Deliverable No. 7.1

Hypermodelling specifications

Grant Agreement No.:	600841
Deliverable No.:	D7.1
Deliverable Name:	Hypermodelling specifications
Contractual Submission Date:	30/03/2014
Actual Submission Date:	30/06/2014

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

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	**CHIC**
Project Full Name:	Computational Horizons In Cancer (CHIC): Developing Meta- and Hyper-Multiscale Models and Repositories for *In Silico* Oncology
Deliverable No.:	D7.1
Document name:	Hypermodelling specifications
Nature (R, P, D, O)[1]	R
Dissemination Level (PU, PP, RE, CO)[2]	PU
Version:	9.0 (Final)
Actual Submission Date:	30/06/2014
Editor: Institution: E-Mail:	Georgios Stamatakos ICCS-NTUA gestam@central.ntua.gr

ABSTRACT:

The aim of this deliverable is to present the initial technological hypermodelling specifications. By hypermodelling, we define the process of developing hypermodels. The fundamental concepts of the domain that CHIC project belongs to, and the terms that have been adopted or proposed in order to describe effectively the CHIC "microenvironment" are explained.

The various components of the CHIC Hypermodelling Framework are presented from a functional point of view. A significant part of the deliverable has been dedicated to the description of a specific component, the so called Generic Stub. The Generic Stub is particularly important since it acts as a "bridge" between two CHIC workpackages, *"WP6: Cancer Models and Hypermodel Design"* and *"WP7: Hypermodelling infrastructure".*

KEYWORD LIST:

hypermodelling specifications, hypermodelling framework, model, hypermodel, hypomodel, generic stub, model wrapper, workflow manager, workflow orchestrator, log manager, fault manager, registry service, annotation services, *in silico* medicine

[1] **R**=Report, **P**=Prototype, **D**=Demonstrator, **O**=Other

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

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

MODIFICATION CONTROL			
Version	**Date**	**Status**	**Author**
1.0	17/01/2014	Index Draft	Daniele Tartarini, USFD Marco Viceconti, USFD Dawn Walker, USFD
2.0	07/03/2014	Draft	Marco Viceconti, USFD Dawn Walker, USFD Debora Testi, CINECA
3.0	08/03/2014	Draft	Fay Misichroni, ICCS-NTUA
4.0	09/03/2014	Draft	Debora Testi, CINECA
5.0	15/03/2014	Draft	Fay Misichroni, ICCS-NTUA
6.0	22/03/2014	Draft	Stelios Sfakianakis, FORTH
7.0	20/06/2014	Draft	Fay Misichroni, ICCS-NTUA
8.0	27/06/2014	Revision	Dimitra Dionysiou, ICCS-NTUA
9.0	30/06/2014	Final	Georgios Stamatakos, ICCS-NTUA

Additional Contributors

- Feng Dong, BED

Contents

Table of figures

1 Executive Summary

Developing robust, reproducible, interoperable and collaborative hypermodels of diseases and normal physiology is a sine qua non necessity if rational, coherent and comprehensive exploitation of the invaluable information hidden within human multiscale biological data is envisaged. Responding to this imperative in the context of both the broad Virtual Physiological Human (VPH) initiative and the paradigmatic cancer domain, CHIC proposes the development of a suite of tools, services and secure infrastructure that will support accessibility and reusability of VPH mathematical and computational hypermodels. The CHIC tools, services, infrastructure and repositories will provide the community with a collaborative interface for exchanging knowledge and sharing work in an effective and standardized way. A number of open source features and tools will enhance usability and accessibility.

In order to ensure clinical relevance and foster clinical acceptance of hypermodelling in the future, the whole endeavour will be driven by the clinical partners of the consortium. Cancer hypermodels to be collaboratively developed by the consortium cancer modellers will provide the framework and the test bed for the development of the CHIC technologies. Clinical adaptation and partial clinical validation of hypermodels and hypermodel oncosimulators will be undertaken.

Work package 7 (WP7) aims at:

- Wrapping and deploying each component model according to a Component Model Generic Stub, which standardises the model's control and data flow and makes possible the orchestration of several models into hypermodels.
- Developing basic annotation and tags management services, to be used for the provision of i) folksonomy annotation and search services, and ii) ontology-base search services.
- Developing an ICT hypermodelling infrastructure that makes possible the construction and execution of hypermodels, formed by component models and relational models.
- Developing hypermodels annotation services and exploring the use of innovative technologies such as the use of linked data or semantic reasoning.
- Deploying all hypermodelling technologies on a production private cloud in order to be used by the CHIC consortium to analyse patients' data.

2 Introduction

2.1 Purpose of this document

This deliverable aims at presenting the initial hypermodelling specifications. By hypermodelling, we define the process of developing hypermodels.

The deliverable begins with an introduction to the fundamental concepts of the domain that CHIC project belongs to, but also with an introduction to the terms that have been adopted or invented in order to describe effectively the CHIC "microenvironment".

The document proceeds with its main objective: to present the various components participating in the CHIC Hypermodelling Framework from a functional point of view. The deliverable dedicates a significant part to describing a specific component, the so called Generic Stub. The Generic Stub is so important because it acts as a "bridge" between *"WP6: Cancer Models and Hypermodel Design",* and *"WP7: Hypermodelling infrastructure".*

2.2 Structure of the Deliverable

Chapter 3 presents a set of definitions that intent to introduce the reader into the domain that the CHIC project belongs to and provides descriptions of fundamental concepts and terms that will be used in the CHIC project. The collaborating efforts of all members of the CHIC consortium have resulted into the collection of a wide set of concepts and terms. Chapter 3 presents a subset of those concepts and terms, the ones that are relative to the content of the current deliverable.

Chapter 4 describes the Generic Stub and the Model Wrapper. These components are fundamental for the overall CHIC modelling architecture, as they provide a generic abstraction of a model. This approach can provide to the partners of the CHIC project a common base for their respective activities.

Chapter 5 presents the various components participating in the CHIC Hypermodelling Framework from a functional point of view. Chapter 6 describes where the CHIC Hypermodelling Framework is located in the overall CHIC architecture and what interconnections exist.

Finally, Chapter 7 presents the initial release plan of the CHIC Hypermodelling Framework and Chapter 8 describes the infrastructure that will host the CHIC Hypermodelling Framework.

3 Definitions

3.1 Introduction

The objective of this chapter is to present a set of definitions that will introduce the reader into the domain that the CHIC project belongs to and to provide the descriptions of the fundamental concepts and terms that will be used in the CHIC project. The collaborating efforts of all members of the CHIC consortium have resulted into the collection of a wide set of concepts and terms. This chapter presents a subset of those concepts and terms: the ones that are relative to the content of the current deliverable.

3.2 Generic definitions

3.2.1 In silico

In silico is an expression used to mean "performed on a computer or via computer simulation". The phrase was coined in 1989 as an analogy to the Latin phrases *in vivo*, *in vitro*, and *in situ*, which are commonly used in biology and refer to experiments done in living organisms, outside of living organisms, and where they are found in nature, respectively[3].

3.2.2 *In silico* medicine

In silico medicine (also known as "computational medicine") is the application of *in silico* research to problems involving health and medicine. It is the direct use of computer simulation in the diagnosis, treatment, or prevention of a disease. More specifically, *in silico* medicine is characterized by modelling, simulation, and visualization of biological and medical processes by use of computers with the goal of simulating real biological processes in a virtual environment[4].

