Poznan University of Technology
Faculty of Computing and Telecommunications
Institute of Computing Science

Doctoral dissertation

METHODS FOR MODELLING AND ASSESSING CONFIDENCE OF HETEROGENEOUS MULTISCALE SIMULATIONS IN HIGH-PERFORMANCE COMPUTING ENVIRONMENTS

Bartosz Bosak, M.Sc. Eng.

Supervisor
Krzysztof Kurowski, Ph. D., Dr. Habil

Poznań, 2025

Abstract in English

This dissertation summarises recent research on multiscale computing methods, leading to the development of abstract models and system-level solutions for multiscale simulations in high-performance computing (HPC) environments. The study presents innovative methods developed at the Poznań Supercomputing and Networking Center, introducing novel approaches to integrating computational models while addressing key challenges such as computational complexity, accuracy, and uncertainty propagation.

The study presents innovative methods that introduce novel approaches to the integration of computational models while addressing key challenges such as computational complexity, accuracy, and uncertainty propagation. Multiscale computing has become a critical tool in scientific and engineering simulations, enabling the integration of models across different spatial and temporal scales. This approach improves simulation precision and applicability, overcoming the limitations of traditional modelling techniques.

Given the complexity of multiscale simulations, frequent data exchanges between models can have a significant influence on uncertainty propagation and lead to reduced result accuracy. The research proposes original procedures for verification, validation and uncertainty quantification (VVUQ), as well as sensitivity analysis (SA), to improve reliability of the complex computational models. In particular, the dissertation addresses challenges related to parallel processing of model evaluations in high-performance systems, inherently connected to execution of uncertainty quantification algorithms.

Another key aspect discussed in the dissertation is the automation and streamlining of the execution of complex computational workflows, multiscale and VVUQ in particular. The research outlines and compares several solutions developed with the aim of simplification of typically cumbersome processes related to the preparation, submission, and management of such computational experiments conducted in HPC environments.

The findings of this study confirm the research hypothesis that the application of computational strategies for multiscale modelling and VVUQ streamlines development, improves execution efficiency, and fosters trust in advanced scientific applications. The proposed methods contribute to the advancement of multiscale computing and VVUQ by improving their usability, efficiency, and reliability in HPC-based execution scenarios. Furthermore, they support the development of next-generation computational frameworks, enabling more accurate, scalable, and reliable multiscale modelling across diverse scientific and engineering domains.

The thesis consists of the following works:

[**P1**] Borgdorff, J., Bona-Casas, C., Mamonski, M., Kurowski, K., Piontek, T., Bosak, B., Rycerz, K., Ciepiela, E., Gubala, T., Harezlak, D., Bubak, M., Lorenz, E., and Hoekstra, A. G. (2012). A distributed multiscale computation of a tightly coupled model using the multiscale modeling

language. *Procedia Computer Science*, 9:596–605. Proceedings of the International Conference on Computational Science, ICCS 2012

[**P2**] Borgdorff, J., Mamonski, M., Bosak, B., Groen, D., Belgacem, M. B., Kurowski, K., and Hoekstra, A. G. (2013b). Multiscale computing with the multiscale modeling library and runtime environment. *Procedia Computer Science*, 18:1097–1105. 2013 International Conference on Computational Science

[**P3**] Borgdorff, J., Mamonski, M., Bosak, B., Kurowski, K., Ben Belgacem, M., Chopard, B., Groen, D., Coveney, P., and Hoekstra, A. (2014). Distributed multiscale computing with muscle 2, the multiscale coupling library and environment. *Journal of Computational Science*, 5(5):719–731

[**P4**] Alowayyed, S., Piontek, T., Suter, J., Hoenen, O., Groen, D., Luk, O., Bosak, B., Kopta, P., Kurowski, K., Perks, O., Brabazon, K., Jancauskas, V., Coster, D., Coveney, P., and Hoekstra, A. (2019). Patterns for high performance multiscale computing. *Future Generation Computer Systems*, 91:335–346

[**P5**] Bosak, B., Piontek, T., Karlshoefer, P., Raffin, E., Lakhlili, J., and Kopta, P. (2021). Verification, validation and uncertainty quantification of large-scale applications with qcg-pilotjob. In Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, a. V., Dongarra, J. J., and Sloot, P. M., editors, *Computational Science – ICCS 2021*, pages 495–501, Cham. Springer International Publishing

[**P6**] Groen, D., Arabnejad, H., Jancauskas, V., Edeling, W. N., Jansson, F., Richardson, R. A., Lakhlili, J., Veen, L., Bosak, B., Kopta, P., Wright, D. W., Monnier, N., Karlshoefer, P., Suleimenova, D., Sinclair, R., Vassaux, M., Nikishova, A., Bieniek, M., Luk, O. O., Kulczewski, M., Raffin, E., Crommelin, D., Hoenen, O., Coster, D. P., Piontek, T., and Coveney, P. V. (2021a). Vecmatk: a scalable verification, validation and uncertainty quantification toolkit for scientific simulations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2197):20200221

[**P7**] Bosak, B., Kopta, P., Kulczewski, M., and Piontek, T. (2025). muqsa – an online service for uncertainty quantification and sensitivity analysis. In Paszyński, M., Barnard, A. S., and Zhang, Y. J., editors, *Computational Science – ICCS 2025 Workshops*, volume 15911 of *Lecture Notes in Computer Science*, pages 57–70. Springer, Cham

[**P8**] Wright, D. W., Richardson, R. A., Edeling, W., Lakhlili, J., Sinclair, R. C., Jancauskas, V., Suleimenova, D., Bosak, B., Kulczewski, M., Piontek, T., Kopta, P., Chirca, I., Arabnejad, H., Luk, O. O., Hoenen, O., Węglarz, J., Crommelin, D., Groen, D., and Coveney, P. V. (2020). Building confidence in simulation: Applications of easyvvuq. *Advanced Theory and Simulations*, 3(8):1900246

[**P9**] Kulczewski, M., Bosak, B., Kopta, P., Szeliga, W., and Piontek, T. (2024). Fostering uncertainty quantification in global challenges with muqsa toolkit. In Wyrzykowski, R., Dongarra, J. J., Deelman, E., and Karczewski, K., editors, *Parallel Processing and Applied Mathematics (PPAM 2024)*, volume 15581 of *Lecture Notes in Computer Science*, pages 35–46. Springer, Cham

Abstract in Polish

W rozprawie doktorskiej podsumowano wyniki badań dotyczących rozwoju metod obliczeń wieloskalowych, których efektem było opracowanie wysokopoziomowych i abstrakcyjnych modeli oraz rozwiązań systemowych wspierających projektowanie i uruchamianie symulacji wieloskalowych w wysokowydajnych środowiskach obliczeniowych (HPC, ang. High-Performance Computing). Przedstawiono innowacyjne metody wprowadzające nowe podejścia do integracji modeli obliczeniowych oraz rozwiązujące kluczowe problemy związane z ich złożonością, dokładnością oraz propagacją niepewności.

W rozprawie zaprezentowano autorskie podejścia do integracji i współdziałania modeli wieloskalowych, które stanowią jedno z kluczowych narzędzi bardzo wielu współczesnych symulacji naukowych i inżynierskich. W efekcie, zaproponowane metody dla modelowania wieloskalowego umożliwiają łączenie modeli opisujących różne skale przestrzenne i czasowe, co pozwala zwiększyć precyzję, wiarygodność oraz zakres zastosowań symulacji, a tym samym przezwyciężyć ograniczenia tradycyjnych technik modelowania i symulacji komputerowych.

Jak wykazano w rozprawie doktorskiej, złożony charakter obliczeń wieloskalowych, w tym częsta wymiana danych pomiędzy powiązanymi modelami, może znacząco wpływać na propagację niepewności i prowadzić do spadku dokładności wyników. W pracy zaproponowano oryginalne procedury weryfikacji, walidacji i kwantyfikacji niepewności (VVUQ, ang. verification, validation and uncertainty quantification), a także analizy Wrażliwości (SA, ang. sensitivity analysis), które mają na celu zwiększenie wiarygodności i powtarzalności złożonych modeli obliczeniowych. Szczególną uwagę poświęcono problemom równoległego przetwarzania i oceny modeli w rozproszonych środowiskach obliczeniowych HPC, co stanowi istotny aspekt realizacji algorytmów oceny niepewności w środowiskach obliczeń dużej skali.

Istotną część rozprawy doktorskiej stanowi również omówienie zagadnień związanych z automatyzacją i usprawnieniem realizacji złożonych przepływów obliczeniowych, w szczególności dotyczących symulacji wieloskalowych oraz procedur VVUQ. Rozprawa omawia i porównuje różne podejścia umożliwiające uproszczenie i optymalizację procesów przygotowania, uruchamiania oraz zarządzania eksperymentami obliczeniowymi wykonywanymi na zasobach obliczeniowych HPC.

Wyniki eksperymentalne badań potwierdzają postawioną hipotezę badawczą, zgodnie z którą zastosowanie strategii obliczeniowych dedykowanych modelowaniu wieloskalowemu oraz procedurom VVUQ znacząco usprawnia proces projektowania symulacji wieloskalowych, zwiększa efektywność ich wykonywania oraz wzmacnia zaufanie do wyników zaawansowanych symulacji naukowych. Opracowane metody przyczyniają się również znacząco do rozwoju nowoczesnych technik obliczeń wieloskalowych i VVUQ, poprawiając ich użyteczność, efektywność i zwiększają niezawodność w środowiskach HPC. Ponadto, wspierają one tworzenie nowej generacji systemów, narzędzi i środowisk obliczeniowych, umożliwiających dokładniejsze, skalowalne i wiarygodne modelowanie wieloskalowe w różnych dziedzinach nauki i inżynierii.

Rozprawa doktorska składa się z następujących prac:

[**P1**] Borgdorff, J., Bona-Casas, C., Mamonski, M., Kurowski, K., Piontek, T., Bosak, B., Rycerz, K., Ciepiela, E., Gubala, T., Harezlak, D., Bubak, M., Lorenz, E., and Hoekstra, A. G. (2012). A distributed multiscale computation of a tightly coupled model using the multiscale modeling language. *Procedia Computer Science*, 9:596–605. Proceedings of the International Conference on Computational Science, ICCS 2012

[**P2**] Borgdorff, J., Mamonski, M., Bosak, B., Groen, D., Belgacem, M. B., Kurowski, K., and Hoekstra, A. G. (2013b). Multiscale computing with the multiscale modeling library and runtime environment. *Procedia Computer Science*, 18:1097–1105. 2013 International Conference on Computational Science

[**P3**] Borgdorff, J., Mamonski, M., Bosak, B., Kurowski, K., Ben Belgacem, M., Chopard, B., Groen, D., Coveney, P., and Hoekstra, A. (2014). Distributed multiscale computing with muscle 2, the multiscale coupling library and environment. *Journal of Computational Science*, 5(5):719–731

[**P4**] Alowayyed, S., Piontek, T., Suter, J., Hoenen, O., Groen, D., Luk, O., Bosak, B., Kopta, P., Kurowski, K., Perks, O., Brabazon, K., Jancauskas, V., Coster, D., Coveney, P., and Hoekstra, A. (2019). Patterns for high performance multiscale computing. *Future Generation Computer Systems*, 91:335–346

[**P5**] Bosak, B., Piontek, T., Karlshoefer, P., Raffin, E., Lakhlili, J., and Kopta, P. (2021). Verification, validation and uncertainty quantification of large-scale applications with qcg-pilotjob. In Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, a. V., Dongarra, J. J., and Sloot, P. M., editors, *Computational Science – ICCS 2021*, pages 495–501, Cham. Springer International Publishing

[**P6**] Groen, D., Arabnejad, H., Jancauskas, V., Edeling, W. N., Jansson, F., Richardson, R. A., Lakhlili, J., Veen, L., Bosak, B., Kopta, P., Wright, D. W., Monnier, N., Karlshoefer, P., Suleimenova, D., Sinclair, R., Vassaux, M., Nikishova, A., Bieniek, M., Luk, O. O., Kulczewski, M., Raffin, E., Crommelin, D., Hoenen, O., Coster, D. P., Piontek, T., and Coveney, P. V. (2021a). Vecmatk: a scalable verification, validation and uncertainty quantification toolkit for scientific simulations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2197):20200221

[**P7**] Bosak, B., Kopta, P., Kulczewski, M., and Piontek, T. (2025). muqsa – an online service for uncertainty quantification and sensitivity analysis. In Paszyński, M., Barnard, A. S., and Zhang, Y. J., editors, *Computational Science – ICCS 2025 Workshops*, volume 15911 of *Lecture Notes in Computer Science*, pages 57–70. Springer, Cham

[**P8**] Wright, D. W., Richardson, R. A., Edeling, W., Lakhlili, J., Sinclair, R. C., Jancauskas, V., Suleimenova, D., Bosak, B., Kulczewski, M., Piontek, T., Kopta, P., Chirca, I., Arabnejad, H., Luk, O. O., Hoenen, O., Węglarz, J., Crommelin, D., Groen, D., and Coveney, P. V. (2020). Building confidence in simulation: Applications of easyvvuq. *Advanced Theory and Simulations*, 3(8):1900246

[**P9**] Kulczewski, M., Bosak, B., Kopta, P., Szeliga, W., and Piontek, T. (2024). Fostering uncertainty quantification in global challenges with muqsa toolkit. In Wyrzykowski, R., Dongarra, J. J., Deelman, E., and Karczewski, K., editors, *Parallel Processing and Applied Mathematics*

(PPAM 2024), volume 15581 of *Lecture Notes in Computer Science*, pages 35–46. Springer, Cham

Acknowledgements

I would like to express my deepest gratitude to all those without whom this work would never have come to fruition.

First and foremost, I wish to thank my supervisor, Dr. Krzysztof Kurowski, whose guidance, encouragement, and enthusiasm for scientific exploration have been invaluable throughout our years of collaboration. His support has shaped my research and inspired my professional growth.

My heartfelt thanks also go to my family, my loving wife Ania and my daughter Weronika, whose smiles, patience, and love have given me strength and joy along the way. I am equally grateful to my mother and late father, who always believed in me and whose unconditional love has been a constant source of comfort and motivation. I would also like to thank my brother, whose early encouragement and inspiration sparked my interest in information technology and set me on this path.

Finally, I would like to express my sincere appreciation to my colleagues from the *QCG team*, Tomasz Piontek and Piotr Kopta, with whom I had the privilege of collaborating on most of the studies presented in this work. Your expertise, dedication, and friendship have been truly invaluable – without either of you, this work would never have materialised.

Contents

Chapter 1

Introduction

In general, computer modelling and simulation refer to the process of constructing and manipulating computational, mathematical, and algorithmic representations of real-world systems or phenomena, with the purpose of performing computer simulations aimed at analysing, predicting, or optimising the behaviour of such systems. In many cases, the traditional computational modelling approach is based on the simulation of a given problem with a selected monolithic algorithm. This algorithm can effectively simulate some phenomena or processes on a bounded scale. However, **nature is observed on an abundance of scales**, and each of those scales may yield additional insight into its workings. Consequently, over the last years the multiscale modelling paradigm has gained attention in various research fields as discussed in [Groen et al., 2014], among others biomedicine [Sloot and Hoekstra, 2009], biology [Hetherington et al., 2007][Qu et al., 2011], energy [Hill, 2008][Luk et al., 2021], astrophysics [Pelupessy, F. I. et al., 2013], and earth sciences [Hill et al., 2004]. Now, with the support of modern software, scientists integrate together several models representing different space and temporal scales for a holistic analysis of real-world systems or phenomena.

The widespread interest in multiscale computing has led to the need for general computational frameworks that are able to run these types of simulations in a systematic and efficient way, as expressed by several authors [Alowayyed et al., 2019][Knap et al., 2016][Nathan Barton et al., 2007]. The particular difficulty in development of such frameworks comes from a computational point of view, since **every submodel of a multiscale model may have not only huge, but different, often even contradictory, hardware and software requirements** [Hoekstra et al., 2019]. For example, take a model with one submodel using a highly parallel fluid dynamics flow solver, requiring a powerful computing cluster with high-speed interconnects; another submodel, a cellular automaton parallelised with OpenMP, performing best on a large SMP system (symmetric multiprocessing with a single and shared main memory); and finally, a submodel using a graphics processing unit (GPU) based powered agent-based modelling toolkit. Therefore, a key challenge is how to integrate these models, given that they require heterogeneous computational resources up to exascale performance, exhibit different execution times, and often operate within complex and dynamic workflows involving loops of varying intensity.

Furthermore, the **complexity of scientific computing increases radically when the quality of the model and the credibility of the results are essential** [Council, 2012]. In such cases, in order to evaluate the uncertainty of a model and to discover essential statistics, which can further help in model optimisation, many model executions need to be run. The available uncertainty quantification methods, like Monte Carlo, Stochastic Collocation of Polynomial Chaos Expansion, even when applied to single-scale monolithic codes, may consume enormous amounts of

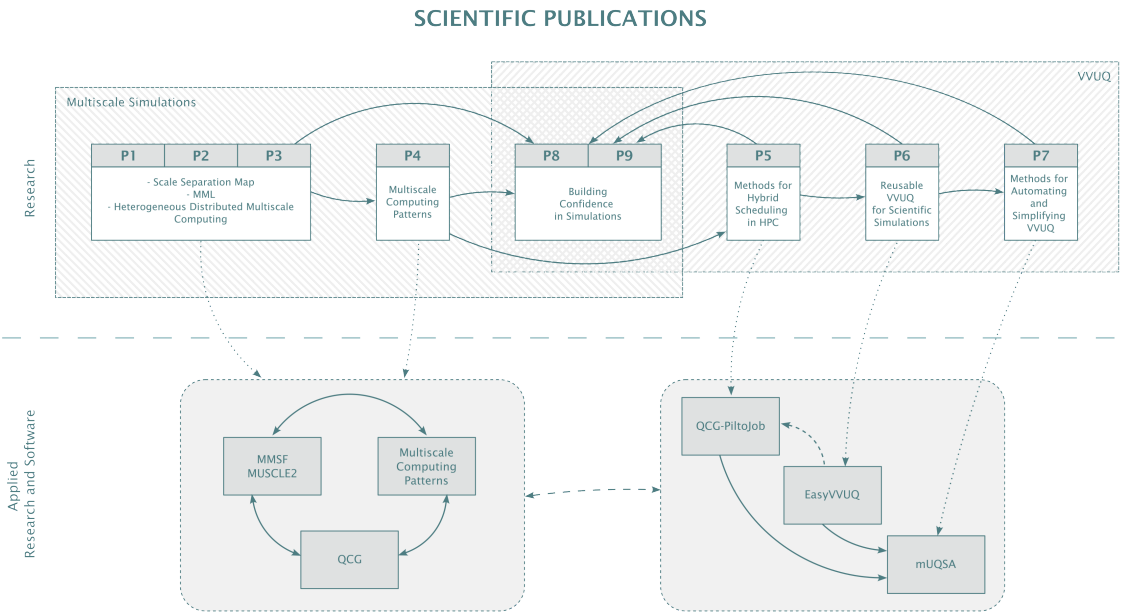


FIGURE 1.1: Two principal research areas are addressed in the dissertation: (1) fundamental scientific research and (2) applied research involving software development, along with an analysis of their interrelations. References P1–P9 correspond to the author's publications cited and discussed within the dissertation.

computing resources and pose significant challenges for scheduling and resource management that should support efficient and straightforward execution of such advanced computations. When a model is built from multiple single-scale models to constitute a multiscale application, the challenges are naturally more evident, as presented in [Ye et al., 2022].

In recent years, collaborative work within several innovative research projects involving the author of this dissertation has led to the development of methodologies and software components that formalise multiscale computations and their **verification, validation, and uncertainty quantification (VVUQ)**, while making them more efficient and accessible to academic and industrial users. The author's contributions in this field are mainly related to the **design and application of advanced automation and computational strategies** in both areas. The visual overview of the **scientific and applied research** conducted, which has been materialised through a set of selected author's publications and software components for this dissertation, is presented in Figure 1.1.

In the context of the first area - that is scientific research around multiscale modelling and computations - the dissertation presents the path that multiscale computing approach has travelled over the past dozen or so years. In principle, it has started with the formalisation of the basic requirements through the conception of **Multiscale Modeling Language (MML)** [Borgdorff et al., 2011], followed by research and development work to design and implement a general **Multiscale Modeling and Simulation Framework** [Borgdorff et al., 2012][**?**][Borgdorff et al., 2014], and then adapting this framework to efficiently support the most popular execution scenarios through the **Multiscale Computing Patterns (MCP)**[Alowayyed et al., 2019]. The particular focus in this dissertation has been on the description of novel contributions to methodology and tools supporting the execution of multiscale simulations on heterogeneous and distributed computing resources.

In relation to the second applied research and software area, the dissertation discusses the problems of **VVUQ for complex computational simulations**. Since many fields of science and engineering, ranging from fusion energy, biomedicine, to sociological domains, such as human

migration [Groen et al., 2021a], use or plan to use multiscale computational models for simulations of critical importance, VVUQ methods are essential and should be incorporated into the process of development of such models to make them fully reliable and actionable. However, the application of VVUQ, especially for multiscale models, is not a trivial task and needs to be guided by a well-defined methodology and supported by specialised software. This work showcases the author's advancements in streamlining uncertainty quantification and sensitivity analysis - core components of VVUQ - through the development of reusable **Uncertainty Quantification Patterns (UQPs)** [Groen et al., 2021a], innovative algorithms such as those leveraging second-level scheduling [Bosak et al., 2021], and robust automation techniques that significantly improve accessibility while preserving high computational efficiency [Bosak et al., 2025].

Finally, the **dissertation highlights the synergy between the developments for multiscale and for VVUQ** by demonstrating how real-life applications have leveraged the underlying both scientific concepts and software [Wright et al., 2020][Kulczewski et al., 2024]. In addition, it contrasts the motivations derived from these real-life use cases with the results of performance and scalability evaluations of key software components, conducted through many experimental studies on dedicated testbeds and fully operational **high-performance computing (HPC) infrastructures**.

1.1 Thesis context

Multiscale modelling, as a field concerned with solving problems that exhibit important features across multiple spatial and temporal scales, represents a natural evolution of computational science. Researchers have discovered that classical monolithic models, which accurately reflect reality within a specific time or spatial scale, become even more useful when integrated with other modules operating at different scales. With the multiscale approach, the model is not limited to some fine-grained or coarse-grained reality but can efficiently cover both, and possibly many in between [Hoekstra et al., 2014].

Early works related to the topic date back to the 1970s, when the idea of micro-macro coupling through homogenisation was proposed in the context of applied mathematics [Bensoussan et al., 1978]. The evolution of numerical algorithms combined with the growth of computing power, occurring over next twenty years, resulted in the development of computational multiscale methods capable to simulate real-life problems, first domain-specific ones, like the Quasicontinuum method [Tadmor et al., 1996] which combined atomistic simulations with continuum mechanics, and then generic, thus applicable to many domains, like Heteregoneous Multiscale Method (HMM) [E et al., 2003]. Multiscale coupling has proven to be a practical and effective approach and is increasingly being incorporated into widely used simulation packages such as GROMACS [Abraham et al., 2024], LAMMPS [Thompson et al., 2022], and NAMD [Phillips et al., 2020].

Meanwhile, newly developed software tools and techniques, such as the Cactus Framework [Goodale et al., 2003], the OpenMI API specification [Gregersen et al., 2007], and the QosCosGrid-MPI library [Agullo et al., 2011], have introduced new approaches to simplifying the development of complex computational scenarios and optimising their execution. These application-driven tools demonstrate the feasibility of implementing multicomponent, heterogeneous, and distributed computing for sophisticated simulation workflows and complex computational tasks.

With these new opportunities and the growing number of multiscale problems, researchers initiated a discussion on the formalisation of multiscale computing, which resulted in the creation of the **Complex Automata (CxA)** paradigm along with the concept of **Scale Separation Map** [Hoekstra et al., 2007]. Soon, these concepts evolved into a more flexible and machine-

readable **Multiscale Modelling Language (MML)** with one of the main objectives to enable the development of general-purpose software tools and services for modelling and automated execution of multiscale models on large-scale computing resources [Falcone et al., 2010]. This goal was ultimately realised through the **Multiscale Modelling and Simulation Framework (MMSF)**, established across two successive European projects: MAPPER [Belgacem et al., 2013] and ComPat [Alowayyed et al., 2018]. MAPPER laid the groundwork by introducing the initial version of MMSF, including the Multiscale Coupling Library and Environment (MUSCLE2) [Borgdorff et al., 2014] and the QCG services [Bosak et al., 2012], which together allowed distributed and heterogeneous computations consistent with MML specifications. Building on this foundation, the ComPat project further extended both the conceptual foundations and the software infrastructure, enhancing MMSF to effectively support key **Multiscale Computing Patterns** [Alowayyed et al., 2019]. This journey continues, as exemplified by the implementation of the next MUSCLE incarnation - MUSCLE3 [Veen and Hoekstra, 2020].

In parallel to the evolution of multiscale modelling, other key branches of computational science have iteratively progressed, namely **verification, validation and uncertainty quantification**, which are now commonly referenced together as **VVUQ** to emphasise their close relationship. The importance of software testing, covering both verification and validation, has been recognised since the early days of computational experimentation [Greenfeld, 1967]. However, its widespread adoption and formalisation gained momentum during the 1970s and 1980s, marked by the emergence of foundational definitions [Schlesinger, 1979] and influential publications underscoring its critical role ([Boehm, 1976], [Boehm, 1979], [Rook, 1986]). This period also saw the initiation of standardisation efforts by various organisations, including the development of the IEEE Standard for Software verification and validation Plans [IEEE, 1986].

Starting from the 1990s, the scientific community faced an insufficiency of the deterministic assessment of models' enabled by classical verification and validation and started to look towards the methods enabling quantification of the model's confidence taking into account all sources of uncertainty. Efforts were directed toward methodological integration of uncertainties into the verification and validation processes through means of validated statistical methods, such as Monte Carlo [Oberkampf et al., 2002] [Oberkampf, 2003], Polynomial Chaos Expansion or Stochastic Collocation [Eldred and Burkardt, 2009]. More attention has also been paid to sensitivity analysis (SA) as it provides highly valuable information on the relative influence of input uncertainty on output within the broader uncertainty quantification (UQ) framework [Saltelli et al., 2008], and automated construction of surrogate models with the help of UQ methods [Smith, 2013]. As the consolidation of the conceptual framework progressed, computational scientists began developing software to help model developers or users incorporate UQ into their computational scenarios. Several libraries and toolkits have emerged, and some of them have gained widespread popularity, such as Chaospy [Feinberg and Langtangen, 2015], Dakota [Adams et al., 2024].

However, it is important to emphasise that UQ is typically computationally intensive, as it requires numerous evaluations of an underlying model. OpenTURNS [Baudin et al., 2015] and Uranie [Blanchard, Jean-Baptiste et al., 2019] are examples of tools developed with such demands in mind, and provide support for parallel executions of UQ evaluations. For large-scale or complex systems, such as multiscale simulations, the associated computational requirements can become substantially greater. This difficulty is further compounded by the technical complexity of managing and executing the required computations efficiently. Consequently, **automation of UQ workflows has been recognised as a critical need for the successful integration of UQ into advanced simulation models** and has been addressed through the work initiated in the VECMA project [Groen et al., 2021b] focused on VVUQ of multiscale simulations. These works have continued over

the next years in the scope of SEAVEA [SEAVEA Project, 2021], PIONIER-LAB [PIONIER-LAB Project, 2022], and HiDALGO2 [HiDALGO2, 2023] projects.

1.2 Contributions

The principal hypothesis advanced in this dissertation contends that

Applying computational strategies for multiscale modelling and VVUQ streamlines development, enhances execution efficiency, and fosters trust in advanced scientific applications.

Therefore, this dissertation presents the author's contributions to methodologies for modelling and assessing confidence in heterogeneous multiscale simulations that depend on large-scale computational infrastructures. It outlines new methods developed for flexible, efficient and intuitive execution of complex computations, such as multiscale simulations and UQ processes, on large-scale computing resources, HPC environments and systems in particular.

The author's contributions can be summarised and categorised into three main thematic areas covering the **development**, **execution**, and **final validation of computational multiscale models**, reflecting in fact the typical lifecycle of computational science workflows, namely:

Streamlining Multiscale Model Development - this area addresses the challenge of streamlining the development process by enabling multiscale applications to be composed automatically from heterogeneous, interoperable modules. The proposed computational strategies, patterns and software frameworks reduce the complexity of integrating diverse submodels, while providing built-in automation that supports reproducible and user-friendly execution on large-scale HPC infrastructures. Key author's contributions include:

- Formulation of novel access and execution solutions for large-scale, distributed computational experiments on heterogeneous HPC resources.
- Development of QCG services and tools that enable seamless, automated deployment and execution of multiscale simulations across distributed computing infrastructures.
- Built-in automation mechanisms simplifying the preparation, submission, and management of computational workflows.
- Design and implementation of the mUQSA service, providing an intuitive web-based environment for automated uncertainty quantification (UQ) and sensitivity analysis (SA) without the need for command-line user interaction.

Enhancing Execution Efficiency of Multiscale Models - the author's research introduces computational strategies that enhance execution efficiency through coordinated use of distributed and heterogeneous computing resources. By leveraging the Pilot Job paradigm, hierarchical scheduling, and pattern-based approaches, the developed methods improve the overall scalability and resource utilisation of multiscale workflows. These advances bridge the gap between theoretical multiscale concepts and practical, high-performance implementations.

- Design and implementation of distributed execution strategies for heterogeneous multiscale simulations, effectively translating the Multiscale Modeling Language into operational, large-scale computing environments.

- Contributions to the development of the MUSCLE2 framework, with emphasis on methods for efficient cross-cluster coupling and model coordination.
- Adoption of Multiscale Computing Patterns to optimise different execution scenarios and improve computational efficiency.
- Development of innovative approaches to hierarchical scheduling, improving resource utilisation and execution efficiency for complex multiscale simulations using the QCG-PilotJob service to manage dynamic workloads in multiscale and VVUQ contexts.
- Design and implementation of integration mechanisms to support the efficient and scalable execution of EasyVVUQ-defined VVUQ procedures using QCG-PilotJob on large-scale computing infrastructures.
- Performance evaluation and scalability analysis of key framework components, in particular MUSCLE2, and QCG in representative scientific multiscale applications.

Fostering Trust in Advanced Scientific Applications - this area focuses on improving the usability and accessibility of these methods while maintaining scientific rigour and computational efficiency. The tools developed bridge the gap between the theoretical validation frameworks and their practical deployment in large-scale HPC environments.

- Integration of VVUQ methodologies into automated workflows and frameworks, ensuring traceability and scientific reproducibility.
- The usefulness of delivering a VVUQ platform via a web-based portal was analysed and demonstrated. A novel method was designed and implemented to enable intuitive, automated execution of uncertainty quantification (UQ) and sensitivity analysis (SA) studies, eliminating the additional complexity typically required when using command-line tools or stand-alone libraries. This advancement is exemplified by the development of the mUQSA service.
- Demonstration of improved reliability and reproducibility of simulation results through real-world case studies in biomedicine, fusion research, epidemiology, and environmental science.
- Conceptual contribution aligning scientific credibility of VVUQ with computational efficiency and usability to foster broader adoption of trustworthy simulation practices.

1.3 Structure of this work

This dissertation is structured as follows.

Chapter 1 provides an overview of the overarching research topics, emphasising a novel distributed multiscale computing paradigm in conjunction with VVUQ strategies. It elaborates on the underlying motivation, addresses the principal scientific and technical challenges, defines the principal hypothesis, and presents the main contributions to the conceptual and methodological frameworks developed in this work.

Chapter 2 sets the stage with a concise historical review of the development of multiscale computing and VVUQ, drawing on key thematic publications that have shaped the field over the past few decades. It provides the theoretical background necessary to understand multiscale

computing and VVUQ in depth, highlighting how the author's work contributes to a consistent methodology for modelling and evaluating the confidence of heterogeneous multiscale simulations executed on HPC systems.

Chapter 3 transitions from theory to applied research, detailing the implementation of the proposed methods through the development of dedicated software components, evaluating their performance and practical usability through real-world applications, and discussing the author's efforts to promote and spread the underlying concepts.

Chapter 4 concludes the dissertation by summarising the research findings and providing a final discussion. Highlights the remaining challenges and outlines prospective directions towards developing a fully integrated system for constructing and executing validated, verified, and optimised multiscale simulations with quantified uncertainty.

In addition, the dissertation highlights the active role of the author in the popularisation of multiscale concepts and VVUQ and the dissemination of the developed methods and solutions. These efforts include the co-organisation of the Multiscale Modelling and Simulation (MMS) workshop series, collocated with the International Conference on Computational Science (ICCS), participating in numerous thematic conferences, and engaging in close collaboration with the broader scientific community at both national and international levels.

Chapter 2

Methodology

Multiscale simulations are already an essential computational method in a variety of research disciplines and provide unprecedented levels of scientific insight at a tractable cost in terms of effort and computing resources. In many applications such simulations are expected to produce results that are robust and actionable, so they can impact decision making outside of basic science, for example in industrial or clinical settings [Groen, 2019].

This chapter introduces a methodology developed to streamline the processes of modelling, execution, and credibility assessment of multiscale models. Particular emphasis is placed on novel methods designed to enable automated and efficient processing of multiscale and uncertainty quantification workflows on large-scale and HPC environments, an area that has been a primary focus of the author throughout this dissertation.

2.1 Multiscale computing

Scientific computation has been a driving force behind the development of computers since the 1940s. However, for many decades, computational models were largely monolithic, each designed to represent specific physical phenomena. For example, molecular dynamics (MD) enabled the simulation of material behaviour at the atomic scale, which allowed the optimisation of its internal properties, while the finite element method (FEM) allowed for the analysis of larger structures at the continuum scale, taking the static information about the material properties as input. These models were distinct and were typically executed as independent computing simulations. **Multiscale simulations emerged from recognition that many physical processes often and naturally span multiple spatial and temporal scales simultaneously.** For example, in a FEM simulation of an aircraft wing, material properties such as elastic moduli or strength thresholds [Mieczkowski et al., 2025] may need to be dynamically computed using lower-scale MD simulations, especially under extreme or localised loading conditions. This hierarchical coupling of models can be extended in both directions: ab initio simulations at smaller scales can inform MD, while larger-scale models, such as computational fluid dynamics (CFD), can be integrated to simulate airflow over the entire aircraft. Such a multiscale model allows for more comprehensive and physically consistent optimisation of aircraft design. Moreover, multiscale simulations are a practical approach to overcome computing scalability issues. Basically, if we would like to compute the full domain of a given problem using a monolithic algorithm, it is often either too resource-demanding or lacks the required precision. With the multiscale paradigm, it can be decided which part of the problem needs precise computations with a fine-grained algorithm, and which does not and can be calculated with a coarse-grained method. In this way, **a large monolithic problem can**

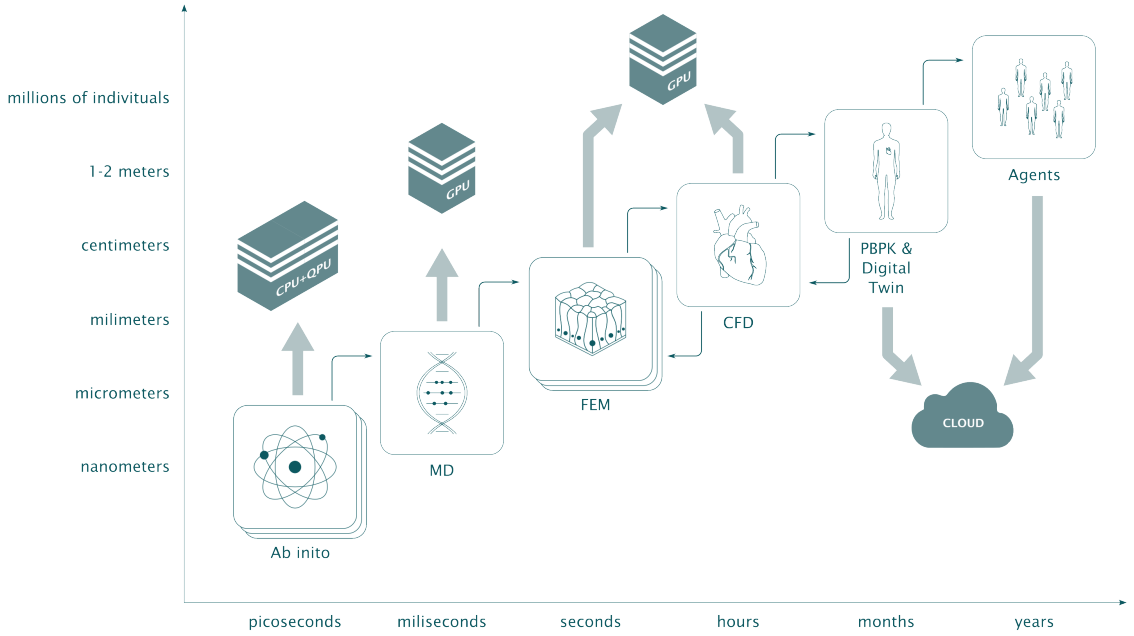


FIGURE 2.1: Conceptual overview of building a multiscale application from single-scale models that reflect phenomena at some space and temporal scales. Each single-scale model can be developed independently and may have distinct resource requirements. The models can be coupled uni-directionally and bi-directionally, as well as there may be multiple instances of a single model, e.g. to enable parameter exploration or uncertainty quantification.

be decomposed into a set of smaller, more manageable subproblems, leading to a more efficient usage of computational resources. Finally, as presented in Figure 2.1, since multiscale applications are intrinsically composed of different submodels, each submodel can be implemented using a different codebase and, to a large extent, executed independently, even on separate and heterogeneous computational resources.

This modularity allows for independent development of individual submodels and their optimisation towards specific architecture (e.g. CPU vs GPU) as well as enables high-flexibility in the construction of a multiscale model in a lego-like fashion. The role of an execution middleware is to synchronise the execution of such complex models on heterogeneous computing environments ensuring good performance of the overall scenario.

The following part of this section discusses the comprehensive methodology constructed for modelling and execution of demanding multiscale applications, while the aspect of newly created methods for distributed and heterogeneous multiscale computing, which being one of the topics of this dissertation, is explored in detail. The development of the demonstrated multiscale computing methodology was an incremental process, with each successive stage building on and improving the outcomes of the previous ones. As a result, several key generalisable concepts related to multiscale simulations have emerged, forming a solid foundation for the design and execution of new multiscale application scenarios. This progression is illustrated in Figure 2.2, which presents a UML-like diagram where each successive concept inherits from those established earlier. In the following sections, each concept will be examined in detail.

2.1.1 Scale Separation Map (SSM)

The concept of **Scale Separation Map (SSM)** emerged with the development of the *complex automata (CxA)* paradigm [Hoekstra et al., 2007]. For the purposes of this discussion, it is sufficient to understand that CxA represents an early paradigm proposed for integrating single-scale models (also known as submodels) to construct multiscale simulations. It was later extended to the more

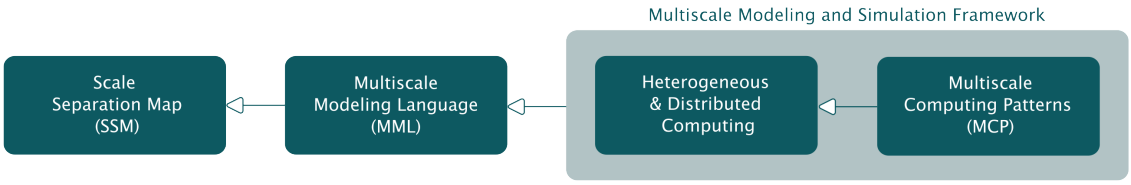


FIGURE 2.2: High-level view on the development process of the presented multiscale computing methodology. The arrows connecting the concepts represent inheritance relationships. Consequently, MML extends SSM, while MCP emerges as the most advanced concept, inheriting from all preceding ones.

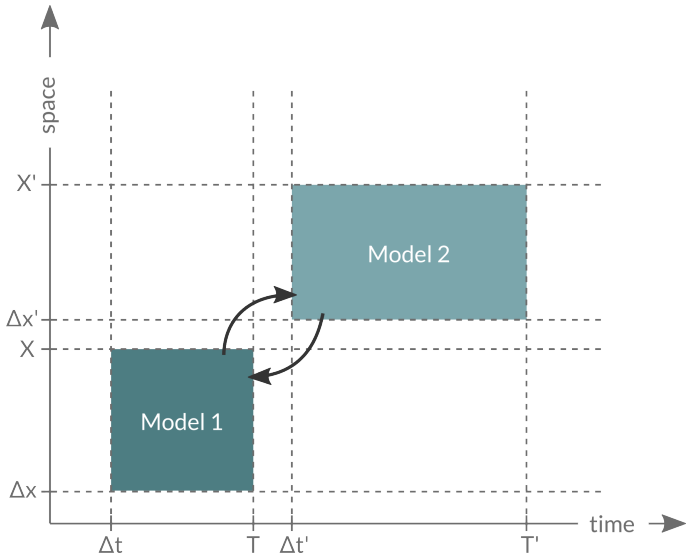


FIGURE 2.3: Simple Scale Separation Map with two submodels with disjoint scales. This represents a classical micro-macro coupling, with fast processes on the micro scale and slow processes on the macroscale. In more complex scenarios the scales may overlap and require specific scale mapping strategies.

general and flexible Multiscale Modeling Language (MML), which will be described in the following section.

The definition of coupling between submodels in CxA, which describes the way these submodels interact, required information about the temporal and spatial scales that the submodels capture. Scientists observed that each submodel operates over a domain characterised by specific **spatial and temporal extents**, as well as **spatial and temporal resolutions**. To better understand the relationships between these submodels, they proposed the Scale Separation Map, a visual representation that helps identify overlaps and gaps across scales, thereby informing appropriate submodel integration strategies. An example Scale Separation Map is shown in Figure 2.3. The horizontal axis represents the temporal scale, while the vertical axis represents the spatial scale. Each box represents a single submodel in the following way: the right and top sides of a box (capital letters) inform about the temporal and spatial lengths of the domain simulated by a given submodel, while the left and bottom sides (deltas) show the temporal and spatial resolution of a submodel's operation. The arrows between the boxes are not essential at this step, but may provide information about the nature of coupling (e.g. direction of communication). Consequently, **based on the mutual positions and connections between submodels, it is possible to infer what type of coupling needs to be implemented between them**.

2.1.2 Multiscale Modeling Language (MML)

The high-level description of a multiscale model provided by the Scale Separation Map is the starting point to understand the nature of couplings between submodels. However, in order

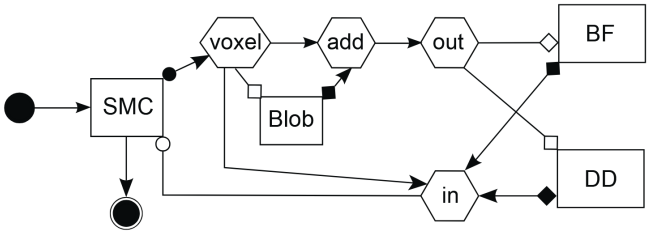


FIGURE 2.4: An example multiscale application coupling scheme described in gMML [Borgdorff et al., 2012]

to specify details of such a model, particularly to enable its automatic execution, a bunch of other information is needed. For example, SSM does not explicitly specify whether the model contains cycles or not; whether it has a fixed number of message exchanges or not, or a fixed number of synchronisation points; and whether there is more than one instance per submodel and whether that amount is fixed. Finally, it does not provide technical details on the model itself: its binary location or resource requirements. To address all these challenges, and to bridge the gap between multiscale modelers and execution environments, the **Multiscale Modeling Language (MML)** was conceived[Borgdorff et al., 2013a]. The MML language introduces a well-defined multiscale modelling terminology that **can be used to describe, verify, analyse, and execute a multiscale model**. What is crucial is that **MML operates on a multiscale level**, above single-scale models; thus it enables independent development of submodels.

MML defines how a particular multiscale model is coupled by explicitly creating a directed graph, typically transformed into a Directed Acyclic Graph (DAG) for execution, with submodel instances as nodes, couplings as edges, and the number of times that the coupling is invoked as edge weights. It also introduces two special components, namely filters and mappers, which, through user-defined data transformations, enable bridging submodels across scales.

The MML specification, as described in [Borgdorff et al., 2012], has two main forms, **xMML** and **gMML.** The gMML is a graphical representation that features the elements listed above and resembles UML diagrams, as shown in Figure 2.4, but does not contain any implementation details or information about scales. It is useful for composing or communicating the architecture of a multiscale model. For machine interpretation, the proposed xMML format captures not only these features but also a wide range of metadata. This includes scale information (inherited from SSM), possible parameter settings, a datatype system, binding ports of submodels and mappers, implementation details such as the number of cores needed per submodel, and also descriptive and documentation facilities. In contrast to gMML, **xMML can be automatically processed and acts as an exchange format for a model**.

Explicitly or implicitly (through deduction), xMML brings essential information required for the model optimisation and its execution, which may be consumed by a specific multiscale execution framework. From this perspective, there are several key execution schemes, outlined in [Borgdorff et al., 2012], that MML defines and that need to be supported by the software implementing the MML specification, namely:

Loosely-coupled and tightly-coupled models. When a model is tightly-coupled, it means that there is a feedback loop (a cycle) within the model and that certain submodels will be revisited. In a loosely coupled model, without a cycle, a submodel can be considered finished when it has sent its information. In other words, a loosely coupled model can be implemented as a DAG workflow. However, in a tightly coupled model, the execution software will need to

keep some submodels waiting while others compute. This is naturally more challenging for both scheduling and execution.

Static or dynamic number of submodel instances. If a number of submodel instances is dynamic, the selected execution software should be able to create new instances on the fly. This is in practice a challenging problem, as it may require new and dynamic allocations of computing resources. In case of a static number of submodels, the required resources can be allocated or reserved in advance.

Static or dynamic number of synchronisation points. When a number of synchronisation points is dynamic, the execution software must be able to handle variable lifetimes of submodels and irregular patterns of message exchange between them. This introduces significant challenges in scheduling and, if not managed carefully, can lead to inefficiencies or performance bottlenecks. In contrast, a static or regular synchronisation pattern allows for predefined and more optimised scheduling, enabling better resource planning and more predictable execution behaviour.

Homogeneous or heterogeneous resource requirements. Although MML largely abstracts away the internal implementation details of the submodels, it allows the specification of certain key information, such as resource requirements. This information helps the execution software determine whether the simulation can run on homogeneous computing resources (e.g., same type of CPU/GPU nodes) or whether it demands heterogeneous resources (e.g., a mix of CPUs, GPUs, or highly specialised hardware) and potentially distributed execution across different HPC systems or computing clusters.

The extensive range of execution scenarios supported by MML provides substantial flexibility for multiscale application developers. However, this flexibility also introduces considerable challenges in both the design and execution of multiscale applications, thereby necessitating dedicated software support. These challenges are particularly pronounced at the execution layer, where models often exhibit complex coupling topologies and comprise heterogeneous submodels with diverse and computationally demanding resource requirements. Therefore, **efficient scheduling and execution of such multiscale models require a robust and adaptable execution environment**.

In the following, this dissertation presents the Multiscale Modeling and Simulation Framework (MMSF) as a reference implementation that supports both the modelling and execution of MML-based multiscale applications, and was successfully used in experimental studies.

2.1.3 Multiscale Modeling and Simulation Framework (MMSF)

The Multiscale Modeling and Simulation Framework (MMSF) has been developed and presented in [Belgacem et al., 2013] and in [Alowayyed et al., 2018] - as a **reference implementation supporting both the definition and execution of MML-based multiscale models**. This section presents the conceptual foundations of MMSF, while the detailed technical description of its individual components is provided in the next chapter.

Basics of MMSF

The core part of MMSF, formulated in [Borgdorff et al., 2014], provided a basic theoretical and methodological framework to construct multiscale simulations in the following four main stages described in [Alowayyed et al., 2019], which are also visually depicted in Figure 2.5:

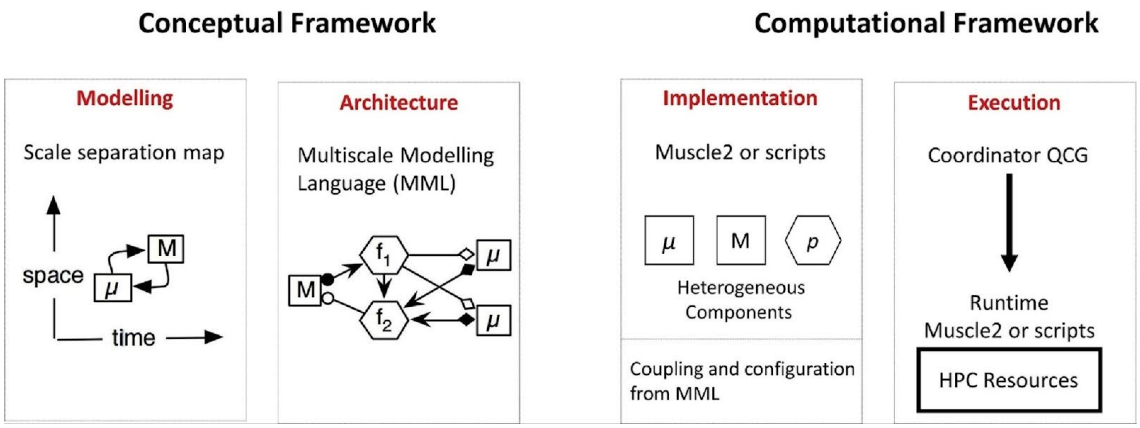


FIGURE 2.5: The fourth main stages of the multiscale model lifetime in the MMSF framework: construction of Scale Separation Map, definition of MML description, implementation of a multiscale model and coordinated execution of a model on HPC resources [Alowayyed et al., 2017]

- First, we model multiscale phenomena as collections of single-scale submodels, then decide on which models interact with each other and how. Single-scale models and couplings are presented within a Scale Separation Map, allowing us to describe and compare multiscale models on a conceptual level.
- Second, we specify the architecture of a multiscale application by detailing single-scale models, their couplings, and interactions using MML.
- Third, we convert these definitions to a fully implemented multiscale model, currently based on *MUSCLE2* [Borgdorff et al., 2013b] (although the concepts of MMSF can also be applied to other coupling environments such as AMUSE [Zwart et al., 2013]).
- Fourth, we deploy and execute the multiscale application on a set of computational resources. Developers and users can run different submodels on different machines, using, for example, *QCG* middleware [Piontek et al., 2016]. This paradigm is known as *Distributed Multiscale Computing* and was introduced in [Borgdorff, 2014].

To validate the theoretical foundations of the MMSF framework, it was necessary to translate these concepts into a concrete system architecture, whose high-level overview is illustrated in Figure 2.6. Given diverse requirements of communities engaged in multiscale simulations, including various types of multiscale couplings and a wide range of underlying computing resources, maintaining modularity has been a key design priority.

MMSF can also be seen from different perspectives: how it supports multiscale application developers, how it supports multiscale application users, and how it automates execution of multiscale applications in a distributed computing environment. These three, partially distinct, operational domains of the MMSF framework can be further characterised as follows:

Support for multiscale application developers

MMSF implements MML conceptions to enable a consistent description of a multiscale model. In order to simplify development of MML description, it provides a dedicated graphical tool (*Multiscale Application Designer - MAD*, being a reference implementation), which allows building models in accordance with gMML, as well as validating and translating them to the xMML format, or execution-engine specific descriptions (like CxA files for MUSCLE2). It is supported by persistent storage of MML descriptions, so once created, the definitions can be reused [Borgdorff et al., 2012].

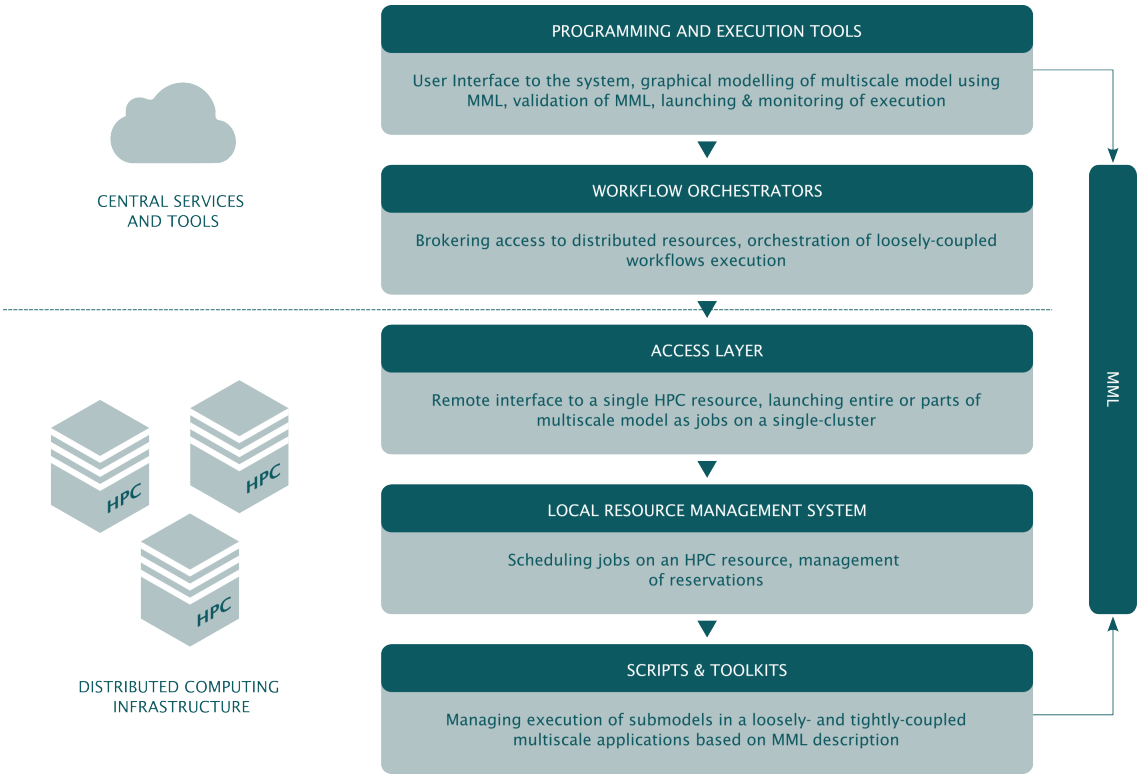


FIGURE 2.6: High-level architecture of the MMSF framework created in the MAPPER project

Support for multiscale application users

Once the model is constructed, the users of the MMSF framework, depending on their preferences and application specificity, may use different interfaces to launch, monitor and manage execution of a model. For fully automated executions of well-defined mature models, they can make use of high-level graphical interfaces like *GridSpace2* [Borgdorff et al., 2012][Ciepiela et al., 2010][Nowakowski et al., 2011] or later developed *QCG-Portal* or *QCG-Now* [Piontek et al., 2016]. In other cases, they may prefer to employ command-line tools such as *QCG-Client* [Radecki, 2014][Bosak et al., 2014].

Execution automation

As discussed in the previous chapter, executing a complex multiscale model remains one of the most significant challenges, particularly when the application demands heterogeneous and distributed computing resources. MMSF addresses this issue employing middleware systems, in particular *QCG* [Borgdorff et al., 2012], which facilitate the integration of computing resources and manage the scheduling of jobs. In practice, while loosely coupled workflows are relatively straightforward to execute, tightly coupled scenarios require specialised mechanisms to enable the co-allocation of computing resources across multiple HPC systems and clusters.

2.1.4 Distributed & Heterogeneous Multiscale Computing

Distributed multiscale computing, discussed in [Borgdorff et al., 2014], represents an approach designed to efficiently exploit limited and often heterogeneous computational resources in the execution of demanding multiscale models. The motivations for distributing multiscale computations are manifold, including the need to access computational resources beyond those available at a single site; to leverage heterogeneous infrastructures comprising clusters equipped with more powerful GPUs [Ciżnicki et al., 2012], high-throughput I/O systems, tightly interconnected CPUs; and to

use specialised local software licences in conjunction with large-scale parallel executions on HPC systems. **Execution of distributed computations may be challenging in many layers, from procedural to purely technical** [Ben Belgacem et al., 2012]. In particular, it is especially complicated to run a tightly-coupled multiscale application in a cross-cluster environment. It requires addressing the following issues: co-allocation of heterogeneous resources; coordination of spawning application processes at multiple sites; and finally, enabling communication between firewalled and NAT-ed systems. These issues required the development of new computational methods and deserve a more detailed description.

Co-allocation of heterogeneous resources

Modern HPC systems are typically managed by batch schedulers (e.g. SLURM, PBS), where users submit jobs along with their resource requirements. These jobs are queued and launched once requested resources are available and scheduling policies are met, preventing oversubscription. In distributed multiscale applications, where components run on different computing resources, each submodel is submitted as a separate job. All of these jobs must start simultaneously or in coordination, which is challenging because the exact start times are unknown at the time of submission. A common approach to this issue is **resource co-allocation, achieved through advance reservation**, a mechanism for booking computing resources in advance, either manually or automatically (e.g., via services like *QCG-Computing BES/AR*). This coordination, as shown in [Borgdorff et al., 2014], can be managed by a metascheduler (e.g. *QCG-Broker*), which aligns availability across multiple systems. Users specify a maximum runtime and a flexible time window; the system attempts to find a matching slot and reserve resources. If any part of the reservation fails, the process is cancelled and repeated, as discussed in [Borgdorff et al., 2012].

Coordination of application spawning

In most parallel toolkits used within a single cluster, there is a master process that spawns worker processes using either Secure Shell Protocol (SSH) or local resource management system (LRMS) native interfaces. This makes the task of exchanging contact information (e.g. listening host and port) between master and workers relatively easy, as the master is always initialised before the workers. With a co-allocated distributed application, the master and workers are started independently, and exchanging information is less trivial. A common solution, which was also employed in MMSF as described in [Borgdorff et al., 2012], is to introduce an external **coordination service**, where the master can publish its contact information and workers can retrieve it as needed. This approach removes the strict requirement for components to start in a predefined order.

Cross-cluster communication

Many HPC systems employ private IP addressing schemes for compute nodes, which complicates direct access to running processes without additional network configuration. In addition, certain sites impose restrictions on the traffic from the outbound network. To enable the execution of tightly coupled multiscale applications across such isolated environments, multiscale frameworks must therefore be explicitly adapted to these networking constraints. One solution proposed in [Borgdorff et al., 2012] for the MUSCLE2 library, aimed at enabling its operation over firewalls and NAT systems, involved limiting the ports that MUSCLE2 processes could use to a predefined port range. Then, to enable cross-cluster communication of submodel processes, a user-space relay daemon, *MUSCLE Transport Overlay - MTO*, was introduced, as described in [Borgdorff et al., 2012] and [Borgdorff et al., 2013b]. This daemon needs to be deployed on each cluster and acts as

a gateway between internal worker nodes and external connections. It must either be reachable from other daemons or be able to initiate outgoing connections to them.

