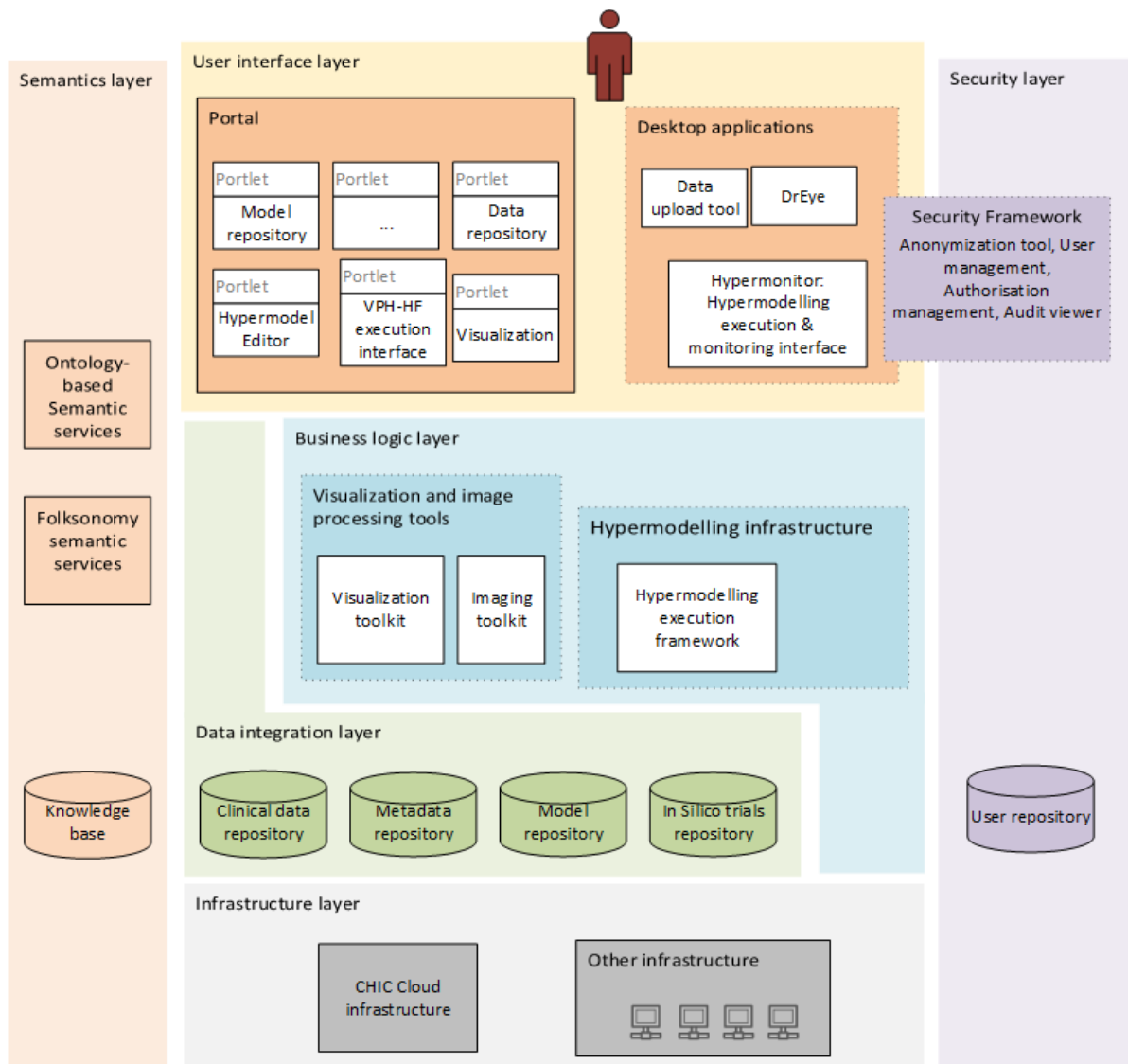
In order to describe the functional view of the system, we need a list of its functional elements, their responsibilities, their interactions and their interfaces, either user interfaces or programmatic ones. An initial description of the CHIC functionalities from the end users point of view has also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

In Figure 1 we depict a high level view of the main functional components of the CHIC platform, organized into various layers and functional groups:

- On top there is the **user interface layer**, which includes the user portal, the main point of interaction of the end users with the rest of the platform, and some standalone applications. The portal can further be decomposed into distinct functional elements, the *portlets*, which act as the user interface of corresponding functional components of the platform.
- Below the user interface layer, the **business logic layer** is located, which includes the visualization and imaging toolkit and the **hypermodelling infrastructure** (VPH-HF); the functionality of this last component will be covered in detail in this deliverable.
- The **data integration layer** sits below, and includes the logical repositories of the system.
- The **infrastructure layer** is located at the bottom, providing support in the other functional components in terms of computational and storage needs.
- Two vertical layers are crossing all the other layers described above: the **semantics layer** and the **security layer**.
  - The semantics layer provides the functionality of a logical integration element of the system, by linking data and functional components via a common, semantic ‘glue’.

- The security layer serves a similar goal, yet on another aspect, by linking data and functional components with a security framework, which guarantees the legal and ethical concerns of the stakeholders.

The layering described above is a logical separation which serves in order to better illustrate the functional groups and their role in the overall functionality of the platform. However, there is no any strict separation of the interaction between the various layers: for example the user portal can also interact with the data integration layer and the repositories, and the business logic layer also interacts directly with the infrastructure layer.



**Figure 1: Functional View - A High Level view of the CHIC functional components**

As already mentioned the topic of this document is the VPH Hypermodelling Framework (VPH-HF). This is a collaborative computational platform providing a complete Problem Solving Environment (PSE) [5] to execute, on distributed computational architectures, sophisticated predictive models involving patient medical data or specialized repositories.

In the following paragraphs, we provide a brief description of all the CHIC components that interact with the hypermodelling framework. For detailed information on the implementation of each of those components please refer to the corresponding deliverables.

## **User Interface Layer**

### CRAF

The Clinical Research Application Framework (CRAF) is the end user application used by Clinicians to run the hypermodels developed in the CHIC project: Nephroblastoma, Non Small Cell Lung Cancer, Glioblastoma, and Prostate Cancer. The objective of this application is to provide a user-friendly environment where the clinician can simulate the cancer evolution according to one of the implemented hypermodels on real patient data in order to address specific clinical questions.

### **Business logic layer**

#### Hypermodelling editor

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

#### Visualization toolkit

The visualization toolkit, developed within CHIC and started via CRAF, consists on a set of modular functionalities allowing the different users to perform from basic to advanced visualization operations.

## **Data Integration layer - CHIC Repositories**

### Model repository

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

### In Silico trial repository

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

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

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

### Clinical data repository

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

### Metadata repository

The metadata repository is the repository for metadata about CHIC resources, namely hypomodels and associated data. This metadata provides descriptions of the resources according to a number of

facets, including multiscale anatomy, units of measurement, biophysical qualities and physiological and pathological aspects. The aim of the repository is to make available annotations of CHIC resources to facilitate, for example, semantic search but also to warrant the semantic coherence and integration of CHIC resources.

### **Infrastructure layer**

#### Private Cloud

The CHIC private cloud is an infrastructural component with the main purpose to provide computational and storage facilities to other components [27].

## 4 VPH-HF Hypermodelling framework

The **VPH Hypermodelling Framework** (VPH-HF) is a collaborative computational platform providing a complete Problem Solving Environment (PSE) [23] to execute, on distributed architectures, sophisticated predictive models involving patient medical data or specialized repositories. It is based on a fully-fledged prototype developed in a previous VPH project, the Osteoporotic VPH (VPH-OP) (<http://www.vphop.eu>), which addressed the estimation of bone fracture risk due to osteoporosis.

The aim of VPH-HF is to improve the effectiveness of diagnosis, prognosis and treatment of specific diseases in clinical practice with the ultimate objective to foster the personalized medicine paradigm and perform in silico clinical trials [24]. In particular, the VPH-HF is customized for the oncological needs targeting two primary users: the **clinician** and the **researcher**. The former needs easy out-of-the-box software tools to analyse patient medical data and simulate cancer behaviour to address specific clinical questions. The latter has a broader profile that includes the creation, modification, and validation of complex integrative models [7]. Researchers can populate the CHIC data and model repositories with their experimental data and provide the integrative/predictive models implemented in the computational format of their convenience.

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

- **Integration of models**, which can be developed with different software tools or libraries and be deployed in the PSE on different hardware and/or operating systems;
- Communication between the models (see Figure 2), which can be classified into two types:
  - the **control flow**, which is the set of instructions that needs to be passed from one sub-model to another or to the system for its execution;
  - the **data flow**, which is the data input-output of each sub-model; in order to define this, the data formats used in both input and output by each sub-model have to be identified.

The underlying assumption is that a hypermodel (i.e. an integrative model or a composition/orchestration of models) can be described as a workflow where its composite hypomodels (i.e. models) are connected to produce an output result from a given input and data from repositories and/or patient specific data. Therefore, a workflow can be represented as a graph where the nodes are models or data repositories, while connections are data or control flows. Two models are connected when an output of the first is an input of the second, while data repositories can be connected to any of the models. In order to build a workflow, hypermodels and hypomodels can be considered as black boxes with a standardized abstract interface exposing input and output ports and control data flow (Figure 2). This interface is well defined within the CHIC project and it ensures the interoperability between all the provided hyper- and hypomodels: it is called **Component Model Generic Stub**. The VPH-HF is compliant with this interface and provides a software implementation that follows the Wrapper pattern [8]. It allows the actual integration of any of the computational instances of the models in a workflow including data and control flow by adapting the parameters from the format used in the actual model to the standard one of the Component Model Generic Stub Interface.

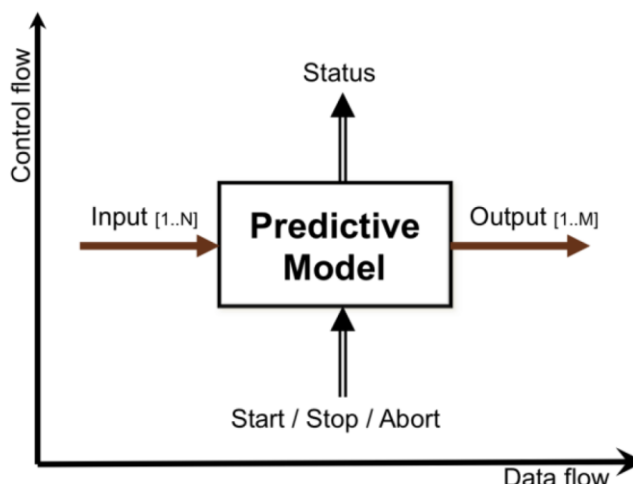


Figure 2: Communication between models: predictive model as a black box

### 4.1 Hypermodelling framework architecture

The most common scenario, that drove the VPH-HF architecture design, was that 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 the CRAF user interface, the user selects an existing hypermodel, or from the hypermodelling editor composes available hypomodels into a new hypermodel, and then requests its execution to the VPH-HF with a certain input set. From the CRAF application the user monitors the execution of the hypermodel, and at the end of the execution s/he can retrieve the hypermodels output set, which has also been stored centrally in the inSilico Trial Repository.

The architecture of VPH-HF has been inspired by the concept of modularity: each component can be used in isolation or ensemble with others to offer more sophisticated functionalities. Each component provides services to other components via a communication protocol over a network such as HTTP, HTTPS and AMQP following the Service Oriented Architecture (SOA) pattern. This approach ensured an effective extension of the VPH-HF prototype developed in the VPH-OP project to the new requirements and scenarios of the CHIC project. All the software components expose a standard interface (API) that potentially allows them to be used in isolation. In particular, the web components of the hypermodelling framework are designed according to the REST architectural style<sup>1</sup> [39].

VPH-HF adopts a client-server architecture in order to process server-side the requests coming from the consumers of the execution framework like the CRAF application (the client). In Figure 3, all the components inside the cloud are part of the backend of the framework. The CRAF application and the Admin panel - dashboard are instead clients, the first external to VPH-HF, the other one internal.

In Figure 3 an overall view of the VPH-HF architecture is shown together with the connections between the internal and external components. Please note that with respect to the previous version, the metadata services have been moved out of VPH-HF and are now managed completely by the corresponding semantic services provided as part of the CHIC platform (for this see more details in D7.3).

This is a brief description of all VPH-HF services functionalities:

<sup>1</sup> Representational state transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web

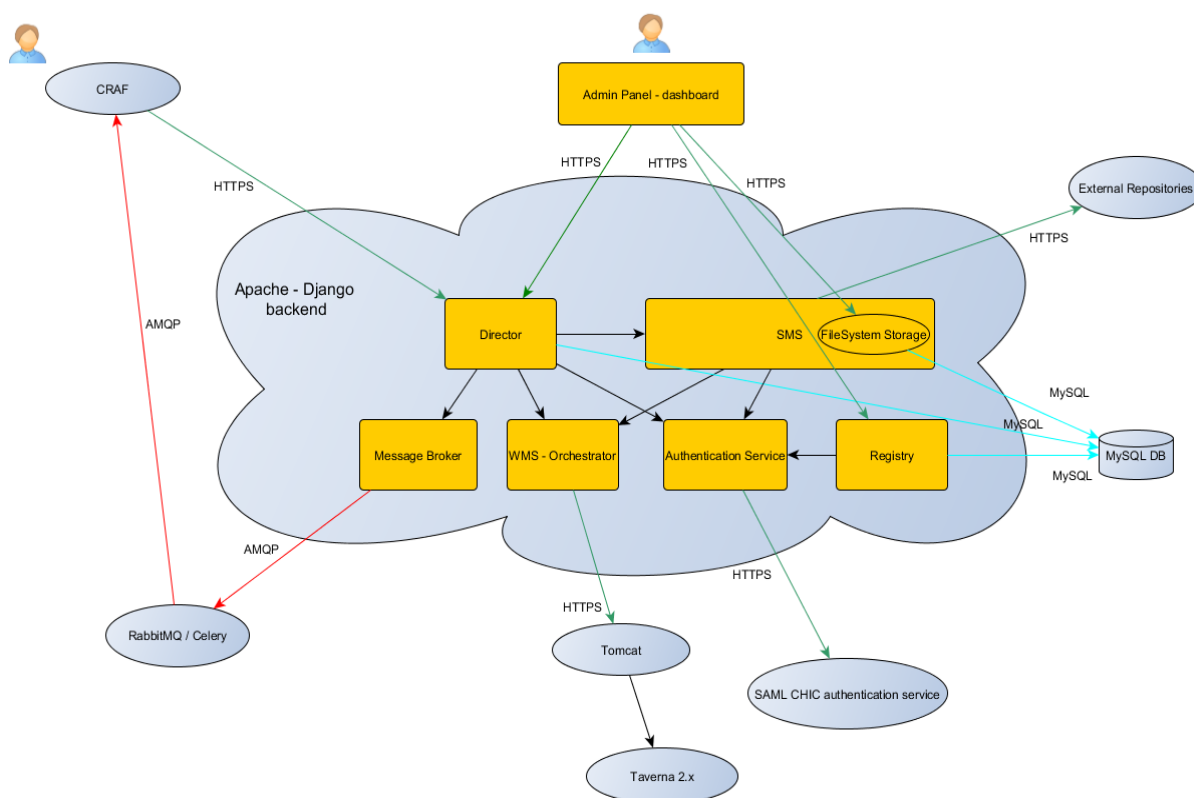
1. **User Interfaces:** these are the components which are used by the end-users to interact with the back-end VPH-HF architecture to allow the creation, submission, and monitoring of a hypermodel execution. The CRAF application is an external component of the VPH-HF that allows selection and execution of existing hypermodels in the cancer domain in order to address clinical questions. From CRAF it is also possible to monitor the execution of the hypermodel and to visualize the output results. Inside the VPH-HF a basic web interface (called “Admin panel-dashboard”) was designed and implemented to execute existing hypermodels on available patient’s data and a dashboard to monitor the execution, retrieve the results and analyse the logs. The latter was implemented in order to provide a simple and direct interface to VPH-HF for independent testing and future deployment after the end of the CHIC project. Hypermodels to be executed with the Taverna workflow Management Service ([www.taverna.org.uk](http://www.taverna.org.uk)) can also be edited using the dedicated Taverna Workflow Editor (also called Taverna Workbench).
2. **Director:** this acts as an interface between the end-user application (CRAF, dashboard) and the backend hypermodel technology and exposes the APIs to access the different VPH-HF functionalities needed from outside the framework. It aims to provide the functionalities to manage the request of a hypo/hypermodel execution. In particular, it submits the execution to the workflow management service passing it the required inputs/parameters, polls the workflow management service to get information on the status and other parameters of the hypo/hypermodel execution and calls the storage services to save the output in a storage repository.
3. **Storage Management Service (SMS):** this allows VPH-HF to interface with different storage solutions. It allows the Director to retrieve the necessary inputs for the execution and to save the outputs. This is the interface also to the CHIC repositories.
4. **Authentication service:** this takes care of confirming the identity of the user and provides access according to the associated permissions. In the CHIC deployment, a SAML token is exchanged for authentication and authorization.
5. **Registry:** It provides in real time a list of all the hypomodels/hypermodels accessible within the VPH-HF instance (for example those available for testing but not yet uploaded into the model repository), including for each of them valid input sets, hypermodel (t2flow) description and (possibly) output sets. The registry is also responsible for the automatic deploy of hypomodels within the VPH-HF infrastructure.
6. **Message Broker (or Message Bus):** this component is responsible for the message validation, transformation and routing. RabbitMQ, a popular implementation of the standard Advanced Message Queuing Protocol (AMQP) was integrated within VPH-HF. AMQP defines an efficient and flexible publish/subscribe interface that is independent of the data model, so it can be used to communicate with external component as Model Repository and CRAF.
7. **Workflow Management Service (WMS) - Orchestrator:** the WMS is the service that orchestrates the hypermodel execution by calling the necessary hypomodels or hypermodels as specified by the Director. This module is also responsible for returning the status of the running hypermodels to the Director. This service in the current implementation is provided by the Taverna server [11].
8. **Caching Service:** Caching is a technique for optimization which, when enabled, stores previous execution output data for selected models. On each request a combination of



(model ID + input values) are checked for a matching cached execution, if found the memorized output files are returned to the Taverna Server instead of re-executing the identical request. If no matching execution is found the model is run as normal and results are added to the cache. Once properly populated the cache of a computationally expensive model can be used to build reduced-order versions of such model, such as surrogate models.

