Article:
Viceconti, M., Bna, S., Tartarini, D. et al. (2018) VPH-HF: A software framework for the execution of complex subject-specific physiology modelling workflows. Journal of Computational Science, 25. pp. 101-114. ISSN: 1877-7503

https://doi.org/10.1016/j.jocs.2018.02.009

VPH-HF: A SOFTWARE FRAMEWORK FOR THE EXECUTION OF COMPLEX SUBJECT-SPECIFIC PHYSIOLOGY MODELLING WORKFLOWS

ABSTRACT

Computational medicine more and more requires complex orchestrations of multiple modelling & simulation codes, written in different programming languages and with different computational requirements, which when validated need to be run many times on large cohorts of patients. The aim of this paper is to present a new open source software, the VPH Hypermodelling Framework (VPH-HF). The VPH-HF overcomes the limitations of most workflow execution environments by supporting both Taverna and Muscle2; the addition of Muscle2 support makes possible the execution of very complex orchestrations that include strongly-coupled models. The overhead that the VPH-HF imposes in exchange for this is small, and tends to be flat regardless of the complexity and the computational cost of the hypermodel being executed. We recommend the use of the VPH-HF to orchestrate any hypermodel with an execution time of 200 seconds or higher, which would confine the VPH-HF overhead to less than 10%. The VPH-HF also provide an automatic caching system over the execution of every hypomodel, which may provide considerable speed-up when the orchestration is run repeatedly over large numbers of patients or within stochastic frameworks, and the input sets are properly binned. The caching system also makes it easy to form large input set / output set databases required to develop reduced-order models, and the framework offers the possibility to dynamically replace single models in the orchestration with reduced-order versions built from cached results, an essential feature when the orchestration of multiple models produces a combinatory explosion of the computational cost.

KEYWORDS

In silico medicine, workflow execution, multiscale modelling.

1. INTRODUCTION

The Virtual Physiological Human (VPH) is a framework of methods and technologies that enables the subject-specific modelling of complex physiological, pathological, and biological processes, across physiological systems and space-time scales [1]. VPH technologies make it possible to estimate quantities that are essential in supporting an accurate medical decision, but that are difficult or impossible to measure directly; for example, predicting the strength of a bone from CT data [2], or predicting the fractional flow reserve of a coronary stenosis form fluoroscopy images [3]. By collecting other, more easily accessible, quantitative information on the subject and combining it with the available mechanistic knowledge available for that specific disease process, VPH models can become useful tools to support the medical decision in a more personalised way [2, 3].

VPH models are *integrative*, in the sense that they can capture and combine knowledge from multiple domains (biophysics, physiology, biology, pathology), multiple organ systems (cardiovascular and respiratory, neuromusculoskeletal, immune, endocrine, etc.), and across

multiple space-time scales (typically from the molecular scale to the whole organism scale) [1].

While in some cases these multiple elements of knowledge are captured into a single, extremely complicated mathematical model, in most cases it is more convenient to capture each element into a separate model, which are then orchestrated into a single simulation. Since there is not a single standardised terminology to refer to these models, hereinafter we will use the terminology used in the CHIC project: as such orchestrations are models of models, we will call them *hypermodels*. Consistently, each composing element will be referred to as *hypomodel*. Hypomodels can be further distinguished in *Component models*, which contain the modelling knowledge, and *Relational models*, which define how specific quantities predicted by one component model transform into the inputs required by another component model (figure 1).

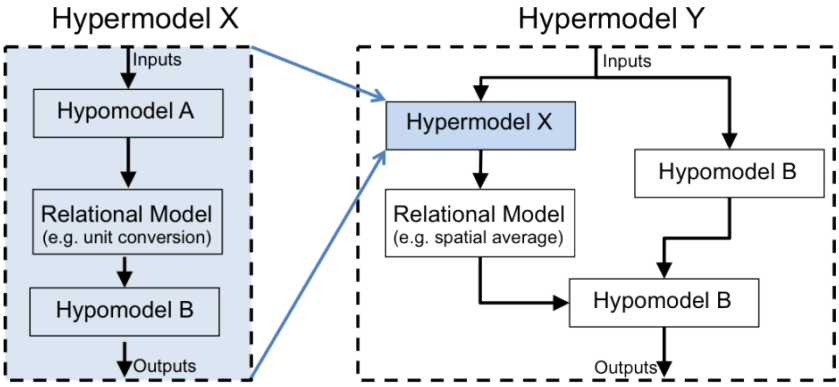

Figure 1. Hypomodels and Relational models can be used to compose more complex models. On the left Hypomodels A and B are connected via a Relational model in the Hypermodel X. The process can be recursive, hypermodel X can be as hypomodel in hypermodel Y.

Table 1 provides a mapping between the terminology used here, and the closest terms used for two other frameworks described below.

Table 1. Terminology mapping between VPH-HF, Taverna, and Muscle2.

VPH-HF	**Muscle2**	**Taverna**
Hypermodel	Model	Workflow
Hypomodel	Sub Model	Nested Workflow
Relational model	Conduit Filter (1:1 relationship), Mapper (N:M relationship) [4]	Shim, Beanshells [5]

Several software tools are available to address this type of problem, including Discovery-Net [6], Kepler [7], VisTrails [8], Anduril [9], GridSpace2 [10], KNIME [11], Galaxy [12], Cuneiform [13], OpenAlea [14], Pegasus [15], and FabSim [16], just to name a few.

A systematic review of all these tools is beyond the scope of this paper; here we focus only on the two that are most relevant here. Taverna [5] and Muscle2 [4] are methodological and computational research frameworks to allow execution and coordination of multiple scientific

software which implement computational models.

Taverna [5] uses a dataflow paradigm, well suited to address the needs of the bioinformatics community [17]. Bioinformatics models rarely require complex patterns of interaction and can be described with Directed Acyclic Graphs (DAG). Taverna also supports the integration of Web Services in the dataflow, since many bioinformatics databases are so large that they can only be queried remotely. More recently, support for conditional and cyclic data flows has been added, but its implementation is optimised for low numbers of iterations (for example, in loops workflows are executed serially, with their full instantiation overhead). Taverna provides an intuitive graphical user interface (Taverna Workbench) that, allowing to build scientific workflows with minimal computational expertise, has been adopted in different domains: bioinformatics [17, 18], biodiversity and ecology [19], astronomy [19, 20], heliophysics [21], statistics [18], and systems biology [22].

Muscle2 addresses the simulation needs of complex biophysics models, such as those used to describe mechano-biology processes. These are frequently modelled with multiscale, strongly coupled models, that pose considerable computational efficiency issues [23-25]. Muscle2 includes a domain-specific language called Multiscale Modelling Language (MML) that allows an elegant formalisation of multiscale problems where several sub-models are coupled [25, 26] . The Muscle2 software suite allows an efficient distributed execution of the models [4, 24, 27]. Muscle2 has been used to model and simulate multiscale models in a range of different domains: Cerebrovascular blood flow [23], irrigation network [28], Reverse engineering of gene regulatory networks [29], turbolence transport [30], in-stent restenosis [31] and biomedical simulations [23] . Recent development on the Muscle2 has added support for an optimised MPI interface in C++ to better exploit parallel HPC systems [27, 32].

The aim of this paper is to present a new open source software, the VPH Hypermodelling Framework (VPH-HF), which was developed in the frame of the EC-funded project "Computational Horizons In Cancer (CHIC): Developing Meta- and Hyper-Multiscale Models and Repositories for In Silico Oncology" (FP7-ICT-600841), hereinafter referred to as the CHIC project. The VPH-HF was developed in the attempt to address the needs of the modellers in the cancer research community: a rapid prototyping platform with reusable multiscale models that could interact using different graph topologies and capable of processing clinically relevant data. Our initial specifications analysis showed that some requirements would have been met by Taverna, some by Muscle2, and some by neither of these. Thus, rather than re-inventing the wheel, we designed VPH-HF on top of Taverna and Muscle2, integrating the two frameworks, and adding functionalities only where necessary. Modelling is an iterative multistage activity, where the model evolves through a gradual process from formulation through verification, validation, uncertainty quantification and ultimately optimisation. The CHIC project computational architecture offers a collection of services that make this iterative process less time-consuming, including a web based graphical editor, a model continuous integration system, a set of repositories for data, metadata and models, and a web-based results dashboard to present the model outputs to the clinical end-users. At the centre of this complex architecture, the VPH-HF ensures the required level of execution services.

2. THE VPH HYPERMODELLING FRAMEWORK

2.1. General specifications and paradigm

The CHIC project computational architecture is schematically represented in Figure 2.

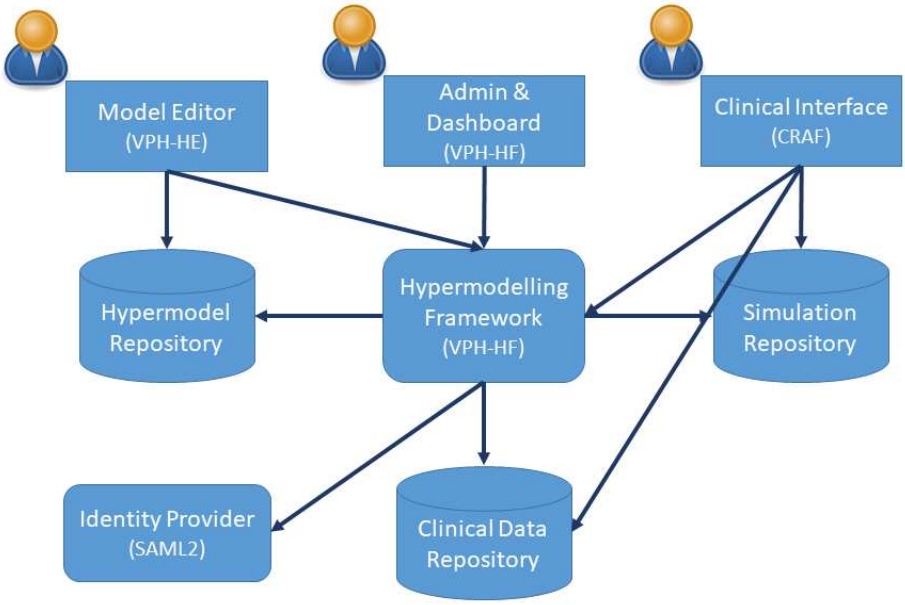

Figure 2. Schematic representation of the CHIC project architecture. Three different type of users interact with VPH-HF: the researcher composing hypomodels into hypermodels using the Model Editor, the system administrator controlling the VPH-HF framework via the Dashboard, the Clinician running hypermodels on patient-specific data to answer specific questions. The arrow symbol represents a link between the caller (arrow start block) and the called (arrow end block) in a web service call.

