Introducing VECMAtk - verification, validation and uncertainty quantification for multiscale and HPC simulations

Derek Groen[1], Robin A. Richardson[2], David W. Wright[2], Vytautas Jancauskas, Robert Sinclair[2], Paul Karlshoefer, Maxime Vassaux[2], Hamid Arabnejad[1], Tomasz Piontek[5], Piotr Kopta[5], Bartosz Bosak[5], Jalal Lakhlili[4], Olivier Hoenen[4], Diana Suleimenova[1], Wouter Edeling[6], Daan Crommelin[6,7], Anna Nikishova[8], and Peter V. Coveney[2,3]

[1] Department of Computer Science, Brunel University London, London, United Kingdom
[2] Centre for Computational Science, University College London, London, United Kingdom
[3] Centre for Mathematics and Physics in the Life Sciences and Experimental Biology, London, University College London, London, United Kingdom
[4] Max-Planck Institute for Plasma Physics - Garching, Munich, Germany
[5] Poznań Supercomputing and Networking Center, Poznań, Poland
[6] Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
[7] Korteweg-de Vries Institute for Mathematics, Amsterdam, The Netherlands
[8] University of Amsterdam, Amsterdam, The Netherlands

Abstract. Multiscale simulations are an essential computational tool in a range of research disciplines, and provide unprecedented levels of scientific insight at a tractable cost in terms of effort and compute resources. To provide this, we need such simulations to produce results that are both robust and actionable. The VECMA toolkit (VECMAtk), which is officially released in conjunction with the present paper, establishes a platform to achieve this by exposing patterns for verification, validation and uncertainty quantification (VVUQ). These patterns can be combined to capture complex scenarios, applied to applications in disparate domains, and used to run multiscale simulations on any desktop, cluster or supercomputing platform.

Keywords: multiscale simulations · verification · validation · uncertainty quantification.

1 Introduction

The overarching goal of computational modeling is to provide insight into questions that in the past could only be addressed by costly experimentation, if at all. In order for the results of computational science to impact decision making outside of basic science, for example in industrial or clinical settings, it is vital that they are accompanied by a robust understanding of their degree of validity.

In practice, this can be decomposed into checks of whether the codes employed are solving equations in an accurate manner (*verification*), solving the correct equations to begin with (*validation*), and providing estimates that comprehensively capture uncertainty (*uncertainty quantification*) [1,2]. These processes, collectively known as VVUQ, provide the basis for determining our level of trust in any given model and the results obtained using it [3]. Recent advances in the scale of computational resources available, and the algorithms designed to exploit them mean that it is increasingly possible to conduct the additional sampling required by VVUQ even for highly complex calculations and workflows. The goal of the VECMA project (www.vecma.eu) is to provide an open source toolkit containing a wide range of tools to facilitate the use of VVUQ techniques in multiscale, multiphysics applications. At this initial stage, these range from fusion and advanced materials through climate and forced population displacement, to drug discovery and personalised medicine.

Multiscale modeling presents particular difficulties as such applications frequently consist of complex workflows where uncertainty propagates through highly varied components, some of which may only be executed conditionally. Additionally, uncertainties may be associated with the process of transforming the output variables from one scale to another, as information is necessarily lost when coarsening a model and some kind of interpolation scheme is necessary when moving to a more fine grained representation. The goal of the VECMA toolkit (VECMAtk) is to provide open source tools which implement VVUQ approaches that range from those which treat components or workflows as an immutable 'black box' to semi-intrusive methods in which components of the workflow may be replaced by statistically representative, but cheaper, surrogate models [4]

Key to VECMAs approach is an understanding of the growing size and diversity of available supercomputers as we move to the exascale [5]. For the first time, the vast number of cores available on modern systems make it conceivable for researchers to execute the ensembles necessary to sample phase space for VVUQ analyses concurrently for even very computationally intensive simulations. Moreover, the use of ensembles of simulations offers an efficient way to use large supercomputers. However, effectively running and managing multiscale workflows composed from components with divergent computational requirements (for example with some models capable of exploiting GPUs and others not) represents a key challenge our tools must address. Nonetheless, advances in hardware continue to present opportunities that can be exploited. For example, some intrusive VVUQ methods which make extensive use of machine learning techniques are able to take advantage of hardware designed to accelerate these techniques (such as fast SSD storage). The support for compute intensive applications and workflow management within VVUQ analyses are key factors motivating the development of the VECMAtk.

The philosophy of VECMAtk is that, as far as possible, we want to be able to add VVUQ to existing scientific workflows without changing researchers conception of the problem they are investigating. This goal informs the conception of

the toolkit which is designed to: (i) be highly modular, (ii) have minimal installation requirements and, (iii) provide control over where (locally or on remote resources) and at what time analysis takes place.

