



Deliverable No. 10.4

The CHIC Hypermodelling Editor and orchestration environment

Grant Agreement No.: 600841

Deliverable No.: D10.4

Deliverable Name: The CHIC Hypermodelling Editor and orchestration environment

Contractual Submission Date: 30/11/2016

Actual Submission Date: 31/03/2017

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)	
CO	Confidential, only for members of the consortium (including the Commission Services)	X



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.:	D10.4
Document name:	The CHIC Hypermodelling Editor and orchestration environment
Nature (R, P, D, O) ¹	R
Dissemination Level (PU, PP, RE, CO) ²	PU
Version:	1.0
Actual Submission Date:	31/03/2017
Editor: Institution: E-Mail:	Stelios Sfakianakis FORTH-ICS ssfak@ics.forth.gr

ABSTRACT:

This deliverable describes the CHIC Hypermodelling Editor as the fundamental tool for the design of new hypermodels. The Editor presents an intuitive graphical user interface where the users construct their hypermodels by connecting different hypomodels in a visual way. The constructed hypermodels are technically validated through the Editor's backend checks on the connections based on the semantic and syntactic annotations of the models' parameters. The Editor is fully integrated with the CHIC model and data repositories and the VPH Execution Framework allowing the execution of the hypermodels and the retrieval of their results.

KEYWORD LIST:

CHIC orchestration, dataflow programming, Hypermodelling editor, visual programming

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 Union Seventh Framework Programme under grant agreement no 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.

MODIFICATION CONTROL			
Version	Date	Status	Author
0.1	01/11/2016	Draft	Stelios Sfakianakis
0.5	20/11/2016	Draft	Stelios Sfakianakis
0.8	15/01/2017	Draft	Stelios Sfakianakis
1.0	31/03/2017	Final	Stelios Sfakianakis

List of contributors

- Stelios Sfakianakis, FORTH
- Kostas Marias, FORTH
- George Zacharioudakis, FORTH
- Manolis Tsiknakis, TEI-C
- Simone Bna, CINECA
- Nikolaos Touser, ICCS
- Eleni Georgiadi, ICCS
- Georgios Stamatakis, ICCS

Contents

1	EXECUTIVE SUMMARY.....	7
2	INTRODUCTION	8
3	DESIGN AND RATIONALE	11
3.1	VISUAL PROGRAMMING	11
3.2	MANAGEMENT OF MODELS AND HYPERMODELS	12
3.3	INTERACTIONS WITH THE REST OF THE PLATFORM	15
3.4	EXECUTABLE REPRESENTATION OF HYPERMODELS	16
3.5	USE OF SEMANTICS IN THE EDITOR.....	16
4	ARCHITECTURAL CONSIDERATIONS	19
4.1	INTERACTIONS DURING THE DESIGN OF HYPERMODELS	21
4.2	HYPERMODEL EXECUTION	22
5	A GUIDED TOUR IN THE EDITOR’S USER INTERFACE	24
5.1	DESIGNING HYPERMODELS	24
5.2	HYPERMODEL EXECUTION	29
6	IMPLEMENTATION.....	33
7	CONCLUSIONS	34
	APPENDIX 1 – ABBREVIATIONS AND ACRONYMS	35

Table of Figures

Figure 1 A simplified view of the scientific method	9
Figure 2 Relations between models and meta-models.....	11
Figure 3 A hypomodel as shown in the Editor.....	12
Figure 4 A generic schema for the information managed by the Hypermodelling Editor.	14
Figure 5 The interactions between the Editor and the backend CHIC services.....	15
Figure 6 Model selection, alignment, and combination (Schulz et al, 2011)	17
Figure 7 The Hypermodelling Editor is a critical component in the CHIC Research domain ...	19
Figure 8 The CHIC orchestration platform	20
Figure 9 Design a new hypermodel sequence diagram.....	21
Figure 10 A simplified view of the trials-experiments design of the InSilico Trial repository .	22
Figure 11 Hypermodel execution sequence diagram.....	23
Figure 12 The mail toolbar of the Editor	24
Figure 13 The list of available models as shown by the Editor.....	25
Figure 14 Information retrieved from the CHIC Model Repository for a given hypomodel....	26
Figure 15 Linking two models together.....	27
Figure 16 An example of a hypermodel designed in the CHIC Hypemodelling Editor	28
Figure 17 Information entered by the user during the registration of a new hypermodel....	29
Figure 18 Loading an existing hypermodel.....	29
Figure 19 The xMML description of a hypermodel	30
Figure 20 Setting the execution inputs	31
Figure 21 A visual indication about the change of status of an execution.....	31
Figure 22 Desktop notification for the completion of an execution	31
Figure 23 The history of past and current experiments.....	32

1 Executive Summary

The purpose of this document is to document the design, the functionality, and the architecture of the CHIC hypermodelling editor. The Hypermodelling Editor of CHIC is the end user software component for designing, triggering and monitoring the execution, and retrieving the results of the CHIC integrative hypermodels. The editor supports the construction of new hyper models in a visual and graphical way. It features a “box and arrows” representation of the hypermodel, where each hypomodel is shown as a rectangle and each data exchange link between two hypomodels is depicted as a line connecting them.

The editor, however, is not an autonomous component in the CHIC platform: it builds upon and integrates with the data, model, and hypermodel repositories and the VPH hypermodel execution framework. The primary goal of the Editor, that is to design new hypermodels, requires accessing the Model Repository and the Semantics Repository, in order for the user to browse, search, and retrieve the available hypo- and hyper- models based on their semantics and metadata based descriptions. At the level of the Hypermodelling Editor no implementation specific information is required for the models that the users have access to. On the other hand, such information is obviously required in the execution framework in order for the designed hypermodels to run. Therefore, there’s a tight integration between the different components to collectively implement the design and execution of the CHIC integrative models.

2 Introduction

The CHIC platform is fundamentally an infrastructure aiming to foster the development of elaborate and reusable **integrative models** (hypermodels) that address the challenges in cancer research and treatment using a multilevel and multiscale approach. This is due to the fact that cancer progression is affected by interconnected processes that operate in multiple time and spatial scales. There's a need therefore to combine the different **component models** that try to simulate specific time-space scales into higher-order **hypermodels** which extend the application range and the potential for clinical adaptation and relevance.

Text Box 1. Some important definitions from Deliverable 7.1

- **Model:** 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.
- **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 hypermodel.
- **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.
- **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.

In the process of coming up with a hypermodel that can describe a natural phenomenon and biological or other process, a researcher will most probably follow the scientific method of observing the phenomenon, formulating a hypothesis that predicts it, and performing several experimental tests to validate or falsify these predictions (Figure 1). In the context of the CHIC, the Hypermodelling Editor strives to be the main tool for the construction of *in silico* hypotheses and experimentation. The users are computational biologists, clinical researchers, and other scientists in the

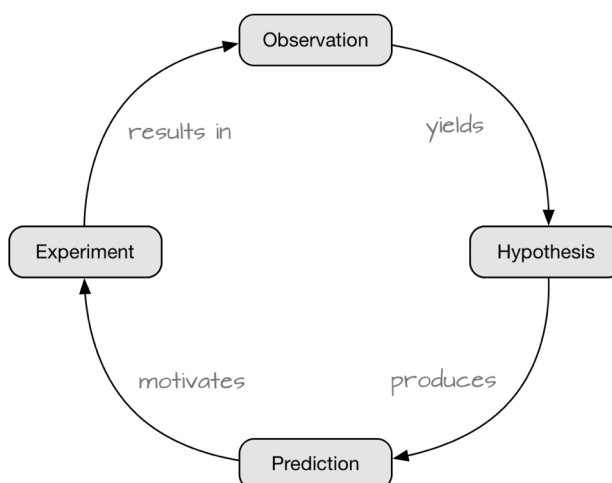


Figure 1 A simplified view of the scientific method

computational biomodelling community that try to gain an understanding on the underlying mechanisms governing the characteristics and function of complex biological systems in health and disease. The emphasis of this tool is not the development of the individual “atomic” models that operate in a single spatio-temporal scale or organizational level (e.g. molecular): such models are better developed using existing frameworks, software libraries, and computational tools, and tested in their specific simulation micro-environments. Rather, the Editor’s *raison d’être* is to provide a user-friendly environment for the visual construction of complex integrative models that operate in multiple scales and levels by the incorporation of other, “simpler” and narrower in scope models. To this end, the Editor abstracts away from the internals and the implementation details of the (hypo)models that fuse into hypermodels, focusing only on their externally visible behavior, the relevant classification, and other annotation properties. For more details on the component (elementary) models, Deliverable 6.1 (“Cancer hypomodelling and hypermodelling strategies and initial component models”) lists the initial list of the CHIC hypomodelling and proposes a cancer model categorization and annotation framework.

The overarching objective of the CHIC Hypermodelling Editor (the “Editor” from now on), that is the construction of comprehensive hypermodels, introduces a number of requirements and desired features to its design. The following are some of these primary requirements to guide the development of the Editor:

- Expose information about existence and availability of models that can be composed together,
- Present interconnection possibilities and check the validity of the models’ integration,
- Indicate the models that need to be developed in order to “fill in the gaps”,
- Support the visually construction of the hypermodels,

- Integrate with the CHIC execution infrastructure in order to test the runtime characteristics and results of the designed hypermodels.

The rest of this document will expand on the design and implementation of this high-level functionality of the Editor. We begin with the rationale behind its design and the provided user experience are presented. Subsequently, we present the technical details, starting with the description of the Editor as a CHIC architectural component and its role and interactions inside the CHIC platform. Finally, a detailed description of an exemplary user session with proper illustrations of the different steps and the user interface elements are provided.