Users access the system through three interfaces: modellers design their models with a visual editor called the *VPH Hypermodelling Editor* (VPH-HE), and manage their execution directly from the VPH-HE or through the Administration & dashboard interface of the VPH-HF; clinical users request the execution of existing hypermodels on selected clinical data and see the results through a specialised web-based user interface (called *Clinical Research Application Framework*, CRAF) that queries the relevant repositories. Users are authenticated through a proprietary identity provider developed by Custodix[1], which exposes a standard Security Assertion Mark-up Language 2.0 (SAML 2.0) interface; the VPH-HF source code can be easily modified to work with any SAML2-based identity providers. Similarly, the hypermodels, simulations, and clinical data repositories are proprietary implementations of the CHIC consortium, not available under an open source license. The interaction of the VPH-HF with the CHIC repositories is based on standard Web Service interfaces, so the VPH-HF source code can be easily modified to work with any repository implementation that is exposed as a web service.

In this paper, we will focus our attention on the components of this architecture that are available as open source projects, the VPH-HF and the VPH-HE.

The long-term objective of the VPH-HF project is to provide a complete Problem-Solving Environment that supports collaborative development and the re-use of complex computational models for computational physiology and computational medicine [33]. Originally developed

[1] https://www.custodix.com

as part of the EU-funded VPHOP project[2] [34], the VPH-HF was completely re-written during the CHIC project. The three most important characteristics of this new implementation of the VPH-HF are:

1. The VPH-HF framework translates two of the best practices in software development, reusability, and continuous integration, to the modelling domain. Through the VPH-HE a user can design new hypermodels by re-using models already available in the repository and test them in a continuous integration process. Every time a model is updated in the model repository, it is versioned and automatically deployed on a computational host. A test suite is run for verifying the correctness and the outcome is published on a continuous integration dashboard;
2. The VPH-HF framework is designed to execute virtually any workflow pattern, to allow rapid prototyping and reusability of models; both Taverna and Muscle2 workflows can be designed and executed through the VPH-HF, offering the best of the two frameworks;
3. The VPH-HF framework offers two key features to improve the computational efficiency of complex workflows composed by reusable models, caching of executions and a surrogate model replacement service.

The underlying assumption on which the VPH-HF operates is that an orchestration of computational models (hypermodel), given an input set, produces an output set through the orchestrated execution of a heterogeneous collection of hypomodels. This orchestration is described in terms of *data flow* (how data sets are created and exchanged between hypomodels) and of *control flow* (the orchestration logic of the hypomodels). In this paradigm, a hypomodel is seen as a black-box object, with input and output ports that define how it manipulates the data flow, and a control port through which the control flow is delivered to the hypomodel. The VPH-HF allows any simulation software to be made compatible with this paradigm, using a software wrapping approach.

The VPH-HE is a web-based environment for designing scientific workflows [35]. It was originally developed in the context of EU-funded TUMOR project to provide an easy, intuitive, and secure environment for the design of integrative, predictive, computational models represented as scientific workflows. In the context of the CHIC project, VPH-HE was greatly extended to integrate with the VPH-HF and the rest of the CHIC infrastructure and also enriched in terms of functionality and user friendliness. It features a graphical frontend supporting the design of complex integrative models through a familiar "box-and-arrows" user interface. The underlying paradigm is that of "data-flow" enriched with complex validity checks for the connections to facilitate the construction of well-founded multilevel hypermodels.

2.2. Architectural design

The architecture of VPH-HF has been inspired by the concept of modularity: each component can be used in isolation or in ensemble with others to offer more sophisticated functionalities. Each component provides services (or interfaces to services) to other components via communication protocols over a network such as HTTP, HTTPS and AMQP[3] following the Service Oriented Architecture (SOA) pattern. All the software components expose a standard interface (API) that potentially allows them to be used in isolation. The web components of

[2] http://cordis.europa.eu/docs/projects/cnect/5/223865/080/publishing/readmore/2009-2104-VPHOP-VPH-MOY-DEF.pdf

[3] https://www.amqp.org/

the hypermodelling framework are designed according to the representational state transfer (REST) architectural style [36].

Two use cases have driven the architectural design:

- *Modellers*: a modeller installs some hypomodels and wraps them with VPH-HF, publishes them in the model repository, and then uses the VPH-HE to create new hypermodels that orchestrate those hypomodels. They then use the VPH-HF to test their execution and to run them for validation studies, until they can be made available to clinical users.
- *Clinical users*: a clinical user uploads the data of a patient in the clinical data repository, then requests the execution of an existing hypermodel on those clinical data, and accesses the results though an easy-to-use user interface.

VPH-HF uses a client-server architecture to process, server-side, the requests coming from the consumers such as the VPH-HE. The modules that form the VPH-HF, and the logical relationships are shown in Figure 3.

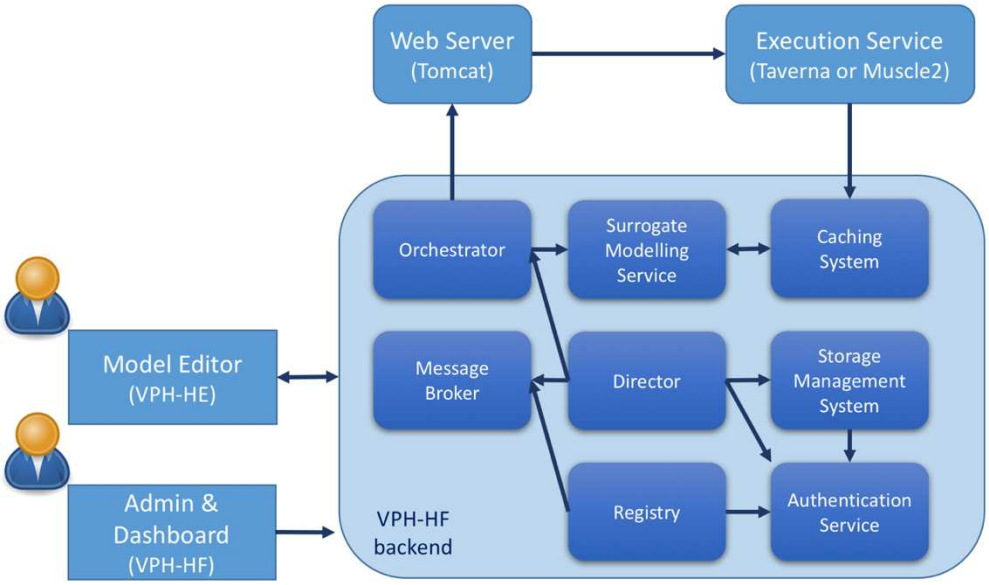

Figure 3: VPH-HF components. The researcher is composing hypomodels into hypermodels using the Model Editor, the system administrator is controlling the VPH-HF framework via the Dashboard. The arrow symbol represents a link between the caller (arrow base block) and the called (arrow head block) in a service call.

Here is a brief description of each module function and its role in the architecture:

1. **User Interfaces**: these are the components which are used by the end-users to interact with the back-end of the VPH-HF architecture to allow the creation, submission, and monitoring of a hypermodel execution. The **VPH-HE** model editor allows the creation and execution of the hypermodels, the monitoring of the execution progress, and the visualization of the output results. The **Admin & Dashboard** web interface was implemented to provide a simple and direct interface to VPH-HF for independent testing; it lets the user execute existing hypermodels on available data, and provides a dashboard to monitor the execution, retrieve the results, and analyse the logs.

2. **Director**: this module coordinates all the other modules of the VPH-HF backend with respect to the execution of a hypermodel and handles the communication to and from the User Interfaces and the Execution Services. It exposes coherent APIs of all VPH-HF hypermodel execution functionalities for the external consumer.

4. **Orchestrator:** this module is the service that orchestrates the hypermodel execution by requesting from the Execution Services the execution of the hypomodels in the correct order as specified by the Director.

5. **Message Broker**: this module is responsible for message validation, transformation, and routing. The message broker is a component that gives the hypermodelling framework a common place where the different components can send and receive messages in an asynchronous way, effectively implementing the decoupling among the web applications. The message broker is used in VPH-HF for:
   - *Push notifications* including the status of the hypomodel execution, status of the hypermodel execution and message errors;
   - *Task submissions* to an asynchronous task queue based on distributed message passing (including submission of a workflow execution, polling and downloading output from the Execution Services).

6. **Storage Management Service (SMS)**: this module provides the VPH-HF with an interface to different storage solutions; specifically, it provides an API for the Director module to retrieve the necessary inputs for the execution and to save the outputs. The SMS module implements two functionalities: a Local File-System Storage Service, and a File Transfer service. The Local File-System Storage Service is a RESTful web service that provides a basic interface to store, retrieve, and delete files in the local file system. This service can be extended in the future to support virtually any storage system. The File Transfer Service is a web service that provides an interface to external repositories and to the Execution Services to perform file transfer operations.

7. **Authentication service**: this service takes care of confirming the identity of the user and provides access to users, whose identity is verified, according to the associated permissions. The Authentication Service was designed to be configurable so that VPH-HF can work with different authentication mechanisms. The default authentication system is a brokered authentication in which an authentication broker (an Identity Provider) is responsible for authenticating the users and issuing time-limited identity tokens to all the services. Such identity tokens can be used by the users to access the project's services.

8. **Registry**: The main role of the Registry service is to register and deploy a hypomodel into the hypermodelling platform in order to be used by the VPH-HE for the hypermodel composition and executed. The Registry uses the model metadata fetched from the Model Repository and the computational infrastructure specifications to automatically wrap a model according to the VPH-HF paradigm. Moreover, it provides, in real time, a list of all the hypomodels/hypermodels accessible within the VPH-HF instance (for example those available for testing but not yet uploaded into the Model Repository), including valid input sets, hypermodel description and (possibly) output sets for each of them.

7. **Caching Service:** Caching is a technique for optimization which, when enabled, stores previous execution input / output data set pairs for selected models. On each request, a combination of (model ID + input values) is checked for a matching cached execution:

if found in the database, the memorized output files are returned to the Execution Services instead of re-executing the identical request. If no matching execution is found, the model is run, and the execution parameters are added to the cache. The immediate benefit of the caching service is that the re-run of models with the same input comes at a negligible computational cost; more importantly, when properly populated, the cache of a computationally expensive model can be used to build reduced-order versions of such models, such as surrogate models.

8. **Surrogate Modelling Service (SUMOS):** Computational implementations of mechanistic models are expected to predict a natural phenomenon with the highest accuracy and the lowest possible computational cost. The trade-off between speed and accuracy is crucial in domains like *in silico* medicine where a simulation is run for each patient (which requires small computing costs), or where the clinical decision that uses the prediction as support needs to be taken in a fixed (short) amount of time. Surrogate modelling (or meta-modelling) is a methodological approach where a high-fidelity model or full-order model is replaced with a simplified lower-fidelity/reduced-order model with a significantly shorter execution time. The SUMOS service provides a set of functionalities for VPH-HF for retrieving surrogate models suitable for replacing a full-order, more computationally expensive models, ranking them by accuracy and execution time over a reference dataset, and substituting them in the execution, with dramatic speed-up, especially for expensive hypomodels that are called recursively.