This means that VECMAtk is a collection of elements that can be reused and recombined in different workflows. Our aim is to define stable interfaces, data formats and APIs that facilitate VVUQ in the widest range of applications. For example, we intend to provide tools which handle the encoding and comparison of variables which are represented as distributions as they propagate through models. The modularity of the toolkit allows us to account for the differing requirements of VVUQ sampling and analysis and provide tools tailored for the needs of both steps of the process. Several software packages or libraries are already available for performing VVUQ (such as OpenTurns [6], UQLab [7], Uncertainpy [8], Chaospy [9], etc.) but in many cases these are closed source and none of them provide the separation of concerns needed to allow the analysis of both small local computaions and highly compute intensive kernels (potentially using many thousands of cores and GPUs on HPC or cloud resources). However, we intend to make use of existing tools where appropriate to provide robust and optimised code for sampling and analysis.

Key to enabling the largest range of researchers to use the toolkit will be the provision of exemplar workflows and documentation, which they can base their own approaches on. Essential to the design of these exemplars is the need to lower the barriers to usage and, naturally, to be transferable to a large range of applications with minimal modification.

2 The VECMA toolkit

The goal of the toolkit is to facilitate the process of researchers mapping the requirements of VVUQ to their specific scientific problem and existing workflow(s). The main factors that shape the development approach of VECMAtk are: a) the need to fit into existing applications with minimal modification effort, b) the wide range of target application domains, c) the flexible and recombinable nature of the toolkit itself, d) the geographically distributed nature of the users and particularly the developers.

To support these needs and characteristics, we have chosen to adopt an evolutionary prototyping approach. In this approach, existing application developers initially establish VVUQ techniques using their own scripts, which they share with the VECMAtk developers together with additional needs that they have not been able to easily address themselves. These scripts and requirements, together with existing libraries, form the base from which the development team develops initial prototypes. The prototypes are then tested and refined at frequent intervals, with the user feedback and integration testing guiding further developments. As a result, some prototypes are reduced in scope, simplified, or removed altogether; and some prototypes are being refined into more advanced, flexible, scalable and robust tools. Regular development meetings within the project help to monitor and disseminate progress as well as providing a venue in

which we ensure that all component development teams are following best development practices (for example making use of version control and continuous integration).

As part of our prototyping process, we identify common workflow patterns and software elements (for example those needed to encode complex parameter distributions) that can be abstracted for re-use in a wide range of application scenarios. We label patterns found in verification and validation contexts *VVPs*, and those associated with uncertainty quantification or sensitivity analysis *UQPs*. The definition of a VVP or UQP should never require the use of any specific execution management platform, as the toolkit is envisioned to provide multiple solutions that facilitate workflows. Examples include diverse sampling algorithms and job types running on heterogeneous resources.

Within VECMAtk, we categorize development into fast-track development, which treats underlying applications as a black box (*non-intrusive*), and deep-track development, which takes into account the coupling mechanisms between single models (*semi-intrusive*) or even the algorithms of the single scale models themselves (*fully intrusive*).

In the scenarios where a single run of an application is demanding or a large number of application replicas has to be executed to guarantee reliable VVUQ, the easy and efficient access to computing power is crucial. In order to fulfil this requirement, the VECMA workflows make benefit of the QCG software (www.qoscosgrid.org) which provides advanced capabilities for execution in a unified way of complex jobs on infrastructure consisting of one or many computing clusters. The QCG infrastructure, which is presented in Figure 1, features at the heart the QCG-Broker service, which is responsible for managing execution of the computational experiment, including the multi-criteria selection of resources. In addition, several QCG-Computing services offer unified and remote access to underlying resources. The QCG services can be accessed with numerous user-level tools [10], of which a few examples are provided in the aforementioned figure.

The combination of different UQPs and VVPs in one application also leads to a cognitively complex workflow structure, where different sets of replicas need to be constructed, organized, executed, analyzed, and actioned upon (i.e. triggering subsequent execution and/or analysis activities). FabSim3 [11] (`https://github.com/djgroen/FabSim3`) is a freely available too that supports the construction, curation and execution of these complex workflows, and allows users to invoke them on remote resources using one-liner commands. In contrast to its direct predecessor FabSim, FabSim3 inherently supports the execution of job ensembles, and provides a plug-in system which allows users to customize the toolkit in a modular and lightweight manner (e.g., evidenced by the minimalist open-source FabDummy plugin). We provide an overview of the FabSim3 architecture in Figure 3 In the context of VECMA FabSim3 plays a key role in introducing application-specific information in the Execution Layer, and in conveniently combining different UQPs and VVPs.

As a planned scenario, Fabsim3 uses QCG-Client or QCG-SDK to submit a so-called Pilot Job. Such a job can be seen as a container for many subjobs that can be started and managed without individual subjobs having to wait for resources to become available, increasing efficiency. Once the Pilot Job is submitted, it may service a number of defined VVUQ subtasks (e.g., a set of runs defined by EasyVVUQ). The QCG Pilot Job mechanism provides two interfaces that may be used interchangeably. The first one allows to specify a file with the description of subjobs and execute the scenario in a batch-like mode, conveniently supporting static scenarios. The second interface is offered with the REST API and it can be accessed remotely in a more dynamic way. It will be used to support scenarios where a number of replicas and their complexity dynamically changes at application runtime.