3 Design and Rationale

3.1 Visual Programming

The Editor features a graphical user interface for the visual construction of the hypermodels. This is a major design choice set early on in the requirement analysis of the Editor because a graphical user interface presents a lower barrier to entry, especially for (ICT) inexperienced researchers. After all, as everyone knows, “a picture is worth a thousand words”, meaning that it is much easier to get a message across using pictures and visual embellishments than it is with words³. It is evident that a scripting or compiled programming language can possibly be a lot more expressive and flexible than a visual notation at the cost of a steeper learning curve and the inclusion of more “ceremony” (irrelevant details). There’s also a clear trade-off in complexity and flexibility, and the current design choices move a lot of complexity in the platform and the implementation of the tools, such as the Editor and the Execution Framework, relieving the users of additional cognitive load: the users of the Editor need to focus only on the domain specific aspects of the hypermodelling building task.

The provision of such visual interface in the Editor allows for an additional advantage: there is a shift of focus from the implementation details of each hypomodel to the abstract level of *metamodels* and the most applicable hypermodelling strategy to use. As shown in Figure 2 below, the metamodel of a given model is a stripped version of it where any technical artefacts of its implementation are removed, leaving only the domain specific information about its role, biological characteristics, etc. (see also **Text Box 1** for the definition of a metamodel).

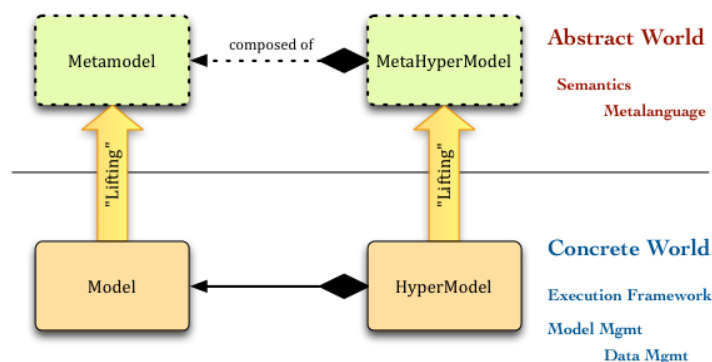


Figure 2 Relations between models and meta-models

The Editor therefore with its visual programming interface supports the abstract representation of the models and hypermodels and effectively hides the complexities of the “concrete world”, such as the execution requirements, the management of input and output data, etc. (the “sausage factory” details).

³ Even in this document, we make gratuitous use of images and figures to convey our messages.

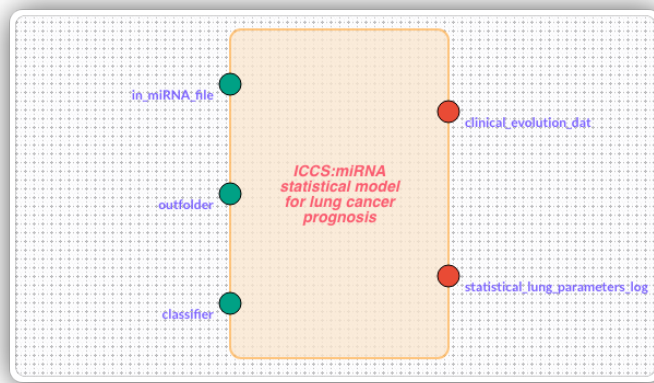


Figure 3 A hypomodel as shown in the Editor

In the graphical representation of the hypermodels in the Editor, each hypomodel is represented by its abstract view (metamodel) and is shown with a minimal set of information that includes only its inputs and outputs, as Figure 3 depicts. The users have of course full access to the whole set of metadata attached to a given model, but in the “drawing canvas” of the Editor each model is represented as a rectangular box with a set of input and output ports (circles) that allow the user to connect this model with other models. Each connection between two models represents a flow of data from one output port to one input port of the corresponding models. The underlying paradigm is therefore that of **Dataflow**, where the hypermodel is a (directed) graph whose nodes are models and data flows from model to model through the edges of the graph.

Each hypermodel can be either a cyclic or acyclic, depending on the models that participate in it. An acyclic graph does not contain cycles and therefore it can have a serialized execution plan at runtime, i.e. the models can be executed one after the other. Some of the CHIC models though are *strongly coupled*, which means that during execution they exchange data through some special “dynamic” ports. These dynamic ports are effectively channels that allow the communication of two concurrently running models and therefore they permit the creation of cycles if, for example, there’s a bidirectional dependency between two models where each model sends and receives data through these channels. The Editor supports the creation of both cyclic and acyclic hypermodels but not the mixing in the same hypermodel of strongly coupled and “weakly” coupled models. This is due to the fact that at the VPH Execution Framework the two types of hypermodels (“weakly-” and “strongly-coupled”) have different execution requirements and are handled differently.

3.2 Management of models and hypermodels

The custodian of models in the CHIC platform is the Model Repository. This is component that has the latest versions of the CHIC models with all the required information that allows their efficient indexing and retrieval. The hypermodels designed in the Editor are also models and therefore the Editor updates the Model Repository whenever a new hypermodel is built or an existing one is updated. In addition to the name (title) and the description information, the Editor provides the following annotation properties for the new hypermodels:

- Inputs (multiple)
- Outputs (multiple)
- Author (“creator”) information
- Publications related information
- Domain specific metadata, like temporal and spatial scale information
- Versioning information, with links to previous versions (the "parent" version that the current one was based on)
- Usage and runtime information, e.g. number of times executed, trial context information for each run, etc.

Whenever such information is sent to the Model Repository it is encoded in the form that the repository supports and uses the agreed coded values (for example, the data types for the inputs and outputs).

The following picture shows a simplified view of the underlying “resource oriented” model of the information that the Editor manages (produces and consumes) for each hypermodel (The entities shown with circles represent “Resources”, identified with URIs, in the sense of Resource Description Framework (RDF)⁴, while the rectangles represent “literals” i.e. plain values):

⁴ Lassila, Ora, and Ralph R. Swick. "Resource description framework (RDF) model and syntax specification." (1999).

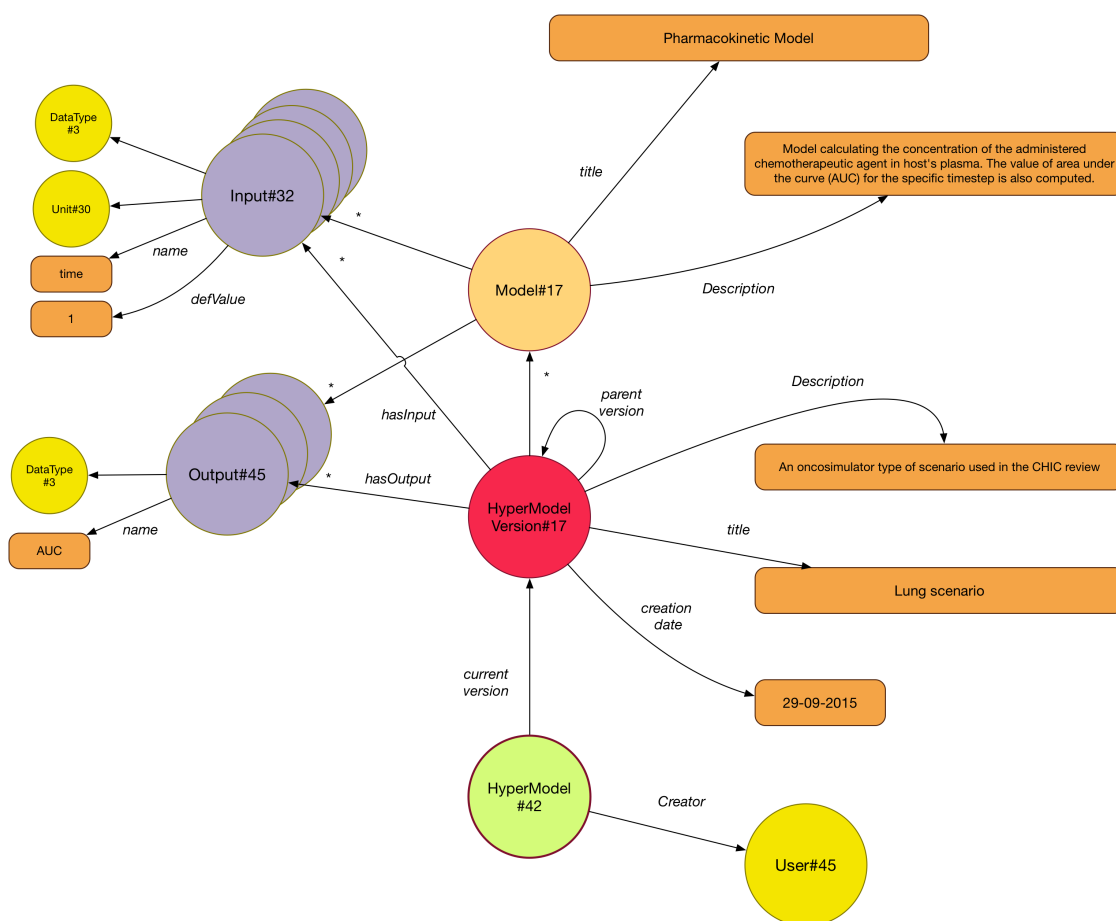


Figure 4 A generic schema for the information managed by the Hypermodelling Editor.