3.2.3 Virtual Physiological Human (VPH)

The Virtual Physiological Human (VPH) is a methodological and technological framework that enables collaborative investigation of the human body as a single complex system. The collective framework makes it possible to share resources and observations formed by institutions and organizations creating disparate but integrated computer models of the mechanical, physical and biochemical functions of a living human body[5].

[3] http://en.wikipedia.org/wiki/In_silico
[4] http://en.wikipedia.org/wiki/In_silico_medicine
[5] STEP Consortium. Seeding the EuroPhysiome: A Roadmap to the Virtual Physiological Human. 5 July 2007

3.3 Scientific and technological definitions

3.3.1 Data

We define data as factual information, whether observed or predicted.

- Observed: generated through observation, measurement etc.
- Predicted: generated through speculative reasoning informed by existing knowledge.

3.3.2 Scientific model

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[6].

3.3.3 Computer model

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).

3.3.4 Metadata

Metadata is “data about data”. The term is ambiguous, as it is used for two fundamentally different concepts (types) [7].

- Structural metadata refers to the design and specification of data structures and is more properly called "data about the containers of data".
- Descriptive metadata, on the other hand, refers to individual instances of application data, i.e. the data content.

3.3.5 Ontology (computer science)

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.

[6] 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.
[7] http://en.wikipedia.org/wiki/Metadata

3.3.6 Folksonomy (computer science)

In information science, a folksonomy is a system of classification derived from the practice and method of collaboratively creating and translating tags to annotate and categorize content. This practice is also known as collaborative tagging, social classification, social indexing, and social tagging.

3.3.7 Orchestration or choreography (technology)

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[8]. However, in cases like ours where the execution of the services is coordinated centrally (autocratic) the term orchestration is preferred.

[8] http://www.infoq.com/news/2008/09/Orchestration

3.4 CHIC project definitions

3.4.1 Model

In 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. The above definition is in alignment with the terms “scientific model” and “computational model” described in sections 3.3.2 and 3.3.3, respectively.

3.4.2 Hypomodel (or component model)

We define as a hypomodel (or component model) 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.

3.4.3 Hypermodel (or composite model or integrative model)

We define as a hypermodel (or composite model or integrative model) 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.

3.4.4 Elementary model

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.

3.4.5 Meta-model, meta-hypomodel, meta-hypermodel

We define as a meta-model the semantic description of a model. The meta-model can be considered as an abstract representation of a model, as it highlights certain properties of the model itself. Consequently, a meta-hypomodel and a meta-hypermodel is the semantic description of a hypomodel and a hypermodel respectively.

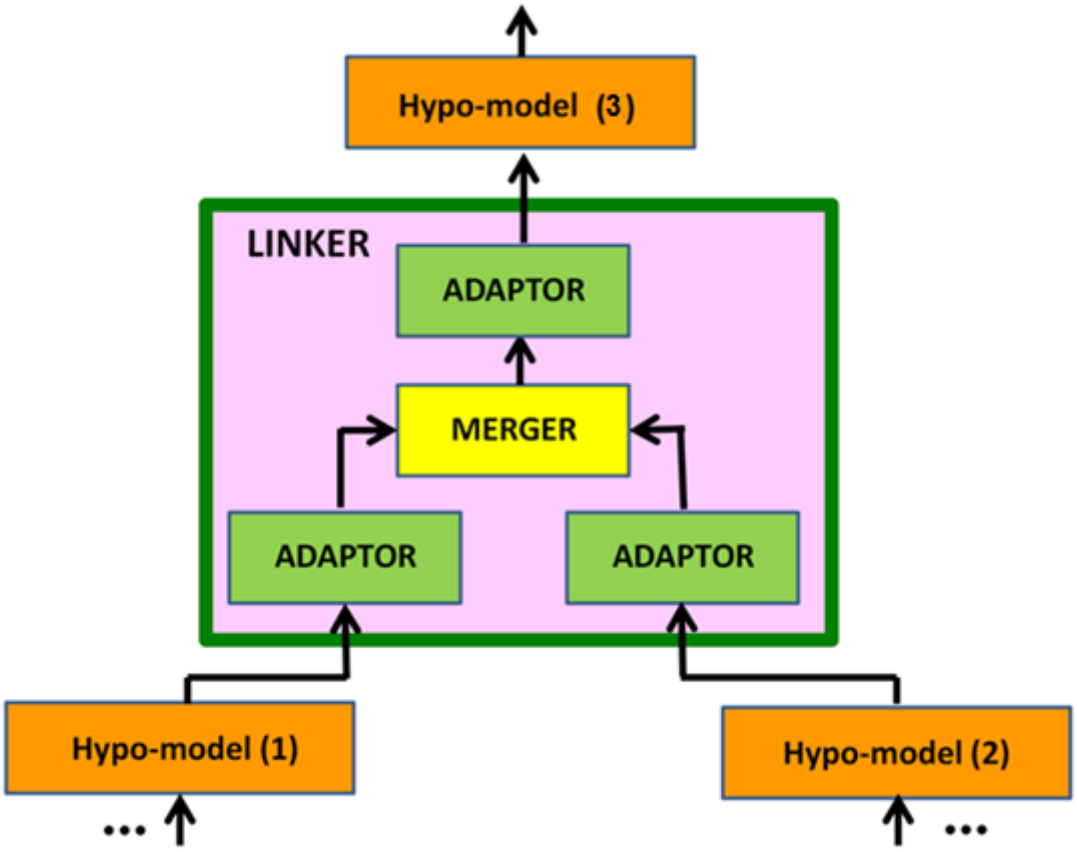

Figure 1 : Graphical representation of the linker, the adaptor and the merger. The whole figure can be viewed as representing a single hypermodel.

3.4.6 Adaptor or relational model

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.

3.4.7 Merger

We define as a merger a piece of software that merges the adequately adapted outputs of two hypomodels.

3.4.8 Linker

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.

3.4.9 Hypermodelling

We define as hypermodelling the process of developing hypermodels.

3.4.10 Hypermodelling infrastructure

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.

3.4.11 Hypermodelling framework

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.

4 Generic Stub and Model Wrapper

4.1 Introduction

The objective of this chapter is to describe the Generic Stub and the Model Wrapper. These components are fundamental for the overall CHIC modelling architecture, as they provide a generic abstraction of a model. This approach can provide to the partners of the CHIC project a common base for their respective activities.

4.2 Composition aspects

In the CHIC project the models are described:

- In the “concrete world”, as computational entities that can be invoked and executed.
- In the “abstract world”, as metamodels that incorporate information about their domain, functionality, semantics, etc.