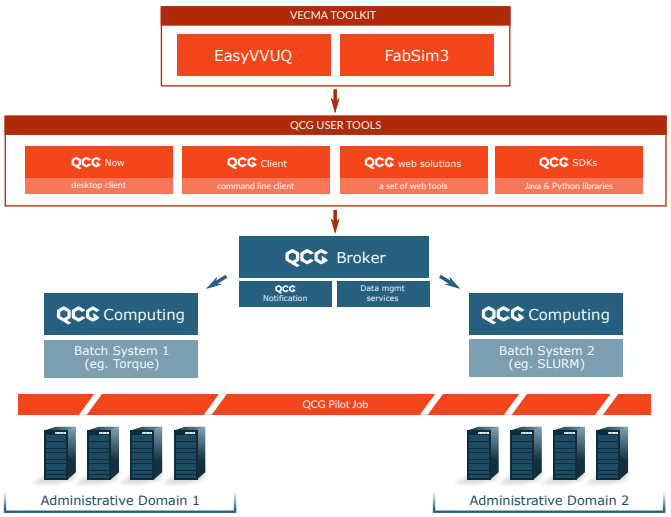

Fig. 1: Simplified overview of QCG usage in VECMA. Jobs requested by the toolkit layer may be farmed out to one or more queues on different computing resources.

EasyVVUQ is a Python library designed to simplify the implementation of creation of (primarily blackbox) VVUQ workflows involving existing applications. The library is designed around a breakdown of such workflows into four distinct stages; sampling, simulation execution, result aggregation, and analysis. The execution step is deemed beyond the remit of the package (it can be handled for instance by FabSim3 or QCG Client), whilst the other three stages are handled separately. A common data structure, the *Campaign*, which contains information on the application being analyzed alongside the runs mandated by the sampling algorithm being employed, is used to transfer information between each stage. The architecture of EasyVVUQ is shown in Figure 2.

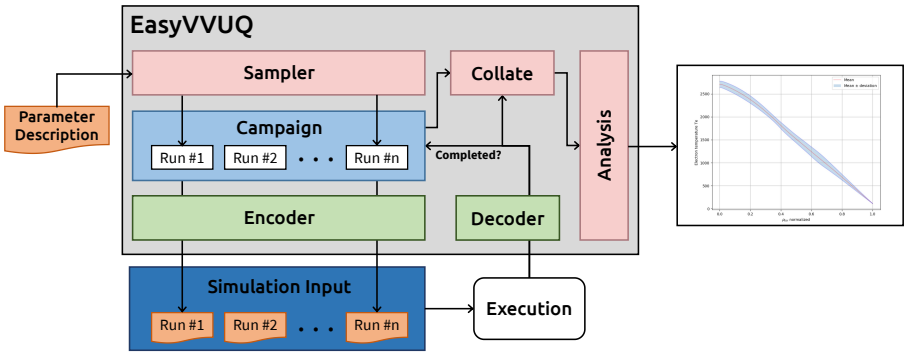

Fig. 2: The architecture of EasyVVUQ. The UQ workflow is split into a sampling, execution and analysis stage, orchestrated via a (persistent) 'Campaign' object.

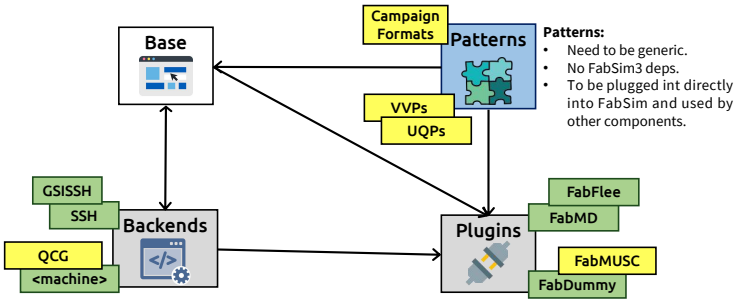

Fig. 3: Essential building blocks in the FabSim3 component of VECMAtk, and their interdependencies. Components in yellow are under development as of June 2019

The user provides a description of the model parameters and how they might vary in the sampling phase of the VVUQ pattern, for example specifying the distribution from which they should be drawn and physically acceptable limits on their value. This is used to define a *Sampler* which populates the Campaign with a set of run specifications based on the parameter description provided by the user. The Sampler may employ one of a range of algorithms such as the Monte Carlo or Quasi Monte Carlo approaches [12]. At this point all of the information is generic in the sense that it is not specific to any application or workflow. The role of the *Encoder* is to create input files which can be used in a specific application. Included in the base application is a simple templating system in which values from the Campaign are substituted into a text input file. For many applications it is envisioned that specific encoders will be needed and the framework of EasyVVUQ means that any class derived from a generic Encoder base class is picked up and may be used. This enables EasyVVUQ to be easily extended for new applications by experienced users. The simulation input is then used externally to the library to execute the simulations. The role of the *Decoder* is twofold, to record simulation completion in the Campaign and to extract the output information from the simulation runs. Similarly to the *Encoder* the *Decoder* is designed to be user extendable to facilitate analysis of a wide range of applications. The Decoder is used in the collation step to provide a combined and generic expression of the simulation output for further analysis (for example the default is to bring together output from all simulation runs in a Pandas dataframe). Following the output collation we provide a range of analysis algorithms which produce the final output data. Whilst the library was originally designed for acyclic 'blackbox' VVUQ workflows development is ongoing to allow the library to be used in more complex patterns.