9. **Surrogate Modelling Service (SUMOD):** Mechanistic models are expected to reproduce or predict with the highest fidelity a natural phenomenon in the shortest amount of time. The trade-off between speed and accuracy is crucial in domains like *in silico* medicine where clinicians would not have sufficient time to wait for long simulations. Surrogate modelling (or meta-modelling) is a methodological approach where a high-fidelity model or full-order model is replaced by a simplified low-fidelity/reduced-order model with a significantly shorter execution time. The SUMOD service provides a set of functionalities for VPH-HF to retrieve surrogate models suitable of replacing a full-order, more computational expensive model, and rank them by accuracy and execution time over a reference dataset.

In the following sections, a more detailed description of each component together with the functionalities and programmatic APIs are provided. For details on the libraries and languages used for the development please see Section 6.1 (VPH-HF Orchestration layer).



**Figure 3: Architecture of VPH-HF Hypermodelling Framework in the CHIC project deployment**

## 4.2 Hypermodelling framework components and implementation

In order to describe the functional view of the hypermodelling framework according to the Rozanski and Woods model, in the following paragraphs we provide a description of its functional elements



and the functionalities they provide, their interactions with other components and their interfaces, either user interfaces (UI) or programmatic interfaces (API).

## 4.2.1 User Interfaces

Figure 3 shows that two clients are expected to interact with the VPH-HF server part of the framework, in particular with the Director service that has the function to manage the submission and execution of a workflow: the CRAF application and the Admin panel-dashboard. The two clients are built with standard web technologies that have the advantage of an easier and better integration with the CHIC portal and no cross-platform issues as for a desktop GUI application.

### 4.2.1.1 CRAF

Due to the high complexity of the modelling work and especially its computational demands, the multiscale modelling environment of CHIC takes advantage of state of the art “cloud” technologies, which are provisioned outside the clinical setting. In order to make this a truly functional framework for accelerating the *clinical* translation of multiscale cancer models a Clinical Research Application Framework (“CRAF”) has been developed to support a unified and simple user experience and provide a “CHIC-in-a-box” abstraction for the clinicians to use in clinical research performed in their premises. To this end, its user interface is designed to be simple and smooth by hiding the complexity of the CHIC platform while, at the same time, demonstrating its full potential for clinical research and empowering the clinician to use the underlying technologies for the benefit of the cancer patient. The aim of CRAF therefore is to provide a single tool in the clinical domain for the clinicians to run the CHIC models that have been verified for clinical use in order to gain potentially valuable information for the best treatment or diagnosis of their patients.

From the technical point of view, CRAF is accessible as an online (web based) application using state of the art HTML5 technologies. Due to this design, CRAF has a fully “responsive” layout which adapts to the screen size of any device, and is suitable for usage from a personal computer, a tablet device or a mobile phone. CRAF is designed using the Material Design<sup>2</sup> principles and controls. The layout is flat and it presents only the necessary information to the end user. We next present some indicative user interactions in CRAF.

In order to access the CRAF the clinician has to provide the proper credentials at the login form and click the login button, as shown in Figure 4.

<sup>2</sup> The visual language of the Material Design, <https://material.google.com>

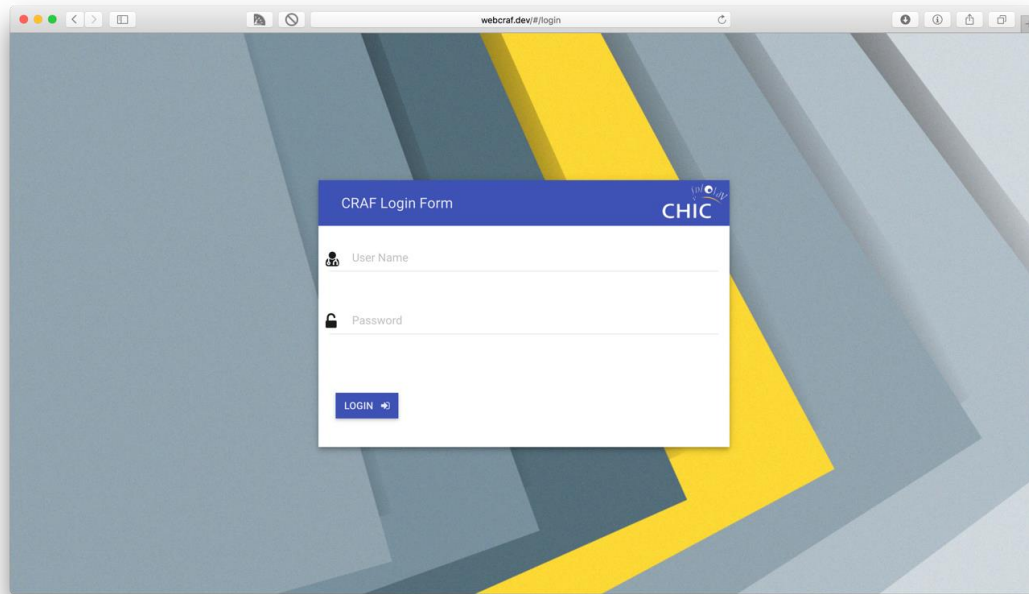


Figure 4: CRAF Login Form

CRAF is integrated with the security infrastructure of the CHIC platform and therefore the users can use their CHIC credentials. If the credentials provided by the user are correct then CRAF successfully verifies the user and provides him/her permission to use the full functionalities of the CRAF application and access the data which are available specifically to him/her (data are filtered based on the user's role and permissions).

As it is shown in Figure 5, CRAF then displays the list of patients the user has access to. The patient list is interactive, and as the user picks another patient from the list, the patient's card at the right side is updated, in order to display the patient's corresponding data. When the desired patient is selected, the user proceeds to the next step (Figure 6), where he/she has to select the question of interest for the patient of interest. The questions displayed in this step are relative to the cancer type of the selected patient.

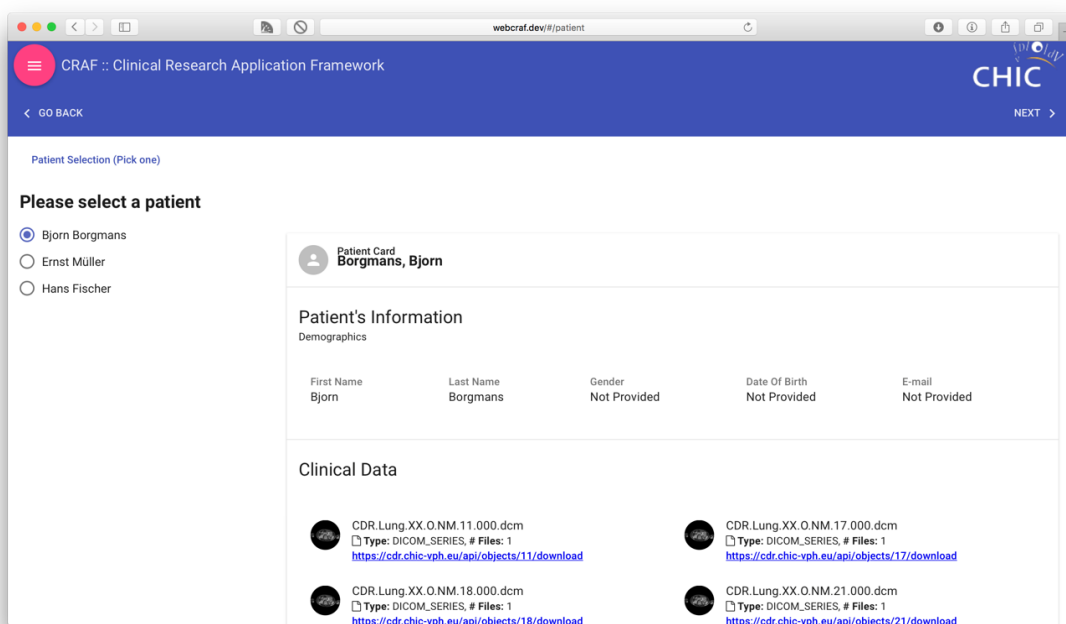
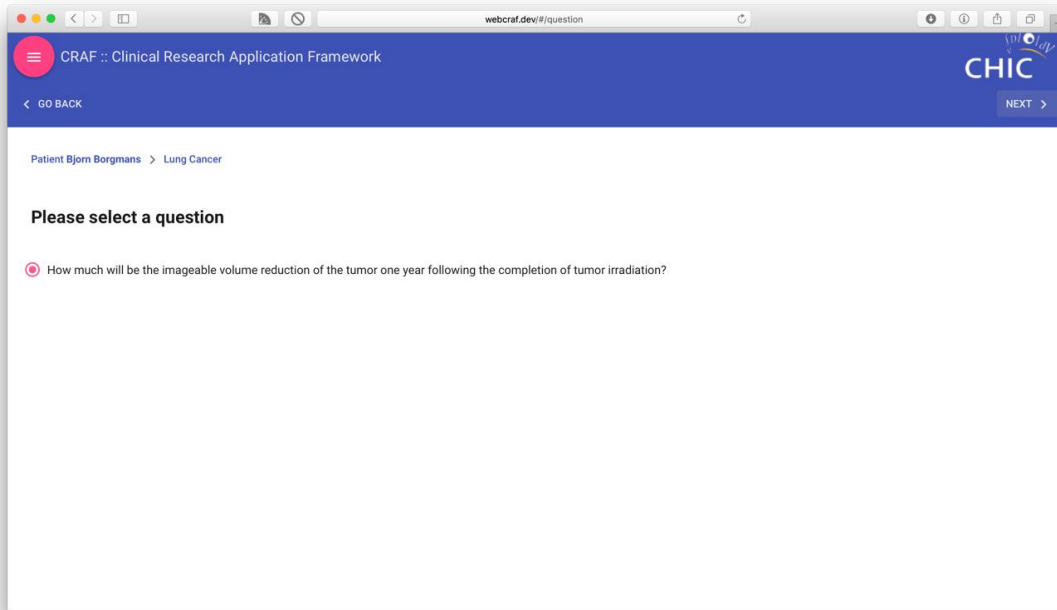
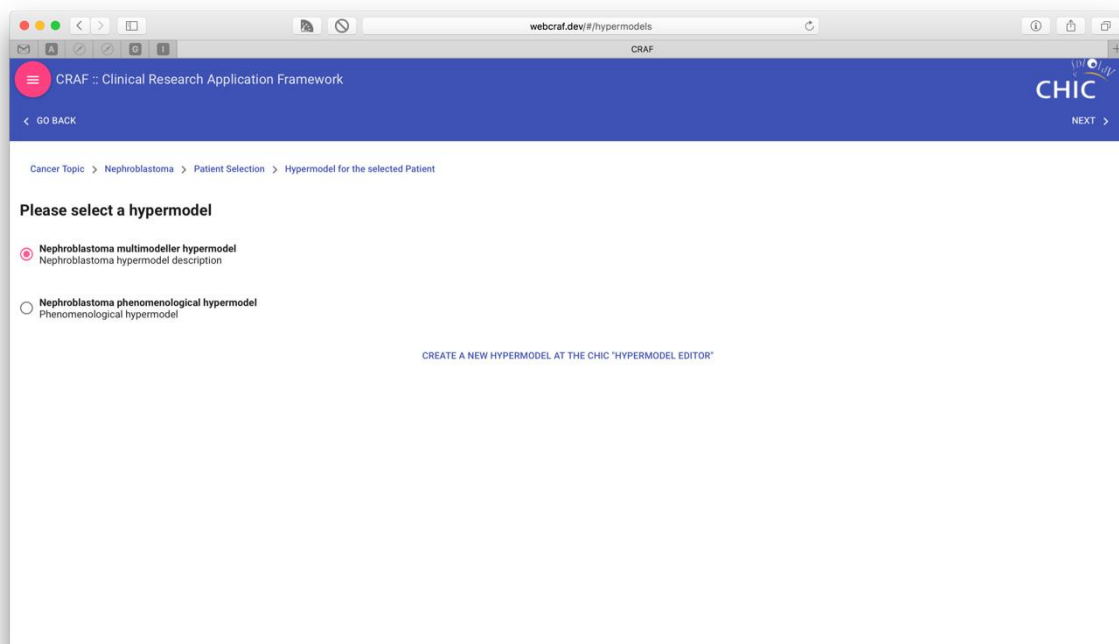


Figure 5: Patient selection



**Figure 6: Selection of question for the chosen patient**

When the patient and the question have been selected, then at the next step (Figure 7) CRAF displays all the available hypermodels that can be selected. By proceeding with the selected hypermodel, at the next step the clinician overviews the input/output parameters for the hypermodel (Figure 88), and if necessary can make any adjustments directly from the CRAF application. When everything is set, the user can proceed to the execution of the model by clicking the "Execute" button. A window appears which notifies the user about the configuration status of the hypermodel (complete or not) and about the status of its execution (successful initialization of the execution of the hypermodel or failure to start). After the user closes the dialog window, CRAF returns to the main screen, ready to configure and execute a new hypermodel.



**Figure 7: Selection of the hypermodel for the chosen patient, question and cancer domain**

**Figure 8: Overview of the input/output parameters before the execution of the chosen hypermodel for the selected patient and the chosen question**

When the execution of the selected hypermodel for the selected patient has finished, the user can look through the “History” screen of CRAF and inspect the results of the execution (Figure 9). The user can view directly on the browser all of the simple type parameters (such as numerical values, strings, images, etc.) or download any output files that cannot be displayed in the browser (such as .dat files, other binary files etc.). There is also the option to download in a single zip file all the outputs, by clicking the "Download" button. Finally, there is the possibility of creating a PDF report containing all the information about the execution, such as the patient’s name, date and time of the run, input parameters set, outputs produced, etc.

**Figure 9: Available outputs of the chosen execution from the history list. All the outputs can be downloaded as a zipped file.**

#### 4.2.1.2 Admin panel-dashboard

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

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

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

**Upload Model**

Select the hypo/hypermodel. Then press the submit button.

**Select a Model File**

Sfoglia... Nessun file selezionato.

Submit

**Workflow list**

All the workflows submitted to the Workflow Management System are listed in the table below.

Id	Title	Start Time	Finish Time	Status	
541	simone test concatenate	2016-04-06T15:25:04.988Z	2016-04-06T15:25:20.331Z	Finished	x↓
737	simo test decoupled pharmacokinetic	2016-06-07T09:41:40.135Z	2016-06-07T09:42:04.031Z	Finished	x↓
739	simo test hello world	2016-06-14T09:44:39.338Z	2016-06-14T09:44:54.916Z	Finished	x↓
749	simo test upenn unito	2016-07-15T09:00:47.243Z	2016-07-15T09:04:32.853Z	Finished	x↓
750	simo test hello world	2016-07-15T15:52:42.108Z	2016-07-15T15:52:57.099Z	Finished	x↓

Built with [Django](#), [Django-REST](#), [Bootstrap](#) and [jQuery](#). Developed and maintained by CINECA and USFD for the **CHIC** project.

**Figure 10: Admin panel-dashboard**

#### 4.2.2 Director

The Director is the component that plays a central role in the hypermodelling framework. It manages the entire process for the execution of a hypermodel (i.e. workflow) and handles authenticated requests from the end-user applications of submission, execution and monitor of workflows.