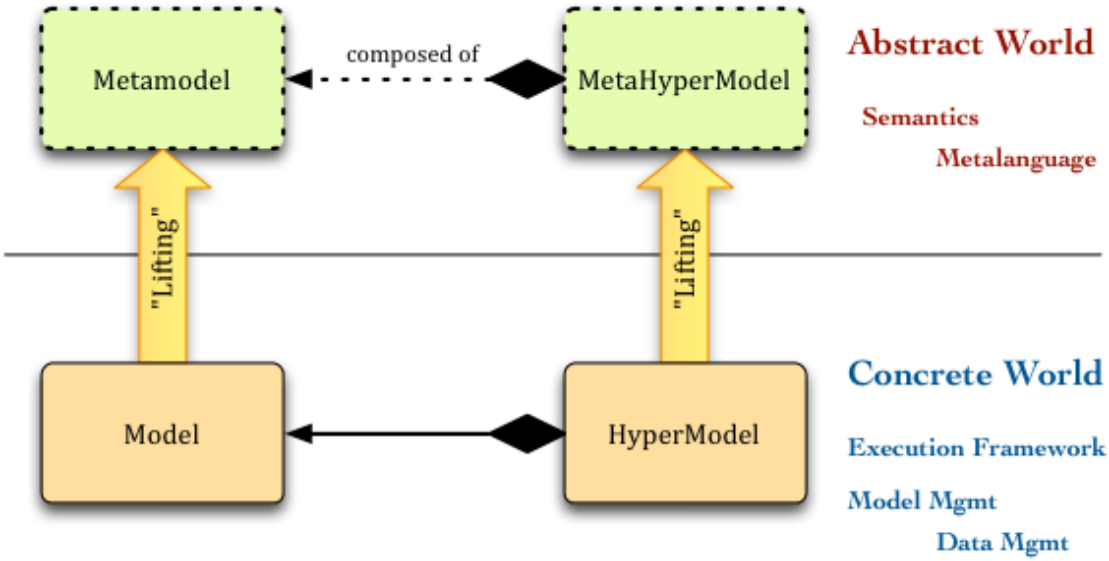

Figure 2 : The different model description and composition aspects

In the “concrete world” the models can be considered as Hypomodels which, when combined together, can be used for building more complex Hypermodels. The composition strategy is based on the notion of Orchestration where the Hypomodels are viewed as software components that interact by exchanging data under the supervision and control of a central Orchestrator.

4.3 The Generic Stub

In order to have the aforementioned interactions in the context of a hypermodel, the platform should be able to accommodate a lot of different types of models both from the computational and the abstract layer.

The "Component Model Generic Stub", or Generic Stub for short, is a template that all models participating in CHIC should comply with in order to be effectively integrated into the rest of the platform.

The Generic Stub is a fundamental component of the CHIC Hypermodelling Framework. It aims at exposing each model in a standardized way, by providing a unified interface for the model execution. This approach allows considering each model as a black-box to be executed without needing details on how it is implemented or on how it is working internally.

More specifically, the role of the Generic Stub is:

- to expose each model to the rest of the CHIC Hypermodelling Framework with the same API
- to handle control flow and data flow for the model.

The Generic Stub must comply with the following specifications:

- Be easy to be used by non-developers (i.e. researchers), possibly working with a GUI environment.
- Allow the mapping from the general syntax of the hypermodel to the local one of the hypomodel.
- Be open to extension, meaning that new models can be wrapped.
- Handle both data and control flow for individual models.

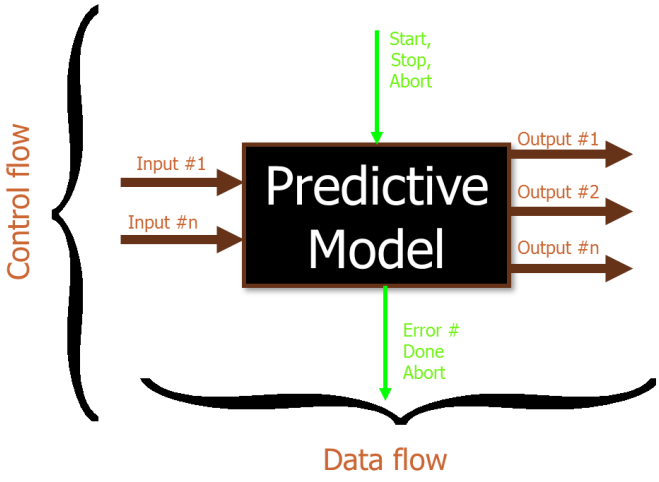

Figure 3 : Generic representation of a model including control flow and data flow.

4.4 Type of models

An initial analysis of the models presented in deliverable *"D6.1: Cancer hypomodelling and hypermodelling strategies and initial component models"* indicates that the majority of the models under consideration are provided mostly as scripts for an interpreted language such as Python, Perl, or Matlab, or as binary C/C++ compiled executables.

Nevertheless, in order to be as generic as possible, we define the following types of models depending on their execution properties:

- "Configurable", which means that the model has been provided with all the needed information to be executed in a generic way. The models encoded in a standard model description language such as SBML belong to this category, assuming that all the inputs and outputs parameter information and simulation requirements are provided. The models that are published as script or executable code in a specific execution environment (e.g. Operating System, programming environment like Matlab, Octave, etc.) are also examples of this type, and the same requirements apply.

- "Static", which means that the models have been published as network accessible resources, for example as Web Services and there are network endpoints for accessing such models. In this case the models themselves are not available, but there are programmatic interfaces to invoke them by supplying the required inputs.

- "Migrating", which applies to models that are provided as virtual machine (VM) images ready to be deployed and run in a cloud, "virtualized" environment. This permits for more flexible ("elastic") deployment options where models are deployed and run "on demand" as load increases.

4.5 The Model Wrapper

The Model Wrapper is an implemented instance of the Generic Stub.

It contains information on how to execute the specific model that has been wrapped and supports operations such as:

- Connection to the storage services to retrieve data and push result files.

- Communication with the registry and the log management to provide information on its execution.

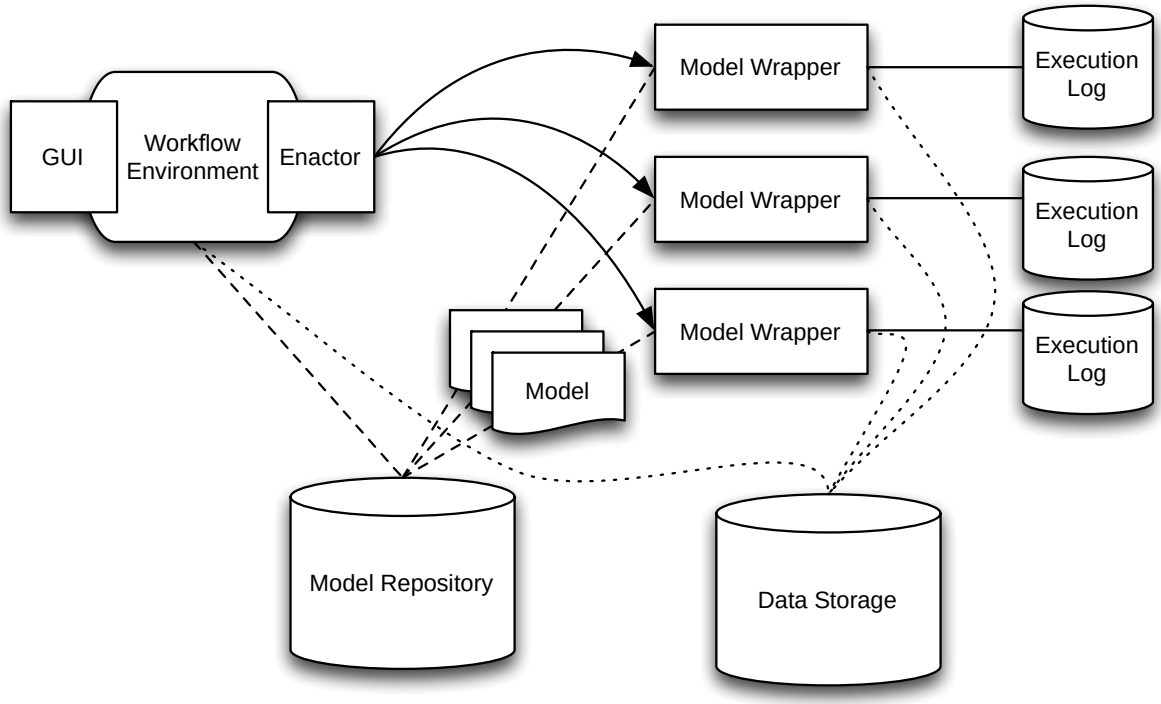

Figure 4 : Graphical representation of interaction of the Model Wrapper with other components.