2.3. Implementation details

2.3.1. Hypomodels wrapper

To be handled by the VPH-HF or recognised by the VPH-HE, any simulation software must be installed in the machine where the Execution Services run (or on any other machine that can be reached with a secure shell (SSH) connection from that machine), and then exposed as a command-line Taverna service, that Taverna indicates as External Tool Service. At that point, the hypomodel is abstracted as any other Taverna service, and can be orchestrated with any other Taverna service.

For hypermodels that require Muscle2 to be executed efficiently, first each hypomodel is installed locally, and enabled to interoperate via Muscle2 (Muscle2 requires message passing, which implies source-code level modifications, in many cases). Once the Muscle2 workflow runs, it can be exposed to Taverna and the VPH-HF using an External Tool Service called *Muscle2 Coupler*. This service wraps the Muscle2 instructions that are extracted from the xMML description of the hypermodel and translated in a Ruby script.

2.3.2. User Interfaces

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 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, HTML5, and Elm[4], a statically typed functional programming language for the Web. The server is built with more conventional technologies, Java as the main programming language and PostgreSQL as the relational database system for the persistence of the managed data. The two tiers communicate using an HTTP-based RESTful API, while the "Server-sent Events" API[5] is used for the "real time" "push" of new information from the backend server to the frontend user interface.

The Admin & Dashboard interface is built using a mix of Javascript (jQuery) and HTML5.

2.3.3. VPH-HF backend

The modules of the VPH-HF backend were implemented in Python. Python has been chosen for its clean syntax, portability, rapid development, and ability to integrate external software modules. Another reason is that Python is the language of Django, the web framework on which the backend of the Hypermodelling engine is built. The Django framework was enriched with the features of the Django REST[6] toolkit to simplify the building of APIs according to the REST architectural style.

The VPH-HF backend modules store their internal data (workflows, files, flags, etc.) into a SQL database using the Django object-relational mapping layer, which supports several relational database products, such as SQlite, MySQL and PostgreSQL.

2.3.4. Orchestrator and Execution services

The VPH-HF is based on the client-server paradigm in which the Orchestrator module coordinate the execution of hypermodels through the Execution Services. The Orchestrator module is implemented in Python 2.7, and invokes a Taverna Server version 2.5.4[7], hosted by an Apache Tomcat web server. When a hypermodel contains Muscle2 execution instructions, a special Taverna module called Muscle2 Coupler is executed, which passes the Muscle2 instructions in the Ruby language to the Muscle2 server for execution.

The orchestrator module requests an execution by sending a hypermodel definition to the Taverna server, written in the Taverna 2 T2Flow format. Once Taverna instantiates the workflows, the Orchestrator sends the appropriate values to the input port of the hypermodel and starts its execution.

For hypermodels defined partially or entirely with Muscle2, the Muscle2 execution is embedded within a Taverna workflow. The Taverna T2Flow includes the Muscle2 Coupler service, which is configured to execute the Muscle2 hypermodel, and copy the results into the Taverna temporary directory for the muscle2 coupler component. Input values are passed to the Muscle2 model in the same way they are passed to any other executable wrapped into a Taverna module.

2.3.5. Message Broker

The message broker module provides content and topic-based message routing using the publish–subscribe pattern. For this function, we chose to use RabbitMQ[8], a popular

[4] http://elm-lang.org/

[5] https://www.w3.org/TR/eventsource/

[6] http://www.django-rest-framework.org/

[7] http://www.taverna.org.uk; the Taverna project has now migrated to Apache: https://taverna.incubator.apache.org

[8] https://www.rabbitmq.com/

implementation of the standard Advanced Message Queueing Protocol (AMQP). AMQP features an efficient and flexible publish/subscribe interface that is independent of the data model. In VPH-HF RabbitMQ is also exposed through Celery[9], an asynchronous task queue/job queue based on distributed message passing. Celery is written in Python and integrates easily into the Django web framework.

2.3.6. Interaction between VPH-HE, VPH-HF, and repositories

The VPH-HE editor and the VPH-HF framework communicate both directly and indirectly by exchanging information through various data repositories: the model repository, which contains the full description of all hypomodels and hypermodels; the clinical data repository, where all patient-specific inputs for the models are stored; and the simulations repository, which contains all input and output sets involved in each execution of each model. The integration of the repositories in the VPH-HF allows for a "separation of concerns" and efficiency in inter-component communication. The sequence of interactions during the hypermodel construction and execution can be represented by the follow steps:

- The modeller users (computational biologists, mathematical modellers, clinical and biomedical researchers) upload their computational models in the Model Repository, accompanied with the executable artefacts and the proper annotation to make them discoverable. When a new model is registered with the Model Repository, a new message is published in the Message Broker that notifies the VPH-HF platform. The new model is then retrieved, and a series of tests are performed to make sure that it's compatible with the execution requirements of the platform. If the tests are successful, the new model is registered in the platform and can then be used in the construction of new hypermodels.
- The VPH-HE Editor retrieves the descriptions of the available hypomodels from the Model Repository and presents them to the user. Based on the descriptions of the models and their annotations, the user builds a new hypermodel by "linking" them (i.e. connecting outputs of a model and the inputs of other models) in a graphical way. New hypermodels are saved to the Models repository alongside the various hypomodels.
- At any time, the user can execute the hypermodel. After some validation checks on the constructed hypermodel, the VPH-HE Editor encodes the graphical representation of the hypermodel in the Multiscale Modelling Language (MML) meta-modelling language [4, 37] using the XML-based representation (xMML). The xMML model description is then saved in the Model Repository as the "executable code" of the new hypermodel along with the pertinent metadata like the creator's identification, date & time of creation, etc.
- For the hypermodel to run, some input data should be also provided. The Editor contacts the Clinical Data Repository to retrieve any patient specific data that can be used as inputs to the hypermodel, subject to the access rights that the specific user has.
- When the patient specific inputs have been selected and any other hypermodel input parameters have been completed by the user, the Editor first prepares a new experiment in the Simulations Repository, and then launches the execution by contacting the Engine and sending it information including the hypermodel identification and the input data to be used. It is important to note that any patient specific data (e.g. medical images) to be used for the execution are sent by reference, i.e. the Editor sends only their URIs that refer to the data objects stored in the Clinical Data Repository. The Clinical Data Repository makes sure that

[9] http://www.celeryproject.org/

all uploaded data are immutable, i.e. they are not altered during any processing. In this way, references to the data can be freely exchanged as they will always refer to the same content.

- When the VPH-HF receives the launch command, it first contacts the Model Repository to retrieve the execution information of the hypermodel referenced in the message. The xMML description is subsequently retrieved and, based on this, the Engine recovers the hypermodel composition and connections required for the execution. The execution environment is then prepared and the hypermodel is run.
- When the hypermodel completes its execution, the Engine updates the experiment record in the Simulations Repository with the result status and the outputs produced during the execution. When the Simulations Repository has been updated, the Engine publishes a new message in the Message Broker to notify the VPH-HE that the hypermodel has concluded its execution. The message contains the experiment identification and all needed correlation information that links to this hypermodel.
- The VPH-HE Editor receives the "execution finished" message and based on the experiment identification number, it retrieves the execution status and the final results from the Simulations Repository. It then allows the user to download the complete set of outputs and execution log.

2.3.7. Authentication service

The default authentication protocol for exchanging authentication and authorization data between security domains in the VPH-HF is the SAMLv2[10] protocol. SAML is an XML-based protocol that uses security tokens to exchange identity assertions of an end-user between an Identity Provider and a Service Provider. SAMLv2 can also support web Single Sign On (SSO) authentication and authorization scenarios.

Caching Service The VPH-HF Caching Service optimises performance by avoiding redundant hypomodel executions on an already processed input dataset. A memorization function maps the input dataset to the corresponding output dataset for each model, using a database. For each model execution request, the set of input files are hashed to create a 'fingerprint' that is used as a lookup in the database for the corresponding output datasets produced in previous model executions. When a model has been executed previously, a database entry is found and the location of a local copy of the output dataset is provided as result, skipping the model execution. Like in the processor's cache, resources are limited, with storage being a constraint. A data storage policy should be considered especially in scenarios where datasets are relatively large.

The cache can be enabled or disabled on a per-model basis depending on need. For instance, the user can disable caching for models that work with sensitive personal data for which appropriate security measures to protect them are not in place. Moreover, the following situations are undesirable for caching:

- Low hit rates (input sets are rarely repeated), and no plan to build a surrogate for that model;
- Non-unique relation between inputs and outputs (e.g. a model using an internal random number generator);
- Restricted access to input or output files.

Figure 4 highlights two additional components required for caching; a database for storage and

[10] http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html

Cache Agent to manage the flow control and caching logic.

The outer boxes represent independent virtual machines, the 'Workflow Orchestration' is a deployed instance of the VPH-HF while the Execution Machine is a remote node where the Execution Services are running the models. By inserting the Cache Agent between the Model Wrapper and the Taverna temporary directory an execution can be replaced with a database load when the cache detects a hit.

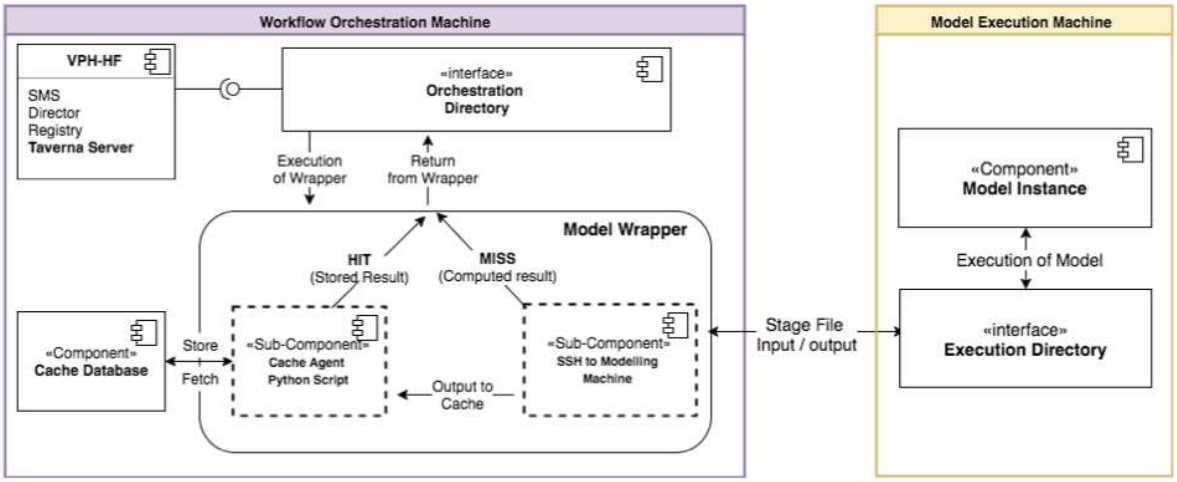

Figure 4. UML diagram of the interaction between the two components forming the caching service module.

This is possible because each input/output is represented as a file.