The Director delegates to the appropriate component of the hypermodelling framework each task related to a request made by the client. For instance, the identity verification is delegated to the Authentication Service, the download and upload of files from the repositories is delegated to the Storage Management Service, the execution of the hypermodel is delegated to the Workflow Management Service, the push notifications or non-blocking operations are delegated to the Message Broker service.

The description of the APIs of the Director service is reported in Appendix section 2.2.

### 4.2.3 Storage Management Service

SMS is a software component that is able to interact with different storage solutions. According to the CHIC needs, SMS implements two functionalities:

- Local File-System Storage Service
- File Transfer service

The Local File-System Storage Service is a REST web service that shows a basic interface to store, retrieve and delete files in the local file system. The files can be stored encrypted using the AES-CBC-256 **symmetric-key** algorithm. This service can be in future extended to support virtually any storage system such as Amazon S3, Lustre, HSF, to name a few.

The File Transfer Service is a web service that provides an interface to all the repositories deployed in the CHIC infrastructure, i.e. the CHIC Model Repository, the CHIC In Silico Trial Repository and the CHIC Clinical Data Repository, and to the WMS to perform file transfer operations. For instance, the SMS can transfer the hypermodel execution output results from the WMS to the In Silico Trial Repository or can transfer the input files from the Clinical Data Repository to the WMS.

The description of the APIs of SMS is reported in Appendix section 2.4.

### 4.2.4 Authentication Service

Every request sent to a VPH-HF component requires authentication, which is handled by the Authentication System developed within WP5. The Authentication system is the VPH-HF component that can be configured to allow access only to the users whose identity is verified and have the correct permissions.

The CHIC authentication system provides a brokered authentication in which an authentication broker (such as an IdP) is responsible for authenticating the users and issuing **identity tokens** to all relevant services. The user can then use such identity tokens to access the project's services. To avoid that the user needs to provide his credentials each time he accesses a different service, the authentication broker can keep the authenticated session open. This results in new identity tokens being issued automatically (Single Sign-on) for each service the user accesses as long as the authentication broker's authenticated (SSO) session is still active (see D5.2 "Security guidelines and initial version of security tools" for complete reference to the CHIC security layer).

The VPH-HF Authentication System was designed to be configurable so that VPH-HF can work with different authentication systems, such as the one adopted in CHIC. The CHIC-HF authentication system has been implemented as an extension and customization of the Django native authentication system (see section 6.1 for more details on this library).

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

The format of the CHIC framework's assertion is defined by the SAML (Security Assertion Markup Language [42]) standard. SAML is an OASIS standard that defines signed identity assertions in an XML-based format. Such an identity assertion contains the user's authentication and authorisation information.

A SAML 2.0 assertion is structured in three parts:

- authentication statements which assert that the user did indeed authenticate with the Identity Provider at a given time using a particular method of authentication.
- attribute statements which assert that the user is associated with certain attributes.
- authorization decision statements which assert the a user is permitted to perform actions on a resource with given evidence.

As said above, the client requests an identity security token to the Idp. The identity token is provided by the Secure Token Service (STS). STS provides the following services:

- **Issue:** this operation issues a new security token based on the credential provided or proven in the request. The operation accepts a request security token (RST) and returns a request security token response (RSTR).
- **Cancel:** this operation cancels a token so that it cannot be used anymore when it is no longer needed. After cancellation the STS will not renew or validate the token.
- **Renew:** a previously issued, possibly expired token, is presented and the same token is returned with new expiration semantics. The requestor must either prove authorised use of the token or be trusted by STS to issue third-party renewal requests.
- **Validate:** this operation evaluates the specified token. The result can be a status, a new token, or both.

In CHIC, VPH-HF plays the role of a REST client. Before calling one of the REST services of the CHIC platform, VPH-HF sends a SOAP request containing an RST (RequestSecurityToken) to the STS. The STS then returns the identity assertion as a SAML token, embedded in a RSTR (RequestSecurityTokenResponse). In SOAP this SAML token is passed to the calling service through a SOAP header. This is not possible in REST though as there are no SOAP headers. In REST, the HTTP headers and request line are part of the message: an HTTP header is therefore the REST equivalent of a SOAP header. A SAML token can then be passed to a REST service through the HTTP authorization header. The authorization header value should be formatted as follows: "SAML <Base 64 encoded compressed SAML token>". The SAML token is hereby compressed with the DEFLATE (zlib) [43] compression algorithm. This ensures that the token fits in the typical 4kb header size limit.

### 4.2.5 Registry

The registry service provides in real time a list of all the hypomodels/hypermodels accessible within the VPH-HF instance (for example those available for testing but not yet uploaded into the model repository), including for each of them valid input sets, t2flow description and (possibly) output sets. It also provides the automatic deployment of hypomodels stored in the Model Repository within the HF infrastructure. The hypomodels that are successfully deployed can be included in a hypermodel workflow and executed in the computational infrastructure.

### 4.2.6 Message Broker

The message broker is a component that gives the hypermodelling framework and more generally the CHIC infrastructure a common place where the components (CRAF, director, external repositories,...) can send and receive messages in an asynchronous way, effectively implementing the decoupling among the applications.



The message broker is used in the hypermodelling framework for:

- **Push notifications** (status of the hypomodel execution, status of the hypermodel execution, ...)
- **Task submissions** to an asynchronous task queue based on distributed message passing (setup of a workflow, download output from the WMS, upload output to an external repositories, ...)

For this component, **RabbitMQ**, a popular implementation of the standard Advanced Message Queueing Protocol (AMQP) was chosen [44]. For the asynchronous task queue based on distributed message passing, **Celery**, a python library that is known to work with Django and RabbitMQ, was used [45].

#### 4.2.7 Workflow Management System - Orchestrator

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

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. Among the great number of Scientific Workflow Management Systems developed in the recent years (for a recent review see [15]), Taverna was considered the most suitable workflow orchestrator for this scenario. Taverna is an open source and domain-independent Workflow Management System developed by the myGrid team 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 graphical creation, editing and running of workflows locally;
- the **Taverna command-line tool** that has the same functionalities of the Taverna Server but exposed with a command-line interface.

The Taverna server has been installed and configured in the CHIC private cloud for managing the execution of workflows started by the director. However, as the work of WP6 developed, it became 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 the final adopted solution was to develop a new component, the **MUSCLE2 Coupler**; this component acts as a second WMS in VPH-HF exposing the functionalities provided by the MUSCLE2 framework [7] to execute non-DAG workflows.

#### 4.2.8 Caching Service

The principle behind the VPH-HF Caching Service is relatively straightforward; at its core is a memorization function which maps for each model a specific set of inputs to a corresponding output set.

This function speeds up workflow requests by trading off a reduction in execution time for an increase in storage space. For each model execution request, the set of input files are hashed to create a 'fingerprint', which acts as an identifier to either, recall or store the set of output files generated by running a model with that unique set of inputs. The cache can be enabled or disabled on a per-model basis depending on need. For instance, the following situations are undesirable for caching.



- Low hit rates, input sets are rarely repeated.
- Variance in output given the same input.
- Restricted access to input or output files.

#### 4.2.8.1 Cache Integration

In this section we cover how the Caching Service fits within the current VPH-HF framework.

This is summarised in Figure 11 which highlights two additional components required for caching; a database for storage and **Cache Agent** to manage the flow control and caching logic.

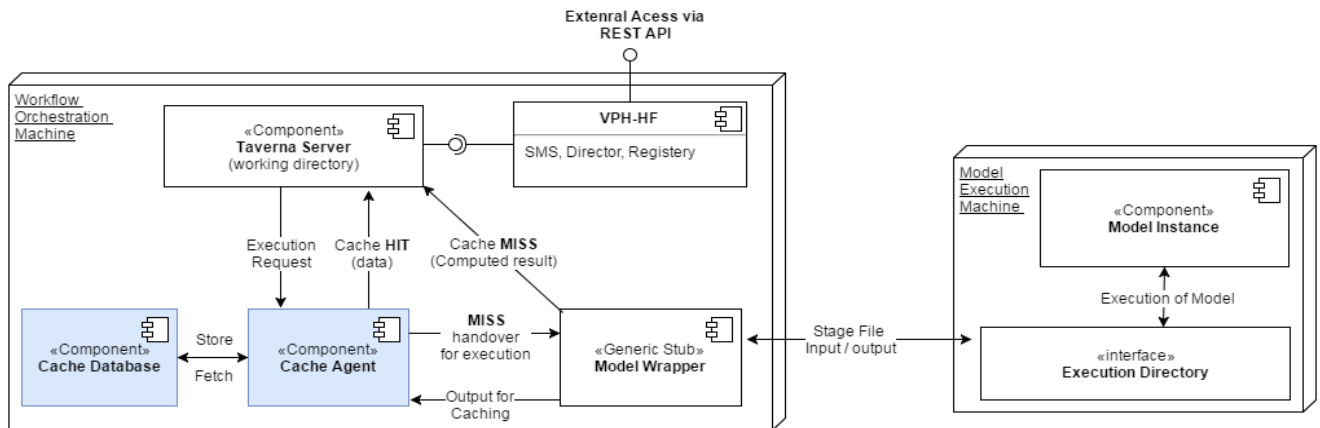


Figure 11: UML diagram of the two caching components Interaction with existing VPH-HF services.

The outer boxes represent independent virtual machines, the 'Workflow Orchestration' is a deployed instance of the Hypermodelling Framework while the Execution Machine is a remote node containing a subset of hypermodel / hypomodels. By inserting the Cache Agent between the Model Wrapper and the Taverna temporary directory an execution can be replaced with a database load when the cache detects a hit. This is possible because each input/output is represented as a file.

##### Cache Miss Flow Control:

1. Taverna Server first populates a temporary working directory with input files.
2. Then the Cache Agent hashes all input files to create a fingerprint
3. The hash or fingerprint is checked against the Cache Database and returns a MISS.
4. The Model Wrapper is executed
  - a. Input files are staged to the remote machine.
  - b. A model is executed.
  - c. The Generated output files are staged back to the Taverna temporary directory
5. The cache agent stores the output files in the cache database using the input hash as a reference
6. Workflow completes and control is handed back to VPH-HF

##### Cache Hit Flow Control:

1. Taverna Server first populates a temporary working directory with input files.
2. Then the Cache agent hashes all input files to create a fingerprint
3. The hash or fingerprint is checked against the Cache Database and returns a HIT.

4. The Cache Agent populates the temporary working directory with matching output files.
5. Workflow completes and control is handed back to VPH-HF

### 4.2.9 Surrogate Modelling Service (SUMOD)

The surrogate modelling service is a software component providing services to be consumed by applications like VPH-HF and CRAF. It is implemented mainly as Python modules exposing an API to retrieve a list of available surrogate models ranked by estimated accuracy and execution time. The default policy for the hypermodelling execution framework is to use the surrogate model available with the highest accuracy and shorter execution time below a required value. Surrogate models validated by the modeller for their use are published in the Model Repository.

The SUMOD also offers the possibility to retrain a surrogate model when new simulation data generated by the full-order model is available. It is therefore coupled with the Caching Service to build an updated dataset of all executions of the full-order model that can be used to retrain the surrogate models to improve their accuracy. When a retrain of a surrogate produces a model with a better accuracy, the modeller can validate the outcome and replace the older instance in the CHIC Model Repository with the new one. Surrogate models follow the same versioning and validation procedure implemented for the models in Model Repository as de facto they can also be considered stand alone models.

The SUMOD and Caching services have been designed to reduce simulation time via two approaches:

- Avoid redundant executions when models have been already run on the provided inputs and thereby outputs are already available
- Replace a model execution with one of its surrogate models of an acceptable accuracy and shorter computational time

In Figure 12 the interaction between SUMOD and other CHIC components is represented with UML sequence diagrams. When a request to execute a model over a given input set is received, VPH-HF checks whether this simulation has already been run and returns the previous results. This approach is implemented at a finer scale for each hypomodels in the Workflow Wrapper. It queries the Cache Agent for previous executions of the hypomodel on the provided input and if this was already run before, stored outputs are returned without replicating the execution. This approach can speedup the execution of any hypomodel belonging to a DAG hypermodel.

The main purpose of having a surrogate model is to accelerate the generation of simulation results. VPH-HF checks before submitting a model execution whether the Surrogate Modelling Service has some surrogate model of required accuracy and maximum execution time. If available, a surrogate model is run and results are returned in shorter time. In any case, a full-order model is scheduled for execution with lower priority to update the cache and training dataset for a future retrain of the surrogate model. The retrain can be automated in some cases but the modeller's validation is mandatory before including the updated surrogate model in the Model Repository. This design allows generating new surrogate models based on future methodologies just reusing the available training set.

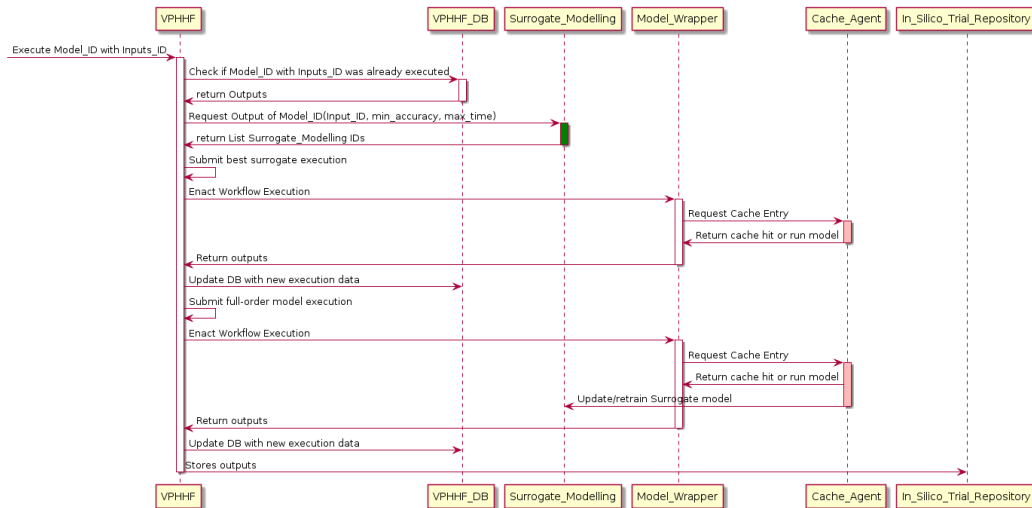


Figure 12: UML sequence diagram illustrating the interactions between the VPHHF and CHIC components when a request for a model execution arrives and the surrogate model is run. The reference model is run in the background to update the training set of the surrogate one for its retraining.

### 4.3 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.

#### 4.3.1 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 a 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.

#### 4.3.2 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.

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 executes a given procedural code.

In the next section, we will analyse the accessibility of models, meaning their execution from the WMS, in its current Taverna-based implementation, and the automatic generation of the model wrapper.

#### 4.3.2.1 Accessibility of models

**Executables** - These can be accessed by the Tool service provided by Taverna. It requires that the host machine of the executable also works as a SSH server. Examples are in the demonstration of the Nephroblastoma hypermodel available in Section 6.3.

**Web services** - Taverna provides natively the functions to access both SOAP and REST web services. In the VPHOP project, the access to XMLRPC services from Taverna was used by the definition of a specific beanshell. 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 Virtual Machine (VM) 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 model can also be executed.

#### 4.3.2.2 Automatic generation of the Model Wrapper

In VPH-HF a tool was developed to generate automatically a Model wrapper from the information stored in the Model Repository. For Static Executables a bash script to be invoked directly by the Taverna Tool service is generated. An example of bash script is reported below.