Note: In practice, technical limitations and operational priorities of HPC systems often result in restrictions on the usage of advance reservation mechanisms. Fully dynamic creation of co-allocation is therefore not feasible. Fortunately, for many multiscale models, these limitations can be mitigated by requesting a small but long reservation just for the low-cost model, while the high-cost model may be scheduled dynamically. These days, the execution of low-cost submodels with limited message exchange can be effectively supported also using virtual machines or container infrastructures.

2.1.5 High-performance Multiscale Computing Patterns

Efficient execution of multiscale applications on HPC systems requires addressing several key challenges, including load balancing (i.e., ensuring adequate resource allocation across single-scale models), fault tolerance (to mitigate potential failures of individual model instances), and energy awareness (optimising energy consumption based on the characteristics of single-scale models, potentially in conjunction with load-balancing strategies). These challenges should be managed in a generic and systematic manner, thus minimising the burden on multiscale application developers. Developers should primarily focus on the implementation of scale-bridging mechanisms and the optimisation of individual single-scale models, while the complexities associated with efficient execution on HPC systems should be abstracted and managed through an additional generic execution layer.

Therefore, as explained in [Alowayyed et al., 2019], MMSF has been extended with a further layer of **Multiscale Computing Patterns (MCPs)** that address specific challenges of execution of the most common multiscale models on High-Performance Computing (HPC) resources. Through *high-level call sequences that exploit the functional decomposition of multiscale models in terms of single-scale models* [Alowayyed et al., 2017], MCPs allow for effective mapping of multiscale applications to common execution patterns and ensure the seamless and efficient execution of such applications on available infrastructure.

Three patterns have been identified, namely: **Extreme Scaling**, **Heterogeneous Multiscale Computing** and **Replica Computing**, and have the following characteristic:

The **Extreme Scaling (ES)** pattern represents a type of multiscale model where one primary single-scale model is coupled to a set of serial and/or parallel auxiliary models on any scale. The primary model is computationally-intensive, energy-consuming, and highly scalable, whereas the auxiliary models are not. As a result, the overall efficiency depends both on the internal performance of the primary model, which is beyond the scope of this discussion, and on how it interacts with the auxiliary components. Serial auxiliary models, in particular, can lead to significant under-utilisation of resources, either because they cannot scale or because they cause the primary model to wait. Addressing such imbalances is a central objective of the ES pattern, which tries to minimise interference between the primary and auxiliary models. This can be achieved through careful load balancing and by optimising communication between primary and auxiliary models.

The **Heterogeneous Multiscale Computing (HMC)** pattern describes a typical macro-micro coupling, where a macroscale solver requires input from multiple microscale model instances - for example, to compute spatially varying quantities like a constitutive equation (e.g. viscosity in a flow problem). The number of microscale models is dynamic and depends on the evolution of the macro-model. To coordinate this process, the HMC pattern introduces a dedicated software component: the HMC manager. The manager's role is to launch the necessary microscale models

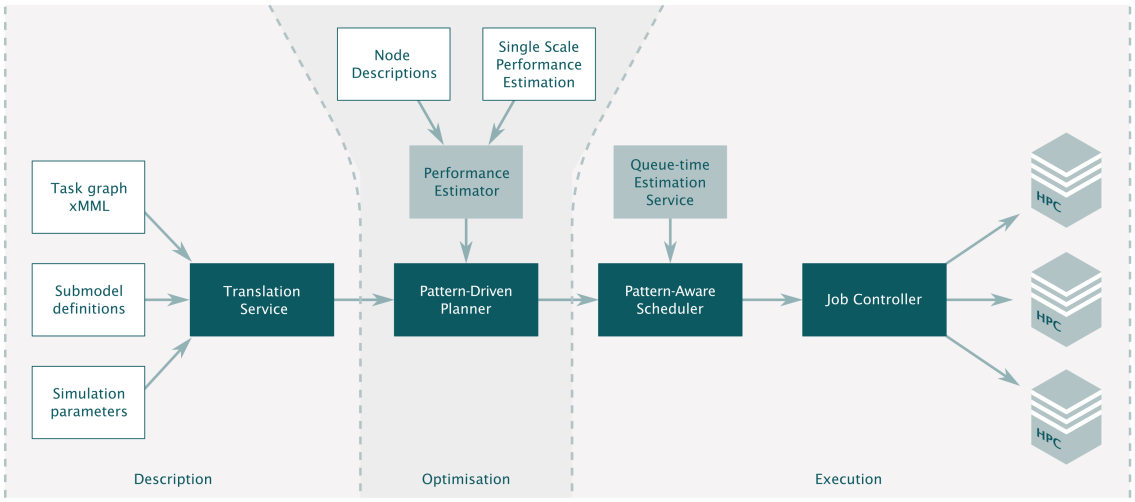


FIGURE 2.7: Conceptual overview of the Multiscale Computing Patterns architecture

while avoiding redundant computations. It does this by caching and reusing results from a database and by interpolating between previous outputs when possible. Since microscale simulations are often numerous and computationally intensive, they can dominate overall computing and energy costs. This makes it crucial for HMC algorithms to reduce the frequency and number of microscale computations.

In the **Replica Computing (RC)** pattern a large number of copies of simulations are needed to produce statistically robust results. These replicas are spawned in the initial step based on the parameter space set for sweeping. Then, simulations and data processing per replica take place. Both static and dynamic flavours of RC are considered. In the static flavour, the number of replicas is known in advance, while in the dynamic flavour replicas can be created during calculations depending on needs. All replicas execute independently of each other. If a replica (i.e., a simulation) fails, the RC pattern affords some level of fault tolerance, taking into account maintaining the overall statistics. The characteristics of the pattern make it relatively easy to spread the replicas over the available computational resources, thus the efficiency of the pattern is conditioned mostly on the efficiency of single replicas.

The MCP concept consists of several key components: a generic task graph that represents the structure of a computational pattern and encodes essential information for scheduling and execution of the multiscale application implementing this pattern; prerecorded performance data for individual single-scale models; an MML (or xMML) description of the multiscale model; and a set of algorithms and heuristics for transforming this information into input for the execution in the target computing environment.

For the practical realisation of this concept, a three-part middleware architecture, outlined in Figure 2.7, and discussed in [Alowayyed et al., 2019], has been proposed. These parts map to the main stages of the "MCP-powered" multiscale application.

The **Description** part provides a high-level specification of a multiscale application, encompassing all essential elements required for its execution. It includes a task graph expressed as xMML, definitions of individual submodels with their resource and dependency requirements, and configuration parameters for simulation. These three pieces of information are then supplied to *Translation Service*, which merges and converts them into a format suitable for the Optimisation part.

The role of **Optimisation** is to create a set of possible execution plans for the multiscale application based on the measurements collected and/or predictions of the performance of submodels

in predefined types of computing nodes. The plans are selected by *Pattern-Driven Planner* based on the evaluation of cost functions (e.g., efficiency, throughput, energy usage, or a combination) on available resources. The selected, most valued execution scenarios are then passed to the Execution part.

The **Execution** part has two main duties: determining the most appropriate execution plan from those proposed by the Optimisation part and ensuring its efficient and reliable execution in a distributed and heterogeneous infrastructure. A key component of this section is *Pattern-aware Scheduler* that uses novel methods to select a specific scheduling plan. The methods differ significantly from the available functionality of existing batch schedulers in HPC systems, such as in SLURM, which assigns tasks to resources based only on the current load of the system and the basic resource requirements specified by the user. In case of the proposed pattern-aware scheduler, more sophisticated system parameters are used to help the scheduler make an optimal decision, including queue-time estimation inferred from historical data and application energy profiles provided by the Optimisation component. Similarly, more specific user's criteria can be involved in the selection of the plan, including either a single time-to-completion criterion or a combination of two criteria: energy consumption and time-to-completion.

In order to effectively support complex, heterogeneous applications, the scheduler employs custom scheduling strategies focused on Multiscale Patterns and mechanisms aimed at enabling distributed execution, like resource brokering, advance reservation, and co-allocation.

Once a scheduling plan is chosen, it is passed to the underlying resources for execution. The dedicated *Job Controller* sets up the distributed computing environment, manages data transfers, and all jobs involved in the simulation. Execution involves the usage of some specialised methods for processing individual tasks, including workflows, job arrays, or Pilot Jobs [Bosak et al., 2021], which are used to optimise execution, reduce overhead, and increase throughput, especially for multiscale patterns like Replica Computing.

2.1.6 Summary

The methodology presented for multiscale computing defines a comprehensive ecosystem for the modelling, development, and execution of multiscale simulations on large-scale, heterogeneous, and distributed computing infrastructures. The Multiscale Modeling Language (MML), the Multiscale Modeling and Simulation Framework (MMSF), and the Multiscale Computing Patterns (MCP) together constitute three generic pillars that support the development and running of a broad spectrum of multiscale applications. A central element of this ecosystem is the execution layer, which must combine efficiency with high flexibility to accommodate the diverse computational and coupling requirements inherent to multiscale scenarios. The proposed mechanisms for distributed and heterogeneous multiscale computing, based on advance reservation and co-allocation of HPC systems, as well as coordinated cross-cluster execution of multiscale models, represent a key enabler in addressing these challenges.

2.2 Confidence of simulations

The application of computing simulations to real-life use cases would be useless without effective methods that can measure how much stakeholders can trust the simulation results produced. The computational models to become widely accepted in decision making processes, such as in medicine and industry, need to be rigoristically assessed as described, for example, in [Wright et al., 2020]

and [Groen et al., 2021a]. To handle this problem, the conception of **VVUQ**, i.e., a comprehensive methodology for **verification**, **validation** and **uncertainty quantification**, has emerged.

According to ASME [ASME, 2025] terminology:

Verification is an assessment of the accuracy of the numerical solution relative to the exact solution of the mathematical model. Verification includes two different activities: (1) *code verification*, which aims to determine whether the computer code correctly implements the mathematical model, and (2) *solution verification*, which checks the numerical accuracy of an output quantity of the computational model for the application of interest. Technically, validation is a software testing procedure - it aims to answer if we develop a model right.

Validation is an assessment of the accuracy of the simulation result relative to experimental measurements of the system responses. It compares the model to real-world observations (data) to ensure that it captures essential features (physical phenomena) of the real system. Validation aims to ensure the credibility of a model for its 'intended use'. Attempts to answer whether we develop the right model.

Uncertainty Quantification (UQ) is an estimate of the total uncertainty in the simulation result due to all sources of error and uncertainty. It uses mathematical methods to quantify the resulting uncertainty, while combined with **sensitivity analysis**, it enables the identification of which input uncertainties have the largest effect on the results of the computing simulations.

For many critical applications which require the highest-level of credibility, all these activities are essential and need to be incorporated in the process of model's development and evaluation. However, the topics of the dissertation focus on the uncertainty quantification and sensitivity analysis, which are typically the most computationally intensive and require advanced methods for automated, scalable and fault-tolerant executions.

2.2.1 Uncertainty quantification and sensitivity analysis

Even verified and validated code can exhibit some level of uncertainty arising from uncertain inputs, parameters, or numerical approximations during computations. At the highest level, as presented by [Roy and Oberkampf, 2011], uncertainty can be broadly classified into two types: **epistemic** and **aleatory**.

Epistemic uncertainty, also referred to as reducible uncertainty, arises from incomplete or imperfect knowledge of the system under investigation. This type of uncertainty can be mitigated by the acquisition of additional data, the enhancement of measurement accuracy, or the refinement of the underlying model. A common approach to quantifying epistemic uncertainty is through **inverse uncertainty quantification** using Bayesian inference, such as Markov Chain Monte Carlo (MCMC) sampling.

In contrast, **aleatory uncertainty**, also known as irreducible uncertainty, originates from inherent randomness in the system or process being modelled. This type of uncertainty cannot be eliminated, but it can be described using probability distributions (e.g., Normal, Uniform, Triangular). Its influence on model output can then effectively be analysed through uncertainty propagation methods (also known as **forward uncertainty quantification**), such as Monte Carlo (MC) sampling or Polynomial Chaos Expansion (PCE).

These two types of uncertainties can be investigated using **nonintrusive model evaluations** - often referred to as black-box approaches, which do not require access to the internal structure of the model and can therefore accommodate even closed-source simulations. The high-level

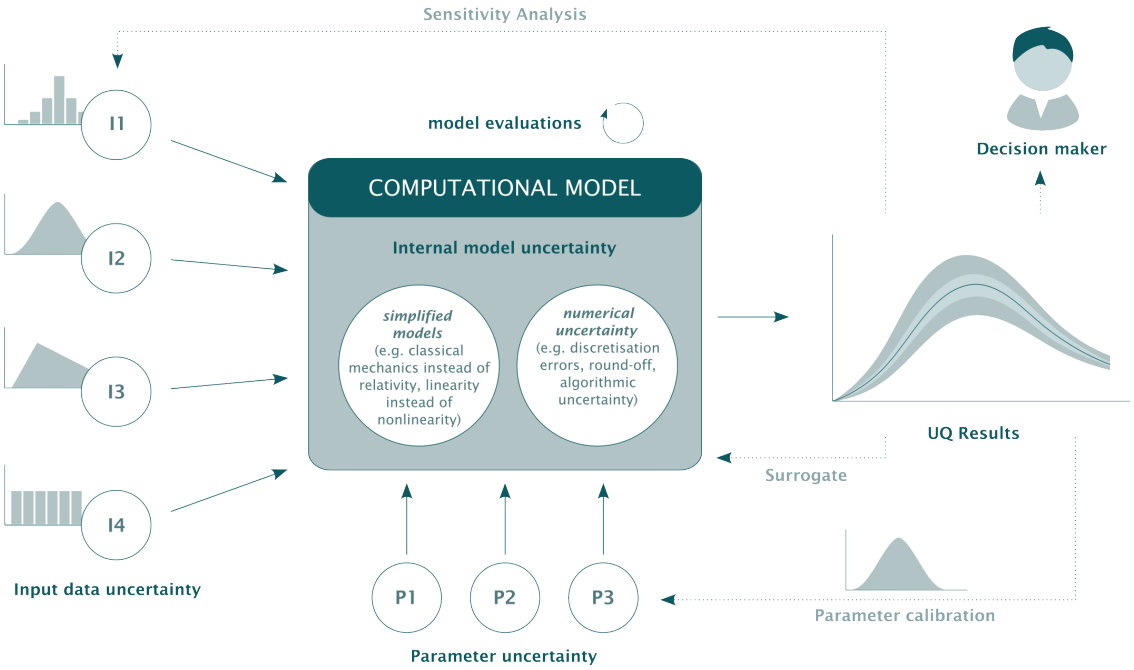


FIGURE 2.8: High-level overview of uncertainty quantification. The model incorporates various sources of uncertainty - such as uncertain inputs, parameters, or internal model variability. These uncertainties are propagated to the output UQ results using dedicated methods, which typically require a large number of model evaluations.

diagram presenting a non-intrusive UQ process is shown in Figure 2.8. In this process, the model is repeatedly evaluated using uncertain input data, uncertain parameters, and inherent internal variability to generate statistically meaningful results. These results can then be used for a variety of purposes, such as performing sensitivity analysis (forward UQ), parameter calibration (inverse UQ), constructing surrogate models, or directly informing stakeholders.

Despite this shared approach, the inverse and forward processes entail fundamentally different computational workflows as discussed, for example, in [Kimpton et al., 2025]. Inverse UQ, particularly when implemented using Bayesian inference, involves iterative updates to the probability distribution for the parameter under study. This characteristic limits the potential for effective parallelisation. The forward UQ process is generally more scalable. Typically, it consists of three main phases:

1. **Identification and characterisation of uncertainties that may affect the model.** These can include uncertain input variables (e.g., initial / boundary conditions, external inputs) and uncertain model parameters (e.g., material properties, empirical coefficients).

2. **Propagation of uncertainties through the model**, which involves evaluating the model under a range of sampled input/parameter values - e.g., using Monte Carlo methods or spectral expansion techniques like Polynomial Chaos Expansion or Stochastic Collocation.

3. **Interpretation of results**, based on statistical analyses of the model outputs, such as mean, variance, percentiles, and confidence intervals. This phase often includes sensitivity analysis to determine which inputs or parameters have the most significant impact on results.

From a computational perspective, forward UQ methods rely on independent model evaluations, making them well-suited for efficient parallelisation. However, the typically large number of required evaluations makes the execution of UQ scenarios computationally demanding. This dissertation proposes novel methods for *in-allocation* execution of model evaluations - discussed in detail later -

which enable efficient and scalable forward UQ on large-scale computing infrastructures such as HPC systems.

Selected UQ and SA methods

A variety of methods exist for performing UQ in both its forward and inverse forms. A selection of widely applied approaches is presented below.

For inverse problems, one of the most widely recognised approaches is **Markov Chain Monte Carlo (MCMC)**, which builds statistical insight into unknown parameters by generating samples from the posterior distribution through an iterative process of model evaluations driven by a Markov process. While MCMC is a powerful and flexible method, which can be effectively used, e.g., for calibration tasks, it suffers from poor scalability due to its sequential nature, which complicates parallelisation and increases computational costs, though techniques like parallel chains or surrogate models can mitigate these limitations.

Forward UQ, as discussed, for example in [Saltelli et al., 2008], benefits from a broad portfolio of available methods that can be chosen depending on model complexity, dimensionality, and computational budget. Common approaches include:

Monte Carlo (MC) – A straightforward sampling-based method that estimates the output uncertainty by repeatedly evaluating the model using random inputs drawn from known distributions. It is simple but computationally expensive.

Quasi-Monte Carlo (QMC) – Similar in spirit to MC, but instead of purely random samples, it uses low-discrepancy sequences (e.g., Sobol or Halton) to fill the input space more uniformly. This typically leads to faster convergence, especially in models with smooth output behaviour. A popular QMC strategy is the Saltelli sampling plan, which is particularly well suited for global sensitivity analysis as it allows efficient computation of Sobol sensitivity indices.

Stochastic Collocation (SC) – Builds a surrogate model by interpolating the model response at selected deterministic points (e.g., sparse grids). It offers higher efficiency than MC, especially for smooth models with moderate dimensionality, and shares theoretical foundations with PCE.

Polynomial Chaos Expansion (PCE) – Builds a surrogate model by expanding the output as a series of orthogonal polynomials of the uncertain inputs. PCE is efficient and provides analytic expressions for output statistics, but can be sensitive to dimensionality and nonlinearity. In practice, PCE and SC share conceptual foundations and applicability.

Gaussian Process Regression (GPR) – A surrogate modelling method that treats the model output as a Gaussian process, providing both predictions and uncertainty estimates. It is highly accurate for smooth, low-dimensional problems, but can be computationally demanding for large datasets.

Forward UQ is often combined with variance-based global sensitivity analysis, which provides comprehensive information on how input uncertainties influence the model output. One of the most widely used approaches is based on **Sobol indices** [Sobol, 2001], which quantify the contribution of each input variable to the output variance. These include first-order indices that capture the effect of individual inputs; higher-order indices that reflect interactions between inputs; and total-effect indices, which account for both individual and interaction effects of a given input. Sobol indices can be efficiently calculated from UQ results using analytical formulas (e.g., derived from PCE or SC surrogate models) or using sampling-based estimators.

2.2.2 Uncertainty quantification for multiscale simulations

The application of UQ for multiscale models may be performed in various ways as presented in [Ye et al., 2021], in particular:

1. The entire compound model is treated as a black-box and the UQ is performed on a whole model. Consequently, it is a non-intrusive UQ of an entire multiscale model.

2. Each submodel of a compound model is treated individually as a black-box with uncertainty propagated between submodels. This is called the semi-intrusive UQ of a multiscale model.

3. The uncertainty is embedded in the logic of the codes of individual submodels. Since the submodels need to be updated to implement this logic, this method is called intrusive UQ of a multiscale model.

In order to enable effective use of the first two approaches, which are more universal, a set of **Uncertainty Quantification and Sensitivity Analysis Patterns (UQPs)** has been proposed in [Ye et al., 2021] for multiscale models, with the aim of guiding the definition and automating execution of related scenarios:

UQP1: Non-intrusive pattern - The simplest UQP which does not exploit the multiscale simulation structure, and considers the multiscale model as a black box that has inputs and produces outputs. The execution of UQ for such a multiscale model does not differ from the execution of a classical single-scale model, so all UQ methods can be applied automatically.

UQP2: Semi-intrusive acyclic pattern - This UQP targets multiscale models with acyclic data flow, meaning there are no feedback loops between submodels. In this configuration, uncertainty propagates unidirectionally - the output uncertainty of one submodel becomes the input uncertainty of the next. This structure enables independent analysis of each submodel by applying UQP1 sequentially. As a result, UQP2 allows flexible integration of different UQ algorithms across submodels and supports separate execution, including the use of distinct computational resources or allocations.

UQP3: Semi-intrusive cyclic pattern - The goal of UQP3 is to enable uncertainty quantification (UQ) in cyclic multiscale models using a semi-intrusive approach. In such models, components interact bidirectionally, creating dependencies between parameters that must be preserved throughout the UQ process. Unlike UQP2, where models can be sampled independently, UQP3 requires conditional sampling based on the state of shared parameters. Conditional sampling ensures that the samples drawn for one model are consistent with the values used or produced by others, thereby preserving the statistical correlation introduced by cyclic feedback. To maintain consistency across the cycle, UQP3 introduces a dedicated component called the Coordinator, which manages inter-model dependencies and ensures coherence throughout the multiscale system.

For all UQPs, specific flavours A and B were also defined. UQPs A aim to improve the efficiency of uncertainty quantification through smarter sampling or coupling strategies (e.g., interpolation), while UQPs B focus on replacing expensive parts of the multiscale model with surrogate models, which can also be automatically generated as a result of UQ procedures.

2.2.3 Multi-level scheduling for UQ tasks

As discussed in the previous part, most of the UQ methods require multiple evaluations of models to obtain statistically meaningful results. This is computationally demanding even for evaluations of traditional single-scale applications, but in the case of multiscale simulations consisting of many coupled single-scale models, the problem becomes a real challenge. In general, there is a need to support simultaneous execution of many single-scale models that may have extremely large and different resource requirements, some single-scale models may need to be attached dynamically, the number of evaluations of single-scale models may not be known in advance, to name just a few difficulties as presented in [Bosak et al., 2021]. This complexity and dynamism require flexible methods that not only enable the execution of various computational scenarios, but also ensure efficiency and ease of use for the user, regardless of whether it is a user-personal computer, cloud infrastructure, HPC system, or even a set of distributed heterogeneous large-scale machines.

One of the main challenges is to overcome limitations resulting from policies of resource providers and local resource management system (LRMS) configurations. For example, the policy at the SuperMUC-NG cluster installed at the Leibniz Supercomputing Centre, in order to promote large-scale computing, allows users to submit and run only a small number of jobs at the same time, which is contradictory to the needs of UQ scenarios. Smaller HPC installations may be less restrictive, but in general the rule remains the same, and there is no straightforward way to flexibly schedule many jobs using basic LRMS mechanisms. **Submitting a large number of small, non-homogeneous tasks directly to the scheduler would be inefficient** as each job would require a separate allocation and scheduling process managed by the central service. If users want to efficiently execute a huge number of conceptually different tasks, they should therefore employ methods that reduce this overhead within a single job allocation. One possible approach is to define a processing scheme in a scripting language, although such ad hoc solutions are neither universal nor flexible and are often prone to errors and inefficiency, as discussed in [Bosak et al., 2021]. Consequently, one of the key required activities was the development of generic methods for multilevel scheduling that are efficient, scalable, portable across a wide range of computing facilities, and easy to integrate with the created surrounding VVUQ ecosystem.

Scalable task execution for UQ using Pilot Jobs

The requirements analysis performed led to the selection of the Pilot Job paradigm [Sfiligoi, 2008] for the defined needs of multiscale UQ scenarios. The particular advantage of this idea is that it adds an additional task management level to the already created allocation in LRMS. As presented in Figure 2.9, **from the perspective of LRMS, the Pilot Job is only a single regular task, but for a user it is a second-level resource management system** that can be administered and used exclusively depending on particular needs. Consequently, in a once-started Pilot Job, a user (or programme) may flexibly execute multiple new tasks with different resource requirements and custom ordering. What is essential, the scheduling of the child tasks is not managed by the LRMS, but it is done on a level of a Pilot Job instance, thus it is much more efficient and releases the parent queuing system from the scheduling of a large number of tasks.

This overreaching capacity of the Pilot Job to handle numerous tasks is essential particularly for UQ scenarios, where, easily, thousands of evaluations need to be executed for a single model. In turn, its support for heterogeneous workflows enables seamless integration with multiscale execution scenarios, and in particular multiscale UQ scenarios, such as those defined by UQP's Replica Computing pattern. Consequently, a dedicated development effort was undertaken to integrate the

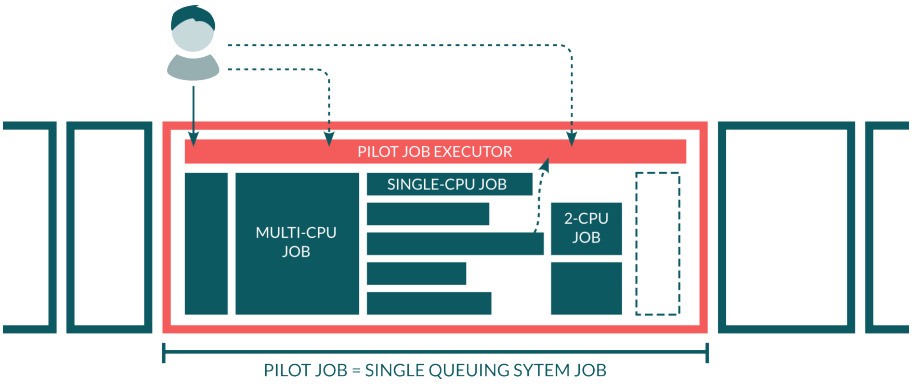


FIGURE 2.9: Conceptual illustration of the Pilot Job paradigm: within a single job submission to the local resource management system, multiple subjobs with distinct resource requirements can be instantiated, either manually by the user or programmatically by the framework.

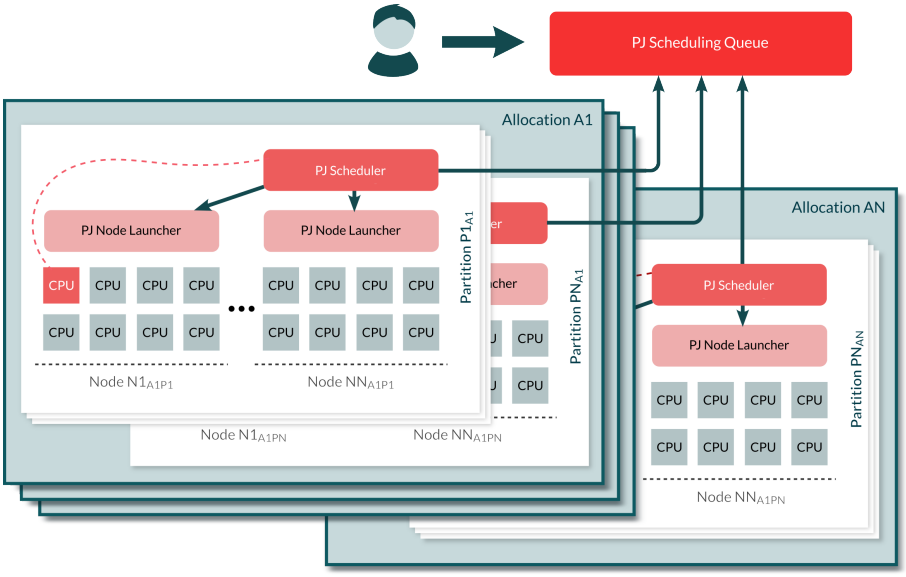


FIGURE 2.10: Hierarchical architecture of the proposed Pilot Job system, enabling cross-allocation task execution through integration with a central Scheduling Queue service.

Pilot Job mechanism into the UQPs, resulting in the implementation of a custom Pilot Job tool, namely **QCG-PilotJob**

The proposed methods for Pilot Job executions of UQ scenarios extend and precise the original basic conception of Pilot Job, with a set of key features aimed at simplification of its application to custom usage scenarios and enhancing its execution on heterogeneous resources. One of the biggest challenges was to meet requirements defined by extremely demanding multiscale applications. In order to reach the performance of hundreds of petaflops or even higher, these applications ultimately require an appropriate architecture. First of all, such an architecture should be scalable: the system should be easy to use, even on a laptop, but also easily extendable, portable, and efficient once the use-cases grow up to require HPC systems. Consequently, the natural choice was to propose a hierarchical structure of components, where top-level services are relieved of the high-intensity processing, which can instead be performed in a distributed manner by lower-level services. As part of this design, the proposed **architecture introduces the concept of a *partition*, representing a subset of resources that can be managed independently and dynamically attached to the optional top-level *Scheduling Queue* service**. This is presented in Figure 2.10.

In addition, the proposed Pilot Job system aligns with the common requirements of typical application scenarios. Specifically, it provides:

- Support for multiprocessor and multinode executions, enabling the use of parallel frameworks such as OpenMP and MPI for individual task execution;
- Management of task dependencies to ensure the correct execution order;
- Fail-recovery mechanisms to resume workflows interrupted prematurely (e.g., due to exceeding allocated time limits);
- Capabilities for execution tracking and performance analysis.

2.2.4 Automation of UQ scenarios execution

Applying UQ to computational models, although often essential for ensuring actionable results, represents an additional layer of effort, both logically and technologically, that extends beyond the core domain of expertise of an individual scientist or engineer. Furthermore, UQ algorithms, while potentially critical to the validity of simulation results, are supplementary to the fundamental scientific model itself. Finally, the increased number of executions needed for reliable UQ imposes significantly higher computational resource demands in comparison to the non-UQ runs. This is especially evident in the case of highly complex models, such as those in multiscale simulations [Wright et al., 2020].

One of the goals of the activities carried out by the author was to develop **methods for user-friendly VVUQ** that are particularly applicable for multiscale applications. The research performed revealed several key areas that are essential for seamless integration and robust execution of the VVUQ workflows with complex and computationally demanding models, but are not covered with available software. Consequently, within the VECMA research project, a generic modular system, the *VECMA Toolkit*, was developed to automate VVUQ. This conception has been further extended by the author of this dissertation to the form of a Software as a Service (SaaS) solution offering a web-native interface and fully automated execution of selected, commonly used uncertainty quantification and sensitivity analysis methods.

Modular toolkit for VVUQ of complex models - VECMA Toolkit

VECMA Toolkit has been proposed as a collection of modular components designed to comprehensively address the diverse requirements of application use cases in regards of VVUQ, with particular emphasis on UQ and SA, and multiscale simulations. The special focus has been given to facilitate execution of nonintrusive UQ and SA procedures on large-scale computing resources and to streamline their integration with existing application workflows. Recognising the wide variety of potential scenarios, the toolkit was designed to support the construction of use-case-specific platforms by allowing users to select only those components that deliver clear value for their particular context, minimising unnecessary configuration and execution overhead. This section focusses on the role of selected key blocks of the toolkit in which the author has been involved in development. For additional context and a broader perspective on the toolkit, the reader is referred to [Groen et al., 2021a]. Moreover, to highlight the available options and demonstrate the flexibility of the toolkit, VECMA illustrated its functionality through real-world application use cases, as described in [Wright et al., 2020].

High-level handling of UQ workflows - EasyVVUQ. Generally, UQ/SA processes used to be tedious and error prone if done in an ad hoc manner, and further exacerbated if the parameter space is large with a corresponding large number of runs (e.g., of order thousands to millions

of runs), or if each simulation is very computationally expensive. There is a need to streamline the definition of the scenario and coordinate the whole process, from sample generation through execution to analysis, while keeping high levels of modularity and flexibility to enable processing of more sophisticated scenarios. The VECMA Toolkit addresses these expectations with the development of a dedicated library, **EasyVVUQ**, that breaks down the VVUQ process into five distinct stages [Richardson et al., 2020]:

1. Application description, which further can be divided into the following items:
   - Encoder - responsible for producing input files for the simulation.
   - Decoder - responsible for parsing the output files of the simulation and extracting the needed values.
   - Execution action - describes how the simulation is run
2. Sampling: this step is very dependent on the VVUQ technique in question. The main task of sampling components is to produce a list of parameter sets for the simulation.
3. Execution: this is handled entirely externally to EasyVVUQ, by local processes or more sophisticated solutions for HPC processing, like Pilot Jobs.
4. Collation: collects the outputs of the simulations and then stores them in the EasyVVUQ database to be retrieved later for analysis.
5. Analysis: perform a statistical analysis on the collected data. This analysis can then inform further actions, such as collecting more samples.

EasyVVUQ is particularly designed to be agnostic to the choice of execution middleware, due to the large range of possible execution patterns that may be required in a local or HPC-executed workflow. One of the supported options, developed as part of this dissertation, is the **integration with the Pilot Job** mechanism over a dedicated execution engine, which enables efficient execution of numerous tasks on HPC resources (this is implemented through integration with QCG-PilotJob). In addition, the library tracks and logs the applications of the sampling elements along the way, allowing a reasonable level of restartability and failure resistance, which is essential for sizeable models as presented in [Groen et al., 2021a] and [Wright et al., 2020].

In addition to EasyVVUQ, VECMA Toolkit also includes **MUSCLE3** [Veen and Hoekstra, 2020], the successor of MUSCLE2, which contains embedded mechanisms for UQ in tightly coupled multiscale simulations.

Infrastructure access and automation of execution - QCG and FabSim3. The access to computational resources for the execution of demanding UQ workflows defined with EasyVVUQ or other related software may be an additional barrier or require additional efforts from users. In order to address this problem and to respond to different expectations, experience, and preferences of users, VECMA Toolkit provides a set of user-facing tools that range from command-line interfaces to high-level GUIs that not only simplify access to resources, but also streamline typical processes undertaken by scientists or engineers interested in VVUQ.

For advanced users and experienced scientists, a Python-based research toolkit, **FabSim3**, enables the automation and management of complex computational workflows, including ensemble runs, remote execution, and code coupling. It supports flexible customisation for new systems and applications, allowing researchers to streamline and debug large-scale simulations by introducing application-specific logic at the execution layer. To efficiently execute jobs across distributed

resources, this toolkit integrates with middleware systems, like **QCG**, that allows submission of various task types - such as parameter sweeps, array jobs, multistep workflows, or Pilot Jobs - on single or multiple HPC clusters.

For users working with well-defined scenarios or those who prefer a more accessible interface, a graphical desktop application, called **QCG-Now**, is also available. Developed as part of this dissertation, it simplifies the submission and monitoring of HPC jobs by offering automatic notifications, streamlined data transfer, and real-time progress tracking through an intuitive interface. Although it offers less flexibility than command-line tools, it provides a higher level of automation and ease of use, making it particularly suitable for structured workflows, such as well-defined UQ studies.

UQ and SA delivered as a service

Advancements in web technologies combined with shifting user preferences have driven the development of **novel methods for conducting uncertainty quantification** in computational models. These developments, which are an integral part of the research presented in this dissertation, resulted in the design of a **mUQSA** system focused on high intuitiveness and advanced automation of UQ / SA processes, as detailed in [Bosak et al., 2025] and exemplified in [Kulczewski et al., 2024]. From a high-level perspective, the goal of mUQSA was to integrate an intuitive web-based user interface, HPC processing, and state-of-the-art UQ algorithms, to provide a comprehensive and versatile environment for widely requested uncertainty-related studies. Consequently, due to the diversity and heterogeneity of possible uncertainty quantification scenarios, the intention was not to build a universal tool, but rather to **support common UQ and SA workflows**, to streamline their integration into users' models. Consequently, the platform offers several well-established methods for forward UQ and computation of sensitivity indices. As illustrated in Figure 2.11, this support is comprehensive and covers the entire workflow, from scenario definition through fully automated execution with parallel processing to presentation of results.

Consequently, the mUQSA platform delivers:

Intuitive wizard for the preparation of the UQ / SA scenario. The wizard guides users through logical steps to define sampled parameters, configure methods, and specify model execution, streamlining UQ integration even for non-experts.

Built-in interactive presentation of results. Users can view output through interactive charts and tables, with access to raw data for more detailed analysis when needed.

Automated execution on an HPC machine. The platform efficiently manages scheduling and execution of computationally intensive tasks on large-scale computational resources, allowing users to simply start experiments while the system runs required calculations in the background.

Support and documentation. Comprehensive documentation and tutorials provide essential guidance, helping users navigate the platform and use advanced features beyond the brief GUI help.

The primary objective of mUQSA is support for complex, but typical use-cases; however, thanks to the intuitiveness of the interface and the automation offered, mUQSA can also be a starting point to get practical knowledge in the fields of uncertainty quantification and sensitivity analysis for those who are not yet familiar with the concepts.

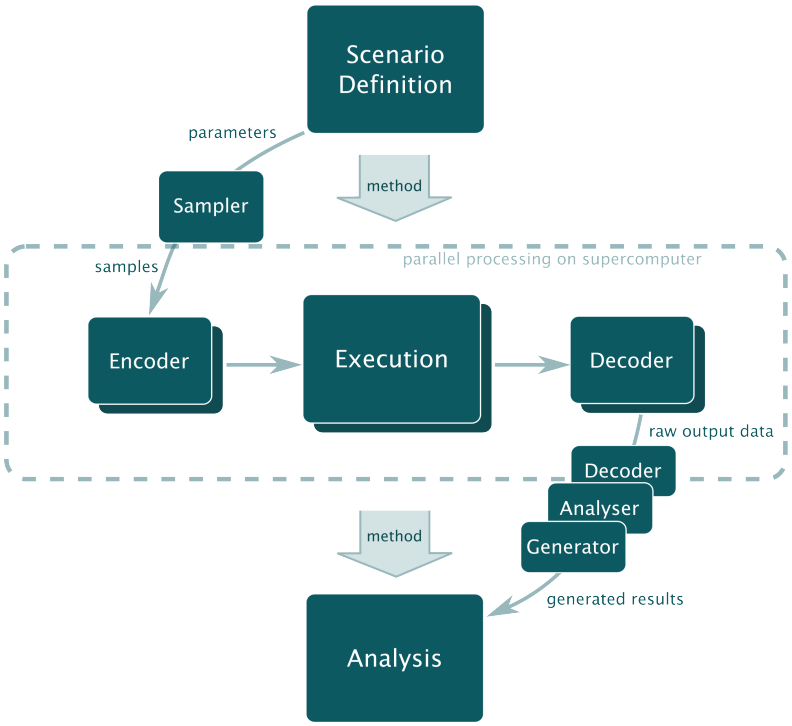


FIGURE 2.11: Processing workflow in the mUQSA platform. Users concentrate on Scenario Definition and Analysis of results, while the rest of processing is fully automated.

2.2.5 Summary

VVUQ of computational models is a powerful methodology that can substantially enhance model performance and increase the reliability and actionability of simulation results. A well-established mathematical framework has been developed to facilitate the integration of VVUQ into a wide range of applications. However, UQ methods typically rely on a large number of model evaluations, which, particularly for complex systems such as multiscale models, results in considerable computational demands and necessitates specialised techniques to achieve efficient execution. Moreover, applying UQ to a model often presents practical challenges and introduces additional implementation overhead. This dissertation introduces the VECMA Toolkit and presents several methods developed to address these challenges. Among them is the Pilot Job paradigm, which, when applied to UQ workflows, enables flexible and scalable execution of complex uncertainty quantification scenarios on HPC systems. The discussion further examines different approaches to user-level automation of UQ, as exemplified by the EasyVVUQ and mUQSA frameworks.

Chapter 3

Experimental Studies and Results

Multiscale simulations and uncertainty quantification are two headline topics in computational science, and they are closely related. Uncertainty quantification not only complements multiscale simulations by making them more trustworthy or improving their performance (e.g., surrogate modelling, sensitivity analysis), but both concepts also share exceptional demands for computing power and pose significant challenges for the middleware systems that must handle the complexity and heterogeneity of the underlying workflows. This chapter presents the concrete achievements of the dissertation, outlining the solutions developed and the results obtained in direct response to these two areas.

3.1 Enabling distributed execution of multiscale simulations

Running a complex multiscale model, particularly consisting of submodels of heterogeneous resource requirements (e.g., CPU vs. GPU vs. SMP), required the development of a middleware system capable of coordinating execution on many, often distributed computational resources.

3.1.1 Methods for advance reservation and co-allocation of computing resources

The first challenge in enabling cross-site execution of multiscale simulations was to implement advance reservation and co-allocation of resources. For loosely coupled models, these capabilities are less critical because the submodels run sequentially one after the other. In such cases, each submodel can be executed as an independent task managed by the LRMS in the usual manner. This scenario is illustrated in Figure 3.1 (a), showing a hypothetical execution of a material science application. Here, execution of each submodel is preceded by a queueing delay, which significantly extends the overall runtime. However, with the proposed advance reservation mechanism, this queueing can be eliminated, allowing submodels to run within a much narrower time window, as shown in Figure 3.1 (b). This improvement is especially valuable for simulations that require the freshest possible input data and rapid computation, such as weather forecasting.

Undoubtedly advance reservation and co-allocation have a critical importance for a distributed execution of tightly-coupled multiscale models, where the submodels have to be executed at the same time and communicate with each other. One of the representative applications following this coupling scheme is the three-dimensional model of in-stent restenosis in coronary arteries (ISR3D) [Borgdorff et al., 2012][Evans et al., 2008]. In summary, in-stent restenosis is the modeled process of excessive tissue regrowth within a stent following balloon angioplasty, caused by vessel injury during stent deployment [Moustapha et al., 2001][Kastrati et al., 2000]. Although its precise mechanisms

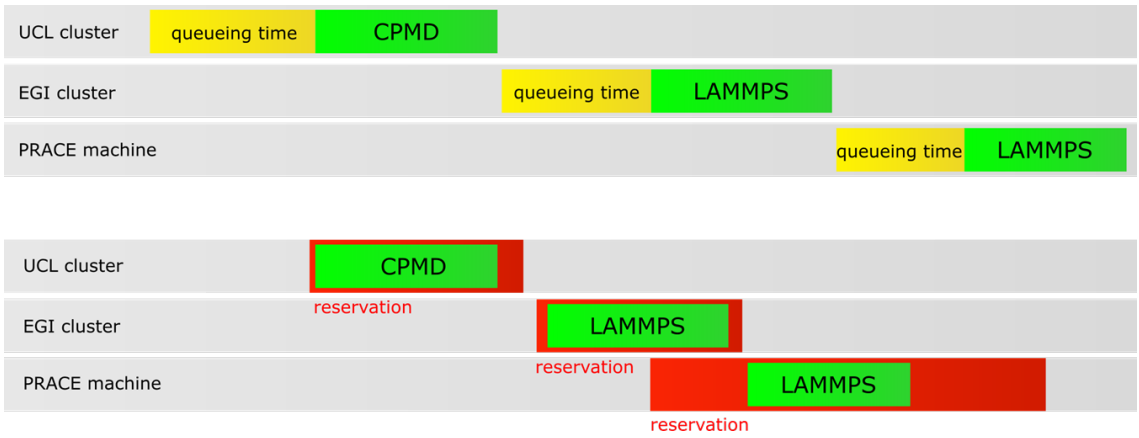


FIGURE 3.1: Two example executions of a materials science multiscale application composed of three loosely coupled submodels running on three computing sites: (a) without advance reservations and (b) with advance reservations.

are unclear, they pose a significant limitation to the success of long-term treatment. ISR3D tackles this challenge by simulating the biological and physical processes driving restenosis across multiple scales. It integrates four single-scale submodels: smooth muscle cell proliferation (SMC), thrombus formation (Blob), blood flow (BF), and drug diffusion from a drug-eluting stent (DD). Each of these models acts on the same spatial scale, but they all exhibit temporal scale separation.

ISR3D has been developed with MUSCLE2. It is tightly coupled because its four submodels exchange data in a cyclic sequence - SMC triggers Blob, which alters geometry for BF and DD, whose results feed back into SMC. This loop, which additionally involves data translation with Mappers, runs thousands of times. Since the submodels are heterogeneous and some of them, as shown in Table 3.1, require significant parallel resources (BF) while others can run just sequentially (SMC), the use of distributed and diverse resources can be justified, as described in [Borgdorff et al., 2012].

Submodel	**Test env.**	**Runtime/iter.**	**Memory**	**Parallel**	**Language**
BF	Huygens	10 min.	4 GB	extremely, using MPI	C++
DD	Mavrino	10 sec.	100 MB	up to 4 cores, using threads	Java
SMC	Reef	10 min.	200 MB	no	C++
Blob	Reef	2 min.	50 MB	no	Fortran
Mappers	Reef	30 sec.	500 MB	no	Java

TABLE 3.1: Resource requirements and runtime statistics of ISR3D submodels when executed on selected test environments. Reported runtimes correspond to a single iteration of the multiscale loop. The mappers are listed as a combined cost in [Borgdorff et al., 2012].

The methods for advance reservation and co-allocation as part of MMSF have been grounded in QCG, a middleware system implemented by the Poznan Supercomputing and Networking Center. QCG uses the advance reservation mechanism available in almost every modern batch system in order to co-allocate resources belonging to two or more resource providers. Following the architecture of the system presented in Figure 3.2, the whole process is managed by QCG-Broker, a metascheduler service, which, whenever possible, uses resource-level QCG-Computing services to book requested resources. The process is automated, but in order to support sites without the QCG-Computing service deployed, the system enables reservations created manually. As presented in [Borgdorff et al., 2012], this process has some similarities with the Two-Phase Commit Protocol known from transactions systems, i.e., when advance reservations were created successfully at all sites, the job is submitted (COMMIT); otherwise, all reservations are cancelled (ROLLBACK) and a new attempt is made.

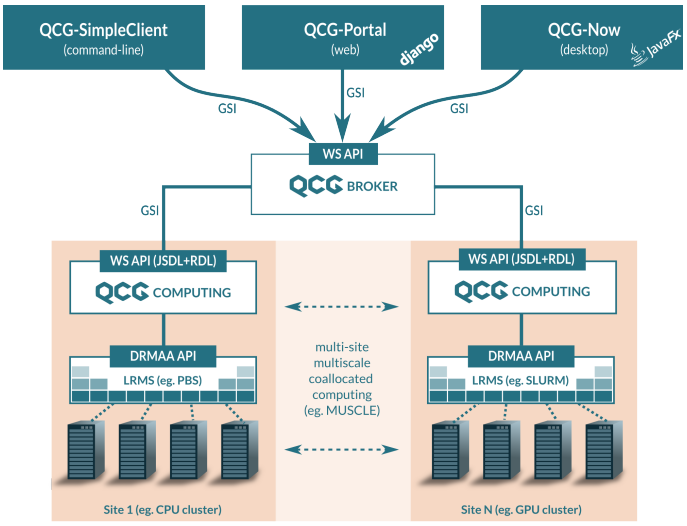


FIGURE 3.2: High-level architecture of QCG system for co-allocated execution of multiscale simulations on multiple HPC systems or distributed computing sites.

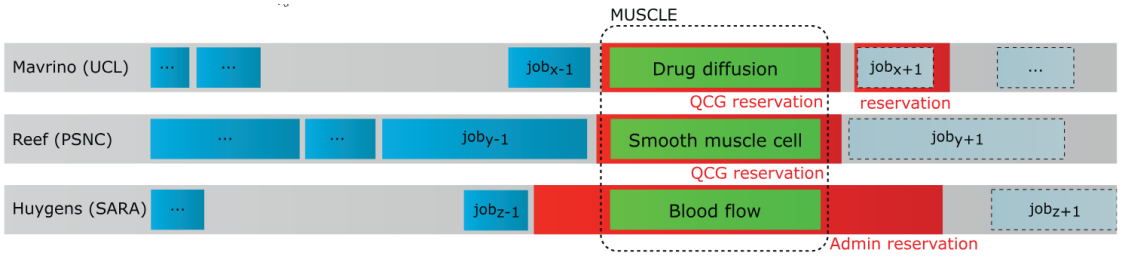


FIGURE 3.3: An example execution of tightly-coupled multiscale ISR3D application on three sites with co-allocation across two dynamic QCG advance reservations and one manual reservation.

Figure 3.3 illustrates an example of co-allocated execution of the ISR3D application on a testbed infrastructure. In this case, the system administrator made a manual advance reservation on the Huygens HPC system in the Netherlands for the execution of BF, while the QCG stack dynamically allocated reservations for DD on the UCL's Mavrino computing cluster in the UK, as well as for SMC, Blob, and Mappers on the Reef computing cluster at PCSS in Poland.

3.1.2 Optimisation of distributed computing with multiscale patterns

The support for advance reservation and co-allocation of resources discussed in the previous section provides a basic mechanism for distributed computing, however, it abstracts from more advanced optimisation aspects. For example, without the additional functionality, the execution of a tightly-coupled multiscale application composed of two submodels of which one is highly parallelised and resource-demanding, while the other is a long-running and sequential, would likely be highly inefficient as the resources reserved for the resource-demanding submodel will wait unused by the time when the sequential submodel is running. Similarly, if there are multiple resources available, which of them should be used in urgent scenarios, and which of them should be used in order to minimise environmental footprint? These kinds of issue have been analysed and addressed with Multiscale Computing Patterns outlined in Section 2.1.5 and presented in detail in [Alowayyed et al., 2019], which extends the MMSF framework. In a multi-phase process, the execution plans are initially generated by the system's Optimisation component based on the previously collected data, while their final selection takes place within the Execution component, using the Pattern-Aware

Scheduler built on top of the QCG middleware, which is aware of the current situation in the infrastructure. The QCG software is also responsible for distributing jobs across multiple clusters.

The following descriptions of Extreme Scaling and Replica Computing patterns derived from [Alowayyed et al., 2019], illustrate the practical benefits achieved through the developed approach.

The **Extreme Scaling** pattern is used for tightly coupled multiscale models where the primary component is highly compute-intensive and scales efficiently to large core counts, while auxiliary components require comparatively little computation but may still take significant time to complete. Because resources allocated to the primary model can remain idle while waiting for auxiliaries, **coordinated interleaving** multiple instances of the full multiscale model provides a relatively simple, yet effective solution. This approach allows otherwise idle resources to be used for additional instances, improving the overall efficiency of execution. This approach is applicable in situations where there is a need to calculate more than one simulation of the same multiscale model; however, it showcases an interleaving paradigm that can be ported to more sophisticated scheduling, which may involve different multiscale models.

The **Replica Computing** pattern is used for workflows that require the execution of multiple independent simulation instances (replicas) to ensure reproducibility, statistical sampling, or uncertainty quantification. In contrast to ES pattern, where the number of instances of models is static and known in advance, here the number of replicas may be balanced as well to reach the optimisation goal. Consequently, there is an obvious trade-off between the number of replicas that must be executed, the minimum number of cores that one single replica needs, and the total number of cores available for the overall job. In the simplest cost-function, where we consider only time-to-solution, all replicas would be run concurrently on the node with the shortest running time per replica. However, real systems impose several constraints that need to be considered in the scheduling process, such as the permitted number of concurrent jobs, time limitations, node availability, and queueing time. The Pattern-Driven Planner, in order to determine the optimal strategy, needs to decide whether to run all replicas on a single, more pewerful HPC system, accepting its constraints, or to distribute replicas across multiple computing resources.

The research and performance evaluation demonstrating the advantages of Multiscale Computing Patterns for several realistic applications are presented and discussed in detail in Section 3.3.

3.1.3 Enhancing communication performance between submodels with MUSCLE Transport Overlay (MTO)

Execution of a tightly-coupled multiscale simulation over a number of distributed resources also requires a mechanism enabling efficient communication between submodules. Sending files is usually not an option as it involves time consuming I/O operations, which are unacceptable for frequent message exchange common for tightly-coupled models. MUSCLE2 contains its own protocol for message exchange based on TCP sockets. This works reasonably well in a single-cluster environment. However, in case of distributed execution, the fact that the majority clusters use a separate, firewalled network with private IP addresses for their worker nodes causes a significant problem: without additional mechanisms or special configuration, a submodel running on one site would not be able to send messages to submodels running on the others. This problem has been addressed by applying the port-range technique [Maassen and Bal, 2007] and implementing a user-space daemon as discussed in [Borgdorff et al., 2012], MUSCLE Transport Overlay (MTO). MTOs are typically launched on the interactive nodes of all sites participating in a multiscale simulation and act as proxies for message exchange between submodels. The only requirement is that a unidirectional connection can be established between every pair of MTOs.

By default, MTO communicates using plain, non-blocking TCP/IP sockets. This is the standard and well-tested mode of operation. To improve performance on wide area networks (WANs), each connection uses a 3 MB local buffer. The MTO prioritises sending data over receiving it: If the outgoing send buffers become too large or too numerous, the system will pause accepting new incoming data until the backlog clears. This avoids overwhelming the network or local memory. Optionally, connections between MTOs can use the MPWide library instead of plain TCP/IP. MPWide is a communication library optimised for high-performance data transfer between distributed HPC sites, often achieving throughput and latency better than standard sockets over WANs.

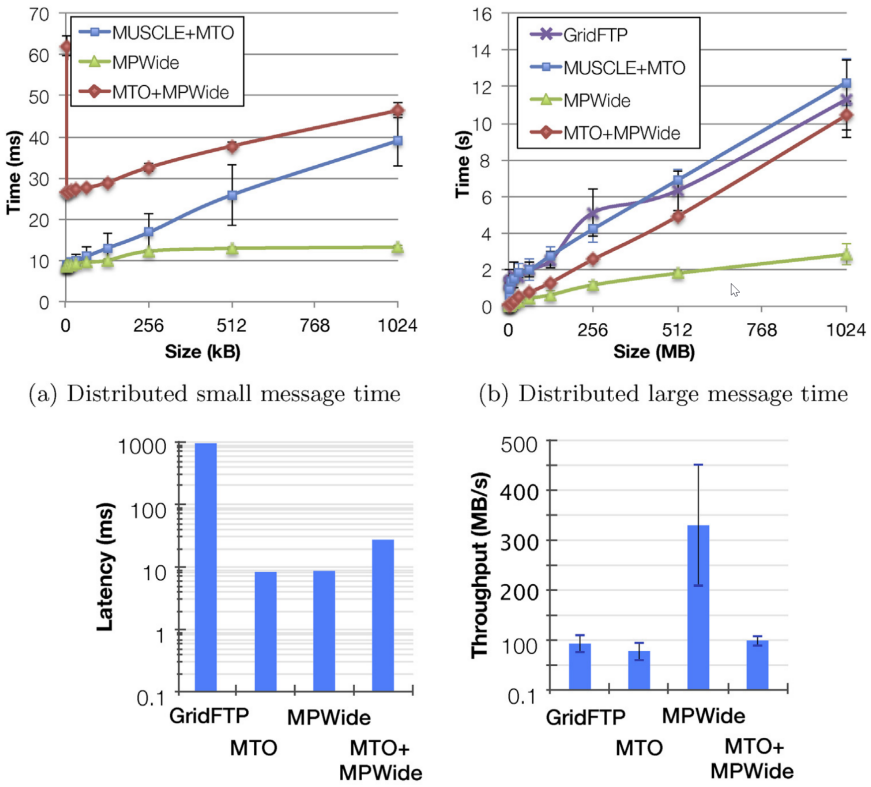
In order to measure MTO performance, a set of tests has been performed that compare the efficiencies of different mechanisms for data transfer between distributed sites. The tests included file transfers using GridFTP [Allcock et al., 2005], data transfers using MPWide alone, and MUSCLE2 transfers over both MTO alone and over MTO combined with MPWide. As presented in Figure 3.4 and discussed in detail in [Borgdorff et al., 2014], the overall performance of MTO communication is highly competitive for application in tightly-coupled multiscale simulations. In particular, the measured latency imposed by MTO was very small, around 10 ms, just 2 ms above the ping time, and 2 orders of magnitude less than the latency of GridFTP. This resulted in much better performance of MTO in comparison to GridFTP for scenarios with a small message size. The tests performed also demonstrated an exceptional advantage of MPWide. Although the early integration between MTO and MPWide, due to poor optimisation, did not behave well, MPWide itself outperformed other options. This suggests that MTO communication could be further optimised through tighter integration with MPWide, particularly for demanding use cases.

3.1.4 Summary

Advanced automation is essential for the efficient execution of complex multiscale models. Within this framework, the advance reservation and co-allocation mechanisms integrated into the MMSF constitute a core component enabling coordinated computation across distributed infrastructures. Building upon the MMSF, the Multiscale Computing Patterns (MCPs) introduce a higher level of automation aimed at optimising the execution efficiency of common multiscale computing scenarios. These capabilities are further augmented by the MUSCLE Transport Overlay (MTO), a dedicated daemon designed to enhance communication performance in tightly coupled models developed using MUSCLE2.

3.2 Pilot Job executions for multiscale simulations and VVUQ

As discussed in the previous section, the execution of numerous jobs on computing clusters can be limited by the number of concurrently run jobs allowed, which is only one of many possible restrictions. Scheduling of replicas in case of Replica Computing pattern, and even more importantly the execution of evaluations of models in UQ, are good examples of scenarios which need a large number of jobs to be executed, so they both are likely to be affected by these limitations. In order to resolve this issue, a new QCG-PilotJob tool has been developed and discussed in [Bosak et al., 2021]. QCG-PilotJob is a lightweight implementation of a second-level scheduler, which means it is able to manage jobs independently and in addition to the cluster's main queuing system. What is unique in QCG-PilotJob is the fact that it runs fully in users-space, thus its operation is not dependent on the cluster's administrators nor constrained by the global configuration or technical limitations.



(a) Distributed small message time

(b) Distributed large message time

(c) Latency

(d) Throughput

FIGURE 3.4: Comparison of message transmission performance between the distributed clusters Huygens and SuperMUC, highlighting the performance of MTO against other communication technologies. (a) Message transfer time on a kilobyte scale (excluding GridFTP, which fluctuates around 890 ms); (b) transfer time on a megabyte scale; (c) latency; (d) throughput.

Consequently, QCG-PilotJob provides much more freedom, flexibility, and dynamicity to manage tasks than is offered by standard LRMSes like SLURM. First, since managing jobs is done in a user space, QCG-PilotJob does not enforce limitations on their number. Furthermore, it allows the definition of complex execution scenarios through the support for DAG workflows.