The control flow to handle a cache miss is:

1. Taverna Server first populates a temporary working directory with input files;
2. Then the Cache Agent hashes all input files to create a fingerprint;
3. The hash or fingerprint is checked against the Cache Database and returns a MISS;
4. The Model Wrapper is executed:
   - 4.1. Input files are staged to the remote machine;
   - 1.1. A model is executed;
   - 1.2. The generated output files are staged back to the Taverna temporary directory;
2. The Cache Agent stores the output files in the Cache Database using the input hash as a reference;
3. The workflow completes, and control is handed back to VPH-HF.

The control flow to handle a cache hit is:

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

The workflow completes, and control is handed back to VPH-HF.

2.3.8. Surrogate Modelling Service

The surrogate modelling service (SUMOS) is a software component providing services to be

consumed by applications like VPH-HF and the VPH-HE Editor (Figure 5). It is implemented mainly as Python modules exposing an API to retrieve a list of available surrogate models ranked by estimated accuracy and execution time. The default policy for the hypermodelling execution framework is to use the surrogate model available with the highest accuracy and shorter execution time below a required value. Surrogate models validated by the modeller for their use are published in the Model Repository.

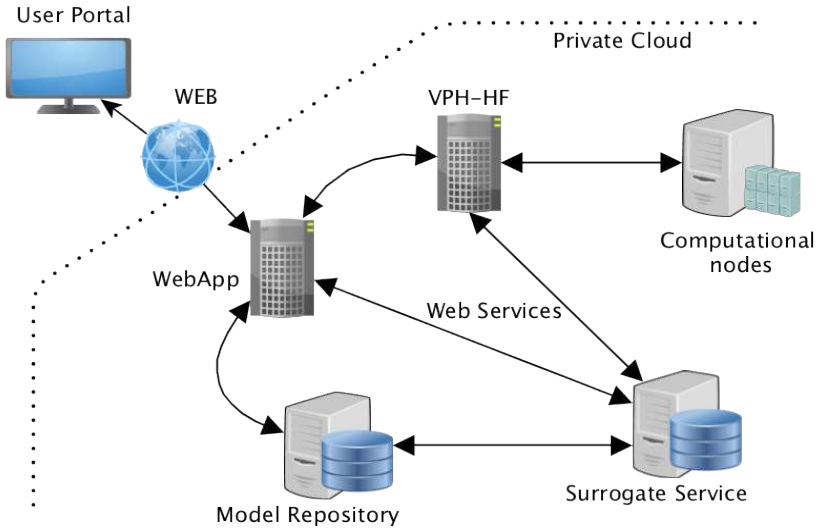

Figure 5. Architecture of the Surrogate Modelling Service. The diagram shows how the services are deployed on physical nodes and how they interact via web services.

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

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

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

2.3.9. Source code availability

The source code of the VPH-HF framework and of the VPH-HE editor are released as open-

source software under the Apache License and hosted on Github[11], from where it can be freely downloaded.

All the components (except for the Repositories that are proprietary software) and libraries which VPH-HF depends on or is connected to are open-source software. To name a few, the Execution services (Taverna v2 and Muscle2) are released in previous projects (myGrid, Mapper) under the LGPL license (v2.1 and v3 respectively); RabbitMQ under the Mozilla Public License v1.1 and Django under the BSD License. A complete list of interdependencies and relative licenses is available in the GitHub VPH-HF repository.

3. METHODS

3.1. An exemplary hypermodel: nephroblastoma growth model

The CHIC architecture supports the creation of robust, reproducible, collaborative models (hypermodels) of disease by composition of reusable, interoperable component models (hypomodels). The cancer domain was used as a paradigmatic example of the hypermodelling of a complex disease. Figure 6 represents the entire nephroblastoma growth hypermodel developed by the CHIC consortium. It consists of one technical hypomodel that performs common pre-processing tasks, and five biologically driven hypomodels that represent distinct aspects of tumour evolution and interaction with the tumour environment.

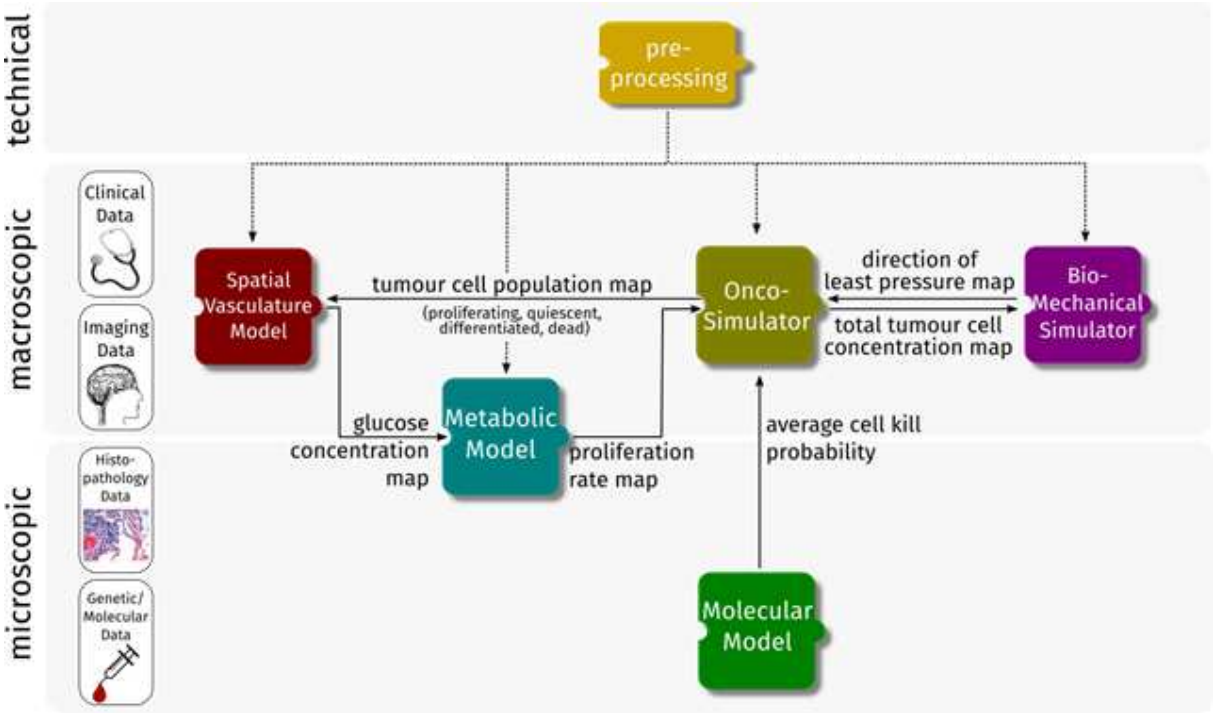

Figure 6. Multiscale model of the growth of a nephroblastoma.

The *Onco-Simulator* [38] is a spatially discrete hypomodel that simulates cancer cell proliferation and treatment effect in function of tumour, treatment, and patient-specific parameters. Environmental parameters influencing cancer growth are represented by four

[11] https://github.com/INSIGNEO/VPH-HF

further hypomodels.

Based on the current cell population in each time step, the *Vasculature Model* solves a reaction diffusion partial differential equation for glucose concentration in the growth domain, based loosely on a spatially resolved version of [39]. This information is used by the *Metabolic Model*[12] to compute the local proliferation of tumour cells within each discrete element of the domain based on encoded knowledge about glucose-dependent biological processes. For example, reduced availability of glucose will result in an increase of the fraction of dormant cells. This spatial map of proliferation rates in turn informs the growth dynamics of the Onco-Simulator.

Similarly, cellular sensitivity or resistance to treatment is a determinant of cancer evolution and treatment outcome. The cell kill probability for a tumour cell is explicitly computed by the *Molecular Model* [40] based on the molecular profile of the patient.

Mechanical forces caused by tumour expansion constitute a further environmental factor that affects tumour development. The *Biomechanical Simulator* [41] uses the finite element method (FEM) to compute the mechanical effect of the growing tumour in a 3D model of healthy and cancerous tissues. The resulting pressure field informs tumour growth direction.

The complete nephroblastoma hypermodel is represented figure 7 in the VPH-HE.

It is represented in VPH-HF as a *hybrid* hypermodel: it combines strongly-coupled hypomodels with weakly-coupled ones to form a hypermodel. The strongly-coupled part, outlined in a red box in Figure 7, comprises the Onco-Simulator, the Vasculature model, the Metabolic model, and the Biomechanical Simulator.

The full Nephroblastoma model is available in the model repository as model #108. We used it to profile the VPH-HF. To evaluate the VPH-HF "overhead" on smaller models, we also profile a few other models available in the repository, which had much smaller execution time (models #93, #94, and #109).

[12] The metabolic hypomodel is currently under publication: it utilizes the generic genome-scale human metabolic network reconstructions and describes the metabolic activity of the chemical reactions at flux level using the Flux Balance Analysis constraint-based method. The model assumes that cancer cells are under a selective pressure to increase their proliferation rate. At each time interval, the glucose concentration is estimated at every position in the computational grid through the vasculature hypomodel. The metabolic model provides information regarding the proliferation rate given the available glucose that is then used from the Onco-Simulator to update its state.

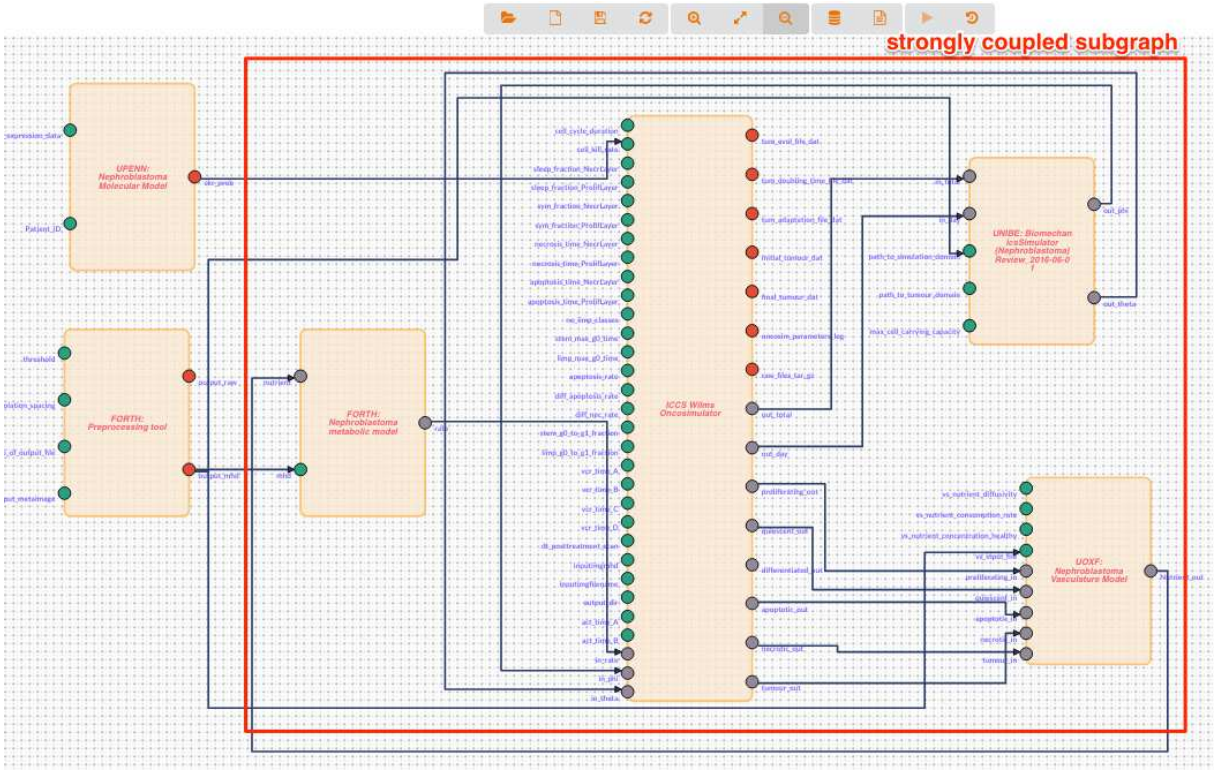

Figure 7. Nephroblastoma growth model representation in the Hypermodelling Editor.