The wrapper has been logically subdivided in 9 sections (marked by a # comment) that performs sequentially:

- Setup of bash env variables
- Read values from numerical inputs in execution sandbox
- Create the temporary remote sandbox folder
- Linking executable in the sandbox
- Copy input files to remote execution folder
- Generate the command line
- Launch remotely the execution of the model
- Rename the output files to the port names
- Copy the results back to VPH-HF

```
#!/usr/bin/env bash
echo "----- VPHHF local startup code -----"
echo "PWD=$(ls -la);

# Setup env variables
COMPUTE_HOST=139.91.210.52
REMOTE_USER="vphhf"
MODEL_NAME=run_multisimul
MODELS_DIR='/media/data/vphhf/MODELS'
MUSCLE_SCRIPT_DIR='/media/data/vphhf/MODELS/muscle-cxa-rb/'
MCR_DIR='/usr/local/MATLAB/R2015b_Runtime/v90'
MCR_CONF='/home/vphhf/VPHHF-config/env-settings/MCR-R2015b-enviroment.conf'
SANDBOX_VPHHF='/home/vphhf/TAVERNA-SANDBOX/'
CWD=`pwd`
# Get unique sandbox folder name
TMP_UNIQUE=`basename $CWD`
echo "TMP_UNIQUE=$TMP_UNIQUE"
SANDBOX_TMP="${SANDBOX_VPHHF}${TMP_UNIQUE}/"
echo "SANDBOX_TMP=$SANDBOX_TMP"
EXECUTABLE_NAME='preprocessing_tool/run_preproc.sh'

# Read values from numerical inputs in execution sandbox
PAR_threshold=`cat threshold`
PAR_interpolation_spacing=`cat interpolation_spacing`
PAR_name_of_output_file=`cat name_of_output_file`
PAR_input_metaimage="input_metaimage"
```

```
echo "----- VPHHF REMOTE startup code -----"
# Create remote temporary sandbox folder
INLINE_SCRIPT='echo `whoami` @ `hostname` '
INLINE_SCRIPT="${INLINE_SCRIPT} ; mkdir -p `echo ${SANDBOX_TMP}`;"
echo "INLINE_SCRIPT=${INLINE_SCRIPT}"
ssh `echo ${REMOTE_USER}@${COMPUTE_HOST}` "${INLINE_SCRIPT}"

# Linking executable in the sandbox
EXECUTABLE_FOLDER="${MODELS_DIR}/`echo $(dirname ${EXECUTABLE_NAME})`"
echo "executable folder= $EXECUTABLE_FOLDER"
#REMOTE_SETUP_CODE="ln -s ${MODELS_DIR}/${EXECUTABLE_NAME} ; "
MUSCLE_SCRIPT_DIR="ln -s ${MUSCLE_SCRIPT_DIR}* . ; "
REMOTE_SETUP_CODE="ln -s ${EXECUTABLE_FOLDER} ${SANDBOX_TMP}"
# EXECUTABLE_NAME="./${$(basename ${EXECUTABLE_NAME})}"
echo "Workflow launched by `whoami`@`hostname`;"
ssh `echo ${REMOTE_USER}@${COMPUTE_HOST}` "${REMOTE_SETUP_CODE}"
SANDBOX_TMP="${SANDBOX_TMP}/"

# Copy input files to remote execution folder
echo "----- Copy input files to remote execution folder-----"
rsync -a --no-links -vhe ssh * "${REMOTE_USER}@${COMPUTE_HOST}:${SANDBOX_TMP}"

# Generate the command line
CLI_string = " ${SANDBOX_TMP}/${EXECUTABLE_NAME} -t ${PAR_threshold} -r
${PAR_interpolation_spacing} -o ${PAR_name_of_output_file} -i
${PAR_input_metaimage} -o preprocessing_tool/output"
echo "CLI_string = $CLI_string "

# Launch remotely the execution of the model and rename the output files to the
port names
echo "Launching remote execution in:
${REMOTE_USER}@${COMPUTE_HOST}:${SANDBOX_TMP}"
ssh `echo ${REMOTE_USER}@${COMPUTE_HOST}` " cd ${SANDBOX_TMP}; ${CLI_string} ; mv
preprocessing_tool/output.raw output_raw ; mv preprocessing_tool/output.mhd
output_mhd ;"

# Copy the results back to VPH-HF
rsync -a --no-links -vhe ssh "${REMOTE_USER}@${COMPUTE_HOST}:${SANDBOX_TMP}" .
```

### 4.3.3 Hypermodel Multiscale modelling

Multiscale modelling consists of hypomodels that have been combined or *coupled*. We distinguish in this work between two multiscale simulation methods:

- **Acyclically coupled** simulations are applications in which hypomodels are run, producing results that in turn are used as input for the execution of the next hypo models. In acyclically coupled simulations there are not hypo models that are mutually dependent during the run. These simulations can be described according to a direct-acyclic graph (DAG) workflow.
- **Cyclically coupled**: simulations have a mutual dependency and at least two hypo models run concurrently or in alternating fashion.

Figure 13 shows several schematic examples of multiscale models that use acyclic coupling and cyclic coupling. As we will see in section 6.3, it is not uncommon for multiscale models to have more than two hypomodel running concurrently: in the nephroblastoma hypermodel case, four hypo models are cyclic coupled using the MUSCLE coupling library [7].

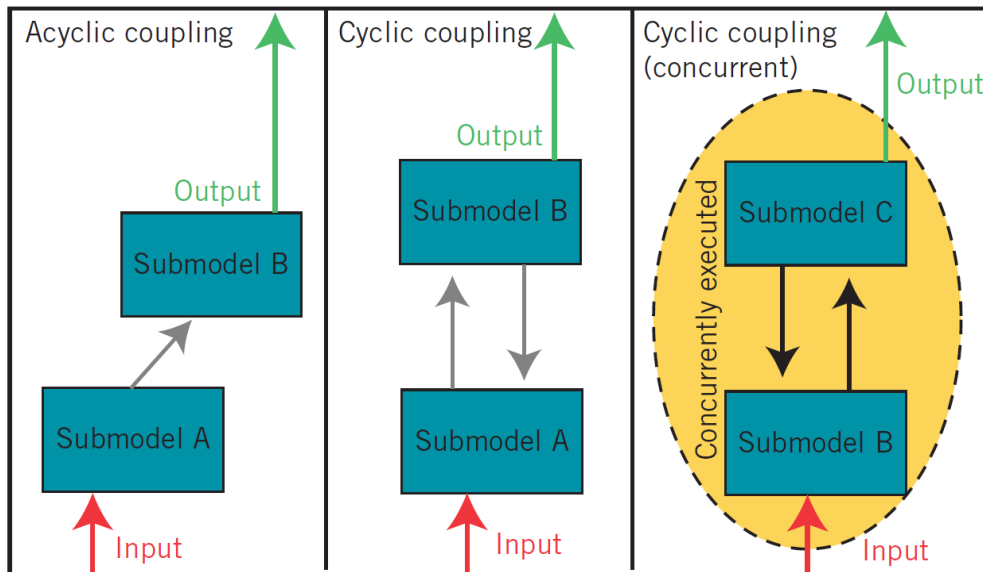


Figure 13: Multiscale simulation methods

#### 4.3.3.1 Acyclically coupled hypomodels

Acyclically coupled hypomodels are static executable models that are not mutually dependent with other codes during the execution. Acyclically coupled hypomodels are represented in a VPH-HF abstract workflow as a Taverna Tool service. According to the Component Model Generic Stub, the acyclically coupled hypomodel is wrapped by a bash script invoked by the Taverna Tool Service.

#### 4.3.3.2 Cyclically coupled hypomodels (aka strongly coupled hypomodels)

The core requirement for the original design of VPH-HF was to guarantee the best performance for the execution of hypermodels. 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 component 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, the parallel execution of two or more codes was required 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 importantly, it became clear that the modellers were not keen to embrace a massive refactoring of model 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 have experimented the extension of VPH-HF with the support for a message-passing paradigm where the models are allowed to communicate with one another. It will, potentially, reduce the amount of refactoring of 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 [[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)], QDB[[1], <http://qdb.io>]) or TCP/IP connections that would support also instances running on different machines in the CHIC Cloud infrastructure.

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.

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

- **Taverna Coupler:** an 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 [7] (developed as part of the EC-FP7 project MAPPER (<http://www.mapper-project.eu/>) to execute non-DAG workflows.

#### 4.3.3.2.1 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 a DAG. Therefore it encodes a state machine to control the models and uses a message-based paradigm to issue 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 had 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 the 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.

The model code had to be modified to read synchronization messages and command/control 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.

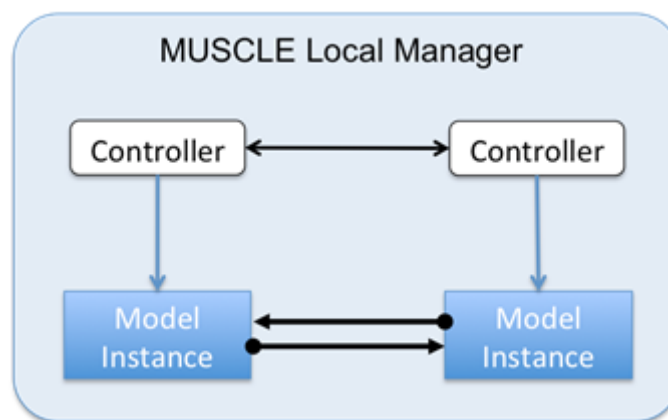
The early test on this approach revealed some drawbacks. The performance is limited by the speed of the Taverna component implementation of a loop. Using files to exchange messages introduces an overhead in tightly coupled execution scenarios where there is a frequent exchange of messages. Last, modellers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

#### 4.3.3.2.2 MUSCLE2 coupler

MUSCLE2 is a platform to execute time-driven multi scale simulations that was developed in the EC FP7 MAPPER project [www.mapper-project.eu/]. 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 MUSCLE2 implementation 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 (<http://www.qoscosgrid.org/trac/qcg>) 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 14: 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.

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 as a second WMS in two different ways not mutually exclusive.

When the workflow is only DAG or non-DAG, the straightforward solution is that Director will decide runtime depending on the workflow topology whether to use Taverna or MUSCLE.

Otherwise a hypermodel (workflow) could be hybrid, for example a DAG workflow embedding a Cyclic Directed Graph. The 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. In order to be general and to accept hybrid workflows composed by DAG workflow embedding a Cyclic Directed Graph WP7 partners opted for the second solution.

#### **4.3.3.2.3 Automatic Generation of the Muscle Configuration file**

The MUSCLE configuration file provides all the information needed in order to run a hypomodel composed by acyclically coupled hypomodels into the MUSCLE engine. The MUSCLE configuration file is written in the Ruby Language and is generated automatically by a script using the information stored in the Model Repository. An example of this file is reported below:

```
require 'fileutils'
# Environment settings
$env['preparation_steps'] = 10
$env['oncosimulator_biomechanics_time_interval']=72
$env['oncosimulator_vasculature_time_interval']=96
abort "Run 'source [MUSCLE_HOME]/etc/muscle.profile' before this script" if not
ENV.has_key?('MUSCLE_HOME')

# Reading input values stored in port files
port_output_dir = IO.read ('output_dir')
port_act_time_A = IO.read ('act_time_A')
port_vcr_time_B = IO.read ('vcr_time_B')
port_vcr_time_C = IO.read ('vcr_time_C')
port_vcr_time_D = IO.read ('vcr_time_D')
port_dt_posttreatment_scan = IO.read ('dt_posttreatment_scan')
port_act_time_B = IO.read ('act_time_B')
port_cell_cycle_duration = IO.read ('cell_cycle_duration')
port_sleep_fraction_NecrLayer = IO.read ('sleep_fraction_NecrLayer')
port_cell_kill_rate = IO.read ('cell_kill_rate')
port_sym_fraction_NecrLayer = IO.read ('sym_fraction_NecrLayer')
port_sleep_fraction_ProlifLayer = IO.read ('sleep_fraction_ProlifLayer')
port_necrosis_time_NecrLayer = IO.read ('necrosis_time_NecrLayer')
port_sym_fraction_ProlifLayer = IO.read ('sym_fraction_ProlifLayer')
port_apoptosis_time_NecrLayer = IO.read ('apoptosis_time_NecrLayer')
port_necrosis_time_ProlifLayer = IO.read ('necrosis_time_ProlifLayer')
port_no_limp_classes = IO.read ('no_limp_classes')
port_apoptosis_time_ProlifLayer = IO.read ('apoptosis_time_ProlifLayer')
port_limp_max_g0_time = IO.read ('limp_max_g0_time')
port_stem_max_g0_time = IO.read ('stem_max_g0_time')
port_diff_apoptosis_rate = IO.read ('diff_apoptosis_rate')
port_apoptosis_rate = IO.read ('apoptosis_rate')
port_stem_g0_to_g1_fraction = IO.read ('stem_g0_to_g1_fraction')
port_diff_nec_rate = IO.read ('diff_nec_rate')
port_vcr_time_A = IO.read ('vcr_time_A')
port_limp_g0_to_g1_fraction = IO.read ('limp_g0_to_g1_fraction')
port_vs_nutrient_diffusivity = IO.read ('vs_nutrient_diffusivity')
port_vs_nutrient_concentration_healthy = IO.read
('vs_nutrient_concentration_healthy')
port_vs_nutrient_consumption_rate = IO.read ('vs_nutrient_consumption_rate')
```

```
# Create Muscle Kernel Instances
model_50=
NativeInstance.new('model_50','./wilms_oncosimulator/bin/oncosimulator',args: " -
IMAGE_MHD
./wilms_oncosimulator/input_files/image_mhd/CHIC_5XIHQGQZ2GDYMIT5KON_TimePoint-
1_labels_tumour_2x2x2.mhd -IMAGE_DIR
./wilms_oncosimulator/input_files/image_raw/CHIC_5XIHQGQZ2GDYMIT5KON_TimePoint-
1_labels_tumour_2x2x2.raw -OUTPUT_DIR #{port_output_dir} -ACT_ADMIN_A
#{port_act_time_A} -VCR_ADMIN_B #{port_vcr_time_B} -VCR_ADMIN_C
#{port_vcr_time_C} -VCR_ADMIN_D #{port_vcr_time_D} -DT_post_treat
#{port_dt_posttreatment_scan} -ACT_ADMIN_B #{port_act_time_B} -Tc
#{port_cell_cycle_duration} -Psleep_Nec #{port_sleep_fraction_NecrLayer} -
CKR_FACTOR #{port_cell_kill_rate} -Psym_Nec #{port_sym_fraction_NecrLayer} -
Psleep_Prol #{port_sleep_fraction_ProlifLayer} -TN_Nec
#{port_necrosis_time_NecrLayer} -Psym_Prol #{port_sym_fraction_ProlifLayer} -
TA_Nec #{port_apoptosis_time_NecrLayer} -TN_Prol
#{port_necrosis_time_ProlifLayer} -NLIMP #{port_no_limp_classes} -TA_Prol
#{port_apoptosis_time_ProlifLayer} -TGomax_limp #{port_limp_max_g0_time} -
TGomax_stem #{port_stem_max_g0_time} -RA_diff #{port_diff_apoptosis_rate} -RA
#{port_apoptosis_rate} -PG0toG1_stem #{port_stem_g0_to_g1_fraction} -RN_diff
#{port_diff_nec_rate} -VCR_ADMIN_A #{port_vcr_time_A} -PG0toG1_limp
#{port_limp_g0_to_g1_fraction}")

model_19= NativeInstance.new('model_19','./UNIBE_BMS/BMS/bin/BMS_MUSCLE',args: "
-s ./UNIBE_BMS/CHIC-SCENARIOS-
LOCAL/NEPHROBLASTOMA/PATIENT_Demo/input/CHIC_5XIHQGQZ2GDYMIT5KON_TimePoint-
1_labels_all_uchar.mhd -a #{port_max_cell_carrying_capacity} -f ./UNIBE_BMS/CHIC-
SCENARIOS-
LOCAL/NEPHROBLASTOMA/PATIENT_Demo/input/CHIC_5XIHQGQZ2GDYMIT5KON_TimePoint-
1_labels_tumour_2x2x2.mhd")

model_37= NativeInstance.new('model_37',"forth_metabolic/metabolic",args: " -i
./forth_metabolic/CHIC_5XIHQGQZ2GDYMIT5KON_TimePoint-1_tumor_2mm.mhd")

model_41=
NativeInstance.new('model_41',"UOXF_Vasculature_Nephroblastoma/bin/vasculature",a
rgs: " -nutrient_diffusivity #{port_vs_nutrient_diffusivity} -
nutrient_concentration_healthy #{port_vs_nutrient_concentration_healthy} -
nutrient_consumption_rate #{port_vs_nutrient_consumption_rate} -input
UOXF_Vasculature_Nephroblastoma/data/inputs/CHIC_5XIHQGQZ2GDYMIT5KON_TimePoint-
1_labels_tumour_2x2x2.mhd")

# Set dynamic port connections between kernel instances
model_50.couple(model_19, 'out_total' => 'in_total')
model_50.couple(model_19, 'out_day' => 'in_day')
model_50.couple(model_41, 'proliferating_out' => 'proliferating_in')
model_50.couple(model_41, 'quiescent_out' => 'quiescent_in')
model_50.couple(model_41, 'differentiated_out' => 'differentiated_in')
model_50.couple(model_41, 'apoptotic_out' => 'apoptotic_in')
model_50.couple(model_41, 'necrotic_out' => 'necrotic_in')
model_50.couple(model_41, 'tumour_out' => 'tumour_in')
model_19.couple(model_50, 'out_phi' => 'in_phi')
model_19.couple(model_50, 'out_theta' => 'in_theta')
model_41.couple(model_37, 'Nutrient_out' => 'Nutrient_in')

model_37.couple(model_50, 'out_rate' => 'in_rate')
```

The Muscle configuration file has been logically subdivided in 4 sections (marked by a # comment) that performs sequentially:

- Setup of bash env variables
- Reading input values stored in port files
- Create Muscle Kernel instances
- Set dynamic port connections between kernel instances

## 4.4 Hypermodelling framework integration into CHIC platform

### 4.4.1 VPH-HF and the Hypermodelling editor

VPH-HF workflows are directed acyclic graphs (DAGs) where nodes represent an acyclically coupled hypomodel or a MUSCLE2 coupler, which wraps a set of models mutually dependent and strongly coupled, and edges correspond to data flows or dependencies between them.