Some remarks on this “resource” view of the information managed by the Editor are in order:

- A hypermodel is created by a user during her/his interaction with the Editor.
- Each hypermodel can have multiple versions, where each version builds upon a previous one (the “parent” version)
- Each hypermodel bears some descriptive metadata, like the title, description, and creation date, but in fact this information can change from one version to another.
- In addition to the descriptive metadata, each hypermodel version consists of its specific set of models.
- The models that collectively comprise a given hypermodel version carry all the annotations managed by the Model Repository, e.g. the names, data types, and default values of their inputs and outputs.
- Some of the inputs and outputs of the models participating in a given hypermodel version are also inputs and outputs of the hypermodel itself.

3.3 Interactions with the rest of the platform

The Execution Framework provides the engine for the actual execution of the hypermodels in the CHIC computational cloud. This is where the hypomodels are run, exchanging data and signals, and producing the final results of the hypermodels. The Hypermodelling editor is the front-end to this infrastructure. The editor allows the end users (computational biologists, researchers, etc.) to design new hypermodels and then submit them to the execution framework in order to test their research hypotheses or validate their assumptions. On the other hand, the model and data repositories are the core components of the CHIC data and model management layer. They provide query-based retrieval of the model and data descriptions in compliance with the metadata schemas adopted in CHIC. To successfully support the envisaged functionality, the Editor needs to cooperate with all of these components. Figure 5 graphically depicts all the relevant communication.

As shown in the figure, the Hypermodelling Editor and the VPH-HF execution framework communicate both directly and indirectly by exchanging information through various data repositories: the model repository, which contains the full description of all hypo- and hyper-models; the clinical data repository, where all patient-specific inputs for the models are stored; and the simulations repository (“inSilico Trial repository”), which contains all input and output sets involved in each execution of each model. The integration of the repositories in the VPH hypermodelling framework allows for the “separation of concerns” and efficiency in the inter-component communication since, most of the times, references to the data (in terms of URIs) instead of the real data are exchanged.

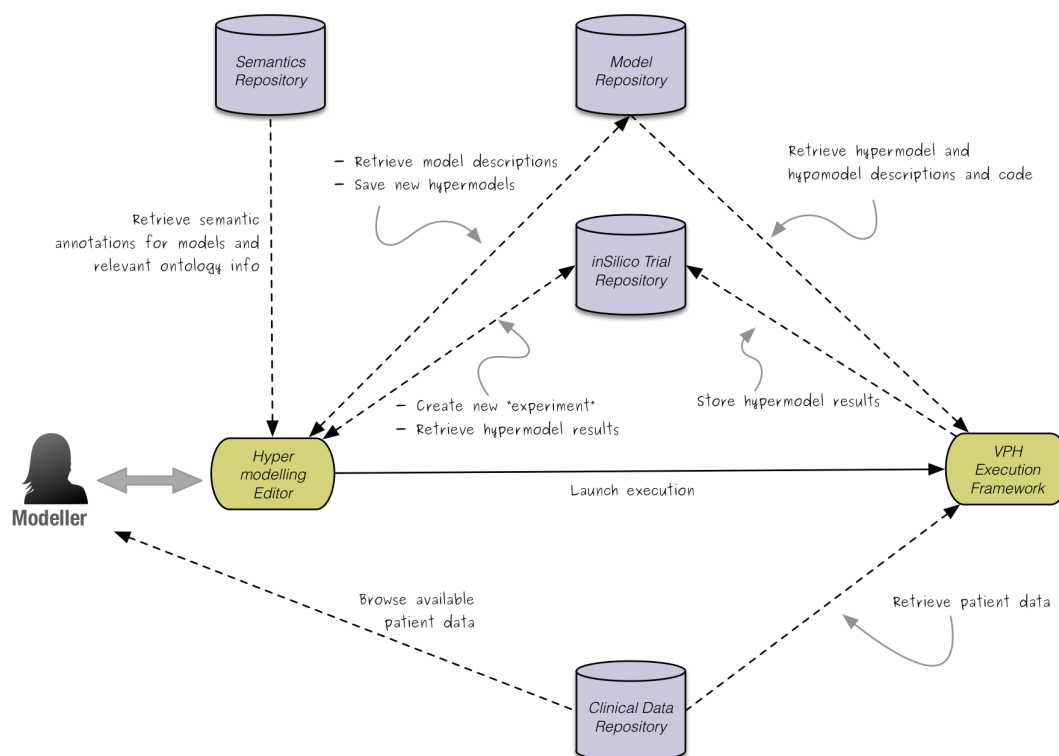


Figure 5 The interactions between the Editor and the backend CHIC services

More detailed descriptions of these interactions and other technical details can be found in Section 4.

3.4 Executable representation of hypermodels

The execution of hypermodels is taken care by the VPH Execution Framework as already mentioned in Section 4. The integration happens through the use of the xMML⁵ as the common hypermodelling language. Essentially, the xMML description of a hypermodel is its “executable code” as far as the Editor is concerned. This format captures the graph representation of the hypermodel in the Editor, i.e. the nodes (models) and the directed edges between them (the data connections).

The xMML format is able to support the representation of both the “weakly-coupled” and the “strongly-coupled” hypermodels. In the case of the strongly coupled hypermodels the connections are always between the “dynamic” ports of the models whereas in the weakly coupled hypermodels the models are connected through their normal input and output ports. Because of the different execution engines, MUSCLE2⁶ in the case of strongly coupled hypermodels versus Taverna Workbench otherwise⁷, each hypermodel in the Editor contains either only strongly coupled models or non-strongly coupled ones, and this is also portrayed in the corresponding xMML description. To convey this information to the VPH Execution Framework, the Editor adds an XML attribute to the xMML description of the hypermodel as a boolean flag with a “true” value if the hypermodel is a strongly coupled one. Another extension point for the use of xMML in the CHIC platform is the use of the model UUIDs (universally unique identifiers) minted by the Model Repository as the identification mechanism for the models used in a hypermodel.

The xMML specification, version 0.4, can be found online at <https://github.com/blootsvoets/xmml> and includes additional features, such as the units used in the connections or the inclusion of the time and space scale information for the models, that are not used because most of this information exists already in the Model Repository that both the Editor and the VPH Execution Framework access.

3.5 Use of semantics in the Editor

The semantic annotation of hypo models can be instrumental in the development of new hypermodels since these annotations can guide the users to the best way they can combine them, or

⁵ Borgdorff, Joris, Eric Lorenz, Alfons G. Hoekstra, Jean-Luc Falcone, and Bastien Chopard. "A principled approach to distributed multiscale computing, from formalization to execution." In *e-Science Workshops (eScienceW)*, 2011 IEEE Seventh International Conference on, pp. 97-104. IEEE, 2011.

⁶ Borgdorff, Joris, Mariusz Mamonski, Bartosz Bosak, Krzysztof Kurowski, M. Ben Belgacem, Bastien Chopard, Derek Groen, Peter V. Coveney, and Alfons G. Hoekstra. "Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment." *Journal of Computational Science* 5, no. 5 (2014): 719-731.

⁷ Deliverable 7.4 contains all the gory details.

to have a coarse categorization of them based on domain specific similarity criteria. Schulz et al⁸ present a range of possibilities for taking advantage of the semantic information attached to computational models and data, as shown in the Figure below:

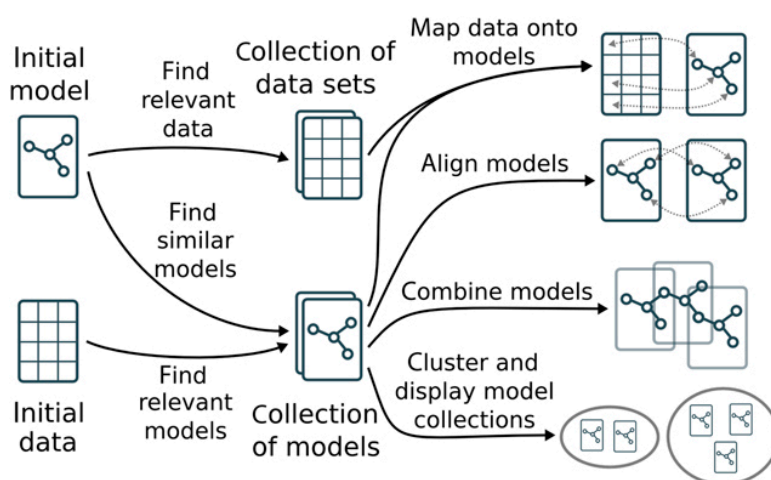


Figure 6 Model selection, alignment, and combination (Schulz et al, 2011)

For the CHIC Hypermodelling Editor we envision two main areas that the semantics can play a significant role, in the so called *Reactive* and *Proactive* usage scenarios:

- On being Reactive: The Editor reacts to the actions of the users in order to check and validate them. It uses the semantic payload of the models and tries to check, for example, whether two hypomodels can be used together in a hypermodel or if they can even be linked by a connection so that the output of one can be given as input to the other.
- On being Proactive: The Editor gathers information about the user and the goal s/he wants to achieve during the design of a hypermodel. It may be the case that it can provide recommendations in a proactive way by intelligent “guessing” of the next hypo models to put in the drawing canvas based either on the models already selected by the user him/herself or according to the similarities to the hypermodels designed by other users.

The Editor leverages the semantics that escort the models in order to support these Reactive and Proactive behaviours. In the following we present concrete examples for these intelligent behaviours.

⁸ Schulz, Marvin, Falko Krause, Nicolas Le Novère, Edda Klipp, and Wolfram Liebermeister. "Retrieval, alignment, and clustering of computational models based on semantic annotations." *Molecular systems biology* 7, no. 1 (2011): 512.