3.1.1. An example of surrogate modelling integration: Oxford growth model

Surrogate modelling integration is demonstrated using a mathematical model of tumour spheroid growth. A previously described computational framework (Chaste) is adopted to model the growth of a two-dimensional tumour spheroid in an oxygen rich environment [42, 43]. Individual cells are explicitly modelled, with progression through the cell cycle governed by the local oxygen tension. Cell-cell mechanical interactions are modelled using linear springs, with neighbours identified using a Delaunay triangulation on the cell centres [44]. Oxygen tension is prescribed on the growing outer boundary of the spheroid and decreases toward the centre due to consumption by cells. The reaction-diffusion equations governing oxygen transport are solved using the finite element method, with temporal re-meshing of the growing domain [43]. A flow-chart for the single hypermodel is shown in [43]. The spheroid initially undergoes rapid growth, at a rate dependent on the rate of oxygen consumption by cells, the oxygen dependent duration of the cell cycle, and the prescribed oxygen tension on the boundary. Growth is limited by oxygen availability, gradually slowing as available oxygen is restricted to a thin annulus on the periphery [45]. While in the test formulation the run-time never exceeds 140 minutes to simulate a week of tumour growth, the clinical application of this model would require its generalisation to 3D, and a time scale for simulated growth of months, rather than weeks, making some order-reduction strategy indispensable.

The formulation of a tailored surrogate model for the Oxford growth mathematical model requires careful analysis and validation, with few examples in the literature for similar applications to date. Since the focus of the present study is mostly on infrastructure, a generic surrogate model is adopted, while research into tailored surrogate models for tumour growth is on-going. A mathematical model (or reference model) can be approximated with a surrogate using a number of different techniques, reviewed in [46-48].

Here we focus on response surface methods, which are overviewed in [49] and in [50]. The adopted method identifies a linear combination of polynomials that minimise the error between surrogate and reference models using the least squares method. A set of samples is generated on a Cartesian rectangular grid in the parameter space of the reference model. Samples and the outputs produced by the reference model are collected in a training set. The Oxford growth model predicts the diameter of the tumour spheroid over time given the initial number of cells, their cell cycle G1 phase duration and the oxygen consumption rate. For the presented example, the cell cycle G1 duration is varied, while the oxygen consumption rate (a rate constant in a first order sink in the oxygen transport reaction diffusion partial differential equation) is fixed at 780 per hour.

Samples are taken in the 2-dimensional space obtained from the Cartesian product of the cell cycle G1 duration parameter and simulated growth time vectors, using 20 evenly distributed values in the interval (12, 40) hours for the former and (0, 178) hours in the latter, divided into 891 steps. Adopted parameter values and ranges are chosen to give qualitatively and quantitatively similar growth behaviours to those observed experimentally in [51]. The least squares method is used to identify the regression coefficients $b_i$that best fit the training set, as described in [49, 51]. We used a fourth order polynomial, including mixed terms from the second order to the fourth order:

$$y = f_4(x_1,x_2) = b_0 + b_1x_1 + b_2x_2 + b_3x_1^2 + b_4x_2^2 + b_5x_1x_2 + b_6x_1^3 + b_7x_1x_2^2 + b_8x_1^2x_2 + b_9x_2^3 + b_{10}x_1^4 + b_{11}x_1^2x_2^2 + b_{12}x_1^3x_2 + b_{13}x_1x_2^3 + b_{14}x_2^4 \quad \text{(Eq. 1)}$$

where $x_1$is the duration of the cell cycle G1, and x_2 is the growth time.

3.2. Execution environment

The two models were built with the VPH-HE and executed through the VPH-HF framework, using Execution Services running on the CHIC private cloud. The private cloud is based on the OpenStack platform[13] and it's built using (in total) 64 physical cores (128 threads w/ hyper-threading technology) of high-end processing power, 1 TB RAM, 17.7 TB storage space over RAID60 configuration (fault tolerance + speed efficiency), and 1Gbit subnet with a pool of 128 floating IPs (individual bridged access to public network). With this configuration, the CHIC private cloud can easily support more than 100 virtual machines (VMs) with the typical configuration of a middle-end server (2-4 vCPUs, 8 GB RAM, 150 GB disk space). The largest of these virtual machines is dedicated to model execution and it consists of 16 vCPU, 128 GB RAM, and almost 1TB disk volume.

The profiling of the surrogate modelling service on the tumour spheroid growth were obtained on a Linux Redhat 7.2 workstation with two Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz and 256GB of RAM.

[13] https://www.openstack.org/

3.3. Profiling

The average time was obtained tracking the progress of multiple executions by noting the time at specific points. Intervals are calculated from these 'time stamps' to show a breakdown of the total execution in proportion to three sub tasks. In particular (Figure 8):

- Client Communication, is the average time needed submitting a new execution request to the VPH-HF API, such as the uploading of input files and waiting for the new job to show a status of INITIALIZED.
- Model Execution is measured by running a RabbitMQ listener in a separate thread during each test run. The time of each message is recorded, giving us the interval between the first hypomodel start time and last hypomodel completion time.
- VPH-HF Processing, is the time taken for staging inputs and processing outputs. Its calculated as 'VPH-HF execution time' - 'Hypomodel(s) execution time'.

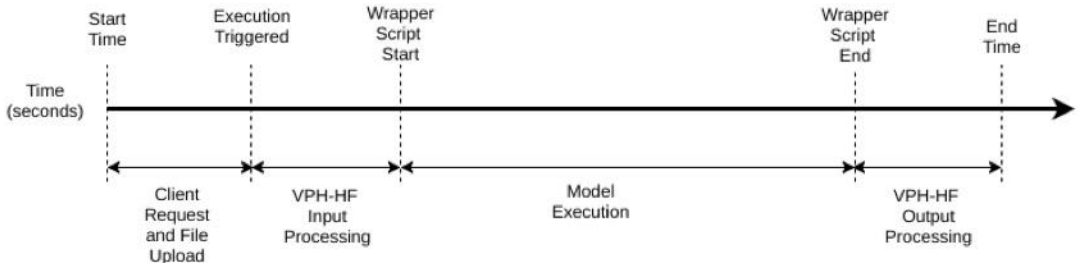

Figure 8. Timeline of a workflow execution in VPH-HF.

4. RESULTS

4.1. Construction of the hypermodels

The exemplary models were installed and configured in the CHIC model repository without any difficulty. The Oxford Growth model is a simple hypomodel, that executes within Taverna in a single step. After the installation of Chaste, it required only 20 minutes to expose the hypomodel in the VPH-HF framework, and run the model checking script, which gave no errors.

The development of the Nephroblastoma growth model required considerably more work due to its complexity. The strongly-coupled part, outlined in a red box in Figure 7, was first built graphically in the VPH-HE and then registered in VPH-HF. The strongly-coupled hypomodels had to be integrated using Muscle2, and the resulting hypermodel was wrapped by the Registry into a Muscle2 Coupler Taverna service. The entire hypermodel was finally built in the VPH-HE by linking the strongly coupled hypermodel with the technical and the molecular hypomodels. Because of the stochastic nature of the model, the execution time depends on the input values, and ranges between 10 and 20 minutes (wall-clock time) to execute on the CHIC private Cloud.

4.2. Execution profiling

Figure 9 shows the non-cached execution time of the nephroblastoma hypermodel (model #108) and its constituent hypomodels we profiled, average over 30 runs with different inputs values.

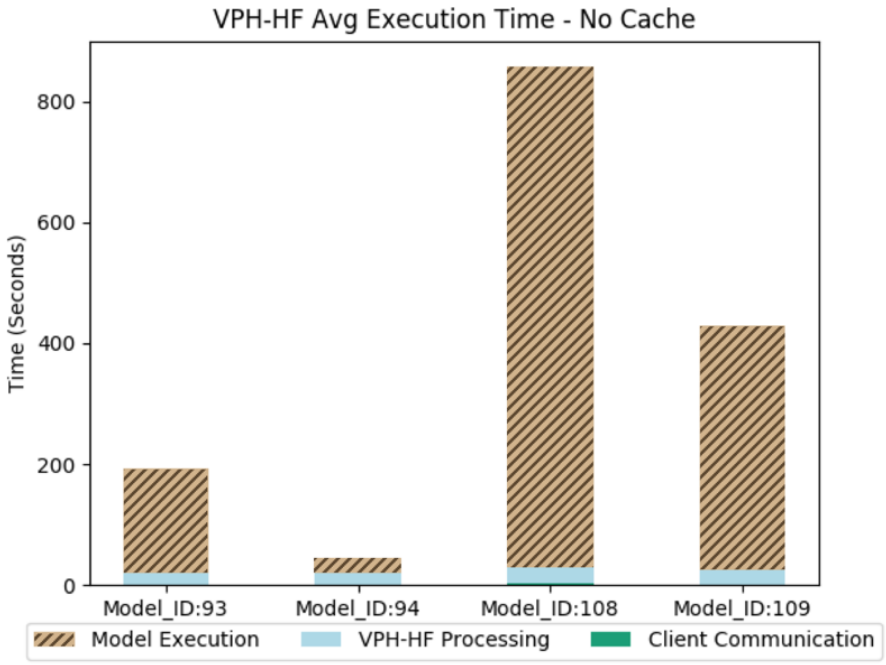

Figure 9. Average non-cached execution time of the nephroblastoma growth hypermodel and its hypomodels over 30 runs with different inputs. The total execution time is divided in separated in communication, VPH-HF latency, and hypermodel execution.

When run with an input set never run before (i.e., no cache), the models we profiled executed in between 20 and 1150 seconds. When run within a 100 points Monte Carlo the nephroblastoma hypermodel would take around 23 hours to execute. The VPH-HF introduces an overhead of only 20-25 seconds per execution, which remains quite constant regardless of the complexity or the computational cost of the model.