More specifically, the Model Wrapper

- Decrypts/encrypts data passed between the storage service and the model.
- Launches the script required in order to execute the model.
- Provides functionality to access the control flow (i.e. kill or stop the wrapper execution).
- Manages the creation of a log database which is sent to the log manager after execution.

In case of "configurable" models the implementation of the Model Wrapper can be a generic one, i.e. one implementation to support the execution and monitoring of a "family" of models. In the other cases the model implementers should provide the Model Wrapper because there is tight coupling between the models themselves and the Model Wrapper.

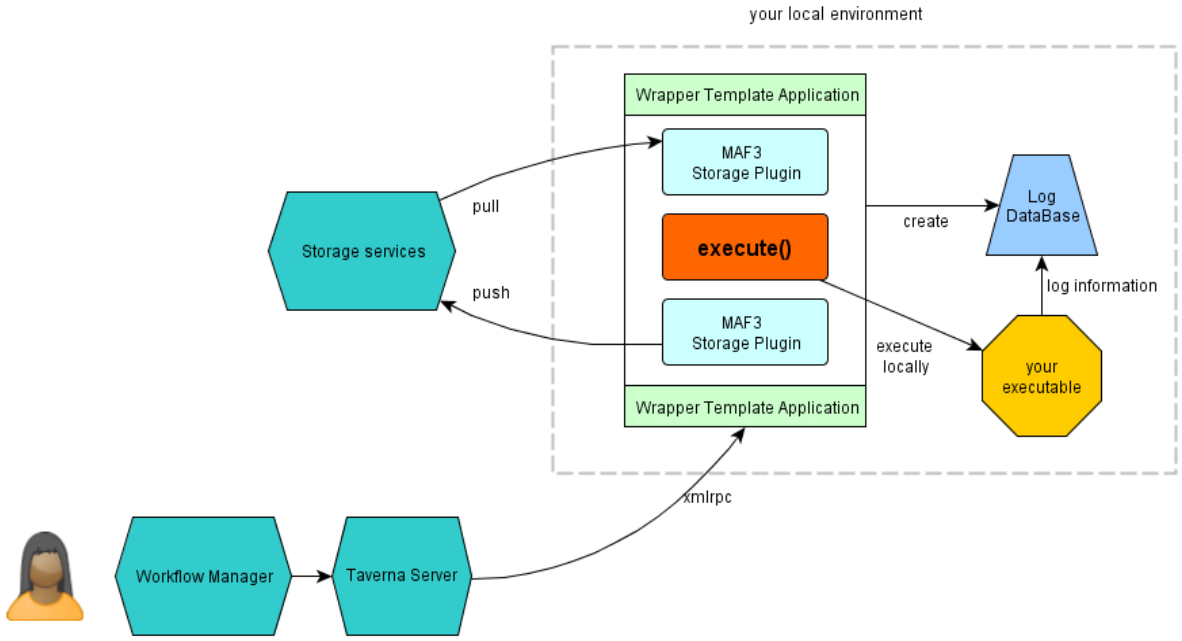

Figure 5 : The role of the Model Wrapper in handling data and control flow between an individual hypomodel and the execution framework.

4.6 Alternative approaches on model wrapping

There are two possible approaches concerning model wrapping:

- The first, used in the VPH-OP project[9], sees the execution framework as a tightly controlled, highly optimized computational environment. New models are deployed on the target hosts, and wrapped by the modellers in collaboration with the framework developers. Once a model is wrapped, and exposed to the MAF3 Bus module, any workflow can invoke it at run time, as far as its status is "running".

- The alternative is to expose each model as an atomic Remote Procedure Call (RPC), to manage the orchestration through direct calls, and to manage the data flow separately (i.e. with replication mechanisms). In this case the hypomodel is exposed using a Remote Procedure Call (RPC) interface. Examples for such interfaces in Thrift IDL and Web Services Definition Language (WSDL).

While the first approach ensures excellent optimization and makes it possible to cope also with computationally and data intensive models, the second might be affected by considerable latencies, especially with large data objects. However, the second approach is much more flexible and requires much less central effort, allowing a more dynamic growth of the model repository.

[9] http://www.vphop.eu/

Both approaches are considered and are being tested in order to find the one that best suits the needs of the CHIC project.

Taking into consideration that the vast majority of the models to be wrapped are described using some interpreted language (Ansys APDL, Matlab, Octave, CellML, SMBL, Python, etc.) we propose to create:

- A wrapper that can be pre-configured by a developer for each modelling environment (Matlab, Octave, Ansys,...) or generic mark-up format (SBML, CellML etc.) that support an interpreted script.

- A simple configuration file that can be supplied by an individual modeller to supply information related to model input and output data, including parameter and data set location and types.

A number of options related to the format of this configuration file are currently under exploration. These include:

- A mark up language format e.g. XML-based file, where the user is required to populate a series of fields denoted by designated tags.

- A script file format (e.g. Python-based).

- A web-interface, through which an individual user can annotate model input/output specifications on a dedicated web form.

Once the necessary hypomodels are wrapped, the CHIC Hypermodelling Framework can orchestrate any execution that combines them in any fashion, managing both control and data flow.

4.7 The Generic Stub from the modeller's point of view

The author/creator of a new hypomodel to be used in CHIC should provide all the information that comprises the Metamodel (i.e. scale, inputs and outputs annotation, descriptive metadata such as model's title, etc.). At the computational level the modeller needs to provide the model itself in one (or many) computable form, and this can be either "configurable", "static", or "migrating", as described in section 4.4.

- In the "static" case the model provider is responsible for hosting the model implementation in his premises, or in leased infrastructure, and also for providing machine readable access to this model. The interface of this network accessed model should be the same as the wrapper interface presented above.

- In the "migrating" case the model provider should supply a ready-made virtual machine instance for the CHIC-compliant cloud infrastructure. After its run time deployment of the instance performed by the Workflow Environment in the context of a hypermodel execution,

the instance should provide a similar wrapper interface in the network address of its deployment.

- The “configurable” case puts a lot more effort in the CHIC execution framework since it makes use of the CHIC generic wrapper interface. This wrapper should follow the interface presented above and there can be multiple realizations of it, depending on the nature of the “configurability”: for example, one wrapper can be implemented for executing SBML models, another one for executing command line tools etc. The execution information is specific to the model implementation and the wrapper should retrieve it from the model repository.

The last case is by far the most important based on the list of models in deliverable “D6.1: Cancer hypomodelling and hypermodelling strategies and initial component models” and also on earlier experiences. Therefore, it is the case where most effort and optimization need to be put on. The requirements and effort imposed to the model creators should be kept at minimum in order for CHIC to gain visibility and attract users.

4.8 Functional requirements

The Model Wrapper must support the following operations (in pseudo code):

- **run_model** `(parameters: key-value-pairs-list, exec-context-id: string, model-id: string, security-info: …, other: …)`

  It returns a "run-id" that uniquely identifies the execution or an error (e.g. if a model was not found in the repository). The supplied inputs include a list of parameter names and their values, the “execution context id” that uniquely identifies the execution of the workflow that triggers the run of this model, the model identifier, security information (e.g. the user id), etc.

- **get_status** `(run-id: string)`

  It returns the status of the specific model execution, e.g. whether it’s RUNNING, STOPPED, etc.

- **get_results** (run-id: string)

  It returns a "key-value-pairs-list" with the model output or error if execution hasn't finished. In case of a file as output the key-value pair will indicate the URIs of the newly created resources.