3.5.1 Examples for Reactive behaviour: Consistency checking of the connections

The connections between hypomodels represent exchange of data, either as a stream of values in the case of “dynamic” inputs and outputs, or as a single value produced by a hypomodel before its termination and used by another one when commencing its execution. The consistency check of these links can use information about the **Units** used (e.g. for time, or volume), the **Data type** (e.g. integer, Boolean flags), the “**semantic type**” that provides a domain specific meaning for the exchanged information (e.g. “cell cycle duration”), etc.

Semantics can play a role both as facilitators for using a common terminology, as in the case of units, and as a translation and semantic linking facility for providing inference and high level matching of the used terms, as in the case of the “semantic type”.

3.5.2 Examples for Proactive behaviour: Model recommendations

The “interface” (i.e. inputs and outputs) of (hypo) models can be also used to provide suggestions on what models to be used next. For example, the data types and semantic type of a model’s outputs can be used as filters for the models that can be linked to. Therefore, the Editor can filter the list of possible models that can be selected and put in the hypermodel. Similar functionality can be supported when the semantic annotations of the models with respect to the “13 perspectives” (See Deliverable D6.1) are taken into account. For example, the “spatial scale” (Perspective II) provides a characterization for the level (atomic, molecular, cellular, etc.) that a hypo model operates on. Based on this information, and possibly the structure of existing hypermodels of the same or different users, the Editor can again propose a limited list of hypomodels for the user to include in the current hypermodel. Of course, in any case, these will be suggestions and not strictly enforced decisions on how the hypermodel should be built.

4 Architectural considerations

The CHIC platform aspires to address two primary roles: First, to foster the VPH research and *in Silico* experimentation, focusing on the construction, execution, and reuse of VPH mathematical and computational hypermodels on representative cancer types; and secondly, to provide the tools and the infrastructure for the deployment of select artifacts of this research into clinical practice and every day healthcare. The Editor addresses needs in the former case, in the CHIC Research Domain, where the model construction and experimentation take place (Figure 7). The results of this Research Domain are potentially “transferred” to the Clinical Domain where they are primarily used by CRAF (see Deliverable 10.5).

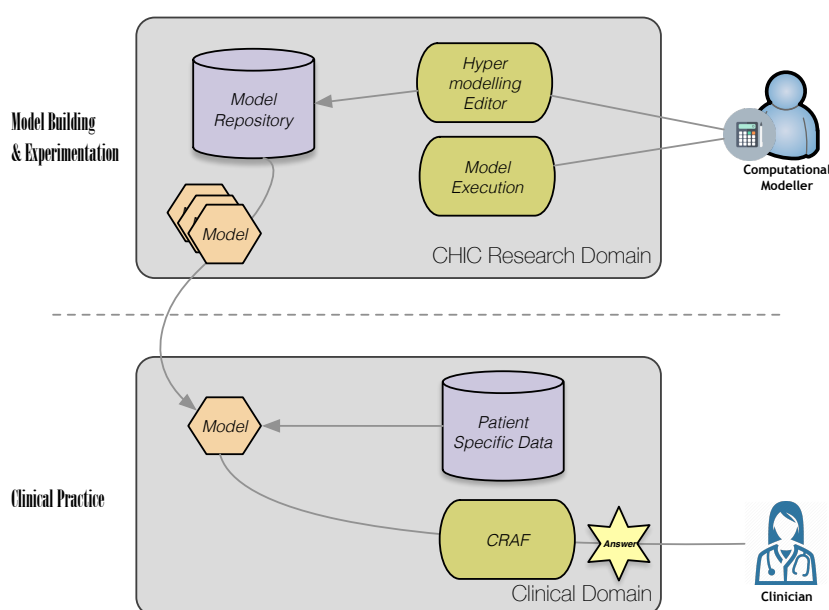


Figure 7 The Hypermodelling Editor is a critical component in the CHIC Research domain

In the CHIC Research Domain, the Editor is part of the Hypermodelling Framework. The Hypermodelling Framework includes, in addition to the Editor, the hypermodelling execution environment, which is responsible for the execution of the models, and the *inSilico* Trial Repository that stores the input and output of the *in silico* simulations along with the complete profile of each simulation. Figure 8 depicts the main components of the Hypermodelling Framework and their interactions with each other and with additional services of the CHIC research platform.

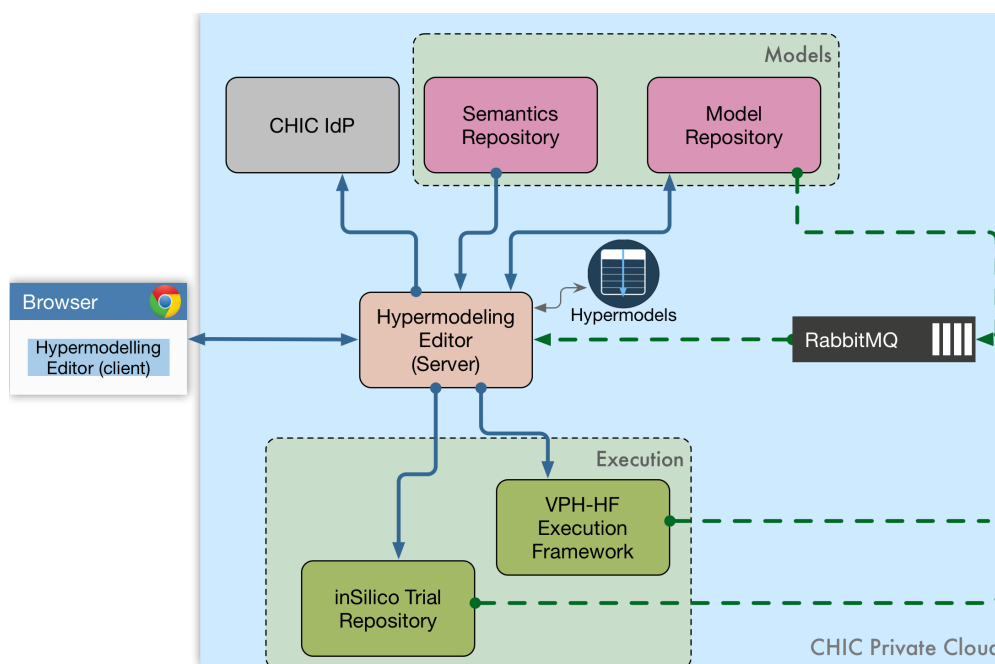


Figure 8 The CHIC orchestration platform

Very briefly, the responsibilities of the major subsystems that comprise the CHIC hypermodelling infrastructure are the following:

- The VPH Execution Framework (VPH-HF) provides the engine for the actual execution of the hypermodels in the CHIC computational cloud. This is where the models run, exchanging data and signals, and producing the final results of the hypermodels. The Execution Framework is a whole suite of components (e.g. Director, the Taverna workflow management system⁹, etc.) and the interested reader can refer to Deliverable 7.4 for more details.
- The model and semantics repositories are the core components of the CHIC model management layer. They provide query-based retrieval of the model descriptions in compliance with the metadata schemas adopted in CHIC (Deliverable 8.4).
- The Hypermodelling editor is the front-end to this infrastructure. The editor allows the end users (computational biologists, researchers, etc.) of the CHIC platform to design new hypermodels and then submit them to the execution framework in order to test their research hypotheses or validate their assumptions. The models to be used as building blocks are retrieved from the Model Repository and their descriptions are augmented with any semantic annotations they have in the Semantics Repository.

The authentication of the users happens through the CHIC Identity Provider. The Editor contacts the CHIC Identity Provider for the user authentication using the SAML-compliant mechanisms and the Secure Token Service (STS) for getting tokens to contact the backend CHIC services (Deliverable

⁹ <http://www.taverna.org.uk/>

5.2.2). Additionally, a message broker (RabbitMQ¹⁰) is introduced as a middleware component for the (almost) real time delivery of important notifications and messages between the services of the Hypermodelling Framework.

In the following paragraphs, we present two exemplary scenarios for the CHIC hypermodelling framework. These use cases are in the core of the CHIC hypermodelling domain, namely the design of a new hypermodel and the execution of the hypermodels and retrieval of the results. For each scenario, we present a sequence diagram, showing the components that interact and the messages that they exchange.

4.1 Interactions during the design of hypermodels

Designing a new hypermodel starts with the user entering the Hypermodelling Editor, as shown in the sequence diagram below. The Editor needs to verify the authenticity of the user and therefore contacts the User Authentication service (the CHIC Identity Provider), which is part of the security services. Of course, in the case that the user has already logged into the CHIC platform (e.g. through the portal) then he/she does not need to provide his/her credentials again but there is still user validation going on behind the scenes to make user that this is a legitimate CHIC user.