4.3. Caching profiling

The idea of adopting the caching service is to avoid running previous identical executions at the expense of an increase of storage space. If the time required by the caching service to pull the output data from the database to the execution sandbox is negligible with respect to the average execution time, the positive impact of caching is considerable.

When all the models being profiled were run again with the same input sets, the VPH-HF automatically retrieved the output sets from the cache. In this case, the average execution time was 20-30 seconds, and most of this time was used by the VPH-HF, with the client-server communication overhead limited to around two seconds, and the model execution (which now is limited to the cache retrieval time) well below one second (Figure 10). This aspect is highlighted in Table 2: without caching, most of the time is spent in the model execution ($T_e/T_{tot} \simeq 1$), with caching the most time-consuming part is shifted to the post-processing phase ($T_p/T_{tot} \simeq 1$).

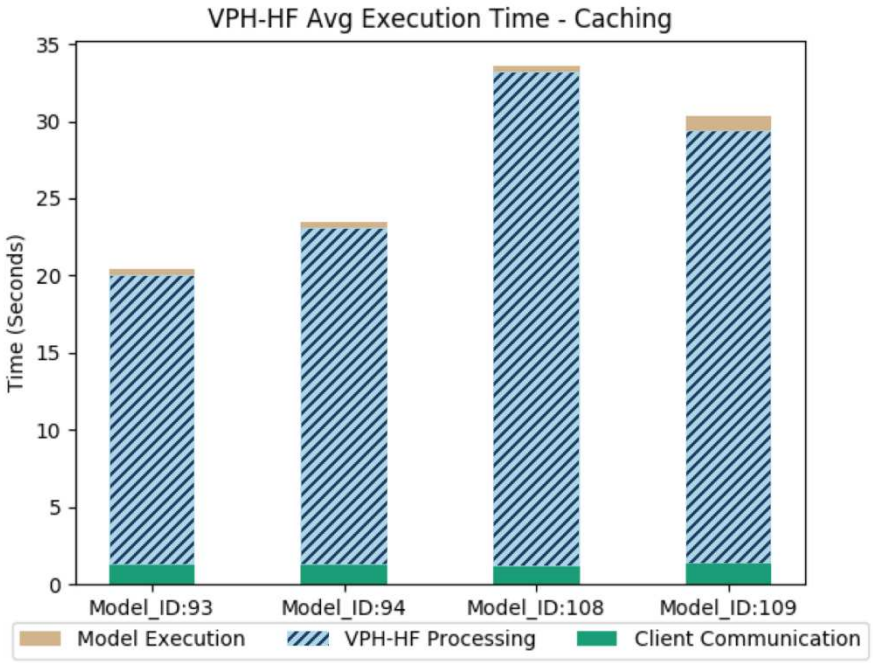

Figure 10. Average cached execution time of the nephroblastoma growth hypermodel and its hypomodels over 6 runs with different inputs, separated in communication, VPH-HF latency, and hypermodel execution.

Of course, for hypermodels with a shorter execution time, multiple inputs, and larger file sizes, the adoption of a cache might not be convenient anymore. Figure 11 plots the execution time averaged over 5 consecutive runs of a generic hypermodel for both cache hit and cache miss cases, as the number of input and the file size is increased. Every input of file size N is made by N elements where one third are random integer values, one third are random floating-point values and one third are 32bits chunks of random data. The plots show that the execution time increases almost linearly with the file size and the slope increases with the number of inputs. Considering the worst case (InPorts = 45, N = 1000000) the execution time for a Cache Hit is around six seconds and the execution time overhead for a cache miss, 12 seconds.

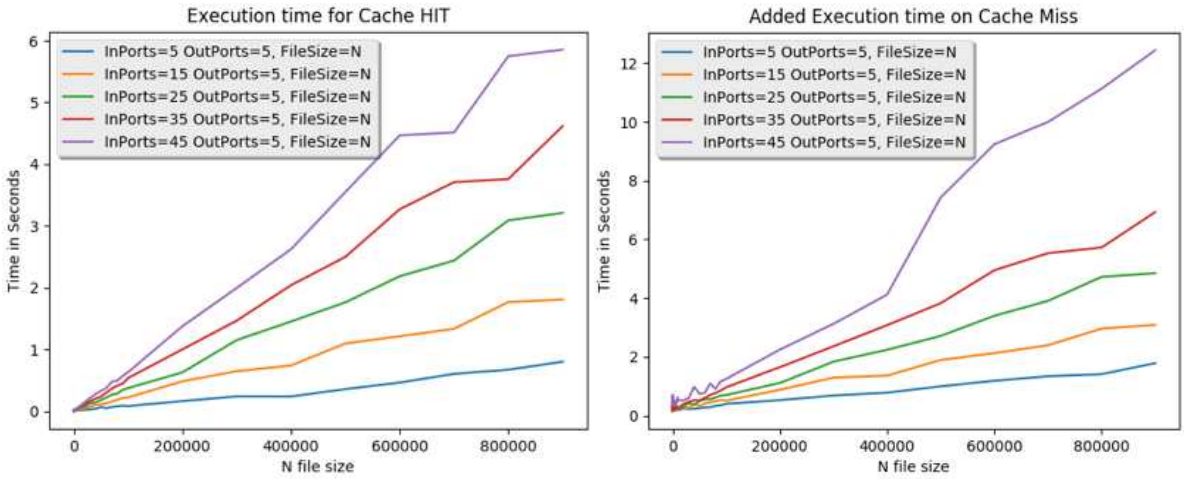

Figure 11. Profiling of the caching service: execution time (in seconds) overhead in case of cache miss (left), and total execution time in case of cache hit (right).

Table 2. Profiling of the caching service: speedup (T_tot_no_caching / T_tot_caching) and relative time consumption of each of the three tasks with/without caching: model execution (T_e/T_tot), VPH-HF processing (T_p/T_tot) and Client communication (T_c/T_tot).

	No-caching			**Caching**			**Speed-up**
Model_ID	*T_e/T_tot*	*T_p/T_tot*	*T_c/T_tot*	*T_e/T_tot*	*T_p/T_tot*	*T_c/T_tot*	
93	0.897	0.096	0.007	0.017	0.921	0.062	9.47
94	0.549	0.417	0.034	0.016	0.930	0.054	1.90
108	0.966	0.031	0.004	0.012	0.954	0.034	25.55
109	0.944	0.052	0.004	0.033	0.924	0.043	14.16

4.4. Impact of surrogate modelling

The Oxford agent-based tumour spheroid growth model, shown in Figure 12(a), is relatively computationally expensive. As shown in Figure 12(b), run-time scale linearly with number of cells, which can in turn increase exponentially as the growth simulation progress. It provides a good exemplary case for the evaluation of a surrogate modelling framework, as the tumour growth dynamics for this situation are well known; in particular, the tumour volume growth can be described as a general logistic function of time [45] (Figure 12(c)).

A 4^{th} order polynomial root-square-mean approximant was found to be sufficiently accurate (Adjusted-R^2 = 0.9934; root mean square error = 20 μm). Reference and surrogate model predictions are shown for 4th order polynomial fits in Figures 12(c). Samples (training data from the reference model) are plotted with black dots, while the response surfaces are plotted as a coloured surface.

Approximating of the Oxford growth model using the 4th order polynomial surrogate model reduces the computational complexity from a linear/exponential to a constant, as shown in Figure 12(d). The execution time for the surrogate models is 0.7s and is mostly due to its implementation as a configurable model reading its configuration from files. The prediction time is 0.015s, with the rest of the time spent accessing the configuration files and running the Python interpreter.

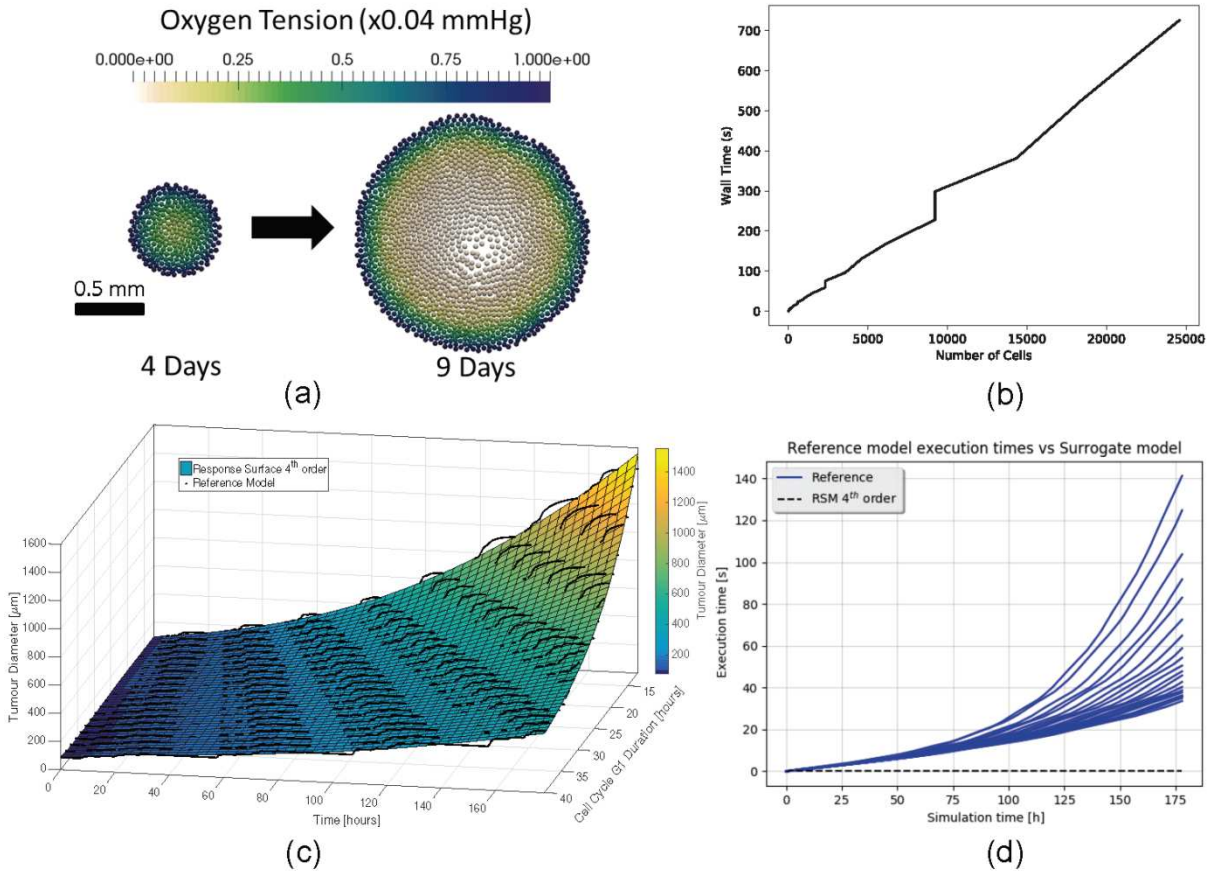

Figure 12. (a) Simulated tumour spheroid growth in an environment with fixed oxygen tension of 25 mmHg (oxygen consumption rate = 666.0 per hour, cell cycle time = 16 hours). (b) The execution or 'wall' time increases linearly with cell number in the studied cases. Discontinuities are due to synchronous division of sub-populations of cells, which is expected in models of this type. (c) Response surface and reference model predictions for the fourth order polynomial with interacting terms. The black dots are the reference model sample values, while the RSM is the coloured surface. Predicted tumour diameters over simulation time as cell cycle time is increased in the range 12-40 hours are shown. (d) The execution time for the reference model and surrogate models compared to the simulation time. The execution time for the surrogate model is approximately constant at 0.7s.