To improve usability, QCG-PilotJob offers two modes of task submission. The first is a basic option, where tasks are described in a JSON file and submitted from the command line when the tool is started. The second is a more advanced option, which allows jobs to be created and managed dynamically through the provided Python API, directly from within a running Python program. This latter approach has been particularly valuable in VECMA scenarios, where flexibility and run-time adaptability are essential.

3.2.1 Application in VVUQ scenarios

EasyVVUQ is a tool for domain experts who work on concrete VVUQ scenarios related to their applications. Although some knowledge of the VVUQ methodology is necessary, these experts should not waste their valuable time setting up the logic of execution of EasyVVUQ workflows on computing resources. Rather, they should focus on purely scientific or engineering aspects. To this end, some approaches for the integration of EasyVVUQ with QCG-PilotJob have been developed that simplify the usage of HPC resources and make it efficient, as described in [Bosak et al., 2021]. These advanced automation mechanisms include:

Integration through FabSim3: QCG-PilotJob has been integrated with the FabSim3 automation toolkit to support the execution of application campaigns consisting of numerous jobs. Since FabSim3 internally uses EasyVVUQ, the combined execution of EasyVVUQ and QCG-PilotJob is also possible.

Integration through the EQI library: EQI, which stands for EasyVVUQ-QCGPilotJob Integrator, is a lightweight library designed to bring optimised processing schemes to selected types of highly demanding EasyVVUQ workflows. It makes use of advanced functionalities of QCG-PilotJob, such as the resume mechanism and iterative jobs.

Direct integration with EasyVVUQ: It is the most straightforward type of integration, where QCG-PilotJob is transparently employed in EasyVVUQ as one of its internal execution engines. This is also the recommended approach for the usage of QCG-PilotJob in the latest versions of EasyVVUQ. However, due to its generic character, this mechanism can be less efficient for the specific scenarios handled by EQI.

3.2.2 Distributed execution

In its developed implementation, QCG-PilotJob launches jobs within a single allocation on a computing resource. This approach has proven useful for many practical use cases; however, it lacks the scalability and flexibility that can be achieved when jobs span multiple allocations that can be dynamically attached. For example, a scenario executed could autonomously decide to add or release computing nodes in response to changing computational demands. To address this limitation, on the basis of the conceptual ideas discussed in Section 2.2.3, a proof-of-concept for the next generation of QCG-PilotJob has been proposed. This prototype follows the architecture outlined in that section, although more work is required to bring it to full maturity. Once fully implemented, it will allow dynamic attachment of not only HPC allocations, but also other types of resources, like Cloud.

3.2.3 Performance evaluation

To assess the practical usefulness of QCG-PilotJob for executing real-world use cases, several performance test campaigns were conducted, ranging from fully synthetic benchmarks to scenarios closely resembling actual production workloads. Early experiments on 5,000 CPU cores demonstrated highly promising results, with more than 99% execution time devoted to user-defined jobs. More specifically, running 20,000 jobs of five minutes each achieved 99.2% resource utilisation, leaving only 0.08% overhead for scheduling, as shown in [Bosak et al., 2021]. This was a motivation to perform subsequent large-scale evaluations on the Eagle, SuperMUC, ARCHER2, and Cartesius supercomputers. To ensure reliable and repeatable execution of the tests, a dedicated QCG-PilotJob Performance Tracking and Analysis Suite was developed. The suite consists of an executable wrapper that records detailed timing information from QCG-PilotJob runs - including precise start and end times of individual jobs - and a command-line tool that generates a wide range of statistics from the collected data. The tests further confirmed the good efficiency of the tool. Even in demanding configurations involving up to 6,400 relatively short jobs on 102,400 CPU cores (see Figure 3.5), QCG-PilotJob in its basic configuration, i.e., without partitioned execution, maintained the efficiency above 91%

Another set of QCG-PilotJob performance tests has been conducted as part of a larger campaign to compare different ensemble submission approaches available in the VECMA Toolkit [Groen et al., 2021a]. In particular, this evaluation contrasted submission of ensemble scenarios directly through

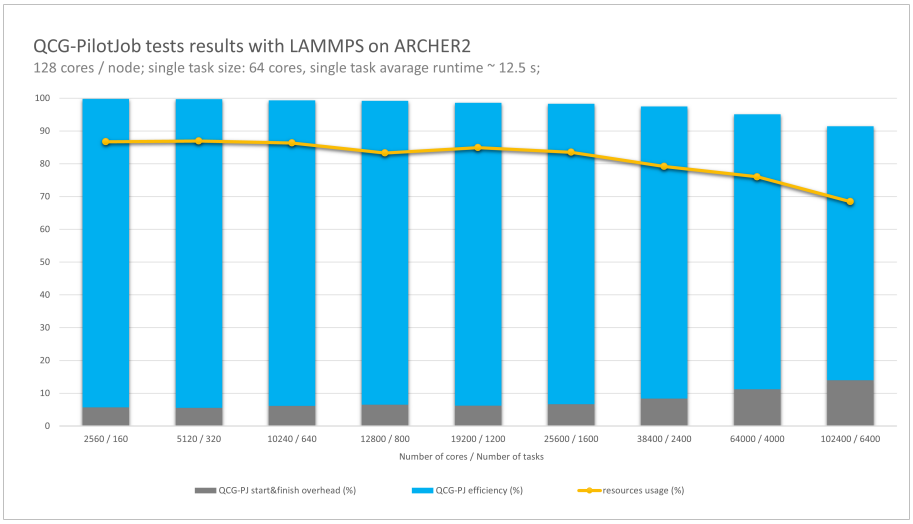


FIGURE 3.5: Results of QCG-PilotJob performance tests executed on ARCHER2 supercomputer with the short jobs involving LAMMPS application execution. The gray part of a bar showcases the time spent on initialisation and finalisation of QCG-PilotJob, which is likely to be neglected in the real situation, where the computational part of jobs will be much longer.

QCG-PilotJob within an already created allocation, with submission through FabSim3 using either its basic mechanism (issuing individual calls to the queuing system) or its more advanced mechanism (submitting batches of jobs to the queuing system, with individual tasks subsequently managed by QCG-PilotJob within a queuing system task). The high-level summary of the results is presented in Table 3.2. In terms of performance, as expected, direct submission through QCG-PilotJob produced significantly better results than either of the two FabSim3 options. However, even the hybrid approach, in which FabSim3 is based on QCG-PilotJob, showed considerable benefits compared to the traditional FabSim3 mechanism, confirming the practical advantages of integrating QCG-PilotJob into higher-level automation toolkits.

TABLE 3.2: Comparison of different submission approaches available in VECMAtk in terms of usability, remote capabilities, and submission overhead

Approach	**Usability**	**Remote file stage?**	**Remote deployment?**	**Submission overhead**	
				per 100 jobs	**per 1000 jobs**
FabSim3 only	excellent	yes	not needed	40–90 s	not attempted
FabSim3+QCG-PJ	excellent	yes	user-space only	33–36 s	250–300 s
QCG-PJ only	good	no	user-space only	< 5 s	40 s

3.2.4 Summary

The Pilot Job paradigm has been widely recognised as an effective approach for executing complex computational workflows composed of numerous conceptually distinct tasks. QCG-PilotJob is a lightweight and flexible implementation of this paradigm, designed with portability, efficiency, and scalability as primary objectives. Extensive large-scale testing campaigns have demonstrated its strong performance and robustness. Building upon these capabilities, QCG-PilotJob has been successfully applied in VVUQ scenarios, particularly through its integration as an execution engine within the EasyVVUQ and FabSim3 frameworks.

3.3 Real-life application multiscale use cases

The developed techniques for distributed multiscale computing and VVUQ have been validated across a range of use cases. This section presents a selection of representative and realistic applications that benefited from the developed methods and software. Specifically, the first two examples demonstrate the practical application of Multiscale Computing Patterns, followed by several examples illustrating the use of VECMA Toolkit and mUQSA for UQ scenarios.

3.3.1 Cell-Based Blood Flow

Cell-Based Blood Flow is a tightly-coupled multiscale model that drops into the Extreme Scaling pattern. Coupled with MUSCLE2, this application includes two auxiliary submodels simulating continuous blood flow (implemented in Palabos - a parallelised Lattice Boltzmann Model) and a primary model of cell-based simulations (HemelB - an Immersed Boundary LBM). In case of Extreme Scaling pattern, the number of single-scale models is known in advance, thus the optimisation focusses on selecting a proper number of nodes and cores for balanced execution of primary and auxiliaries. The auxiliary submodels require little computation, but they may be time consuming, while the primary model is compute-intensive but scales efficiently to large core counts. Standard, singular execution of the application would be highly inefficient as the primary model would need to wait on auxiliaries. The Extreme Scaling pattern, as shown in [Alowayyed et al., 2019], interleaves two instances, so that the resources from the large allocation designated for handling a primary model do not have to wait idle until the auxiliaries finish their computations, but run a second instance of the primary. In a balanced ES pattern, it is usually optimal to run all single-scale models on the same resource. In the case of Call-Based Blood Flow application, naive scheduling yields only 0.35 resource efficiency, wasting over 2,800 core hours due to idle waiting. Interleaving reduces this waste to 1,000 core hours, doubling the efficiency [Alowayyed et al., 2019].

3.3.2 Binding Affinity Calculator

Binding Affinity Calculator (BAC) is an automated workflow for computing ligand–protein binding affinities via free energy calculations. It combines classical molecular dynamics (MD) with the molecular mechanics Poisson–Boltzmann surface area method. As discussed in [Alowayyed et al., 2019], to ensure reproducibility, 25 replica MD simulations need to be performed per calculation, as MD is highly sensitive to initial conditions. This makes BAC a clear example of the **Replica Computing** pattern. In order to enusre efficient execution of the replicas in a distributed infrastructure, the Multiscale Patterns software takes static and dynamic information about the underlying resources. For example, when the BAC application was executed in an environment composed of two HPC environments, namely SuperMUC and Eagle, the software needed to take into account specific restrictions of the former one as SuperMUC restricted users to have only eight concurrent jobs in a "general" queue. This limitation was not set on the Eagle HPC system. The Pattern-Driven Planner needed to determine the optimal strategy: whether to run all replicas on SuperMUC, which was generally faster, accepting its constraints, or to distribute replicas across multiple HPC systems. In practice, this may involve batching jobs to run a batch of 8 jobs at a time on SuperMUC and some other batch of jobs at a time on Eagle, and coordinating execution across sites. Figure 3.6 shows the theoretical computation times, derived from previously collected performance measurements, for different numbers of batches executed on SuperMUC in two scenarios: 80 replicas and 40 replicas. For the case with 80 replicas, the optimal performance is

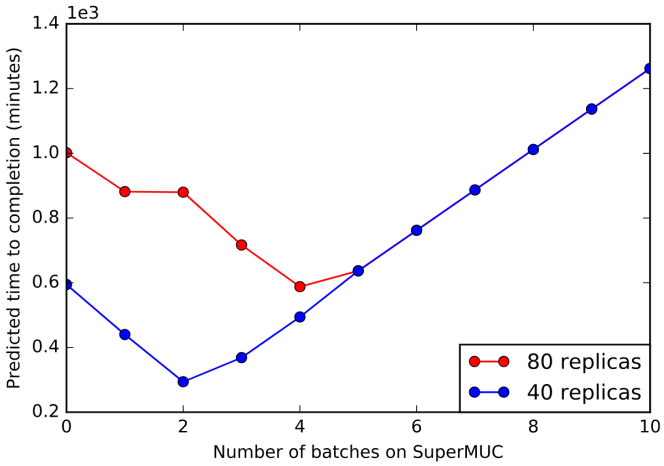


FIGURE 3.6: Theoretical time of running multi-replicas simulations across two HPC systems (SuperMUC and Eagle located in Germany and Poland respectively), as a function of number of batches (i.e., sets of 8 concurrent jobs) on SuperMUC. The remainder of the replicas are run as batches of up to 12 concurrent jobs on Eagle.

achieved with 4 batches of 8 jobs each run on SuperMUC and the rest of jobs on Eagle, whereas for 40 replicas, the best results are obtained with 2 batches run on SuperMUC and the rest on Eagle.

In-silico tests have also been performed, and their results differ slightly. It is caused by multiple additional factors, such as expected queuing time, that were not taken into account in the scheduling algorithms available at the time [Alowayyed et al., 2019]. The issue of missing information on queueing time was later addressed by the introduction of Queue-Time Estimation Service [Jancauskas et al., 2019].

3.3.3 Fusion

Thermonuclear fusion, a promising carbon-free energy source, requires simulations that capture multiscale plasma dynamics. In this process, a coupled workflow of equilibrium, turbulence, and transport models was assessed through uncertainty quantification. In this scenario, as presented in [Groen et al., 2021a], EasyVVUQ was launched within an allocation created by the QCG stack, and it used the EQI library to efficiently launch ensembles through QCG-PilotJob. The process, initiated via command-line tools, made it possible to automate execution and carry out a wide variety of experiments with different methods and models.

3.3.4 Air pollution

Precise prediction of air quality in urban environments requires large-scale simulations that couple meteorological and geophysical models on multiple scales. A major challenge is the lack of accurate emission databases, which should contain pollutant rates for point sources (e.g., industrial chimneys), line sources (e.g., road traffic), and area sources (e.g., household heating). To address these uncertainties, in the Urban Air Pollution model, ensembles of simulations were executed on HPC resources and analysed through automated workflows, as shown in [Groen et al., 2021a] and [Wright et al., 2020]. EasyVVUQ samples were run inside pre-allocated resources via EQI and QCG-PilotJob. High-level steering was enabled both via the command line using the QCG-Client tool and through a dedicated application scheme in QCG-Now, which simplified the workflow and made it more accessible to end-users.

3.3.5 Epidemiology

Epidemiological modelling is a crucial tool for understanding the dynamics of infectious diseases, testing intervention strategies, and guiding public health policy. This became particularly evident during the COVID-19 pandemic, when reliable forecasts were needed to support decision making. The Flu and Coronavirus Simulator (FACS) model [Mahmood et al., 2022] allowed the simulation of epidemic spread not only at the national, but also at the local level in city boroughs, such as those in London. To validate the code and analyse parameter sensitivities, ensemble simulations across multiple scenarios were executed in accordance with the discussion in [Groen et al., 2021a]. In this case, the executions were automated through FabSim3's FabCovid19 plugin that coordinated the processing of EasyVVUQ's algorithms with the use of QCG-PilotJob for the execution of ensembles.

In a similar integration scheme, which relies on FabSim's coordination, CovidSim [Ferguson et al., 2020], an epidemiological code used by UK decision makers, was evaluated and compared against both its early uncertainties-free results and the actual progression of the epidemic. The results of these studies are reported in [Edeling et al., 2021].

3.3.6 Renewable Energy Sources

The global challenges connected to climate change and the associated shift towards renewable energy sources, such as wind farms or solar plants, require precise weather forecasts aimed at estimating future energy production, as well as potential damages caused by extreme weather conditions, which are essential for both plant owners and distribution system operators (DSOs). The work presented in [Kulczewski et al., 2024] discusses the possibilities of the application of uncertainty quantification and sensitivity analysis to the multiscale RES framework based on two large-scale models, namely the Weather Research and Forecasting (WRF) Model [Powers et al., 2017], a mesoscale weather prediction system, and EUlerian LAGrangian (EULAG) [Kurowski et al., 2011], which runs on a local scale. In particular, the paper demonstrates how the web-native mUQSA platform, based on the VECMA toolkit and integrating its core components EasyVVUQ and QCG-PilotJob, has been used to automate and streamline the execution of UQ and SA on an RES-damage module, enabling efficient analysis of the impact of complex terrain on existing or planned overhead electrical networks.

3.3.7 Summary

The selected use cases illustrate the practical applicability of the developed solution in realistic scenarios. Specifically, the first two use cases highlight the benefits that Multiscale Computing Patterns provide to multiscale models, while the remaining four demonstrate the application of VECMA Toolkit components for UQ and SA, with the last of these showcasing mUQSA, a web-native platform that extends VECMAtk.

3.4 Evolution of techniques for execution of multiscale and VVUQ scenarios

The development of core methods and associated computational techniques, initially for multiscale simulations and later for VVUQ, progressed in a consistent way. It began with the creation of basic mechanisms to execute loosely coupled and tightly coupled workflows in a homogeneous, single-cluster environment. Building on this foundation, the MAPPER project significantly advanced the framework by enabling distributed computations. In the subsequent ComPat project, the concept

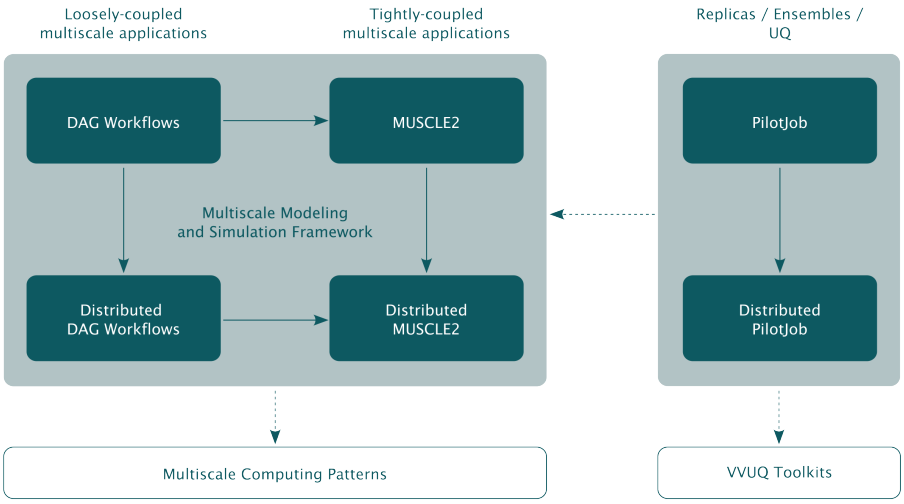


FIGURE 3.7: Evolution of key techniques for multiscale simulations and VVUQ. The Multiscale Modeling and Simulation Framework forms the foundation for Multiscale Computing Patterns, while Pilot Job executions play a critical role in VVUQ. Ultimately, Pilot Job can also be incorporated into MMSF.

of Multiscale Computing Patterns was introduced to improve the efficiency of typical multiscale execution scenarios. Finally, within VECMA, the Pilot Job paradigm was incorporated to support the highly efficient execution of large numbers of jobs on HPC resources, particularly in the context of VVUQ. This evolution is summarised in the form of a dependency graph presented in Figure 3.7.

More detailed information on individual techniques is presented in Table 3.3. In order to showcase their evolution, the techniques are presented in an incremental manner: for each technique, the listed benefits highlight the advances compared to the preceding approach, while the limitations point to the gaps that are addressed by subsequent developments.

Although the work presented here concentrated on multiscale simulations and VVUQ, the techniques developed are more widely applicable. Key examples include enabling distributed computations and supporting scenarios with numerous tasks within a single computing allocation through the pilot job mechanism, both of which are increasingly common in modern computational science.

3.5 Development and deployment of automation tools for the management of advanced computational workflows

In addition to developing core techniques for the efficient execution of advanced computational scenarios, particularly multiscale simulations and VVUQ, it was equally important to design intuitive, user-orientated tools that correspond to different general methods of interaction, such as command-line, graphical and web-based interfaces, to satisfy diverse users' needs and thus automate and simplify the creation and processing of computational scenarios. This section highlights progress in this area by presenting several tools developed, deployed and tested with the active involvement of the author of this dissertation and subsequently comparing them.

3.5.1 QCG-Client

Experienced researchers are typically familiar with Linux command line interfaces and local resource management systems. Although CLI tools are often harder to start working with and lack high-level automation, their universal nature is a considerable advantage. To support such users, a command-line QCG-Client tool has been developed. Thanks to its integration with the QCG-Broker service,

TABLE 3.3: Incremental evolution of multiscale and VVUQ execution techniques with their reference implementations, benefits, and limitations

Technique	Implementation	Application	Benefits	Limitations
DAG workflows	Local scripts, FabSim3, QCG	Loosely-coupled multiscale models	Easy to implement, simple BASH script can be sufficient	Lacks support for tightly-coupled models and heterogeneous resource requirements
Distributed DAG workflows	FabSim3, QCG	Loosely-coupled multiscale models	Support for multiscale models with heterogeneous requirements	Lack of support for tightly-coupled models
MUSCLE2 execution	MUSCLE2	Tightly-coupled multiscale models	Concurrent execution of many single-scale models	Lack of support for heterogeneous resource requirements
Distributed MUSCLE2 execution	MUSCLE2, MTO, QCG	Tightly-coupled multiscale models	Concurrent execution of many single-scale models with heterogeneous requirements	Poor efficiency for many typical scenarios
Multiscale Computing Patterns	MCP Software including; Pattern-driven planner, QCG Pattern-aware scheduler	Extreme Scaling, Heterogeneous Multiscale Computing, Replica Computing	Optimised execution of commonly used Multiscale Computing Patterns	Limited support for scenarios with a large number of submodels executions
Pilot Job execution	QCG-PilotJob	Scenarios consisting of numerous computational tasks, e.g., Replica Computing, UQ	High efficiency of execution of numerous tasks inside a single allocation, allocation-contained execution	Static size of allocated resources
Distributed Pilot Job execution	QCG-PilotJob 2.0*	Dynamic or highly resource-demanding scenarios with numerous tasks	High flexibility, including adaptable size of reserved resources	The need for centralized queuing and controlled allocation management

the tool enables the submission of different types of job, including complex workflows, parameter sweep tasks, and array jobs, on single or multiple clusters. Importantly, the interface resembles that of well-known LRMs, such as PBS, which means that the learning curve is usually not a barrier. The universal nature of CLI tools enables them to integrate relatively easily with other components. This was demonstrated through the integration of QCG-Client with FabSim3, aimed at providing advanced scheduling capabilities for FabSim3 use cases; for more details, see [Groen et al., 2021a].

3.5.2 QCG-Now

QCG-Now and its predecessor QCG-Icon, are graphical desktop programmes developed to enable the submission and management of jobs on HPC resources directly from the personal computers of users. QCG-Now supports user-friendly definition of input parameters for computational tasks (see Figure 3.8), automatic data sending and receiving, and allows users to track the execution progress directly from its graphical interface, as demonstrated in [Groen et al., 2021a]. Although less flexible than command-line tools, QCG-Now may be a better choice for the execution of well-defined

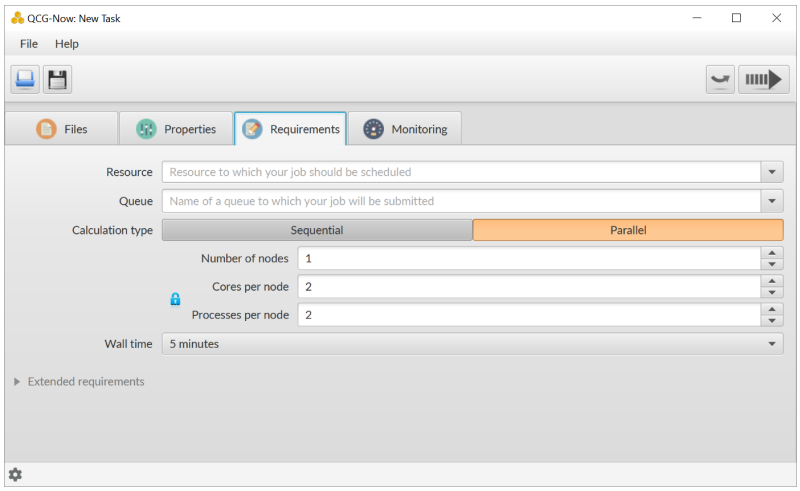


FIGURE 3.8: Example view of the QCG-Now graphical interface showing the input parameter specification for a sample computational task

multiscale or UQ scenarios. In such cases users may benefit from simplicity of use, higher-level of automation, and possibility of tracking the progress of long-lasting executions, such as those represented by EasyVVUQ and pilot job workflows. The undisputed benefit of the solution is the functionality that it provides for integration with other desktop programmes and the local filesystem. For example, QCG-Now can be configured in a relatively easy way as an executor for domain-specific graphical applications, enabling transparent execution of intensive computations on remote large-scale resources.

3.5.3 QCG-Portal

The evolution of web technologies has enabled significant advances in the development of user-level tools, culminating in the creation of the comprehensive QCG-Portal, accessible directly through a Web browser (see Figure 3.9). One of its key advantages is the ability to execute computational tasks without requiring any local software installation. Beyond this, the portal provides several user-orientated features that surpass earlier desktop applications. The first of them is a high degree of customisation: unlike QCG-Now, where the interface was largely generic, QCG-Portal makes it easy to design use-case-specific interfaces for input parameter specification, application monitoring, and results presentation. For enhanced data handling, QCG-Portal is integrated with the IBIS Data Management System, providing seamless access to remote storage spaces for both users and computational jobs. Finally, the portal exposes an REST API, enabling straightforward integration with external software components, thereby streamlining the inclusion of large-scale computations in high-level scientific or business processes, as discussed in [PCSS, 2023].

3.5.4 mUQSA

mUQSA is a web-based graphical tool designed to facilitate the definition and automate the execution of common UQ and SA scenarios. Provided as a Software as a Service (SaaS) solution, as described in [Bosak et al., 2025], it can be employed by computational model developers and users who wish to quickly assess credibility or explore parameter significance, as well as by less experienced researchers seeking to learn the fundamentals of UQ. The tool provides step-by-step guidance in defining UQ / SA scenarios, automates their execution using EasyVVUQ and QCG-PilotJob, and presents the results in a clear and visually appealing manner, as shown in Figure 3.10.

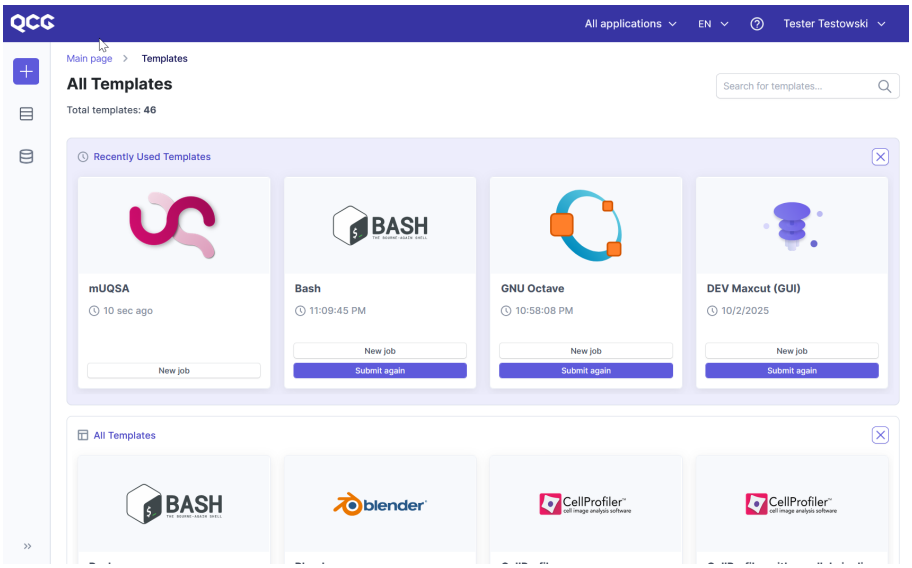


FIGURE 3.9: One of the QCG-Portal views, displaying the list of available application templates

FIGURE 3.10: Results presentation in mUQSA, featuring interactive charts and tables. mUQSA is a dedicated graphical user interface built upon the QCG-Portal framework.

Built on top of the core mechanisms of QCG-Portal, mUQSA illustrates how custom application interfaces can be developed: While QCG-Portal handles generic services such as computational task management and task presentation views, mUQSA focusses on tailoring the interface to the specifics of UQ / SA applications.

3.5.5 Comparative analysis of user-level automation tools

For a clear presentation of the differences between the user-level tools described or mentioned above, Table 3.4 offers a comparative overview of their features. The overview also takes into account the standard LRMS command-line interface, positioning it alongside the other tools to highlight contrasts. Although the table primarily juxtaposes existing concrete tools, it also illustrates the general differences between the types of access methods they implement. To complement this comparison, in Figure 3.11 a radar graph is provided to visually display the differences in automation and flexibility of the tools.

Table 3.4: Comparison of different user-level automation tools and standard LRMS interface

Tool	Access method	Deployment	Domain-oriented	Customisability	Data mgmt automation	Workflow support	Multi-cluster access	Distributed	Automation level	Flexibility level
LRMS	command line	Linux (cluster)	no	low	low	medium	no	no	3.6	5.3
QCG-Client	command line	Linux (local)	no	low	medium	high	yes	yes	6.3	7.0
QCG-Now	desktop	Win/OSX/ Linux	no	low	high	low	no	no	8.1	6.8
QCG-Portal	web	web	no	high	high	medium	no	no	9.1	10.0
mUQSA	web	web	yes	low	high	low	no	no	10.0	4.9

In the context of this analysis, automation is a metric that shows how the tool helps in the execution of scenarios for which it was created, and the flexibility metric shows how easy it is to adjust the tool to different scenarios or specific needs. For the calculation of automation and flexibility levels, custom formulas have been proposed that calculate weighted sums of individual parameters that were taken into account in comparison. Although partially subjective, the results clearly show the progression that occurred over the last dozen or so years.

Both command-line tools, i.e., the basic LRMS interface and QCG-Client, are characterised by relatively low automation levels. Their users must perform many tasks manually, which is often difficult and time consuming. At the same time, command-line interfaces are typically generic in nature and expose the full range of functionality offered by the underlying services. This is also the case for the tools under analysis, with QCG-Client, in particular, ranking highly in terms of flexibility.

A higher level of automation than in the case of command-line tools is demonstrated by a desktop programme, QCG-Now. Among others, QCG-Now provides a user-friendly GUI and takes care of data management, making its usage much more accessible to new users, and streamlining the execution of common scenarios. However, the generic interface of the tool limits the possibilities of its adjustment to meet the expectations of the target users. QCG-Now is also less flexible than

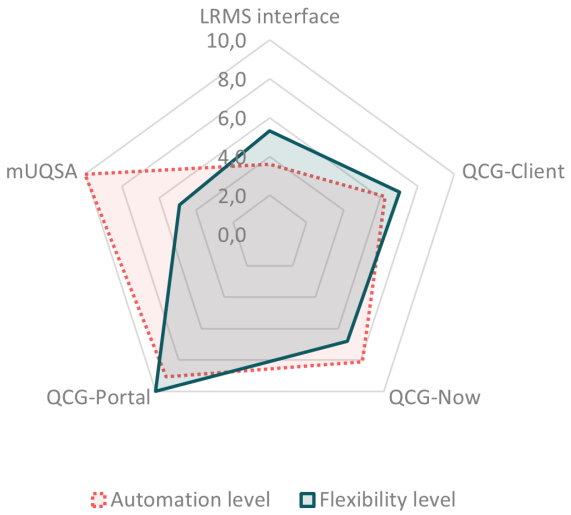


FIGURE 3.11: Comparison of flexibility and automation levels across different tools

its command-line counterparts as it supports popular, but only certain schemes of application executions: implementation of a new scheme may require updates in the source code of the tool.

Many of the issues mentioned above are addressed by QCG-Portal, which, through its generic core and the concept of application templates, provides a high degree of automation while maintaining strong flexibility. Technically, an application template serves as an application-specific configuration that can be created relatively easily by a system administrator. These templates allow for extensive customisation to meet individual requirements, both at the execution layer and at the interface layer. Consequently, administrators can define the exact commands needed to run a single task or an entire workflow on a cluster, and design the graphical interface to match the expectations of different stakeholders.

Furthermore, QCG-Portal supports the integration of externally developed application interfaces, greatly expanding its customisation options. A good example is mUQSA, where a custom wizard has been created for input specification, while results are presented through a dedicated, interactive webpage. Consequently, through the interface adjusted to the functionality offered, mUQSA further increases the level of automation. mUQSA is clearly not a universal tool and, therefore, offers limited flexibility, but this should not be regarded a disadvantage, as its purpose is to streamline the execution of specific scenarios, that is, UQ and SA.

3.5.6 Summary

A portfolio of high-level user-orientated automation tools has been developed to complement the core mechanisms for distributed execution of multiscale models and VVUQ described in this dissertation. These tools, ranging from generic command-line interfaces and desktop applications to specialised web portals, offer varying levels of automation and flexibility. Collectively, this suite enables the framework to accommodate diverse user requirements and to support a broad spectrum of execution scenarios.

Chapter 4

Conclusions and Summary

Multiscale simulations and uncertainty quantification (UQ) represent two computational domains characterised by exceptionally high demands for computing power. Both fields also require highly flexible scheduling and execution mechanisms capable of managing sophisticated workflows and ensuring efficient utilisation of large-scale, often distributed, often heterogeneous, and high-performance environments. Addressing these requirements requires advanced automation, systematic workflow management, and generic software solutions that can abstract the complexity of such complex multiscale computing simulations.

This dissertation introduces a comprehensive methodology and a toolbox of applied research solutions designed to effectively support scientists and engineers in the development, evaluation, and large-scale execution of multiscale models on HPC and distributed computing infrastructures. The proposed holistic approach defines various methods and software components for multiscale computing, structured around three generic pillars: the Multiscale Modeling Language (MML), the Multiscale Modeling and Simulation Framework (MMSF), and the Multiscale Computing Patterns (MCPs). Together, these components enable the modelling, design, and efficient execution of complex multiscale simulations across heterogeneous and geographically distributed HPC environments. A key component is the execution layer, which combines efficiency with high flexibility through mechanisms for advance reservation, co-allocation of HPC resources, and coordinated cross-cluster execution of multiscale models.

The presented concept of Multiscale Computing Patterns (MCPs) extends the automation capabilities of MMSF by providing reusable execution templates that optimise resource utilisation and computational efficiency for common classes of multiscale problems. These patterns encapsulate best practices for orchestrating heterogeneous submodels and managing their coupling relationships. Complementing these capabilities, the MUSCLE Transport Overlay (MTO) has been developed as a dedicated communication daemon that enhances data transfer performance in tightly coupled models implemented using MUSCLE2, further improving the scalability and responsiveness of distributed simulations.

In parallel, the dissertation explores challenges associated with verification, validation, and uncertainty quantification (VVUQ) of complex computational multiscale models, an area that plays a crucial role in enhancing reliability and trust in simulation outcomes. To this end, and thanks to some relevant research projects, in particular MAPPER, ComPat and most importantly VECMA, it introduces the VECMA Toolkit, which provides a set of interoperable methods and tools facilitating the integration of VVUQ processes into multiscale workflows. Among these, the Pilot Job paradigm, implemented as the lightweight and scalable QCG-PilotJob system, enables flexible and efficient execution of computationally demanding UQ studies. Extensive large-scale

testing campaigns have demonstrated its robustness, portability, and performance. Building upon these capabilities, QCG-PilotJob has been successfully integrated as an execution engine within the EasyVVUQ and FabSim3 frameworks, effectively supporting distributed and parallel execution of complex UQ workflows on HPC systems.

A portfolio of high-level user-orientated automation tools has also been developed and tested experimentally to complement the core execution mechanisms. This toolbox, consisting of command-line interfaces, desktop applications, and specialised web portals, provides varying degrees of automation and flexibility, enabling users to configure, execute, and monitor computational experiments across diverse HPC systems with minimal effort. Collectively, these tools enhance accessibility and reproducibility, allowing both expert and non-expert users to efficiently exploit large-scale and HPC resources.

The key research topics addressed in this dissertation focus on the automation and simplification of complex multiscale computational workflows, specifically targeting three principal challenges:

- synchronisation and coordination of distributed multiscale computing workflows;
- multi-level scheduling strategies for large-scale uncertainty quantification studies; and
- intuitive user-level tools for seamless execution of computational experiments on remote resources.

The practical applicability of the solutions proposed in this dissertation is demonstrated through multiple multiscale real-world applications. The first set of applications showcases the benefits of Multiscale Computing Patterns for distributed multiscale modelling, while the remaining examples illustrate the successful application of the VECMA Toolkit components, particularly QCG-PilotJob and EasyVVUQ, as well as the mUQSA platform, to advanced uncertainty quantification and sensitivity analysis workflows. Extensive performance evaluations confirm the efficiency, scalability, and portability of the developed methods, demonstrating their potential to advance the state-of-the-art in large-scale, automated multiscale, and uncertainty-aware computing.

Today, in the era of the AI technology revolution and one step away from the quantum advantage, justified questions may arise whether the developed solutions remain valuable. Undoubtedly, rapid technological progress continues to open new opportunities and naturally shifts the scientific community's attention toward emerging computational paradigms. However, AI and quantum computing are unlikely to fully replace classical methods in the near future [Jiao et al., 2024]. Consequently, multiscale simulations and conventional UQ techniques will continue to play a key role, progressively enhanced and complemented by data-driven models and quantum algorithms.

Furthermore, many of the approaches presented in this dissertation, originally designed for classical HPC, can be directly adapted to address the challenges introduced by AI and quantum technologies. For example, in numerous multiscale applications, selected submodels can be replaced with AI-generated surrogates or quantum algorithms [Ma et al., 2023], with minimal impact on the overall computational framework. This integration aligns naturally with the principles of distributed and heterogeneous multiscale computing, where individual submodels may require distinct types of computational resources. Traditionally, these resources were CPU or GPU nodes; now, the landscape simply expands to include specialised AI accelerators and various types of emerging quantum processing units (QPUs).

However, a similar direct mapping is not always feasible for UQ methods. Techniques that perform effectively for physics-based models (such as PCE or SC) tend to be unsuitable for AI- or quantum-based systems. However, new UQ methodologies specifically tailored for these paradigms are emerging, frequently adapted from existing approaches. [He et al., 2025] Although they differ in

formulation, they generally exhibit comparable or even greater computational complexity, in both scale and nature, and therefore require sophisticated management strategies inspired and adapted from their classical counterparts.

Consequently, it should be stated that developed methods keep validity not only for usage in fields and in a scope for which they were designed but that they are also portable to handle new challenges defined by an ongoing revolution.

In practice, it means that the proposed solutions need to be constantly improved to achieve the requested quality and adapted to meet the emerging expectations, and ultimately to constitute a comprehensive platform for the trusted heterogeneous multiscale simulations. From this perspective, the following development paths for the future are considered.

Automation of multiscale and UQ workflows. This is an obvious continuation of the activities described in this dissertation. The introduced methods and software components implemented require further development in order to improve ease of use and computational efficiency.

Integration of new computational approaches. The majority of the described solutions have been targeted at HPC processing. New computational paradigms, such as Cloud or Quantum Computing, need to be included more explicitly into the architecture and the overall proposed methodology.

Modular and cloud-ready services. By leveraging cloud-native approaches, the service layer of the proposed solutions can facilitate the adoption of developed mechanisms and enhance the flexibility, scalability, and portability of their usage, including hybrid executions across classical HPC, cloud, and container-based resources.

Application-driven services and tools. With core services and tools achieving the required level of technical maturity, it becomes possible to develop solutions targeted at specific applications and domain-orientated use cases.

Finally, in addition to the development paths outlined above, it is important to recognise that the concepts and solutions introduced in this dissertation, although demonstrably effective, are still relatively unfamiliar to the broader scientific and engineering communities. Therefore, active efforts in dissemination and community engagement will be essential to enhance their visibility, foster adoption, and maximise their long-term impact. In this regard, the continued development of intuitive high-level user interfaces and advanced automation mechanisms will play a pivotal role in achieving these objectives by increasing the accessibility, usability, and efficiency of the underlying technologies, thus broadening their reach across diverse user groups.

Knowledge Dissemination and Promotion

Alongside the development of approaches for modelling and assessing the confidence of heterogeneous multiscale simulations in HPC environments, the author of this dissertation carried out a number of activities aimed at dissemination and promotion of the solutions.

The first group of activities relates to the author's contribution to the preparation of numerous scientific papers, published in recognised journals such as Journal of Computer Science [Borgdorff et al., 2014], Future Generation Computer Systems [Alowayyed et al., 2019], Advanced Theory and Simulations [Wright et al., 2020] and Nature Computational Science [Edeling et al., 2021]. In addition, the author delivered several presentations at international conferences, including ISC [ISC, oing], HiPEAC [HiPEAC, oing], and PPAM [PPAM, oing]. For more than a decade, he has served as co-chair of the Multiscale Modelling and Simulation workshop series [Krzhizhanovskaya et al., 2015][MMS, 2025], co-hosted with the ICCS conference [ICCS, oing], and has co-organised numerous other events, like workshops, trainings, and hackathons devoted to the popularisation of knowledge and promotion of developed methods and solutions.

Many of the software components developed or co-developed by the author have been released as open-source, with some made available for public collaboration via the GitHub platform (e.g. EasyVVUQ, QCG-PilotJob, EQI). Several of the components have also been integrated into VVUQ-orientated toolkits, namely VECMA Toolkit [Groen et al., 2021b] and SEAVEA Toolkit [SEAVEA Project, 2021], which are designed to provide comprehensive modular frameworks that can be adapted to various application scenarios.

Furthermore, the author contributed significantly to the preparation of tutorials, handbooks, Jupyter notebooks, textual guides, and other instructional resources, as well as to the development of dedicated web pages showcasing the proposed solutions. As co-leader of the Multiscale Laboratory within the PIONIER-LAB project [Multiscale Laboratory, 2023], he played a leading role in establishing a range of services designed to promote and facilitate multiscale computations and uncertainty quantification for both scientific and industrial communities in Poland and abroad.

Publication Reprints

The dissertation consists of the introductory section and the following nine original publications:

Publication [P1]

Borgdorff, J., Bona-Casasa, C., Mamonski, M., Kurowski, K., Piontek, T., **Bosak, B.**, Rycerz, K., Ciepiela, E., Gubala, T., Harezlak, D., Bubak, M., Lorenz, E., Hoekstra, A.: *A distributed multiscale computation of a tightly coupled model using the Multiscale Modeling Language.* In Procedia Computer Science 9, 596 – 605 (2012). https://doi.org/10.1016/j.procs.2012.04.064

Ministry points / conference: 140

Contribution of authors:

- Joris Borgdorff
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, MML, ISR3D, MUSCLE, Results, Conclusions and discussion)
- Alfons G. Hoekstra
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Conclusions and discussion)
  - Review and editing of the paper
- Carles Bona-Casas, Eric Lorenz
  - Coauthorship of the text of publication (Sections: MML, ISR3D, MUSCLE)
- **Bartosz Bosak**, Mariusz Mamonski
  - Coauthorship of the text of publication (Sections: Cross-cluster execution with QosCosGrid, Results)
- Krzysztof Kurowski, Tomasz Piontek
  - Coauthorship of the text of publication (Sections: Cross-cluster execution with QosCosGrid, Conclusions and discussion)
- Katarzyna Rycerz, Eryk Ciepiela, Tomasz Gubala, Daniel Harezlak
  - Coauthorship of the text of publication (Sections: High level composition and execution tools, Results)
- Marian Bubak
  - Coauthorship of the text of publication (Sections: High level composition and execution tools, Conclusiosn and discussion)
  - Review and editing of the paper

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 596 – 605

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

A distributed multiscale computation of a tightly coupled model using the Multiscale Modeling Language.

Joris Borgdorff[a,*], Carles Bona-Casas[a], Mariusz Mamonski[b], Krzysztof Kurowski[b], Tomasz Piontek[b], Bartosz Bosak[b], Katarzyna Rycerz[c,d], Eryk Ciepiela[d], Tomasz Gubala[a,d], Daniel Harezlak[d], Marian Bubak[a,c], Eric Lorenz[a], Alfons G. Hoekstra[a]

[a]*Section Computational Science, University of Amsterdam, the Netherlands*
[b]*Poznań Supercomputing and Networking Center, Poznań, Poland*
[c]*Instutute of Computer Science, AGH University of Science and Technology, Kraków, Poland*
[d]*CYFRONET, AGH Univerisity of Science and Technology, Kraków, Poland*

Abstract

Nature is observed at all scales; with multiscale modeling, scientists bring together several scales for a holistic analysis of a phenomenon. The models on these different scales may require significant but also heterogeneous computational resources, creating the need for distributed multiscale computing. A particularly demanding type of multiscale models, tightly coupled, brings with it a number of theoretical and practical issues. In this contribution, a tightly coupled model of in-stent restenosis is first theoretically examined for its multiscale merits using the Multiscale Modeling Language (MML); this is aided by a toolchain consisting of MAPPER Memory (MaMe), the Multiscale Application Designer (MAD), and Gridspace Experiment Workbench. It is implemented and executed with the general Multiscale Coupling Library and Environment (MUSCLE). Finally, it is scheduled amongst heterogeneous infrastructures using the QCG-Broker. This marks the first occasion that a tightly coupled application uses distributed multiscale computing in such a general way.

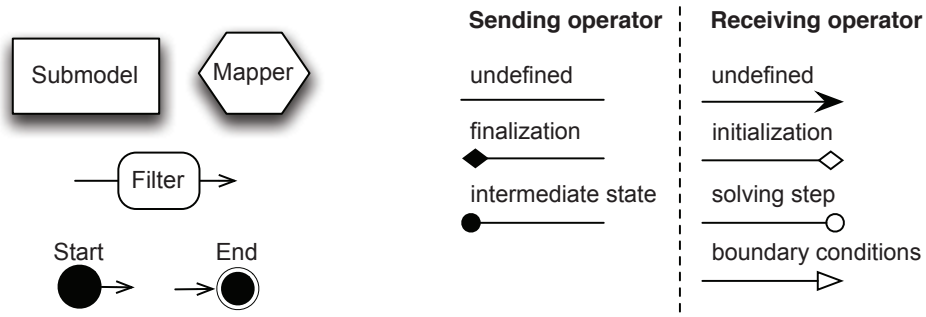
Keywords: multiscale modeling, distributed multiscale computing, MML, multiscale modeling language, in-stent restenosis, MUSCLE, QCG-Broker, Gridspace

1. Introduction

Nature is observed on an abundance of scales, and each of those scales may yield additional insight in its workings. Consequently, scientists from a range of disciplines are now using multiscale modeling to connect models and data from different scales in a holistic approach to understand and control nature [1–3].

*Corresponding author
Email addresses: J.Borgdorff@uva.nl (Joris Borgdorff), C.BonaCasas@uva.nl (Carles Bona-Casas), mamonski@man.poznan.pl (Mariusz Mamonski), krzysztof.kurowski@man.poznan.pl (Krzysztof Kurowski), piontek@man.poznan.pl (Tomasz Piontek), bbosak@man.poznan.pl (Bartosz Bosak), kzajac@agh.edu.pl (Katarzyna Rycerz), e.ciepiela@cyfronet.pl (Eryk Ciepiela), t.gubala@cyfronet.pl (Tomasz Gubala), d.harezlak@cyfronet.pl (Daniel Harezlak), bubak@agh.edu.pl (Marian Bubak), E.Lorenz@uva.nl (Eric Lorenz), A.G.Hoekstra@uva.nl (Alfons G. Hoekstra)

doi:10.1016/j.procs.2012.04.064



(a) gMML elements (b) gMML edge terminals

Figure 1: The elements that make up gMML. In (a) the computational elements are shown, in (b) are the edge terminals corresponding to the current operation in the submodel.

Meanwhile, there has been some effort to formalize this type of modeling [4–6], often proposing methods to subdivide a multiscale model into multiple single scale models.

From a computational point of view, every submodel of a multiscale model may have different, even contradictory, hardware and software requirements. For example, take a model with one submodel using a highly-parallel fluid dynamics flow solver, requiring a cluster with Infiniband interconnects; another submodel, a cellular automaton parallelized with OpenMP, performing best on a large SMP machine; and finally, a submodel using a GPU-powered agent based modeling toolkit. Moreover, two submodels might require different specialized proprietary software, while having no sites available with licenses for both. This situation is only exacerbated if the multiscale model is tightly coupled, requiring frequent communication between its submodels. Such a case demands distributed multiscale computing, as was recognized by five scientific communities behind the MAPPER project[1].

From the biomedical domain, the multiscale three-dimensional model of in-stent restenosis (ISR3D) is an example of a tightly coupled application with heterogeneous submodels [7]. It models a stenosed blood vessel after stenting to determine if and how a restenosis could occur. The two-dimensionsonal version, ISR2D, already has published results [8], but ISR3D is far more computationally demanding and requires distributed multiscale computing.

This contribution shows how a tightly coupled multiscale model can be described, specified, and executed on distributed resources. First, the application ISR3D is described with the high-level multiscale modeling language (MML) [5, 9, 10]. Once this is done, it can be specified using the MAPPER Memory (MaMe), Multiscale Application Designer (MAD), after which the application is managed by GridSpace Experiment Workbench (EW) [11, 12]. At the same time, ISR3D is implemented using the multiscale coupling library and environment (MUSCLE) [13], which handles the communication between submodels. Finally, the application is scheduled on distributed resources using the QosCosGrid stack [14], including EGI, PRACE, and a local resource. To our knowledge, this is the first time that a tightly coupled multiscale application had a distributed execution in such a general and automated way. The case of ISR3D forms a validation point for the aforementioned MML, the tools to convert MML into an executable experiment, and for distributing a tightly coupled multiscale model.

2. Multiscale modeling language (MML)

To bridge the gap between multiscale modelers and execution environments, the multiscale modeling language (MML) was conceived [5, 9, 10]. This language introduces a well-defined multiscale modeling terminology that can be used to describe, verify, analyze, and execute a multiscale model. The foundations of MML are clearly defined in [5], but will be summarized here.

[1] http://www.mapper-project.eu/

First, it asks from multiscale modelers that they decompose a multiscale phenomenon in multiple single scale phenomena. These phenomena form the basis for single scale models or submodels, while their interactions are grounds for couplings between the submodels. Combined, these constitute a multiscale model. This step is aided by making a scale separation map (SSM) [5, 6], showing the scales and interactions of the phenomena involved and the extent of scale separation they have.

The next step, after the single scale models are clear, is to analyze the coupling topology of the multiscale model. A coupling topology describes how a multiscale model is coupled by explicitly creating a directed graph with submodel instances as nodes, couplings as edges, and number of times that the coupling is invoked as edge weights. When considering how to execute a multiscale model, a few properties of the coupling topology are of interest: whether it contains cycles or not; whether it has fixed edge weights, or fixed number of synchronization points; and whether there is more than one instance per submodel and whether that amount is fixed.

When a coupling topology is cyclic, it means that there is a feedback loop within the model and that certain submodels will be revisited; we call this a tightly coupled model. In a loosely coupled model, without a cycle, a submodel can be considered as finished when it has sent its information. However, in a tightly coupled model, execution software will need to keep some submodels waiting while others compute.

If the number of submodel instances is dynamic, the execution software and the model will have to communicate about how many submodel instances should be created. When the number of synchronization points is dynamic, execution software needs to know when a submodel is needed or finished. In MML, the coupling topology itself needs to be explicitly specified but its properties can be deduced. It does not specify how execution software should communicate with a multiscale model, rather, it leaves this to the implementation of that software.

Although the coupling topology gives a few interesting properties, for a well-defined and full runtime model a more precise specification is necessary. With MML it is possible to specify submodels and submodel instances but also their scale, computational requirements, and implementation details. Couplings are made explicit using the concept of conduits that bind to specific ports of submodels. Submodels should not be aware of other submodels, rather, data messages between submodels are manipulated by applying user-defined conduit filters to conduits. For distributing or collecting messages so-called fan-out and fan-in mappers are used. With these elements MML includes most features for a software architecture description.

For human interaction, MML has a graphical representation called gMML. This features the elements listed above, as shown in Figure 1, but does not contain any implementation details or information on scales. It is useful for composing or communicating the architecture of a multiscale model.

For machine interpretation, the XML format xMML captures these features, but also a wide range of metadata. This includes scale information, possible parameter settings, a datatype system, binding ports of submodels and mappers, implementation details such as number of cores needed per submodel, but also descriptive and documentation facilities. In contrast to gMML, xMML can be automatically processed and it acts as an exchange format of a model.

Once a multiscale modeler has implemented a model and fully described it with MML, it is possible for software to verify, analyze, and execute it.

3. A three-dimensional in-stent restenosis model (ISR3D)

Coronary heart disease (CHD) causes about 1.9 million deaths per year in Europe, making it Europe's most common cause of death [15]. The most common expression of CHD is arteriosclerosis, a thickening and hardening of blood vessels due to the build-up of atheromatous plaque. The thickening, causing a significant decrease in luminal area of the blood vessel, is called a stenosis, and has the common intervention of stent-assisted balloon angioplasty. In this intervention, a balloon is inserted in the blood vessel and inflated at the stenosis, causing the stent to be placed at that point. The stent acts as a strut or scaffold after the operation, compressing the plaque and keeping the lumen open. However, in some cases, this treatment results in in-stent restenosis (ISR), an excessive regrowth of tissue due to the injury caused by the stent deployment [16, 17]. The precise factors causing in-stent restenosis are yet unknown, although there have been multiple suggestions.

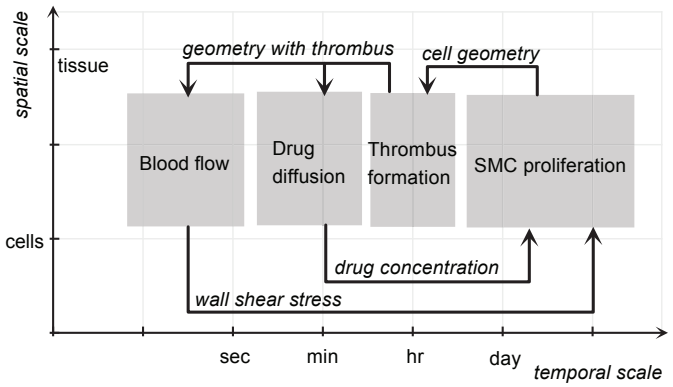


Figure 2: The scale separation map (SSM) of the ISR model, containing four submodels: blood flow, drug diffusion, thrombus formation and smooth muscle cell proliferation. The vertical axis is the logarithmic spatial size, the horizontal axis the logarithmic temporal size.

The three-dimensional model of in-stent restenosis (ISR3D) is a tool designed to explore which factors could be the main contributors to the formation of the restenosis. To this end, it models the part of a blood vessel where a stent has been placed. After evaluating the processes involved in in-stent restenosis [18], ISR3D is built on the hypothesis that smooth muscle cell proliferation drives the restenosis, and is in turn affected most heavily by wall shear stress of the blood flow and by growth inhibiting drugs diffused by a drug-eluting stent. With the model, the effect of different drug intensities, physical stent designs, and effects of wall shear stress can be evaluated. ISR3D is preceded by a two-dimensional model of in-stent restenosis (ISR2D) which has a similar model architecture [7] and has published results [8]. However, ISR2D is inherently limited by its two-dimensional design, which could not account for a full stent design, realistic cell growth, or exact blood flow. On the other hand, ISR3D requires far more computation; both cell proliferation and blood flow calculation are an order of magnitude more expensive in 3D.

When analyzed from a multiscale modeling perspective, ISR3D consists of four single scale models or submodels: smooth muscle cell proliferation (SMC), thrombus formation (Blob), blood flow (BF), and drug diffusing from a drug-eluting stent (DD). Each of these models act on the same spatial scale, but they all exhibit temporal scale separation, as can be seen on the scale separation map (SSM) of ISR3D in Figure 2.

Interactions in ISR3D are initiated by SMC, which initializes its cell placement and cell metadata. Then, for each iteration of the SMC, it sends the cell and stent geometry to Blob, which calculates if any thrombus formation takes place due to back-flow of the blood. Blob sends the modified geometry so that the blood flow and drug diffusion can be calculated. Afterwards, the drug diffusion and wall shear stress is mapped to the individual smooth muscle cells and SMC calculates the next iteration of smooth muscle cell proliferation. This loop continues a fixed number of times, in the order of magnitude of a few thousand, after which the model exits.

The coupling topology of ISR3D is tightly coupled since it contains a cycle, seen in the SSM. Each submodel has one instance; and the number of iterations or synchronization points is fixed by a parameter. From an computational point of view, the case of ISR3D is only complex in that it is tightly coupled.

The full specification of ISR3D is shown as gMML in Figure 3. The MML specification of ISR3D contains a few mappers not mentioned above, which do simple data transformations but are necessary to ensure that the different single scale models are not aware of other submodels and their internal representation or scales. In practice, these mappers also have a computational cost and should be treated appropriately.

The submodels used in ISR3D are heterogeneous: implemented with programming languages Java, C++, and Fortran; serial and well parallellized; and with particle and grid-based domains. Each of the submodels and mappers are custom made, with the exception of BF, which uses the Palabos Lattice Boltzman simulator[2]. A detailed listing of the implementation of the submodels is given in Table 1.

[2] `http://www.palabos.org`

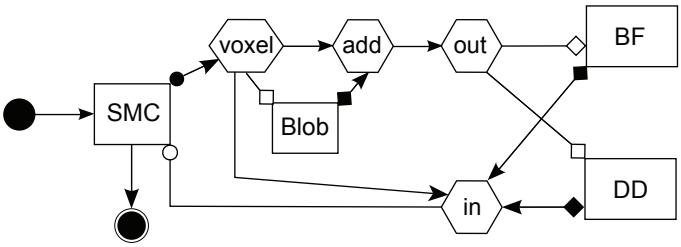


Figure 3: The gMML of the ISR3D model described in Section 3. The meanings of the elements are described in Figure 1. The in and out hexagons are fan-in and fan-out mappers, distributing and collecting data to and from DD and BF.

Table 1: Runtime statistics of different submodels. The test environment is specified in Table 2, then an order of magnitude runtime per iteration and memory used. The mappers are listed as a combined cost.

Submodel	Test environment	Runtime/iter.	Memory	Parallel	Language
BF	Huygens	10 min.	4 GB	extremely, using MPI	C++
DD	Mavrino	10 sec.	100 MB	up to 4 cores, using threads	Java
SMC	Reef	10 min.	200 MB	no	C++
Blob	Reef	2 min.	50 MB	no	Fortran
Mappers	Reef	30 sec.	500 MB	no	Java

4. Software

4.1. MUSCLE

Implementing a multiscale model in a modular way is possible in several coupling environments; due to a close compatibility with MML we have chosen to use the multiscale coupling library and environment (MUSCLE) [13] to implement ISR3D with. For a multiscale model, MUSCLE is in charge of handling communication between different submodels. As such, submodels and conduits are explicitly defined in MUSCLE, as are conduit filters. Its core is programmed in Java but it also supports C, C++, and Fortran. Mappers can also be implemented in MUSCLE but are not yet available as a separate entity. MUSCLE is currently being improved to support more elements of MML, making it more flexible towards scheduling, and support more high-performance computing necessities such as OpenMP, MPI, and GridFTP.