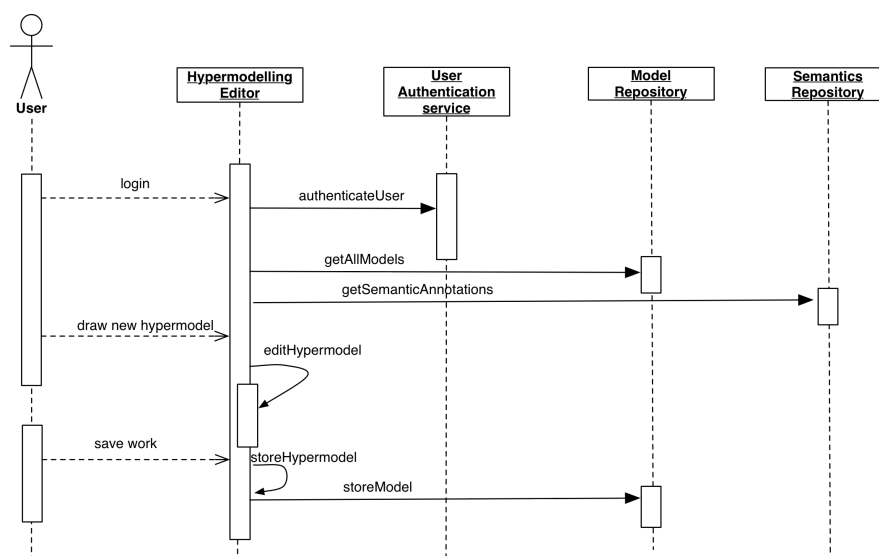


Figure 9 Design a new hypermodel sequence diagram

After the user's successful login, the Editor contacts the Model Repository and retrieves the available models using the "getAllModels" operation. The Editor subsequently contacts the Semantics Repository to retrieve all the semantics-based annotations, for example the parameters metadata and the model specific metadata for each of the managed models. Based on the retrieved hypomodels and their metadata, the user is then able to "draw" (design) a new hypermodel by

¹⁰ <https://www.rabbitmq.com/>

linking their inputs and outputs in a graphic way. At any time, the user is able to save her/his work and then the constructed hypermodel is saved in the local hypermodel repository as a new version. Optionally, the hypermodel can also be saved as a new model in the Model Repository using the “storeModel” operation.

4.2 Hypermodel execution

The execution of a hypermodel is a more complex use case. The first complexity is the setup of a new *trial* and new *experiment* that is linked with the selected hypermodel. These concepts are described in Deliverable 8.1 but basically a trial binds together a specific hypermodel and a set of patient or other data (“subjects”) that are either used as input or produced by the execution(s) of this hypermodel. Figure 10 below presents a simplified view of this design. An experiment in this design corresponds to a single execution of the hypermodel using and producing some of the linked “subjects”.

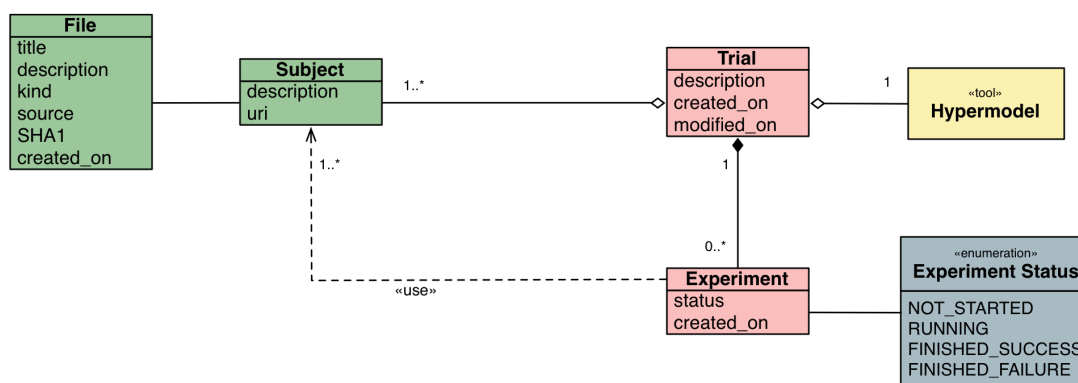


Figure 10 A simplified view of the trials-experiments design of the InSilico Trial repository

The second “complexity” (or interesting point) in this use case relates to the actual execution of the hypermodel. This includes the communication with the Hypermodelling Execution Framework, the retrieval of the model implementation details from the Model Repository, the monitoring of the execution, and the presentation of the final results.

In more details, Figure 11 presents the components and their interactions during the submission of a hypermodel for execution. More specifically, this sequence diagram includes the following activities:

1. **Login:** The user logs into the Hypermodelling editor and the editor authenticates him/her by contacting the CHIC user authentication service. This is identical to the authentication process described in the previous paragraph.
2. **Hypermodel selection:** The Editor then retrieves the available hypermodels from its local hypermodel database. The user selects one of those hypermodels and the most recent version of the selected one is subsequently loaded.
3. **Launch of execution.** The user presses a button and, in the dialog that appears, s/he can specify the input values to be used for the execution of the hypermodel. The Editor is then performing the following tasks:
 - a. Stores the xMML description of the hypermodel in the Model Repository or updates it if needed.

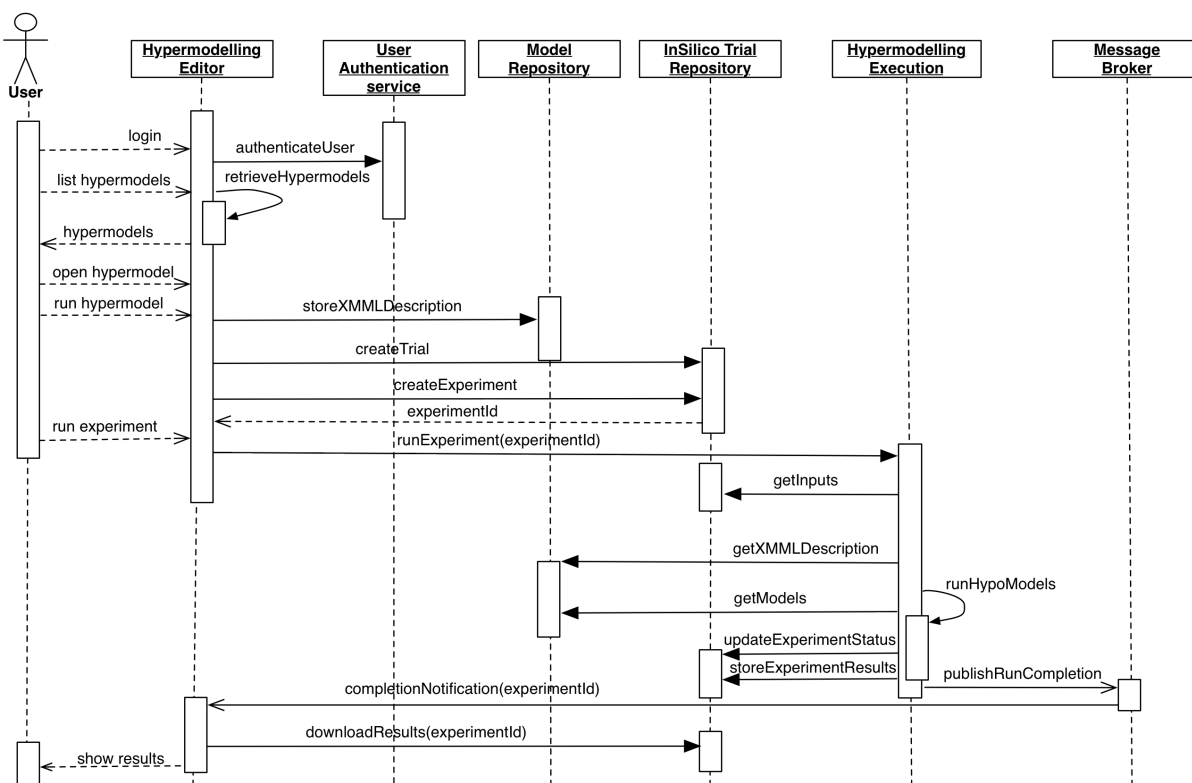


Figure 11 Hypermodel execution sequence diagram

- b. Creates a “trial” and the corresponding experiment in the inSilico Trial Repository, and gets back the new experiment id. During the creation of the Trial, the provided input data are also saved as a File (of the “Baclava” format¹¹) associated with the input Subject in the inSilico Trial Repository.
 - c. Submits the information about the new experiment (i.e. the experiment Id and the associated hypermodel) in the CPH Execution Framework
4. **Hypermodel execution:** The execution framework retrieves the experiment information (most importantly, its inputs) for the inSilico Trial Repository. It also contacts the Model Repository to get the xMML description of the hypermodel to run and the implementation details for each hypomodel included in the hypermodel. Additionally, it updates the status of the experiment (Figure 10) to “RUNNING” and finally to “FINISHED_SUCCESS” when all the hypomodels have completed their execution. After the completion of the execution, it also saves the final results in the InSilico Trial repository as the output “Subject” of the experiment, and publishes a new “experiment completed” message in the Message Broker.
5. **Presentation of results:** The Editor is notified about the successful termination of the hypermodel by receiving the notification of the completed experiment from the Message Broker. It subsequently retrieves the complete information set for the specific experiment (i.e. hypermodel execution) which includes the produced files and results. Finally, it shows these results to the user.

¹¹ The so called Baclava format is the XML-based file format used by the Taverna Workflow Management System (<http://www.taverna.org.uk/>) to save input data and results of the workflow runs.

5 A guided tour in the Editor's user interface

The Editor is accessible through a modern web browser so no installation or other “client” side configuration is needed. It supports the single sign on (SSO) of the CHIC users, which means that the users that have already logged into the CHIC portal or any other application of the platform are immediately identified by the Editor and do not need to enter their credentials (e.g. passwords) again. Therefore, the user identity and their roles are immediately available and subject to any authorization decision made by the Editor or any of the downstream architectural components, such as the CHIC HF, which are reached through the Editor¹².

The user interface of the Editor is intentionally simple so that its users can familiarize with it very quickly. A toolbar on the top of its web page contains a number of buttons for its core functionality (Figure 2).