The VPH-HF engine is based on a data flow model of execution [12, 29-34], in which each node can start computation only once all its input data are available. The VPH-HF engine delegates the responsibility for keeping track of the status of each node and for invoking its execution when the predecessor nodes have finished execution and its input data are available to Taverna. If one of the nodes is a MUSCLE2 coupler, Taverna delegates in turn the choreographer responsibility to the MUSCLE engine (see Figure 15). A similar approach based on hybrid workflows has been recently shown in [10].

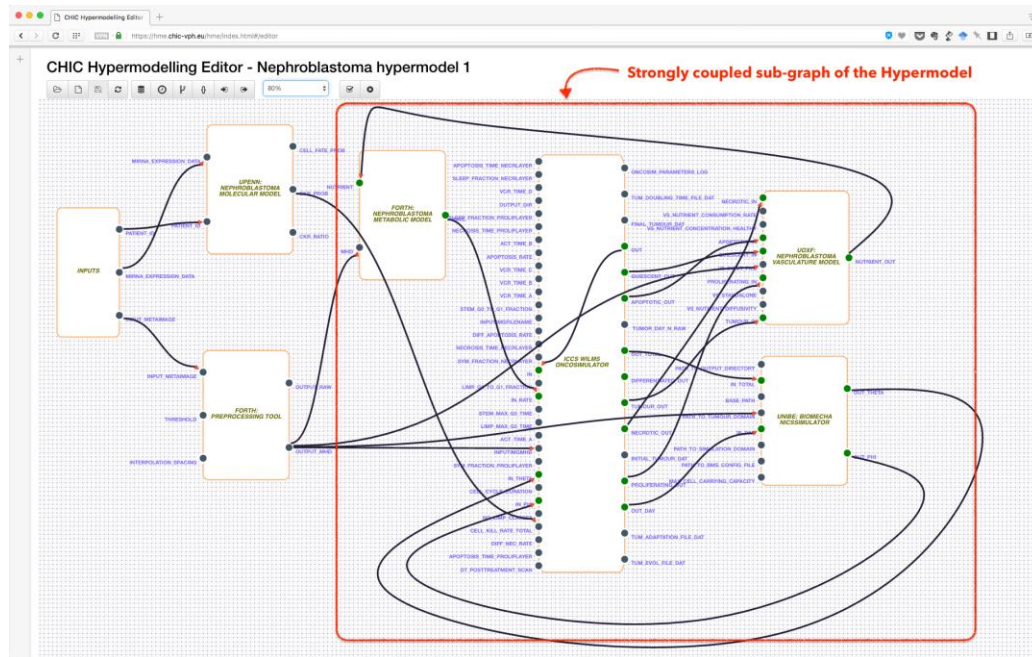
The infrastructure used in CHIC is a private cloud based on Openstack (<https://www.openstack.org/>) with VMs characterized by multiple processors. When executing workflows with independent tasks on multi-cores machines, the ones that are ready can be executed in parallel. The engine uses the hyper-threading technology to handle the execution of concurrent tasks.

A maximum number of concurrent running workflow (**RunningLimit**) is prescribed according to the hardware characteristics of the computational infrastructure. If a workflow is started and the number of running workflow exceeds the RunningLimit, the workflow is queued up to the **RunLimit** value. If the RunLimit threshold is exceeded, an error is raised.

Even if the CHIC infrastructure does not include a computer cluster, the VPH-HF technology can interact with a job-scheduler like PBS Torque to submit concurrent tasks, which in turn distributes them for execution on different cluster nodes.

A VPH-HF workflow is defined and stored in the Model Repository using the CHIC hypermodelling language. The CHIC hypermodelling language is able to describe both pure DAGs and hybrid workflows. For pure DAGs the Taverna's T2flow format can also be used.

The Hypermodelling editor supports the design of the so-called “strongly coupled” models and the simpler, acyclic-graph based ones. The user is oblivious to the underlying complexities introduced by the need of some models to work together using the “dynamic” connections that is the connections where the exchange of data happens continuously at the runtime. The Editor therefore allows the fusion of both types of models in the same hypermodel as viewed by the user. The following picture depicts an exemplar case where the area delineated with the red line is the part of the hypermodel containing the strongly coupled models.



There are therefore three types of hypermodels depending on the type of the models participating in them:

- **Type A:** Simple, no “strongly-coupled” models. In this case each hypomodel is executed sequentially and the output it produces when it completes its execution is given to next one(s).
- **Type B:** Strongly-coupled models communicating with each other. Here, the models run concurrently, and during their execution they exchange data using their so-called “dynamic” inputs and outputs. The final output of the hypermodel comes from the “static” (i.e. no dynamic) outputs of the strongly coupled models, when all of them finish their execution.
- **Type C:** Hybrid hypermodels, containing both simple and strongly connected/coupled hypomodels. This is the most complex case and the figure above shows one such example. In this case, there’s one (or more) sub-graph(s) of the hypermodel containing only strongly coupled models and another sub-graph that contains only the “un-coupled” hypomodels.

The Editor allows the design of all three of the above cases and here we will focus in the hybrid case since it’s the most general as it allows the mixing of different types of models and generalizes the other two cases.

To support the end user in building such strongly connected or hybrid hypermodels the Editor has been developed with many checks in place to make sure that no invalid connections are made. In particular, the following rules are always checked during the validation of the connections:

- Every “dynamic” output of a model is connected to a “dynamic” input of another model. This means that the strongly coupling between two models requires the commitment of both of them in order to exchange a stream of data at runtime. Therefore, we cannot have a hypermodel that contains a single “strongly-coupled” model, since, as in tango, two models are needed. Another implication is that between two strongly coupled models, a simple no-dynamic model cannot intervene.
- In reverse, each “dynamic” input of a model can only be connected to a “dynamic” output of another model. This means, for example, that the user cannot set a “default” value for this input to be used at runtime, as the model requires a flow of multiple values instead of a single one.



- Each “dynamic” connection between two models needs to be annotated with timing information by the user. This timing information provides the synchronization details between the producer (source model) and the receiver (target model) for the exchange of data using the said connection. For example, the user can specify that the model with the “dynamic” output will produce (send) a “message” of data every second and therefore the model receiving this message (i.e. the one with the “dynamic” input) should be able to accept the data at the same rate.
- In the case of hypermodels that do not contain models with “dynamic” inputs and outputs, no “cycles” are permitted. This means that only models with “dynamic” inputs can accept (dynamic) output of other models that are further down the execution line. Therefore, whenever a hypermodel is built using simple, sequentially executed models it is really a “Directed Acyclic Graph” (DAG) while cycles are only possible if the hypermodel contains two or more “strongly-coupled” models.

The above rules enable the Editor to identify the strongly connected sub-graph of the hypermodel (i.e. the part of the hypermodel containing the strongly coupled hypo models) and address the execution requirements of the three types of hypermodels explained above, using the following algorithm:

- (a) If the hypermodel does not contain any strongly-coupled hypomodel then it’s a simple one (Type A)
- (b) Otherwise, the hypermodel should contain at least two “strongly coupled” hypomodels. We randomly select one of these strongly coupled hypomodels, and perform what is known as a “breadth-first graph traversal” algorithm: Following the “dynamic” outputs of the starting strongly-coupled model, we locate all the connected “strongly-coupled” models that it directly interacts with. For each “new” strongly-coupled model found, we repeat the same process, that is, we follow its “dynamic” outputs to locate other models even further. Based on the rules described above there cannot be a simple (not strongly coupled) model among the strongly coupled ones and therefore this algorithm identifies the “connected component” / subgraph of the hypermodel that contains only strongly-coupled hypomodels. The set of the “strongly connected” hypomodels identified corresponds to a hypermodel of Type B contained as a sub-graph in the original hypermodel.
- (c) If there are still “strongly-coupled” hypomodels not visited in the graph traversal algorithm described in step (b), we repeat the same process until all the strongly-coupled hypomodels in the original hypermodel have been assigned to an “internal” hypermodel of Type B.
- (d) If there are still hypomodels left, then we have the case of Type C hypermodel. The remaining hypomodels are simple ones and participate in a sub-graph, not necessarily connected, of the original hypermodel.

Following the above algorithm, in general a given hypermodel designed in the Editor can be decomposed into zero, one, or more strongly-coupled (Type B) hypermodels and zero or one Type A hypermodels. This decomposition is important because of the different execution requirements the two types of hypermodels have. So, despite its appearance in the hypermodelling editor a given hypermodel can be stored as two or more hypermodels in the Model Repository and separately executed by the VPH-HF. This decomposition is maintained in the Editor while the rest of the platform need not be aware of this.

As an example, please consider again the hypermodel shown in the Figure above. The overall hypermodel consists of a strongly-coupled (Type B) hypermodel, shown inside the red rectangle, and a simple hypermodel. The Editor in this case requests the deployment of the strongly-coupled sub-hypermodel as a MUSCLE hypermodel, and the deployment of complete hypermodel, which contains the executions of the simple hypermodels and the MUSCLE one. Therefore, two executable

hypermodels are made available: the MUSCLE hypermodel and the complete hypermodel that uses, as a single (hypo)model, the MUSCLE one together with the no-strongly coupled hypomodels.

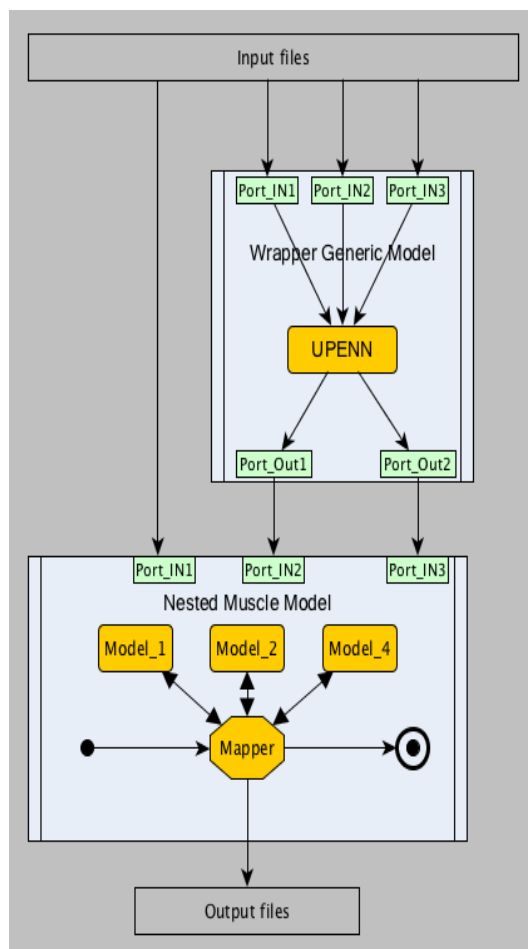


Figure 15: Hybrid Workflow Taverna-Muscle

#### 4.4.1.1 Hypermodelling language

The hypermodelling language is used by the Hypermodelling Editor to describe the composition of a hypermodel/workflow. The workflow composition is an important phase of the workflow management life cycle. Workflow composition methods can be basically divided into two main categories: textual and graphical compositions.

Textual systems are widely deployed in the business community where the workflows are defined by the developers who customize it according to the end-user requirements. In this context, the BPEL (Business Process Execution Language) is the de-facto standard for web-service based workflows [[https://en.wikipedia.org/wiki/Business\\_Process\\_Execution\\_Language](https://en.wikipedia.org/wiki/Business_Process_Execution_Language)].

On the other hand, within the scientific community, the workflows are composed by modellers/researchers that usually don't have deep skills in software programming. This is the reason why graphical representations of workflows are more widely used.

CHIC is clearly an example of the second scenario so a hypermodelling editor has been developed to help the modellers in the design process of a hypermodel.

In the CHIC hypermodelling framework a workflow has a graphical representation, useful for the modellers, and a machine-processable representation that varies according to the WMS. Taverna 2.x uses an XML-based DAG format called T2Flow. In order to decouple the hypermodelling editor from



the WMS we decided to use an intermediary language that is generated by the hypermodelling editor and translated by Director to the language used by the WMS.

Several mark-up languages have been proposed in literature in the Biology domain: CellML [36], developed out of the cardiac modelling community, Systems Biology Mark-up Language (SBML) [35], FieldML [37], a mark-up language primarily for modelling physiological structures and their physics, the In Silico Mark-up Language (ISML) [38] developed by the Japanese Physiome project. These languages have been designed for modelling biological phenomena through pure mathematical description. Each language being based on MathML restricts their expressivity in modelling and are not suitable in the multiscale field where the approach needs more algorithmic descriptions and the capability to describe the coupling of many sub-models, usually originating from different fields of science.

To describe the implementation of cancer models in an abstract manner and to be able to couple and reuse models developed from different Research Institutes, an extensive literature survey revealed two mark-up languages that address the specific domain of cancer/multiscale modelling: TumorML [13], a mark-up language developed within the TUMOR project to describe computational cancer models, and MML, proposed recently by A. Hoekstra et al [5, 1, 2] within the MAPPER project. The Multiscale Modelling Language (MML) aims specifically at specifying the architecture of a multiscale simulation program.

The XML version of MML, called xMML, fits all the CHIC needs and has been adopted as the intermediary language between the Director service and the Hypermodelling Editor.

#### 4.4.2 VPH-HF and CRAF

The Clinical Research Application Framework is the end user application used by Clinicians to run the hypermodels developed in the CHIC project: Nephroblastoma, Non Small Cell Lung Cancer, Glioblastoma, and Prostate Cancer. The coupling between VPH-HF and CRAF is shown in Figure 3.

CRAF communicates with the Director of the VPH-HF server using the SAML standard data format over the HTTPS protocol in order to submit and execute a hypermodel. This is a synchronous communication over the HTTP protocol where the response contains the identification (UUID) of the newly created run of the corresponding hyper model. On the other hand, since the exact completion time point is not known, and a continuous “polling” mechanism would be too inefficient, the execution status information is sent asynchronously. In particular, CRAF receives back notifications from VPH-HF about the status of the hypermodel (OPERATING, FINISHED\_SUCCESSFULLY, FINISHED\_ERRONEOUSLY) with RabbitMQ as intermediary. Following this approach CRAF and VPH-HF are operationally decoupled. VPH-HF sends all the notifications to RabbitMQ using a topic and “durable **exchange**” named “vphhf” with a **routing key** set to “workflow.<workflow\_uuid>.status”, where <workflow\_uuid> is a unique identifier of the workflow run. CRAF instead creates a message queue bound to the ‘vphhf’ exchange with a binding key set to “workflow.\*.status” in order to listen to the status of all the workflows (for the definitions of the technical terms see [45]).

#### 4.4.3 VPH-HF and Authentication/authorisation

The CHIC authentication system provides a brokered authentication in which an authentication broker (such as an IdP) is responsible for authenticating the users and issuing **identity tokens** to all relevant services. The identity token is provided by the Secure Token Service, and VPH-HF is a non-browser REST client of the STS service. The two services communicate using the SAML standard data format over the HTTPS protocol. VPH-HF calls the STS service for issuing a new security token based on the user credentials or for validating a token in authenticated request.

#### **4.4.4 VPH-HF and the CHIC repositories**

In CHIC we have three repositories: the Model repository where the models used for the construction and the execution of hypermodels are stored, the In Silico Trial Repository where in-silico trial are stored, and the Clinical Data Repository which stores clinical data.

VPH-HF communicates with all the three repositories using the SAML standard data format over the HTTPS protocol. The Storage Management Service is the component of the hypermodelling framework that downloads clinical data from the Clinical Data Repository during the setup of a workflow and uploads the output result of a hypermodel execution in the In Silico Trial Repository.

Director updates the status of the Trial Experiment in the In Silico Trial Repository and retrieves from the Model Repository the information for the automatic generation of the wrapper and muscle configuration scripts and the binaries to be deployed in the execution framework.

## 5 Hypermodel execution

### 5.1 *Hypermodel execution scenario*

The most common scenario in CHIC is that 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 the CRAF web interface or the CRAF desktop application the user selects an existing hypermodel, or from the hypermodelling editor composes available hypomodels into a new hypermodel and then requests its execution.