4.2. High level composition and execution tools

To facilitate MML-based composition and execution of multiscale applications such as ISR3D, a set of supporting tools have been developed, depicted in Figure 4. First, MAPPER Memory (MaMe)[3] is a semantics-aware persistence store to record MML specifications of submodels and their scales. The information from MaMe is then fetched by the Multiscale Application Designer (MAD)[4] – a user friendly visual

[3] `http://gs2.mapper-project.eu/mame`
[4] `http://gs2.mapper-project.eu/mad`

Table 2: Computational aspects of the machines that are referred to in the text. Administrative aspects are listed in Table 3. The number of cores and the amount of memory is listed per node.

Name	Processor	Clock speed	Cores	Memory	Batch System	Middleware
Huygens	IBM Power6	4.70 GHz	32	128 GB	LoadLeveler	UNICORE
Reef	Intel Xeon	2.40 GHz	8	16 GB	Torque+Maui	QCG-Computing
Mavrino	Intel Xeon	2.66 GHz	4	16 GB	SGE	QCG-Computing

Table 3: Administrative information on the resources described in the text.

Name	Provider	Location	Infrastructure
Huygens	SARA	Amsterdam, The Netherlands	PRACE Tier-1
Reef	PSNC	Poznań, Poland	EGI (PL-Grid)
Mavrino	University College London	London, England	Campus Resource

composition tool that can connect single scale models to form multiscale simulation. MAD can transforms a high-level MML description into an executable experiment that contains a MUSCLE configuration file and can be executed in the GridSpace Experiment Workbench (EW).

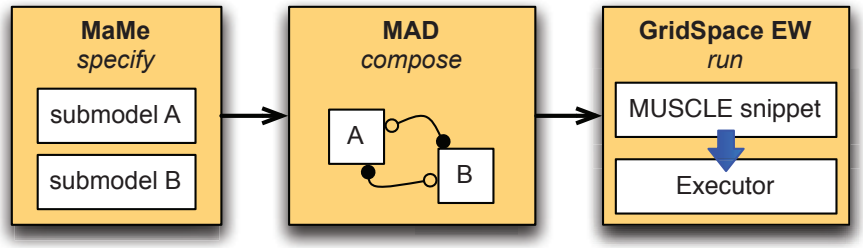


Figure 4: Multiscale programming and execution tools. MAPPER Memory (MaMe) registers information about MML submodels and mappers; Multiscale Application Designer (MAD) supports a user in composing simulation from those submodels and transforms MML into an executable experiment executed in GridSpace Experiment Workbench

MaMe is based on the idea of semantic integration [19]. It supports the exchange and reuse of MML specifications by other tools via a REST interface, but also provides a web interface for human users.

MAD supports application composition which is implemented as a sequence of drag-and-drop operations on graphical representations of MaMe components. On a conceptual and visual level, it is used to create gMML. When connections are created between the nodes MAD is able to perform various export procedures including xMML and the GridSpace executable format. Exported xMML contains MAD annotations about the positions of the elements in the MAD tool, so that when importing xMML, the visual composition persists.

The GridSpace Experiment Workbench[5] (EW) [11, 12] supports execution and result management of infrastructure independent experiments. Experiments are applications composed of code fragments (called snippets) that can be expressed either in general-purpose scripting programming languages (Bash, Ruby, Perl etc.) or domain-specific languages (CxA in MUSCLE, LAMMPS, Matlab, etc). Snippets are evaluated by respective programs called interpreters. GridSpace provides also set of so called Executors that are responsible for snippets execution on various computational resources - servers, clusters, grid via direct SSH on User Interface (UI) machine or interoperability layer such as QCG (see Section 4.3). Each snippet can then be run on different resource.

4.3. Cross-cluster execution with QosCosGrid

Running multiscale application in cross-cluster environment requires addressing the following issues: co-allocation of heterogeneous resources; coordination of spawning application processes at multiple sites; and finally, enabling communication between firewalled and NAT-ed systems.

4.3.1. Co-allocation of heterogenous resources

All modern HPC systems are managed by Local Resource Management Systems (LRMS) [20], often referred to as batch systems. In such environments a user will submit an application for execution (called a

[5] `https://gs2.mapper-project.eu`

job), together with its resource requirements instead of running it directly. At a later time, an LRMS will start the application when the requested resources are available and all local policies are met, thus preventing oversubscription of resources. With cross-cluster multiscale applications, every single model (seen as single job in a given LRMS) must be started at approximately the same time, leading to a problem, as the starting time of jobs are not known prior submission. One possible solution of this problem, known as resource co-allocation, is exploiting the Advance Reservation mechanism. In a nutshell, Advance Reservation (AR) is a reservation created either manually by administrator or automatically by a system (like QCG-Computing BES/AR service[6] [21]) for one or many jobs. An advance reservation is associated with a start and end time, a set of resources, and a list of users that may use this reservation. Once an AR is created, the system guaranties availability of resources for a particular group of users in a given time frame, as long as no system failure occurs.

The QosCosGrid stack uses the AR mechanism available in almost every modern batch system, in order to co-allocate resources belonging to two or more resource providers. The whole process is managed by the QosCosGrid metascheduler: the QCG-Broker service[7] [14]. Users provide an upper limit on application runtime and a time window within which the application should start. QCG-Broker tries to find the earliest time when the requested amount of resources can be booked, and creates an AR for it through the QCG-Computing service. Sites that do not have the QCG-Computing service installed can accept an AR created manually; QCG-Broker then tries to create a schedule based on the manual reservation and availability of other resources as depicted in Figure 5. This process has some similarities with the Two Phase Commit Protocol [22] known from transactions systems, i.e., when advance reservations were created successfully at all sites, the job is submitted (COMMIT); otherwise, all reservations are cancelled (ROLLBACK).

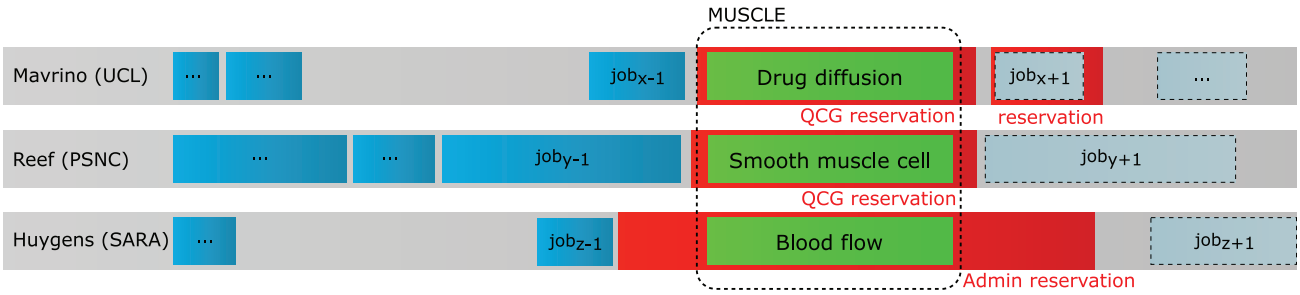


Figure 5: An example of resources co-allocation at three sites

4.3.2. Coordination of application spawning

In most parallel toolkits used within single clusters there is a master process that spawns worker processes either using SSH or LRMS native interfaces. This make the task of exchanging contact information (e.g. listening host and port) between master and workers relatively easy as the master is always initialized before the workers. With a co-allocated distributed application the master and workers are started independently, and exchanging information is less trivial. In the QosCosGrid stack, the QCG-Coordinator service accepts contact information from the master, and provides it to any requesting workers. This relaxes the requirement that the kernels must be started in some particular order.

4.3.3. Cross-cluster communication

Majority of clusters use private IP addresses for their worker nodes, thus accessing any process running within a job is not possible without additional effort. In addition, some of sites impose restrictions on outgoing traffic. In order to distribute multiscale applications that among many clusters, MUSCLE had to be adapted for firewalled and NAT-ed environments. Firstly, a solution based on the port-range technique [23] was implemented, a mechanism which limits the ports numbers MUSCLE uses to some predefined range.

[6] http://www.qoscosgrid.org/trac/qcg-computing
[7] http://www.qoscosgrid.org/trac/qcg-broker

Secondly, communication between worker nodes of two clusters located in different administrative domains had to be enabled. This was solved by implementing a user-space daemon: MUSCLE Transport Overlay (MTO). This daemon is deployed at an interactive node, or any other node that is accessible from both external hosts and worker nodes, of all clusters involved in a multiscale simulation. Every MTO listens on a separate address for external and internal requests. The external port must be either accessible from all the other interactive nodes or the MTO must be able to connect to the external ports of all the others MTO (i.e. uni-directional connection is needed between every pair of MTOs).

Another issue was that private IP addresses used for worker nodes are not globally unique. Consequently, MUSCLE port ranges are enforced to be disjoint among all sites. Under this assumption the tuple $< IP; port >$ is globally unique.

5. Results

In Section 3 an MML description of ISR3D was created. To create a software application it needs to be integrated with a software framework that can run a tightly coupled multiscale model. To this end, each of the submodels and mappers of ISR3D were given a MUSCLE wrapper. In particular, BF acted as a MUSCLE controller that executed a Palabos simulation with MPI.

The information on ISR3D that is presented in Section 3, such as its scale separation map and MML, could be entered in straightforward manner in MaMe and MAD. First, the individual submodels and mappers are entered in MaMe, including information on scales, submodel ports and datatypes. In MaMe, it is also possible to enter preliminary or default parameter settings. Once this is done, the gMML of ISR3D was constructed using MAD, by connecting the respective ports of submodels and mappers of ISR3D together, and exported to a MUSCLE configuration file in GridSpace EW. After this step, the MML description has served its purpose of precisely describing the model computational requirements and is no longer used. This configuration file contained all parameters set in MaMe and all couplings defined in the MAD. In the GridSpace EW, the machines that different submodels should be scheduled on can be specified. Then, it was straightforward to run the simulation by simply pressing start.

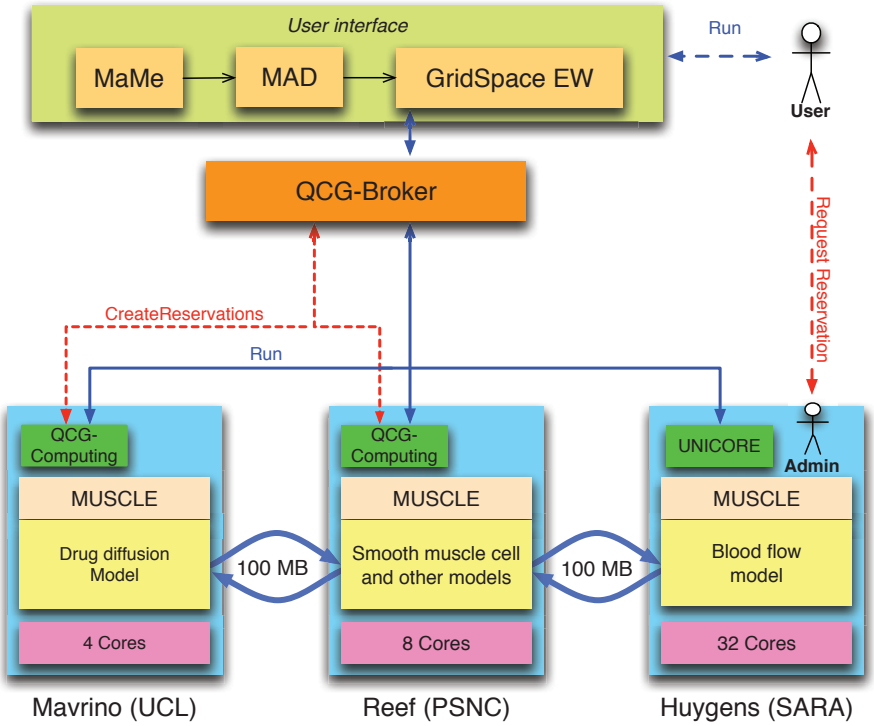


Figure 6: Overall architecture of deployment used during MAPPER demonstration

An example scenario of a tightly coupled model, ISR3D, running on distributed resources, will be described below and is depicted in Figure 6. One of the unique features of this scenario is the integration of resources provided by EGI, PRACE and local infrastructures. As can be seen from the figure, both drug diffusion and blood flow were computed on different hosts from the other submodels. For BF there is a very good reason, it is the only submodel of ISR3D that is extremely well parallelized, and which can make use of a many-core machine. On the other hand, DD could also have been computed on the same host as SMC, however, this scenario is also to show the viability of the approach sketched.

With MUSCLE handling the communication between submodels, problem described in Section 4.3 presented itself, where Huygens did not allow MUSCLE to open ports of worker nodes to the outside. This was fixed by using the MUSCLE Transport Overlay to relay communications to the Reef machine, which had a more liberal security policy.

Meanwhile, QCG-Broker made the reservations to the machines that were scheduled in GridSpace EW, aiming to create a co-allocation. It started by using the manual reservation on Huygens and then proceeded to make an advance reservation on Mavrino and Reef. The LRMS then started the submodels on the different machines, relying on MUSCLE to handle communication between the submodels. Once the model was finished, QCG-Broker collected the data that was generated and returned it to a QCG host, where it can be collected by the user.

During the time that the model was running, three hosts were reserved, however, the resources were not used efficiently in this process; future work should focus on this aspect. Notably, the Huygens machine sat idle when BF was not computing, for instance when SMC was computing. During the run, only few iterations of ISR3D were performed, for scientific results more iterations will be run. On a run-time of more than a week, initialization times by Gridspace EW and QCG-Broker are negligible, being in the order of minutes. Communication overheads between submodels are also relatively small in this case, in the order of seconds [13] compared to a communication frequency of several minutes (in Table 1).

6. Conclusions and discussion

In this contribution we believe to have shown the first generalizable distributed multiscale execution of a tightly coupled multiscale model, in this case, ISR3D. This was achieved by using recent foundations by way of the Multiscale Modeling Language, and tools based on that language: MAPPER Memory and the Multiscale Application designer. Since these tools were integrated with the application manager GridSpace Experiment Workbench, that in turn supported MUSCLE and QCG-Broker as execution tools, ISR3D was executed on heterogeneous infrastructure.

By executing this scenario, ISR3D has the possibility to generate many more results. Other tightly coupled multiscale models in the MAPPER project are anticipated to follow the same approach, further steadying and substantiating it. With this approach gaining more users, also its performance will have to be measured and compared with others.

The inconvenience of using manual reservations on the Huygens machine is a political one, and one of the aims of MAPPER is also to improve support for advance reservation on e-Infrastructure.

One aspect in particular, resource usage and scheduling, should be explored further. In the scenario that was sketched here, the Huygens machine was partially idle while the blood flow submodel was not active. By using more advanced load balancing mechanisms, such as running multiple applications simultaneously to keep all resources active, this may be circumvented. Alternatively, by supporting the task graph for MML [5], and dividing an execution of a multiscale model, submodels could be dynamically scheduled to resources, creating no unnecessary idle reservations.

Acknowledgements

The authors of this contribution would like to thank Hannan Tahir at the University of Amsterdam for sharing his experience with ISR2D; Jules Wolfrat and Axel Berg at SARA, Amsterdam for providing access to the Huygens machine; Steve Brewer and his colleagues at EGI; and Stephen Zasada at University College London for providing access to the Mavrino machine.

This research presented in this contribution is partially supported by the MAPPER project, which receives funding from the EC's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° RI-261507.

References

[1] P. M. A. Sloot, A. G. Hoekstra, Multi-scale modelling in computational biomedicine, Briefings in bioinformatics 11 (1) (2010) 142–152. doi:10.1093/bib/bbp038.

[2] J. Dada, P. Mendes, Multi-scale modelling and simulation in systems biology, Integrative Biology (3) (2011) 86–96. doi:10.1039/c0ib00075b.

[3] G. D. Ingram, I. T. Cameron, K. M. Hangos, Classification and analysis of integrating frameworks in multiscale modelling, Chemical engineering science 59 (2004) 2171–2187. doi:10.1016/j.ces.2004.02.010.

[4] W. E, B. Engquist, X. Li, W. Ren, E. Vanden-Eijnden, The heterogeneous multiscale method: A review, Communications in Computational Physics 2 (3) (2007) 367–450.

[5] J. Borgdorff, J.-L. Falcone, E. Lorenz, C. Bona-Casas, B. Chopard, A. G. Hoekstra, Foundations of Distributed Multiscale Computing: Formalization, Specification, Analysis and Execution, Journal of Parallel and Distributed Computing, submitted (2011) 1–31.

[6] A. G. Hoekstra, E. Lorenz, J.-L. Falcone, B. Chopard, Toward a Complex Automata Formalism for MultiScale Modeling, International Journal for Multiscale Computational Engineering 5 (6) (2007) 491–502. doi:10.1615/IntJMultCompEng.v5.i6.60.

[7] A. Caiazzo, D. J. Evans, J.-L. Falcone, J. Hegewald, E. Lorenz, B. Stahl, D. Wang, J. Bernsdorf, B. Chopard, J. Gunn, R. Hose, M. Krafczyk, P. Lawford, R. Smallwood, D. Walker, A. G. Hoekstra, A Complex Automata approach for In-stent Restenosis: two-dimensional multiscale modeling and simulations, Journal of Computational Science 2 (1) (2011) 9–17. doi:10.1016/j.jocs.2010.09.002.

[8] H. Tahir, A. G. Hoekstra, E. Lorenz, P. V. Lawford, D. R. Hose, J. Gunn, D. J. Evans, Multiscale simulations of the dynamics of in-stent restenosis: impact of stent deployment and design, Interface Focus 1 (3) (2011) 365–373. doi:10.1098/rsfs.2010.0024.

[9] J.-L. Falcone, B. Chopard, A. G. Hoekstra, MML: towards a Multiscale Modeling Language, Procedia Computer Science 1 (1) (2010) 819–826. doi:10.1016/j.procs.2010.04.089.

[10] J. Borgdorff, J.-L. Falcone, E. Lorenz, B. Chopard, A. G. Hoekstra, A principled approach to distributed multiscale computing, from formalization to execution, in: Proceedings of the 7th IEEE International Conference on e-Science, IEEE Computer Society Press, Stockholm, Sweden, 2011.

[11] E. Ciepiela, D. Harezlak, J. Kocot, et al., Exploratory programming in the virtual laboratory, in: Proceedings of the International Multiconference on Computer Science and Information Technology, Wisla, Poland, 2010, pp. 621–628.

[12] P. Nowakowski, E. Ciepiela, D. Harezlak, J. Kocot, M. Kasztelnik, T. Bartynski, J. Meizner, G. Dyk, M. Malawski, The collage authoring environment, Procedia CS 4 (2011) 608–617.

[13] J. Hegewald, M. Krafczyk, J. Tölke, A. G. Hoekstra, An agent-based coupling platform for complex automata, in: Computational Science–ICCS 2008, 2008, pp. 227–233. doi:10.1007/978-3-540-69387-1_25.

[14] K. Kurowski, W. Back, W. Dubitzky, L. Gulyás, G. Kampis, M. Mamonski, G. Szemes, M. Swain, Complex System Simulations with QosCosGrid, in: Proceedings of the 9th International Conference on Computational Science: Part I, ICCS '09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 387–396. doi:http://dx.doi.org/10.1007/978-3-642-01970-8_38.

[15] S. Allender, V. Peto, P. Scarborough, A. Kaur, M. Rayner, Coronary heart disease statistics., Tech. rep. (2008).

[16] A. Moustapha, A. R. Assali, S. Sdringola, W. K. Vaughn, R. D. Fish, O. Rosales, G. Schroth, Z. Krajcer, R. W. Smalling, H. V. Anderson, Percutaneous and surgical interventions for in-stent restenosis: long-term outcomes and effect of diabetes mellitus, Journal of the American College of Cardiology 37 (7) (2001) 1877–1882. doi:10.1016/S0735-1097(01)01231-1.

[17] A. Kastrati, D. Hall, A. Schömig, Long-term outcome after coronary stenting, Current Controlled Trials in Cardiovascular Medicine 1 (1) (2000) 48–54.

[18] D. J. W. Evans, P. V. Lawford, J. Gunn, D. Walker, D. R. Hose, R. H. Smallwood, B. Chopard, M. Krafczyk, J. Bernsdorf, A. G. Hoekstra, The application of multiscale modelling to the process of development and prevention of stenosis in a stented coronary artery, Philosophical Transactions of the Royal Society A 366 (2008) 3343–3360. doi:10.1098/rsta.2008.0081.

[19] T. Gubala, M. Bubak, P. M. Sloot, Semantic Integration of Collaborative Research Environments, Information Science Reference IGI Global, 2009, Ch. XXVI, pp. 514–530.

[20] M. Bozzo-Rey, M. Jeanson, M. Nguyen, C. Gauthier, M. Barrette, P. Vachon, K. Gaven-Venet, H. Lu, S. Allen, A. Veilleux, Design, deployment and bench of a large infiniband hpc cluster, in: High-Performance Computing in an Advanced Collaborative Environment, 2006. HPCS 2006. 20th International Symposium on, IEEE, 2006, pp. 8–8.

[21] M. Mamoński, GFD.179 – Smoa Computing HPC Basic Profile Adoption – Experience Report, Tech. rep., Open Grid Forum (2011).

[22] G. Weikum, G. Vossen, Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery, Morgan Kaufmann Pub, 2002.

[23] J. Maassen, H. Bal, Smartsockets: solving the connectivity problems in grid computing, in: Proceedings of the 16th international symposium on High performance distributed computing, ACM, 2007, pp. 1–10.

Publication [P2]

Borgdorff, J., Mamonski, M., **Bosak, B.**, Groen, D., Belgacem, M. B., Kurowski, K., and Hoekstra, A. G.: *Multiscale computing with the multiscale modeling library and runtime environment.* Procedia Computer Science, 18:1097–1105. 2013 International Conference on Computational Science (2013). https://doi.org/10.1016/j.procs.2013.05.275

Ministry points / conference: 140

Contribution of authors:

- Joris Borgdorff
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, Coupling library, Runtime environment, Use cases, Conclusions and Future Work)
- Alfons G. Hoekstra
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, Conclusions and Future Work)
  - Review and editing of the paper
- **Bartosz Bosak**, Mariusz Mamonski
  - Coauthorship of the text of publication (Section: Runtime environment)
- Krzysztof Kurowski
  - Coauthorship of the text of publication (Section: Runtime environment)
  - Review and editing of the paper
- Derek Groen, Mohamed Ben Belgacem
  - Coauthorship of the text of publication (Section: Use cases)
  - Review and editing of the paper

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 1097 – 1105

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

Multiscale computing with the multiscale modeling library and runtime environment

Joris Borgdorff[a,*], Mariusz Mamonski[b], Bartosz Bosak[b], Derek Groen[c], Mohamed Ben Belgacem[d], Krzysztof Kurowski[b], Alfons G. Hoekstra[a]

[a] *Computational Science, Faculty of Science, University of Amsterdam, Amsterdam, the Netherlands*
[b] *Poznań Supercomputing and Networking Center, Poznań, Poland*
[c] *Centre for Computational Science, University College London, London, United Kingdom*
[d] *CUI, University of Geneva, Carouge, Switzerland*

Abstract

We introduce a software tool to simulate multiscale models: the Multiscale Coupling Library and Environment 2 (MUSCLE 2). MUSCLE 2 is a component-based modeling tool inspired by the multiscale modeling and simulation framework, with an easy-to-use API which supports Java, C++, C, and Fortran. We present MUSCLE 2's runtime features, such as its distributed computing capabilities, and its benefits to multiscale modelers. We also describe two multiscale models that use MUSCLE 2 to do distributed multiscale computing: an in-stent restenosis and a canal system model. We conclude that MUSCLE 2 is a notable improvement over the previous version of MUSCLE, and that it allows users to more flexibly deploy simulations of multiscale models, while improving their performance.

Keywords: multiscale modeling, distributed multiscale computing, MUSCLE

1. Introduction

Multiscale modeling is a way to gain knowledge about complex systems by explicitly modeling the interaction between phenomena on different scales. It has had attention of diverse research fields [1], amongst others biomedicine [2], biology [3], physics [4], chemistry [5] and earth sciences [6]. The need for a general computational framework that is able to run these types of simulations is expressed by several authors [7–9].

The aspiration for distributed multiscale computing has given impulse to the Multiscale Coupling Library and Environment 2 (MUSCLE 2), which builds upon an earlier environment built by Hegewald *et al.* [10]. It is a portable and lightweight framework to implement and execute multiscale models, and if needed to run them on distributed resources. An overview of MUSCLE is shown in Figure 1.

*Corresponding author
Email addresses: J.Borgdorff@uva.nl (Joris Borgdorff), mamonski@man.poznan.pl (Mariusz Mamonski), bbosak@man.poznan.pl (Bartosz Bosak), d.groen@ucl.ac.uk (Derek Groen), mohamed.benbelgacem@unige.ch (Mohamed Ben Belgacem), krzysztof.kurowski@man.poznan.pl (Krzysztof Kurowski), A.G.Hoekstra@uva.nl (Alfons G. Hoekstra)

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.275

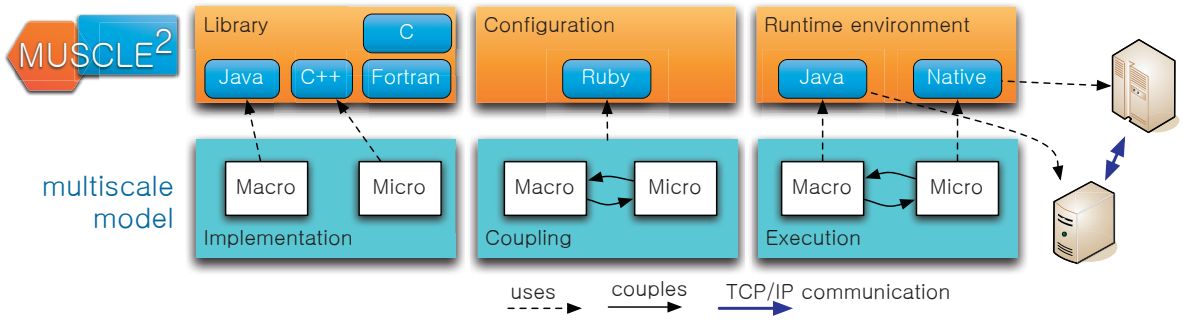


Figure 1: Overview of MUSCLE 2: with the *library*, submodels are implemented with any of the APIs, with the *configuration* those submodels are instantiated and coupled, and in the *runtime environment* they can be executed on a variety of machines.

It assumes that a multiscale model is split into multiple coupled single scale submodels as proposed by the Multiscale Modeling and Simulation Framework [11–13], preceded by the Complex Automata theory [7, 8]. As a result, MUSCLE is a component-based framework, where each submodel has inputs and outputs that can be coupled in a general way. It recognizes that submodels execute on different temporal scales and keeps simulation time between submodels in sync when they communicate. So-called mappers apply scale bridging to in-transit data so that the submodels themselves really have a single scale without knowledge of other single scale models that may be part of the overall multiscale model.

From a runtime perspective on MUSCLE 2, the submodels may be written in Java, C, C++, or Fortran. Submodels may be implemented with OpenMP or MPI and only need MUSCLE (and consequently Java and Ruby) to be run on a node reachable with TCP/IP. Submodels themselves can be as computationally costly as needed. Within one simulation, one submodel could for instance use hundreds of cores on a supercomputer, whereas another may have to make use of GPU-computing, and yet another needs high I/O performance. Execution between distributed resources and bypassing of firewalls is made possible by the message forwarder MUSCLE Transport Overlay, which is developed and packaged with MUSCLE. This is a change from traditional model coupling toolkits such as CCaffeine [14, 15] or the Model Coupling Toolkit [16, 17], which are not focussed on multiscale modeling and do not tend to run in a distributed fashion.

In this contribution, we will describe the way in which a multiscale model can be implemented with MUSCLE 2 in Section 2, and what runtime options it possesses in Section 3, also in comparison with the original MUSCLE. Finally, Section 4 shows how it has been applied to a multiscale model of in-stent restenosis. MUSCLE 2 is publicly available at `http://apps.man.poznan.pl/trac/muscle`.

2. Coupling library

The foundations that MUSCLE is based on are described by the Multiscale Modeling and Simulation Framework (MMSF) [11–13] and their computational descriptions in the multiscale modeling language (MML) [13, 18]. To better understand further sections, we will shortly repeat the contents of the framework before we introduce the features of MUSCLE. The API of MUSCLE 2 as well as part of its implementation is derived from MUSCLE 1 [10], which was based on the Complex Automata theory [7, 8].

2.1. Theoretical background

At the end of the modeling stage, MMSF suggests that a multiscale model should be built up out of coupled single scale models, or submodels. Moreover, a multiscale model should have a notion of time, and the time of different submodels has to remain consistent. These single scale models should be dependent only on their input and output, and not of other single scale models. As a consequence, scale bridging has to be done between single scale models, to make sure that they receive data that are relevant on their scale. For example, a micro-submodel will usually not need to process data about the

entire domain, but only needs to receive a small portion of the domain to do its computations. The macro-submodel on the other hand should not concern itself with splitting up the data so that it exactly matches the domain of the micro-submodels. The inputs and outputs of submodels are coupled with so-called conduits.

Scale bridging methods can be applied to data that is being sent over the conduits, either with conduit filters for data transformation or time averaging or interpolation, or by mappers, which may combine the data of multiple conduits, and send multiple outputs. To make a model self-contained, sources and sinks may also be attached to conduits so that submodels can be tested in isolation or external data sources can be consulted. In the example of the micro-submodels and a macro-submodel, the macro-submodel would send its domain over a conduit. The domain would then be split into multiple subdomains in a model-specific way, and divided amongst the micro-submodels. These micro-submodels would send their data to a mapper again, which would combine their data into one dataset and send that as a single input to the macro-submodel.

2.2. Library

Developers can implement submodels of a multiscale model with the Java, C, C++, or Fortran API, or a combination thereof. Among them, the Java API is closest to the MUSCLE core and as such is the most expressive. It offers an API for sending and receiving any type of objects, advanced logging, output redirection, time manipulation, and formal MMSF constructs such as formal submodels, mappers, conduit filters, sources, and sinks. In complex topologies, simulation time can be manipulated to ensure that all submodels are processing at the correct simulation time. In C++, free-form submodels and mappers can be written, which can send and receive strings, arrays, matrices, maps and lists, and do logging and output redirection. For C and Fortran the functionality is similar but only arrays and strings can be sent and received. For C++, C, or Fortran an accompanying Java class can be written that handles the more advanced capabilities of MUSCLE, such as advanced data structures or time manipulation. For each of the functionalities, example code is provided with MUSCLE and documentation is provided on the public MUSCLE webpage[1].

Only little code is needed to write a submodel or filter in MUSCLE, and it is straightforward to integrate in existing code, since only the send and receive statements are necessary to add. In Java, the statements to receive data from an input port `dataInput` and send it to a different port `dataOuput` would look like:

```
double[] data = (double[])in("dataInput").receive();
out("dataOutput").send(data);
```

The names of the ports are arbitrary, but they will be used for the coupling in the next section. For more advanced type-checking, each port can be initialized beforehand with the respective Java classes they should receive.

Programming languages that do not have a MUSCLE 2 API, such as Ruby, Python or Scala, may make use of MUSCLE by writing an intermediate interface, at the sacrifice of slightly reducing portability and introducing dependencies to other software.

Compared to MUSCLE 1, the submodel implementation may be much more succinct, by handling certain default initialization methods in the runtime system. Especially the C++ code is much clearer, since it no longer requires use of the Java Native Interface. The C and Fortran routines are newly added. By separating the implementation and library part of MUSCLE 1, MUSCLE 2 now also does not confront the user with MUSCLE implementation details, such as the interface with the Java runtime environment in C or C++, or the Java Agent Development Environment (JADE) library in the Java code. A beneficial side-effect is that code written by users is less susceptible to breaking due to changes in the runtime environment. Additionally, integrating MUSCLE into existing code has thus been simplified.

[1] MUSCLE 2 documentation: `http://apps.man.poznan.pl/trac/muscle/wiki/Documentation`

2.3. Configuration

The parameterization and setting up the coupling topology of the multiscale model is done in a Ruby file with MUSCLE helper classes. From the legacy of MUSCLE 1, this is called the CxA (Complex Automata) file. Any of the Ruby language features may be used to make the final coupling topology, for instance precomputing a mesh-like network, reading environment variables, or reading the topology from a separate file. Conduit filters are applied by adding them as an array to any conduit.

Parameters may be given as submodel local variables or as global variables, to make sure that all submodels use the same domain definitions for instance. Again, the file and thus the parameters may be scripted or precomputed. Time and space scales can be written in human readable terms (e.g., `'2 hr'`, `'1 meter'`) or in SI notation (e.g., `'1 ps'`, `'4.8E-3 m'`), without loss of precision.

If convenient, the Java classpath, library path, or the MPI executable can be specified in the configuration file. This will, however, make the configuration file partially dependent on the runtime environment. To make it more independent from the runtime environment, environment variables can be substituted in these paths.

An example of a configuration file is given below, with instances Macro and Micro coupled only from Macro to Micro.

```
# instance name and class name
cxa.add_instance('Macro', 'nl.uva.cs.Macro')

# C++ submodels can use the MUSCLE NativeKernel that will manage the executable
cxa.add_instance('Micro', 'muscle.core.standalone.NativeKernel')
# Let the NativeKernel know where the executable is
cxa.env['Micro:command'] = ENV['MODEL_HOME'] + '/bin/micro'

# Set the time scales
cxa.env['max_timesteps'] = '2 hr'
cxa.env['Macro:dt'] = '4 min'
cxa.env['Micro:dt'] = '1 ms'
cxa.env['Micro:T'] = '1 s'

# Attach Macro.data_out to Micro.data_in
cxa.cs.attach('Macro' => 'Micro') {
  tie('dataOutput', 'dataInput')
}
```

3. Runtime environment

The runtime environment of MUSCLE 2 is portable and designed for distributed computing. Compared to the release of MUSCLE 1, MUSCLE 2 excels in portability, distributed computing, performance, and usability. It can be executed on a local machine or on supercomputers, through the command-line interface or with queueing systems [19]. Data transmission is decentralized so communication performance can be optimized by the user by running closely tied submodels on machines that are physically close. To launch a submodel or a set of submodels, the user runs the `muscle2` command. When this command is invoked, as shown in Figure 2, a Ruby script first parses the command-line arguments and evaluates the configuration file, passing a fixed coupling description to the MUSCLE core. One or more submodels are then started by a so-called Local Manager, which runs in the Java Virtual Machine. The tasks of the Local Manager are to check for error conditions within a single Java Virtual Machine and to listen for data connections. One so-called Simulation Manager per multiscale simulation acts as a white pages service and it is the only centralized component of MUSCLE. To limit its runtime overhead, results of queries to the Simulation Manager are cached by the Local Managers.

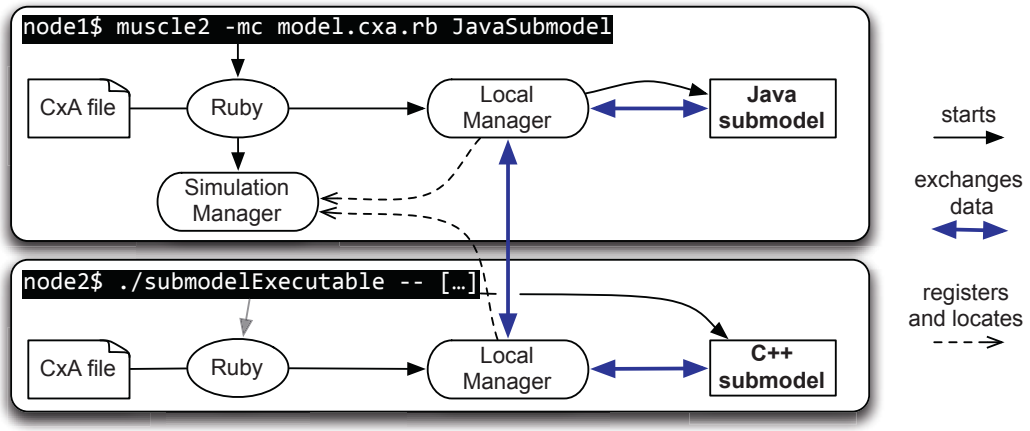


Figure 2: Example of the execution of a multiscale model containing a Java submodel and a C++ submodel. In the top left corner of each rounded rectangle is the command that is executed on different nodes. In the top rectangle, a Java submodel and the Simulation Manager are started in a single command. In the bottom rectangle, an executable is directly started, and Ruby (and thus MUSCLE) is started indirectly through the initialization method of the MUSCLE 2 API.

To keep submodels and their runtime behavior separated, each submodel is managed by its own instance controller. The instance controller registers itself with the Simulation Manager and handles any errors that a submodel might cause. The controller is also an intermediary for any messages that a submodel sends or receives. For C++, C, and Fortran submodels, the controller first converts data to Java objects and then sends it to other submodels. If necessary, it also starts the executable with the submodel, and checks if it keeps running. In the current implementation, each controller runs in its own thread, so that each submodel does its computations independently. Consequently, a limit to the number of threads in many operating systems (for example, a limit of about 2000 threads per process on OS X) causes the number of submodels per Local Manager to be restricted. For most purposes, where submodels do heavy calculations, this is not a problem, since the CPU-cores will be overtaxed far below this limit. However, for instance a network simulation of a multitude of very simple nodes might find it necessary to group several nodes into a single submodel to limit the number of created threads.

Exceptions in most parts of the system work fail-fast, so that if a submodel fails with an exception, all other submodels are also halted. The instance controller does this by notifying the Local Manager, all attached conduits, and the Simulation Manager, so that all submodels in the current Java Virtual Machine are halted, as well as attached submodels in other Java Virtual Machines. Submodels that are started after the exception occurred are notified by the Simulation Manager. The reason for this behavior is to limit the amount of runtime resources consumed after a part of the simulation has become invalid.

A few different paths of communication exist within MUSCLE. First of all, any messages within the same Local Manager are passed through shared memory, with a copy operation if needed to guarantee isolation of data of different submodels. Messages between one Local Manager and another use the serialization library MessagePack [20] over TCP/IP, because of its small encoding length and high speed. Between instance controllers and C/C++/Fortran submodels the XDR [21] serialization library is used, which has a higher computational overhead but is widely available, since it is installed with most *nix/BSD operating systems.

The dependencies for the runtime environment are Ruby and Java, to be installed on a node with a direct TCP/IP connection to the node that should be executed on. Installation is done with a single command, with only the CMake build tool and basic build environment required.

The command-line interface for MUSCLE is clean and simple. For instance, the command

```
muscle2 --main --allinstances --cxa model.cxa.rb
```

runs the Simulation Manager (`--main`) and a Local Manager with all instances (`--allinstances`) using the configuration file `model.cxa.rb` (`--cxa`).

3.1. Distributed computing

When heterogeneous or large scale computing is considered, distributed computing is an option. However, many HPC systems use a private IPv4 addressing scheme which is incompatible with direct communications between systems. Therefore, the MUSCLE Transport Overlay (MTO) is packaged with MUSCLE 2. The MTO is started on all systems that must participate in a simulation. Using a port-mapping technique, data transfers that are intended for different clusters are then forwarded by the MTO.

MTO relies on the ASIO (ASynchronous Input Output) subsystem of the Boost library. There is an ongoing effort to integrate MPWide [22] into the MTO to increase messaging speed between different clusters and to remove the dependency on Boost.

To facilitate cross-cluster simulations, MUSCLE is integrated with the QosCosGrid middleware stack [23]. This stack provides automation of the process of submitting cross-cluster simulations by: co-allocating resources on multiple sites (using the Advance Reservation mechanism); staging input/output files to/from every system involved in the simulation; managing the submission of sub-jobs; providing a locator service; and finally, allowing to have a peek at the current output of every submodel from a single location.

To run submodel Macro on node1 and Micro on node2 with the MTO, the following two commands are sufficient, given that the MTO has been set up:

```
node1$ muscle2 -mc model.cxa.rb --intercluster Macro
node2$ muscle2 --manager node1:9000 -c model.cxa.rb --intercluster Micro
```

Here, the Simulation Manager is started only once, and other MUSCLE execution just specifies the managers location and port. If there is a direct TCP/IP connection between node1 and node2, the MTO is not needed, nor is the `--intercluster` argument.

3.2. Comparison with the previous version of MUSCLE

In contrast with the previous MUSCLE implementation, MUSCLE 2 is compatible with Ruby versions 1.8.7 and 1.9.x, and Java 6 and 7. The number of Java library dependencies has been drastically decreased. For instance, it does not rely on the Java Agent Development Environment (JADE) for its communication, which makes it possible to optimize communication protocols and serialization algorithms. In addition, without JADE, MUSCLE 2 has more control over the initialization and finalization of a simulation and does so with less overhead.

Unlike its predecessor, it does not use the Java Native Interface, since this proved much less portable and user-friendly than providing a separate C++ library. Moreover, this C++ library is compatible with MPI and OpenMP, which the JNI interface was not.

The build system has been generalized using CMake as an engine, and it is now customizable per site by simply setting environment variables. Likewise, runtime settings such as library paths are now recognized as environment variables by MUSCLE 2, to make it more customizable for middleware.

Quantitatively, the message latency of MUSCLE 2 is 45 times lower, at 15 μs, the message throughput is five times higher at 1.9 GiB/s, and the initialization and finalization overhead 6.6 times lower at 0.68 seconds. The experiments for these results were conducted with a basic back-and-forth messaging Java code, sending messages with sizes from 1 kB to 8 MB, taking the minimum communication time as a latency and using a linear regression to get the throughput. Startup times are calculated by starting these submodels, let them send one empty message and quit. This was measured on an Apple iMac with a dual-core Intel i3 3.2 GHz processor.

MUSCLE 2 is also able to handle larger messages, up to a gigabyte, while MUSCLE 1 is not made to handle messages larger than 10 MB.

4. Use cases

Within the MAPPER project [24], at least four applications are successfully using MUSCLE 2: a cardiovascular biomedical model of in-stent restenosis (ISR3D) [25], a hydrological canal system model [26], a systems biology optimizer of a gene regulatory network, and a high-energy physics plasma simulator.

The model of in-stent restenosis couples, amongst others, smooth muscle cell proliferation in the coronary artery wall to the blood flow through the vessel [25]. This interaction is hypothesized to be associated to a restenosis after stenting [27]. The blood flow submodel code is highly parallelized using MPI, while the smooth muscle cell proliferation code is less scalable and uses OpenMP. It is successfully being run on distributed computing resources with MUSCLE 2 [28, 29]. Before MUSCLE 2, the model did run in specifically set up environments but not on e-Infrastructure in general.

Water management is a main concern in our modern society, in particular for water supply, electricity production and transport. The canal application simulates canal systems like irrigation canals or rivers by coupling canal segments together through water junctions. Due to the size of the canal system and the large variation of the flow, a fully resolved 3D free surface flow computation is not feasible and, thus, a multiscale computational approach is needed [26]. For example, some canal sections where the flow is stable can simply be modeled by a 1D shallow model, whereas other sections need a 3D free surface model to precisely capture the flow properties, such as studying the water behavior around a gate. With tools developed in the MAPPER project, notably MUSCLE 2, those different sections can be connected and easily executed on distributed grid infrastructure.

Each of these models have submodels with high computational demands, ranging from tens up to thousands of CPU cores. Moreover, they transfer messages with sizes from less than 1 MB up to 100 MB. In the case of ISR3D, the overhead of using MUSCLE 2 has been shown to be insignificant compared to the computational cost of the submodels [29].

5. Conclusions and Future Work

We have presented the design and features of the MUSCLE 2 coupling environment and reviewed its application in the multiscale modeling of in-stent restenosis and a canal system. MUSCLE 2 is based on the strong foundations of the multiscale modeling and simulation framework, as well as the multiscale modeling language. Because of MUSCLE 2's modular setup, which clearly separates the API, the coupling, and the runtime environment, users can easily modify parts of a multiscale model in a plug and play fashion. A multiscale model, consisting of two or more coupled single scale models, can be conveniently deployed and executed on a set of distributed computing resources. We conclude that MUSCLE 2 is a valuable addition for multiscale modeling and simulation in cases where flexibility and performance are important, and especially when the multiscale model relies on two or more different simulation codes.

Acknowledgements

This research presented in this contribution is partially supported by the MAPPER project, which receives funding from the EC's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° RI-261507.

References

[1] D. Groen, S. J. Zasada, P. V. Coveney, Taxonomy of Multiscale Computing Communities, in: e-Science Workshops (eScienceW), 2011 IEEE Seventh International Conference on, 5-8 Dec. 2011, Stockholm, Sweden, 2011, pp. 120–127. doi:10.1109/eScienceW.2011.11.

[2] P. M. A. Sloot, A. G. Hoekstra, Multi-scale modelling in computational biomedicine, Briefings in bioinformatics 11 (1) (2010) 142–152. doi:10.1093/bib/bbp038.

[3] J. Southern, J. Pitt-Francis, J. Whiteley, D. Stokeley, H. Kobashi, R. Nobes, K. Yoshimasa, D. Gavaghan, Multi-scale computational modelling in biology and physiology, Progress in Biophysics and Molecular Biology (96) (2008) 60–89. doi:10.1016/j.pbiomolbio.2007.07.019.

[4] W. E, B. Engquist, X. Li, W. Ren, E. Vanden-Eijnden, The heterogeneous multiscale method: A review, Communications in Computational Physics 2 (3) (2007) 367–450.

[5] G. D. Ingram, I. T. Cameron, K. M. Hangos, Classification and analysis of integrating frameworks in multiscale modelling, Chemical engineering science 59 (2004) 2171–2187. doi:10.1016/j.ces.2004.02.010.

[6] C. W. Armstrong, R. W. Ford, G. D. Riley, Coupling integrated Earth System Model components with BFG2, Concurrency and Computation: Practice and Experience 21 (6) (2009) 767–791. doi:10.1002/cpe.1348.

[7] A. G. Hoekstra, E. Lorenz, J.-L. Falcone, B. Chopard, Toward a Complex Automata Formalism for MultiScale Modeling, International Journal for Multiscale Computational Engineering 5 (6) (2007) 491–502. doi:10.1615/IntJMultCompEng.v5.i6.60.

[8] A. G. Hoekstra, A. Caiazzo, E. Lorenz, J.-L. Falcone, Complex automata: multi-scale modeling with coupled cellular automata, in: A. G. Hoekstra, J. Kroc, P. M. A. Sloot (Eds.), Simulating Complex Systems by Cellular Automata, Springer-Verlag Berlin, Heidelberg, 2010, pp. 29–57. doi:10.1007/978-3-642-12203-3_3.

[9] J. Dada, P. Mendes, Multi-scale modelling and simulation in systems biology, Integrative Biology (3) (2011) 86–96. doi:10.1039/c0ib00075b.

[10] J. Hegewald, M. Krafczyk, J. Tölke, A. G. Hoekstra, An agent-based coupling platform for complex automata, in: ICCS 2008, LNCS 5102, Springer-Verlag Berlin Heidelberg, 2008, pp. 227–233. doi:10.1007/978-3-540-69387-1_25.

[11] B. Chopard, J.-L. Falcone, A. G. Hoekstra, J. Borgdorff, A Framework for Multiscale and Multiscience Modeling and Numerical Simulations, in: C. Calude, J. Kari, I. Petre, G. Rozenberg (Eds.), LNCS 6714, Springer-Verlag Berlin Heidelberg, 2011, pp. 2–8. doi:10.1007/978-3-642-21341-0_2.

[12] J. Borgdorff, J.-L. Falcone, E. Lorenz, B. Chopard, A. G. Hoekstra, A principled approach to distributed multiscale computing, from formalization to execution, in: Proceedings of the IEEE 7th International Conference on e-Science Workshops, IEEE Computer Society Press, Stockholm, Sweden, 2011, pp. 97–104. doi:10.1109/eScienceW.2011.9.

[13] J. Borgdorff, J.-L. Falcone, E. Lorenz, C. Bona-Casas, B. Chopard, A. G. Hoekstra, Foundations of distributed multiscale computing: Formalization, specification, and analysis, Journal of Parallel and Distributed Computing 73 (2013) 465–483. doi:10.1016/j.jpdc.2012.12.011.

[14] B. A. Allan, R. Armstrong, D. E. Bernholdt, F. Bertrand, K. Chiu, T. L. Dahlgren, K. B. Damevski, W. R. Elwasif, M. Govindaraju, D. S. Katz, J. A. Kohl, M. Krishnan, J. W. Larson, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. Mcinnes, J. Nieplocha, B. Norris, J. Ray, T. L. Windus, S. Zhou, A Component Architecture for High-Performance Scientific Computing, International Journal of High-Performance Computing Applications 20 (2) (2006) 163–202. doi:10.1177/1094342006064488.

[15] B. A. Allan, R. Armstrong, Ccaffeine Framework: Composing and Debugging Applications Iteratively and Running them Statically. , Tech. Rep. SAND2005-1135C, Sandia National Laboratories (2005).

[16] D. Kim, J. W. Larson, K. Chiu, Toward Malleable Model Coupling, Procedia Computer Science 4 (2011) 312–321. doi:10.1016/j.procs.2011.04.033.

[17] J. W. Larson, R. L. Jacob, I. Foster, J. Guo, The model coupling toolkit, in: V. N. Alexandrov, J. J. Dongarra, B. A. Juliano, R. S. Renner, C. J. K. Tan (Eds.), ICCS 2001, LNCS 2073, Springer-Verlag Berlin Heidelberg, 2001, pp. 185–194. doi:10.1007/3-540-45545-0_27.

[18] J.-L. Falcone, B. Chopard, A. G. Hoekstra, MML: towards a Multiscale Modeling Language, Procedia Computer Science 1 (1) (2010) 819–826. doi:10.1016/j.procs.2010.04.089.

[19] S. J. Zasada, M. Mamonski, D. Groen, J. Borgdorff, I. Saverchenko, T. Piontek, K. Kurowski, P. V. Coveney, Distributed Infrastructure for Multiscale Computing, in: 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), IEEE, 2012, pp. 65–74. doi:10.1109/DS-RT.2012.17.

[20] MessagePack library, Messagepack 0.6.6, http://msgpack.org (2012).

[21] Sun Microsystems, XDR: External Data Representation standard, RFC 1014 (Jun. 1987).
URL http://www.ietf.org/rfc/rfc1014.txt

[22] D. Groen, S. Rieder, P. Grosso, C. d. Laat, S. P. Zwart, A lightweight communication library for distributed computing, Computational Science & Discovery 3 (1) (2010) 015002. doi:10.1088/1749-4699/3/1/015002.

[23] B. Bosak, J. Komasa, P. Kopta, K. Kurowski, M. Mamoński, T. Piontek, New capabilities in QosCosGrid middleware for advanced job management, advance reservation and co-allocation of computing resources–quantum chemistry application use case, Building a National Distributed e-Infrastructure–PL-Grid (2012) 40–55.

[24] The MAPPER project, http://www.mapper-project.eu/ (2010).

[25] A. Caiazzo, D. J. W. Evans, J.-L. Falcone, J. Hegewald, E. Lorenz, B. Stahl, D. Wang, J. Bernsdorf, B. Chopard, J. Gunn, D. R. Hose, M. Krafczyk, P. V. Lawford, R. H. Smallwood, D. Walker, A. G. Hoekstra, A Complex Automata approach for In-stent Restenosis: two-dimensional multiscale modeling and simulations, Journal of Computational Science 2 (1) (2011) 9–17. doi:10.1016/j.jocs.2010.09.002.

[26] M. Ben Belgacem, B. Chopard, A. Parmigiani, Coupling Method for Building a Network of Irrigation Canals on a Distributed Computing Environment, in: Cellular Automata, LNCS 7495, Springer, Berlin, Heidelberg, 2012, pp. 309–318. doi:10.1007/978-3-642-33350-7_32.

[27] D. J. W. Evans, P. V. Lawford, J. Gunn, D. Walker, D. R. Hose, R. H. Smallwood, B. Chopard, M. Krafczyk, J. Bernsdorf, A. G. Hoekstra, The application of multiscale modelling to the process of development and prevention of stenosis in a stented coronary artery, Philosophical Transactions of the Royal Society A 366 (2008) 3343–3360. doi:10.1098/rsta.2008.0081.

[28] J. Borgdorff, C. Bona-Casas, M. Mamonski, K. Kurowski, T. Piontek, B. Bosak, K. Rycerz, E. Ciepiela, T. Gubała, D. Harezlak, M. Bubak, E. Lorenz, A. G. Hoekstra, A Distributed Multiscale Computation of a Tightly Coupled Model Using the Multiscale Modeling Language, Procedia Computer Science 9 (2012) 596–605. doi:10.1016/j.procs.2012.04.064.

[29] 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, P. V. Coveney, Flexible composition and execution of high performance, high fidelity multiscale biomedical simulations, Interface Focus, accepted (2013) arXiv:1211.2963.

Publication [P3]

Borgdorff, J., Mamonski, M., **Bosak, B.**, Kurowski, K., Ben Belgacem, M., Chopard, B., Groen, D., Coveney, P., and Hoekstra, A.: *Distributed multiscale computing with MUSCLE 2, the multiscale coupling library and environment.* Journal of Computational Science, 5(5):719–731 (2014). https://doi.org/10.1016/j.jocs.2014.04.004

Ministry points / journal: 100

Contribution of authors:

- Joris Borgdorff
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, Design, Performance, Use cases / ISR3D, Conclusions, Appendix A)
- Alfons G. Hoekstra
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, Design, Use cases / ISR3D, Conclusions)
- **Bartosz Bosak**, Mariusz Mamonski
  - Coauthorship of the text of publication (Sections: Design, Performance, Appendix A)
- Krzysztof Kurowski
  - Coauthorship of the text of publication (Sections: Design, Conclusions)
  - Review and editing of the paper
- Mohamed Ben Belgacem
  - Coauthorship of the text of publication (Section: Use cases / Hydrology application)
- Bastien Chopard
  - Coauthorship of the text of publication (Section: Use cases / Hydrology application)
  - Review and editing of the paper
- Derek Groen
  - Coauthorship of the text of publication (Sections: Design / Performance)
- Peter Coveney
  - Coauthorship of the text of publication (Section: Introduction)
  - Review and editing of the paper

Journal of Computational Science 5 (2014) 719–731

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment

J. Borgdorff[a,*], M. Mamonski[b,1], B. Bosak[b], K. Kurowski[b], M. Ben Belgacem[c], B. Chopard[c], D. Groen[d], P.V. Coveney[d], A.G. Hoekstra[a,e]

[a] Computational Science, Faculty of Science, University of Amsterdam, Amsterdam, The Netherlands
[b] Poznań Supercomputing and Networking Center, Poznań, Poland
[c] Computer Science Centre, University of Geneva, Carouge, Switzerland
[d] Centre for Computational Science, University College London, London, United Kingdom
[e] National Research University ITMO, Saint-Petersburg, Russia

ARTICLE INFO

Article history:
Received 4 November 2013
Received in revised form 24 March 2014
Accepted 8 April 2014
Available online 18 April 2014

Keywords:
Distributed multiscale computing
Multiscale modelling
Model coupling
Execution environment
MUSCLE

ABSTRACT

We present the Multiscale Coupling Library and Environment: MUSCLE 2. This multiscale component-based execution environment has a simple to use Java, C++, C, Python and Fortran API, compatible with MPI, OpenMP and threading codes. We demonstrate its local and distributed computing capabilities and compare its performance to MUSCLE 1, file copy, MPI, MPWide, and GridFTP. The local throughput of MPI is about two times higher, so very tightly coupled code should use MPI as a single submodel of MUSCLE 2; the distributed performance of GridFTP is lower, especially for small messages. We test the performance of a canal system model with MUSCLE 2, where it introduces an overhead as small as 5% compared to MPI.

1. Introduction

Multiscale modelling and simulation is of growing interest [21], with appeal to scientists in many fields such as computational biomedicine [33], biology [34], systems biology [14], physics [15], chemistry [27] and earth sciences [3]. Meanwhile, there are efforts to provide a more general way of describing multiscale models [26,38,7], including our Multiscale Modelling and Simulation Framework [13,23,24,8]. This framework describes the process of constructing a multiscale model by identifying and separating its scales, defining a multiscale model as a set of coupled single scale models. It then provides a computational modelling language and environment to create and deploy such models on a range of computing infrastructures. For an example of biomedical applications in this context, see [19].

In this paper we present a means to implement multiscale models as described in this theoretical framework: The Multiscale Coupling Library and Environment 2 (MUSCLE 2). It takes a component-based approach to multiscale modelling, promoting modularity in its design. In essence, it treats single scale models as a separate components and facilitates their coupling, whether they are executed at one location or multiple. It is open source software under the LGPL version 3 license and is available at http://apps.man.poznan.pl/trac/muscle. MUSCLE 1 [22] generally had the same architecture and it was based on the Complex Automata theory [23,24] and focussed on multi-agent multiscale computing. The differences between MUSCLE 1 and 2 are discussed in Appendix A.3, and amount to sharing only 4% of their code. The main goal of creating a successor to MUSCLE 1 was to support simulations on high performance computing infrastructures.

Distributed computing is a way to take advantage of limited and heterogeneous resources in combination with heterogeneous multiscale models. There are several motivations for distributing the computation of a multiscale model: to make use of more resources than available on one site; making use of heterogenous resources such as clusters with GPGPUs, fast I/O, highly interconnected CPU's, or fast cores; or making use of a local software license on one

* Corresponding author. Tel.: +31 20 525 7446.
E-mail addresses: J.Borgdorff@uva.nl, joris@jorisborgdorff.nl (J. Borgdorff), bbosak@man.poznan.pl (B. Bosak), krzysztof.kurowski@man.poznan.pl (K. Kurowski), mohamed.benbelgacem@unige.ch (M. Ben Belgacem), bastien.chopard@unige.ch (B. Chopard), d.groen@ucl.ac.uk (D. Groen), p.v.coveney@ucl.ac.uk (P.V. Coveney), A.G.Hoekstra@uva.nl (A.G. Hoekstra).

[1] Mariusz Mamonski (1984–2013) suddenly deceased after the first submission of this article. He had a major role in the development, deployment, tests, support, and software compatibility of MUSCLE 2.

http://dx.doi.org/10.1016/j.jocs.2014.04.004

machine and running a highly parallel code on a high-performance cluster. Projects such as EGI and PRACE make distributed infrastructure available, and software that uses it is then usually managed by a middleware layer to manage distributed computing for users [39].