- **get_log** `(run-id: string)`

  It returns the logged execution information (e.g. error messages).

- Some "control" operations: **stop**, **start**, **abort**, etc. all accepting the "run-id" parameter.

5 CHIC Hypermodelling Framework (CHIC-HF)

5.1 Introduction

The CHIC Hypermodelling Framework (CHIC-HF) consists of a set of components that facilitate the execution of hypermodels. The objective of this chapter is to present the various components participating in the CHIC Hypermodelling Framework from a functional point of view.

5.2 Basic characteristics of the CHIC Hypermodelling Framework

The architecture of the CHIC Hypermodelling Framework is designed in such a way so as to comply with the following characteristics:

- to be possible to replace or modify components without disrupting other services, ensuring easy maintenance, extension, and update of the system
- to be possible to easily add in the future new components or new features to the architecture, i.e. new hypomodels or specialised modules
- to provide interfaces by which end-users applications can access information concerning the running workflow/modules in order to be presented to the end-user and modellers via proper user interfaces

In order to achieve this, the CHIC Hypermodelling Framework has been decomposed into different components. The partner in charge of each component has selected the most suitable technology for either implementing new modules/services or extending/modifying ones that have been developed in the context of previous projects, such as TUMOR[10] and VPH-OP[11].

The core components of the CHIC Hypermodelling Framework are:

- The Generic Stub (already presented in chapter 4)
- The Workflow Orchestrator
- The Workflow Manager

The following components are also part of the CHIC Hypermodelling Framework, as they are responsible for necessary supplementary tasks. These components are:

- The Log Manager

[10] http://tumor-project.eu/
[11] http://www.vphop.eu/

- The Fault Manager
- The Authentication Manager

Additionally, the CHIC Hypermodelling Framework includes several services that aim at assisting the interaction with other components of the CHIC framework. These services are:

- The Storage Service
- The Annotation Service

Finally, we also describe two tools that are directly connected with the CHIC Hypermodelling Framework:

- The Hypermodelling Editor
- The Hypermonitor

During the development of the various components of the CHIC Hypermodelling Framework, changes in the architecture or/and the functional components might occur. These modifications will be reported in the future CHIC Hypermodelling Framework documentation.

The following sections contain the description and the minimum requirements/specifications for the interfaces of each component. The aforementioned specifications will help minimize the interdependencies between the components and will facilitate a more effective implementation. During the development phase, the specifications of the interfaces will be accordingly updated when needed, from a functional or an integrative point of view.

5.3 Workflow Orchestrator

Description

The Workflow Orchestrator is the component that is responsible for the execution of already saved hypermodels. It acts as an orchestrating engine that triggers the execution of the participating hypomodels and supplementary software components, such as linkers, and passes the data around, from the source sub-model (producer) to the target sub-model (consumer). Thus, this service contains all the information on the sequence diagrams. Consequently, it launches the sub-models in the right order and, when a sub-model stops or ends its execution, it retrieves the relative information.

Taverna

Taverna[12] is an open source and domain-independent Workflow Management System[13]. Taverna consists of:

- The Taverna Server that acts as the 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.

Taverna Server represents the orchestrator of the CHIC Hypermodelling Framework.

Functional requirements

The Workflow Orchestrator must support the following operation (in pseudo code):

- **execute_workflow** `(exec-context-id: string, storage-id: string, security-info: …, input-file-list: string, output-file-list: string, parameters: key-value-pairs-list)`

  It triggers the execution of the participating sub-model and passes the data around, from the source sub-model (producer) to the target sub-model (consumer). It takes as input information: which is the uniquely identified associated workflow to be executed (exec-context-id), where is the data repository (storage-id), the inputs and outputs files and any other additional parameter necessary to the execution of the sub-models.

[12] http:/www.taverna.org.uk/

[13] Workflow Management System: a suite of tools used to design and execute scientific workflows.

5.4 Workflow Manager

Description

The Workflow Manager is the dispatcher component which executes a given hypermodel on the data of a given patient. While the Workflow Orchestrator communicates with single hypomodels for execution, the Workflow Manager role is to check the user credentials, to create a workflow execution session, to connect to the registry in order to warn about problems before starting the actual execution, and to log about the progress of execution or about the appearance of errors. It is also the interface between the end-user applications and the back-end technology.

In summary, the Workflow Manager is the workflow dispatcher that:

- handles association between users and workflows,
- handles communication between the end user applications and other hypermodel core modules, and
- permanently serializes workflows related information and creates an execution context for the workflow to run (e.g. stages needed data, creates temporary folders, etc).

Functional requirements

The Workflow Manager must support the following operations (in pseudo code):

- **submitWorkflow** `(storage-id: string, security-info: …, input-definition-file: string, wf-definition-file: string, parameters: key-value-pairs-list)`

  It prepares the workflow for execution by assigning it the unique id (provided in output as "execution context id"), gets the user permissions (security-info), and collects the file describing the workflow (wf-definition-file) and its data input and output description (input-definition-file).

- **startWorkflow** `(exec-context-id: string)`

  It starts the workflow execution and returns an error or a success code.

- **stopWorkflow** `(exec-context-id: string)`

  It stops the selected workflow execution and returns an error or a success code.

- **get_WorkflowInformation** `(exec-context-id: string)`

  It returns information concerning the workflow under consideration such as the status (i.e. initialised, running, completed), the owner, the creation time, the start and finish time, etc.

- **get_WorkflowsList** ()

  It returns information on all workflows.

5.5 Registry Service

Description

Each one of the modules or services available for composing a workflow must be registered. The information related to the registered modules/services is contained in a database, called the Registry. The Registry contains information about the description and the status of the modules and services, such as availability (down, free, busy), location, description, etc.

The Registry Service is the component that is responsible for accessing and managing the Registry.

Functional requirements

The Registry Service must support the following operations (in pseudo code):

- **register_service** (name: string, service-uri: string, description: string, status: string, endpoint: string)

  It allows adding a new service or module into the Registry and returns either an error code or the service or module unique id. The "endpoint" type will be different according to the type of generic stub to be added (i.e. wsdl for a web service, storage location for a script, etc.). The "status" represents the information on the running of the service (i.e. running, down, stopped, etc.) to be used later on by the Workflow Manager and by other services.

- **unregister_service** (service-id: string)

  It removes the specific service or module from the Registry.

- **get_ServiceInformation** (service-id: string)

  It returns the information stored for a specific service or module into the Registry.

- **get_ServicesList** ()

  It returns the list of all available services or modules stored into the Registry along with their information.

5.6 Log Manager

Description

The Log Manager is the component that is responsible for collecting the logs produced during execution of each sub-model. The logs are provided by each sub-model during its execution in the workflow. The logs produced can be accessed by the various end-user applications, such as the Workflow Manager and the Fault Manager.

The Log Manager requests a detailed log from other modules by polling each of them periodically. It saves the result of the call in a database record. It also provides APIs for accessing the database. The information gathered can be queried later by the Workflow Manager to make it also available to the end-users applications.

Functional requirements

The Log Manager must support the following operations (in pseudo code):

- **get_log** `(date-range: string, exec-context-id: string, run-id: string, security-info: …)`

  It gets the log in a certain time range for the specific module (run-id) in the specific workflow execution (exec-context-id) to be added to the database.

- **update_log** `(exec-context-id: string, run-id: string, security-info: …)`

  It updates the information on the specific execution into the database of the logs.

5.7 Fault Manager

Description

The Fault Manager is the component that is responsible for the management of exceptions and errors in the execution of sub-models. Additionally, it is responsible for dealing with errors and failures of the architectural and workflow components.