3 Initial applications

The following section gives an overview of applications that currently use the VECMAtk and are guiding the development of new features for the toolkit. We present a detailed look at a fusion calculation and brief description of climate, population displacement, materials, force field, and cardiovascular modeling.

3.1 Fusion example

Heat and particle transport in a fusion device play a major role in the performance of the thermo-nuclear fusion reaction. Current understanding is that turbulence arising at the micro space and time scales is a key factor for such transport which has effects on much larger scales. In order to bridge such disparate spatiotemporal scales, multiscale simulation are being developed, for instance in [13] by coupling a gyro-fluid 3D turbulence submodel to a 1D transport solver that evolves the temperature profiles over the macro scales. The turbulence submodel provides heat fluxes from which the transport coefficients are derived, but its output is inherently noisy. Hence, the calculated profiles are exposed to this

noisy signal, and uncertainties will propagate from one submodel to the next. Additional uncertainties come from external sources (which can be prescribed or coming from additional models, e.g heating and current drive), and from experimental measurements against which the simulation could be validated, which leads to complex scenario as depicted in Figure 4.

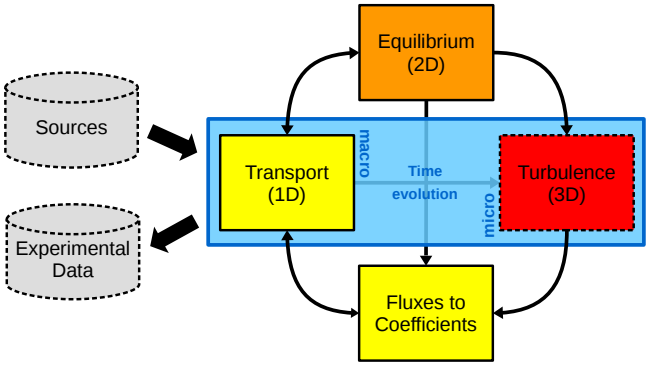

Fig. 4: Schematic view of the targeted fusion application. In this workflow, the components which produces uncertain data are drawn with dashed lines and the color scheme reflects compute intensity: while *Transport* and *Fluxes to Coefficients* are very cheap models usually implemented by serial codes, *Equilibrium* could be parallelized within a node, while *Turbulence* codes typically require parallelization on a large number of nodes.

The goal is therefore to produce simulated quantities of interest (temperature profile, density, etc..) and their confidence intervals and propagate this additional information through a complex cycling workflow made of several components which have different properties, computational costs and uncertain inputs. This information should allow for better validation of the interpretative simulation against experimental results (for existing tokamaks machines) as well as to increase the confidence of predictive simulations.

Following previous work on quantifying the propagation of uncertainty through a 'black-box' model in fusion plasma [14], we started to apply a polynomial chaos expansion method (PCE) to the cheaper single-scale models from Figure 4 but taken separately (each one becomes a black-box). The method doesn't require changes to the underlying model equations and provides a quantitative measure for which combinations of inputs have the most important impact on the results. The PCE coefficients are used to compute statistical metrics (typically mean and covariance) that are essential to have the basic descriptions of the output distribution. To compute those coefficients we use a quadrature scheme which depends on the polynomial type which itself depends on the probability distribution of uncertain inputs. As a result, integration rules along each axis

can be calculated using a tensor product. Initial results for the 1D transport model are depicted in Figure 5.

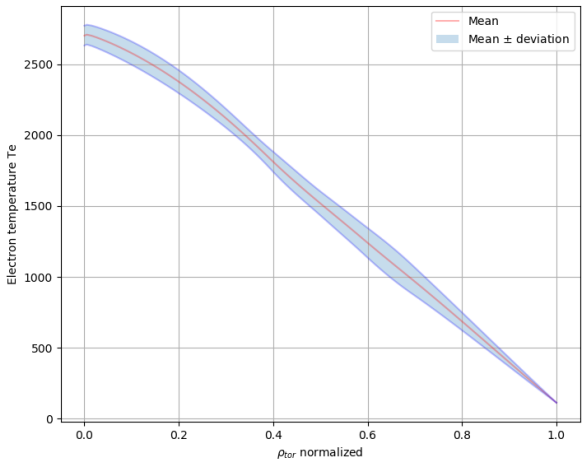

Fig. 5: Uncertainty estimation results of the temperature profile for the 1D transport model from Figure 4.

Implementation of this case in Python, given in the snippet of code in Figure 6, shows how such use case (using here the PCE method and based on available library chaospy[9]) matches the different steps of the logical structure of EasyVVUQ (Figure 2).

Even with a limited number of uncertain parameters resulting from the turbulence code (e.g 8 for a fluid code using a flux-tube approximation), and assuming these parameters are not correlated, this method will necessitate approximately 1.7 millions of runs of *Fluxes to Coefficients* and *Transport* codes if we want to calculate their propagation through these models. As these are the cheapest codes in the application, this represents only 512 cpu-hours of an embarrassingly parallel job which remains tractable with traditional means. But when the complexity of the models and the number of parameters increase, such quantities of runs with potentially large frange of runtimes becomes very challenging and will require using advanced capabilities of 'smart' pilot-job software.