Quite a few open and generic component-based computing frameworks already exist, for instance the CCA [2] with CCaffeine [1], the Model Coupling Toolkit (MCT) [28,29], Pyre [11], or OpenPALM [12]; see the full comparison by Groen et al. [21]. The Model Coupling Toolkit has a long track-record and uses Fortran code with MPI as a communication layer so it potentially makes optimal use of high-performance machines. OpenPALM uses TCP/IP as a communication layer and it is packaged with a graphical user interface to couple models. Both frameworks provide some built-in data transformations. MUSCLE 2 uses shared memory for models started in the same command and TCP/IP for multiple commands. An advantage over the other mentioned frameworks is that it provides additional support for distributed computing and for Java. However, it has fewer built-in data transformations available and does not provide tools for implementing the contents of single scale models, so it should be combined with domain-specific libraries.

There are many libraries for local and wide-area communications, apart from MPI implementations and the ubiquitous TCP/IP sockets. MPWide [20], for instance, is a lightweight library that optimises the communication speed between different supercomputers or clusters; ZeroMQ [25] is an extensive communication library for doing easy and fast message passing. To use them for model coupling these libraries have to be called in additional glue code. MUSCLE 2 optionally uses MPWide for wide area communication because of its speed and few dependencies.

So far MUSCLE 2 is being used in a number of multiscale models, for instance a collection of parallel Fortran codes of the Fusion community [17], a gene regulatory network simulation [37], a hydrology application [5], and in a multiscale model of in-stent restenosis [10,6,19].

In this paper, we introduce the design of MUSCLE 2 in Section 2, including the theoretical background of the Multiscale Modelling and Simulation Framework, MUSCLE 2's Programming Interface (API) and runtime environment. The performance and startup overhead of MUSCLE 2 is measured in Section 3 in a number of benchmarks. Finally, in Section 4 two applications that use MUSCLE are described, principally a multiscale model of a complex canal system, for which additional performance tests are done.

2. Design

MUSCLE 2 is a platform to execute time-driven multiscale simulations. It takes advantage of the separation between the submodels that together form the multiscale model, by treating each submodel as a component in a component-based simulation. The submodels individually keep track of the local simulation time, and synchronise time when exchanging messages.

A strict separation of submodels is assumed in the design of MUSCLE 2, so the implementation of a submodel does not dictate how it should be coupled to other submodels. Rather, each submodel sends and receives messages with specified ports that are coupled at a later stage. When coupling, modellers face their main scientific challenge: to devise and implement a suitable scale bridging method to couple single scale models. MUSCLE 2 supports the technical side of this by offering several functional components, described in Section 2.1.

The runtime environment of MUSCLE 2 executes a coupled multiscale model on given machines. It can run each submodel on an independent desktop machine, local cluster, supercomputer, or run all submodels at the same location. For instance, when one or more submodels have high computational requirements or require alternate resources such as GPU computing, these submodels can be executed on a suitable machine, while the others are executed on a smaller cluster. A requirement is that a connection can be established between submodels, and that a message can only be sent to currently running submodels. For some models a local laptop, desktop or cluster will suffice; MUSCLE 2 also works well in these scenarios. Technical details about the runtime environment can be found in Appendix A.

MUSCLE 2 is separated into an API, which submodel code uses, a coupling scripting environment that specifies how the submodels will be coupled, and a runtime environment, that actually executes the multiscale model on various machines. The library is independent from the coupling, which is in turn independent from the runtime environment. As a result, a submodel is implemented once and can be coupled in a variety of ways, and then executed on any suitable set of machines. Additionally, future enhancements to the runtime environment are possible without changing the library.

2.1. Theoretical background

To generally couple multiscale models, a framework describing the foundations of multiscale modelling [13,8,27,26] and its repercussions on multiscale computing [7,16] was conceived. It starts by decomposing a phenomenon into multiple single scale phenomena using a scale separation map as a visual aid. Based on these single scale phenomena, single scale models are created; see Fig. 1. Ideally, these single scale models are independent and rely only on new messages at specific input ports, while sending messages with observations of their state at output ports. By coupling output ports to input ports using so-called conduits, a multiscale model is formed. Assuming a time-driven simulation approach, each message is associated with a time point, which should be kept consistent between single scale models.

The theoretical framework distinguishes between acyclically and cyclically coupled models. In the former, no feedback is possible from one submodel to the other, while in the latter a submodel may give feedback as often as needed. This distinction has many computational implications, such as the need to keep submodels active in cyclically coupled models, or the recurring and possibly dynamic need for computing resources. MUSCLE 2 focusses on cyclically coupled models by keeping submodels active during the entire simulation, whereas workflow systems tend to focus on acyclically coupled models.

Listing 1. Submodel execution loop in MUSCLE 2

```
do {
    init()
    while (!endCondition()) {
        intermediateObservation()
        incrementTime()
        solvingStep()
    }
    finalObservation()
} while (restartSubmodel())
```

To facilitate consistency, submodels each have a fixed submodel execution loop as in Listing 1, consisting of initialisation, a loop with first an observation and then a solving step, and then a final observation. This loop can be restarted as long as a submodel with a coarser time scale provides input for the initial condition. During initialisation and solving steps, only input may be requested, and during the observations, only output may be generated. Although this is the general contract, submodel implementations in MUSCLE 2 may diverge from this loop, for example if it would increase performance.

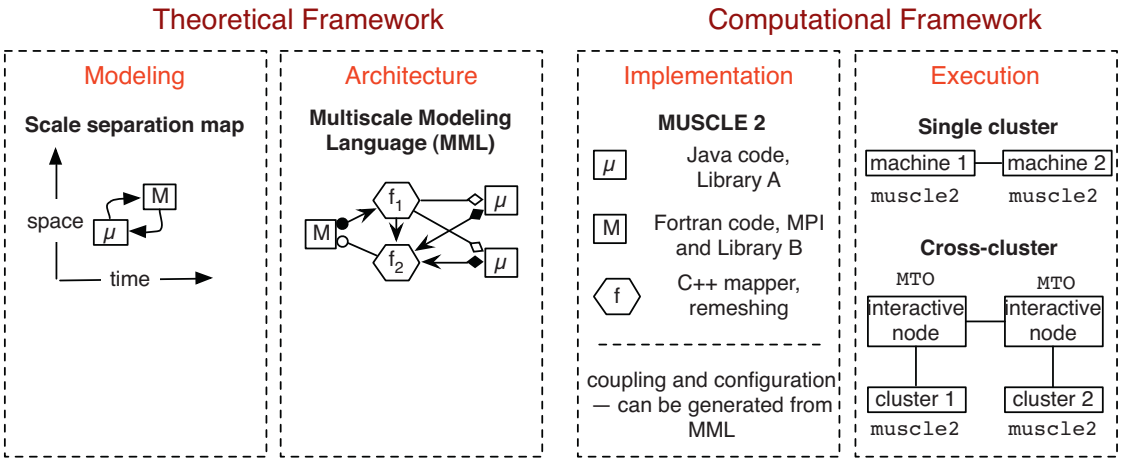


Fig. 1. Overview of the approach presented in this paper. First, use the multiscale modelling and simulation framework to characterise a multiscale model: create a scale separation map (for example with macro model M and micro model μ), and translate it to the computationally oriented multiscale modelling language, including any mappers f_1 and f_2 to do scale bridging and/or data conversions (Section 2.1). The architecture is implemented and coupled with MUSCLE 2 (Sections 2.2 and 2.3), and finally, executed on a single machine or cluster with plain MUSCLE 2 (Section 2.4), or cross-cluster using the MUSCLE Transport Overlay (MTO, Section 2.5).

Submodels should remain independent and as such the data expected by a submodel will not automatically match the observation of another. For this purpose data can be modified in transit, thus implementing scale bridging methods, either by light-weight conduit filters, which change data in a single conduit, or by mappers, which may combine the data of multiple sources or extract multiple messages from a single observation. Finally, the input for a submodel may not be available from another submodel but rather from an external source, or conversely, an observation might only be used outside the model. In that case, terminals may be used: sources to provide data and sinks to extract data.

Each of the concepts mentioned in this paragraph is defined in the multiscale modelling language (MML). In MML these concepts are well-defined and accessible for automated analysis, for example to predict deadlocks or the total runtime of a simulation. Also, the configuration file of MUSCLE 2 can be generated from MML.

2.2. Library

The library part of MUSCLE 2 consists of Java, C, C++, Python, Fortran APIs and coupling definitions in Ruby. The API's for these languages can each: send and receive arrays, strings, and raw bytes; do logging; and stage in- and output files. Other programming languages and additional libraries may use the MUSCLE 2 API as long as it can interface with one of the listed programming languages. Send calls have non-blocking semantics whereas receive calls are blocking by default but may be used as non-blocking instead. An example of sending and receiving data with MUSCLE 2 is shown in Table 1. The formal submodel loop in Listing 1 is advised but not enforced.

The Java API, in addition to the API's of the other languages, allows implementing formal MML constructs such as formal submodels, filters, and mappers, and sending and receiving advanced data structures like Java classes. Because MUSCLE 2 allows multiple languages in a multiscale model, filters and mappers can also be used in models that otherwise do not use Java. In all cases, the API is non-invasive and does not force a certain programming paradigm, which makes it straightforward to incorporate in existing code.

2.3. Configuration

The configuration of a multiscale model and the coupling is done in a Ruby file. In this file, submodels and their scales are specified, parametrised, and coupled to each other. Mappers, conduit filters, sources and sinks are also added to the coupling topology here. A conduit can be configured with multiple filters; predefined-filters such as data compression, or custom filters such as data transformations or conversions. Because the configuration is a Ruby script the coupling topology can be automatically generated, for instance to set up a ring or grid topology, or to read a network from a file.

Below is an example of the configuration of a multiscale model with one macro-model and one micro-model, with a single coupling from macro to micro.

```
# Add the classpath of the submodels, using environment
# variable $MODEL_HOME
add_classpath ENV['MODEL_HOME'] + '/build/classes'

# Create a submodel instance 'macro' with implementing
# Java class 'mypackage.Macro'
macro = Instance.new('macro', 'mypackage.Macro')

# Set the macro timestep to 1 hour, and the total
# simulation time to 1 day
macro['dt'] = '1 hour'
macro['T'] = '1 day'

# For 'micro', use a predefined MUSCLE MPI submodel
# and specify the executable
micro = Instance.new('micro',
             'muscle.core.standalone.MPIKernel')
micro['command'] = ENV['MODEL_HOME'] + '/build/micro'

# Couple the port 'macroVariable' of macro to port
# 'environValue' of micro
macro.couple(micro, 'macroVariable' => 'environValue')
```

2.4. Runtime environment

The runtime environment of MUSCLE 2 is designed to be light-weight and portable, and to provide high performance. MUSCLE 2 is supported on Linux and OS X. Data is transmitted between submodels and mappers, both called instances, using direct and thus decentralised message passing.

Each MUSCLE 2 simulation has a single Simulation Manager and one or more Local Managers, as shown in Fig. 2. The Simulation Manager keeps track of which instances have started and what their location is. The Local Manager starts the instances that were assigned to it in separate threads and listens for incoming connections. Instances will start computation immediately but they will block and become idle as soon as they try to receive a message that

Table 1
Sending (first row) and receiving (second row) a message in MUSCLE 2 in various programming languages.

Java	C++	Fortran

`double[] dataA = new double[100];`
`out("portA").send(dataA);`
	`double* dataA = new double[100];`
	`muscle::env::send("portA", dataA,`
	`100, MUSCLE_DOUBLE);`
	`delete [] dataA;`
		`real*8 :: dataA(1:100)`
		`call MUSCLE_Send("portA", dataA,`
		`%REF(100), %REF(MUSCLE_DOUBLE))`

`double[] dataB = (double[])`
`in(''portB'').receive();`
	`size_t len;`
	`double* dataB = (double*)`
	`muscle::env::receive("portB",`
	`(void*)0, len, MUSCLE_DOUBLE);`
	`muscle::env::free_data(dataB,`
	`MUSCLE_DOUBLE);`
		`integer :: len`
		`real*8 :: dataB(1:100)`
		`call MUSCLE_Receive("portB", dataB,`
		`len, %REF(MUSCLE_DOUBLE))`
		`call MUSCLE_Free(dataB,`
		`%REF(MUSCLE_DOUBLE))`

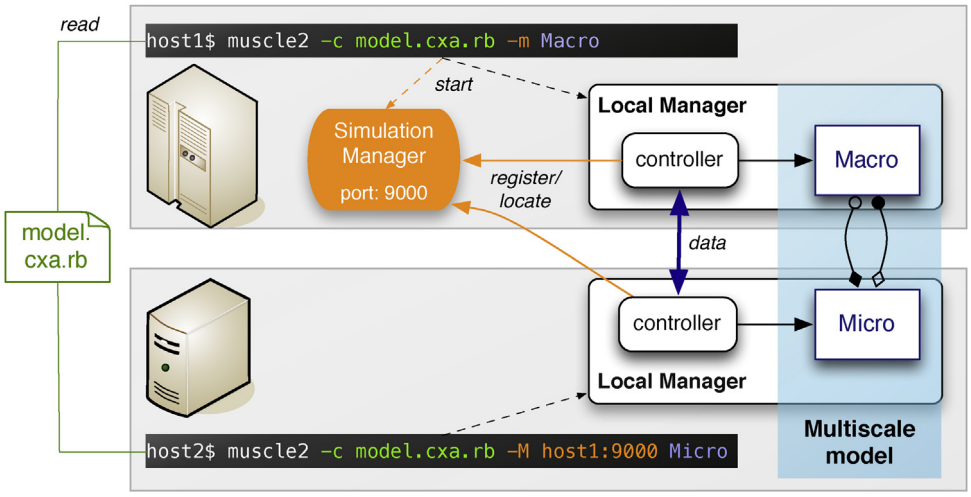


Fig. 2. An example of the MUSCLE runtime environment, explained in Section 2.4, doing a distributed execution of the multiscale model described in file `model.cxa.rb` on machines `host1` and `host2`. The register and data arrows are both TCP/IP connections. The Macro and Micro rectangles make up a multiscale model.

has not yet been sent, or try to send a message to an instance that has not been started.

A so-called native instance is a compiled instance that runs as a separate executable. Its controller is still implemented in Java and therefore the executable will try to establish a TCP/IP connection with this controller, which will then do all communication with other instances and with the Simulation Manager. A native instance may be serial or use threading, OpenMP, or MPI.

Message-passing mechanisms that are used are shown in Fig. 3. Messages within a Java Virtual Machine are sent using shared memory. To insure independence of data between instances, the data is copied once before it is delivered from one instance to the other unless otherwise specified. Messages between Local Managers and between the Local and Simulation Managers are sent over TCP/IP, which is available everywhere and inherently allows communication with between machines. The MessagePack serialisation library [32] is used for these communications because of its efficient packing. The connection between a native instance and its controller uses the XDR [36] serialisation library because it is already installed in most Unix-like systems.

2.5. Cross-cluster computing

Because MUSCLE 2 uses TCP/IP for message passing between instances, it can communicate over the internet and within clusters. However, most HPC systems prevent direct communication between submodels running on different clusters. Therefore, MUSCLE 2 provides the user space daemon MUSCLE Transport Overlay (MTO). It runs on an interactive node of an high-performance cluster and will forward data from MUSCLE 2 instances running on the cluster to the MTO of another cluster, which will forward it to the intended MUSCLE 2 instance. By default, it does this with plain non-blocking TCP/IP sockets, but it can also use the MPWide 1.8 [20] library. MPWide's goal is to optimise the performance of message-passing over wide-area networks, especially for larger messages.

Because the MUSCLE 2 instances that make up a distributed simulation have to run concurrently and their in- and output files have to be managed, cross-cluster simulations are tedious to do manually. For this reason the MUSCLE 2 framework was integrated with the QosCosGrid (QCG) middleware stack [9]. The QCG middleware stack offers users advanced job management and resource management capabilities for e-Infrastructure. It will manage the execution of a distributed MUSCLE 2 simulation from a central location, reducing the input and management needed from the user.

3. Performance

The main benefit of MUSCLE 2 is the library and coupling environment that it provides. However, for many if not all multiscale simulations, performance is equally important. The performance of MUSCLE 2 has two aspects: the overhead it introduces and the messaging speed that it provides. These were measured on four resources: an iMac (a local desktop machine), Zeus (a community cluster accessible through EGI or PL-GRID), Huygens (a PRACE Tier-1 resource from SurfSARA, the Netherlands) and SuperMUC (a PRACE Tier-0 resource from Leibniz-Rechenzentrum, Germany). See Table 2 for their details.

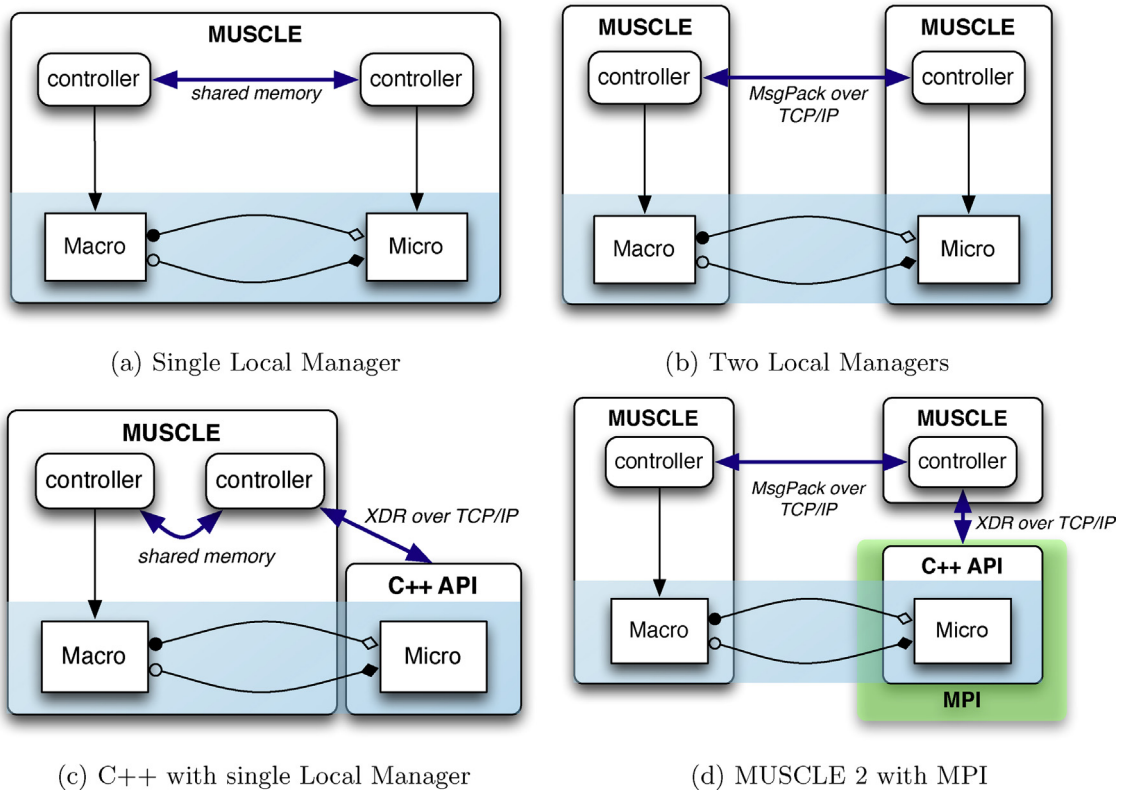


Fig. 3. Different ways in which MUSCLE 2 can be executed. A box with the MUSCLE label indicates a Local Manager. Thick edges indicate communication between instances. They are labelled with the means of communication.

Table 2
Computing resources used in performance testing.

Name	Infrastructure	Location	Processor	Cores per node
iMac	Local desktop	Amsterdam, The Netherlands	Intel i3 3.2 GHz	2/4[a]
Zeus	PL-Grid, EGI	Krakow, Poland	Intel Xeon 2.4 GHz	4
SuperMUC	PRACE Tier-0	Munich, Germany	Intel Xeon 2.7 GHz	16/32[a]
Huygens	PRACE Tier-1	Amsterdam, The Netherlands	IBM Power6 4.7 GHz	32/64[a]
Cartesius	PRACE Tier-1	Amsterdam, The Netherlands	Intel Xeon 2.7 GHz	12/24[a]
Gordias	Local cluster	Geneva, Switzerland	Intel Xeon 2.4 GHz	8
Scylla	Local cluster	Geneva, Switzerland	Intel Xeon 2.4 GHz	12

[a] Two HyperThreads per core.

3.1. Overhead

With MUSCLE's runtime overhead figures, a user can estimate what the impact of MUSCLE will be on the execution time and memory for a given multiscale model. To test the overhead we will start a varied number of submodels and conduits, to evaluate their impact on CPU and memory usage. The overhead will be measured on an iMac and on SuperMUC.

To test the overhead in execution time, MUSCLE is started with 30 different submodel counts n, from 1 to 1000, and 36 different conduit counts m, from 0 to 50,000. The submodels are created in a configuration script, in which each submodel adds a conduit to each of the following m/n submodels, wrapping around to the first submodel if there are less than m/n succeeding submodels. Once the simulation has started, each submodel sends and receives one empty message through each conduit, and then exits. This way, all submodels must be active simultaneously for a small amount of time, like they would be in a normal simulation. Since a submodel with native (C/C++/Fortran) code needs to start an additional executable, it is measured separately. The amount of time spent on Java garbage collection is not separately measured. Software versions on iMac are Java 1.7.0_7 and Ruby 1.9.3; on SuperMUC they are Java 1.6.0_32 and Ruby 1.8.7. The minimal overhead a is determined by taking the minimum value encountered. The additional runtime overhead b per submodel and c per conduit is determined by fitting the data to the equation $a + bn + cm$, where n are the number of submodels and m are the number of conduits. The additional runtime per native submodel was fitted to a linear curve separately. The minimum memory overhead was taken as the memory of running with one submodel, all other values were separately fitted to a linear curve.

The results for this experiment are listed in Table 3. With the highest number of submodels and conduits (1000 and 50,000 respectively), execution took 7.1 seconds on the iMac and 6.6 seconds on SuperMUC; the lowest runtimes were 0.68 and 1.2 s respectively. For most if not all multiscale simulations, even 7.1 s overhead will not be significant compared to the overhead of running the simulation, and for multiscale simulations with less than 10 submodels, the overhead will be close to a second.

The memory consumption was measured in a similar way as runtime overhead, except here ten submodel counts from 1 to 1000 where used, and separately thirteen conduit counts from 0 to 50,000, each started four times. The Java Virtual Machine of the Local Manager was set up with an initial heap size of 1 GB

Table 3
Runtime and memory consumption[a] of MUSCLE, on a local iMac and the PRACE machine SuperMUC (see their details in Table 2). Entries marked '–' were not measured. We assume that memory consumption on both machines is similar, since they both use 64-bit Intel processors. The first row (Overhead) indicates the overhead of MUSCLE without starting any submodels, the other rows show additional overhead to this baseline.

	iMac runtime	SuperMUC runtime	iMac memory
Overhead	0.77 s	1.2 s	73 MB
Additional per submodel	1.6 ms	1.6 ms	168 kB
Additional per local conduit	0.11 ms	0.10 ms	3.4 kB
Additional per port with remote coupled port	–	–	1.1 MB
Additional per native submodel	24 ms	–	1.7 MB

[a] Stated memory sizes are multiples of 1024 (kilo)bytes.

and with a maximum heap size of 3 GB. Since MUSCLE uses Java and Ruby, exact memory consumption will differ per execution and it will include free space that their respective runtime engines have reserved. However, with enough memory allocation a trend does emerge. If multiple MUSCLE instances are started for a single multiscale model, additional buffers need to be reserved for communicating with other Local Managers. Therefore ports that are coupled to a port of a submodel with another Local Manager are measured separately, as are submodels with native code.

The results are listed in Table 3. With these figures, and taking into account the memory consumption of the individual submodels, a user can estimate how many submodels will fit in the memory of a single machine. As a result of the allocated buffers, ports coupled to a port on an other Local Manager take a large amount of memory, 1.1 MB. Similarly, a native executable linked to MUSCLE uses at least 650 kB of memory, and in Java an additional serialisation buffer is allocated. On a machine with 4 GB of memory per core, each core could accommodate up to 20,000 submodels with 10 local conduits each, up to 350 submodels with 10 remote conduits, or up to 300 native submodels with 10 remote conduits. In most scenarios this is more than sufficient, and the number of submodels will instead be limited by the computational cost of the submodel code.

3.2. Message speed

The performance of MUSCLE communication is compared with approaches that modellers would usually use for composite models. The two most prevalent methods of communication of our current users are file-based or MPI-based. The former is often used to couple different codes together, whereas the latter is used to form fast monolithic codes. For remote communication, GridFTP is a popular alternative and MPWide is a well-optimised one. We will compare these methods with the communication speed offered by MUSCLE 2. Both latency and throughput of the methods will be computed.

3.2.1. Single machine

For the local communications we will compare speeds of file copy, MPI, MUSCLE 1 and MUSCLE 2. Each of the tests is done with message size 0 kB and 2^i kB, with i ranging from 0 to 16, which is up to 64 MB. Since MUSCLE 1 will not send messages larger than 10 MB, its measurements are limited to i ranging from 0 to 13 (8 MB). Per message size, a message is sent back and forth 100 times, so it makes 100 round trips. The time to send one message of a certain size is calculated as the average over the round trip times, divided by two. The latency is calculated as the minimum time to send a message. The message times are then fitted to a linear curve $ax + b$ for message size x, where throughput is calculated as $\frac{1}{a}$ and b is taken as the latency.

For applications without a coupling library, a simple way to transfer data from one process to another is to write to a file which another process may read. The operating system might cache this file so that the read operation is fast. This scenario is simulated by creating files as messages. One round trip is taken as copying a file and copying it back using the systems file copy, which is equivalent to writing and reading a file twice.

For a monolithic model, possibly with multiple substructures or threads, MPI is a well-known and very fast option. This paradigm, however, gives none of the plug and play advantages that MUSCLE 2 has, nor does it keep time in sync between submodels, nor is it easy to combine resources of different providers. In our experiment, messages are sent by one MPI process, then received and sent back by another with the same executable.

To test MUSCLE 2, first we take the situation that all instances have a Java implementation and a single machine is sufficient to run them. As described in Section 2.4, messages are then sent through shared memory. Next, we take two MUSCLE 2 processes that communicate with TCP/IP, for when a user wants to prioritise one process over the other, for instance. Finally, we take two instances that both have a C++implementation.

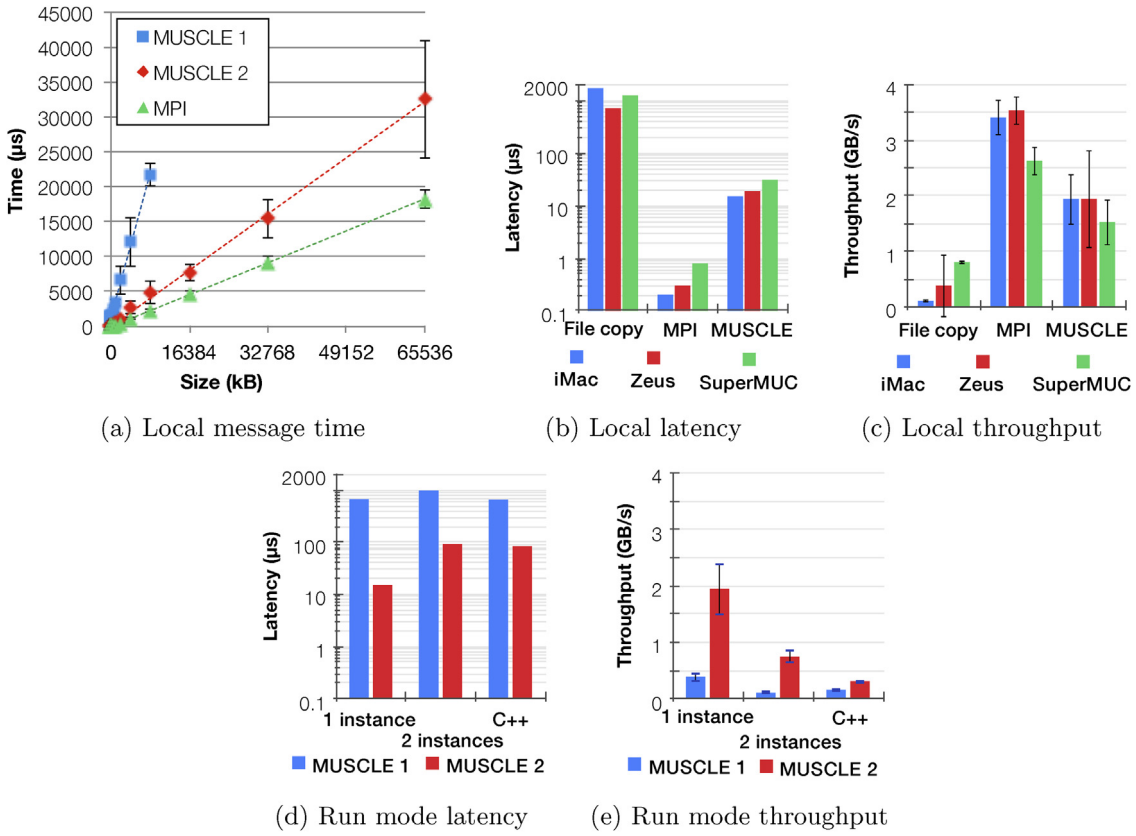
The file copy, MPI and MUSCLE 2 scenarios are tested on the iMac (local desktop), Zeus (cluster), and SuperMUC (supercomputer). MUSCLE 1 is only tested on the iMac due to portability issues.

The results are plotted in Fig. 4. The standard deviation for the latency is very low and is not shown. Obviously, copying data has a higher latency and lower throughput than the alternatives. The latency of MPI is clearly the lowest and MPI has the highest throughput as well, which would be expected because it uses highly optimised native code. MUSCLE falls in the mid-category, and is thus a serious contender if neither a monolithic nor a file-based simulation is desired. These results do signal that for optimal performance of a very tightly integrated code, MPI could be preferred over MUSCLE 2. Of course, this MPI code can then be used in MUSCLE 2 as a single submodel, so that MUSCLE can take care of starting the submodel and coupling it with other codes.

Comparing the run-modes of MUSCLE, and MUSCLE 1 and MUSCLE 2, a few remarks can be made. First, the latency of C++is lower than having two Local Managers, which is surprising: with C++a message is first serialised with XDR, sent, passed through shared memory in Java and then serialised again to be sent to another C++program. With two Local Managers, a message is serialised once with MessagePack and directly used. So although the throughput of MessagePack is higher, its latency is worse. Second, MUSCLE 1 falls far behind MUSCLE 2 in all cases, since it uses the JADE system to send messages and overall has a less optimised code. That the performance of MUSCLE 1 is most similar to MUSCLE 2 in the C++scenario, is because the Java Native Interface (JNI) transfers data between Java and C++faster than TCP/IP sockets can. JNI was removed in MUSCLE 2 due to the portability issues that it caused, see Appendix A.3 for the details.

3.2.2. Distributed computing

Besides local message speed, distributed message speed is also important for computing on large infrastructures. Although the main bottleneck will usually be the available network bandwidth, software does have an influence on message speed. In this section we will compare the speed of three possible technologies to



(a) Local message time (b) Local latency (c) Local throughput

(d) Run mode latency (e) Run mode throughput

Fig. 4. The performance of the communication methods described in Section 3.2. (a) shows a plot of the time to send a single message, along with linear fit. (b) and (c) show the performance of sending a message within a machine, for three machines (the latency is averaged as it showed little variation). (d) and (e) show the performance of sending messages on a local machine with MUSCLE 1 and MUSCLE 2, by starting a model: in a single MUSCLE instance; with two coupled MUSCLE instances; or, with C++submodels. The standard error of the latency measurements was negligible so it is only shown for the throughput.

do wide area network communication: MPWide 1.8, GridFTP 0.8.9, and MUSCLE 2 with the MTO. MPWide is designed specifically for optimally making use of the available bandwidth by using packet pacing, multiple streams per connection and adapted buffer sizes. GridFTP [18] is a dedicated file transfer service run by EGI and PRACE sites. MUSCLE uses the MTO, which by default uses a single plain TCP/IP socket per connected MTO but can also be used in conjunction with MPWide.

Each test was performed between a PRACE Tier-1 site Cartesius in Amsterdam and the PRACE Tier-0 site SuperMUC in Garching, Munich (more details in Table 2). They send a message from Amsterdam and back again, using message sizes 0 kB and 2^i kB, with i ranging from 0 to 20, which is up to 1 GB. For each message size up to 1 MB, hundred messages were sent, for messages ranging from 2 MB to 1 GB ten messages were sent. The TCP/IP route from Cartesius to SuperMUC uses the high-speed PRACE network. The average ping time over 50 consecutive pings on this route was 15.2 ms.

In all applications the standard settings were used. For MPWide the number of streams must be specified and was set to 128 streams. Although GridFTP can open multiple TCP streams for a transfer, firewall settings prevented it to do so from SuperMUC to Cartesius, so in this experiment it used only one.

The results of the test are shown in Fig. 5. For very small messages both MUSCLE 2 with the MTO and MPWide come very close to the ping time, adding up to 2 ms. When the MTO uses MPWide internally the latency goes up considerably because it uses an additional management layer. GridFTP has to do a certificate handshake before when connecting, which takes significantly longer at about 890 ms. With large messages its performance is much better, at 90 MB/s, although it does show an occasional bump when the hand-shake can not be processed immediately. MUSCLE 2 with the MTO did a bit worse and MUSCLE 2 with the MTO using MPWide did a bit better. Plain MPWide performance was much better than the other methods for messages larger than 128 kB. This indicates that further efforts to integrate the MTO and MPWide may be beneficial.

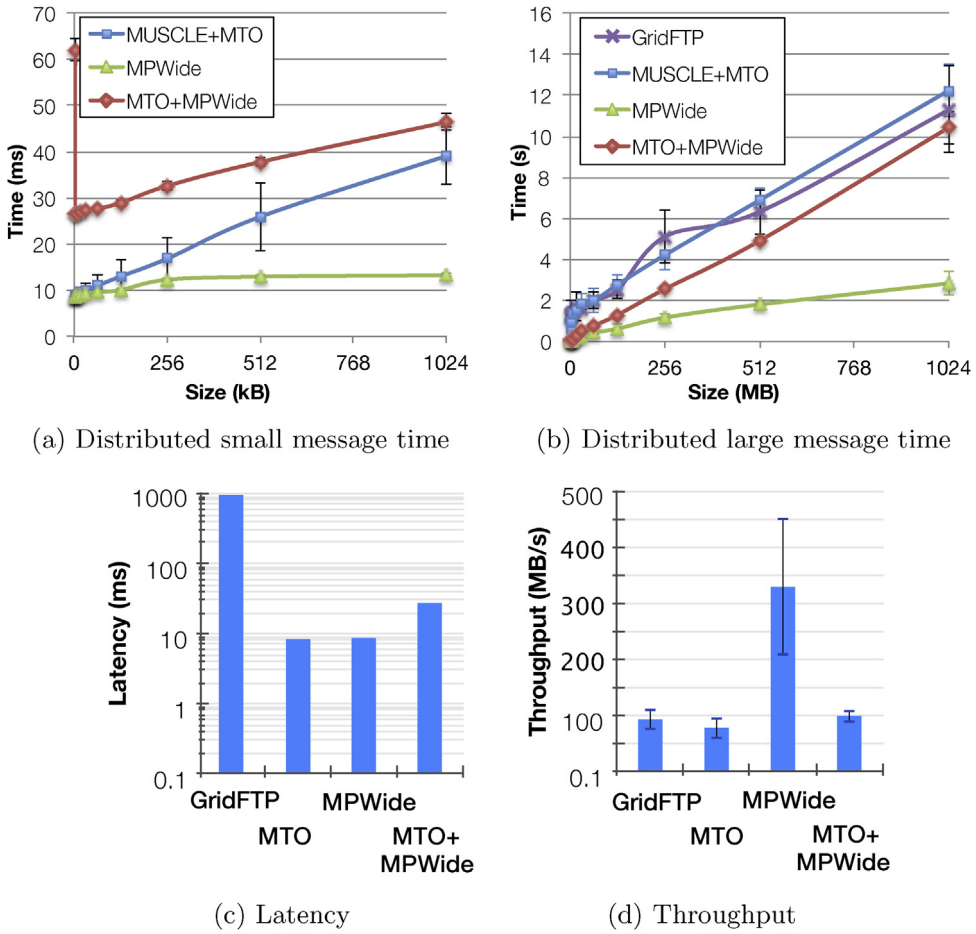
4. Use cases

To show the real-time usage of MUSCLE 2 as well as its practical performance we will show how it is applied to a multiscale model of a canal system and specifically to the submodel of one canal section. Next, a multiscale model of in-stent restenosis shows the heterogeneity of submodels that can be coupled.

4.1. Hydrology application

An optimal management of rivers and waterways is a necessity in modern society to ensure an adequate supply of water, in particular for agriculture, electricity production or transportation [31]. An important requirement is to control the water level and sediment transport in populated areas [30]. These problems can be addressed through computer simulation in combination with optimisation methods.

Many of such hydrology problems can be implemented using a “Lego based philosophy” [4,5], where river or water sections are modelled by submodels and connected with mappers, based on the topology of existing canal systems. A submodel can for instance implement a 3D Free Surface (3DFS) model and be connected to a 1D shallow water submodel. Because of their different resolution and time step this gives a multiscale system. The decomposition into



(a) Distributed small message time

(b) Distributed large message time

(c) Latency

(d) Throughput

Fig. 5. The performance of sending a message between Huygens and SuperMUC. (a) shows the time to send a message on a kilobyte scale, and excludes GridFTP which fluctuates around 890 ms. (b) shows the time to send a message on a megabyte scale. The other two plots show the fitted values of the data.

submodels allows a distributed execution, which may be necessary for larger canal systems.

Our use case consists of a 3D cavity flow problem solved with the Lattice Boltzmann (LB) [35] numerical method. The submodels are implemented with the Palabos toolkit,[2] which uses MPI for parallelisation. The aim is to evaluate the time overhead induced by the use of the MUSCLE API when performing distributed computations of hydrodynamical problems. As illustrated in Fig. 6, the computational domain (here a 3D cavity) is divided across several parallel clusters and information should be exchanged between them at each iteration. This use case itself has full scale overlap but will be coupled to different time scales when a canal system is simulated.

Listing 2. Pseudo-code of the cavtiy3d example.

```
f_init()
for (iterations < maxIterations) {
    collideAndStream()
    gatherBoundaryData()
    sendReceiveBoundaryData()
    updateBoundaryData()
}
```

Listing 2 gives the pseudo-code of the algorithm used by the numerical method. During each loop iteration (line 2), the submodel computes the flow on line 3 using the parallel Lattice Boltzmann method. On Line 4, each submodel retrieves boundary data from all MPI processes in the same job and submodel. Line 5 establishes the coupling between submodels. In this case, the submodel sends and receives boundary data using the MUSCLE API hidden in the `sendReceiveBoundaryData()` function. On line 6, each section updates its boundary according to the data received from the other submodels.

To show the performance of MUSCLE 2 when it is used in an actual problem, we will consider the performance of the 3D cavity submodel described above. Our benchmark will consist of running a monolithic code first and comparing its runtime with using two MUSCLE submodels. A detailed treatment has been made by Belgacem et al. [4]; here we show some results obtained by using more CPUs.

The computational domain of the canal we will use, as depicted in Fig. 6, has a length L_x of 13,000 m, a width L_y of 40 m and a depth L_z of 10 m. The spatial resolution Δx may vary and will determine the problem size: decreasing Δx implies increasing the domain size.

For the benchmark, we will evaluate three scenarios:

1. a monolithic simulation of the canal on a single cluster;
2. a simulation with two canal submodels on the same cluster, coupled using MUSCLE; and
3. a simulation with two coupled canal submodels on different clusters, coupled using MUSCLE 2.

The first case shows what the performance of a usual monolithic model with MPI is, the second what the cost is in splitting that

[2] Palabos: http://www.palabos.org/.

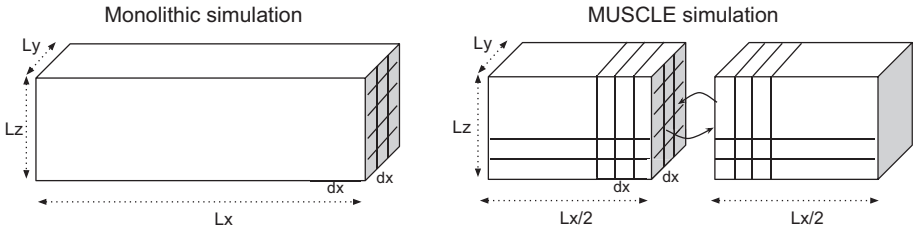


Fig. 6. How a 3D cavity can be split into equal parts for use with MUSCLE, where L_x is its length, L_y its width and L_z its depth, and dx the resolution at which it is resolved.

into multiple parts using MUSCLE, and the third what the cost is of distributing it with MUSCLE.

The execution time of these scenarios is indicated T_{mono}, T_{local}, and T_{distr}, respectively. In scenarios 2 and 3, the canal section computed in scenario 1 is split equally amongst the submodels called left and right. Each simulation carries out 100 iterations, and this is repeated three times. We varied the number of grid points per metre $N = \frac{1}{\Delta x}$ from 0.5 to 4, with a step size of 0.5. For the total domain this means varying the problem size from under 820 thousand grid points to over 340 million points, scaling with N^3. The MUSCLE communication volume, however, only scales with N^2, so computation will dominate computation for increasing N.

The simulations are run on the Gordias and Scylla clusters (for their details see Table 2). The monolithic execution is done with 100 cores of the Gordias cluster. Likewise, the MUSCLE execution is done with 100 cores, but here the left and right section run on 50 cores each. The local MUSCLE execution is run on Gordias whereas the in distributed one, both clusters are used. In the local execution we first ran the MUSCLE Simulation Manager in a separate node so that it had a fixed address before the job started. In the distributed scenario, QCG-Broker takes care of queuing the jobs and starting the Simulation Manager.

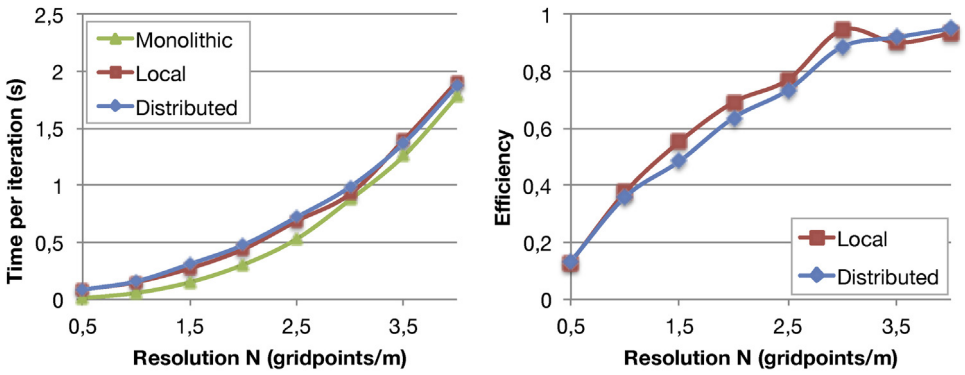
Fig. 7(a) shows the results of the benchmark of T_{mono}, T_{local} and T_{distr} on Gordias cluster. We measured the average time per iteration. If we compare T_{mono} and T_{local}, we see that the difference between execution times varies very little over all values of Δx. The main bottleneck seems to be a fixed synchronisation overhead due to waiting for messages between submodels.

Regarding the distributed execution (Fig. 7(b)), the efficiency values $\varepsilon_{\text{distr}} = T_{\text{mono}}/T_{\text{distr}}$ and $\varepsilon_{\text{local}} = T_{\text{mono}}/T_{\text{local}}$ show the same behaviour; i.e, we observe a large communication ratio with small values of N, and vice versa. This goes to the extent that for $N = 4$, using only MPI is just 5% more efficient than using MUSCLE. $\varepsilon_{\text{distr}}$ is smaller than $\varepsilon_{\text{local}}$ for smaller problem but for large values of N $\varepsilon_{\text{distr}}$ is slightly higher, which can be explained if we look at the detailed plots.

The runtimes of the two sections on the Gordias cluster, per operation of the pseudo-code 2, are very similar, as shown in Fig. 8(a) and (b). For large N, the fraction of time spent actually calculating increases steadily. For smaller N, however, most of the time is spent in waiting for messages from the other submodel, so if one submodel was slower then the other would have to wait until it was finished and vice-versa. In the distributed experiment however, the submodel on Scylla (Fig. 8(d)) was computed consistently faster, which means the submodel on Gordias (Fig. 8(c)) needs to wait far less. This gives a lower average time per iteration for situations that depend more on computational time than on communication time.

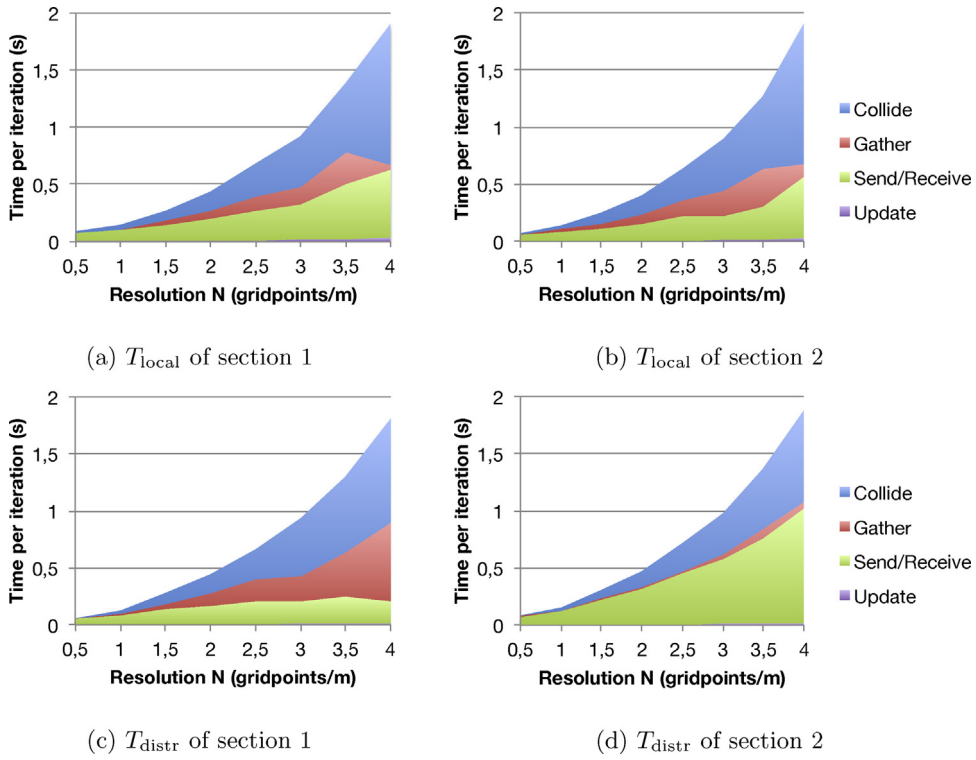
4.2. ISR3D

The three-dimensional model of in-stent restenosis in coronary arteries (ISR3D), covered in [6,19] and first described in [10], originally used MUSCLE 1 which was replaced by MUSCLE 2. In-stent restenosis is the recurrence of stenosis after stenting a stenosed blood vessel. The model is multiscale in its time scale, where smooth muscle cells proliferation in the blood vessel wall is modelled on a time scale of hours to months, and the blood flow and shear stress is computed on a time scale of milliseconds to a second. It couples a submodel using the C++API with OpenMP to another using C++with MPI, and uses Fortran and Java in other submodels and mappers. ISR3D has routinely been executed between sites in different countries using MUSCLE with the MTO, running the parallel submodels on a highly parallel machine and the serial parts on another site. When the original custom blood flow submodel code needed to be replaced with the Palabos library, the plug-and-play character of MUSCLE 2 proved useful, since other submodels did not have to be altered in this operation.



(a) Runtime (b) Efficiency compared to monolithic runtime

Fig. 7. The performance of the three execution scenarios of the cavity 3D model, for the number of grid points per metre N, as described in Section 4.1.



(a) T_{local} of section 1 (b) T_{local} of section 2

(c) T_{distr} of section 1 (d) T_{distr} of section 2

Fig. 8. The runtime of different operations in the local and distributed cases, where (a)–(c) run on the Gordias cluster and (d) runs on the Scylla cluster, and (a) and (b) are run concurrently as are (c) and (d). The operations match the pseudo-code in Listing 2: `collideAndStream()` [Collide]; `getBoundaryData()` [Gather]; `sendReceiveBoundaryData()` [Send/Receive]; and `updateBoundaryData()` [Update].

5. Conclusions

In this contribution, we have introduced and discussed the component-based and flexible design of MUSCLE 2, and its distributed computing capabilities. It is based on a general approach to multiscale modelling and simulation [7,24,23] combined with the multiscale modelling language [16,7]. Because of its modular setup, clearly separating API, coupling, and runtime environment, users can modify parts of a multiscale model without affecting the rest. A multiscale model implemented with MUSCLE 2 can be executed on distributed computing resources at any stage. Moreover, submodel code written in Java, C, C++, Python, or Fortran, and using serial code, MPI, OpenMP, or threads can freely communicate with other submodels using different technologies.

The overhead of starting MUSCLE 2 for multiscale models with a reasonable amount of submodels is shown to be low, both time- and memory-wise. For local computing MUSCLE 2 is shown to be more efficient than file based message passing, but it has a factor two lower throughput than MPI and up to 30 μs higher latency. For parts of a multiscale model where MPI is better suited, such as performing a lattice method or doing agent based simulations, MUSCLE 2 can simply run that part as a submodel with MPI, and the multiscale model will still have the advantages of flexible coupling and execution.

For distributed computing, the MUSCLE Transport Overlay transfers data from one high-performance computing centres to another. Its efficient transfers easily surpass GridFTPs speed for smaller messages and give performance similar to GridFTP for large messages. Using MTO with MPWide gives slightly better performance on the high-speed PRACE network, but plain MPWide still much faster, so the integration between the MTO and MPWide will be further examined.

For a canal system model, MUSCLE 2 makes it easier to generate canal topologies by flexible coupling and being able to distribute different parts of the canal system. Moreover, for canal sections with sufficiently large problem sizes, the performance of MUSCLE 2 is competitive with using a single monolithic code. It will need distributed computing for larger problems when a local cluster does not provide enough resources; this turns out not to be very detrimental to performance.

Acknowledgements

We would like to thank Jules Wolfrat and Axel Berg at SurfSARA, Amsterdam for providing access to the Huygens and Cartesius machines. The work made use of computational resources provided by PL-Grid (Zeus cluster) and by hepia in Geneva (Gordias cluster).

This research presented in this contribution is partially supported by the MAPPER project, which receives funding from the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement N RI-261507.

Appendix A. Technical details of the MUSCLE 2 runtime environment

To increase the separation between the model and the runtime environment each mapper or submodel has its own instance controller that will do the actual communication with other parts of the simulation. When an instance controller starts up it first tries to register to the Simulation Manager. It then queries the Local Manager for the location of all the instances that it has a sending conduit to. The Local Manager will then query the Simulation Manager in a separate thread if it does not know the location. When an instance is finished, its instance controller will deregister it at the Simulation Manager.

Although each instance controller and thus each instance uses a separate thread by default, it is also possible to implement submodels asynchronously. MUSCLE 2 will be able to manage a large

number of light asynchronous submodels in a small number of threads. This leads to both lower memory usage and faster computation since there are far fewer thread context switches but it makes the submodel code slightly more complex and, if not properly coded, prone to race conditions.

Error handling, throughout the program, is designed to work fail-fast. If an uncaught exception occurs in one instance, MUSCLE 2 assumes that continuing the simulation will not yield valid results and it will try to shut down all other instances. This behaviour was implemented to prevent wasting resources on systems that charges end users for the total wall-clock time used by a simulation. It also prevents deadlocks when an instance still expects data from another that has already quit. MUSCLE 2 does not provide error recovery, instead each submodel should handle its own checkpointing, if needed.

A.1. Implementation of the MUSCLE Transport Overlay (MTO)

The MUSCLE 2 Transport Overlay (MTO) is a C++user space daemon. It listens for connections from MUSCLE 2 on a single cluster, and keeps in contact with MTO's on other clusters. It forwards any data from MUSCLE 2 intended for another cluster to that clusters MTO. To identify the MTO associated to a MUSCLE 2 TCP/IP address, each MTO mandates a separate port range to MUSCLE 2.

The default connection between MTO's uses plain non-blocking TCP/IP sockets, and this is well tested. To optimise speed over wide area networks, it has a local buffer of 3 MB and it will prefer sending over receiving up to the point that it will not allow more incoming data if the send buffers are too large or numerous. The MPWide 1.8 [20] library is optionally enabled for connections between MTO's. MPWide is a library to optimise message-passing performance over wide-area networks, especially for larger messages. This option currently only works between a pair of MTO's and the performance depends on the connection between the clusters, but there are ongoing efforts to increase the compatibility.

A.2. QosCosGrid and MUSCLE 2 integration

We identified two main integration points of the QosCosGrid software stack and MUSCLE 2. First, the location (IP address and port) of the MUSCLE Simulation Manager can be exchanged automatically with other MUSCLE Local Managers via the QCG-Coordinator service – a global registry that offers blocking call semantics. Moreover, this relaxes the requirement that the Simulation Manager and Local Managers must be started in some particular order. The second benefit of using the QosCosGrid stack with MUSCLE is that it automates the process of submission of cross-cluster simulations by: co-allocating resources and submitting on multiple sites (if available, using the Advance Reservation mechanism); staging in- and output files to and from every system involved in a simulation; and finally, allowing users to peek at the output of every submodel from a single location.

A.3. Comparison between MUSCLE 1 and MUSCLE 2

MUSCLE 2's main goal is to run (distributed) multiscale simulations on high performance computing infrastructure. The largest changes in MUSCLE since MUSCLE 1 involve decoupling functionalities. The separation between the library and runtime environment makes the system more usable, since users now do not need to go through MUSCLE internals to do basic operations like getting model parameters, and this in turn makes submodel code less susceptible to being incompatible with newer versions of MUSCLE. The separation of C/C++/Fortran code from the main Java code makes compilation much more portable. Finally, the separation of message passing code and the communication method allows choosing more efficient serialisation and communication methods when able.

In terms of portability, MUSCLE 2 comes with all Java prerequisites so they do not have to be installed manually. Moreover, the number of required Java libraries has been drastically reduced. Notably, MUSCLE 2 no longer relies on the Java Agent Development Environment (JADE) for its communication. This way, the MUSCLE 2 initialisation sequence and communication routines are more transparent, which in turn lead to numerous performance enhancements to communication protocols and serialisation algorithms. As a result, MUSCLE 2 can handle messages up to a gigabyte, while MUSCLE 1 will not handle messages larger than 10 MB. Although distributed execution was already possible with MUSCLE 1, it only worked for specifically set up environments, whereas MUSCLE 2 will run with most standard environments.

In MUSCLE 1 the Java Native Interface (JNI) was used to couple native instances. Although JNI is an efficient way to transfer data from and to Java, it gave MUSCLE 1 usability and portability issues and introduced incompatibilities with OpenMP and MPI. In MUSCLE 2, submodels must link to the MUSCLE 2 library instead, at a penalty of doing communications between Java and C++with the somewhat slower TCP/IP.

Additional new features of MUSCLE 2 include a CMake-based build system, having standardised and archived I/O handling, more flexible coupling, and automated regression tests.

References

[1] B.A. Allan, R. Armstrong, Ccaffeine framework: composing and debugging applications iteratively and running them statically, in: Compframe 2005 workshop, Sandia National Laboratories, Atlanta, GA, 2005.

[2] B.A. Allan, R. Armstrong, D.E. Bernholdt, F. Bertrand, K. Chiu, T.L. Dahlgren, K.B. Damevski, W.R. Elwasif, M. Govindaraju, D.S. Katz, J.A. Kohl, M. Krishnan, J.W. Larson, S. Lefantzi, M.J. Lewis, A.D. Malony, L.C. Mcinnes, J. Nieplocha, B. Norris, J. Ray, T.L. Windus, S. Zhou, A component architecture for high-performance scientific computing, Int. J. High-Perform. Comput. Appl. 20 (2006) 163–202.

[3] C.W. Armstrong, R.W. Ford, G.D. Riley, Coupling integrated Earth System Model components with BFG2, Concurr. Comput.: Pract. Exper. 21 (2009) 767–791.

[4] M. Ben Belgacem, B. Chopard, J. Borgdorff, M. Mamonski, K. Rycerz, D. Harezlak, Distributed multiscale computations using the MAPPER framework, Proc. Comput. Sci. 18 (2013) 1106–1115, 2013 International Conference on Computational Science.

[5] M. Ben Belgacem, B. Chopard, A. Parmigiani, Coupling method for building a network of irrigation canals on a distributed computing environment, in: G.C. Sirakoulis, S. Bandini (Eds.), ACRI 2012, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 309–318.

[6] J. Borgdorff, C. Bona-Casas, M. Mamonski, K. Kurowski, T. Piontek, B. Bosak, K. Rycerz, E. Ciepiela, T. Gubała, D. Harezlak, M. Bubak, E. Lorenz, A.G. Hoekstra, A distributed multiscale computation of a tightly coupled model using the multiscale modeling language, Proc. Comput. Sci. 9 (2012) 596–605.

[7] J. Borgdorff, J.L. Falcone, E. Lorenz, C. Bona-Casas, B. Chopard, A.G. Hoekstra, Foundations of distributed multiscale computing: formalization, specification, and analysis, J. Parallel Distrib. Comput. 73 (2013) 465–483.

[8] J. Borgdorff, J.L. Falcone, E. Lorenz, B. Chopard, A.G. Hoekstra, A principled approach to distributed multiscale computing, from formalization to execution, in: Proceedings of the IEEE 7th International Conference on e-Science Workshops, 5–8 December 2011, IEEE Computer Society Press, Stockholm, Sweden, 2011, pp. 97–104.

[9] B. Bosak, J. Komasa, P. Kopta, K. Kurowski, M. Mamoński, T. Piontek, New capabilities in QosCosGrid middleware for advanced job management, advance reservation and co-allocation of computing resources-quantum chemistry application use case, Building a National Distributed e-Infrastructure-PL-Grid, 2012, pp. 40–55.