If an exception occurs during the execution of a sub-model, the Fault Manager immediately provides to the user the relevant information and possibly the intermediate results. The detection of an exception in the execution of one model allows other models to free their resources or/and switch to other tasks. This can be managed in conjunction with the Workflow Manager.

The Fault Manager is actually a service. This service monitors the logs to detect fault situations, which may also be reported by the user, and, according to the type of problem, launches the appropriate procedure. It is possible to create different behaviours, called policies, such as the ABORT, RESUME, STOP, etc.

Functional requirements

The Fault Manager must support the following operations (in pseudo code):

- **set_Policy** `(name: string, description: string, type: string)`

  It allows setting the policy to be executed in association to different types of fault events. It returns an error code.

- **get_Policy** ( )

  It returns the list of available policies for fault management in terms of name, description and type.

5.8 Authentication Service

Description

The Authentication Service is the component that is responsible for authenticating the user into the CHIC Hypermodelling Framework. This service will allow the easy integration of the security layer being developed in the context of *"WP5: IT Architecture",* by providing to the CHIC Hypermodelling Framework a unique API to get the user information. The Authentication Service allows obtaining user credentials from different authorisation mechanisms, such as OpenID, SAML etc.

The Authentication Service provides the mechanisms to authenticate the user into the system, and it is equipped with features for accounting and granting permissions only to certain parts of the CHIC Hypermodelling Framework. The service obtains valid credentials and creates validation ticket/security information, which is passed to all other requiring services.

Functional requirements

The Authentication Service must support the following operations (in pseudo code):

- **validate_user**`(user-id: string)`:

  It validates the user identity by connecting to the authentication mechanism available in CHIC and returns the appropriate status code.

- **get_permissions**`(user-id: string)`:

  It returns for the specific user the associated "security info" like the permission level.

5.9 Storage service

The Storage service is the component that is responsible for connecting to the repositories where models and input/output data are stored. It might include also the services for data translation and transformation.

The repositories themselves are not part of the CHIC Hypermodelling Framework, but they are considered fundamental components of the CHIC platform (being developed in the context of "*WP8: Model and Data Repositories"*). Below we present in brief these repositories and their corresponding interfaces. More details can be found in deliverables "D8.2: Design of the CHIC repositories" and "D10.2 Design of the orchestration platform, related components and interfaces".

Model repository

The model repository is the physical entity where the modules used in construction and execution of hypermodels are stored. The aforementioned modules can be hypomodels, hypermodels, linkers, data transformation tools and other tools.

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

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).
- The results produced when the *in silico* trial is completed (the output of the hypermodel execution).

The *in silico* trial repository will incorporate a REST read-write API for searching, retrieving and storing *in silico* trial instances. This API will be secured (authentication and authorization) according to the adopted security framework in CHIC.

Functional requirements

The Storage Service must support the following operations (in pseudo code):

- **data_pull** `(data-id: string, security-info: …)`

  It retrieves from the data repository the data specified in the uri (data-id) and returns an error code.

- **data_push** `(exec-context-id: string, security-info: …, file-name: string)`

  It uploads/stores the file into the repositories and returns its MD5 for checking the correctness of the upload and the file URI.

5.10 Annotation service

Description

The Annotation service is the component that is responsible for annotating the resources, which are created or modified by the CHIC Hypermodelling Framework. For example, if new data are created at the end of the hypermodel execution, these new data will need to be properly annotated.

This URL format to be used will have a base URL, followed by a resourceID that is unique to the whole project, followed by a set of options through which various aspects of the resource can be accessed. The REST approach will be most likely adopted, so the URL schema will be something like:

- http://baseurl/resourceid/

  It returns all metadata

- http://baseurl/resourceid/function

  It returns details on inputs, outputs, scale, functionality.

- http://baseurl/resourceid/description

  It returns a human readable description of the resource.

- http://baseurl/resourceid/execution

  It returns list of inputs and their metadata.

Functional requirements

The Annotation Service must support the following operations for setting and retrieving metadata (in pseudo code):

- **set_metadata** `(resource-id: string, metadata: key-value-pairs-list, metadata-schema: string)`

  It allows setting the metadata to be associated to the specific resource according to a pre-defined metadata schema.

- **get_metadata** `(resource-id: string)`

  It returns the list of the available metadata for a specific resource.

5.11 Hypermodelling Editor

Description

The Hypermodelling Editor is the component that is responsible for the modification or design of new hypermodels as “workflows”. To make the composition process easier for users, it will be provided as a web-based tool with a graphical user interface capable of assisting the user in the construction of models and the visualization of the saved hypermodels. This editor will not expose specific programmatic interfaces, but it will provide the Workflow Manager with workflow files for execution in a standardized format.

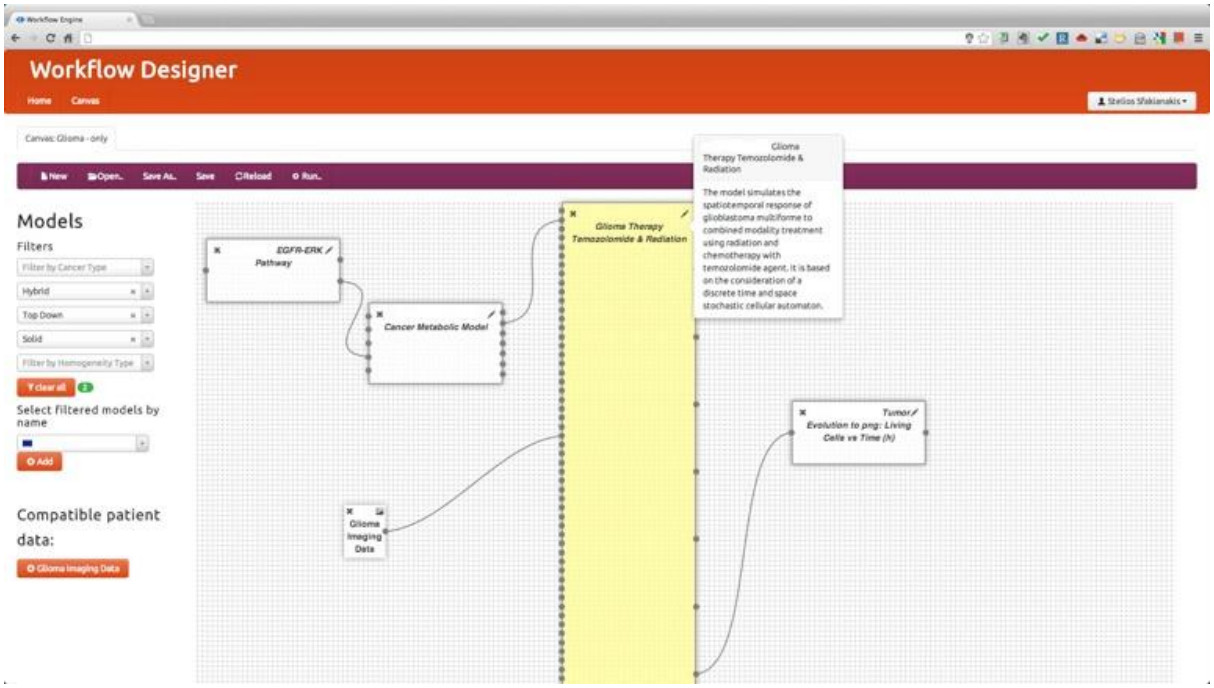

Figure 6: A mockup interface of the Hypermodelling Editor.