3.2 Variety of other applications

Climate Computational models for atmospheric and oceanic flows are central to climate modeling and weather forecasting. As these models have finite resolutions, they employ simplified representations, so-called parameterizations, to

[9] `https://github.com/jonathf/chaospy`

```
# Input file containing parameters of interest
coret = read("ets_coretransp_in.cpo", "coretransp")

# Uncertain parameters (numbers n=2) and distributions
diff_eff = coret.values[0].te_transp.diff_eff
dist_1 = cp.Normal(diff_eff[0], 0.2*diff_eff[0])
dist_2 = cp.Normal(diff_eff[1], 0.2*diff_eff[1])

# Joint probability distribution
dist = cp.J(dist_1, dist_2)

# Hermite polynomials (order k=4)
P = cp.orth_ttr(k, dist)

# Encoder: preparing the sampling
nodes, weights = cp.generate_quadrature(order=k+1, domain=dist, rule='Gaussian')
samples_te = [0]*(k+2)**n

# Evaluate the computational model in the sample points (nodes)
for i in range((k+2)**n):
    # Encoder: simulation input
    tmp_file_in  = update_coret("/ets_coretransp_in.cpo", nodes.T[i])

    # Run execution: call transport code
    run_ets(tmp_file_in, tmp_file_out)

    # Decoder: aggregate the results
    corep = read(tmp_file_out, "coreprof")
    samples_te[i] = corep.te.value

# Collate: create approximate solver
te_hat = cp.fit_quadrature(P, nodes, weights, samples_te)

# Analysis: compute statistical informations
mean     = cp.E(te_hat, dist)
variance = cp.Var(te_hat, dist)
std      = cp.Std(te_hat, dist)
```

Fig. 6: Source code snippet showing the usage of the PCE treating a 1D transport model (*ets*) as a black box. In this code, alias `cp` refers to the chaospy library.

account for the impact of unresolved processes on larger scales. An example is the treatment of atmospheric convection and cloud processes, which are important for the atmospheric circulation and hydrological cycle, but are unresolved in global atmosphic models. These parameterizations are a source of uncertainties: they have parameters that can be difficult to determine, and even their structural form can be uncertain.

A computationally very expensive approach to improve parameterizations and reduce uncertainty is by locally replacing the parameterization with a high-resolution simulation. In [15] this is applied regionally, by nesting the Dutch Atmospheric Large-Eddy Simulation (DALES) model in a selected number of global model columns, replacing the convection parameterization in these columns. The local DALES models run independently from each other and only exchange mean profiles with the global model. While this set-up allows for the use of massively parallel computer systems, running a cloud-resolving simulation in every single model column of a global model remains computationally unfeasible.

Within VECMA we are therefore developing tools for surrogate modeling. More specifically, we aim for statistically representative, data-driven surrogate models that account for the uncertainties in subgrid-scale responses due to their internal nonlinear and possibly chaotic dynamics. The surrogates are to be constructed from a (limited) set of reference data, with the goal of accurately reproducing long-term statistics, in line with the approach from [16,17].

Forced displacement Accurate predictions of forced population displacement can help governments and NGOs in making decisions as to how to help refugees, and efficiently allocate humanitarian resources to overcome unconscious consequences [18].We enable these simulations by establishing an automated agent-based modeling approach, FabFlee, which is a plugin to FabSim3 that uses the Flee agent-based simulation code. Flee forecasts the distribution of incoming refugees across destination camps [19], while FabFlee provides an environment to run simulation ensembles under various policy constraints, such as forced redirections, camp and border closures [20]. It also provides supports data curation and processing [21,22]. At time of writing, we are combining FabFlee with EasyVVUQ to more efficiently perform sensitivity analysis for varying agent awareness levels and speed limits of refugee movements [20].

Materials modeling Prediction of nanocomposite material properties requires multiscale workflows that capture mechanisms at every characteristic scale of the material, from chemical specificity to engineering testing conditions. For nanocomposite systems, the characteristic time and length of its nanostructure and macrostructure are so far apart, their respective dynamics can be simulated separately. We use DealLAMMPS[23,24], a new program that simulates the nanoscale with LAMMPS molecular dynamics, and the macroscale is simulated using deal.II, a finite element solver. Boundary information is passed from the FEM model to the LAMMPS simulations, the stresses arising from these changes are used to propagate the macroscale model. This workflow creates a

vast number of short nanoscale, unpredictable at runtime, simulations at each macroscale timestep. Handling the execution of these task requires automation and coordination between several resources. As just the nanoscale errors due to uncertainty in the boundary condition and even the stress measured for a boundary change are correlated, complex and expensive to calculate. Over several iterations, errors can proliferate and careful consideration and understanding of these is needed from tools that VECMAtk can provide.