[10] A. Caiazzo, D.J.W. Evans, J.L. Falcone, J. Hegewald, E. Lorenz, B. Stahl, D. Wang, J. Bernsdorf, B. Chopard, J. Gunn, D.R. Hose, M. Krafczyk, P.V. Lawford, R.H. Smallwood, D. Walker, A.G. Hoekstra, A complex automata approach for in-stent restenosis: two-dimensional multiscale modeling and simulations, J. Comput. Sci. 2 (2011) 9–17.

[11] Center for Advanced Computing Research, Caltech, Pyre: A Python Framework, 2005 http://www.cacr.caltech.edu/projects/pyre/

[12] CERFACS, OpenPALM 4.1.4, 2013 http://www.cerfacs.fr/globc/PALM_WEB/

[13] B. Chopard, J.L. Falcone, A.G. Hoekstra, J. Borgdorff, A framework for multiscale and multiscience modeling and numerical simulations, in: C. Calude, J. Kari, I. Petre, G. Rozenberg (Eds.), LNCS 6714, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 2–8.
[14] J. Dada, P. Mendes, Multi-scale modelling and simulation in systems biology, Integr. Biol. (2011) 86–96.
[15] W.E.B. Engquist, X. Li, W. Ren, E. Vanden-Eijnden, Heterogeneous multiscale methods: a review, Commun. Comput. Phys. 2 (2007) 367–450.
[16] J.L. Falcone, B. Chopard, A.G. Hoekstra, MML: towards a multiscale modeling language, Proc. Comput. Sci. 1 (2010) 819–826.
[17] Y. Frauel, D. Coster, B. Guillerminet, F. Imbeaux, A. Jackson, C. Konz, M. Owsiak, M. Plociennik, B. Scott, P. Strand, Easy use of high performance computers for fusion simulations, Fusion Eng. Des. 87 (2012) 2057–2062.
[18] Globus, GridFTP 5.2, 2012 http://www.globus.org/toolkit/data/gridftp/
[19] 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, P.V. Coveney, Flexible composition and execution of high performance, high fidelity multiscale biomedical simulations, Interface Focus 3 (2013) 20120087.
[20] D. Groen, S. Rieder, P. Grosso, C. de Laat, S. Portegies Zwart, A lightweight communication library for distributed computing, Comput. Sci. Discov. 3 (2010) 015002.
[21] D. Groen, S.J. Zasada, P.V. Coveney, Survey of multiscale and multiphysics applications and communities, IEEE Comput. Sci. Eng (2013), http://dx.doi.org/10.1109/MCSE.2013.47 (preprint).
[22] J. Hegewald, M. Krafczyk, J. Tölke, A.G. Hoekstra, An agent-based coupling platform for complex automata, in: ICCS 2008, LNCS 5102, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 227–233.
[23] A.G. Hoekstra, A. Caiazzo, E. Lorenz, J.L. Falcone, Complex automata: multi-scale modeling with coupled cellular automata, in: A.G. Hoekstra, J. Kroc, P.M.A. Sloot (Eds.), Simulating Complex Systems by Cellular Automata, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 29–57.
[24] A.G. Hoekstra, E. Lorenz, J.L. Falcone, B. Chopard, Toward a complex automata formalism for multiscale modeling, Int. J. Multiscale Comput. Eng. 5 (2007) 491–502.
[25] iMatix Corporation, ZeroMQ: The Intelligent Transport Layer, 2013 http://www.zeromq.org/
[26] G.D. Ingram, I.T. Cameron, Challenges in multiscale modelling and its application to granulation systems, Dev. Chem. Eng. Mineral Process. 12 (2004) 293–308.
[27] G.D. Ingram, I.T. Cameron, K.M. Hangos, Classification and analysis of integrating frameworks in multiscale modelling, Chem. Eng. Sci. 59 (2004) 2171–2187.
[28] D. Kim, J.W. Larson, K. Chiu, Toward malleable model coupling, Proc. Comput. Sci. 4 (2011) 312–321.
[29] J.W. Larson, R.L. Jacob, I. Foster, J. Guo, The model coupling toolkit, in: V.N. Alexandrov, J.J. Dongarra, B.A. Juliano, R.S. Renner, C.J.K. Tan (Eds.), ICCS 2001, LNCS 2073, Springer-Verlag, Berlin, Heidelberg, 2001, pp. 185–194.
[30] P.O. Malaterre, J.P. Baume, Modeling and regulation of irrigation canals: existing applications and ongoing researches, in: IEEE International Conference on Systems, Man, and Cybernetics, IEEE, 1998, pp. 3850–3855.
[31] O. Marcou, B. Chopard, S. El Yacoubi, Modeling of irrigation channels – a comparative study, Int. J. Mod. Phys. C 18 (2007) 739.
[32] MessagePack, Messagepack 0.6.6, 2012 http://msgpack.org
[33] P.M.A. Sloot, A.G. Hoekstra, Multi-scale modelling in computational biomedicine, Brief. Bioinform. 11 (2010) 142–152.
[34] J. Southern, J. Pitt-Francis, J. Whiteley, D. Stokeley, H. Kobashi, R. Nobes, K. Yoshimasa, D. Gavaghan, Multi-scale computational modelling in biology and physiology, Prog. Biophys. Mol. Biol. (2008) 60–89.
[35] S. Succi, The Lattice Boltzmann Equation: For Fluid Dynamics and Beyond, Numerical Mathematics and Scientific Computation, Oxford University Press, Oxford, UK, 2001.
[36] Sun Microsystems, XDR: External Data Representation Standard, RFC 1014, 1987.
[37] M. Swain, T. Hunniford, W. Dubitzky, J. Mandel, N. Palfreyman, Reverse-engineering gene-regulatory networks using evolutionary algorithms and grid computing, J. Clin. Monit. Comput. 19 (2005) 329–337.
[38] A. Yang, W. Marquardt, An ontological conceptualization of multiscale models, Comput. Chem. Eng. (2009) 822–837.
[39] S.J. Zasada, M. Mamonski, D. Groen, J. Borgdorff, I. Saverchenko, T. Piontek, K. Kurowski, P.V. Coveney, Distributed infrastructure for multiscale computing, in: 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), IEEE, 2012, pp. 65–74.

J. Borgdorff received a BSc in Mathematics and in Computing Science (2006) and an MSc in Applied Computing Science (2009) from Utrecht University. He is currently PhD candidate at the Computational Science group of the University of Amsterdam, researching the formal background of multiscale and complex systems modeling and the applied aspects of distributed multiscale computing.

M. Mamonski (1984–2013) received his diploma in Computer Science at the Poznan University of Technology (Laboratory of Computing Systems) in 2008. He started working at the Application Department of the Poznan Supercomputing and Networking Center in 2005. Since then he has contributed to several research EU projects, in particular: GridLab, InteliGrid, BREIN and QosCosGrid, and he was involved in national and European e-infrastructure projects: PL-Grid and MAPPER. His research primarily focussed on web services, queueing systems and parallel execution and programing environments. He was an active member of the Open Grid Forum Distributed Resource Management Application API (OGF DRMAA) working group.

B. Bosak received his M.Sc. degree in computer science from Poznan University of Technology in POLAND (Laboaratory of IT Systems in Management). Since 2007 he has been working at the Application Department of Poznan Supercomputing and Networking Center as a system analyst and developer. His research interests concern Grids, HPC, communication in distributed environments and service integration in SOA. He was participant of a diverse European and national projects, including BREIN (FP6), MAPPER (FP7) and PL-Grid.

K. Kurowski holds the PhD degree in Computer Science and he is leading now Applications Department at Poznan Supercomputing and Networking Center, Poland. He was actively involved in many EU-funded R&D projects in the areas of Information Technology and Grids over the last few years, including GridLab, inteliGrid, HPC-Europa, or QosCosGrid. He was a research visitor at University of Queensland, Argonne National Lab, or CCT Louisiana University. His research activities are focused on the modeling of advanced applications, scheduling and resource management in networked environments. Results of his research efforts have been successfully presented at many international conferences and workshops.

M. Ben Belgacem is a PhD student in computer science at the university of Geneva. His thesis is concerned with the large scale distributed system and computational fluid dynamics. He received his Master degree and engineering degree in computer science from ENSI (Tunisia). Mohamed currently works at hepia (Geneva) and the university of Geneva as a research assistant.

B. Chopard received a PhD in Theoretical Physics from the University of Geneva (1988). He then spent two years in the Laboratory for Computer Science, at the Massachusetts Institute of Technology and one year at the Center for High Performance Computing in the Research Center in Jülich, Germany. He is now professor at the Department of Computer Sciences of the University of Geneva. His research interests concern the modeling and simulation of complex systems on parallel computers. A large part of his work concerns the field of cellular automata, lattice gas and lattice Bolzmann techniques. Numerical simulation of biomedical applications is an important part of his current research activities. He published about 200 papers and is co-author of a textbook on Cellular Automata modeling of Physical systems (Cambridge University Press, 1998).

D. Groen is a post-doctoral research associate in the CCS at University College London, specialised in multi-scale simulation and parallel/distributed computing. He has worked with a wide range of applications, including those using lattice-Boltzmann, N-body and molecular dynamics methods, and participated in several EU-funded IT projects. He finished his PhD in 2010 at the University of Amsterdam, where he investigated the performance of N-body simulations run across geographically distributed (supercomputing) infrastructures. Derek currently works on modelling cerebrovascular blood flow and clay-polymer nanocomposites, using multiscale methods.

Professor P.V. Coveney holds a Chair in Physical Chemistry and is Director of the Centre for Computational Science (CCS), an Honorary Professor in Computer Science and a member of CoMPLEX at UCL. He is also Professor Adjunct within the Medical School at Yale University. He is active in a broad area of interdisciplinary theoretical research including condensed matter physics and chemistry, life and medical sciences including collaborations with clinicians. He is a founding editor of the new Journal of Computational Science and to date has published more than 320 scientific papers and edited 20 books.

A.G. Hoekstra studied Physics holds a Ph.D. in Computational Science from the University of Amsterdam. Currently he is an associate professor in Computational Science at the Institute for Informatics of the Faculty of Science of the University of Amsterdam. His research focuses on applications of mesoscopic models, mostly biomedical, on multiscale modeling and simulation, and efficient mapping of such models to state-of-the-art computing environments. He coordinates the Master's program in Computational Science at the University of Amsterdam. He has organized international conferences in the field of high-performance computing, computational science, mesoscopic and multiscale modeling, and biomedical optics. He has published over 100 peer reviewed research papers, several book chapters, and monographs and books.

Publication [P4]

Alowayyed, S., Piontek, T., Suter, J., Hoenen, O., Groen, D., Luk, O., **Bosak, B.**, Kopta, P., Kurowski, K., Perks, O., Brabazon, K., Jancauskas, V., Coster, D., Coveney, P., and Hoekstra, A.: *Patterns for high performance multiscale computing.* Future Generation Computer Systems, 91:335–346 (2019). https://doi.org/10.1016/j.future.2018.08.045

Ministry points / journal: 140

Contribution of authors:

- Saad A. Alowayyed
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, Multiscale computing patterns and high performance multiscale computing, Design of multiscale computing patterns software, Applications of the multiscale computing patterns software, Discussion and conclusions)
- Alfons G. Hoekstra
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, Applications of the multiscale computing patterns software / Extreme Scaling (Cell based blood flow), Discussion and conclusions)
  - Review and editing of the paper
- Peter Coveney
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, Applications of the multiscale computing patterns software / Replica computing (Binding affinity calculator), Discussion and conclusions)
  - Review and editing of the paper
- Krzysztof Kurowski
  - Coauthorship of the text of publication (Sections: Discussion and conclusions)
  - Review and editing of the paper
- Tomasz Piontek
  - Coauthorship of the text of publication (Section: Design of multiscale computing patterns software / Resource allocation and execution component, Discussion and conclusions)

  - Review and editing of the paper

- Vytautas Jancauskas
  - Coauthorship of the text of publication (Section: Design of multiscale computing patterns software / Resource allocation and execution component)
- **Bartosz Bosak**, Piotr Kopta
  - Coauthorship of the text of publication (Section: Design of multiscale computing patterns software / Resource allocation and execution component)
- Derek Groen
  - Coauthorship of the text of publication (Sections: Design of multiscale computing patterns software, Applications of the multiscale computing patterns software / Replica computing (Binding affinity calculator))
- James L. Suter
  - Coauthorship of the text of publication (Section: Applications of the multiscale computing patterns software / Replica computing (Binding affinity calculator))
- David Coster, Olivier Hoenen, Onnie Luk
  - Coauthorship of the text of publication (Section: Applications of the multiscale computing patterns software / Extreme scaling (Fusion application))
- Oliver Perks, Keeran Brabazon
  - Coauthorship of the text of publication (Sections: Design of multiscale computing patterns software / Optimisation component, Applications of the multiscale computing patterns software)

Future Generation Computer Systems 91 (2019) 335–346

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Patterns for High Performance Multiscale Computing

S. Alowayyed [a,b,*], T. Piontek [c], J.L. Suter [d], O. Hoenen [e], D. Groen [f,d], O. Luk [e], B. Bosak [c], P. Kopta [c], K. Kurowski [c], O. Perks [g], K. Brabazon [g], V. Jancauskas [h], D. Coster [e], P.V. Coveney [d], A.G. Hoekstra [a,i]

[a] Computational Science Lab, Institute for Informatics, Faculty of Science, University of Amsterdam, The Netherlands
[b] King Abdulaziz City for Science and Technology (KACST), Riyadh, Saudi Arabia
[c] Poznań Supercomputing and Networking Center, Poznań, Poland
[d] Centre for Computational Science, University College London, United Kingdom
[e] Max-Planck-Institut für Plasmaphysik, Garching, Germany
[f] Department of Computer Science, Brunel University London, United Kingdom
[g] ARM Ltd., Warwick, United Kingdom
[h] Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften, Garching, Germany
[i] ITMO University, Saint-Petersburg, Russian Federation

HIGHLIGHTS

- We introduce the idea of the Multiscale Computing Patterns (MCP).
- We present the MCP software for High Performance Multiscale Computing.
- To simplify and automate the execution of complex multiscale simulations on HPC.
- Also to provide both application-specific and pattern-specific performance optimisation.
- We test the performance and the resource usage for three multiscale models (two MCPs).
- We demonstrate how the software automates resource selection and load balancing.

ARTICLE INFO

Article history:
Received 9 January 2018
Received in revised form 18 July 2018
Accepted 27 August 2018
Available online 10 September 2018

Keywords:
Multiscale computing
High performance computing
Modelling methodology
Distributed computing
Model coupling

ABSTRACT

We describe our Multiscale Computing Patterns software for High Performance Multiscale Computing. Following a short review of Multiscale Computing Patterns, this paper introduces the Multiscale Computing Patterns Software, which consists of description, optimisation and execution components. First, the description component translates the task graph, representing a multiscale simulation, to a particular type of multiscale computing pattern. Second, the optimisation component selects and applies algorithms to find the most suitable mapping between submodels and available HPC resources. Third, the execution component which a middleware layer maps submodels to the number and type of physical resources based on the suggestions emanating from the optimisation part together with infrastructure-specific metrics such as queueing time and resource availability. The main purpose of the Multiscale Computing Patterns software is to leverage the Multiscale Computing Patterns to simplify and automate the execution of complex multiscale simulations on high performance computers, and to provide both application-specific and pattern-specific performance optimisation. We test the performance and the resource usage for three multiscale models, which are expressed in terms of two Multiscale Computing Patterns. In doing so, we demonstrate how the software automates resource selection and load balancing, and delivers performance benefits from both the end-user and the HPC system level perspectives.

1. Introduction

Multiscale modelling & simulation has become a well-established way to study complex phenomena that encompass multiple space and time scales [1]. In this approach, a multiscale model is constructed by combining, or *coupling*, a collection of single-scale submodels, each of which captures processes on a

* Corresponding author at: Computational Science Lab, Institute for Informatics, Faculty of Science, University of Amsterdam, The Netherlands.
E-mail addresses: S.A.Alowayyed@uva.nl (S. Alowayyed), A.G.Hoekstra@uva.nl (A.G. Hoekstra).

https://doi.org/10.1016/j.future.2018.08.045

distinct space and time scale; see e.g. [2–4]. Multiscale modelling is widely used in most areas of science and engineering [5], such as biomedicine [6–8], fusion [9,10], material science [10,11], energy [12] and engineering [10,13]. It is self-evident that any high-fidelity multiscale model must employ substantial high performance computing resources, since the individual single scale models comprising it themselves have to run on such machines.

In addition to specific multiscale applications, a number of tools and frameworks which assist in multiscale computing have been established. These range from domain-specific frameworks such as AMUSE [14] and OASIS-MCT [15] to solver-specific frameworks such as the MOOSE framework for finite-element codes [16] and fully generic frameworks [1,3,17–19] encompassing related coupling tools such as the Multiscale Coupling Library and Environment 2 (MUSCLE2) [20].

We have previously developed the Multiscale Modelling and Simulation Framework (MMSF) [3,17–19], which provides a theoretical and methodological framework for constructing multiscale simulations in four main stages. First, we model multiscale phenomena as collections of single-scale submodels then decide on which models interact with each other and how. Single scale models and couplings are presented within a Scale Separation Map, allowing us to describe and compare multiscale models on a conceptual level. Second, we specify the single scale models, their couplings and interactions using the Multiscale Modelling Language [1,18]. Third, we convert these definition to a fully implemented multiscale model, currently relying on MUSCLE2 [20] (although the concepts of the MMSF can also be applied to other coupling environments such as AMUSE [14]). An important property of MUSCLE2 is the separation of concerns that it affords. Submodels are not aware of any other components. Moreover, required adaptations are minimal on the level of a submodel in order for it to be incorporated into a multiscale model implemented with MUSCLE2. Fourth, we deploy and execute the multiscale application on a set of computational resources. Developers and users can run different submodels on different machines [4], using for example the QCG middleware [21], a paradigm that we call Distributed Multiscale Computing [4].

Knap et al. [22] have previously proposed a distributed multiscale computing framework that supports the on-demand execution of microscale models coupled to a macroscale model (very similar to one of the computing patterns we proposed in [23]), using large scale supercomputing resources. Although their framework has, to our knowledge, not yet been applied outside the domain of materials science for which it was originally created, the authors do propose a general conceptual framework that could be adopted for use in other disciplines. This resonates with our vision of generic multiscale computing environments, where a separation of concerns is achieved between multiscale modelling & simulation on the one hand, and deploying and executing a multiscale simulation in a given HPC environment on the other.

Our "Lego-based" philosophy for the construction and execution of multiscale applications relies on single scale submodels and their interactions, and results in more degrees of freedom for both programming and executing a multiscale simulation. To efficiently execute multiscale applications on high-end HPC machines, a number of challenges have to be addressed, such as load balance (providing resources to each of the single scale models), fault tolerance (sometimes instantiations of single scale models may fail) and energy awareness (depending on properties of single scale models, potentially also in combination with load balancing, energy aware optimisation). Our intention is that these challenges are handled in a generic way, as far as possible avoiding the imposition of that burden on the developers of multiscale applications. Those developers should take care of the scale bridging mechanisms and the efficiency of the single scale models, while the challenges of execution within a High Performance Computing (HPC) environment should be addressed through a generic layer added to MMSF that we call *Multiscale Computing Patterns* (MCPs) [23].

We defined MCPs as "high-level call sequences that exploit the functional decomposition of multiscale models in terms of single scale models" [23], and distinguished three patterns: Extreme Scaling, Heterogeneous Multiscale Computing and Replica Computing. Each of these patterns is described using a generic task graph that aids in understanding how to best map these patterns to HPC resources. In addition to the generic task graph, an MCP contains performance information about single scale models, an XML-based specification of the multiscale application named xMML [20], and a set of algorithms and heuristics used to combine this into input files for the execution environment. In this paper, we report on the design and implementation of the MCP software, and present the first results of executing multiscale simulations using MCPs, including discussions on the added value of using such solutions for High Performance Multiscale Computing. Here, we mainly integrate these MCPs with MMSF to increase the effectiveness by means of which we can develop, deploy and execute multiscale simulations on existing petascale and emerging exascale resources [23].

The MCP software architecture consists of a *description component*, an *optimisation component* and an *execution component*. In the description component the software uses the task graph of the specific multiscale model, in combination with auxiliary information (e.g., definitions of single-scale models), to identify the type of pattern and create input definitions for the optimisation component. In the optimisation component, the software selects and applies a set of optimisation algorithms to identify a range of efficient mappings of the submodels in the application to specific HPC resources. Lastly, the execution component is a middleware layer which identifies the optimal mapping of submodels to the available resources, taking additionally into account queueing times and resource occupancy. Moreover, the execution component deploys and executes the application, with all of its submodels on the target resources. Three examples of using Multiscale Computing Patterns software are illustrated and examples of cost functions are worked out, showing that a wide range of variables for Multi-Objective Optimisation algorithms can be chosen. The idea is that Multiscale Computing Patterns software will automatically detect which cost functions and algorithms to select based on the type of pattern and user requirements.

The structure of our paper is as follows. We describe the MCPs in Section 2, and introduce the Multiscale Computing Patterns software and its components in Section 3. In Section 4, we provide by way of proof of concept three examples of the use of the Multiscale Computing Patterns software. Finally, we provide a discussion and conclusion in Section 5.

2. Multiscale computing patterns and high performance multiscale computing

In this section, we discuss the concept of Multiscale Computing Patterns and express the MCPs as generic task graphs. For full details, we refer to Alowayyed et al. [23]. Fig. 1 shows the generic task graphs for all three computing patterns.

The Extreme Scaling (ES) pattern represents a type of multiscale model where one *primary* single-scale model is coupled to a set of serial and/or parallel *auxiliary* models on any scale as shown in Fig. 1((a) and (b)). The primary model[1] is compute intensive, energy hungry, and highly scalable, whereas the auxiliary models are not. Therefore, the efficiency of this type of multiscale models is highly dependent on the efficiency of the primary model and

[1] We assume one primary model here. In practice, ES could consist of more than one primary model.

the primary–auxiliary interactions. Assuming that developers have implemented the primary model efficiently, the main aim is to reach a minimal interference between primary and auxiliaries. This can be done using load balancing while ensuring minimal communication between primary and auxiliaries. The serial auxiliary model can give rise to strong underutilisation of available resources (e.g. if it does not scale to a large number of cores), and special mechanisms to handle such situation are required.

The Heterogeneous Multiscale Computing (HMC) pattern (Fig. 1(c)) represents the typical form of macro-micro coupling, where the numerical solver at the macro-scale level requires input from multiple micro scale model instantiations (for instance to compute a spatially varying quantity, such as for example a constitutive equation, say a viscosity in a flow problem). Thus, the number of micro-scale models is dynamic and largely dependent on the dynamic evolution of the macro-model. The HMC manager has some control over the number of micro-scale models, by preventing redundant calculations (by storing results of previous microscale simulations in a HMC database and where possible extracting results from the database, e.g. by interpolations between results obtained earlier), and spawning extra micro-scale models, when necessary. Typically, the number of micro-scale models will be very large and a single microscale run may require substantial computing resources and, hence, dominate computing and energy cost.

In the Replica Computing (RC) pattern a large number of copies of tera- and/or peta-scale simulations are needed to produce statistically robust results. These replicas are not part of an overarching structure like HMC, but are spawned in the initial step. In this step, the parameter space for parameter sweeping is set. Then, simulations and data processing per replica take place. Both static and dynamic flavours of RC are considered in Fig. 1((d), (e)). All replicas execute independently of each other. If a replica (i.e. a simulation) fails, the RC patterns affords some level of fault tolerance, taking into account maintaining the overall statistics. This is the main difference with HMC. On the other hand, HMC and RC are similar in terms of load balance issues.

3. Design of multiscale computing patterns software

The Multiscale Computing Patterns software consists of three parts: (1) the Description Part, where the user describes the multiscale application, (2) the Optimisation Part, where the software predicts and optimises the application performance, (3) and the Execution Part, where the application is deployed using an underlying resource allocation service (in our case the QCG middleware). We present the components of the Multiscale Computing Patterns software, and their interrelations, in Fig. 2.

The logical description and the complete set of requirements of a multiscale application is collected in the description component. This part relies on concepts from the Multiscale Modelling and Simulation Framework. It is helpful to facilitate the work of the end user and provide a single input mechanism for all multiscale applications, as well as detecting the type of MCP. The MMSF xMML description file was extended for Replica Computing to accommodate the notation of the number of replicas. The optimisation component determines which MCP optimisation applies, collects required performance figures, and calculates the relevant metrics (e.g. parallel efficiency, throughput, energy usage). Based on these results, a constrained optimisation is performed resulting in a small set of the most suitable execution scenarios which are passed to the execution component. The role of the execution component is to select the best allocation plan, based on the availability of the requested resources and cost criteria (time to complete, energy usage), and to start and monitor the execution.

3.1. Description component

The Description component (top layer in Fig. 2) contains an architecture-agnostic definition of the multiscale application, and its main requirements. It builds on concepts from the Multiscale Modelling and Simulation Framework. The description component consists of a task graph, submodel definitions, simulation and middleware parameters and user information, all feeding into the Translation Service.

The task graph is expressed in the form of a highly adaptable textual description (xMML, see [20]) which is used to detect motifs (repetitive submodels and dependencies). The task graph is also needed to observe workflow related issues such as the expected frequency of communication between submodels.

Submodel definitions contain all the information required for a single-scale model to run. This includes information on submodel-specific dependencies, and the resource requirements for each submodel (e.g., mandatory use of GPU-architectures, or a minimal memory requirement per core). The description component may rely on previously developed tools such as MAD/MaMe [24], and can already leverage existing configuration information from the FabSim automation environment [25].

All the simulation and middleware parameters are collected in a separate component. This includes input parameters, the required environment modules and resource limits for the overall simulation (all submodels and coupling library). Also, this component holds all information needed to compose the multiscale simulation (e.g. using MUSCLE2) and to execute the simulation (e.g. using QCG Middleware [26]) using (distributed) HPC infrastructures such as the Experiment Execution Environment (EEE). This component is designed such that existing known machine configurations from FabSim (machines.yml) can be directly reused in the context. Similarly, user-specific information can be directly reused from existing FabSim configurations (machines_user.yml). We present an example of the reuse of FabSim information within this context as part of the Binding Affinity application described in Section 4.2.

These three pieces of information are then supplied to a translation service, which merges and converts them into a format suitable for the optimisation component. Currently, the translation tool is application specific, and produces two xml files as output. One file, matrix.xml, is shown in Listing 1 in Appendix A and contains templated information from the submodel definitions. The other file, multiscale.xml shown in Listing 2 (Appendix A), has information from the simulation and middleware parameters.

3.2. Optimisation component

The main software tool within the Optimisation component is the Pattern-Driven Planner. This tool requires input from both the Translation service as well as the Node Description List. The node description list is updated regularly to reflect the current status of available nodes in the targeted supercomputers, and contains information of node types. A single node type represents a set of nearly identical nodes in terms of hardware configuration (e.g. processor type). The node types should be defined based on knowledge gathered a priori by the middleware from the infrastructure provider. Table 1 shows an example of node types.

The second layer contains the Pattern-Driven Planner and the Performance Estimator components. The Pattern-Driven Planner component collects measurements and/or predictions of performance of submodels, under various execution scenarios, from the Performance Estimator. Then, it uses this information to compute required cost functions (e.g efficiency, throughput, energy usage, or a combination) on available resources. Given the specific MCP and all other available information, the Pattern-Driven Planner performs a constrained optimisation against these cost functions,

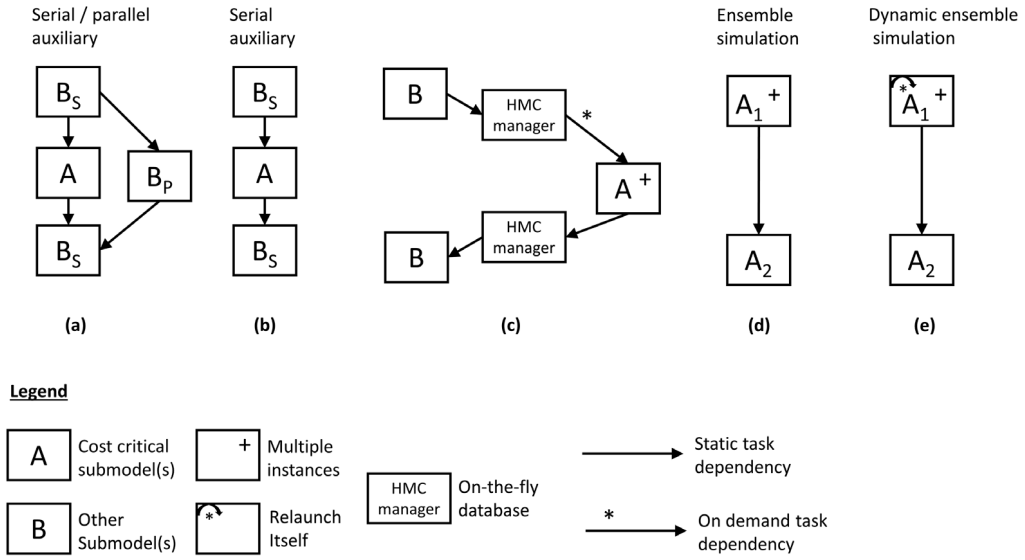


Fig. 1. Generic task graphs for the Extreme Scaling computing pattern (a,b), the Heterogeneous Multiscale Computing pattern (c) and the Replica Computing pattern (d,e). (a) shows the case where the auxiliaries B_P are running in parallel with the primary model A, while in both (a) and (b) auxiliaries B_S are in series with the primary model. In (c) multiple instances of the cost critical micro submodel (A) are called on-demand by the macro submodel (B). The macro-scale solver requires input from micro-scale solvers at every time step. (d) shows the case where multiple instances of submodels interact in phases, while in (e) the same operation is shown with addition to a self-relaunch mechanism.

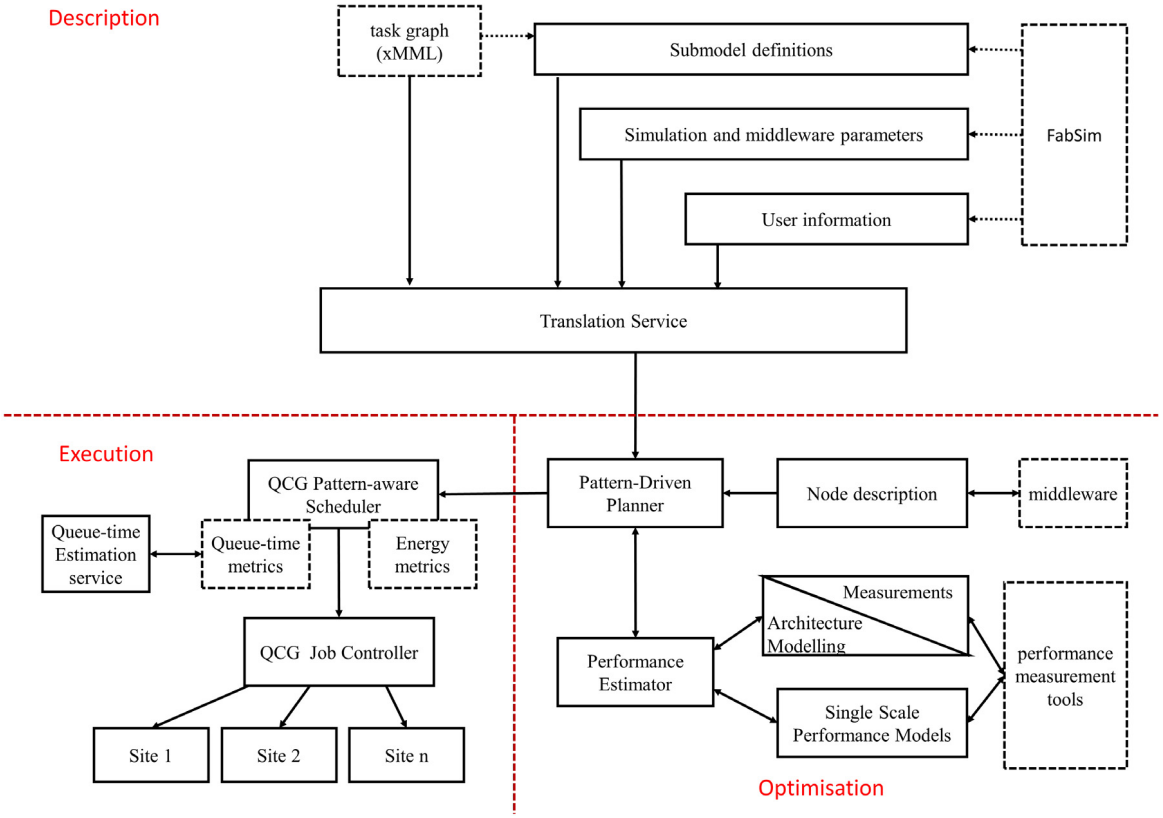


Fig. 2. Architecture of the Multiscale Computing Patterns software. The dashed-line boxes represent external components (which either exist separately or are under development).

and provides a selection of particularly suitable execution scenarios to the Execution component. The Execution component will then select the optimal execution plan based on chosen specified cost criteria (time to completion, energy consumption), by taking into account additional information only available to the middleware (e.g. estimated queueing time, live information on availability of resources, etc.).

The Measurements and Architecture Modelling components respectively store and calculate submodel performance information as a function of the chosen number of cores/nodes and problem size. The Single Scale Performance Model captures the scalability of submodels with respect to problem sizes and number of processors. The Performance Estimator, in turn, relies among others on the Single Scale Performance Model to obtain, interpolate and/or

Table 1
Example of node types, an input to the Pattern-Driven Planner. Note that RAM per node is in Giga bytes. Processor type **1** Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60 GHz, **2** Intel(R) Xeon(R) CPU E5-2680 @ 2.70 GHz, **3** Intel(R) Xeon(R) CPU E7-4870 @ 2.40 GHz and **4** Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz.

Type name		# of nodes	Processors per node	Cores per node	RAM per node	Processor type
Host	Type					
Eagle	haswell_64	492	2	28	64	**(1)**
	haswell_128	460			128	
	haswell_256	52			256	
Supermuc	Thin	9216	2	16	32	**(2)**
	Fat	205	4	40	256	**(3)**
Stfc	Default	118	2	16	64	**(4)**

calculate performance results for the multiscale model. For example, this could be achieved by interpolating between performance results of adjacent problem sizes in the multiscale model and/or, relying on performance models, to predict the performance using a core count for which no measured values have yet been obtained.

In the Measurements component, the overall cost of the submodel as a function of problem size is measured for 1 to n cores on the first node in a specific node type, and for 2 to N nodes assuming full occupancy on each node for a given number of iterations. In the baseline case, the cost is represented as execution time, but note that these calculations can also be done for other metrics of cost such as energy. The actual measurements can be obtained from tools designed specifically for performance profiling tools, such as Allinea MAP [27], the tool of choice in our research. A template of measurements is shown in Listing 3 in Appendix A; this measurement listing might contain specially marked values (i.e NA), for node types where a specific single scale model is not supported.

The Architecture Modelling software is established to provide predictions for existing machines, but also for non-existing emerging exascale configurations. This allows users to assess how MCPs could optimally benefit from such hypothetical machines, or contribute in co-evolution of such new architectures.

Based on performance results from the Performance Estimator, the Pattern-Driven Planner groups types of nodes into classes depending on the similarity of performance figures, type of computing pattern and cost criteria (e.g efficiency, makespan time, energy usage, resource usage, ...) computed as cost function. Then, using multi-objective optimisation, the tool will generate a small number of alternative execution scenarios. The importance of the alternatives here is to give the Execution component the freedom to choose from a set of resources with comparable performance per submodel depending on the availability of these resources as well as on the variation in queue times. We will enhance this component to extend the patterns with capabilities to also consider issues related to energy awareness and fault-tolerance. All in all, for each pattern we will formulate constrained optimisation problems that as output will deliver alternative execution profiles to the Execution component.

The exact output of the Pattern-Driven Planner to the Execution component will be several allocation plans and other requirements, as described in the next section, to run multiscale application. Here, the output file holds information about the kernels and corresponding helpers, the classes of node types and a set of allocation plan. The allocation plan is a specific mapping of the multiscale model to resources. Listing 4 in Appendix A shows the template of the node classes and allocation plans parts.

3.3. Resource allocation and execution component

The responsibility for the execution component is twofold. First, it needs to select the best allocation plan from the plans provided by the optimisation component. The selection pertains to the mapping of computational kernels to a specific set of physical resources of defined types, taking into account the (sometimes conflicting) requirements of users and resource providers. Second, once an optimal plan has been selected, this component needs to ensure the efficient and reliable execution of the application within the distributed heterogeneous infrastructure.

The execution component is mainly provisioned using the QCG environment[2] [24], a mature middleware system deployed in many HPC centres across Europe. QCG delivers a set of ready to use components that can be installed and managed at each site, irrespective of the internal policies or local queueing systems. To fulfil expectations and objectives of both users and resource providers, QCG features and extendable brokering service which allows for customised brokering algorithms and strategies. In addition, QCG provides support for advance reservation, co-allocation and workflows, enabling the execution management of multi-kernel applications, with both cyclic and acyclic dependencies, on a distributed e-Infrastructure [20,21].

Deploying multiscale simulations on production e-infrastructures gives rise to a number of challenges that are difficult to anticipate prior to the execution component. For example, the user objective for an immediate job start, e.g. through means of advance reservation, needs to be harmonised with the provider's objective for high resource utilisation. In addition, the Pattern Driven Planner provides plans that are likely to be optimal from a user perspective, but have not yet incorporated the constraints imposed by the presence of other workloads in the e-infrastructure environment.

The *QCG Pattern-aware Scheduler* (which is part of QCG Broker) calculates which of the plans provided by the Pattern Driven Planner is optimal with respect to the objectives of all involved stakeholders. In doing so it takes into account the current and historical load on e-infrastructure resources, including both the occupancy of the actual resources and the queue lengths. The Pattern-aware Scheduler can perform this optimisation with respect to required cost criteria, either a single time to completion criterion or a combination of two criteria, total energy expenditure and time to completion. Here, the time to completion is calculated by adding the predicted queueing time (predicted by Queue Time Prediction Service to QCG) and execution time (provided by the optimisation component). In the energy optimisation case, QCG Scheduler firstly selects a set of candidate plans which finish according to the QCG time to completion prediction in the requested period of time and then it selects an optimal plan with the minimal total energy expenditure (calculated and given by the optimisation component). Through its direct integration with the QCG environment, the Pattern-aware Scheduler accounts for the multi-kernel nature of multiscale application and the fact that each kernel may behave differently in the context of performance and energy-usage when executed on different resource types [26,28,29].

The QCG Pattern-aware Scheduler relies on a plugin-like architecture to gather all required information (see dashed boxes in the Execution component of Fig. 2). For example, the scheduler uses the *Queue time metrics plug-in* to get precise knowledge about the expected queue time on available resources. This plug-in is integrated with external resource-level components, in this case the *Queue-time Estimation service*. Similarly, we are planning to implement an Energy metrics plug-in and combine it with the QCG Pattern-aware scheduler.

As the new brokering module uses new types of input parameters to specify the requirements of the MCPs, we have extended the job description interface and revised several internal schemas used to exchange information between the components in QCG-Broker service. We present several key fragments of the this extended

[2] www.qoscosgrid.org.

Table 2
Resources used for the measurements in Sections 4.1 and 4.2.

Resource	CPU architecture	Cores
SuperMuc	Intel(R) Xeon(R) CPU E5–2680	147 456
	Intel(R) Xeon(R) CPU E7–4870	8 200
Eagle	Intel(R) Xeon(R) CPU E5-2697 v3	2 408

description in Appendix F. Here, all jobs described using a *patternTopology* element will be processed using the new scheduling engine.

Based on the result of the QCG Pattern-aware Scheduler, the QCG Job Controller module prepares the execution environment by transferring input data and starting the job submission to one or more distributed resources. The resources in our e-infrastructure are made accessible to QCG Job Controller using services implementing the Basic Execution Service (BES) interface [30].

QCG Job Controller contains a set of specific adaptations to address the requirements for efficiently executing high performance multiscale simulations using high-end e-Infrastructures. Both the QCG-Broker interface and the core capabilities of the service components have been extended to support a range of pattern-based multiscale jobs. Specifically, to allow efficient execution of the Replica Computing Pattern applications, we have incorporated two additional QCG mechanisms: workflows and job arrays. We incorporated a modified version of existing workflow mechanisms [31], eliminating the need to transfer data between subsequent tasks executed on the same resource, and simplifying the execution of workflows in parameter sweep tasks. The job arrays functionality allows a set of independent tasks to be run on a resource and be considered as a single QCG task. In the Replica Computing Pattern these sets of subtasks can be scheduled by the middleware to be executed on various clusters, thereby balancing the overall load on the infrastructure. Job arrays not only help to reduce the management complexity of all tasks executed separately, but increase the overall throughput of the system and decreases the total time to completion of a simulation.

4. Applications of the multiscale computing patterns software

In this section, we present three exemplar applications from different scientific domains (one from Fusion research and two from biomedicine, being cell based blood flow modelling and the Binding Affinity Calculator BAC) to demonstrate the capabilities practical usage of the MCP software, and the benefits in terms of application performance. Our applications are mapped to two different computing patterns, with Fusion and cell based blood flow modelling mapped to the Extreme Scaling (ES) pattern and BAC to the Replica Computing (RC) pattern. Applications for HMC are currently under development. In addition, details of the required steps and various code snippets at each level of the software stack are presented from the perspective of the application developer. All performance figures presented are measured at two supercomputers that participated in the studies, namely SuperMUC [32] (Tier-0 HPC from Leibniz-Rechenzentrum, Germany), and Eagle [33] (Polish national grid clusters from Poznan Supercomputing and Network Center, Poland). Further details are listed in Table 2.

4.1. Extreme scaling

In ES, the ultimate goal is to ensure minimal interference between the primary model and the auxiliaries. It may happen (as in the example of the cell based blood flow simulation) that the auxiliary models induce large waiting times for the primary model, thus potentially wasting resources and reducing resource usage. The Multiscale Computing Patterns software detects this situation automatically, and then interleaves two multiscale simulations, executing both at the same time [23]. This mechanism would increase the resource usage efficiency. For ES, the efficiency of the multiscale model ϵ_M can be calculated as [23]:

$$\epsilon_M = \frac{\epsilon_P}{\frac{T_{aux}(P)}{T_P r(P)} + 1}, \tag{1}$$

and the resource usage efficiency (R) as:

$$R = \frac{\sum_i T_i P_i}{T \sum_i P_i}. \tag{2}$$

where P_i is the number of cores used for submodel i, T_i is the execution time on submodel i excluding waiting times, T is the total execution time including waiting times, and ϵ_P the efficiency of the primary model.

Fusion application

Nuclear fusion has the potential to produce clean and carbon-free energy, as physicists hope to demonstrate with ITER, which is a tokamak device that uses magnetic fields to confine plasma. However, the grand challenge of long-term plasma confinement requires the understanding of interactions between very small-scale turbulence and large scale plasma behaviour [9,34]. Therefore, having a robust multiscale computing scheme to study this interaction has become a vital goal in the fusion community. Our targeted fusion application simulates the time evolution of a plasma's 1D profiles (for instance electron temperature) in the tokamak core with a transport code, while under the influence of anomalous transport coefficients computed by a 3D turbulence code and periodic 2D equilibrium reconstruction [34]. The transport, turbulence, and equilibrium codes are submodels developed separately and are well-benchmarked. These submodels share a common data interface and are embedded into MUSCLE2 as kernels, which allows for straightforward coupling through a simple and configurable script as described in [9]. Such simulation is essentially multiscale in time, and corresponds to the ES computing pattern depicted in Fig. 1(b). The turbulence code is the primary submodel in this application because it requires the vast majority of the computational power compared to the other submodels.

The starting point in the description component of the software is to compose a task graph in xMML format. This text file (shown in Appendix B) contains the list of submodels involved, time and space scales and input/output data of each submodel, and coupling between submodel pairs through their respective input/output data. If desired, the user can deploy the jMML tool [20] to generate the task graph from the xMML [2] (displayed in Appendix D). Besides visual representations, the jMML tool can turn the content from a task graph into a skeleton configuration for MUSCLE2. The designer of the coupled application can implement submodels as MUSCLE2 kernels and other simulation parameters (either global or specific to a kernel) into the MUSCLE2 configuration file [20]. An example of the fusion application's configuration file, which is written as a ruby script, is displayed in Appendix C. Note that at this stage, the user can directly connect to a cluster where all executables, libraries and input data are present, and write an ad-hoc script to be submitted to the local batch queue system. However, the burden of manually adjusting the configuration and selecting the optimal cluster lies on the user every time wants to run a simulation. The MCP software has features that relieve these burdens from the user by automatically selecting the best configuration for a given performance metric, as described in further detail in the remainder of this subsection.

The task graph is submitted and parsed by the Translation service along with other specifics provided by the developer, such

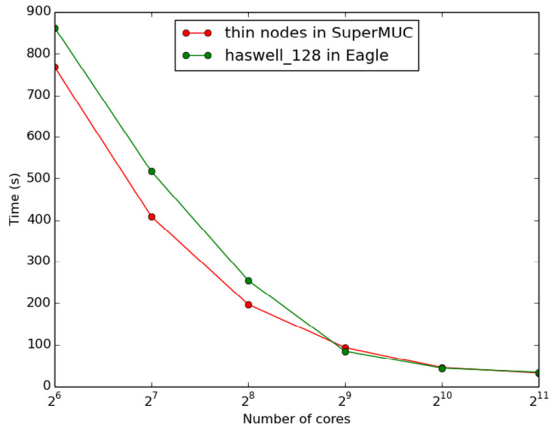


Fig. 3. Runtime on different resources for one iteration of the primary submodel in the fusion application.

Table 3
Performance for ES applications, namely Fusion and cell based blood flow modelling (RBC). T is the execution time (excluding waiting times) for primary (Pr) and auxiliaries (aux) on P_{Pr} and P_{aux} number of cores in seconds, ϵ is the efficiency for the primary (Pr) and the multiscale model (M) and R is the resource usage.

Simulation	Host	P_{Pr}	T_{Pr} (S)	P_{aux}	T_{aux} (S)	ϵ_{Pr}	ϵ_M	R
Fusion	SuperMuc	1024	49017	1	780	1.0618	0.9774	0.91
RBC	Eagle	1036	1936.7	168	1531.6	0.7066	0.3946	0.367
RBC_{alt}	Eagle	1036	3081.34	168	3081.34	0.7066	0.3954	0.746

as additional submodel definitions, details on middleware, simulation parameters, and user information. In the current implementation, the Translation service is a python script which, as a result, creates two template xml files: matrix.xml and multiscale.xml. Matrix.xml contains information related to single scale submodels, such as measurements of their performance. An example of benchmark data on scaling of the primary submodel for two types of nodes is shown in Fig. 3. Multiscale.xml contains information related to the coupled application. These two templates are prefilled with information from the xMML file and can be completed by the application designer. An example is given in Appendix E.

Next, the outputs of the Translation service (matrix.xml and multiscale.xml) are passed on to the Pattern-Driven Planner, which in turn generates an XML job script for the selected middleware (the QCG). Currently, the Pattern-Driven Planner proposes three optimal plans that minimise the cost, and an example of such plans is shown in Appendix F. Currently, these plans are drafted based on the measurements of runs performed manually by the application designer. The next stage will be to enhance the Performance Estimator such that it can benchmark on-the-fly and interpolate on settings for which no performance data is available.

Finally, the job script from the Pattern-Driven Planner is submitted to the QCG. QCG selects one of three plans and starts the simulation on the system(s) involved. For the fusion application, and in general for most ES applications, it is more sensible to select a plan in which all submodels run on a single site, for auxiliaries do not require much resources. In that case, we should only care about serialisation due to serial auxiliary models and how that could impact the execution [23]. Also note that the speedup of the primary model is super-linear because the efficiency was calculated with 64 cores instead of one core, which may lead to latency hiding. For fusion, the time of the primary model spent in waiting for the auxiliaries is not that large, as shown in Table 3, so no additional actions are required, and, therefore, the resource usage is high.

In particular, QCG selects the thin nodes in SuperMUC to run the fusion simulation (see Table 3). A production run with 1000 iterations using 2048 cores was completed successfully using the software scheme described earlier. The entire run was completed in approximately 11.1 h, or 22733 core hours. Among the three submodels, the primary submodel (a turbulence code based on gyrofluid theory) took about 17 s per iteration, while the transport and equilibrium auxiliary submodels took 1 and 3 s, respectively. However, the fusion plasma in this particular example needs approximately 4000 iterations to reach equilibrium state. Therefore, improving efficiency becomes essential as future simulations require more computing time. The current simulation couples the submodels in series. One way to speed-up the simulation is to run auxiliary submodels in parallel when possible, which is theoretically the case for the timescale-less equilibrium submodel. This idea is preliminary and its validation is necessary before such transformation is added as a possible optimisation technique in the Pattern-Driven Planner.

The ultimate goal for the fusion application is to use a more sophisticated turbulence model, namely replacing the gyrofluid model with a gyrokinetic model, to simulate plasma in the core of a tokamak. In addition, the ability to simulate plasma of a much larger volume would be necessary to understand possible instabilities that could destroy plasma confinement in the ITER tokamak. These goals require an extensive amount of computing resources, as well as intelligent and highly optimised coupling approaches. The Multiscale Computing Patterns software have demonstrated initial success with a smaller-scale problem. With further improvements, we envision that these patterns can efficiently handle future exascale calculations involving ITER and gyrokinetic simulations.

Cell based blood flow simulation

In this application we couple continuous blood flow simulations implemented in Palabos [35] (a fully parallelised open source Lattice Boltzmann Model) to cell based blood flow simulations implemented in the Hemocell suspension simulation framework (an Immersed Boundary Lattice Boltzmann Model (IB-LBM)) [36–39]. Specifically, we couple two continuous fluid fields (C_L and C_R, which are serial auxiliary models) to the inlet and outlet of HemoCell, in order to provide the correct in- and outflow conditions to the more expensive suspension simulation (P, the primary model in this application), and to keep the domain for the cell based blood flow simulation as small as possible. This application has also been coupled using MUSCLE2. The performance measurements are shown in Fig. 4. As is clear, in this case the auxiliary models (C_L and C_R) require a small amount of computing and only execute on a small core count. However, the primary model, HemoCell, is compute hungry, but at the same time scales very well to a much larger core count, even so that the execution time of the primary becomes comparable to the execution time of the auxiliary submodels. This situation was analysed in [23] and calls for a more advanced scheduling of the pattern, basically interleaving two instantiations in order to make best use of the available computing resources. The MCP software is able to orchestrate such more advanced scheduling of multiscale applications.

Table 3 shows that the resource usage for running this application is 0.35. This is due to the large waiting times of both primary and serial auxiliaries in the naive scheduling, which means wasting 1122 cores hours (1.08 h per core) for primary and 1755 (12.5 h per core) for auxiliary models by doing nothing but waiting. To solve this, we interleave two different instantiations with each other, as proposed in [23]. This mechanism was coordinated using wait/notify semantics [10]. By doing so, we doubled the resource usage efficiency by reducing the wasted cores to 887 and 152 core hours for primary and auxiliaries models respectively. By implementing more advanced load balancing algorithms and selecting the right number of cores for the primary and auxiliary models, we can increase the resource usage efficiency even more. We are currently realising such more advanced features of the MCP software.

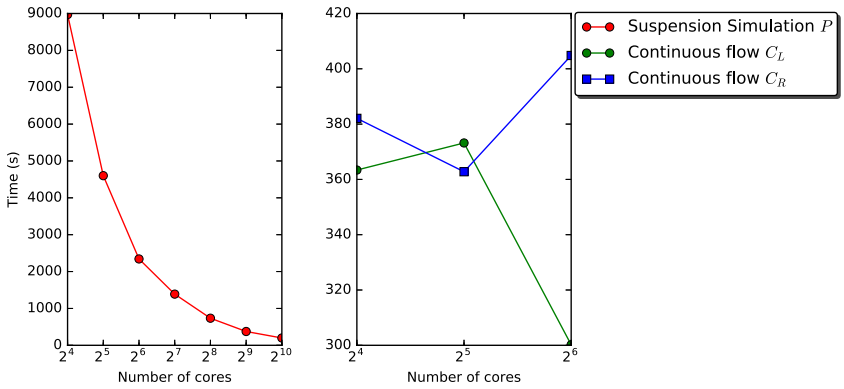


Fig. 4. Total runtime of cell based blood flow modelling submodels on Eagle [33] *haswell*_128 nodes. Note the difference in scale between the primary suspension model (left) and the auxiliaries continuous flow models (right).

4.2. Replica computing (binding affinity calculator)

The procedure for replica computing is similar to that for Extreme Scaling (Section 4.1). The starting point for all RC pattern applications is the task graph, via an xMML textual description. A "multiplicity" tag in the "instance" node of the xMML description indicates that multiple instances (replicas) are required for that submodel. The Translation service detects the presence of this tag, identifies that the RC pattern is required and that the associated cost function in the Multiscale Computing Patterns software should be invoked.

The Translation service uses submodel definitions in separate files. To illustrate this, we describe the process for the Binding Affinity Calculator (BAC) [40], an automated molecular simulation based free energy calculation workflow tool, which we use to calculate ligand-protein binding affinities. Rapid and accurate calculation of binding free energies is of major concern in drug discovery and personalised medicine. The underlying computational method is based on classical molecular dynamics (MD). These MD simulations are coupled to the molecular mechanics Poisson–Boltzmann surface area (MMPBSA) method to calculate the binding free energies. For purposes of reliability, ensembles of replica MD calculations are performed for each method, and we have found that about 25 of these are required per MD simulation in order to guarantee reproducibility of predictions. This is due to the intrinsic sensitivity of MD to the initial conditions, since the dynamics is chaotic. Therefore, BAC is an ideal example of the replica computing pattern. BAC consists of a workflow where, within each replica, the output from one submodel (NAMD) is used as input to the next submodel (AmberTools). For more information, we refer to [40,41].

BAC previously used the FabSim [25] tool extensively to perform simulation runs and, therefore, we have added an option to the Translation service to allow the matrix.xml and multiscale.xml files to be completed (as much as possible) through reading of FabSim configuration files. This demonstrates the potential versatility of our MCP approach, which should enable relatively straightforward integration with existing multiscale execution environment as, in this case, FabSim. For example, it uses the machines.yml configuration file from FabSim, which lists the configuration settings of submodels on remote resources (e.g., location of libraries and required execution flags). Additional information specific to for the Translation service (and not required by FabSim) can also be added to this file, including restrictions on the submodel (GPU/CPU compatibility, max/min number of cores, etc.). This allows submodel information to be reused if it is required for different multiscale applications. Then, FabSim compatible YaML file (shown in Appendix G) are used to assist the completion of matrix.xml and multiscale.xml. BAC currently does not use a coupling library (such as MUSCLE), so no additional files are required. However, in the future we foresee hybrid MCPs, where each replica could for instance be a full-blown ES by itself, and then such additional information would be needed.

Following the procedure outlined for the ES pattern, the user passes matrix.xml and multiscale.xml to the Optimisation component. Unlike the ES pattern, the user does not need to specify the required number of cores for the overall simulation. This is decided by the Performance Estimator by calculating the cost function.

Finding a cost function for RC that will generate resource allocation plans is different to that for ES. First, there is an obvious trade-off between the number of replicas that must be executed, the minimum number of cores that one single replica needs, and the total number of cores available for the overall job. The performance data for RC uses the minimum time per replica for different node types in different hosts, as shown in Fig. 5 for a single BAC replica. This data is collected in the Single Scale Performance Model. In the simplest cost-function, where we consider only time to solution, all replicas would be run concurrently on the node with the shortest running time per replica. However, there are several constraints that the Performance Estimator must also consider such as queue constraints (number of concurrent jobs, time limitations node availability and queueing time).

Most supercomputers have a limit on the number of jobs that can be run or queued at any moment in time per user. For example, on SuperMUC machine, the maximum number of jobs that can be run concurrently on the thin nodes in the "general" queue is 8, while there are no restrictions on the Eagle machine.

As an example, if we have two RC applications which require 40 and 80 replicas respectively, the Pattern-Driven Planner needs to calculate which is faster: running all replicas at one supercomputer SuperMUC (while taking into account the constraint of concurrently running 8 jobs per user) or distributing the jobs among different hosts, for example, across SuperMUC and Eagle, using the functionality in QCG to run across multiple resources. To illustrate how this could be coordinated, let us take the hypothetical situation that there is also a 12 job limit on concurrent jobs running on Eagle to mimic the workload. In Fig. 6, we show the time to completion as a function of the number of "batches" running on SuperMUC, where a "batch" is defined as a set of 8 concurrent running jobs on SuperMUC. The remainder of the replicas are run on Eagle (again in "batches" of up to 12 jobs).

Fig. 6 shows we estimate that for 40 replicas, the shortest time to completion is for 2 "batches" to be run on SuperMUC, while for 80 replicas, the minimum time to completion is for 4 "batches" to be run on SuperMUC. This assumes that all replicas take the same time (the shortest time-to-completion from our benchmarking), that all the replicas are independent (no communication) and that each "batch" runs directly after the other. It is clear there is a

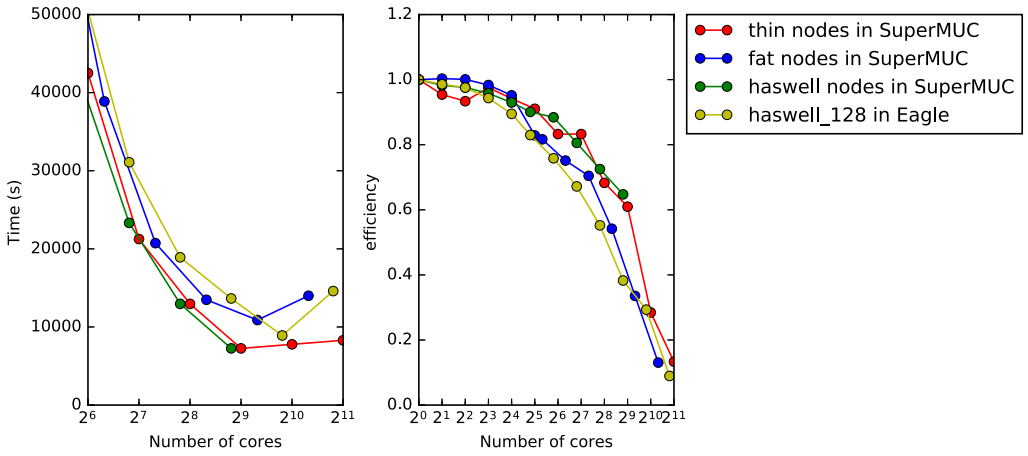


Fig. 5. Time and efficiency per replica (NAMD kernel) on different number of cores on different node types.