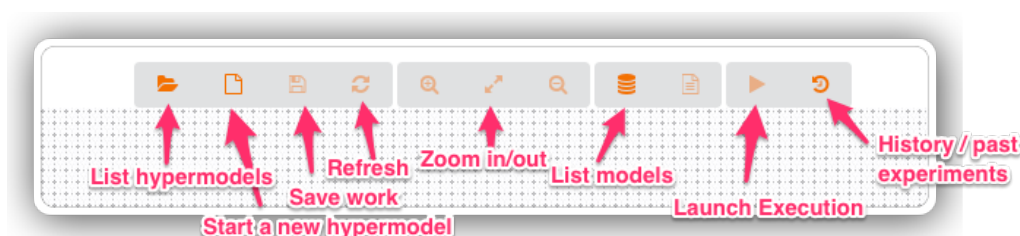


Figure 12 The main toolbar of the Editor

5.1 Designing hypermodels

The selection of the models to be composed into a new hypermodel can be started by the user pressing the “List models” button. A dialog shown in Figure 13 appears that lists the available models together with some important information for each of them.

¹² This is supported by the SAML machinery that is introduced in the CHIC security framework (Deliverable 5.3).

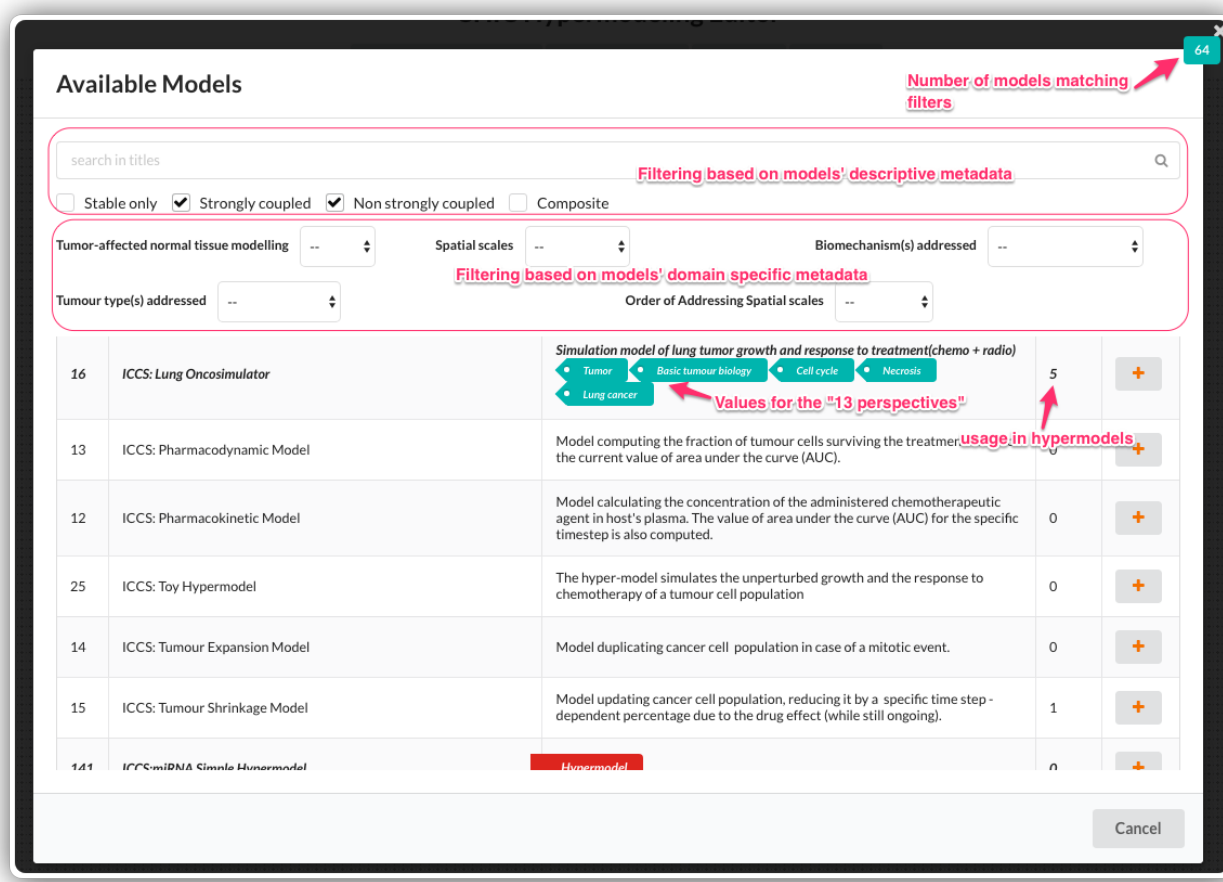


Figure 13 The list of available models as shown by the Editor

This dialog allows the user to filter the models based on the following criteria:

- The name of the model. The user can specify a number of terms or prefixes of words that will be searched in the titles of the models.
- The characteristics of the model, for example whether it's strongly or weakly coupled, if it's in fact hypermodel ("composite"), or stable. The stability of the model is determined by the Model Repository and its support for versioning (See Deliverable 8.4) and it means that the model has been "frozen", i.e. the modeller will not make any further breaking changes in the code of the model.
- The annotation of the models with respect to the "thirteen perspectives" (Deliverable 6.1). In particular, the user can filter the models based on their spatial scales (e.g. molecular, cellular), the order of addressing those (i.e. "bottom-up", "top-down", "middle-out"), the bio-mechanisms they simulate (e.g. angiogenesis), and the specific cancer type (e.g. Wilm's tumor).

The filtering of the models is done in (almost) real time and every time some filter condition changes the Editor updates the table of the matches and the number of matching results in the upper right corner of the dialog (see Figure 13). Each row in the table of the matching models contains

information about the model's title and description, the number of hypermodels that this model has been used in, and the terms that it's annotated with regarding the thirteen perspectives. The user can finally press the "Add" button and the selected model is put into drawing canvas of the hypermodel.

In the main drawing area of the Editor each model is shown as a rectangle ("box") with a number of input and output ports (shown as circles, see Figure 3 in page 12). When the user "double-clicks" the visual representation of a model in the Editor, another dialog shows more information about the corresponding model (Figure 14). In particular, in addition to the name, description, and values of perspectives, this dialog also presents the data types, permitted range of values, the units, the default values, the names, and the descriptions of each input and output of the model. This is information that comes directly from the Model Repository and its entry for the specific model.

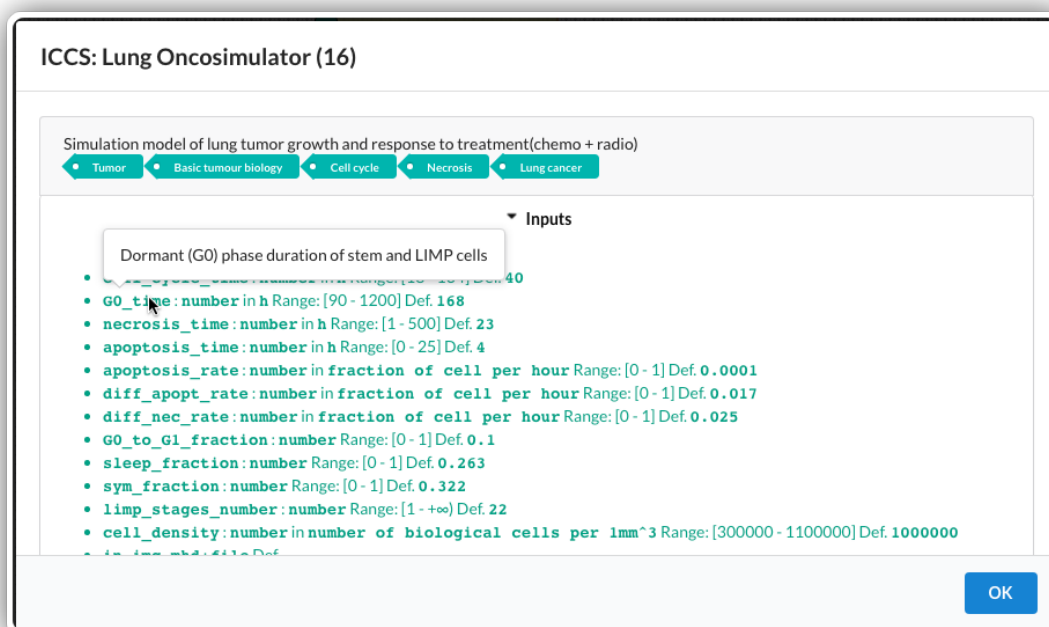


Figure 14 Information retrieved from the CHIC Model Repository for a given hypomodel

When the user adds two or more models in the drawing area, s/he can connect them by “dragging” an output of a source model and “dropping” it into an input of a target model, as Figure 15 (attempts to) show(s).