CRAF creates in the In Silico Trial repository a new subject and a new experiment and send all the information needed to Director to submit a new workflow: subject id, experiment id, hypermodel description url, and baclava url. All the requests sent to a VPH-HF component are authenticated. Director asks SMS to download the hypermodel description (in xMML/t2flow format) from the Model repository and sends it to Taverna to submit a new workflow execution. CRAF is notified by Director if the workflow has been successfully created. In the next step, CRAF requests to Director the execution of the hypermodel and is notified together with the In Silico Trial repository of the start of the execution. Director submits to a celery task queue the setup and execution of the workflow. When the task is ready, the baclava file is downloaded from the Model repository and parsed to check the presence of URLs. If a URL is found, the corresponding file is downloaded and uploaded to Taverna. All the other input values in the baclava are set in Taverna. When the setup is finished, Director triggers the workflow execution and starts polling Taverna for the status of the execution. During the workflow execution, the wrapper of each hypomodel is run. In the paragraph 4.3.2.2 we described in details the instructions executed by the model wrapper. When the execution of the workflow is finished, Director asks SMS to download the output files from Taverna in zip format and upload them in the In Silico Trial using the subject id previously given by CRAF. CRAF and the In Silico Trial repository are finally notified of the termination of the hypermodel execution and the user's dashboard is updated.

All the steps that occur in the hypermodel execution scenario are collected in a sequence diagram shown in Figure 16. In the next section, we will concentrate on the data flow during of a workflow among the components of the CHIC infrastructure.

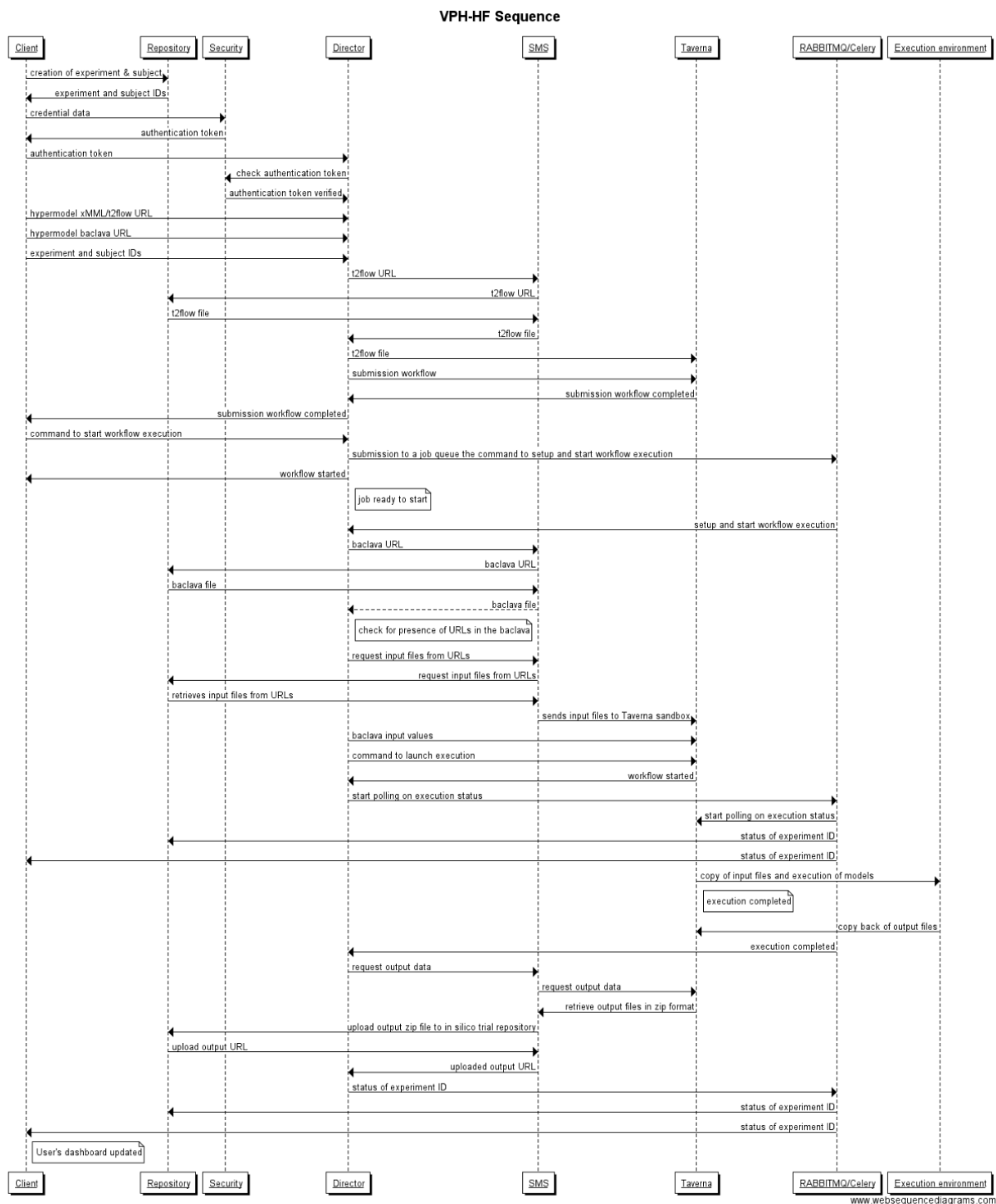


Figure 16: hypermodel execution sequence diagram

## 5.2 Data flow

Most of the hypomodels developed in CHIC require files as input: if these files are patient data, they are stored in the Clinical Data Repository, otherwise (i.e. parameters for a hypomodel execution) they are stored in the Silico Trial Repository. These files need to be transferred into the execution framework in order to be read by the hypomodels during the execution. At the same time at the end of the execution, the output results have to be stored back in the repositories. It is clear that a complex data flow occurs inside the cloud during the life of a workflow. This is shown in Figure 17.

The starting point of the data flow of a workflow is a **CHIC repository**. The input files of the workflow are downloaded from the CHIC repository by the Storage Management Service temporarily in the server and uploaded in the Taverna Server using the available API. Taverna stores them in a unique folder for each workflow, called from now on **workflow sandbox**. When the execution of the workflow is triggered, Taverna will create a unique folder for each task of the workflow and replicate there the input port files, called from now on **task sandbox**. If the uploaded files are an input port of the task, a symbolic link for each of them is created in the task sandbox. The next step is accomplished by the model wrapper that replicates the task sandbox in the computational infrastructure (**remote task sandbox**), and transfers the files there. At the end of the model execution, the wrapper copies back the output files of the execution in the task sandbox. When the workflow is finished, the output of the workflow is compressed in the zip format and uploaded in the In Silico Trial repository. Finally, the workflow sandbox and task sandboxes are deleted.

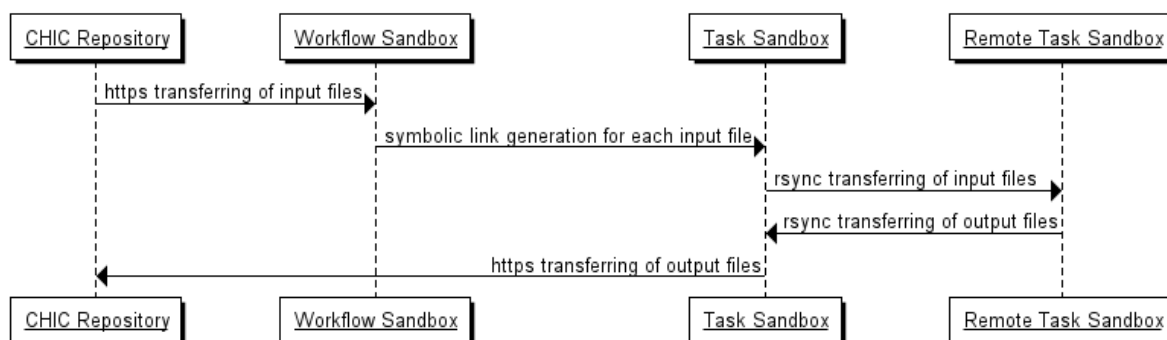
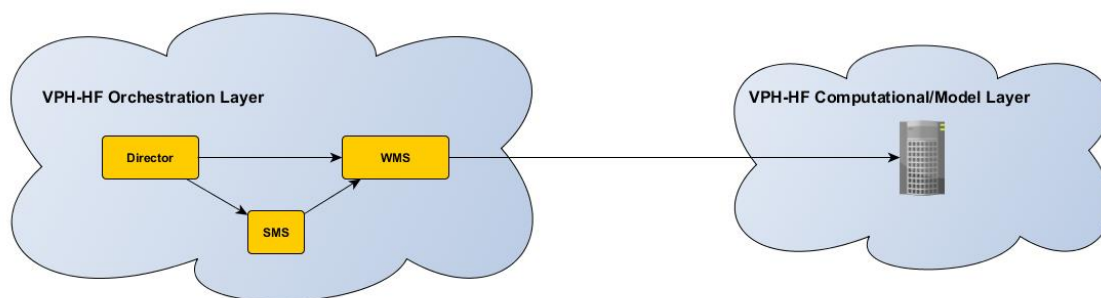


Figure 17: Data flow of a workflow in the CHIC infrastructure

## 6 CHIC-HF deployment

The production nodes are hosted by FORTH in the CHIC private cloud and are configured consistently with the technical specifications and to comply with the other software components specifications highlighted in deliverable D5.1.1 “The CHIC technical architecture – initial version”). The assumption underlying the configuration choice is that in the CHIC deployment (CHIC-HF), the Hypermodelling Framework will be deployed in a private Cloud in a scenario where the computational resources are located in the same institution’s computational facility. In order to test more complex scenarios and allow a better insight in terms of analysing the framework performance during the development, two test nodes with different hardware configurations, one hosted in CINECA and one in USFD, have been created. For more information about the hardware and software configurations of the test nodes, see D7.2 “First release of the hypermodelling framework deployed on test nodes”.

As described at the beginning of this document, two different abstraction layers are identified in the deployment of the framework in the CHIC private cloud: 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, 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. In the CHIC production deployment the two layers are decoupled and deployed on different machines/nodes. The reason is that the Computational/Model layer may require substantial computational resources with respect to the Orchestration Layer that are normally available only on HPC clusters. For the first prototype release we opted to use a VM with a high number of cores and huge amount of memory in order to run more than one model concurrently and to satisfy the hardware requirements of all the models that will be developed during the CHIC project (Figure 18 - Top). In the future we can migrate the Computational/Model Layer in a HPC cluster and use a job-scheduler to distribute the task among the cluster nodes (Figure 18 - Bottom). Therefore it is envisaged that some models could be installed on a dedicated machine and/or the user will have the chance to select the most appropriate target machine for the execution of his/her hypermodel.



**Figure 18 - (Top)** The software layers on the FORTH production node. The orchestration layer on top comprises components developed by USFD and CINECA, the WMS is provided by a third party engine, the Taverna Server. The computational/Model Layer is composed by the models currently provided by WP6 partners deployed on a single virtual machine in a cloud.

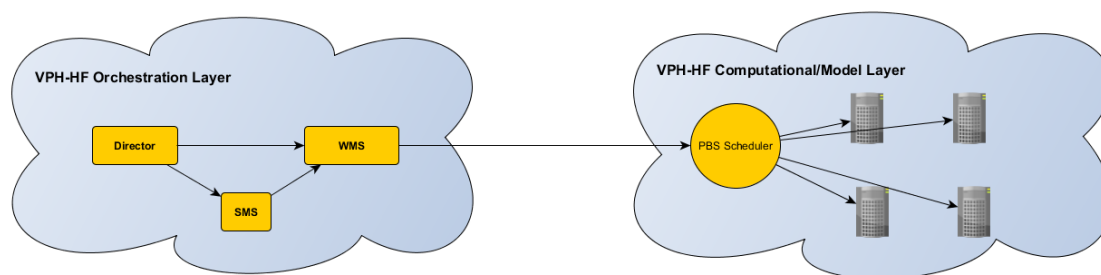


Figure 18 - (Bottom) Complex future scenario where the Computational/Model Layer is a cluster with a PBS scheduler that distributes the work among the computational nodes.

## 6.1 VPH-HF Orchestration Layer deployment

The FORTH production node hosting the VPH-HF Orchestration layer is configured to support the VPH-HF components. 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 16 virtual CPU, 16GB of RAM memory, 100GB storage space.

The VPH-HF code is hosted in a Github repository (<https://github.com/INSIGNEO/VPH-HF>) and will be released under the APACHE license at the end of the CHIC project.

The code can be installed automatically from scratch using a script. We provide also a vagrant script to create a fully-fledged VM ready to be used.

The VPH-HF framework contains a list of web-based applications, which rely on several software dependencies. They are subdivided in Figure 19 in three tiers according to the multi-tier architecture (presentation layer, logic layer, data layer).

- **Presentation layer:** the presentation layer consists of the web interface (the admin panel/dashboard) where the user can graphically interact with the backend VPH-HF services. In order to enhance the user experience, we built our GUI on top of the two frameworks Bootstrap and JQuery.
- **Logic Layer:** the logic layer controls the application functionalities. All the VPH-HF server components except the WMS are Django-REST based application served by an Apache HTTP server. The Workflow Management Service is composed by a Taverna Java Servlet served by a Tomcat Web Server. The message broker of the framework is carried out by RabbitMQ which delivers asynchronous tasks to the job queue Celery.
- **Data Layer:** we have chosen MySQL for storing the workflow data model, the file data model for the SMS internal storage and the registry data model. The files are stored separately from the DB in a file system storage area.



Figure 19: VPH-HF software dependencies

In the next paragraphs we list the most important features of the VPH-HF software dependencies in order to motivate our technology choices.

### 6.1.1 Presentation Layer

#### Bootstrap

Bootstrap is a free and open-source front-end web framework for designing websites and web applications (<http://getbootstrap.com/>). It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.

#### jQuery

jQuery is a fast, small, and feature-rich JavaScript library that simplifies the client-side scripting of HTML (<https://jquery.com/>, <https://en.wikipedia.org/wiki/JQuery>). It makes things like HTML document traversal and manipulation, event handling, animation, and building Ajax applications much simpler with an easy-to-use API that works across a multitude of browsers.



## 6.1.2 Logic Layer

### Django

Django is a free and open source web application framework, written in Python, which follows the model–view–controller (MVC) architectural pattern. It encourages rapid development and clean, pragmatic design. It allows high-performing, elegant Web application building.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and “pluggability” of components, rapid development, and the principle of “don't repeat yourself (DRY)”. Python is used throughout, even for settings, files, and data models.

The core Django framework consists of:

- An object-relational mapper, which mediates between data models (defined as Python classes) and a relational database (“Model”).
- A system for processing requests with a web templating system (“View”).
- A regular-expression-based URL dispatcher (“Controller”).
- An optional automatic administrative CRUD interface that is generated dynamically through introspection and configured via admin models.
- A lightweight, standalone web server for development and testing.
- A form serialization and validation system, which can translate between HTML forms and values suitable for storage in the database.
- A template system that uses the concept of inheritance borrowed from object-oriented programming
- A caching framework, which can use any of several cache methods.
- Support for middleware classes, which can intervene at various stages of request processing and carry out custom functions.
- An internal dispatcher system, which allows components of an application to communicate events to each other via pre-defined signals.
- An internationalization system, including translations of Django's own components into a variety of languages.
- A serialization system, which can produce and read XML and/or JSON representations of Django model instances.
- A system for extending the capabilities of the template engine.
- An interface to Python's built in unit test framework.

### Django-REST

Django-REST (<http://www.django-rest-framework.org/>) is a powerful and flexible framework built on top of Django that makes it easy to build and test REST Web APIs due to the Web Browsable API technology. Django-REST makes exposing models easy but at the same time is customizable all the way down.

Some of the features of Django-REST framework are:

- Authentication Policies including packages for OAuth1a and OAuth2
- Serialization that supports ORM and non-ORM data sources
- Extensive documentation, well-tested and great community support

## Taverna

Taverna ([https://en.wikipedia.org/wiki/Apache\\_Taverna](https://en.wikipedia.org/wiki/Apache_Taverna)) is an open source software tool for designing and executing workflows, initially created by the myGrid project under the name *Taverna Workbench*, now a project under the Apache incubator. Taverna is used by users in many domains, such as bioinformatics, cheminformatics, medicine, astronomy, social science, music, and digital preservation.

Taverna Workbench provides a desktop authoring environment and enactment engine for scientific workflows. The Taverna workflow engine is also available separately, as a Java API, command line tool or as a server served by Tomcat.