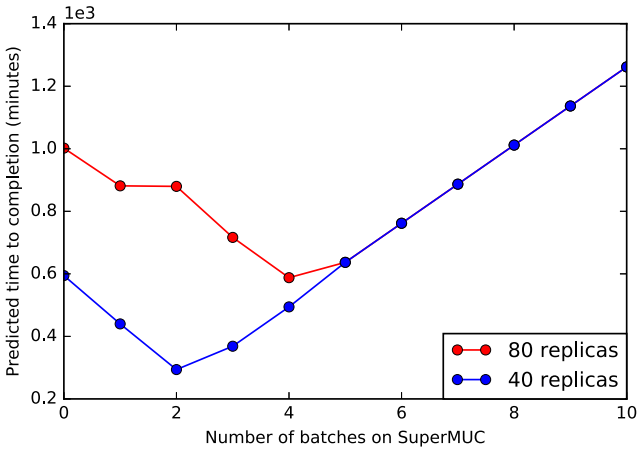


Fig. 6. Theoretical time of running multi-replicas simulations across 2 resources (SuperMUC and Eagle), as a function of number of "batches" (i.e. sets of 8 concurrent jobs) on SuperMUC. The remainder of the replicas are run as batches of up to 12 concurrent jobs on eagle.

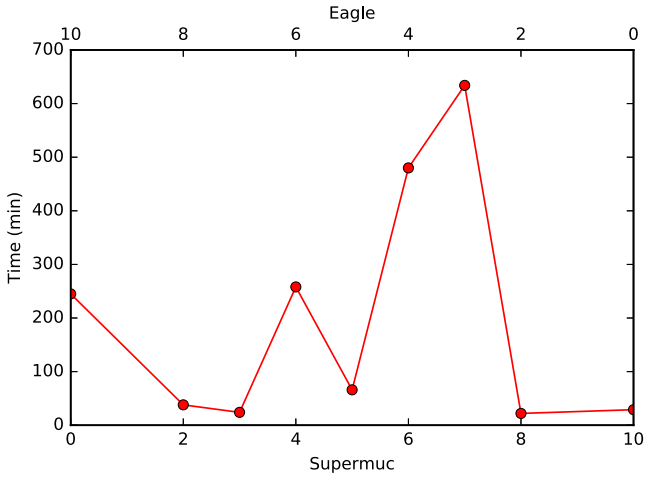


Fig. 7. Time to completion of multi-replicas simulations across 2 resources (SuperMUC and Eagle), as a function of number of replicas, ten in total, distributed on SuperMUC and Eagle.

limitation to this model; it will only be realistic if the time spent in the queue is very short. Otherwise, the time to completion could be very different to that predicted in Fig. 6, and we could envisage the most efficient split in replicas across resources being completely changed if the queueing times are very different across the resources. The estimation of queueing time will be investigated and to incorporated into the middleware in the future (as described in Section 3.3).

The output file description to the execution component is unified among all computing patterns as described in Section 3.3. QCG also have the ability to distribute replicas to the intended machines and gather the results in one place. Fig. 7 shows timings of test BAC runs. In these studies, we run 10 replicas across two supercomputers, SuperMUC and Eagle. By running 8 replicas on SuperMUC and the rest on Eagle we reach the least time-to-completion.

To quantify this speedup [42], we would compare the best timing of distributing replicas T_{distr} with the best timing of running them on SuperMUC (with batch time) T_{local} . The speedup is calculated as:

$$S = \frac{T_{local}}{T_{distr}},$$

and the speedup is 1.2 for our set of studies. This means that at the moment of running this set of replicas, we would gain a speedup due to the varied queuing time. The queueing time will be predicted and hosts will be automatically selected by QCG based on the knowledge of run time and queueing time as stated before. Table 4 summarises the results from the BAC application for time-to-completion runs in Fig. 7.

Table 4
Performance model for RC application, namely BAC. N is the total number of replicas, P_R is the number of cores per replica, T_R is the Time per replica in seconds, T_{local} and T_{distr} are the shortest total simulation time (including queueing times) for a local and a distributed runs and S is the Speedup.

Simulation	N	P_R	T_R (S)	T_{local} (S)	T_{distr} (S)	S
BAC	10	10	20	29	24	1.208

5. Discussion and conclusions

We have introduced and described the Multiscale Computing Patterns software, which extends the Multiscale Modelling and Simulation Framework to enable high performance multiscale computing based on three generic patterns. We demonstrated its usage and added-value for three different types of multiscale applications: fusion and cell based blood flow simulation, both as examples of Extreme Scaling, and binding affinity calculation as an example for Replica Computing. In addition, these multiscale models are based on different coupling approaches, including

MUSCLE2, as well as coupling using scripts and the FabSim automation toolkit.

We implemented and demonstrated the Extreme Scaling and Replica Computing computing patterns. The third computing pattern, Heterogeneous Multiscale Computing, will be implemented, discussed and demonstrated in future work. In the current implementation, each of the demonstrated applications highlights specific strengths of our software approach. For the fusion application, the software abstracts the complication of HPC and chooses the most appropriate number of cores to obtain the required cost criteria (i.e. time to completion). For blood flow, our approach enabled the use of double the resources otherwise accessible. Lastly, for binding affinity calculations, our approach serves to abstract away the choice as to whether the replicas should all run on one and the same computer or be distributed across multiple computers. This automated scheduling approach, which recommends execution across two resources, delivers a time-to-completion speedup of 1.2 compared to the scheduling of all replicas on a single resource.

The Multiscale Computing Patterns software maintains a separation of concerns in three areas. The top layer, the Description component, represents the logical description of the multiscale model. This is the part that is most 'visible' to the application users and developers. The Optimisation component is focused on performance aspects, and provides a number of optimisation criterion based on the type of the multiscale computing pattern and the required criteria. Finally, the Execution component integrates a range of functionalities from the underlying e-infrastructure, and uses the information from the Description and Optimisation components to create and run execution scenarios, each optimised either for minimal time to completion, or minimal energy consumption (given a fixed time to completion requirement). This modular implementation helps multiscale model developers to concentrate on optimising the single scale models of which the application is comprised, without needing to go into details about the HPC machines. The developer can choose the optimisation criteria required.

In this work, we have assembled a range of powerful functionalities for optimising and deploying multiscale applications on large scale HPC infrastructures operating at the multi-petascale, and presented an application-agnostic approach which reduces the development effort required for these purposes. We plan to release the software described here shortly. Generic approaches to High Performance Multiscale Computing are highly sought after across scientific disciplines, and indeed we have already begun propagating the first elements of our approach to other application domains such as astrophysics and materials modelling.

Acknowledgements

We acknowledge partial funding from the European Union Horizon 2020 research and innovation programme under grant agreement No 671564 for the ComPat project (http://www.compat-project.eu/). SA acknowledges funding from King Abdulaziz City for Science and Technology (KACST), Saudi Arabia. AGH acknowledges partial financial support from the Russian Scientific Foundation via grant #14-11-00826. P.V.C. thanks the MRC Medical Bioinformatics project (MR/L016311/1), the EU H2020 CompBioMed grant (http://www.compbiomed.eu, Grant No. 675451) and funding from the UCL Provost. This research was also supported in part by the PLGrid Infrastructure including dedicated HPC resources at the Poznan Supercomputing and Networking Center.

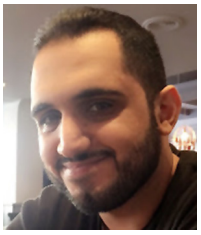
Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.future.2018.08.045.

References

[1] A.G. Hoekstra, B. Chopard, P.V. Coveney, Multiscale modelling and simulation: a position paper, Philos. Trans. Ser. A Math. Phys. Eng. Sci. 372 (2014) 20130377.
[2] A.G. Hoekstra, E. Lorenz, J.-L. Falcone, B. Chopard, Toward a complex automata formalism for multiscale modeling, Int. J. Multiscale Comput. Eng. 5 (6) (2007) 491–502.
[3] B. Chopard, J. Borgdorff, A.G. Hoekstra, A framework for multi-scale modelling, Phil. Trans. R. Soc. A 372 (2014) 20130378.
[4] J. Borgdorff, J.L. Falcone, E. Lorenz, C. Bona-Casas, B. Chopard, A.G. Hoekstra, Foundations of distributed multiscale computing: Formalization, specification, and analysis, J. Parallel Distrib. Comput. 73 (4) (2013) 465–483.
[5] D. Groen, S.J. Zasada, P.V. Coveney, Survey of multiscale and multiphysics applications and communities, Comput. Sci. Eng. 16 (2) (2014) 34–43.
[6] H. Tahir, C. Bona-Casas, A.J. Narracott, J. Iqbal, J. Gunn, P. Lawford, A.G. Hoekstra, Endothelial repair process and its relevance to longitudinal neointimal tissue patterns: comparing histology with in silico modelling, J. R. Soc. Interface / R. Soc. 11 (94) (2014) 20140022.
[7] 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, P.V. Coveney, Flexible composition and execution of high performance, high fidelity multiscale biomedical simulations, Interface Focus 3 (2) (2013) 20120087.
[8] A.G. Hoekstra, S. Alowayyed, E. Lorenz, N. Melnikova, L. Mountrakis, B. van Rooij, A. Svitenkov, G. Závodszky, P. Zun, Towards the virtual artery: a multiscale model for vascular physiology at the physics-chemistry-biology interface, Phil. Trans. R. Soc. A 374 (2080) (2016) 20160146.
[9] O. Hoenen, L. Fazendeiro, B.D. Scott, J. Borgdorff, A.G. Hoekstra, P. Strand, D.P. Coster, Designing and running turbulence transport simulations using a distributed multiscale computing approach, in: EPS 2013, Europhysics Conference Abstracts, Vol. 37D, no. 37, 2013, pp. P4.155.
[10] 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, B. Chopard, Performance of distributed multiscale simulations, Phil. Trans. R. Soc. A 372 (2014) 20130407.
[11] J.L. Suter, D. Groen, P.V. Coveney, Chemically specific multiscale modeling of clay-polymer nanocomposites reveals intercalation dynamics, tactoid self-assembly and emergent materials properties, Adv. Mater. 27 (6) (2015) 966–984.
[12] D.J. Hill, Nuclear energy for the future, Nature Mater. 7 (9) (2008) 680.
[13] R. Delgado-Buscalioni, P.V. Coveney, Continuum-particle hybrid coupling for mass, momentum, and energy transfers in unsteady fluid flow, Phys. Rev. E 67 (4) (2003) 46704.
[14] S.P. Zwart, S. McMillan, A. van Elteren, I. Pelupessy, N. de Vries, Multi-physics simulations using a hierarchical interchangeable software interface, Comput. Phys. Comm. 184 (3) (2013) 456–468.
[15] S. Valcke, The OASIS3 coupler: a European climate modelling community software, Geosci. Model Dev. 6 (2) (2013) 373.
[16] D. Gaston, C. Newman, G. Hansen, D. Lebrun-Grandie, MOOSE: A parallel computational framework for coupled systems of nonlinear equations, Nucl. Eng. Des. 239 (10) (2009) 1768–1778.
[17] P.M.A. Sloot, A.G. Hoekstra, Multi-scale modelling in computational biomedicine, Brief. Bioinform. 11 (1) (2009) 142–152.
[18] A.G. Hoekstra, E. Lorenz, J.-L. Falcone, B. Chopard, Towards a complex automata framework for multi-scale modeling: Formalism and the scale separation map, in: Y. Shi, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science–ICCS 2007, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 922–930.
[19] A.G. Hoekstra, A. Caiazzo, E. Lorenz, J.-L. Falcone, B. Chopard, Complex automata: multi-scale modeling with coupled cellular automata, Simul. Complex Syst. Cell. Autom. (2010) 29–57.
[20] J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. Ben Belgacem, B. Chopard, D. Groen, P.V. Coveney, A.G. Hoekstra, Distributed multiscale computing with MUSCLE 2, the multiscale coupling library and environment, J. Comput. Sci. 5 (5) (2014) 719–731.
[21] T. Piontek, B. Bosak, M. Ciznicki, P. Grabowski, P. Kopta, M. Kulczewski, D. Szejnfeld, K. Kurowski, Development of science gateways using QCG-lessons learned from the deployment on large scale distributed and HPC infrastructures, J. Grid Comput. 14 (4) (2016) 559–573.
[22] J. Knap, C.E. Spear, O. Borodin, K.W. Leiter, Advancing a distributed multi-scale computing framework for large-scale high-throughput discovery in materials science, Nanotechnology 26 (43) (2015) 434004.
[23] S. Alowayyed, D. Groen, P.V. Coveney, A.G. Hoekstra, Multiscale computing in the exascale era, J. Comput. Sci. 22 (2017) 15–25.
[24] B. Bosak, P. Kopta, K. Kurowski, T. Piontek, M. Mamoński, New QosCosGrid middleware capabilities and its integration with European e-infrastructure, in: eScience on Distributed Computing Infrastructure, Springer, 2014, pp. 34–53.

[25] D. Groen, A.P. Bhati, J. Suter, J. Hetherington, S.J. Zasada, P.V. Coveney, FabSim: facilitating computational research through automation on large-scale and distributed e-infrastructures, Comput. Phys. Comm. 207 (2016) 375–385.
[26] K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Weglarz, Multicriteria Aspects of Grid Resource Management, in: International series in Operations Research and Managment Science, vol. 64, 2003, pp. 271–294.
[27] C. January, J. Byrd, X. Oró, M. O'Connor, Allinea MAP: Adding energy and OpenMP profiling without increasing overhead, in: Tools for High Performance Computing 2014, Springer, 2015, pp. 25–35.
[28] K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Weglarz, A multicriteria approach to two-level hierarchy scheduling in grids, J. Sched. 11 (5) (2008) 371–379.
[29] K. Kurowski, A. Oleksiak, W. Piątek, T. Piontek, A. Przybyszewski, J. Węglarz, DCworms-A tool for simulation of energy efficiency in distributed computing infrastructures, Simul. Model. Pract. Theory 39 (2013) 135–151.
[30] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, M. Theimer, OGSA Basic Execution Service version 1.0, 2007.
[31] B. Bosak, J. Komasa, P. Kopta, K. Kurowski, M. Mamoński, T. Piontek, New capabilities in QosCosGrid middleware for advanced job management, advance reservation and co-allocation of computing resources-quantum chemistry application use case, in: Building a National Distributed e-Infrastructure-PL-Grid, Springer, 2012, pp. 40–55.
[32] Leibniz-Rechenzentrum, SuperMUC Petascale System. URL https://www.lrz.de/services/compute/supermuc/.
[33] Poznań-Supercomputing-Networking-Center, Eagle. URL https://wiki.man.poznan.pl/hpc/index.php/Eagle.
[34] G.L. Falchetto, D. Coster, R. Coelho, B.D. Scott, L. Figini, D. Kalupin, E. Nardon, S. Nowak, L.L. Alves, J.F. Artaud, Corrigendum: The European integrated tokamak modelling (ITM) effort: achievements and first physics results (2014 Nucl. Fusion 54 043018), Nucl. Fusion 54 (9) (2014) 99501.
[35] FlowKit-Ltd, Palabos. URL www.palabos.org.
[36] L. Mountrakis, E. Lorenz, O. Malaspinas, S. Alowayyed, B. Chopard, A.G. Hoekstra, Parallel performance of an IB-LBM suspension simulation framework, in: International Conference on Computational Science, Elsevier, Reykjavík, Iceland, 2015, p. 10.
[37] G. Závodszky, B. van Rooij, V. Azizi, A.G. Hoekstra, Cellular level in-silico modeling of blood rheology with an improved material model for red blood cells, Front. Phys. 8 (2017) 563.
[38] HEMOCELL A high-performance framework for dense cellular suspension flows. URL https://www.hemocell.eu/.
[39] G. Zavodszky, B. van Rooij, V. Azizi, S. Alowayyed, A.G. Hoekstra, Hemocell: a high-performance microscopic cellular library, Procedia Comput. Sci. 108 (2017) 159–165.
[40] S.K. Sadiq, D. Wright, S.J. Watson, S.J. Zasada, I. Stoica, P.V. Coveney, Automated molecular simulation based binding affinity calculator for ligand-bound HIV-1 proteases, 2008.
[41] A.P. Bhati, S. Wan, D.W. Wright, P.V. Coveney, Rapid, accurate, precise, and reliable relative free energy prediction using ensemble based thermodynamic integration, J. Chem. Theory Comput. 13 (1) (2017) 210–222.
[42] A.G. Hoekstra, P.M.A. Sloot, Introducing Grid speedup Γ : A scalability metric for parallel applications on the grid, in: P.M.A. Sloot, A.G. Hoekstra, T. Priol, A. Reinefeld, M. Bubak (Eds.), Advances in Grid Computing - EGC 2005, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 245–254.

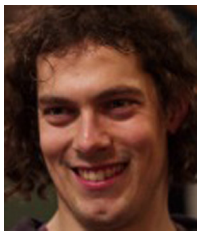


Saad A. Alowayyed is a Ph.D. candidate in Computational Science Lab (CSL) at the University of Amsterdam. His thesis is concerned with Multiscale computing patterns. He received his Master in high performance computing from the University of Edinburgh. Saad also works as a researcher in King Abdulaziz for Science Technology (KACST).

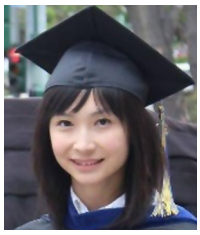
Tomasz Piontek graduated from Poznań University of Technology in Computer Science. He is a member of Applications Department at Poznań Supercomputing and Networking Center, Poland and head of the Large Scale Applications and Services Department. He has been involved in many EU-funded R&D projects in the areas of distributed and parallel computing, including QosCosGrid and MAPPER. His research activities are focused on the modelling of advanced applications for modern hybrid HPC architectures, and online services scalability and availability.

James L. Suter received his M.Chem. from the University of Oxford and Ph.D. from the University of Cambridge under the supervision of Professor Michiel Sprik. His research addresses multiscale materials modeling and analysis of clay and graphene nanocomposites using high-performance computing.

Olivier Hoenen is a Post Doctoral Research Associate at the Max-Planck-Institut fuer Plasmaphysik (IPP) in the Numerical Methods for Plasma Physics Division, specialized in parallel computing and adaptive methods. He was involved in the Infrastructure Project for the ITM-TF, and in the EU FP7 project EUFORIA. He participated in the MAPPER project where he was interested in distributed multiscale simulations for plasma physics. He is involved in the EUROfusion Infrastructure and Support Activities where he maintains the integrated modelling and simulation platform and participates to further developments of the ITER platform IMAS.



Derek Groen is a Lecturer in Simulation and Modelling at Brunel University London, and a visiting Lecturer at University College London. He specializes in multiscale simulation, high performance computing and automation, and has published 45 peer-reviewed papers. He has created multiscale models and performed validation studies using the HemeLB bloodflow simulation environment, and won the ARCHER Early Career Impact Award in 2015 as a result. He obtained his PhD from the University of Amsterdam in 2010, where he ran large cosmological simulations geographically distributed across up to four supercomputers.

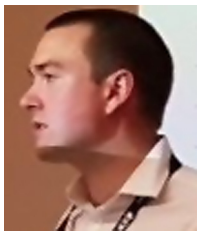


Onnie Luk is a Ph.D. graduate from the University of California at Irvine, where she conducted her research on the role of convective cell in nonlinear interaction of kinetic Alfven waves. She also has experience in analyzing magnetometer data obtained by the Galileo spacecraft. Currently, she is a post doctoral researcher at the Max-Planck-Institut fÃijr Plasmaphysik. She is part of the ComPat project, in which she explores time bridging methods to connect turbulence and transport models in a component-based multiscale fusion plasma simulation.

Bartosz Bosak received his master's degree in computer science in 2007 from Poznań University of Technology in Poland (Laboaratory of IT Systems in Management). Since 2006 he has been working at the Application Department of Poznan Supercomputing and Networking Center as a systems analyst and developer. His research interests include widely undrestood support for large scale computations on Grid, Cloud and HPC infrastructures, multiscale computing as well as system integration. He was a participant of a variety of European and national projects including BREIN, MAPPER, ComPat, Geant and PLGrid.

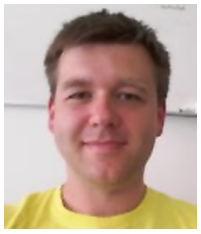
Piotr Kopta received his M.Sc. degree in Computer Science from the Technical University of Częstochowa in 2002. Currently he is an systems analyst at the Poznań Supercomputing and Networking Center. His research interests concern high performance computing in particulary new computational architectures.

Krzysztof Kurowski holds a Ph.D. degree in Computer Science. He graduated from Poznan University of Technology. He has been leading Applications Department at Poznan Supercomputing and Networking Center in Poland since 2008. He has been actively involved in many international R&D projects in the area of computer science, HPC, and ICT, including GridLab and ComPat. He was a research visitor at University of Queensland, Argonne National Lab, and at CCT Louisiana University. His research activities are focused on advanced applications, computing simulations, scheduling and resource management in networked environments.



Oliver Perks is a Field Application Engineer at Arm, providing application porting and optimisation support to customers, through the use of the Arm HPC tools. Oliver obtained his Ph.D. from Warwick University, in profiling of HPC applications, and subsequently moved into industry to practice performance optimisation for large scale production workloads. Having joined Allinea, Oliver began working on a number of H2020 projects, including ComPat, and continues that involvement as a representative of Arm.

Keeran Brabazon has a Ph.D. in Scientific Computation from the University of Leeds. In his current role at Arm he specialises in the performance analysis of high-performance computing (HPC) simulations, with a focus on the identification of specific application performance bottlenecks through targeted sparse data collection. Keeran works in the team developing the Arm Forge and Performance Reports HPC tools.



Vytautas Jančauskas got his Ph.D. from Vilnius University in 2016. The subject of the thesis was evaluating the performance of multi-objective optimisation methods. He worked as a lecturer and assistant lecturer for more than 3 years. Supervised undergraduate computer networks course at the Faculty of Mathematics and Informatics at Vilnius University. Vytautas contributed extensively to various open-source projects, via the Google's Summer of Code program. He has joined the ComPat project at LRZ in 2016.

David Coster is a senior researcher at the Max-Planck-Institut für Plasmaphysik, where he leads the Edge Physics group in the Division of Tokamak Physics, one of the two theory divisions at IPP Garching. He was the Project Leader for Integrated Modelling Project 3 (Core and Edge Transport) within the EFDA ITM Task Force. As well as being a Deputy Task Force Leader, Dr Coster was the deputy coordinator of the seventh European Framework Programme (FP7) project EUFORIA and was also involved in the FP7 project MAPPER. His own research has concentrated on understanding the behaviour of the edge and scrape-off layer regions of the tokamak plasma. He is also a member of the EUROfusion IT committee and deputy coordinator of the sub-committee developing an approach to Open Data within EUROfusion.

Peter Coveney holds a Chair in Physical Chemistry, is Director of the Centre for Computational Science, and is an Honorary Professor in Computer Science at UCL. He is also Professor Adjunct at Yale University, he is active in a broad area of interdisciplinary theoretical research including condensed matter physics and chemistry, materials science, life and medical sciences. He has published over 400 scientific papers, edited 16 journal "theme issues", and authored three books, including two best-selling popular science books.

Alfons Hoekstra holds a Ph.D. in Computational Science from the University of Amsterdam and currently is a professor in Computational Science at the University of Amsterdam and the national research university ITMO, St Petersburg, Russia. His research focuses on multi-scale multi-science modelling, large-scale simulations, and high performance computing, mainly in the biomedical domain and complex systems science. He has a long-standing expertise in Computational Biomedicine, Complex Systems simulations, and high performance parallel and distributed computing. He has published over 250 research papers. He currently leads the Computational Science Lab at the University of Amsterdam.

Publication [P5]

Bosak, B., Piontek, T., Karlshoefer, P., Raffin, E., Lakhlili, J., and Kopta, P.: *Verification, validation and uncertainty quantification of large-scale applications with QCG-PilotJob.* In Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M., editors, *Computational Science – ICCS 2021*, pages 495–501, Cham. Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-77977-1_39

Ministry points / conference: 140

Contribution of authors:

- **Bartosz Bosak**
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, Related work, Objectives, Use cases, Summary and future work)
- Piotr Kopta
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Objectives, Performance evaluation, Summary and future work)
- Tomasz Piontek
  - Authorship of the idea underlying the paper, review and editing
  - Coauthorship of the text of publication (Sections: Introduction, Summary and future work)
- Paul Karlshoefer, Erwan Raffin
  - Coauthorship of the text of publication (Sections: Performance evaluation)
- Jalal Lakhlili
  - Coauthorship of the text of publication (Sections: Use cases)

Verification, Validation and Uncertainty Quantification of large-scale applications with QCG-PilotJob

Bartosz Bosak[1], Tomasz Piontek[1], Paul Karlshoefer[2], Erwan Raffin[2], Jalal Lakhlili[3], and Piotr Kopta[1]

[1] Poznań Supercomputing and Networking Center, Poznań, Poland
[2] CEPP – Center for Excellence in Performance Programming, Atos, France
[3] Max-Planck Institute for Plasma Physics - Garching, Munich, Germany

Abstract

Efficient execution of large-scale and extremely demanding computational scenarios is a challenge for both the infrastructure providers and end-users, usually scientists, that need to develop highly scalable computational codes. Nevertheless, at this time, on the eve of exa-scale supercomputers, the particular role has to be given also to the intermediate software that can help in the preparation of applications so they can be efficiently executed on the emerging HPC systems. The efficiency and scalability of such software can be seen as priorities, however, these are not the only elements that should be addressed. Equally important is to offer software that is elastic, portable between platforms of different sizes, and easy to use. Trying to fulfill all the above needs we present QCG-PilotJob, a tool designed to enable flexible execution of numerous potentially dynamic and interdependent computing tasks in a single allocation on a computing cluster. QCG-PilotJob is built on many years of collaboration with computational scientists representing various domains and it responses to the practical requirements of real scientific use-cases. In this paper, we focus on the recent integration of QCG-PilotJob with the EasyVVUQ library and its successful use for Uncertainty Quantification workflows of several complex multiscale applications being developed within the VECMA project. However, we believe that with a well-thought-out design that allows for fully user-space execution and straightforward installation, QCG-PilotJob may be easily exploited in many other application scenarios, even by inexperienced users.

1 Introduction

The success of scientific research can be evaluated based on its applicability for solving real-world problems. Not surprisingly, before computational simulation codes are used in production, their robustness needs to be strictly proven. This applies to both, the quality of the code itself and, even more importantly, the reliability of the generated results. To this end, scientists employ VVUQ procedures to verify, validate, and precisely quantify the uncertainty of calculations.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_39

The inherent characteristic of the majority of available techniques is multiple evaluations of models using different input parameters selected from the space of possible values. This is a computationally demanding scenario even for evaluations of traditional single-scale applications, but in the case of multiscale simulations that consist of many coupled single-scale models, the problem becomes a real challenge. There is a need to support simultaneous execution of single-scale models that may have extremely large and different resource requirements, some single-scale models may need to be attached dynamically, the number of evaluations of single-scale models may not be known in advance, to name just a few difficulties. Within the VECMA project[4] we are trying to resolve all these issues with efficient and flexible tools. One of key elements in VECMA toolkit[5] [4] is EasyVVUQ[6] [8], the user-facing library that brings VVUQ methods to many different use-cases. However, EasyVVUQ itself abstracts from the aspects of execution of model evaluations on computational resources and outsources this topic to the external tools. One of them is QCG-PilotJob[7] (QCG-PJ) developed by Poznan Supercomputing and Networking Center as part of QCG (Quality in Cloud and Grid) middleware [7]. Its functionality, based on the idea of so called *Pilot Job*, which manages a number of subordinate tasks, is essential for the flexible and efficient execution of VVUQ scenarios that inherently consists of many tasks, from which some may be relatively small.

The rest of this paper is structured as follows. In Section 2 we present a brief overview of related work. In Section 3 we describe basic objectives and functionality of QCG-PJ. Next, in Section 4, we introduce a few schemes of integration between EasyVVUQ and QCG-PJ that have been developed in the VECMA project and then we present a range of application use-cases that already use QCG-PJ. The results of performance tests conducted so far are outlined in Section 5. Finally, in Section 6, we conclude and share main plans for the future.

2 Related work

The problem of efficient and automated execution of a large-number of tasks on computing clusters managed by queuing systems is known from decades. One of the most recognized systems that deal particularly with the pilot job style of execution on computing resources is RADICAL-Pilot[6], being developed as part of the RADICAL-Cybertools suite. In contrast to QCG-PJ, which can be easily installed in a user's home directory, RADICAL-Pilot is not a self-contained component and needs to be integrated with external services. There are also several solutions having some commonalities with the QCG-PJ idea, but they are focused primarily on the workflow orchestration rather than on the efficiency and flexibility of computing on HPC machines. An example of a mature system is here Kepler[2], which addresses the need for the HTC execution of parts of

[4] https//www.vecma.eu
[5] https://www.vecma-toolkit.eu
[6] https://github.com/UCL-CCS/EasyVVUQ
[7] https://github.com/vecma-project/QCG-PilotJob

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_39

the workflows on clusters. Further examples are Swift-T[9] and Parsl[1], which share a common goal to effectively support data-oriented workflows, or Dask[8], which aims to enable parallel processing for analytical workflows.

3 Objectives

The access to HPC systems is regulated by the policies of resource providers and restricted by local resource management system configurations as well as their implementations. For example, the policy at SuperMUC-NG cluster installed at the Leibniz Supercomputing Centre[9], in order to promote large-scale computing, allows users to submit and run only a small number of jobs at the same time. Smaller HPC installations may be less restrictive, but in general, the large tasks have a priority over small tasks, and the rule remains the same: there is no way to flexibly schedule many jobs with basic mechanisms. If users want to efficiently run a huge number of conceptually different tasks they need to employ solutions that can mitigate the regulations on a level of single allocation. One of the possible approaches is to define a processing scheme in a scripting language, but this is neither generic nor flexible and possibly prone to many bugs and inefficiency. The other, recommended approach is the utilisation of specialised software, like QCG-PJ.

Functionality

The basic idea of QCG-PJ is to bring an additional tasks management level within the already created allocation. As it is presented in Figure 1, from the queuing system's perspective, QCG-PJ is only a single regular task, but for a user, it is a second-level lightweight resource management system that can be administered and used on an exclusive basis.

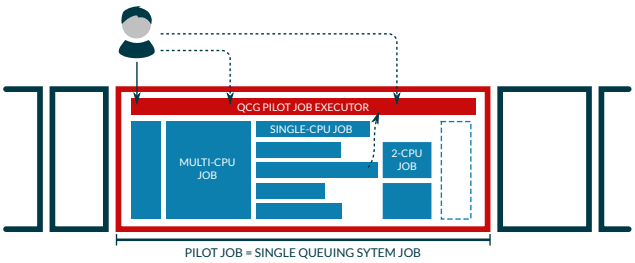


Fig. 1. The general computation scheme in QCG-PilotJob

As SLURM manages resources of a cluster, QCG-PJ manages resources of an allocation and ensures that tasks are scheduled efficiently. That being said,

[8] https://dask.org
[9] https://doku.lrz.de/display/PUBLIC/Job+Processing+with+SLURM+on+SuperMUC-NG

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_39

users or alternatively client software components can interact with it in a similar way as with any queuing system. Through a dedicated lightweight service called QCG-PJ Scheduler, they can submit new tasks with specific requirements, list submitted tasks or cancel them.

In order to enhance usability, QCG-PJ offers two ways of submission of tasks: firstly, it is possible to provide task definitions in a form of JSON file and submit this file from a command line at startup of the QCG-PJ, and secondly, it is possible to use the provided python API for dynamic creation and management of jobs directly from a running python program.

Moreover, the tool provides a few supplementary built-in features that can be recognised as particularly useful for selected scenarios. Among others, it allows defining dependencies between tasks as well as it offers resume mechanism to support fault tolerance at a workflow level.

Architecture towards exascale

One of the biggest challenges at the very beginning of QCG-PJ development was to ensure its ability to meet requirements defined by extremely demanding multiscale applications. In order to reach the performance of hundreds of petaflops or even higher, these applications ultimately call for, it was particularly important to design an appropriate architecture. First of all, such architecture should be scalable: the system should be easy to use, even on a laptop, but also easily extendable, portable and efficient once the use-cases grow up to require HPC resources. Consequently, the natural choice was to propose a hierarchical structure of components, where top-level services are released from the high-intensive processing that can be performed in a distributed way by low-level services. In consequence, the QCG-PJ architecture includes a concept of partition, which reflects a subset of resources that are managed separately and can be dynamically attached to the optional top-level QCG-PJ Scheduling Queue service. This is presented in Figure 2.

In the presented full-scale deployment scenario, QCG-PJ Scheduling Queue is an entry point to the system and keeps global information about all tasks that should be processed. One or multiple QCG-PJ Scheduler services, associated with the elementary partitions, can request Scheduling Queue for a portion of tasks to execute. Once the tasks are completed, the schedulers report this information back to the central service. Consequently, resources coming from a single or many allocations, also from a single or many HPC clusters, can be robustly integrated and offered as a single logical concept, while the communication overhead is minimised.

Having a closer look at the logic present within a single partition, two elements should be noted. The first of them is the possibility to reserve a core for QCG-PJ Scheduler. This option is useful when processing done by the Scheduler service significantly influences the actual computations. The second element is the Node Launcher service. This component is designed to improve the startup efficiency of single-core tasks.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_39

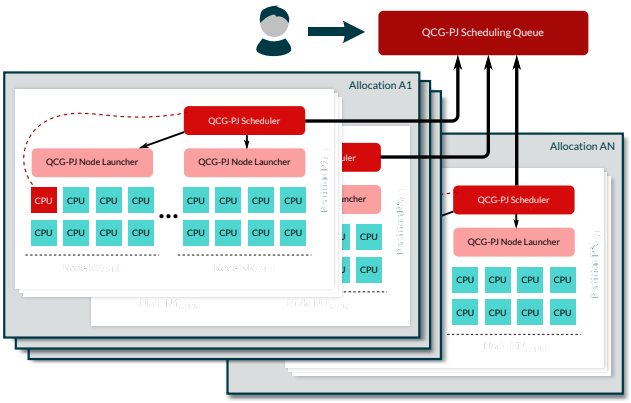


Fig. 2. QCG-PilotJob Architecture Overview

4 Use cases

Integration with EasyVVUQ

EasyVVUQ is a tool for domain experts who work on concrete VVUQ scenarios related to their applications. We argue that these experts shouldn't spend their valuable time to set-up the logic of execution of EasyVVUQ workflows on computing resources. Rather, they should focus on purely scientific or engineering aspects. To this end, VECMA toolkit provides a few approaches for the integration of EasyVVUQ with QCG-PJ so it can be efficient and natural for the domain scientist. Currently, there are the following possibilities:

Direct integration with EasyVVUQ: It is the most straightforward type of integration, where QCG-PJ is transparently employed in EasyVVUQ as one of its internal execution engines. Although at the moment of writing, this type of integration is not yet completed and doesn't benefit from more advanced features offered by QCG-PJ, e.g. iterative tasks, it is expected to be the preferred one at some point in the future.

Integration through the EQI library: EQI[10], which stands for EasyVVUQ-QCGPilotJob Integrator, is a lightweight library designed to bring optimised processing schemes to selected types of highly-demanding EasyVVUQ workflows. It makes use of advanced functionalities of QCG-PJ, like resume mechanism and iterative jobs.

Integration through FabSim3: QCG-PJ has been integrated with the FabSim3 automation toolkit[11] in order to support demanding application campaigns. Since FabSim3 internally uses EasyVVUQ, the combined execution of EasyVVUQ and QCG-PJ is also possible.

[10] https://github.com/vecma-project/EasyVVUQ-QCGPJ
[11] https://github.com/djgroen/FabSim3

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_39

Applications

At the moment of writing, there are already several application teams from VECMA that use QCG-PJ for their professional research. For instance, scientists from Max-Planck Institute for Plasma Physics use EasyVVUQ to quantify the propagation of uncertainty in a fusion turbulence model which is computationally expensive. It is a 3D parallel code and needs 512 to 16384 MPI cores [5]. Running a UQ campaign on such models require a very large number of jobs. Thanks to EQI and QCG-PJ, it was possible to execute the required simulations in a single batch allocation. In a similar way, the QCG-PJ tool has been employed for UQ of the UrbanAir application developed by PSNC [10]. It is also worth mentioning recent studies, where FabSim3 and QCG-PJ have been employed for UQ performed on the CovidSim epidemiological code [3].

5 Performance evaluation

Ultimately, the performance of QCG-PJ will be the most determining factor for its usability. Since the early days of its development, scalability and accessibility are evaluated repeatedly on large European supercomputers such as SuperMUC-NG at LRZ and Eagle/Altair at PSNC.
Thus far, test runs involving 100 dual-socket nodes, which equates to around 5000 CPU cores, showed very promising results, with more than 99% of time spent in the user-defined pilot jobs. More specifically, 20.000 pilot jobs with a runtime of five minutes each kept 99.2% of the available resources occupied.
With these promising results, we plan on conducting experiments with actual scientific applications which involve much larger node counts, alongside an exhaustive scalability study. Additionally, these tests were conducted by users which are not directly involved in the development of QCG-PJ, which in turn further contributed to the accessibility of the API.

6 Summary and future work

In this paper, we shortly introduced the concepts and features of QCG-PilotJob system and depicted how it is used by VECMA project to support demanding VVUQ scenarios. The progress made to several large-scale applications, when they successfully employed QCG-PJ, allows us to rank the current usability of QCG-PJ relatively high. Nevertheless, there are still ongoing works aimed to enhance the quality and functionality of the software. In regards to the former, since individual SLURM configurations can pose challenges for QCG-PJ and require its adaptation and customization, we are in the process of extensive tests of the tool on high-end European clusters. For instance, PSNC is in the process of deploying QCG-PJ to SURF (Amsterdam) and ARCHER2 (Edinburgh). Ultimately, we want to ensure that QCG-PJ is easily deployable and works efficiently on a large variety of machines and configurations. In terms of new functionality, our aim is to complete the implementation of the global Scheduling Queue service as well as to provide a dedicated monitoring solution.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_39

Acknowledgments This work received funding from the VECMA project realised under grant agreement 800925 of the European Union's Horizon 2020 research and innovation programme. We are thankful to the Poznan Supercomputing and Networking Center for providing its computational infrastructure. We are also grateful to the VECMA partners for the invaluable motivation.

References

1. Babuji, Y., Woodard, A., Li, Z., Katz, D.S., Clifford, B., Kumar, R., Lacinski, L., Chard, R., Wozniak, J.M., Foster, I., Wilde, M., Chard, K.: Parsl: Pervasive parallel programming in python. In: Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing. p. 25–36. HPDC '19, New York, NY, USA (2019). https://doi.org/10.1145/3307681.3325400
2. Cabellos, L., Campos, I., del Castillo, E.F., Owsiak, M., Palak, B., Płóciennik, M.: Scientific workflow orchestration interoperating htc and hpc resources. Computer Physics Communications **182**(4), 890–897 (2011). https://doi.org/10.1016/j.cpc.2010.12.020
3. Edeling, W., Arabnejad, H., Sinclair, R., Suleimenova, D., Gopalakrishnan, K., Bosak, B., Groen, D., Mahmood, I., Crommelin, D., Coveney, P.V.: The impact of uncertainty on predictions of the covidsim epidemiological code. Nature Computational Science **1**(2), 128–135 (2021). https://doi.org/10.1038/s43588-021-00028-9
4. Groen, D., Arabnejad, H., Jancauskas, V., Edeling, W.N., Jansson, F., Richardson, R.A., Lakhlili, J., Veen, L., Bosak, B., Kopta, P., Wright, D.W., Monnier, N., Karlshoefer, P., Suleimenova, D., Sinclair, R., Vassaux, M., Nikishova, A., Bieniek, M., Luk, O.O., Kulczewski, M., Raffin, E., Crommelin, D., Hoenen, O., Coster, D.P., Piontek, T., Coveney, P.V.: Vecmatk: a scalable verification, validation and uncertainty quantification toolkit for scientific simulations. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **379**(2197), 20200221 (2021). https://doi.org/10.1098/rsta.2020.0221
5. Luk, O., Hoenen, O., Bottino, A., Scott, B., Coster, D.: Compat framework for multiscale simulations applied to fusion plasmas. Computer Physics Communications (2019). https://doi.org/10.1016/j.cpc.2018.12.021
6. Merzky, A., Turilli, M., Titov, M., Al-Saadi, A., Jha, S.: Design and performance characterization of radical-pilot on leadership-class platforms (2021)
7. Piontek, T., Bosak, B., Ciznicki, 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**, 559–573 (2016)
8. Richardson, R., Wright, D., Edeling, W., Jancauskas, V., Lakhlili, J., Coveney, P.: Easyvvuq: A library for verification, validation and uncertainty quantification in high performance computing. Journal of open research software **8**, 1–8 (2020)
9. Wozniak, J.M., Armstrong, T.G., Wilde, M., Katz, D.S., Lusk, E., Foster, I.T.: Swift/t: Large-scale application composition via distributed-memory dataflow processing. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. pp. 95–102 (2013). https://doi.org/10.1109/CCGrid.2013.99
10. Wright, D.W., Richardson, R.A., Edeling, W., Lakhlili, J., Sinclair, R.C., Jancauskas, V., Suleimenova, D., Bosak, B., Kulczewski, M., Piontek, T., Kopta, P., Chirca, I., Arabnejad, H., Luk, O.O., Hoenen, O., Weglarz, J., Crommelin, D., Groen, D., Coveney, P.V.: Building confidence in simulation: Applications of easyvvuq. Advanced Theory and Simulations **3**(8), 1900246 (2020)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_39

Publication [P6]

Groen, D., Arabnejad, H., Jancauskas, V., Edeling, W. N., Jansson, F., Richardson, R. A., Lakhlili, J., Veen, L., **Bosak, B.**, Kopta, P., Wright, D. W., Monnier, N., Karlshoefer, P., Suleimenova, D., Sinclair, R., Vassaux, M., Nikishova, A., Bieniek, M., Luk, O. O., Kulczewski, M., Raffin, E., Crommelin, D., Hoenen, O., Coster, D. P., Piontek, T., and Coveney, P. V.: *Vecma toolkit: a scalable verification, validation and uncertainty quantification toolkit for scientific simulations*. Philosophical Transactions of the Royal Society A, 379(2197):20200221 (2021). https://doi.org/10.1098/rsta.2020.0221

Ministry points / journal: 100

Contribution of authors:

- Derek Groen
  - Authorship of the idea underlying the paper and coordination of writing
  - Coauthorship of the text of publication (Sections: Introduction, Related work, The VECMA toolkit (VECMAtk), Overview of the VECMAtk components, Exemplar applications, Toolkit performance and scalability, Conclusion)
  - Coauthorship of the text of publication (Sections: Introduction, Related work, The VECMA toolkit (VECMAtk), Overview of the VECMAtk components, Exemplar applications, Toolkit performance and scalability, Conclusion)
- Hamid Arabnejad, Diana Suleimenova
  - Coauthorship of the text of publication (Sections: Overview of the VECMAtk components / FabSim3, Exemplar applications / Forced Human Migration, Toolkit performance and scalability)
- Peter V. Coveney
  - Authorship of the idea underlying the paper and supervision
  - Coauthorship of the text of publication (Sections: Introduction, Related work, Exemplar applications, Toolkit performance and scalability, Conclusion)
- **Bartosz Bosak**
  - Coauthorship of the text of publication (Sections: Overview of the VECMAtk components / QCG tools and VECMAtk workflows, Toolkit performance and scalability)
- Tomasz Piontek, Piotr Kopta
  - Coauthorship of the text of publication (Sections: Overview of the VECMAtk components / QCG tools, Toolkit performance and scalability)

- Robin A. Richardson, David W. Wright, Wouter N. Edeling, Vytautas Jancauskas
  - Coauthorship of the text of publication (Sections: Overview of the VECMAtk components / EasyVVUQ)
- Jalal Lakhlili
  - Coauthorship of the text of publication (Sections: Overview of the VECMAtk components / EasyVVUQ, Exemplar applications / Fusion)
- Laurens Veen
  - Coauthorship of the text of publication (Section: Overview of the VECMAtk components / MUSCLE3, Exemplar applications / Biomedicine)
- Anna Nikishova
  - Coauthorship of the text of publication (Section: Exemplar applications / Biomedicine)
- Onnie O. Luk, Olivier Hoenen, David Coster
  - Coauthorship of the text of publication (Section: Exemplar applications / Fusion)
- Michał Kulczewski
  - Coauthorship of the text of publication (Section: Exemplar applications / Urban air pollution)
- Maxime Vassaux, Robert C. Sinclair
  - Coauthorship of the text of publication (Section: Exemplar applications / Advanced materials)
- Fredrik Jansson
  - Coauthorship of the text of publication (Section: Exemplar applications / Climate)
- Daan Crommelin
  - Coauthorship of the text of publication (Section: Exemplar applications / Coronavirus modelling)
- Paul Karlshoefer, Erwan Raffin, Nicolas Monnier
  - Coauthorship of the text of publication (Section: Toolkit performance and scalability)
- Mateusz Bieniek
  - Coauthorship of the text of publication (Section: The Vecma toolkit (VECMAtk))

PHILOSOPHICAL TRANSACTIONS A

royalsocietypublishing.org/journal/rsta

Research

Cite this article: Groen D *et al*. 2021 VECMAtk: a scalable verification, validation and uncertainty quantification toolkit for scientific simulations. *Phil. Trans. R. Soc. A* **379**: 20200221.
https://doi.org/10.1098/rsta.2020.0221

Accepted: 10 November 2020

One contribution of 15 to a theme issue 'Reliability and reproducibility in computational science: implementing verification, validation and uncertainty quantification *in silico*'.

Subject Areas:
computer modelling and simulation, software, statistics

Keywords:
multiscale simulations, verification, validation, uncertainty quantification

Author for correspondence:
D. Groen
e-mail: derek.groen@brunel.ac.uk

VECMAtk: a scalable verification, validation and uncertainty quantification toolkit for scientific simulations

D. Groen[1,2], H. Arabnejad[1], V. Jancauskas[3], W. N. Edeling[4], F. Jansson[4,11], R. A. Richardson[2,5], J. Lakhlili[6], L. Veen[5], B. Bosak[7], P. Kopta[7], D. W. Wright[2], N. Monnier[8], P. Karlshoefer[8], D. Suleimenova[1], R. Sinclair[2], M. Vassaux[2], A. Nikishova[9], M. Bieniek[2], Onnie O. Luk[6], M. Kulczewski[7], E. Raffin[8], D. Crommelin[4,10], O. Hoenen[6], D. P. Coster[6], T. Piontek[7] and P. V. Coveney[2,9]

[1]Department of Computer Science, Brunel University London, London, UK
[2]Centre for Computational Science, University College London, London, UK
[3]Leibniz Supercomputing Centre, Garching, Germany
[4]Centrum Wiskunde and Informatica, Amsterdam, The Netherlands
[5]Netherlands eScience Center, Amsterdam, The Netherlands
[6]Max Planck Institute for Plasma Physics - Garching, Munich, Germany
[7]Poznań Supercomputing and Networking Center, Poznań, Poland
[8]CEPP - Center for Excellence in Performance Programming, Atos Bull, Rennes, France
[9]Computational Science Lab, Institute for Informatics, University of Amsterdam, Amsterdam, The Netherlands

THE ROYAL SOCIETY PUBLISHING

[10]Korteweg-de Vries Institute for Mathematics, Amsterdam, The Netherlands
[11]Department of Geoscience and Remote Sensing, Delft University of Technology, Delft, The Netherlands

DG, 0000-0001-7463-3765; HA, 0000-0002-0789-1825; VJ, 0000-0002-6051-210X; WE, 0000-0003-4734-7960; DS, 0000-0003-4474-0943; AN, 0000-0003-4813-9282; MKB, 0000-0002-3065-5417; OL, 0000-0003-0560-4797; ER, 0000-0003-0359-5372; PC, 0000-0002-8787-7256

We present the VECMA toolkit (VECMAtk), a flexible software environment for single and multiscale simulations that introduces directly applicable and reusable procedures for verification, validation (V&V), sensitivity analysis (SA) and uncertainty quantication (UQ). It enables users to verify key aspects of their applications, systematically compare and validate the simulation outputs against observational or benchmark data, and run simulations conveniently on any platform from the desktop to current multi-petascale computers. In this sequel to our paper on VECMAtk which we presented last year [1] we focus on a range of functional and performance improvements that we have introduced, cover newly introduced components, and applications examples from seven different domains such as conflict modelling and environmental sciences. We also present several implemented patterns for UQ/SA and V&V, and guide the reader through one example concerning COVID-19 modelling in detail.

This article is part of the theme issue 'Reliability and reproducibility in computational science: implementing verification, validation and uncertainty quantification *in silico*'.

1. Introduction

Computational models play an ever-growing role in predicting the behaviour of real-world systems or physical phenomena [2], using equations and/or heuristics to encode the natural laws and theories of the world we inhabit. Because models are a simplified representation of real-world systems, they can behave differently for a variety of reasons. Key model inputs, such as initial conditions, boundary conditions and important parameters controlling the model, are often not known with certainty or are inadequately described [3]. For example, in the case of human migration simulations [4], the model may require knowledge of a wide range of input parameters, such as details of transport modes, roads, settlement populations and conflict zones, before we can execute the simulation. However, often these parameters are not precisely known or cannot be obtained with high accuracy.

Another source of discrepancy between the model and reality are the assumptions and simplifications that are made as part of creating the conceptual model. Simplifications can reduce the computational cost of models, but also make them less accurate. And assumptions are by definition uncertain, which makes it necessary to test the model accuracy when these assumptions are adjusted to match realistic alternative scenarios.

The appropriate level of accuracy and reliability in the results can be obtained by ensuring *not only* that the computational model accurately represents the underlying mathematical model and its solution (*Verification*) [5,6], but the degree to which a model is an accurate representation of the real world based on comparisons between computational results and experimental data (i.e. the deviation of the model from reality) (*Validation*) [7], and how variations in the numerical and physical parameters affect simulation outcomes (*Uncertainty Quantification*). Collectively, the processes involved in evaluating our level of trust in the results obtained from models are known as VVUQ. VVUQ processes provide the basis for determining our level of trust in any given model and the results obtained using it [2,8,9].

Many scientific modelling challenges involve complex and possibly multiscale, multiphysics phenomena, the results of which are unavoidably approximate. VVUQ then becomes essential

royalsocietypublishing.org/journal/rsta *Phil. Trans. R. Soc. A* **379**: 20200221

to determine whether the results can be trusted, and whether decision makers can rely on them to guide subsequent actions, making our simulations *actionable*. Indeed, the impact of scientific computing relies directly on its trustworthiness [6], and VVUQ provides the basis for determining our level of trust in any given model and the results obtained using it [2].

Within this paper, we describe the latest release of the VECMA toolkit. VECMA (www.vecma.eu) is an EU-funded initiative that focuses on enabling VVUQ for large-scale and complex simulations. The toolkit we present here facilitate the use of VVUQ techniques in (multiscale) applications, as well as a range of Verification and Validation Patterns (VVPs) to enable a systematic comparison of simulation results against a range of validation targets, such as benchmarks or measurements. We review a range of related research and development efforts in §2, and then provide a brief description of the toolkit as a whole in §3. We describe the key aspects of each component in the toolkit in §4 and present a range of exemplar applications in §5, while we discuss on the main performance and scalability aspects of the toolkit in §6. Lastly, we share our main conclusions in §7.

2. Related work

Several other toolkits share a subset of the added values that VECMAtk provides. In the area of VVUQ, a well-known toolkit is Design Analysis for Optimization and Terascale Applications (DAKOTA)[1] [10], which provides a suite of algorithms for optimization, UQ, parameter studies, and model calibration. DAKOTA is a powerful tool, but has a relatively steep learning curve due to the large number of tools available [11] and offers no way to coordinate resources across concurrent runs [12,13]. Similarly, there are other toolkits that help with UQ directly, such as UQTK [14] and UQLab.[2] One particularly efficient method to handle UQ for a range of applications is the multi-level Monte Carlo Method [15].

In the area of VVUQ using HPC, there are several other relevant tools. OpenTURNS [16] focuses on probabilistic modelling and uncertainty management, connects to HPC facilities, and provides calibration/Bayesian methods and a full set of interface to optimization solvers. Uranie leverages the ROOT [3] framework to support a wide range of uncertainty quantification (UQ) and sensitivity analyses (SA) activities using local and HPC resources. A key requirement for performing many types of UQ and SA is the ability to effectively run large ensembles of simulations runs. In addition to QCG-PJ, presented in this paper, there are tools such as RADICAL-Cybertools [17] that can be used to initiate and manage large simulation ensembles on peta and exascale supercomputers. In the area of surrogate modelling, GPM/SA [18,19] helps to create surrogate models, calibrate them to observations of the system and give predictions of the expected system response. There is also a portfolio of available solutions for rapidly processing user-defined experiments consisting of large numbers of relatively small tasks. The examples are Swift/T [20] and Parsl [21], both of which support execution of data-driven workflows.

Another range of relevant related tools include more statistically oriented approaches. For instance, Uncertainpy [22] is a UQ and SA library that supports quasi-Monte Carlo (QMC) and polynomial chaos expansions (PCE) methods. PSUADE [23] is a toolbox for UQ, SA and model calibration in non-intrusive ways [24], while DUE [25] assesses uncertain environmental variables, and generates realizations of uncertain data for use in uncertainty propagation analyses. PyMC3 [26] is a Python package for Bayesian statistical modelling and probabilistic machine learning which focuses on Markov Chain Monte Carlo approaches and variational fitting. Similarly, SimLab[4] offers global UQ-SA based on non-intrusive Monte Carlo methods.

[1]See https://dakota.sandia.gov.

[2]See https://www.uqlab.com.

[3]See http://root.cern.ch.

[4]See https://ec.europa.eu/jrc/en/samo/simlab.

UQLab [27] and SAFE [28] are Matlab-based tools that provide support for UQ (using e.g. PCE) and SA (using e.g. Sobol's method), respectively.

3. The VECMA toolkit (VECMAtk)

The main objectives of VECMAtk are to facilitate the implementation of (a) uncertainty quantification and sensitivity analysis patterns (UQPs) and (b) verification and validation patterns (VVPs). The UQP implementations automate routines for uncertainty quantification and sensitivity analysis, while the VVP implementations enable verification and validation procedures for high performance (multiscale) computing applications. Ye *et al.* present the concepts of UQP on an algorithmic level [29], while an algorithmic paper on VVPs is still in preparation. Both implementations support remote execution on petascale and emerging exascale resources, as well as execution on local machines. In addition, we support a range of existing VVUQ mechanisms and provide tools for users to facilitate the adoption of VVUQ in their applications. As we want our toolkit to be general purpose, taken up by users and flexible, we have four main factors that shape our development approach:

(i) the need to fit into existing applications with minimal modification effort,
(ii) the need to support any application domain,
(iii) the flexible and recombinable nature of the toolkit itself, and
(iv) the geographically distributed nature of the users and particularly the developers.

As a result, we position VECMAtk as a collection of elements that can be reused and recombined in different workflows, interlinked with stable interfaces, data formats and application programming interface (APIs), to facilitate VVUQ in any application. VECMAtk specifically allows users with (multiscale) applications to combine generic, lightweight patterns to create a system-specific VVUQ approach that covers all relevant space and time scales, including the scale-bridges between them. The exemplar applications (which we present in §5) map to all three of the established multiscale computing patterns: Extreme Scaling (ES), Replica Computing (RC) and Heterogeneous Multiscale Computing (HMC) [30].

(a) Release schedule and changes compared to the initial release

The VECMA toolkit is an open-source and open development project, meaning that the latest source and development notes can be accessed at any time. In terms of releases, we have adopted a schedule with minor and major releases. Minor releases are made publicly every three months, are advertised within the project, and have limited amount of additional documentation and examples. The first major release was made in June 2019; the ensuing two releases will be made in June 2020 and June 2021, and will be public and fully advertised. They are accompanied with extensive documentation, tutorials, examples, training events and dedicated uptake activities. In addition to these two release types, we provide informal periodic updates to the master code branches, documentation and the integration of the components.

Since our first major release in June 2019, all VECMAtk components have improved scalability, a range of new features and capabilities, extended and revised application tutorials, improved technical documentation, and a wide range of user-requested bugfixes. We present the fundamental improvements and changes in the development of each VECMAtk component since the first major release as part of appendix A.

4. Overview of the VECMAtk components

We present a graphical overview of the various components in VECMAtk in figure 1. VECMAtk contains four main components: *EasyVVUQ* [31] simplifies the implementation and use of VVUQ workflows with focus on large scale scenarios, *FabSim3* [32] helps to automate computational

Figure 1. Overview of the main components, and their role within the VECMA toolkit. (Online version in colour.)

research activities, *MUSCLE 3* [33] supports the coupling multiscale applications, and the *QCG tools*[5] facilitate advanced workflows, and the process of design and execution of applications, using HPC infrastructures. All these components can be flexibly combined with other VECMAtk and third party components to create diverse application workflows. In addition, no single component is essential to all applications: for example, FabSim3 and QCG-Client are to a limited degree interchangeable, and at least one application fully relies on a third party tool (the Mogp emulator[6]) instead of EasyVVUQ.

In this section, we summarize each of the components that comprise VECMAtk, while we refer to the VECMAtk website https://www.vecma-toolkit.eu/ for more detailed technical information.

(a) EasyVVUQ

EasyVVUQ [31,34] is a Python library designed to facilitate implementation of advanced non-intrusive VVUQ techniques, with a particular focus on high performance computing, middleware agnosticism, along with single-scale and multiscale modelling. Generally, non-intrusive VVUQ workflows involve sampling the parameter space of the simulation. This usually means running the simulation multiple times. This process is tedious and error prone if done in an *ad hoc* manner, and further exacerbated if the parameter space is large with a corresponding large number of runs (e.g. of order thousands to millions of runs), or if each simulation is very computationally expensive. EasyVVUQ allows one to coordinate the whole process—from sample generation through execution to analysis—although the execution stage is external to EasyVVUQ, and can be done by tools such as QCG-PilotJob, FabSim3 or Dask, or even manually. EasyVVUQ seeks to be agnostic to the choice of execution middleware, due to the large range of possible execution patterns that may be required in an HPC workflow. Additionally, the library carefully tracks and logs applications of the sampling elements along the way, allowing a reasonable level of restartability and failure resistance.

EasyVVUQ breaks down the VVUQ process into five distinct stages.