The editor supports the design of the hypermodels using simple “point ‘n’ click” and “drag ‘n’ drop” methods. The result is an abstract representation of the hypermodels without technical details being displayed or affecting the way the user designs them. The model information is retrieved from the model repository taking into account the semantic and syntactic annotations of the models. The abstract hypermodel is represented as an acyclic graph that is transmitted in a well-defined format to the Workflow Manager, where it’s made concrete by choosing the specific implementations of the models and setting up the execution context.

Functional requirements

The operations that must be supported by the Hypermodelling Editor have been presented in deliverable *"D5.1.1: The CHIC technical architecture"*. For consistency, we also include them in this deliverable (in textual format):

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

5.12 Hypermonitor

Once a hypermodel has been defined, a user can execute it with given input data, and monitor its execution, using a very powerful but somehow complex tool called Hypermonitor.

The Hypermonitor is a fat client application that connects to the CHIC Hypermodelling Framework, and lets the user login, see which hypomodels are available for execution, define complex workflows that orchestrate them, execute such workflows, and observe the logging as the hypermodel executes. This low level tool is essential for debugging the framework itself, for investigating issues with specific hypermodels and for monitoring the status of the production CHIC Hypermodelling Framework.

The Hypermonitor is a MAF3 C++ multi-platform implementation with QT-based user-interface. The module must connect to the repositories and to the hypermodelling framework. The Hypermonitor is an end-user module and thus does not expose an API.

The Hypermodelling Editor can invoke the Hypermonitor with the hypermodel to be run passed as a parameter, and can connect to the CHIC Hypermodelling Framework in order to run immediately a newly edited hypermodel. It connects to the hypermodelling infrastructure via a MAF3 proprietary communication layer[14], implemented, for efficiency, in XML-RPC.

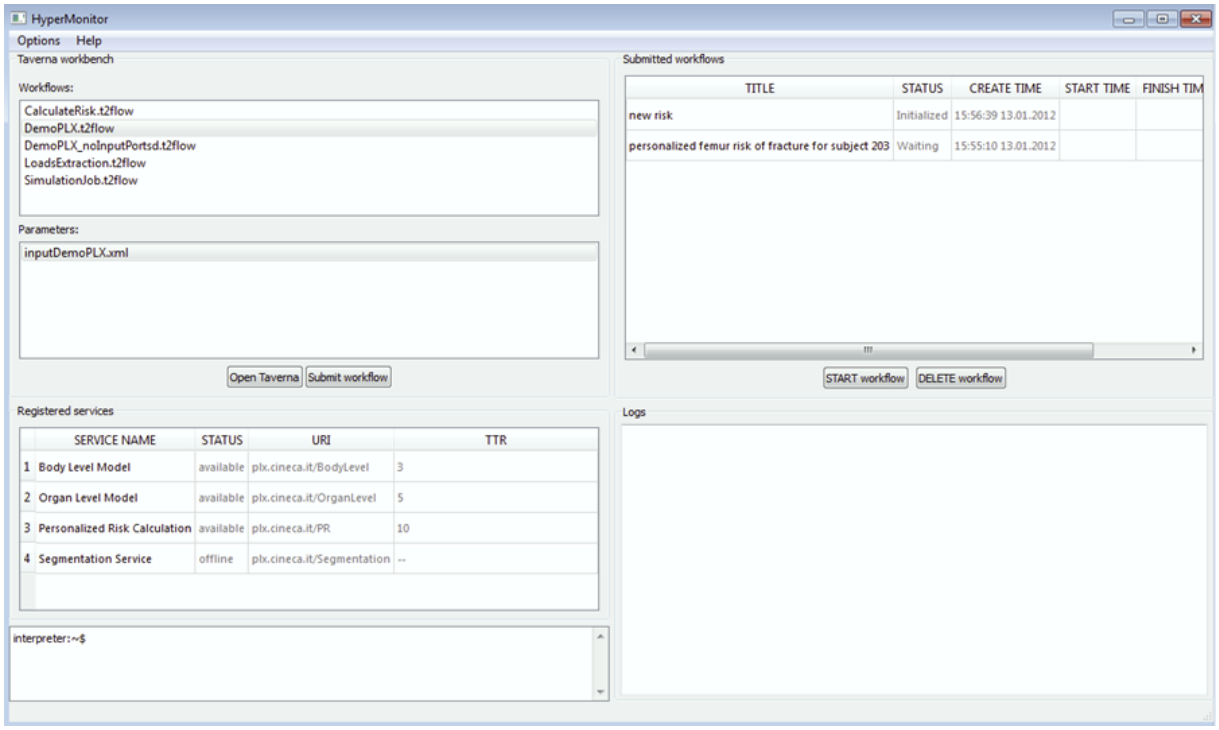

Figure 7 : The Hypermonitor GUI.

[14] Communication layer: it is the message exchange service between the different services and sub-models.

Functional requirements

The operations that must be supported by the Hypermonitor have been presented in deliverable *"D5.1.1: The CHIC technical architecture"*. For consistency, we also include them in this deliverable (in textual format):

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

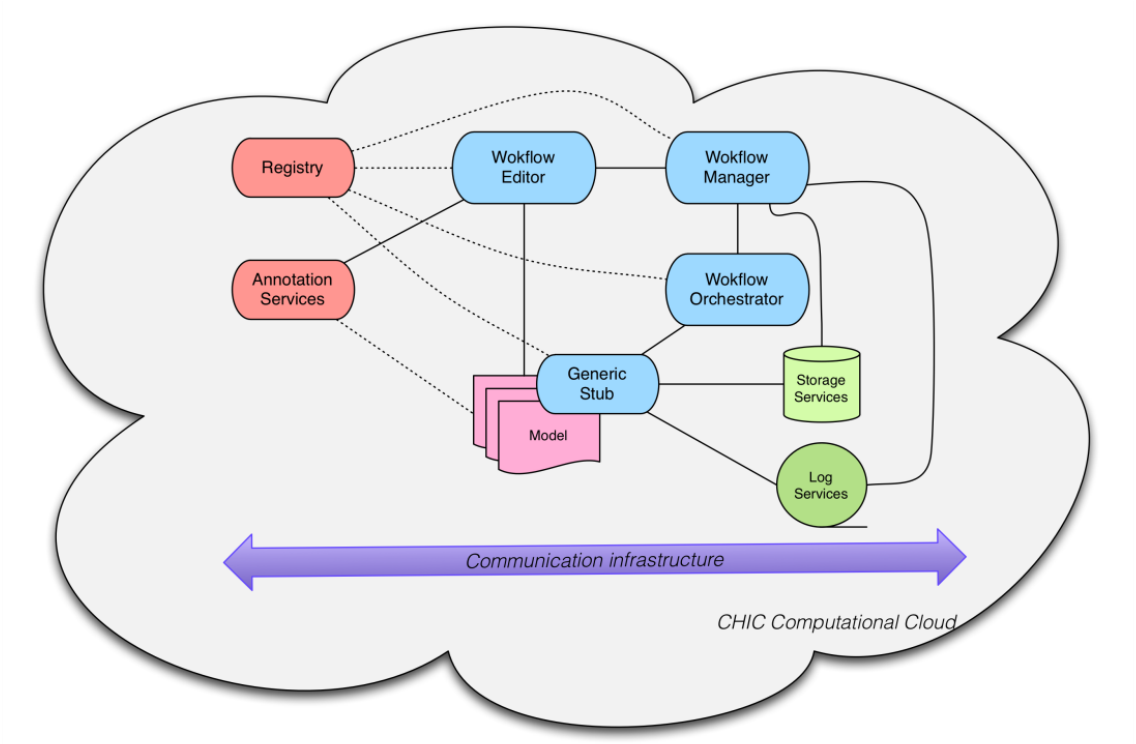

Figure 8 : A logical diagram of the CHIC Hypermodelling functional components and their interactions.