5. DISCUSSION

The aim of this paper was to present a new open source software, the VPH Hypermodelling Framework (VPH-HF), which allows users to compose complex hypermodels (orchestrations of models) from a heterogeneous collection of hypomodels developed with different

programming languages and/or modelling software, and execute them efficiently in a production environment, where each hypermodel is expected to be executed many times with different input sets, each typically presenting a single patient, or a single control of a single patient.

The VPH-HF addressed most specifications emerged in the support of computational oncology applications. The integration of Taverna provided all essential workflow execution services, and the possibility to reuse all components and workflows made available by the large Taverna community. The integration of Muscle2 provided the support for the execution of very complex, strongly coupled, hypermodels. The overhead that the VPH-HF imposes is small and tends to be flat regardless of the complexity and the computational cost of the hypermodel being executed. We recommend the use of the VPH-HF to orchestrate any hypermodel with an execution time of 200 seconds or higher, which would confine the VPH-HF overhead to less than 10%.

An important factor in the adoption of a new technology is the steepness of the learning curve. While complex models are designed, it is easier in many cases formulate the in term of simple orchestrations of multiple hypomodels; this approach offers a more rapid prototyping, where single hypomodels can be replaced with others with moderate effort. Once the hypermodel architecture is defined, the introduction of more complex execution flows, and the possibly to strongly couple two or more hypomodels, can make the hypermodel more formally rigorous, and in some cases even more efficient. By incorporating the best of both worlds, VPH-HF enable the rapid-prototyping typical of Taverna with the capability of building complex, strongly coupled workflows typical of Muscle2.

The introduction of an automatic caching system over the execution of every hypomodel can provide considerable speed-up when the hypermodel is run repeatedly over large numbers of patients, and the input sets are properly binned. The caching system also makes it easy to form large input set / output set databases required to develop surrogate or reduced-order models, which can gradually replace the most computationally expensive hypomodels. These technologies can render subject-specific modelling technologies cost-effective and time-effective, two frequently stringent requirements for the clinical adoption of these technologies.

Also, the possibility to dynamically replace hypomodels with surrogate versions built from cached results might be the only viable option when we need to build strongly-coupled hypermodels affected by combinatory explosion. A typical example is an hypermodel that couples a stiff Ordinary Differential Equation (ODE) model characterised by a very large number of computationally cheap iterations, with a Partial Differential Equation (PDE) model characterised by a smaller number of very expensive iterations. When two such hypomodels are coupled, the combined computational cost can become intractable. With the VPH-HF technology it is straight-forward to perform a parameter sweep of the PDE model, create a large cache of results, and use it to build a surrogate model, that when coupled to the ODE keeps the computational cost of the ODE/PDE hypermodel comparable to that of the ODE hypomodel alone. The test on the Oxford tumour model confirmed that where a satisfactory surrogate model can be formulated, the acceleration of simulations can be substantial.

Another important feature of the VPH-HF is its integration with model and clinical trial repositories, making an ideal technology to deploy production environments for subject-specific modelling. As this field is emerging from its pioneering days, the need for production environments, such as those developed by the CHIC project, will become more and more important.

This study presents several limitations. The most important is that some parts of the CHIC framework will not be released under an Open Source license. With reference to Figure 2, the CRAF clinical interface is probably too specific for application targeted by CHIC to be of any general interest, and non-essential for the redeployment of VPH-HF; however, the hypermodels, simulations, and clinical data repositories, as well as the identity provider components are necessary for the VPH-HF to run. Regarding the identity provider, it should be quite straightforward to replace the call to the CHIC proprietary service to a public identity provider service such as that offered by the Google Identity Platform, or the various available SAML identity providers based on Shibboleth, for example. For the repositories, it should be equally straightforward to modify the VPH-HF to read and write the data currently held in the various repositories from a local file system. These two modifications would allow the VPH-HF to run 'out of the box', and we hope the Open Source community behind this project will introduce them soon.

The second limitation is that we conducted the profiling only on one hardware configuration. However, while the specific figures might change, the VPH-HF overhead is so limited that such changes would unlikely change our conclusions.

The third limitation is the qualitative way we reported the benefits of the VPH-HF in the construction of hypermodels. The development of a large complex hypermodel remains in a large part a complex activity, which involves considerable specialism; thus, it would be impossible to offer any reproducible quantification of the effort required to build hypermodels. However, the use of a visual editor such as the VPH-HE, and a models' repository with automated model-checking scripts can go a long way to reduce the time and effort required.

A last limitation, this of the VPH-HF itself, is that currently the VPH-HF does not directly support the distributed computing features offered by Taverna and Muscle2. Following the security and accessibility specifications imposed by the CHIC project, the VPH-HF was developed and deployed exclusively on a private cloud. The configuration adopted in the project consisted of virtual machines (VM) with the role of application server (running VPH-HF and Taverna Server), high-end VM (running MUSCLE2 and the models) and VM for repositories and VPH-Editor. A different configuration for a public cloud can be implemented in the future extending the VPH-Editor to provide accounting service and a publicly accessible interface. While it is already possible to set statically in the wrapper the IP address of the machine to which VPH-HF connects to, a skilled developer could configure a specific deployment of the VPH-HF so that the underlying workflow execution services exploits such features, for example interfacing with a scheduler like PBS deployed on a large-scale supercomputer. Looking from the system administrator perspective, VPH-HF is easy to install and configure as it requires only SSH tunnels to transfer data and run remote simulations without cumbersome system configurations.

In conclusion, the VPH-HF offers a powerful technology for the development and execution of complex hypermodels used in subject-specific biomedical modelling.

6. ACKNOWLEDGEMENTS

This study was partially funded by the European Commission FP7 programme through the project "Computational Horizons In Cancer (CHIC): Developing Meta- and Hyper-Multiscale Models and Repositories for In Silico Oncology" (Grant reference FP7-ICT- 600841). Prof Viceconti also received support from the UK Engineering and Physical Sciences Research Council (EPSRC), through the Frontier Engineering Award MultiSim, "Modelling complex

and partially identified engineering problems-Application to the individualised multiscale simulation of the musculoskeletal system" (Grant reference No. EP/K03877X/1).

7. REFERENCES

[1] J. W. Fenner, B. Brook, G. Clapworthy, P. V. Coveney, V. Feipel, H. Gregersen, D. R. Hose, P. Kohl, P. Lawford, K. M. McCormack, D. Pinney, S. R. Thomas, S. Van Sint Jan, S. Waters, and M. Viceconti, "The EuroPhysiome, STEP and a roadmap for the virtual physiological human," *Philos Trans A Math Phys Eng Sci,* vol. 366, no. 1878, pp. 2979-99, Sep 13, 2008.

[2] M. Qasim, G. Farinella, J. Zhang, X. Li, L. Yang, R. Eastell, and M. Viceconti, "Patient-specific finite element estimated femur strength as a predictor of the risk of hip fracture: the effect of methodological determinants," *Osteoporos Int,* vol. 27, no. 9, pp. 2815-2822, Sep, 2016.

[3] P. D. Morris, D. A. Silva Soto, J. F. A. Feher, D. Rafiroiu, A. Lungu, S. Varma, P. V. Lawford, D. R. Hose, and J. P. Gunn, "Fast Virtual Fractional Flow Reserve Based Upon Steady-State Computational Fluid Dynamics Analysis: Results From the VIRTU-Fast Study," *JACC Basic Transl Sci,* vol. 2, no. 4, pp. 434-446, Aug, 2017.

[4] J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. Ben Belgacem, B. Chopard, D. Groen, P. V. Coveney, and A. G. Hoekstra, "Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment," *Journal of Computational Science,* vol. 5, no. 5, pp. 719-731, 9//, 2014.

[5] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble, "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud," *Nucleic Acids Res,* vol. 41, no. Web Server issue, pp. W557-61, Jul, 2013.

[6] A. Rowe, D. Kalaitzopoulos, M. Osmond, M. Ghanem, and Y. Guo, "The discovery net system for high throughput bioinformatics," *Bioinformatics,* vol. 19 Suppl 1, pp. i225-31, 2003.

[7] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows." pp. 423-424.

[8] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "Vistrails: Enabling interactive multiple-view visualizations." pp. 135-142.

[9] K. Ovaska, M. Laakso, S. Haapa-Paananen, R. Louhimo, P. Chen, V. Aittomaki, E. Valo, J. Nunez-Fontarnau, V. Rantanen, S. Karinen, K. Nousiainen, A. M. Lahesmaa-Korpinen, M. Miettinen, L. Saarinen, P. Kohonen, J. Wu, J. Westermarck, and S. Hautaniemi, "Large-scale data integration framework provides a comprehensive view on glioblastoma multiforme," *Genome Med,* vol. 2, no. 9, pp. 65, Sep 07, 2010.

[10] E. Ciepiela, L. Zaraska, and G. D. Sulka, "GridSpace2 Virtual Laboratory Case Study: Implementation of Algorithms for Quantitative Analysis of Grain Morphology in Self-assembled Hexagonal Lattices According to the Hillebrand Method," *Building a National Distributed e-Infrastructure–PL-Grid: Scientific and Technical Achievements*, M. Bubak, T. Szepieniec and K. Wiatr, eds., pp. 240-251, Berlin, Heidelberg: Springer

Berlin Heidelberg, 2012.

[11] S. Beisken, T. Meinl, B. Wiswedel, L. F. de Figueiredo, M. Berthold, and C. Steinbeck, "KNIME-CDK: Workflow-driven cheminformatics," *BMC Bioinformatics,* vol. 14, pp. 257, Aug 22, 2013.

[12] E. Afgan, D. Baker, M. van den Beek, D. Blankenberg, D. Bouvier, M. Čech, J. Chilton, D. Clements, N. Coraor, C. Eberhard, B. Grüning, A. Guerler, J. Hillman-Jackson, G. Von Kuster, E. Rasche, N. Soranzo, N. Turaga, J. Taylor, A. Nekrutenko, and J. Goecks, "The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update," *Nucleic Acids Research,* vol. 44, no. Web Server issue, pp. W3-W10, 2016.

[13] J. Brandt, M. Bux, and U. Leser, "Cuneiform: a Functional Language for Large Scale Scientific Data Analysis." pp. 7-16.