Molecular dynamics force fields Molecular dynamics calculations are used not only in materials modeling but in a wide range of other fields. Choices made in the design of these calculations such as the parameterization of the force field describing chemical components within the system and cut-offs used for long range interactions can have a profound effect on the results obtained and their variability. One field in which this of particular interest in free energy calculations which are increasingly widely used in modern drug design and refinement workflows. The binding affinity calculator (BAC) [25] we have developed automates this class of calculation, from model building through simulations and analysis. In order to understand the impact of forcefield parameter decisions on calculations performed using the BAC we are creating new workflows which incorporate sensitivity analysis through EasyVVUQ. The use of protocols based on ensemble simulations (known as TIES and ESMACS [26]) will give us the ability to adjust the simulation duration and ensemble size in order to robustly determine the uncertainty of results for comparison. This effort builds on previous work which uses Pilot Job manager to handle the execution of ensembles of multiple runs to enable bootstrap error statistics to be applied to calculations for individual protein-ligand pairs.

Cardiovascular simulation Haemodynamic simulations provide a non-invasive means of estimating flow rates, pressures and wall shear stresses in the human vasculature. Through MRI and CT scans, patient specific models may be used, with clinical applications such as predicting aneurysm rupture or treatment. We use a 3D lattice-Boltzmann solver, HemeLB [27], to simulate the continuum dynamics of bloodflow through large and highly sparse vascular systems efficiently[28]. Clinical application requires us to understand how our results are sensitive to changes in basic input parameters such as blood viscosity, mesh resolution, rheology model, and consequently the uncertainty estimates we may make regarding the quantities of interest (pressures, velocity field, wall shear stresses etc). A recent validation study focused on HemeLB simulations of a real patient Middle Cerebral Artery (MCA), using transcranial Doppler measurements of the blood velocity profile for comparison, as well as exploring the effects of a change in rheology model or inlet flow rate on the results [29]. We are now running full 3D simulations of the entire human arterial tree, with an aim of also integrating the venous tree and a cyclic coupling to state-of-the-art heart models. This necessarily introduces an even greater number of input parameters which, when combined with the great computational expense of the submodels,

presents a real challenge for validation and verification of our application, and particularly with regards to a computationally feasible sensitivity analysis and uncertainty quantification within such a system.

Biomedical model Coronary artery stenosis is a cardiovascular disease of narrowing of the coronary artery due to clustering of fatty plaque. A common treatment is to dent the fatty plaque into the artery wall and to deploy a stent in order to keep the artery open. However, up to 10% cases end up in the re-narrowing of the artery due to an excessive healing response, which is called in-stent restenosis (ISR) [30]. The multiscale in-stent restenosis model simulates this process at different time scales [31].

In [32], uncertainty of the response of the two-dimensional ISR model on the cross-sectional lumen area was estimated and analyzed. Uncertainty quantification showed up to 15% aleatory and about 25 % total uncertainty in the model predictions. Additionally, sensitivity analysis identified the endothelium regeneration time as the most influential parameter on the model response.

In [33], the semi-intrusive multiscale metamodeling uncertainty quantification method was applied to improve the performance of the uncertainty analysis. Depending on the surrogate used, the simulation time of the semi-intrusive method was up to five times faster than of a Monte Carlo method. In future work, the semi-intrusive metamodeling method will be applied to the three dimensional version of the ISR model employing the VECMAtk, since a black-box approach is not feasible for this application due to high computational demand [34].

4 Roadmap and release strategy

For the VECMA toolkit we have adopted a release schedule incorporating two types of release. Minor releases are tagged every 3 months, advertised within the project, and made available to the public without dedicated advertising. These releases incorporate new functionalities and components, and are accompanied by a limited amount of additional documentation and examples. Major releases will be made in June 2019, June 2020 and December 2020, fully advertised and made available publicly. The major releases will be accompanied with extensive documentation and examples, and we will hold training events and other dedicated uptake activities associated with these releases. We are at present able to guarantee formal support for the VECMA toolkit up to June 2021, providing us with 6 months of time to future-proof VECMAtk at least until 6 months after the final planned release of the toolkit. The June 2019 VECMAtk release will contain FabSim3, EasyVVUQ, as well as functionalities provided by QCG, and we expect additional components to be introduced in later releases. In-between releases, users will be able to check out any version of the code for FabSim3 and EasyVVUQ, as we are maintaining an open development environment.

4.1 Containerization of VECMAtk

Containerization [35] is an operating system level virtualization method used to deploy and run distributed applications without launching an entire virtual machine (VM) for each application. Containers allows us to create a virtual machine (VM) in very light-weight fashion compared to the traditional virtualization approach, as they share the operating system kernel, and they also share resources such as storage and networking. Additionally, application containers may be migrated to other computing environment, such as computing system, clouds, or other environments without requiring code changes, i.e, application platform portability. Each container includes the runtime components – such as files, environment variables and libraries – necessary to run the desired software, and run on a single control host, accessing a single kernel. There are few container implementation possibilities, but in our project, Docker [36] is used for the containerization of our software. A typical Docker workflow to create images, pull images, publish images, and run containers from a `Dockerfile` is shown in Figure 7. A `Dockerfile` is a script that contains collections of commands and instructions that will be automatically executed in sequence in the docker environment for building a new docker image. For example, a container could be a based on Ubuntu OS running a version of python with pre-installed library required by a given application. This container can be started and stopped in the same way as a VM. Finally, we can push our container image to a Docker registry, i.e, Docker Hub, for the world to use.