Some of the features are:

- Taverna workflows can invoke general SOAP/WSDL or REST Web services, and more specific SADI, BioMart, BioMoby and SoapLab Web services. It can also invoke R statistical services, local Java code, external tools on local and remote machines (via ssh), do XPath and other text manipulation, import a spreadsheet and include sub-workflows.
- Taverna Workbench includes the ability to monitor the running of a workflow and to examine the provenance of the data produced, exposing details of the workflow run as a W3C PROV-O RDF provenance graph, within a structured Research Object bundle ZIP file that includes inputs, outputs, intermediate values and the executed workflow definition.
- Taverna includes the ability to search for services described in BioCatalogue to invoke from workflows. However, services do not need to be described within BioCatalogue to be included in workflows as they can be added from a WSDL Web Service description or entered as a REST URI pattern.
- Taverna also includes the capability to search for workflows on myExperiment [47]. My experiment is social website for sharing among researchers research objects such as Taverna workflows [48]. The Taverna Workbench can download, modify and run workflows discovered on myExperiment, and also upload created workflows in order to share them with others using the social aspects of myExperiment.
- Taverna workflows do not need to be executed within the Taverna Workbench. Workflows can also be run by:
  - a command line execution tool,
  - remote execution server that allow Taverna workflows to be run on other machines, on computational grids, clouds, from Web pages and portals,
  - online workflow designer and enactor OnlineHPC.
- Taverna allows pipelining and streaming of data. This means that services downstream in the workflow can start as soon as the first data item is received, without waiting for the whole data list to become available from upstream services and iterations. Taverna services execute in parallel when possible, as Taverna workflows are primarily data-driven rather than control-driven.

## MUSCLE

MUSCLE couples heterogeneous computational multiscale models of multiple disciplines that are normally developed in different research institute. Its aim is to provide a uniform platform to implement sub-models using varying programming languages, execute them on heterogeneous machines and couple them across different networks. In addition, it considers the temporal and spatial scale at which the sub-models are operating.

The library contains Java, C and C++ APIs, allowing developers to easily use Java code or C, C++ or Fortran as a language of choice for model implementation [<http://www.mapper-project.eu/web/guest/muscle>].

#### RabbitMQ

RabbitMQ (<https://www.rabbitmq.com/>) is messaging broker that implements the standard Advanced Message Queueing Protocol (AMQP) [<https://www.amqp.org/>]. AMQP defines an efficient and flexible publish/subscribe interface that is independent of the data model.

Messaging enables software applications to connect and scale. Applications can connect to each other, as components of a larger application, or to user devices and data. RabbitMQ gives applications a common platform to send and receive messages in an asynchronous way and a safe place to store messages alive until they are received. This allows the decoupling of applications.

#### Celery

Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well.

The execution units, called tasks, are executed concurrently on a single or more worker servers using multiprocessing, Eventlet, or gevent. Tasks can execute asynchronously (in the background) or synchronously (wait until ready). Celery can be configured to use RabbitMQ as messaging broker.

#### Apache HTTP Server

The Apache HTTP Server (<https://httpd.apache.org/>) ("httpd") was launched in 1995 and it has been the most popular web server on the Internet since April 1996. It has been developed and maintained in the open-source Apache HTTP Server Project and works in modern operating systems including UNIX and Windows. The target of the project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards.

#### Apache Tomcat

The Apache Tomcat software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. The Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket specifications are developed under the Java Community Process. Apache Tomcat can be used to connect client applications with Taverna using the HTTP protocol, de facto working as a HTTP server.

### **6.1.3 Data Layer**

#### MySQL

MySQL is the world's most popular open source database. Key features of MySQL are:

- **Scalability and Flexibility.** The MySQL database server provides the ultimate in scalability, sporting the capacity to handle deeply embedded applications with a footprint of only 1MB to running massive data warehouses holding terabytes of information. Platform flexibility is a stalwart feature of MySQL with all flavours of Linux, UNIX, and Windows being supported. And, of course, the open source nature of MySQL allows complete customization for those wanting to add unique requirements to the database server.
- **High Performance.** With high-speed load utilities, distinctive memory caches, full text indexes, and other performance-enhancing mechanisms, MySQL offers all the right ammunition for today's critical business systems.
- **High Availability.** MySQL offers a variety of high-availability options from high-speed master/slave replication configurations, to specialized Cluster servers offering instant failover, to third party vendors offering unique high-availability solutions for the MySQL database server.

- Robust Transactional Support. MySQL offers one of the most powerful transactional database engines. Features include complete ACID (atomic, consistent, isolated, durable) transaction support, unlimited row-level locking, distributed transaction capability, and multi-version transaction support where readers never block writers and vice-versa. Full data integrity is also assured through server-enforced referential integrity, specialized transaction isolation levels, and instant deadlock detection.
- Web and Data Warehouse Strengths. Features like main memory tables, B-tree and hash indexes, and compressed archive tables that reduce storage requirements by up to eighty-percent make MySQL a strong standout for both web and business intelligence applications.

## 6.2 Computational/Model Layer deployment

The node/VM in FORTH that hosts the Computational/Model Layer has been configured in order to run the models provided by partners of the WP6 and described in deliverable D6.3 “Initial Standardized Cancer Hypermodels” [46]. The models were first provided as monolithic hypermodels whose hypomodels could not be isolated and run independently. WP7 and WP6 teams collaborated to decouple the hypomodels composing each hypermodel and to verify implications in performance. Some technical solutions have been provided and analysed to solve the problem. The adopted solution is to abstract the hypo models as “*communicating processes*”: they are not passive anymore, 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 orchestrator and the message passing library have been provided in the MUSCLE2 environment.

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

- Gross Component Models Based on Simplified Universal Growth Laws (UNITO)
  - Format: Matlab® interpreted code.
  - Third-party software/libraries: Matlab®
- Molecular Model (UPENN)
  - Format: Python interpreted code.
  - Third-party software/libraries: python, COPASI (<http://copasi.org/>)
- PreProcessingTool (FORTH)
  - Format: Python interpreted code.
  - Third-party software/libraries: python, anaconda (<https://www.continuum.io/>)
- The Wilms Oncosimulator (ICCS)
  - Format: Binary executable (C++)
  - Third-party software/libraries: MUSCLE
- The Biomechanics Simulator (UBERN)
  - Format: Binary executable (C++)
  - Third-party software/libraries: coupled with external software FEBio (<http://febio.org/>)
- Vascular Angiogenesis Component Model (UOXF)
  - Format: Binary executable (C++); the executable is standalone (linked to included shared libraries using RPATH/\$Origin).
  - Third-party software/libraries: built using Chaste (<http://chaste.cs.ox.ac.uk>). Chaste itself depends on Boost, PETSc, VTK, Xerces-c and XSD.
- The Metabolic model (FORTH)
  - Format: Binary executable
  - Third-party software/libraries: N. A.

- The Lung Oncosimulator (ICCS)
  - Format: Binary executable (C++)
  - Third-party software/libraries: MUSCLE
- NSCLC machine learning based response predictor (ICCS)
  - Format: Matlab® interpreted code
  - Third-party software/libraries: Matlab®

## 6.3 The CHIC Wilms Oncosimulator hypermodel example

### 6.3.1 Introduction

In this section we describe the Wilms Oncosimulator hypermodel which is the reference use case for the development of the hypermodelling infrastructure. The Wilms Oncosimulator hypermodel simulates the nephroblastoma tumour growth and response to pre-operative chemotherapy. It models cancer cell multiplication, cellular response to treatment and spatial expansion (as described analytically in Deliverable 6.3 [46]). It consists of the following six executable hypomodels:

- **Molecular component model (UPENN):** The molecular component is an integrated cellular signaling and trafficking model that combines receptor (EGFR) mediated Ras/MAPK and PI3K/AKT pathways which are important regulators of cell fate with TP53 mediated DNA damage response pathway. This integrated model takes as input miRNA expressions and treatment information for a specific patient and calculates the cell kill probability. This patient-specific cell kill probability is then passed to the Wilms oncosimulator. Under the hood, this model combines a continuous time process modeled by ordinary differential equations with a discrete time process modeled by boolean state transition rules through dynamic information exchange at each time step of the process with the larger biological timescale. Cell to cell timescale variations, tissue heterogeneity etc are taken into account by repeating the simulations over an ensemble of such systems with different model parameter/input conditions and calculating the average quantities.
- **Image pre-processing tool (FORTH):** The image pre-processing tool defines the tumour morphology by creating a metaimage that is provided as input to some of the hypermodels (oncosimulator, BMS,...). In the produced metaimage, different subregions of interest are noted with distinct integer labels (e.g. 255 for tumour area, 0 for normal tissue). During the preprocessing procedure, the input image is resampled (using interpolation methods) in order to create an isotropic space with cubic voxels, where the size of the voxels is specified as an input to the tool (usually 1mm1 or 2mm size). Furthermore, in order to assist some hypermodels to reduce computational resources, the provided three-dimensional representation corresponds to a 'region of interest' of the entire image scan (cropping), which is centered on the tumour, with user defined space around it in order to permit growth simulations.
- **Wilms oncosimulator (ICCS):** The Wilms Oncosimulator [Stamatakis et al., 2011; Georgiadi et al., 2011] models cancer cell proliferation, response to combined chemotherapy treatment of actinomycin and vincristine and spatial evolution of the tumour. It is a cellular automata model consisting of equivalence classes and decision calculators which set the evolution and interaction rules between the classes. The Oncosimulator addresses visible clinical tumours and has been designed to incorporate patient specific data such as imaging, histopathological, molecular and treatment data.
- **Biomechanics (diffusion and mass effect) component (UBERN):** The Biomechanical Simulator (BMS) is a component-model for the simulation of bio-mechanical aspects of macroscopic tumour growth. It relies on the Finite Element Method (FEM) to compute mechanical stresses and strains resulting from tumour growth or shrinkage in a patient-

specific anatomy. The resulting information (3D stress, strain, displacement and pressure maps) can be used as input to other hypo-models. In the Wilms Oncosimulator Hypermodel, BMS derives the most likely direction tumour growth from computed pressure maps. This information is exchanged with the Wilms oncosimulator to guide the spatial evolution of tumour growth and shrinkage.

- **Vasculature / angiogenesis component (UOXF):** The Vasculature component models the delivery, consumption and diffusion of chemical species in a tumour as a function of a temporally evolving vessel volume fraction and cell population. It solves the associated linear elliptic partial differential equations for transport on a regularly spaced lattice using the Finite Difference Method. In the Wilms Oncosimulator Hypermodel the Vascualture component receives the cell population from the Wilms oncosimulator hypomodel, calculates the 3D glucose concentration field assuming a fixed vessel volume fraction and plentiful glucose supply in non-tumour tissue. It then passes the field to the Metabolic network hypomodel.
- **Metabolic network hypomodel (FORTH):** The tumor microenvironment significantly affects the metabolic activity and rewiring. As tumor grows well-vascularized regions providing sufficient nutrients to cancer cells can coexist with nutrient-limited regions within the tumor mass. The metabolic hypomodel utilizes the generic genome-scale human metabolic network reconstructions and describes the metabolic activity of the chemical reactions at flux level using the Flux Balance Analysis constrained-based method. The model assumes that cancer cells are under a selective pressure to increase their proliferation rate. At each time interval, the glucose concentration is estimated at every position in the computational grid through the vasculature hypomodel. The spatiotemporal-dependent inflow of glucose flux is constrained by the Michaelis-Menten kinetics model and an instantaneous optimization problem is solved for each cell position and time point. The metabolic model provides information regarding the proliferation rate given the available glucose that is then used from the Oncosimulator to update its state.

The hypermodel communication scheme is depicted in Figure 20 (retrieved from D6.3 [46]).

### 6.3.2 Building and deployment in the hypermodelling framework

The building and deployment of a hypermodel is a complex process that requires the usage of many services developed in the CHIC hypermodelling infrastructure.

The first step is the creation of the hypomodels composing the hypermodel and their registration in the Model Repository. For the Wilms Oncosimulator hypermodel, the six hypomodels described above have to be registered in the Model Repository together with the executables and libraries, the command-line arguments with the default values, the name of the executable and the kind of the model (for a reference on how to configure a hypomodel in the Model Repository see D8.4 [49]).

Once all the hypomodels are registered in the Model Repository, the modeller can start building the hypermodel in the hypermodelling editor by composing the available hypomodels. The graphical final result for the Wilms Oncosimulator hypermodel is shown in Figure 21. A xMML description of the hypermodel is also generated and stored in the Model Repository, which is required by VPH-HF for the execution.

In the next step, the binaries and library dependencies are deployed in the computational infrastructure together with the wrappers and muscle configuration files. As we explained in the section 4.3, the acyclically coupled hypomodels (Preprocessing Tool and Molecular model) are wrapped with bash scripts (for an example see paragraph 4.3.3.2.1) while the cyclically coupled hypomodels (Wilms Oncosimulator, Biomechanics Simulator, Metabolic model and Vasculature model) are wrapped using a muscle configuration file (for an example see 4.3.3.2.3). As we already wrote in section 4.3, the wrappers are generated in an automatic way.



At this point, the hypermodel can be run in VPH-HF by providing the hypermodel description in xMML format and an input set.

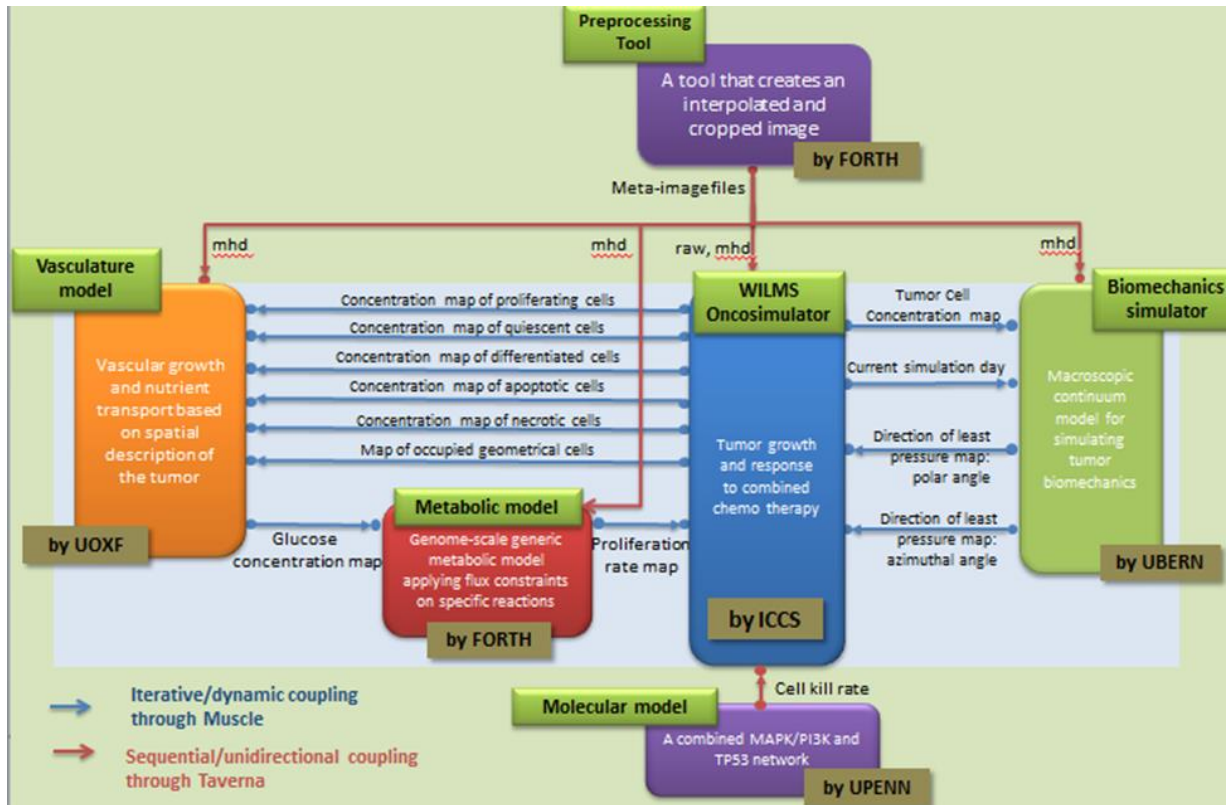
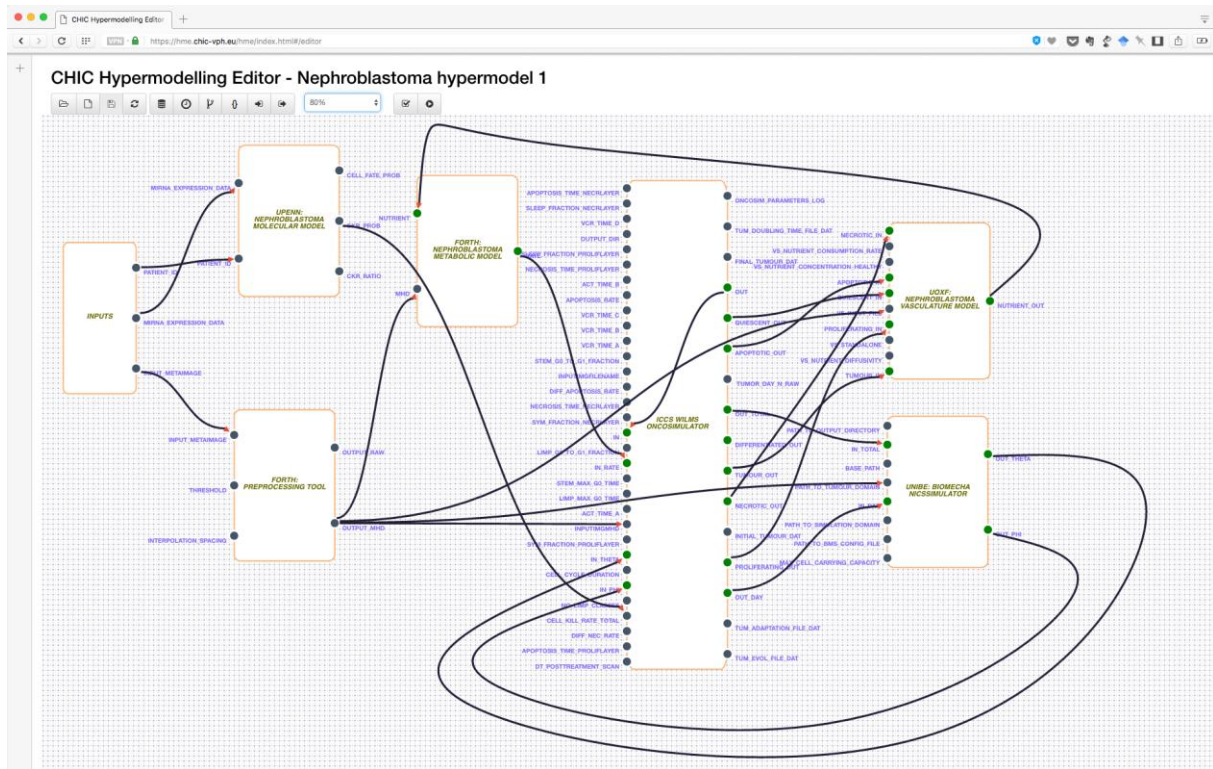


Figure 20: Wilms Multimodeler Hypermodel: integration scheme of Wilms oncosimulator, preprocessing tool, metabolic, molecular, vasculature and biomechanic hypomodels.