[14] C. Pradal, C. Fournier, P. Valduriez, and S. Cohen-Boulakia, "OpenAlea: Scientific Workflows Combining Data Analysis and Simulation."

[15] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. d. Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Gener. Comput. Syst.,* vol. 46, no. C, pp. 17-35, 2015.

[16] D. Groen, A. P. Bhati, J. Suter, J. Hetherington, S. J. Zasada, and P. V. Coveney, "FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures," *Computer Physics Communications,* vol. 207, pp. 375-385, 2016/10/01/, 2016.

[17] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics,* vol. 20, no. 17, pp. 3045-3054, 2004.

[18] P. Li, J. I. Castrillo, G. Velarde, I. Wassink, S. Soiland-Reyes, S. Owen, D. Withers, T. Oinn, M. R. Pocock, C. A. Goble, S. G. Oliver, and D. B. Kell, "Performing statistical analyses on quantitative data in Taverna workflows: An example using R and maxdBrowse to identify differentially-expressed genes from microarray data," *BMC Bioinformatics,* vol. 9, no. 1, pp. 334, 2008/08/07, 2008.

[19] A. R. Hardisty, F. Bacall, N. Beard, M. P. Balcazar-Vargas, B. Balech, Z. Barcza, S. J. Bourlat, R. De Giovanni, Y. de Jong, F. De Leo, L. Dobor, G. Donvito, D. Fellows, A. F. Guerra, N. Ferreira, Y. Fetyukova, B. Fosso, J. Giddy, C. Goble, A. Guntsch, R. Haines, V. H. Ernst, H. Hettling, D. Hidy, F. Horvath, D. Ittzes, P. Ittzes, A. Jones, R. Kottmann, R. Kulawik, S. Leidenberger, P. Lyytikainen-Saarenmaa, C. Mathew, N. Morrison, A. Nenadic, A. N. de la Hidalga, M. Obst, G. Oostermeijer, E. Paymal, G. Pesole, S. Pinto, A. Poigne, F. Q. Fernandez, M. Santamaria, H. Saarenmaa, G. Sipos, K. H. Sylla, M. Tahtinen, S. Vicario, R. A. Vos, A. R. Williams, and P. Yilmaz, "BioVeL: a virtual laboratory for data analysis and modelling in biodiversity science and ecology," *BMC Ecol,* vol. 16, no. 1, pp. 49, Oct 20, 2016.

[20] N. A. Walton, D. K. Witherwick, T. Oinn, and K. Benson, *Taverna and Workflows in the Virtual Observatory*, 2008.

[21] A. Le Blanc, J. Brooke, D. Fellows, M. Soldati, D. Pérez-Suárez, A. Marassi, and A.

Santin, “Workflows for Heliophysics,” *Journal of Grid Computing,* vol. 11, no. 3, pp. 481-503, 2013/09/01, 2013.

[22] P. Li, T. Oinn, S. Soiland, and D. B. Kell, “Automated manipulation of systems biology models using libSBML within Taverna workflows,” *Bioinformatics,* vol. 24, no. 2, pp. 287-9, Jan 15, 2008.

[23] D. Groen, J. Borgdorff, C. Bona-Casas, J. Hetherington, R. W. Nash, S. J. Zasada, I. Saverchenko, M. Mamonski, K. Kurowski, M. O. Bernabeu, A. G. Hoekstra, and P. V. Coveney, “Flexible composition and execution of high performance, high fidelity multiscale biomedical simulations,” *Interface Focus,* vol. 3, no. 2, pp. 20120087, 2013.

[24] J. Borgdorff, M. Ben Belgacem, C. Bona-Casas, L. Fazendeiro, D. Groen, O. Hoenen, A. Mizeranschi, J. L. Suter, D. Coster, P. V. Coveney, W. Dubitzky, A. G. Hoekstra, P. Strand, and B. Chopard, “Performance of distributed multiscale simulations,” *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences,* vol. 372, no. 2021, pp. 20130407, 2014.

[25] J. Borgdorff, C. Bona-Casas, M. Mamonski, K. Kurowski, T. Piontek, B. Bosak, K. Rycerz, E. Ciepiela, T. Gubala, D. Harezlak, M. Bubak, E. Lorenz, and A. G. Hoekstra, “A Distributed Multiscale Computation of a Tightly Coupled Model Using the Multiscale Modeling Language,” *Procedia Computer Science,* vol. 9, pp. 596-605, 2012/01/01/, 2012.

[26] J.-L. Falcone, B. Chopard, and A. Hoekstra, “MML: towards a Multiscale Modeling Language,” *Procedia Computer Science,* vol. 1, no. 1, pp. 819-826, 2010/05/01, 2010.

[27] S. Alowayyed, D. Groen, P. V. Coveney, and A. G. Hoekstra, “Multiscale computing in the exascale era,” *Journal of Computational Science,* vol. 22, pp. 15-25, 2017/09/01/, 2017.

[28] M. Ben Belgacem, B. Chopard, and A. Parmigiani, "Coupling Method for Building a Network of Irrigation Canals on a Distributed Computing Environment." pp. 309-318.

[29] A. E. Mizeranschi, M. T. Swain, R. Scona, Q. Fazilleau, B. Bosak, T. Piontek, P. Kopta, P. Thompson, and W. Dubitzky, “MultiGrain/MAPPER: A distributed multiscale computing approach to modeling and simulating gene regulation networks,” *Future Generation Computer Systems,* vol. 63, pp. 1-14, 2016/10/01/, 2016.

[30] O. Hoenen, L. Fazendeiro, B. Scott, J. Borgdorff, A. Hoekstra, P. Strand, and D. Coster, *Designing and running turbulence transport simulations using a distributed multiscale computing approach*, 2013.

[31] H. Tahir, A. G. Hoekstra, E. Lorenz, P. V. Lawford, D. R. Hose, J. Gunn, and D. J. W. Evans, “Multi-scale simulations of the dynamics of in-stent restenosis: impact of stent deployment and design,” *Interface Focus,* vol. 1, no. 3, pp. 365-373, 2011.

[32] M. Ben Belgacem, and B. Chopard, “MUSCLE-HPC: A new high performance API to couple multiscale parallel applications,” *Future Generation Computer Systems,* vol. 67, pp. 72-82, 2//, 2017.

[33] D. Tartarini, K. Duan, N. Gruel, D. Testi, D. Walker, and M. Viceconti, "The VPH Hypermodelling framework for cancer multiscale models in the clinical practice." pp. 1-4.

[34] M. Viceconti, "Collaborative Modeling and Simulation: The Virtual Physiological

Human Vision (Full Paper)." pp. 229-234.

[35] V. Sakkalis, S. Sfakianakis, E. Tzamali, K. Marias, G. Stamatakos, F. Misichroni, E. Ouzounoglou, E. Kolokotroni, D. Dionysiou, D. Johnson, S. McKeever, and N. Graf, "Web-based workflow planning platform supporting the design and execution of complex multiscale cancer models," *IEEE J Biomed Health Inform,* vol. 18, no. 3, pp. 824-31, May, 2014.

[36] R. T. Fielding, "Architectural styles and the design of network-based software architectures," PhD Thesis, University of California, Irvine, 2000.

[37] J. Borgdorff, E. Lorenz, A. G. Hoekstra, J.-L. Falcone, and B. Chopard, "A principled approach to distributed multiscale computing, from formalization to execution." pp. 97-104.

[38] G. S. Stamatakos, D. D. Dionysiou, N. M. Graf, N. A. Sofra, C. Desmedt, A. Hoppe, N. K. Uzunoglu, and M. Tsiknakis, "The "Oncosimulator": a multilevel, clinically oriented simulation system of tumor growth and organism response to therapeutic schemes. Towards the clinical evaluation of in silico oncology," *Conf Proc IEEE Eng Med Biol Soc,* vol. 2007, pp. 6629-32, 2007.

[39] P. Hahnfeldt, D. Panigrahy, J. Folkman, and L. Hlatky, "Tumor development under angiogenic signaling: a dynamical theory of tumor growth, treatment response, and postvascular dormancy," *Cancer Res,* vol. 59, no. 19, pp. 4770-5, Oct 01, 1999.

[40] R. W. Tourdot, R. P. Bradley, N. Ramakrishnan, and R. Radhakrishnan, "Multiscale computational models in physical systems biology of intracellular trafficking," *IET Syst Biol,* vol. 8, no. 5, pp. 198-213, Oct, 2014.

[41] C. P. May, E. Kolokotroni, G. S. Stamatakos, and P. Buchler, "Coupling biomechanics to a cellular level model: an approach to patient-specific image driven multi-scale and multi-physics tumor simulation," *Prog Biophys Mol Biol,* vol. 107, no. 1, pp. 193-9, Oct, 2011.

[42] G. R. Mirams, C. J. Arthurs, M. O. Bernabeu, R. Bordas, J. Cooper, A. Corrias, Y. Davit, S. J. Dunn, A. G. Fletcher, D. G. Harvey, M. E. Marsh, J. M. Osborne, P. Pathmanathan, J. Pitt-Francis, J. Southern, N. Zemzemi, and D. J. Gavaghan, "Chaste: an open source C++ library for computational physiology and biology," *PLoS Comput Biol,* vol. 9, no. 3, pp. e1002970, 2013.

[43] J. M. Osborne, A. G. Fletcher, J. M. Pitt-Francis, P. K. Maini, and D. J. Gavaghan, "Comparing individual-based approaches to modelling the self-organization of multicellular tissues," *PLoS Comput Biol,* vol. 13, no. 2, pp. e1005387, Feb, 2017.

[44] F. A. Meineke, C. S. Potten, and M. Loeffler, "Cell migration and organization in the intestinal crypt using a lattice-free model," *Cell Prolif,* vol. 34, no. 4, pp. 253-66, Aug, 2001.

[45] S. Benzekry, C. Lamont, A. Beheshti, A. Tracz, J. M. Ebos, L. Hlatky, and P. Hahnfeldt, "Classical mathematical models for description and prediction of experimental tumor growth," *PLoS Comput Biol,* vol. 10, no. 8, pp. e1003800, Aug, 2014.

[46] A. Forrester, A. Sobester, and A. Keane, *Engineering Design via Surrogate Modelling: A Practical Guide*: John Wiley & Sons, 2008.

[47] J. P. Kleijnen, "Regression and Kriging metamodels with their experimental designs in

simulation: a review," *European Journal of Operational Research,* vol. 256, no. 1, pp. 1-16, 2017.

[48] S. Koziel, and X.-S. Yang, *Computational optimization, methods and algorithms*: Springer, 2011.

[49] A. I. Khuri, and S. Mukhopadhyay, "Response surface methodology," *Wiley Interdisciplinary Reviews: Computational Statistics,* vol. 2, no. 2, pp. 128-149, 2010.

[50] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments, 3rd Edition*: Wiley & sons, 2015.

[51] A. D. Conger, and M. C. Ziskin, "Growth of mammalian multicellular tumor spheroids," *Cancer Res,* vol. 43, no. 2, pp. 556-60, Feb, 1983.