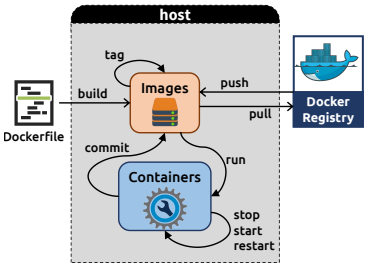

Fig. 7: A typical Docker workflow

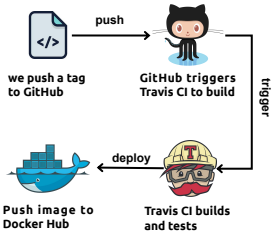

Fig. 8: GitHub + TravisCI + Docker Hub

In this work, we set up the FabSim github repository with a `Dockerfile` and Travis file, as it shown in Figure 8. After each successful test executed by Travis, based on the `Dockerfile`, a docker image is configured and built, and pushed into Docker Hub. For this work, A Docker image is provided as well through the Docker Hub[10]. And, the docker bundle for easy deployment is available in Docker Hub via : `docker pull vecmafabsim3/fabsimdocker`.

[10] https://hub.docker.com/r/vecmafabsim3/fabsimdocker

However, although Docker is one of most popular container technologies for software reproducibility, it has low adoption in the HPC world since it requires users with root access to run Docker and execute a containerized applications. To support high performance computing use cases, where users should only have access to their own data, singularity [11] can be used as a solution for container system in HPC environment. Singularity containers differ from Docker containers in several important ways, including the handling of namespaces, user privileges, and the images themselves. As part of the ongoing work, we intend to provide a singularity container alongside docker image for this toolkit.

5 Conclusions

In this paper we have outlined the design and development process used in the creation of the VECMA toolkit for validation, verification and uncertainty quantification (VVUQ) of multiscale HPC applications. A number of exemplar applications from a diverse range of scientific domains (fusion, climate, population displacement, materials, drug binding affinity, and cardiovascular modeling) are being used to test and guide the development of new features for the toolkit. Through this work we aim to make VVUQ certification of complex, multiscale workflows on high end computing facilities a standard practice.

6 Acknowledgements

We acknowledge funding support from the European Union's Horizon 2020 research and innovation programme under grant agreement 800925 (VECMA project, www.vecma.eu).

References

1. Oberkampf, W.L., DeLand, S.M., Rutherford, B.M., Diegert, K.V., Alvin, K.F.: Error and uncertainty in modeling and simulation. Reliability Engineering & System Safety **75**(3) (2002) 333 – 357
2. Oberkampf, W.L., Roy, C.J.: Verification and validation in scientific computing. Cambridge University Press (2010)
3. Roy, C.J., Oberkampf, W.L.: A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing. Computer Methods in Applied Mechanics and Engineering **200**(25) (2011) 2131 – 2144
4. Nikishova, A., Hoekstra, A.G.: Semi-intrusive uncertainty quantification for multiscale models. arXiv preprint arXiv:1806.09341 (2018)
5. Alowayyed, S., Groen, D., Coveney, P.V., Hoekstra, A.G.: Multiscale computing in the exascale era. Journal of Computational Science **22** (2017) 15 – 25
6. Baudin, M., Dutfoy, A., Iooss, B., Popelin, A.L.: Openturns: An industrial software for uncertainty quantification in simulation. Handbook of Uncertainty Quantification (2017) 2001–2038