6 CHIC Hypermodelling Framework and CHIC Architecture

The CHIC architecture follows a generic, “archetypical” layered design that is depicted in the figure below.

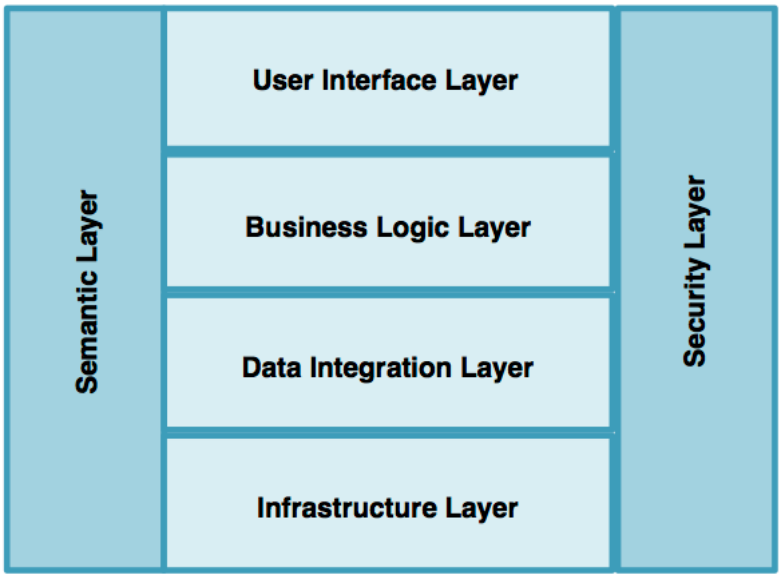

Figure 9 : The abstract layered view of the CHIC architecture

At the lower layer there are the cloud infrastructure and resource virtualization mechanisms that support the deployment of the CHIC platform. The “Data integration” layer above offers the data management functionality and provides the upload, storage, provenance, and linking of the data that are handled throughout the CHIC system. The “Business Logic” layer is the core of the CHIC platform providing the hypermodelling facilities, the linking of models and their execution. In the upper layer the “User Interface” includes the CHIC Portal and other end user applications which, through their graphical interfaces, support the end user scenarios. On the vertical axis, the semantic and security layers offer the respective non-functional (quality) attributes of the system.

The CHIC Hypermodelling Framework therefore resides in the “Business Logic” layer of the architecture. It has direct access to the data management facilities of the CHIC platform and takes advantage of the infrastructure layer for the efficient execution of the hypermodels, the security services for the authentication and authorization of the users, and the semantics and model annotation facilities for the models publication, discovery, and composition.

Further elaborating on this “Business Logic” layer, the CHIC Hypermodelling Framework is not a single software entity. It consists of a number of functional components that are described in more detail in the previous sections.

From the stakeholders presented in the “D5.1.1: The CHIC technical architecture”, the ones that are more related to the CHIC Hypermodelling Framework are the Model Providers, the Researchers, the Clinicians, the Software developers and the Administrators.

7 Hosting infrastructure

The system will be deployed in the premises of FORTH, in the form of a private computational and storage cloud. In terms of hardware resources, the cloud infrastructure allows for maximum elasticity and flexibility by effectively adapting to the load of any given time. The current minimal specifications include:

- 300 GB of RAM
- 9TB of storage
- 16 cores Intel® Xeon® Processor E5-2690 and 4 cores Intel® Xeon® Processor E7520 (Dell PowerEdge R720 and SC 1425 Servers series)

In terms of the software the OpenStack[15] open source cloud computing software has been installed on the machines using the Linux Ubuntu 12.04 operating system[16].

[15] https://www.openstack.org
[16] http://www.ubuntu.com

8 CHIC-HF Release plan

- 31/12/2013 (PM9): Specification of Generic Stub initially defined.
- 31/03/2014 (PM12): First version of the Tagging System released.
- 31/03/2014 (PM12): First CHIC Hypermodel Framework deployed in cloud infrastructure.
- 31/06/2014 (PM15): Hypermonitor released.
- 31/06/2014 (PM15): Model Wrapper deployed according to Generic Stub.
- 30/09/2014 (PM18): Specifications for Generic Stub updated.
- 30/09/2014 (PM18): Initial integration with the Storage Services completed.
- 31/03/2015 (PM24): Folksonomy and Ontology annotation and search services deployed.
- 31/03/2015 (PM24): Strongly encrypted data service ready.
- 30/06/2015 (PM27): Specification of Generic Stub finalized.
- 31/3/2016 (PM36): Hypermodel annotation services deployed.
- 31/3/2016 (PM36): Hypermodel editor, development and execution application ready.
- 30/7/2016 (PM40): Final CHIC Hypermodelling Framework deployed on main node.
- 30/9/2016 (PM42): Provision of distributed logging services.
- 31/03/2017 (PM48): Metahypermodels annotation completed.

9 Conclusion

Deliverable D7.1 has provided the initial hypermodelling specifications of the CHIC project. It is noted that the term *hypermodelling* means the process of developing hypermodels.

In the present report the following important items have been included inter alia: i) a set of definitions aiming at introducing the reader to the domain that the CHIC project belongs to and ii) a set of fundamental concepts and terms that are utilized by the CHIC project. The collaborative efforts of the CHIC consortium members have resulted to the collection and/or the proposition of a wide set of concepts and terms. In this document only a subset of these concepts and terms pertaining to its content has been included.

A significant part of the deliverable has been dedicated to the description of a specific component, the so called Generic Stub. The Generic Stub is particularly important because it serves as a "bridge" between two CHIC workpackages "WP6: Cancer Models and Hypermodel Design" and "WP7: Hypermodelling infrastructure". Deliverable "D6.1: Cancer hypomodelling and hypermodelling strategies and initial component models" has served as the driver for the formulation of the Generic Stub.

The Generic Stub along with the Model Wrapper are fundamental components for the overall CHIC modelling architecture, as they provide a generic abstraction of a model. This approach can provide a common base for the actions of all CHIC partners.

In addition to the two aforementioned components, detailed descriptions of all components of the CHIC Hypermodelling Framework have been included. The focus has been put on the functional perspective. We have presented where the CHIC Hypermodelling Framework is located in the overall CHIC architecture and what interconnections exist.

Last but not least, a release plan of the CHIC Hypermodelling Framework has been outlined.

It should be noted that during the development of the various components of the CHIC Hypermodelling Framework, changes in the architecture and/or the functional components may need to be made. Such modifications will be reported in future related documents.

Appendix – Abbreviations and acronyms

API	Application Programming Interface
BED	University of Bedfordshire
CHIC	Computational Horizons in Cancer
FORTH	Foundation for Research and Technology – Hellas
HF	Hypermodelling Framework
ICCS	Institute of Communication and Computer Systems
IDL	Interface Description Language
IT	Information Technology
REST	Representational state transfer
RPC	Remote Procedure Call
SOA	Service Oriented Architecture
SaaS	Software as a Service
SBML	Systems Biology Markup Language
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TUMOR	Transatlantic Tumour Model Repositories
UI	User Interface
URI	Uniform resource identifier
URL	Uniform Resource Locator
USFD	University of Sheffield
VM	Virtual Machine
VPH	Virtual Physiological Human
VPH-OP	Osteoporotic Virtual Physiological Human
WP	Work Package
XML	Extensible Markup Language