**Figure 21: Wilms Oncosimulator multimodeller hypermodel schema in the hypermodelling editor**

## 7 Conclusion

In this deliverable, the architecture of the new VPH-HF framework is described both from a user and a developer point of view, providing a description of the final implementation of all its components, functionalities, APIs and technological choices. The VPH-HF description includes the presentation of its architecture and its components together with the integration of VPH-HF with the other CHIC platform parts.

The document also includes information on the source code repository and on the CHIC final deployment.

As a practical example, the nephroblastoma hypermodel execution in VPH-HF on the CHIC private cloud is also reported.



## 8 References

- [1] J. Borgdorff, E. Lorenz, A. G. Hoekstra, J.-L. Falcone, B. Chopard, “A principled approach to distributed multiscale computing, from formalization to execution”, “Seventh IEEE International Conference on e-Science Workshops”, 2011.
- [2] J. Borgdorff, J.-L. Falcone, E. Lorenz, C. Bona-Casas, B. Chopard, A.G. Hoekstra, “Foundations of distributed multiscale computing: Formalization, specification, and analysis”, “Journal of Parallel Distributed Computing”, vol. 73, pp. 465-483, 2013.
- [3] K. Vahi, I. Harvey, T. Samak, D. Gunter, K. Evans, D. Rogers, I. Taylor, M. Goode, F. Silva, E. Al-Shakarchi, G. Mehta, E. Deelman, A. Jones, “A Case Study into Using Common Real-Time Workflow Monitoring Infrastructure for Scientific Workflows”, “Journal of Grid Computing”, vol. 11, pp. 381-406, 2013 (DOI: 10.1007/s10723-013-9265-4)
- [4] K. Vahi, I. Harvey, T. Samak, D. Gunter, K. Evans, D. Rogers, I. Taylor, M. Goode, F. Silva, E. Al-Shakarchi, G. Mehta, A. Jones and E. Deelman, “A General Approach to Real-time Workflow Monitoring”
- [5] J.-L. Falcone, B. Chopard, A. G. Hoekstra, “MML: towards a Multiscale Modeling Language”, *Procedia Computer Science*, vol. 1, pp. 819-826, 2012..
- [6] D. Groen, S. J. Zasada and P. V. Coveney, “Survey of Multiscale and Multiphysics Applications and Communities”, “Computing in Science and Engineering”, vol. 16 (2), pp. 34-43, 2014.
- [7] 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.
- [8] 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).
- [9] 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.
- [10] M. Abouelhoda, S. A. Issa and M. Ghanem, “Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support”, “BMC Bioinformatics”, vol. 13 (77), 2012 (<http://www.biomedcentral.com/1471-2105/13/77>)
- [11] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi and C. Goble, “The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud”, “Nucleic Acids Research” vol. 41, 2013.
- [12] V. Curcin, M. Ghanem, “Scientific workflow systems - can one size fit all?”, “Proceedings of the 2008 IEEE, CIBEC 2008.
- [13] D. Johnson, J. Cooper and S. McKeever, “Tumor ML: Concept and Requirements of an In Silico Cancer Modelling Markup Language”,
- [14] B. Chopard, J. Borgdorff, A. G. Hoekstra, “A framework for multi-scale modelling”, *Philosophical Transactions of The Royal Society A*, vol. 372: 20130378, <http://dx.doi.org/10.1098/rsta.2013.0378>
- [15] FP7 - ICT - 269978, VPH-Share, WP6: User Access Systems, D6.1: “Workflow Composition/Management: State of the Art”, Version 2.2, 2011.

- 
- [16] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, I. Wang, "Programming Scientific and Distributed Workflow with Triana Services Concurrency and Computation: Practice and Experience", Special Issue: Workflow in Grid Systems, vol. 18 (10), pp. 1021-37, 2006.
  - [17] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, P. Li, "Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows", "Bioinformatics", vol. 20 (17), pp. 3045-54, 2004.
  - [18] E. Deelman, G. Singh, M. H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, D. Katz, "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems", "Scientific Programming Journal", vol. 13 (3), pp. 219-37, 2005.
  - [19] T. S. Deisboeck, Z. Wang, P. Macklin, and V. Cristini, "Multiscale Cancer Modeling," Annu. Rev. Biomed. Eng., vol. 13, no. 1, pp. 127–155, Aug. 2011.
  - [20] T. S. Deisboeck and G. S. Stamatakis, "Multiscale Cancer Modeling". CRC Press, 2010.
  - [21] H. M. Byrne, "Dissecting cancer through mathematics: from the cell to the animal model," Mar. 2010.
  - [22] P. Kohl and M. Viceconti, "The virtual physiological human: computer simulation for integrative biomedicine II," Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 368, no. 1921, pp. 2837–2839, Jun. 2010.
  - [23] "Computer as thinker/doer: problem-solving environments for computational science," IEEE Computational Science and Engineering. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=326669>. [Accessed: 29-Oct-2014].
  - [24] G. Clermont, J. Bartels, R. Kumar, G. Constantine, Y. Vodovotz, and C. Chow, "In silico design of clinical trials: A method coming of age," Critical Care Medicine, vol. 32, no. 10, pp. 2061–2070, Oct. 2004.
  - [25] A. Anderson and V. Quaranta, "Integrative mathematical oncology," 2008.
  - [26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: elements of reusable object-oriented software. 1994.
  - [27] O. Sefraoui, M. Aissaoui, and M. Eleuldi, OpenStack: toward an open-source solution for Cloud Computing Platform Using OpenStack. International Journal of Computer ..., 2012.
  - [28] E. Skounakis, V. Sakkalis, K. Marias, K. Banitsas, and N. Graf, "DoctorEye: A multifunctional open platform for fast annotation and visualization of tumors in medical images," presented at the 2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 3759–3762.
  - [29] E. Elmroth, F. Hernandez, J. Tordsson, "Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment". Future Generation Computer System 2010, vol. 26(2), pp. 245-256
  - [30] M. Shields, "Control-versus data-driven workflows", In Workflows for e-Science, Springer 2007, pp.167-173.
  - [31] B. Ludascher, M. Weske, T. Mcphillips, S. Bowers, "Scientific workflows: business as usual?" In proceedings of the 7th International Conference on Business Process Management, BPM'09, Springer-Verlag, 2009, pp. 31-47.

- [32] T. McPhillips, S. Bowers, B. Ludascher, "Collection-oriented scientific workflows for integrating and analysing biological data", *Data integration in the Life Sciences (DILS)*, 2006, vol. 4075, pp. 248-263.
- [33] G. Kahn, D. Macqueen, "Coroutines and networks of parallel processes", In *Information Processing 77*, North Holland Publishing Company, 1977, pp. 993-998.
- [34] E. Ashford, L. David, "Static scheduling of synchronous data flow programs for digital signal processing", *IEEE Trans Comput*, 1987, vol. 36, pp. 24-35.
- [35] M. Hucka et al., *Evolving a Lingua Franca and Associated Software Infrastructure for Computational Systems Biology: The Systems Biology Markup Language (SBML) Project*, *Systems Biology*, vol. 1, no. 1, pp. 41-53, June 2004.
- [36] C.M.Lloyd et al., *CellML: its future, present and past*, *Progress in Biophysics and Molecular Biology*, vol. 85, pp. 433-450, 2004.
- [37] G. Richard Christie et al., *FieldML: Concepts and Implementation*, *Philosophical Transactions of the Royal Society A*, vol. 367, no. 1895, pp. 1869-1884, May 2009.
- [38] Y. Asai et al., *Specifications of insilicoML 1.0 : A Multilevel Biophysical Model Description Language*. *Journal of Physiological Sciences*, vol. 58, no. 7, pp. 447-458, December 2008.
- [39] Roy T Fielding, Richard N. Taylor, "Principled Design of the Modern Web Architecture", *ACM Transactions on Internet Technology (TOIT)* 2 (2) (2002) 115–150, doi:10.1145/514183.514185, ISSN 1533-5399.
- [40] S.Stamatakis, E.Ch.Georgiadi, N.Graf, E.A.Kolokotroni, and D.D.Dionysiou, "Exploiting Clinical Trial Data Drastically Narrows the Window of Possible Solutions to the Problem of Clinical Adaptation of a Multiscale Cancer Model", *PLOS ONE* 6(3), e17594, 2011.
- [41] E.C.Georgiadi, D.D.Dionysiou, N.Graf, G.Stamatakis, "Towards In Silico Oncology: Adapting a Four Dimensional Nephroblastoma Treatment Model to a Clinical Trial Case Based on Multi-Method Sensitivity Analysis." 2012 Nov; 42(11):1064-78. doi: 10.1016/j.compbimed.2012.08.008.
- [42] OASIS, *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, 20wsd5, <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>
- [43] Network Working Group, *RFC 1951 DEFLATE Compressed Data Format Specification*, version 1.3, 1996.
- [44] Advanced message queuing protocol (AMQP). Web: <http://www.amqp.orgs>
- [45] RabbitMQ. Web: <http://www.rabbitmq.com>
- [46] FP7 - ICT - 600841, CHIC - Computational Horizons in Cancer, WP6: Cancer Models and Hypermodel design, D6.3 Initial Standardized Cancer Hypermodels, 2016
- [47] <http://www.myexperiment.org/>
- [48] <https://en.wikipedia.org/wiki/MyExperiment>
- [49] FP7 - ICT - 600841, CHIC - Computational Horizons in Cancer, WP8: Model and Data Repositories, D8.4 Report on the final system, 2016

## Appendix 1 – Abbreviations and acronyms

<i>AES-CBC</i>	Advanced Encryption Standard-Cipher Block Chaining
<i>AMQP</i>	Advance Message Queuing Protocol
<i>API</i>	Application Programming Interface
<i>DAG</i>	Direct Acyclic Graph
<i>FTP</i>	File Transfer Protocol
<i>GUI</i>	Graphical User Interface
<i>HPC</i>	High Performance Computing
<i>HTTP</i>	HyperText Transfer Protocol
<i>IDP</i>	IDentity Provider
<i>MML</i>	Multiscale Modelling Language
<i>MUSCLE</i>	MULTiscale Coupling Library & Environment
<i>OAEP</i>	Optimal Asymmetric Encryption Padding
<i>PSE</i>	Problem Solving Environment
<i>RDF</i>	Resource Description Framework
<i>REST</i>	Representational State Transfer
<i>RFC</i>	Request For Comment
<i>RPC</i>	Remote Procedural Call
<i>RSA</i>	Initials of the surnames Ron Rivest, Adi Shamir, and Leonard Adleman; it is a public-key cryptography algorithm
<i>SAML</i>	<i>Security Assertion Markup Language</i>
<i>SMS</i>	Storage Management System
<i>SOAP</i>	Simple Object Access Protocol
<i>SPARQL</i>	Protocol and RDF Query Language
<i>SSO</i>	Single Sign On
<i>STS</i>	Secure Token Service
<i>SUMOD</i>	Surrogate MODelling service
<i>UML</i>	Unified Modelling Language
<i>VM</i>	Virtual Machine
<i>VPH-HF</i>	Virtual Physiological Human – Hypermodelling Framework
<i>VPH-OP</i>	Virtual Physiological Human – OsteoPorotic
<i>W3C</i>	World Wide Web Consortium
<i>WEBDAV</i>	WEB Distributed Authoring and Versioning
<i>WMS</i>	Workflow Management System
<i>XML</i>	eXtensible Markup Language

## Appendix 2 – API description

### 2.1 Workflow data model

Workflow data model	
id	INT
owner	CHAR(100)
workflow_title	CHAR(100)
workflow_description	CHAR(400)
workflow_comment	CHAR(400)
model_url	URL
inputset_url	URL
workflow_id	INT
experiment_id	INT
subject_out_id	INT
url_output	URL
status	CHAR
create_time	DATETIME
start_time	DATETIME
finish_time	DATETIME
expiry_time	DATETIME
model_log	URL
exit_code	INT

## 2.2 Director API description

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

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

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

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

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

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

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



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

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

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

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

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

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

## 2.3 File Object Data Model

File object data model	
id	INT
url	URL
filepath	CHAR
is_encrypted	BOOL
md5sum	CHAR
size	INT
created	DATETIME

## 2.4 SMS API description

HTTP Method: GET      URL: /api/sms/storagelist/		
Description	Get a list of the storage solutions (only filesystem is available)	
Returns	200 http status code and a Json object containing the list of the storage solutions if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET      URL: /api/sms/storagelist/filesystem/filelist/		
Description	Get a list of file objects stored in the local filesystem storage	
Returns	200 http status code and a Json object containing the list of the file objects stored in the local filesystem storage if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: POST URL: /api/sms/storagelist/filesystem/filelist/		
Description	Store a file in the local filesystem storage	
Encoding	multipart/form-data	
Parameters passed through request body	filepath	Required – blob
Returns	201 http status code and a Json object containing the new created file object stored in the local filesystem storage if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET URL: /api/sms/storagelist/filesystem/filelist/{id}/		
Description	Get the file object {id} stored in the local filesystem storage	
Returns	200 http status code if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the file object {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

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

HTTP Method: GET URL: /api/sms/storagelist/filesystem/filelist/{id}/content		
Description	Get the file object {id} content (aka the blob) stored in the local filesystem storage	
Returns	200 http status code and the blob if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the file object {id} is not found	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

HTTP Method: GET      URL: /api/sms/repolist/{repo_name}/{resource_list_name}/{id}/content		
Description	Get the resource {id} from the resource list {resource_list_name} stored in the repository {repo_name}	
Returns	200 http status code if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the name of the repository is not in the list of the available repositories	
	406 http status code if the {resource_list_name} is not acceptable (filelist and objectlist are acceptable)	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>



<b>HTTP Method: GET      URL:</b> <b>/api/sms/repolist/{repo_name}/{resource_list_name}/{id}/copyto/wmslist/{wms_name}/runlist/{run_uuid}/wd/in/{filename}</b>		
Description	Copy the resource {id} from the resource list {resource_list_name} stored in the repository {repo_name} to the wms {wms_name} in the working directory of the run {run_uuid} named {filename}	
Returns	200 http status code if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the name of the wms or the name of the repository is not in the list of the available wms and repositories, respectively	
	406 http status code if the {resource_list_name} is not acceptable (filelist and objectlist are acceptable)	
	500 http status code if an internal server error occurs	
HTTP Header	Name: Authorization	Value: SAML auth=<Base 64 Encoded compressed SAML token>

<b>HTTP Method: GET      URL:</b> <b>/api/sms/wmslist/{wms_name}/runlist/{run_uuid}/wd/out/copyto/repolist/{repo_name}/subject/{subject_out_id}</b>		
Description	Copy the results of the run {run_uuid} from the wms {wms_name} to the repository {repo_name} in the subject {subject_out_id}	
Returns	200 http status code if the request has finished successfully	
	400 http status code if a bad request error occurs	
	403 http status code if authentication token is not provided or not verified	
	404 http status code if the name of the wms or the name of the repository is not in the list of the available wms and repositories, respectively	

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