[11] https://singularity.lbl.gov

7. Marelli, S., Sudret, B.: Uqlab: A framework for uncertainty quantification in matlab. In: Vulnerability, Uncertainty, and Risk: Quantification, Mitigation, and Management. (2014) 2554–2563
8. Tennøe, S., Halnes, G., Einevoll, G.T.: Uncertainpy: A python toolbox for uncertainty quantification and sensitivity analysis in computational neuroscience. bioRxiv (2018) 274779
9. Feinberg, J., Langtangen, H.P.: Chaospy: An open source tool for designing methods of uncertainty quantification. Journal of Computational Science **11** (2015) 46–57
10. Piontek, T., Bosak, B., Ciżnicki, M., Grabowski, P., Kopta, P., Kulczewski, M., Szejnfeld, D., Kurowski, K.: Development of science gateways using qcg — lessons learned from the deployment on large scale distributed and hpc infrastructures. Journal of Grid Computing **14**(4) (Dec 2016) 559–573
11. Groen, D., Bhati, A.P., Suter, J., Hetherington, J., Zasada, S.J., Coveney, P.V.: Fabsim: Facilitating computational research through automation on large-scale and distributed e-infrastructures. Computer Physics Communications **207** (2016) 375–385
12. Sobol, I.: On quasi-monte carlo integrations. Mathematics and Computers in Simulation **47**(2) (1998) 103 – 112
13. Luk, O., Hoenen, O., Bottino, A., Scott, B., Coster, D.: Compat framework for multiscale simulations applied to fusion plasmas. Computer Physics Communications (2019)
14. Preuss, R., von Toussaint, U.: Uncertainty quantification in ionsolid interaction simulations. Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms **393** (2017) 26 – 28 Computer Simulation of Radiation effects in Solids Proceedings of the 13 COSIRES Loughborough, UK, June 19-24 2016.
15. Jansson, F., van den Oord, G., Pelupessy, I., Grönqvist, J., Siebesma, A., Crommelin, D.: Regional superparameterization in a global circulation model using large eddy simulations. (in press) (2018)
16. Verheul, N., Crommelin, D.: Data-driven stochastic representations of unresolved features in multiscale models. Commun. Math. Sci **14**(5) (2016) 1213–1236
17. Verheul, N., Viebahn, J., Crommelin, D.: Covariate-based stochastic parameterization of baroclinic ocean eddies. Mathematics of Climate and Weather Forecasting **3**(1) (2017) 90–117
18. Groen, D.: Simulating refugee movements: Where would you go? Procedia Computer Science **80** (2016) 2251–2255
19. Suleimenova, D., Bell, D., Groen, D.: A generalized simulation development approach for predicting refugee destinations. Scientific Reports **7:13377** (2017)
20. Suleimenova, D., Groen, D.: How policy decisions affect refugee journeys in South Sudan: A study using automated ensemble simulations. Submitted to the Journal of Artificial Societies ans Social Simulation (2019)
21. Suleimenova, D., Bell, D., Groen, D.: Towards an automated framework for agent-based simulation of refugee movements. In Chan, W.K.V., DAmbrogio, A., Zacharewicz, G., Mustafee, N., Wainer, G., Page, E., eds.: Proceedings of the 2017 Winter Simulation Conference, Las Vegas, Nevada, IEEE (2017) 1240–1251
22. Chan, N., Suleimenova, D., Bell, D., Groen, D.: Modelling refugees escaping violent events: A feasibility study from an input data perspective. In Anagnostou, A., Meskarian, R., Robertson, D., eds.: Proceedings of the Operational Research Society 9th Simulation Workshop, Worcestershire, England (2018) 156–163

23. Vassaux, M., Richardson, R.A., Coveney, P.V.: The heterogeneous multiscale method applied to inelastic polymer mechanics. Phil. Trans. R. Soc. A **377:20180150** (2019)
24. Vassaux, M., Sinclair, R.C., Richardson, R.A., Suter, J.L., Coveney, P.V.: The role of graphene in enhancing the mechanical properties of epoxy resins. Advanced Theory and Simulations **2** (2019) 1800168
25. Sadiq, S.K., Wright, D., Watson, S.J., Zasada, S.J., Stoica, I., Coveney, P.V.: Automated molecular simulation based binding affinity calculator for ligand-bound hiv-1 proteases. Journal of Chemical Information and Modeling **48**(9) (2008) 1909–1919 PMID: 18710212.
26. Wan, S., Bhati, A.P., Zasada, S.J., Wall, I., Green, D., Bamborough, P., Coveney, P.V.: Rapid and reliable binding affinity prediction of bromodomain inhibitors: A computational study. Journal of Chemical Theory and Computation **13**(2) (2017) 784–795 PMID: 28005370.
27. Mazzeo, M.D., Coveney, P.V.: HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries. Computer Physics Communications **178**(12) (2008) 894–914
28. Patronis, A., Richardson, R.A., Schmieschek, S., Wylie, B.J., Nash, R.W., Coveney, P.V.: Modelling patient-specific magnetic drug targeting within the intracranial vasculature. Frontiers in physiology **9** (2018) 331
29. Groen, D., Richardson, R.A., Coy, R., Schiller, U.D., Chandrashekar, H., Robertson, F., Coveney, P.V.: Validation of patient-specific cerebral blood flow simulation using transcranial doppler measurements. Frontiers in Physiology **9** (2018) 721
30. Fernández-Ruiz, I.: Interventional cardiology: Drug-eluting or bare-metal stents? Nature Reviews Cardiology **13**(11) (2016) 631–631
31. Caiazzo, A., Evans, D., Falcone, J.L., Hegewald, J., Lorenz, E., Stahl, B., Wang, D., Bernsdorf, J., Chopard, B., Gunn, J., Hose, R., Krafczyk, M., Lawford, P., Smallwood, R., Walker, D., Hoekstra, A.: A complex automata approach for in-stent restenosis: Two-dimensional multiscale modelling and simulations. Journal of Computational Science **2**(1) (2011) 9 – 17
32. Nikishova, A., Veen, L., Zun, P., Hoekstra, A.G.: Uncertainty quantification of a multiscale model for in-stent restenosis. Cardiovascular Engineering and Technology **9**(4) (Dec 2018) 761–774
33. Nikishova, A., Veen, L., Zun, P., Hoekstra, A.G.: Semi-intrusive multiscale metamodeling uncertainty quantification with application to a model of in-stent restenosis. Philosophical Transactions A (2018)
34. Zun, P.S., Anikina, T., Svitenkov, A., Hoekstra, A.G.: A comparison of fully-coupled 3D in-stent restenosis simulations to in-vivo data. Frontiers in Physiology **8**(May) (may 2017) 284
35. Docker: Docker for the Virtualization Admin. eBook (2016)
36. Docker: Docker documentation `https://docs.docker.com`.