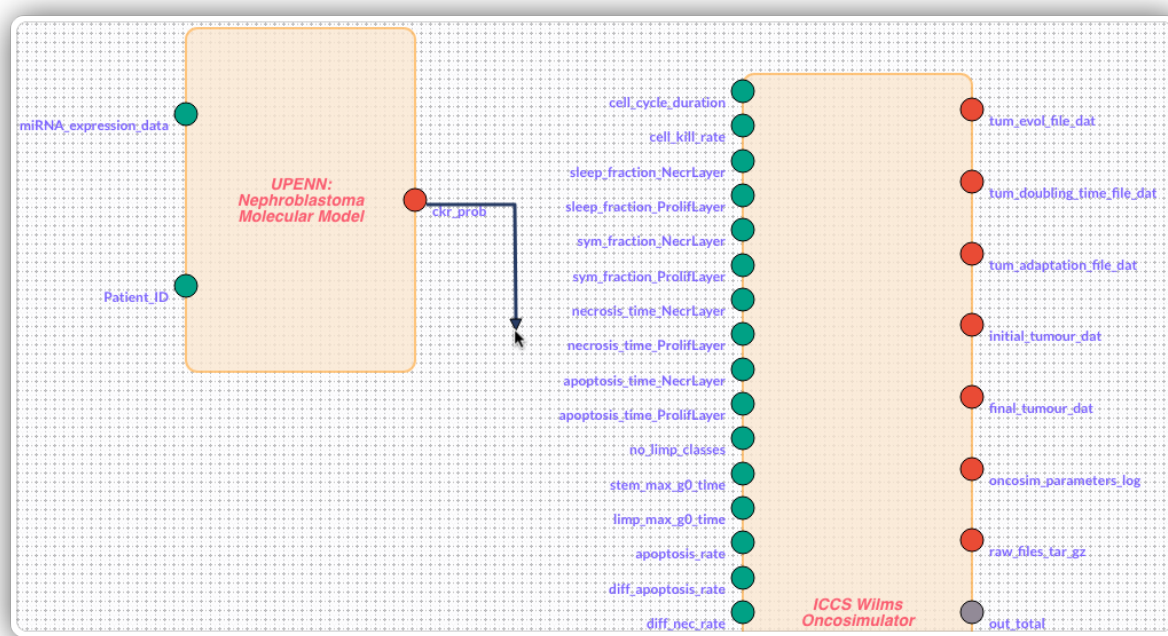


Figure 15 Linking two models together

The editor checks the validity of these connections according to the following rules:

- The source and target port in the connection (i.e. the output of the source model and the input of the target model) should have the data type, for example both should be numbers.
- The source and target port in the connection should be of the same kind with respect to “coupling”. For example, the user cannot connect a “dynamic” output of a model to a normal input of another model, because they have different activation behaviours at the runtime¹³.
- The source and target port in the connection should be of the semantic type, for example “duration of a process”, and use the same Units (e.g. hours).
- In the case of non-strongly coupled hypermodels, no cycle should be created by the connection, either directly or indirectly.

The final view of a nearly complete hypermodel is similar to what is shown in Figure 16.

¹³ A “dynamic” output will potentially transfer multiple values during the execution of its model. On the other hand, a “normal” output will supply a single value after the termination of its model.

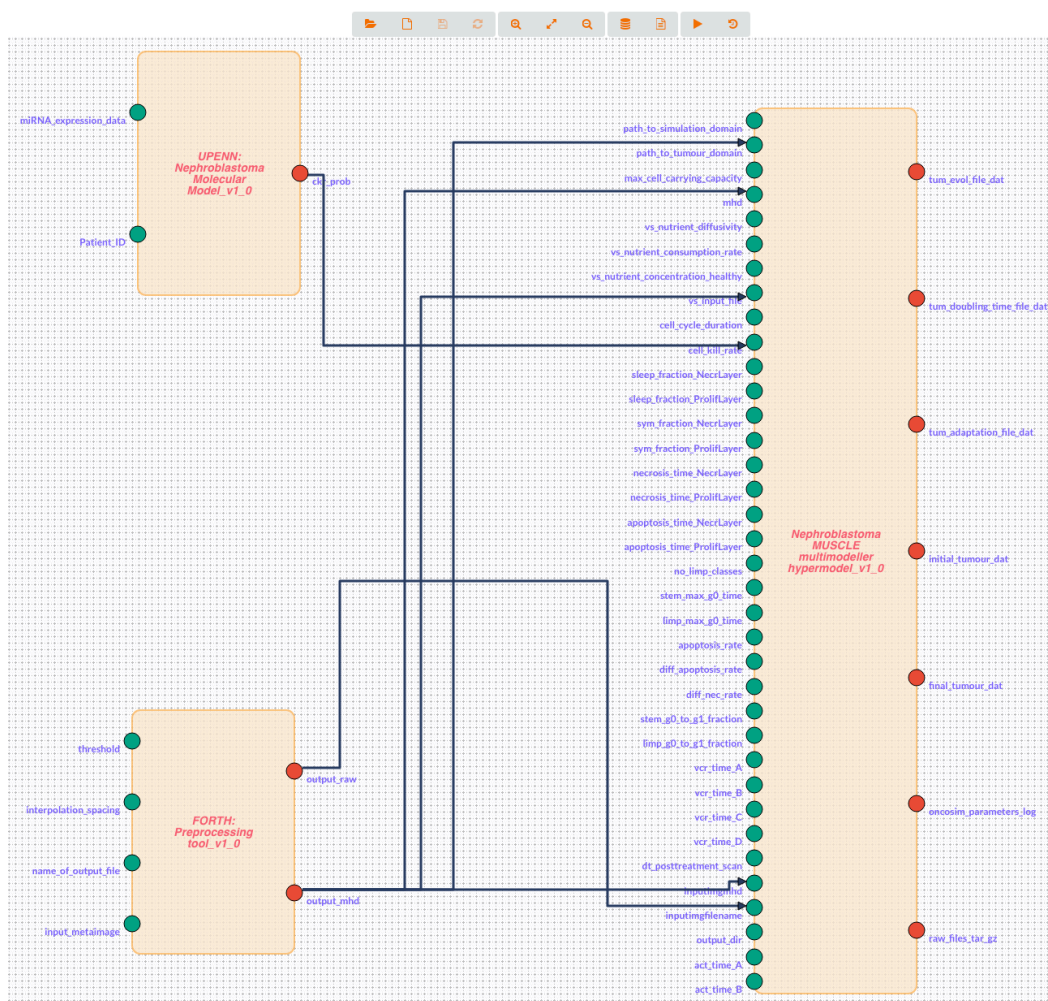


Figure 16 An example of a hypermodel designed in the CHIC Hypermodelling Editor

Anytime the user desires, s/he can save the work by clicking the corresponding button in the toolbar. The Editor then presents a dialog like the one shown in figure below, where the user can supply a name and description for the new hypermodel or update them in the case of an existing one. The Editor shows also the internal id (UUID) of the hypermodel as well as its version. When the hypermodel description is saved in the backend server of the Editor, its version number will be increased.

Nephroblastoma multimodeller hypermodel

ID: ff33edd7-e296-4846-83f2-818b62bfc990 version: 361

Name
Nephroblastoma multimodeller hypermodel

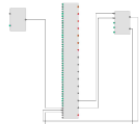
Description
This is the CHIC Wilm's Tumour specific hypermodel

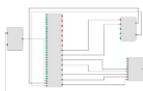
Save Cancel

Figure 17 Information entered by the user during the registration of a new hypermodel

At a later time, the user is able to open one of its previously edited hypermodels (Figure 18). For each hypermodel, the Editor presents its title, the creation and the last modified dates, and the most recent version.

Available Hypermodels

 **Test Lung hypermodel**
 • Tumor • Metabolism • Basic tumour biology • Cell cycle • Necrosis • Biomechanics
 ID: 2fef1e12-71ab-4914-8786-6d10fdaee20
 Created : 15/02/2017 11:16 Updated : 11/04/2017 11:32 (version: 335) Open

 **Nephroblastoma strongly coupled!!**
 • Tumor • Metabolism • Basic tumour biology • Biomechanics • Angiogenesis
 ID: 7941e7b3-5980-4812-b12b-6bcb9cb40c49
 Created : 22/11/2016 10:31 Updated : 03/03/2017 09:22 (version: 317) Repository id: 117 Open

Cancel

Figure 18 Loading an existing hypermodel

5.2 Hypermodel execution

The execution of a hypermodel is another important feature of the Editor. As explained in paragraph 3.4 this is supported through the transformation to the xMML format done by the Editor. An example of this can be viewed in Figure 19. This format is accessible to the user, although normally it's of no use to a typical user.

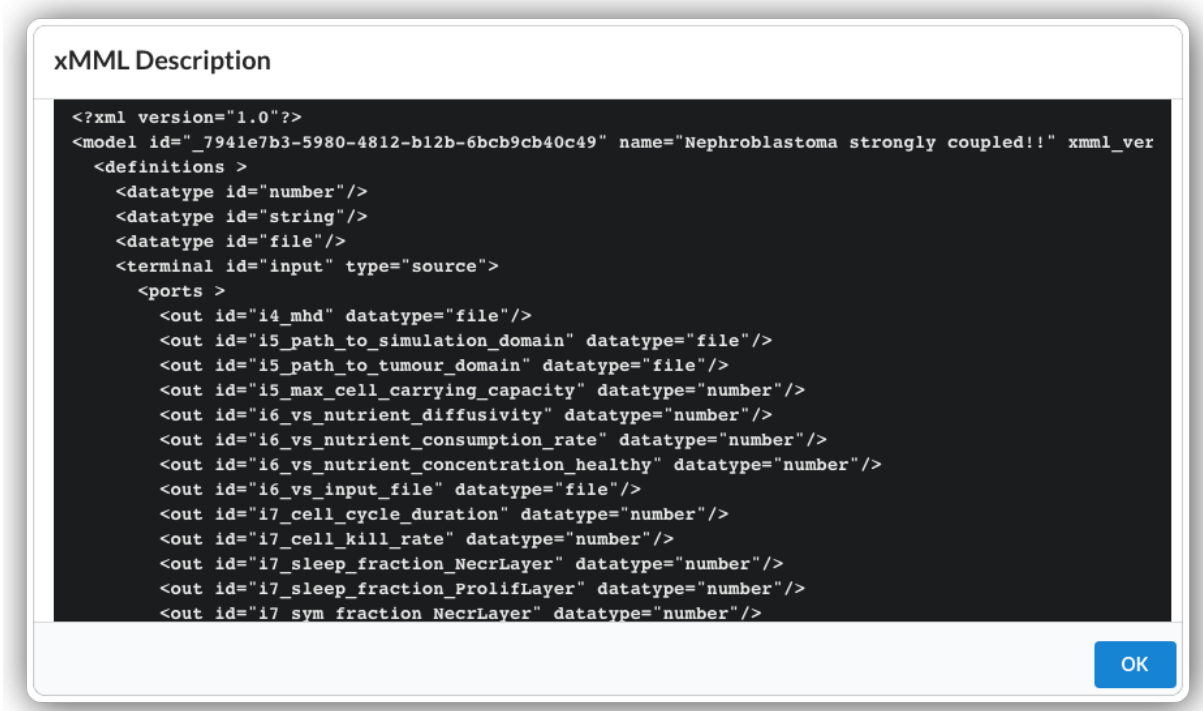


Figure 19 The xMML description of a hypermodel

The actual execution of the hypermodel starts when the user presses the corresponding button in the tool bar. The first thing is then to specify the inputs to be used and yet another dialog appears for that reason (Figure 20). In this inputs dialog, the user can specify the inputs for all the models in the hypermodel that have not been connected with another model. The default values of the models could be used either per model or for all the models in the hypermodel by pressing the appropriate buttons. Finally, the user can also indicate whether a “cached” version of a model should be used, in order to speed up the execution of the whole hypermodel. Please see Deliverable 7.4 for the details on the caching functionality offered per model by the VPH Execution Framework.

After the successful submission of the hypemodel in the CHIC execution framework, the user can logged out and reconnect in the future, without affecting the execution, since the models are run on the CHIC computational platform.

Execution Inputs (only models with non-connected inputs are shown)

▼ ICCS:miRNA statistical model for lung cancer prognosis (1)

in_miRNA_file

https://cdr.chic-vph.eu/api/files/18268/download

outfolder

/

classifier

./ICCS_LungStatisticalModel/config/BaysianClassifier.mat

Fill default values

Clear values

☒ Use caching

Fill All default values

Clear all values

Run!

Cancel

Figure 20 Setting the execution inputs

During the execution of the hypermodel there are visual indications of the execution progress. The Editor keeps an open communication channel with the backend services so whenever an execution finishes, a notification is received, and the user interface of the Editor provides an indication in the tool bar (Figure 21) and a desktop notification is shown in the user's local machine¹⁴ (Figure 22).

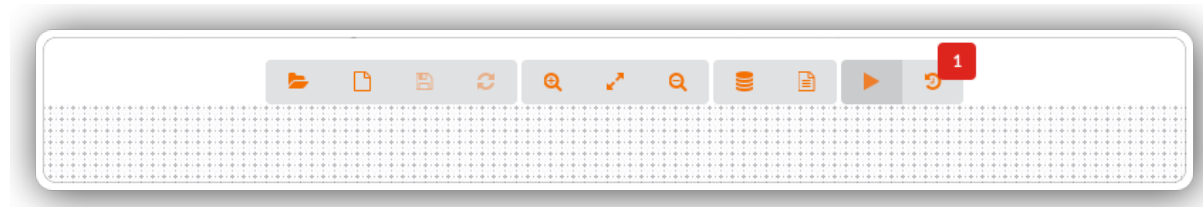


Figure 21 A visual indication about the change of status of an execution

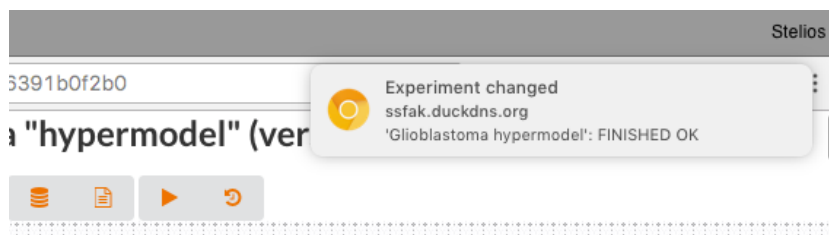








Figure 22 Desktop notification for the completion of an execution

¹⁴ Using the Web Notification API of HTML5: <https://www.w3.org/TR/notifications/>

The Editor cooperates with the VPH Execution Framework, providing the execution identifier for the previous submissions and retrieving the running status of all participating hypomodels. Since there may be multiple hyper models executed at the same time for the same user, the Editor keeps a log with all the previously submitted hypermodels and their execution identifiers. Thus, by clicking the History button, which also shows the number of notifications received, the user is able to retrieve the results of current and past executions that he had submitted (Figure 23). The users are then allowed to download and inspect the results of past executions. Each execution trace is stored centrally, in the backend infrastructure of the Editor, and kept for historical reasons, reproducibility, and provenance.

Experiments

#	Hypermodel	Status	
822	ICCS:miRNA Simple Hypermodel (ver. 357)	Running	
821	ICCS:miRNA Simple Hypermodel (ver. 357)	Finished	
820	Pheno2 (ver. 359)	Failed	
819	ICCS:miRNA Simple Hypermodel (ver. 357)	Finished	
818	ICCS:miRNA Simple Hypermodel (ver. 357)	Finished	
817	ICCS:miRNA Simple Hypermodel (ver. 357)	Finished	
			

OK

Figure 23 The history of past and current experiments

6 Implementation

The Hypermodelling Editor consists of two main components: the frontend is a web application running in the user's browser that retrieves the models' information from the backend server. This two tier architecture allows for greater flexibility in the user interface, while at the same time it offers greater scalability and performance, since most data are stored in the backend server where they are subject to a more heavyweight processing. Due to the different operational contexts, different technologies are used for the implementation of the two components. The browser-based frontend is built using a mix of Javascript, HTML5¹⁵, and Elm¹⁶, a statically typed functional programming language for programming on the Web. The server is built with more conventional technologies, Java as the main programming language and PostgreSQL¹⁷ as the relational database system for the persistence of the managed data.

The two tiers communicate using an HTTP-based RESTful¹⁸ API, while the "Server-sent Events" API¹⁹ is used for the "real time" "push" of new information from the backend server to the frontend user interface. This, in addition to the use of the message broker (see Figure 8) for receiving the execution information of the hypermodels and the "model change" events from the Model Repository, provides a truly responsive and message driven setup in compliance with the principles of the Reactive Manifesto²⁰.

¹⁵ Pilgrim, Mark. *HTML5: Up and Running: Dive into the Future of Web Development*. O'Reilly Media, Inc., 2010. ISBN: 978-0-596-80602-6

¹⁶ <http://elm-lang.org/>

¹⁷ <https://www.postgresql.org/>

¹⁸ Richardson, Leonard, and Sam Ruby. *RESTful web services*. O'Reilly Media, Inc., 2008.

¹⁹ Hickson, Ian. *Server-Sent Events*. 3 February 2015. W3C Recommendation. URL: <https://www.w3.org/TR/eventsource/>

²⁰ <http://www.reactivemanifesto.org/>

7 Conclusions

The CHIC Hypermodelling Editor is a vital component of the CHIC hypermodelling infrastructure since it allows the experimentation and exploration in the construction of new multilevel integrative models. It is tightly coupled and well-integrated with the backend CHIC platform, which includes the Model and Data repositories, the semantics and metadata RDF triplestore, the security services, and the VPH-HF execution framework.

The implementation of the Editor strives to be generic, but of course the level of integration and coupling is too high to be used in isolation. This is an inevitable consequence of the design of the platform so as to be performant but also to eliminate duplication of effort between the technical partners. For example, the *in Silico* Trial repository is an independent component of the system that is used as the intermediary between the Editor and the Execution Framework for the persistence of the inputs and results of the model executions. An alternative approach could have been for the Trial repository to be part of the Execution Framework at the cost of additional effort for the partners developing the execution system. On the other hand, the Editor requires a semantics triplestore to be in place but it is independent of the precise semantic terms used for the validity checks in the connections. So, there are underlying assumptions and component dependencies for the deployment of the Editor but these should be addressed in a case-by-case way if the Editor is to be used outside CHIC.

We are actively supporting this “graduation” of the Editor from the CHIC environment by publishing it as open source software. The code of both the server and the client side is available under the Apache 2.0 open source license and can be found respectively at <https://github.com/cbml-forth/hme-server> and <https://github.com/cbml-forth/hme>. As described in Deliverable 12.5 (“Final Plan for the Use and Dissemination of Foreground”) the whole VPH Hypermodelling Framework is released as open source to provide both a valuable learning resource for the computational modelling community and a starting point for the expansion and the development of more computational tools.

Appendix 1 – Abbreviations and acronyms

<i>API</i>	Application Programming Interface
<i>CDR</i>	Clinical Data Repository
<i>CRAF</i>	Clinical Research Application Framework
<i>DICOM</i>	Digital Imaging and Communications in Medicine
<i>PIMS</i>	Personal Identification Management System
<i>PACS</i>	Picture Archiving and Communication System
<i>RDF</i>	Resource Description Framework
<i>SOA</i>	Service Oriented Architecture
<i>STS</i>	Secure Token Service
<i>URI</i>	Uniform Resource Identifier
<i>UUID</i>	Universally Unique Identifier
<i>VPH</i>	Virtual Physiological Human
<i>VPH-HF</i>	Virtual Physiological Human Hypermodelling Framework
<i>xMML</i>	XML Format of the Multiscale Modelling Language