(i) Application description, which further can be divided into the following items:
   (a) Encoder: responsible for producing input files for the simulation.
   (b) Decoder: responsible for parsing the output files of the simulation and extracting the needed values.
   (c) Execution action: describes how the simulation is run.

[5]See https://www.qoscosgrid.org.

[6]See https://github.com/alan-turing-institute/mogp_emulator.

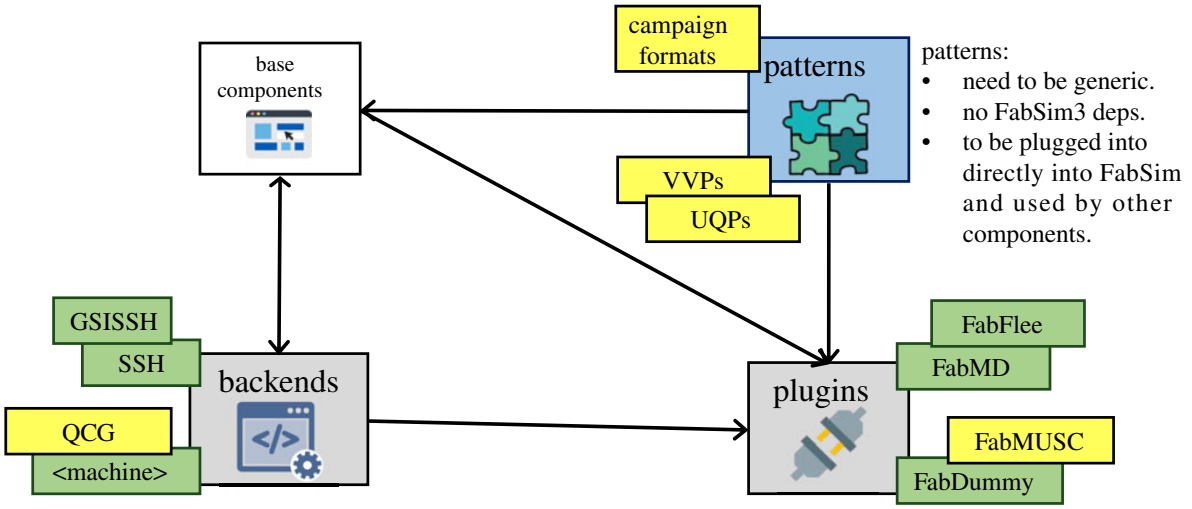


Figure 2. High-level overview of the FabSim3 architecture, showcasing a few of the key plugins, patterns and back-end functionalities available in the tool. Boxes in green (darker) are completed, while boxes in yellow (lighter) are working, but subject to further extensions and improvements. (Online version in colour.)

(ii) Sampling: this step is very dependent on the VVUQ technique in question. The main task of sampling components is to produce a list of parameter sets for the simulation.
(iii) Execution: this is handled entirely externally to EasyVVUQ.
(iv) Collation: collects the outputs of the simulations and then stores them in the EasyVVUQ database, to be retrieved later for analysis.
(v) Analysis: perform statistical analysis on the collected data. This analysis can then inform further actions such as collecting more samples.

EasyVVUQ currently has support for variance-driven sensitivity analysis, stochastic collocation and polynomial chaos expansion methods, as well as more basic statistical analysis methods such as bootstrapping. Additionally, all components of EasyVVUQ are easily extensible, allowing users to customize every stage of the VVUQ workflow to suit their needs. Nevertheless, for many applications the built-in components provide out-of-the-box functionality, requiring the user only to define input and output formats and perform analysis.

(b) FabSim3

FabSim3[7] [32] is a Python-based toolkit for automating complex computational research activities which has several aims: First it helps to automate and simplify the creation, management, execution and modification of complex application workflows. To do so, FabSim3 supports functionalities such as ensemble runs, remote executions, iterative processing and code couplings. Second, it aims to help researchers become quicker and more systematic in handling complex applications in particular. To support this, FabSim3 is designed to be easy to customize for use on new machines and with new simulation codes, and automatically curates a wide range of environment and state variables to aid the testing and debugging of application runs. As usage examples, researchers may want to run and rerun static configurations, run a range of slightly different workflows, define new types of complex applications altogether or construct a routine to automatically validate their code.

FabSim3 also supports a range of adaptable mechanisms for code coupling, and domain-agnostic code patterns that provide building blocks for enabling VVUQ in new applications. We provide an overview of the FabSim3 architecture in figure 2.

In the context of VECMAtk, FabSim3 plays a key role in introducing application-specific information in the Execution Layer, enabling users to combine different UQPs and VVPs, and

[7]See https://github.com/djgroen/FabSim3.

providing an approach to curate large sets of production runs. Also, FabSim3 can use QCG-Client or QCG-SDK to submit ensemble runs via a 'pilot job' mechanism [1,35] (see §4d(ii)), which boosts efficiency by starting and managing sub jobs without each of them needing to individually wait for resources to become available.

(c) MUSCLE 3

The third version of the Multiscale Coupling Library and Environment [33,36] (MUSCLE 3) is intended to simplify the coupling of temporally or spatially scale-separated submodels into a single (distributed) simulation. At run-time, the submodels are started in parallel, and exchange information via the network. Submodels and other simulation components are unaware of each other as MUSCLE 3 delivers the messages to receivers given in a configuration file. The Multiscale Modelling and Simulation Framework (MMSF, [37,38]) may be used to determine how the submodels should be coupled given their relative spatial and temporal scales.

MUSCLE 3 supports time-scale separated macro-micro couplings, including automatically re-running the micro model as often as needed to match the macro model's time steps. It also supports space scale separation, which entails coupling a single macro model to a set of micro models, via support of sets of multiple submodel instances and special vector ports that can be used to communicate with them. Model settings (parameters and technical configuration) are put in the single configuration file, and are transmitted to the individual instances automatically. Components may be inserted into the simulation that generate a parameter overlay, which combined with vector ports and sets of instances yields an ensemble, for example, UQ. MUSCLE 3 supports a range of UQPs, including those for semi-intrusive UQ [29,39]. With MUSCLE 3, this can be done by changing the configuration file to add a few more components and modifying the connections; the submodels can remain unaltered.

(d) QCG tools

The QCG suite of tools help to facilitate advanced workflows, and the process of design and execution of applications, using HPC infrastructures. Four specific QCG components form part of VECMAtk: QCG-Client, QCG-PJ, EQI and QCG-Now, which we now describe in turn.

(i) QCG-Client

QCG-Client is a command line tool to access remote computing infrastructure. The tool allows submission of different types of jobs, including complex workflows, parameter sweep tasks and array jobs, on single or multiple clusters. It uses the QCG-Broker service to manage the execution of workflows, e.g. through multi-criteria selection of resources. QCG-Client can be deployed as a container and is in this form integrated with FabSim3. This allows users to select from both command-line tools, QCG-Client and FabSim3, when they access QCG services.

(ii) QCG Pilot Job

To perform UQ procedures, we must be able to flexibly and efficiently execute a large number of simulations. This is because we need a large and dynamically created parameter-space, which in turn feeds into an ensemble of model executions to get statistically correct results. Within VECMAtk, we use QCG-PilotJob (QCG-PJ) by default to fulfil this requirement, primarily because it can be installed automatically by users without admin rights, and because of its scalability and limited dependency footprint. For instance, QCG-PJ can be set up by users without requiring other components from the QCG environment.

QCG-PJ is designed to schedule and execute many small jobs inside one scheduling system allocation on an HPC resource. Direct submission of a large group of jobs to a scheduling system can result in a long completion time as each single job is scheduled independently and waits in a queue. In addition, the submission of a group of jobs is often restricted (and sometimes even

prohibited) by system administrators. Some systems do support bespoke job array mechanisms, but these mechanisms normally only allow jobs with identical resource requirements. Jobs that are part of a multiscale simulation or application workflow by nature vary in requirements and therefore need more flexible solutions.

To use QCG-PJ, a user submits a job with a QCG-PJ Manager instance, which is executed like a regular job. However, unlike regular jobs this job internally manages a user-defined workflow consisting of many sub-jobs. The manager executes commands to submit jobs, cancel jobs and report resources usage; it may either be preset or listen dynamically to requests from the user. QCG-PJ then manages the resources and jobs in the system, taking into account resource availability and mutual dependencies between jobs. Users can interact with QCG-PJ either using a file-based or network-based interface. The file-based approach works well for static scenarios where the number of jobs is known in advance, while the network interface allows users to dynamically send new requests and track execution of previously submitted jobs at run-time. Although intended for HPC resources, QCG-PJ Manager also supports a local execution mode to allow users to test their scenarios on their own machines (with requiring a scheduling system allocation).

(iii) EQI

Many workflows involving EasyVVUQ require a large number of jobs to be executed, and EasyVVUQ delegates this responsibility to other tools (either existing, or user created). QCG-PJ is the main tool available in VECMAtk to fulfil that purpose, and it offers a flexible but generic API that is less appropriate for specific use cases. To provide an API that is specifically adjusted to EasyVVUQ, we have developed EQI, which is an acronym for the (E)asyVVUQ-(Q)CG-PJ (I)ntegration API. This API introduces domain-oriented concepts, such as predefined tasks for execution and encoding, that facilitate the easy integration of EasyVVUQ workflows with QCG-PJ. With the API, users can perform VVUQ computations with QCG-PJ, invoking HPC resources. The development life-cycle of EQI is synchronized with the releases of both EasyVVUQ and QCG-PJ, to ensure a persistently up-to-date API.

(iv) QCG-Now

Since the traditional command-line interfaces may be perceived by many users to be too complicated, within VECMAtk we decided to provision a easy-to-use graphical tool for users. QCG-Now is graphical desktop programme that enables the submission and management of jobs on HPC resources. It also supports automatic notifications, data sending and receiving. Since the first release of QCG-Now last year, several improvements have been made: for instance it is now integrated with the QCG-Monitoring system which allows users to track progress of execution of their tasks directly from its graphical interface. This capability is particularly important for VECMAtk users who want to keep track of the progress of long-lasting executions of EasyVVUQ and QCG-PJ workflows. Another new feature allows users to store frequently used execution schemes as quick templates for easy reuse. We present a graphical demonstration of the tool in figure 3.

(e) VECMAtk workflows

In figure 4, we show the main components and a few examples as to how they have been combined, leveraging added values where relevant while maintaining a limited deployment footprint. These components can be combined, but also integrated with third party components.

Up to now, the following combinations have been particularly common:

Figure 3. QCG-Now with the embedded QCG-Monitoring view. The integration with QCG-Monitoring allows users to track progress of execution of applications in a graphical way, for example presenting dynamically updated tables, images or, as in this screenshot, charts. (Online version in colour.)

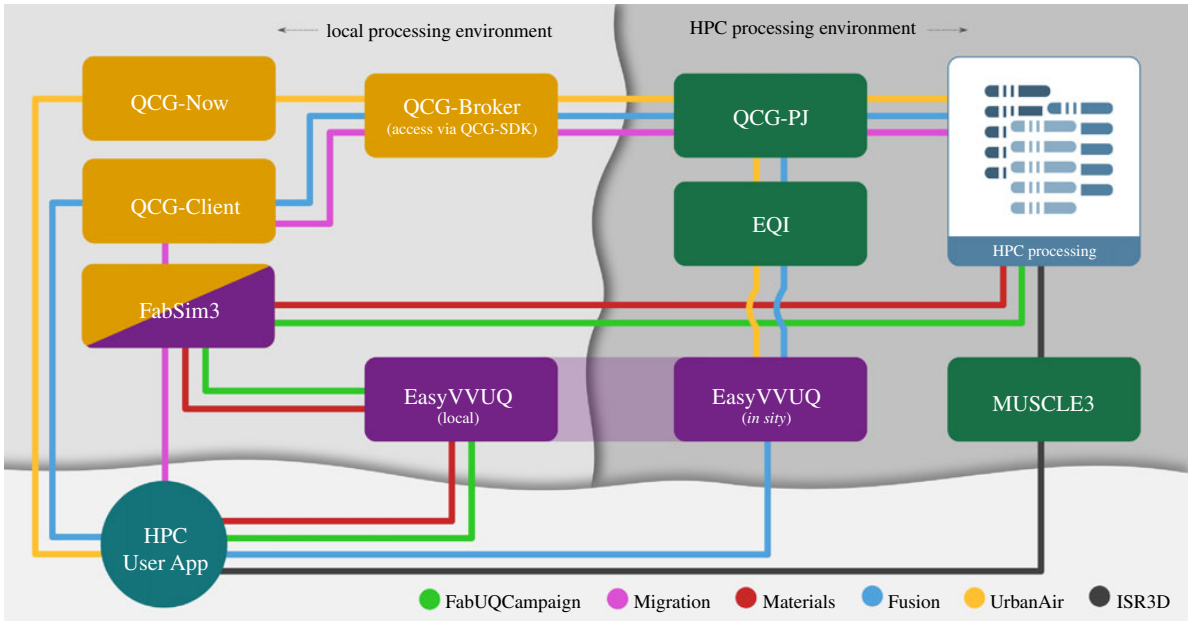


Figure 4. 'Tube map' showing which VECMAtk components are used in the various exemplar applications. VECMAtk components are given in boxes, and the application tutorials are indicated using coloured lines. Note that code using the EasyVVUQ [34] library may be located either on the local desktop for ease of use or on a remote HPC resource for improved performance. Source: https://www.vecma-toolkit.eu/toolkit/. (Online version in colour.)

— FabSim3 + QCG-Client: enables users to submit their job to QCG-Broker to schedule them across multiple remote (QCG-supporting) machines. Additionally, FabSim3 tool support other job scheduling system such as SLURM,[8] Portable Batch System (PBS).[9]

[8]See https://slurm.schedmd.com/overview.html.

— FabSim3 + EasyVVUQ: enables users to automate the VVUQ processes into their applications workflow. After creating their EasyVVUQ campaigns, users can convert them to FabSim3 ensembles using a one-liner (`campaign2ensemble`) command and submit the ensemble jobs through FabSim3. Then they convert results back to EasyVVUQ using the `ensemble2campaign` command, where it is decoded and further analysed.
— FabSim3 + QCG Pilot Job: enables users to create and manage pilot jobs using FabSim3 automation.
— EasyVVUQ + QCG Pilot Job + EQI: enables EasyVVUQ users to execute their tasks directly using pilot jobs.

There are also several examples of integrations with third party components. For instance, there is an application example that uses the mogp emulator in place of EasyVVUQ,[10] and a working EasyVVUQ example that relies on Dask instead of QCG-PilotJob[11] for ensemble job submission. Other integration are indeed possible as well, and the inclusion of a different job management and execution engine may be beneficial for individual applications and/or resources. To name a few examples, RADICAL-Cybertools can offer exceptional performance on machines where it is set up [17], Swift/T [20] can facilitate complex data flows with minimal serialization while Parsl [21] is optimized for the fast and flexible execution of Python programs and Jupyter notebooks.

As indicated in figure 1 earlier, there are four components that provide a suitable entry point for new users to the toolkit. EasyVVUQ facilitates users that initially focus on incorporating VVUQ in their simulations. FabSim3 is aimed at users that initially focus on enabling complex and curated workflows, which could involve ensemble or dynamic execution. QCG-Now is intended for users that prefer a graphical interface for managing their simulations and VVUQ workflows. And MUSCLE3 is well suited for users whose primary concern is multiscale code coupling.

5. Exemplar applications

With VECMAtk, we aim to provide a toolkit that can be applicable to any domain/application where UQ and SA are required, and computational modelling and simulation can be applied. To achieve this aim, we rely on an increasing number of leading-edge scientific applications to support the testing and development of the toolkit. Within this paper, we present a selection of these applications which cover the following topics: (a) future energy sources, (b) human migration, (c) climate, (d) advanced materials, (e) urban air pollution and (f) biomedicine and (g) epidemiology. For each application we show how the toolkit provides added value in terms of enabling UQ. In addition, we describe how we use Verification and Validation Patterns (VVPs) to help facilitate V&V for the first two applications. In each of these concise application descriptions, we focus on how VECMAtk is being used by a range of different applications, which components are primarily employed/invoked, and what primary benefits the toolkit delivers for the application users.

(a) Fusion

Thermonuclear fusion has the potential of becoming a new source of energy that is carbon-free. The dynamics of fusion plasmas span a wide range of scales in both time and space, as, for example, micro-instabilities formed in plasma turbulence can interrupt the overall macroscopic transport and destroy confinement. To simulate such phenomena, we have built a multiscale fusion workflow that includes equilibrium, turbulence and transport physics to model the plasma dynamics [40].

[9]See https://www.pbspro.org/.

[10]See https://github.com/alan-turing-institute/mogp_emulator.

[11]See https://easyvvuq.readthedocs.io/en/dev/dask_tutorial.html.

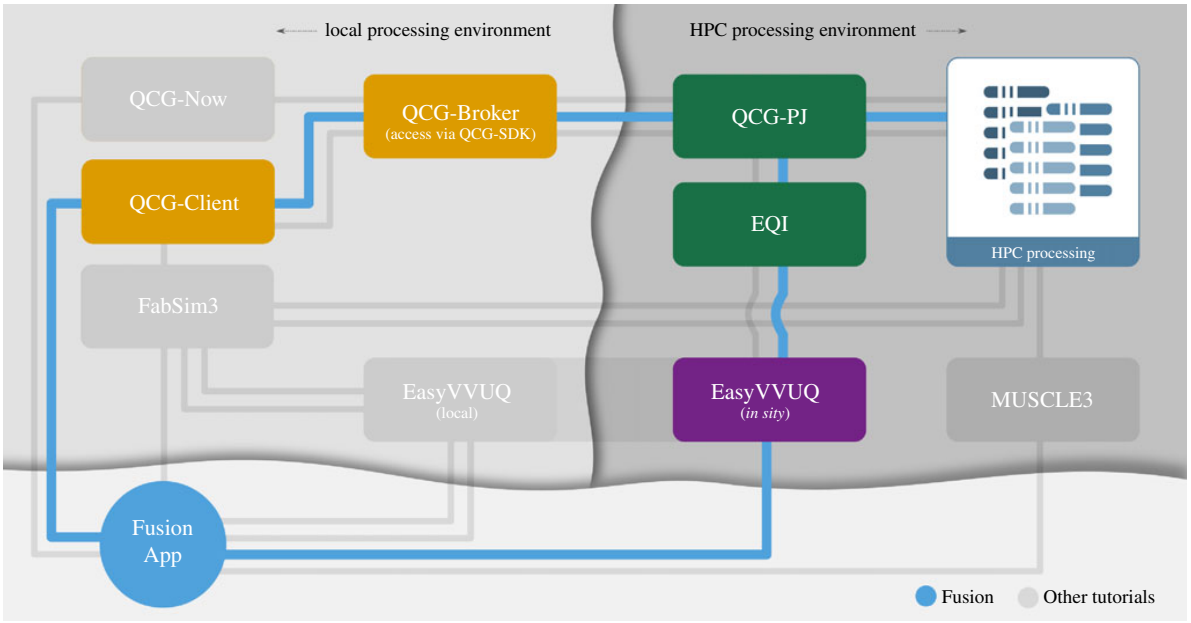


Figure 5. 'Tube map' showing which VECMAtk components are used by the Fusion application. (Online version in colour.)

Our central purpose is to ensure that the results are reproducible and can become a reliable representation of the experiment measurements. To achieve this, it is key to use VVUQ approaches.

We integrated UQ methods using the EasyVVUQ library and performed a parallel execution of samples in a single batch allocation with QCG-PJ through the EQI. Those tools, available as part of the VECMA toolkit (figure 5) and in conjunction with the coupled workflow nature of the fusion application, have proven to be simple to use and to allow a large variety of experiments with different methods and models (see [41] for further details). We also performed a validation using the `ValidationSimilarity` VVP, which is implemented in EasyVVUQ and help us find similarities between the probability distributions of the quantities of interest in our simulation results and in the experimental measurements [42]. For this purpose, we can choose between several relevant metrics, such as Hellinger distance [43], Jensen–Shannon distance which is a symmetrized and smoothed version of the Kullback–Leibler divergence [44] and Wasserstein metrics [45].

(b) Forced human migration

Forced migration reached record levels in 2019, and forecasting it is challenging as many forced population datasets are small and incomplete, and armed conflicts are often unpredictable in nature [46]. Nevertheless, forced migration predictions are essential to improve the allocation of humanitarian support by governments and NGOs, to investigate the effects of policy decisions and to help complete incomplete data collections on forced population movements. Through the use of the FLEE[12] agent-based simulation code, we model and forecast the distribution of incoming refugees across destination camps for African countries [47].

The VECMA toolkit provides us with the ability to automatically construct, execute and analyse ensemble simulations of refugee movements [48], to efficiently compute the sensitivities of key parameters in our simulation [4], to couple different types of models to form multiscale workflows [49] and to facilitate the rapid execution of large job ensembles on supercomputers. We also use a VVP named `EnsembleValidation` to systematically validate our simulations against data from a range of conflicts (see e.g. here [50]). This VVP performs a validation on the output directory of each run and uses an aggregation function to combine all output validation results into a combined metric.

[12]See https://github.com/djgroen/flee-release.

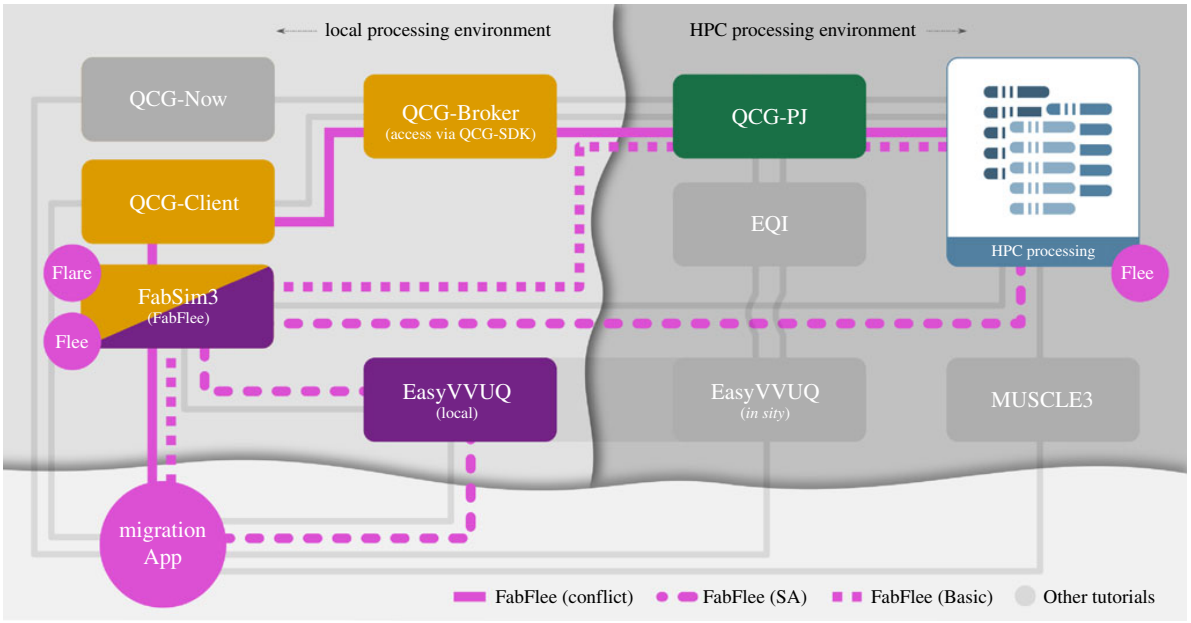


Figure 6. Tube Map showing which VECMAtk components are used in the FabFlee plugin. VECMAtk components are given in boxes which enable users to include their added values while retaining a limited deployment footprint. The black and grey lines define the FabFlee path using VECMAtk components for execution. (Online version in colour.)

We summarize our main application workflows in figure 6. We rely on the FabFlee plugin in FabSim3 to automate the simulation activities required to analyse different policy decisions, to provide the VVP functionality and to facilitate coupling with other models and data sources. Examples include an acyclic coupling to a conflict model [49], but we are also working on a cyclic macro–micro model for the South Sudan conflict. We also use EasyVVUQ with FabFlee to perform sensitivity analyses for varying agent awareness levels, speed limits of refugee movements, location move chances and other simulation parameters [4]. Lastly, we use QCG Pilot Job to facilitate the very large number of runs required to do this analysis, which is required for our more advanced workflows that so far involved up to 16 384 jobs.

(c) Climate

For climate modelling, the range of space and time scales present in (geophysical) turbulent flow problems poses challenges, as this range is too large to be fully resolved in a numerical simulation. As such, micro-scale effects on the resolved macroscopic scales must be taken into account by empirical parametrizations (e.g. [51]). These parametrizations often involve a number of coefficients, for which the value is only known in an approximate manner. In addition, they are subject to the so-called model error or model-form uncertainty, which is the uncertainty introduced due to the assumptions made in the mathematical form of the parametrization scheme.

Within VECMA, we focus on the uncertainty in time-averaged climate statistics of geophysical flow models, due to various assumptions made in the parametrizations. EasyVVUQ allows us to assess the uncertainty in the output statistics due to the imperfectly known coefficients. In addition, we currently use EasyVVUQ for uncertainty quantification of an established local atmospheric model, DALES [52,53], which is used to simulate atmospheric processes including turbulence, convection, clouds and rain. Here, we quantify the uncertainty in model output from different sources, including uncertain physical input parameters, model choices and numerical settings, and the stochastic nature of turbulence modelling. To run the EasyVVUQ ensemble of simulations on HPC resources we use FabSim3, see figure 7. To address the more fundamental model-form uncertainty, we are currently investigating the use of data-driven stochastic surrogates to replace traditional deterministic parametrizations; see e.g. [55,56] for recent results.

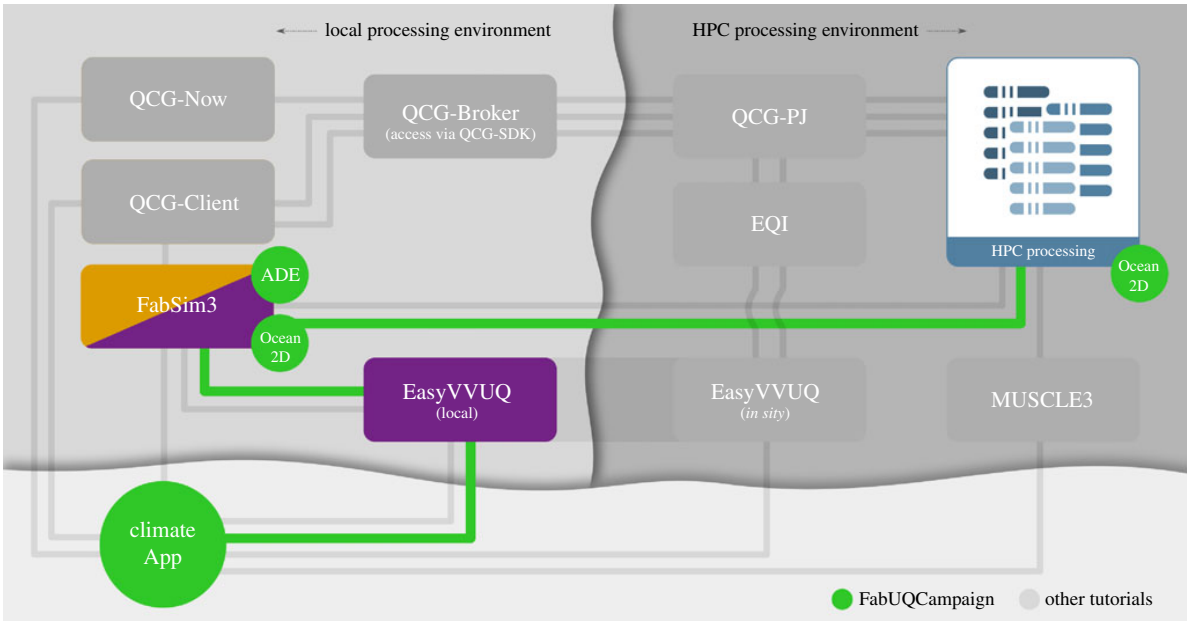


Figure 7. A Tube Map showing the VECMA components used in the climate application. Here 'FabUQCampaign' is a plugin used to run an ensemble of EasyVVUQ samples on HPC resources. Furthermore, 'ADE' and 'Ocean2D' are two example applications (advection diffusion equation and a two-dimensional ocean model) for which tutorials can be found in [54]. (Online version in colour.)

(d) Advanced materials

Developing novel advanced materials can be a very expensive and time consuming process, often taking over 20 years from initial discovery to application.[13] Chemical and material modelling techniques are well developed for single scales, but engineering applications require understanding across many scales [57–59]. Making actionable predictions with these modelling techniques, to improve on the laborious experimental development process, will require harnessing multiple simulation techniques and providing tight error bars on their predictions.

Using VECMAtk, one can generate, execute, collate and analyse ensemble simulations of mechanical tests in an automatic fashion. This allows us to effectively gather statistics on a system of interest and explore an input parameter space with minimal human oversight.

To do this, we use EasyVVUQ to explore an input parameter space by generating ensembles of simulations and to collate the results into an easily analysable object. To distribute the large number of simulations to be run in each ensemble, we use FabSim3 to automate the submission of jobs on remote HPC resources.

(e) Urban air pollution

Predicting air quality in complex urban areas is a challenging field where researchers need to balance accuracy against a practical turnaround time. There are numerous models for predicting contamination transport and dispersion, ranging from simple Gaussian models (that are fast and cheap but not necessarily accurate) to computational fluid dynamics simulations that are slower, more costly and potentially very accurate.

In all cases the most important, and often missing part, is an accurate emission database that contains pollutant emission rates for different types of sources: point (e.g. industrial chimneys), line (e.g. road transportation) and areas (e.g. house heat appliances). The pollutants we consider are NO_2/NO_x, SO_2 and two types of particulate matter (also known as floating dust): $PM_{2.5}$ for particles 2.5 µm or less in diameter and PM_{10} for particles 10 µm or less in diameter. We

[13]See https://www.mgi.gov/.

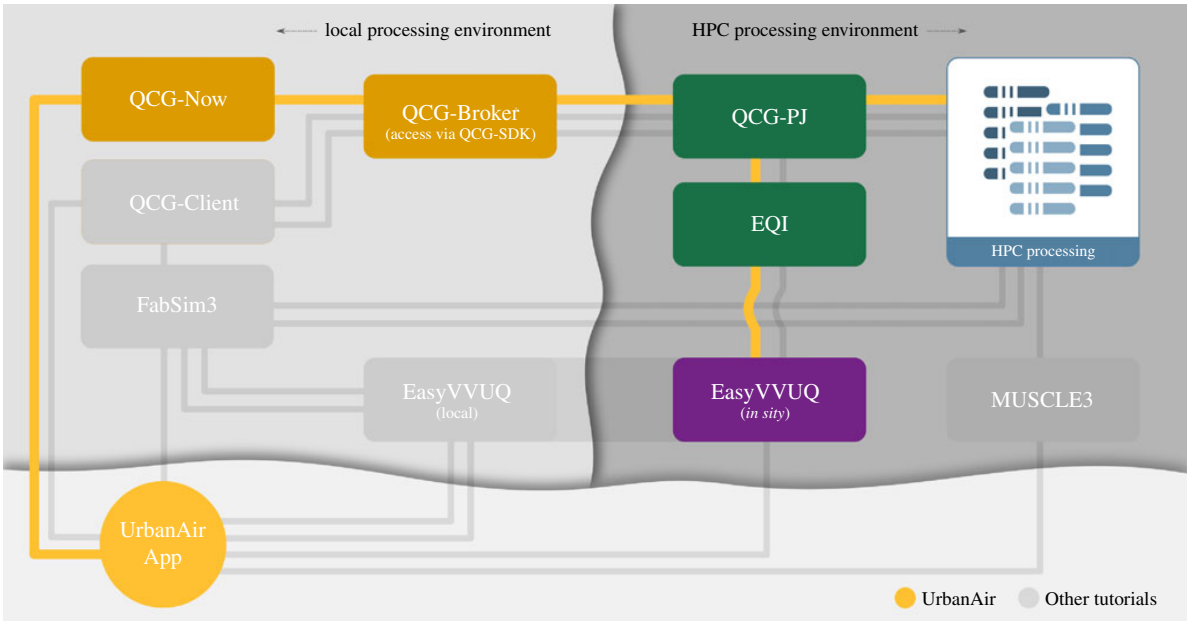


Figure 8. 'Tube map' showing which VECMAtk components are used by the UrbanAir application. (Online version in colour.)

use a mesoscale weather prediction model WRF [60] and an all-scale geophysical flow solver EULAG [61] in our multiscale simulation [62] to address uncertainties coming from incomplete emission database, and to provide a quantitative air quality prediction [63].

We use VECMAtk to automate sampling, ensemble generation, execution of simulation codes on HPC machines, collating results and post-execution analysis. This automation simplifies the provision of better prediction results, because we analyse many different initial conditions and can automatically provide mean results (or some weighted ones) from all simulations. We are also able to run sensitivity analysis of the input parameters to identify and select the most important ones, thus limiting the number of required ensembles for future runs.

In terms of tools, we rely on EasyVVUQ, coupled with QCG-PJ using EQI. Using Python we define the parameter space to be sampled, application to be run in HPC environment, hardware requirements (e.g. number of nodes and cores), and the parameters to be analysed after simulations ended. We then use QCG client and QCG-PJ to submit and control jobs on the HPC resources, EasyVVUQ to generate the samples, EQI for enabling the samples to be executed with QCG-PJ and EasyVVUQ to analyse the results. We summarize our workflow in figure 8.

(f) Biomedicine

In-stent restenosis (ISR) is the renewed occurrence of arterial stenosis (narrowing of an artery) after it was initially treated by installing a metal stent, as a result of excessive cell growth. The results of uncertainty and sensitivity analysis for a two-dimensional model of in-stent restenosis (ISR2D) are presented in [64]. Improved effectiveness of uncertainty propagation was achieved by applying a semi-intrusive metamodelling method and the results are demonstrated in [65] and in [66].

Current research on the design of the UQ and SA experiments for the three-dimensional model of the ISR model (ISR3D) is ongoing. The ISR3D model is implemented using MUSCLE 3 discussed in §4c, which allows performing more advanced semi-intrusive algorithms. In particular, we plan to train a metamodel on the fly and combine that with a cross-validation test of the effect of the approximation error on the results of the micro model.

(g) Coronavirus modelling

Since 2019, the world has been severely affected by the spread of the SARS-CoV-2 coronavirus, and the COVID-19 disease it causes. Although many models are able to approximate the viral

spread on the national level, few solutions are available to forecast the spread on a local level, e.g. in towns or city boroughs. Within the HiDALGO project[14] we are working on the Flu And Coronavirus Simulator (FACS)[15] to help address this challenge.

We use VECMAtk primarily to systematically validate the code, analyse parameter sensitivities, execute ensemble simulations to account for aleatory uncertainty in the code, and to easily produce ensemble forecasts which involve a wide range of scenarios, applied to a range of boroughs in London.

To achieve this, we primarily rely on FabCovid19, which is a FabSim3 plugin, as well as QCG-PJ. Both tools combined allow us to automate these tasks, and seamlessly create, run and post-process ensemble simulations. We also use EasyVVUQ to analyse parameter sensitivities. To demonstrate in what way the VECMA toolkit adds value to our application in a concrete way, we present an small example sensitivity analysis study in appendix B.

6. Toolkit performance and scalability

The VECMA toolkit has been developed for use with large supercomputers, and therefore needs to be both fast and scalable. In this section, we present a range of key performance aspects of the toolkit, and discuss the advantages and limitations of VECMAtk in terms of scalability and exascale readiness.

The main performance-critical aspects of the toolkit involve:

(a) The sampling of parameter values and creation of large numbers of simulation inputs.
(b) The submission, execution and retrieval of large ensembles of simulation jobs.
(c) The efficient movement of data between local and remote resources.
(d) The efficient movement of data between coupled models.

We will now briefly discuss the performance-related characteristics of VECMAtk in relation to these four potential bottlenecks:

Sampling parameter values and creating large numbers of simulation inputs. There are multiple ways in which EasyVVUQ helps with the issues involved in sampling many and expensive simulation outputs. First of all, EasyVVUQ uses an SQL backend database that can handle 10000s of samples or more. It also tracks the status of job execution, allowing users to cope with hardware failures by restarting failed jobs without having to rerun successful ones. Furthermore, EasyVVUQ allows one to sample in stages—appending more samples if results are not yet sufficiently robust, without losing the results already in the database. Lastly, EasyVVUQ can delegate the ensemble job submission to other scalable components, such as Dask or QCG-PJ. These components then execute part or all of the simulations on HPC resources.

Submission and execution management of large ensembles of simulation jobs. Many scientific applications need to run large ensemble simulations (1000+ runs) to perform UQ and SA, which cannot be executed as individual jobs on most supercomputers due to scheduler constraints, and require a pilot job mechanism such as QCG-PJ. QCG-PJ has been shown to efficiently execute 10000 jobs with less than 10% overhead, even if those jobs only last for one second each.[16]

To improve the FabSim3 ensemble submission performance, we integrated it with QCG-PJ and enabled multi-threaded job submission from the local machine. To demonstrate the benefit of this, we measured the total elapsed time of the job submission to a remote machine for the epidemiology[17] application. Owing to limitation of maximum number of submitted job per user[18]

[14]See http://hidalgo-project.eu.

[15]See https://facs.readthedocs.io.

[16]See VECMA Deliverable 5.2: https://www.vecma.eu/wp-content/uploads/2019/12/VECMA_D5.2_First-Report-Infrastructure_PSNC_20191208.pdf.

[17]See https://github.com/djgroen/FabCovid19.

[18]The maximum number of jobs a user can have running and pending at a given time is 5000. If this limit is reached, new submission requests will be denied until existing jobs in this association complete.

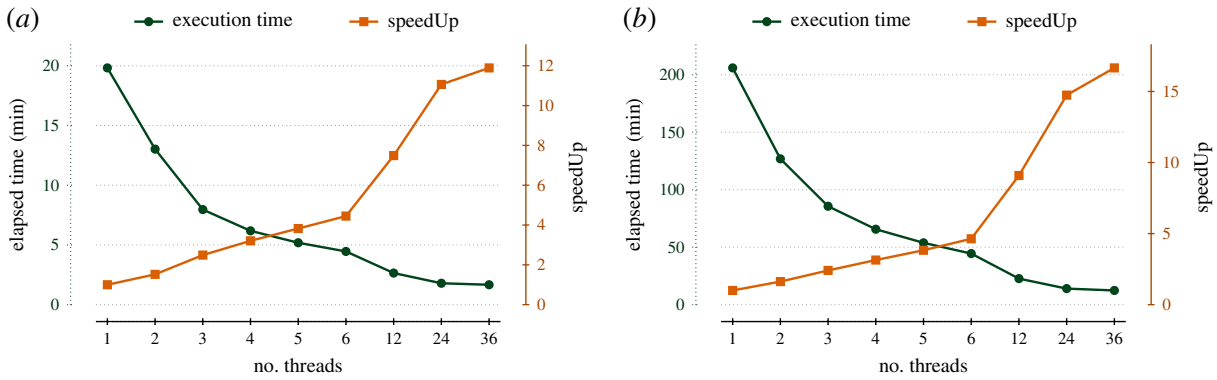


Figure 9. Time required to submit 15121/4865 jobs with FabSim3 (with/without QCG-PJ) relative to the number of job submission threads used. Graph is made using average of 10 repetition of each ensemble size. Please note that, here we only measure the job submission overhead, so, queuing time and job execution on computing nodes are not considered in our test. (*a*) ensemble size = 15121, QCG-PilotJob = True, (*b*) ensemble size = 4865, QCG-PilotJob = False. (Online version in colour.)

Table 1. Comparison of the three main ensemble job submission approaches in VECMA, in terms of general usability, whether files are staged to and from the remote resource as part of the approach, the need for remote deployment work and the overall performance.

approach	usability	remote file stage?	remote deployment?	submission overhead	
				per 100 jobs	per 1000 jobs
FabSim3 only	excellent	yes	not needed	40–90 s	not attempted
FabSim3 + QCG-PJ	excellent	yes	user-space only	33–36 s	250–300 s
QCG-PJ only	good	no	user-space only	<5 s	40 s

on the Eagle supercomputer, we use ensembles sizes of 4865 and 15 121 runs, and use up to 36 threads for the submission process. We enabled pilot jobs for the larger ensemble, but not for the smaller ensemble. We present the results of this scalability test in figure 9.

For FabSim3 ensembles without the use of QCG-PJ, we find that the job submission overhead is 2-3 seconds per job when using a single thread, but decreases to less than a second per job when using four or more threads. When using FabSim3 with QCG-PJ, the job submission overhead reduces by a factor 2 for ensembles with 100 jobs to about 0.35 s per job. The overhead further diminishes for larger ensembles, as we measure an overhead of about 0.275 s per job for ensembles with 1000 jobs. In table 1, we compare the three main ensemble submission approaches in VECMA in terms of performance, usability and in regards to the need of any remote deployment.

Efficient movement of data between local and remote resources. Large simulations, as well as large ensembles of smaller simulations, are often accompanied by the production of large amounts of complex data. Within the VECMA project, we find that the organizing and re-organizing of these data are a cognitively intensive task that is prone to human error if not automated. Both the FabSim3 and EasyVVUQ tools provide a range of data structure conventions that transparently automate low-level data management aspects, allowing users to focus on the complexities that occur at higher levels. For instance, FabSim3 automatically curates input and output and while EasyVVUQ automates the encoding and decoding of simulation files. In addition, FabSim3 and QCG-Client automatically perform file staging to and from HPC resources.

In terms of performance, we chose to focus less on optimizing file transfer rates (although FabSim3 and QCG-Client do support GridFTP), and more on limiting the number of file transfers altogether. The clearest example of this optimization is the tight integration of EasyVVUQ directly with QCG-PJ. Using EQI, users can use HPC resources to generate, run and analyse their

simulation ensembles for VVUQ purposes, and do this iteratively and dynamically within a single Python script. In this way, all the computations are brought to where the data are, and input files only need to be staged in once in advance, even if the workflow contains a large number of (dynamic) iterations.

The efficient movement of data between coupled models. MUSCLE 3 is primarily positioned to address this bottleneck, and provides added value for instance by eliminating the need for file I/O in the coupling.

7. Conclusion

In this paper, we have presented the VECMA toolkit for the verification, validation and uncertainty quantification (VVUQ) of single and multiscale applications. As showcased by the wide range of applications that use the toolkit already, VECMAtk is unique in its ability to combine a wide range of VVUQ procedures with a streamlined automation approach, and trivially accessible capabilities to execute large job ensembles on pre-exascale resources. In addition, VECMAtk components can be flexibly combined, allowing users to take advantage of parts of the toolkit while retaining a very limited deployment footprint.

As part of this paper, we have described a number of exemplar applications from a diverse range of scientific domains (fusion, forced migration, climate, advanced materials, urban air pollution, biomedical simulation and coronavirus modelling), all used by researchers today. All these examples are open source and accompanied with tutorials on http://www.vecma-toolkit.eu/, allowing new users to exploit them as building blocks for their own applications.

Through the VECMA toolkit, we aim to make VVUQ certification a standard practice and to make it simpler to quantify uncertainties, parameter sensitivities and errors in scientific simulations. VECMAtk is used to establish these procedures irrespective of the application domain, allowing key VVUQ algorithms to be reused across disciplines. Because VVUQ is essential to ensure that the key simulation results are indeed actionable, the VECMA toolkit is an important tool to help ensure that scientific simulations can be responsibly used to inform decision-making.

Data accessibility. This article has no additional data.

Competing interests. We declare we have no competing interests.

Funding. This work was supported by the VECMA project, which has received funding from the European Union Horizon 2020 research and innovation programme under grant agreement no. 800925. The development of MUSCLE3 and its respective description was supported by The Netherlands eScience Center and NWO under grant no. 27015G01 (e-MUSC project). The development of the migration and coronavirus modelling applications was supported by the EU-funded HiDALGO project (grant agreement no. 824115). The calculations were performed in the Poznan Supercomputing and Networking Center.

Appendix A. Improvements since first VECMAtk release

In table 2, we present the fundamental improvements and changes in the development of each VECMAtk component since the first major release.

Appendix B. Demonstration of a single example VVUQ analysis: COVID-19 application

To demonstrate the added value offered by VECMAtk more concretely, we showcase one specific VVUQ procedure example, using the Flu And Coronavirus Simulator. We perform sensitivity analysis across six different input parameters of FACS[19] to identify their sensitivity relative to our quantity of interest (QoI). In table 3, we provide the default value for each parameter along

[19]See https://facs.readthedocs.io.

Table 2. The list of modifications and enhancements in the development of each VECMAtk component since the first annual release in June 2019.

VECMAtk Component	list of improvements and changes in development
FabSim3	— added support for multi-threaded job management
	— added automated installation and configuration of FabSim3 on different OS
	— fixed synchronizing/copying for a large number of files and folders
	— vastly improved the performance of `run_ensemble`
	— revamped the documentation
EasyVVUQ	— added support for vector-valued quantities of interest
	— developed an extensive unit testing suite
	— added support for Dask[a] as an execution back-end
	— improved sparse-grid Stochastic Collocation sampler
	— dimension-adaptive Stochastic Collocation sampler
QCG Pilot Job	— developed the node agent for more efficient launching of jobs across many nodes
	— added more optimal way of handling iterative jobs
	— increased the test base and test code coverage
	— simplified execution of bash scripts as Pilot Job's tasks
	— bugfixes for execution of multi-node Pilot Jobs and OMP jobs
MUSCLE3	— newly added; contributed by the e-MUSC project
	— submodel coupling for spatial and/or temporal scale separation
	— submodel instance sets and ensembles
	— messages may contain grids, lists, dictionaries and basic types
	— support for Python, C++, Fortran and MPI, with tutorials and API documentation
	— automatic building of its dependencies on a variety of platforms
EasyVVUQ-QCGPilotJob	— provided a new API that simplifies the execution of EasyVVUQ scenarios using the QCG Pilot Job system
	— introduced a new mechanism to support custom encoders usage
	— made available a detailed tutorial that can be helpful for new users
QCG-Now	— developed an internal view for monitoring of applications (integration with QCG-Monitoring)
	— provided quick-template functionality for frequently submitted tasks

[a]See https://dask.org.

with the range of likely values. We use the Chaospy library[20] in EasyVVUQ to generate samples from the input parameters. Specifically, in this example we used the stochastic collocation method with a sparse-grid sampling plan of 15 121 samples, which we then convert to simulation inputs using the EasyVVUQ encoder, and submit to the HPC resource using FabSim3. Once execution has concluded, we then decode and collate the results and perform a Sobol sensitivity analysis relative to our QoI (number of deaths over time).

We present the first-order Sobol sensitivity indices for each parameter in table 3 and in figure 10. These global sensitivity indices [67] measure the fraction of the output variance (over time here), that can be attributed to a single input parameter. During the first 20 days, all parameters except for the (non-mild) recovery period have a significant effect on the number of deaths. However, as the simulation progresses, the number of deaths is mainly sensitive to the

[20]See https://pypi.org/project/chaospy.

Table 3. Defining parameter space for the uncertain parameters of the Flu And Coronavirus Simulations (FACS).

parameters	type	default value	uniform range
infection rate	float	0.07	(0.0035, 0.14)
mortality period	float	8.0	(4.0, 16.0)
recovery period	float	8.0	(4.0, 16.0)
mild recovery period	float	8.05	(4.5, 12.5)
incubation period	float	3.0	(2.0, 6.0)
period to hospitalization	float	12.0	(8.0, 16.0)

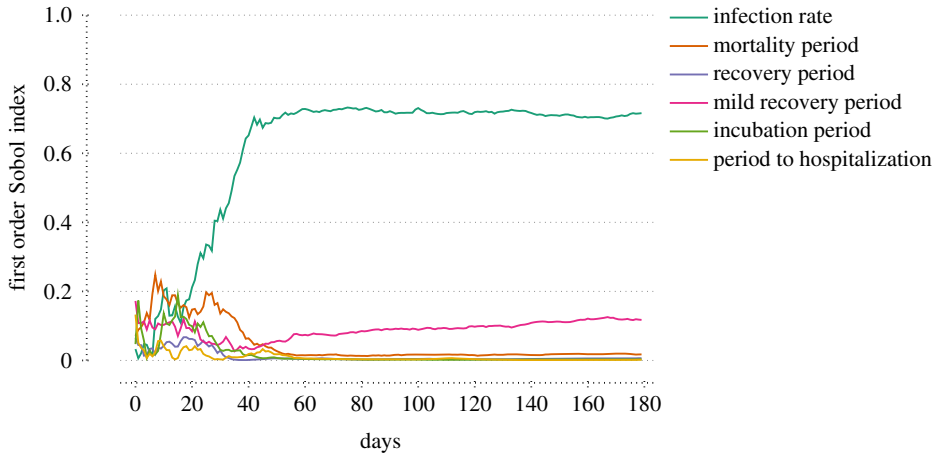


Figure 10. The first-order Sobol indices for each of the uncertain parameters of the Flu And Coronavirus Simulations (FACS) for the London Borough of Brent. (Online version in colour.)

infection rate (which describes how quickly the infection spreads) and the mild recovery period (which describes how long people with mild COVID remain ill and infectious).

References

1. Groen D *et al.* 2019 Introducing vecmatk-verification, validation and uncertainty quantification for multiscale and hpc simulations. In *Int. Conf. on Computational Science, Faro, Portugal*, pp. 479–492. Berlin, Germany: Springer. (doi:10.1007/978-3-030-22747-0_36)
2. Roy CJ, Oberkampf WL. 2011 A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing. *Comput. Methods Appl. Mech. Eng.* **200**, 2131–2144. (doi:10.1016/j.cma.2011.03.016)
3. National Research Council of the National Academies. 2012 Assessing the reliability of complex models: Mathematical and statistical foundations of verification, validation, and uncertainty quantification. National Academies Press. (doi:10.17226/13395)
4. Suleimenova D, Arabnejad H, Edeling WN, Groen D. 2021 Sensitivity-driven simulation development: a case study in forced migration. *Phil. Trans. R. Soc. A* **379**, 20200077. (doi:10.1098/rsta.2020.0077)
5. Schwer LE. 2009 Guide for verification and validation in computational solid mechanics. In *the 20th Int. Conf. on Structural Mechanics in Reactor Technology*. New York, NY: American Society of Mechanical Engineers. See https://repository.lib.ncsu.edu/bitstream/handle/1840.20/23659/3_paper_2010.
6. Oberkampf WL, Roy CJ. 2010 *Verification and validation in scientific computing*. Cambridge, UK: Cambridge University Press.
7. Simmermacher T, Tipton G, Cap J, Mayes R. 2015 The role of model V&V in the defining of specifications. In *the Conf. Proc. of the Society for Experimental Mechanics Series*,

Orlando, FL (eds H Atamturktur, B Moaveni, C Papadimitriou, T Schoenherr). Model Validation and Uncertainty Quantification, vol. 3. pp. 257–263. Cham, Switzerland: Springer. (doi:10.1007/978-3-319-15224-0_27)

8. Binois M, Gramacy R, Ludkovski M. 2018 Practical heteroscedastic Gaussian process modeling for large simulation experiments. *J. Comput. Graph. Stat.* **27**, 808–821. (doi:10.1080/10618600.2018.1458625)
9. Baker E, Gramacy R, Huang J, Johnson L, Mondal A, Pires B, Sacks J, Sokolov V. 2020 Stochastic simulators: an overview with opportunities. (http://arxiv.org/abs/2002.01321)
10. Adams BM, Bohnhoff WJ, Dalbey KR, Eddy JP, Eldred MS, Gay DM, Haskell K, Hough PD, Swiler LP. 2009 DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 5.0 user's manual. Sandia National Laboratories, Technical Report. SAND2010-2183. (doi:10.2172/991841)
11. Lin G, Engel DW, Eslinger PW. 2012 Survey and evaluate uncertainty quantification methodologies. Pacific Northwest National Lab.(PNNL), Richland, WA (United States). (doi:10.2172/1035732)
12. Foley SS, Elwasif WR, Bernholdt DE, Shet AG, Bramley R. 2010 Many-task applications in the integrated plasma simulator. In *the 3rd Workshop on Many-Task Computing on Grids and Supercomputers*, pp. 1–10. IEEE. (doi:10.1109/MTAGS.2010.5699425)
13. Elwasif WR, Bernholdt DE, Pannala S, Allu S, Foley SS. 2012 Parameter sweep and optimization of loosely coupled simulations using the DAKOTA toolkit. In *the 15th Int. Conf. on Computational Science and Engineering, Nicosia, Cyprus*, pp. 102–110. Piscataway, NJ: IEEE. (doi:10.1109/ICCSE.2012.24)
14. Debusschere B, Sargsyan K, Safta C, Rai P, Chowdhary K. 2018 UQTk: a flexible Python/C++ Toolkit for Uncertainty Quantification. Albuquerque, NM: Sandia National Lab. (SNL-NM).
15. Giles M. 2015 Multilevel Monte Carlo methods. *Acta Numerica* **24**, 259–328. (doi:10.1017/S096249291500001X)
16. Baudin M, Dutfoy A, Iooss B, Popelin A. 2015 Open TURNS: an industrial software for uncertainty quantification in simulation. In *Handbook of uncertainty quantification* (eds R Ghanem, D Higdon and H Owhadi). Cham, Switzerland: Springer. See http://arxiv.org/abs/1501.05242.
17. Balasubramanian V, Jha S, Merzky A, Turilli M. 2019 Radical-cybertools: middleware building blocks for scalable science. See http://arxiv.org/abs/1904.03085.
18. Gattiker JR. 2008 Gaussian process models for simulation analysis (GPM/SA) command, function, and data structure reference. Los Alamos National Laboratory, Technical Report LA-UR-08-08057. See https://www.lanl.gov/org/docs/gpmsa-command-ref.pdf.
19. Gattiker J, Myers K, Williams B, Higdon D, Carzolio M, Hoegh A. 2017 Gaussian process-based sensitivity analysis and Bayesian model calibration with GPMSA. In *Handbook of uncertainty quantification*, pp. 1–41. Cham, Switzerland: Springer. (doi:10.1007/978-3-319-11259-6_58-1)
20. Wozniak JM, Armstrong TG, Wilde M, Katz DS, Lusk E, Foster IT. 2013 Swift/T: large-scale application composition via distributed-memory dataflow processing. In *2013 13th IEEE/ACM Int. Symp. on Cluster, Cloud, and Grid Computing*, pp. 95–102. (doi:10.1109/CCGrid.2013.99)
21. Babuji Y *et al.* 2019 Parsl: pervasive parallel programming in python. In *Proc. of the 28th Int. Symp. on High-Performance Parallel and Distributed Computing (HPDC '19)*. Association for Computing Machinery, New York, NY, USA, 25–36. (doi:10.1145/3307681.3325400)
22. Tennøe S, Halnes G, Einevoll GT. 2018 Uncertainpy: a python toolbox for uncertainty quantification and sensitivity analysis in computational neuroscience. *Front. Neuroinformatics* **12**, 1–29. (doi:10.3389/fninf.2018.00049)
23. Lawrence Livermore National Laboratory. 2016. *Non-intrusive uncertainty quantification: PSUADE*. Livermore, CA: Lawrence Livermore National Laboratory. See https://computing.llnl.gov/projects/psuade-uncertainty-quantification.
24. Hittinger JA, Cohen BI, Klein RI. 2010 *Uncertainty quantification in the fusion simulation project verification and validation activity*. Livermore, CA: Lawrence Livermore National Laboratory. (doi:10.2172/1119966)

25. Brown JD, Heuvelink GBM. 2007 The data uncertainty engine (DUE): a software tool for assessing and simulating uncertain environmental variables. *Comput. Geosci.* **33**, 172–190. (doi:10.1016/j.cageo.2006.06.015)
26. Salvatier J, Wiecki TV, Fonnesbeck C. 2016 Probabilistic programming in Python using PyMC3. *PeerJ Comput. Sci.* **2**, e55. (doi:10.7717/peerj-cs.55)
27. Marelli S, Sudret B. 2014 UQLab: a framework for uncertainty quantification in Matlab. In *the 2nd Int. Conf. on Vulnerability, Risk Analysis and Management, Liverpool, UK*, pp. 2554–2563. (doi:10.1061/9780784413609.257)
28. Pianosi F, Sarrazin F, Wagener T. 2015 A Matlab toolbox for global sensitivity analysis. *Environ. Model. Softw.* **70**, 80–85. (doi:10.1016/j.envsoft.2015.04.009)
29. Ye D, Veen L, Nikishova A, Lakhlili J, Edeling W, Luk OO, Krzhizhanovskaya VV, Hoekstra AG. 2021 Uncertainty quantification patterns for multiscale models. *Phil. Trans. R. Soc. A* **379**, 20200072. (doi:10.1098/rsta.2020.0072)
30. Alowayyed S, Groen D, Coveney PV, Hoekstra AG. 2017 Multiscale computing in the exascale era. *J. Comput. Sci.* **22**, 15–25. (doi:10.1016/j.jocs.2017.07.004)
31. Jancauskas V, Lakhlili J, Richardson RA, Wright DW. 2020 EasyVVUQ. See https://github.com/UCL-CCS/EasyVVUQ.
32. Groen D, Bhati AP, Suter J, Hetherington J, Zasada SJ, Coveney PV. 2016 FabSim: facilitating computational research through automation on large-scale and distributed e-infrastructures. *Comput. Phys. Commun.* **270**, 375–385. (doi:10.1016/j.cpc.2016.05.020)
33. Lourens V. 2020 MUSCLE 3: the multiscale coupling library and environment. See https://github.com/multiscale/muscle3.
34. Richardson RA, Wright DW, Edeling W, Jancauskas V, Lakhlili J, Coveney PV. 2020 EasyVVUQ: a library for verification, validation and uncertainty quantification in high performance computing. *J. Open Res. Softw.* **8**, 1–8. (doi:10.5334/jors.303)
35. Luckow A, Santcroos M, Weidner O, Merzky A, Maddineni S, Jha S. 2012 Towards a common model for pilot-jobs. In *Proc. of the 21st Int. Symp. on High-Performance Parallel and Distributed Computing*, pp. 123–124.
36. Veen LE, Hoekstra AG. In press. *Easing multiscale model design and coupling with muscle 3. Computational Science – ICCS 2020*. Berlin, Germany: Springer.
37. Borgdorff J, Falcone J, Lorenz E, Bona-Casas C, Chopard B, Hoekstra AG. 2013 Foundations of distributed multiscale computing: formalization, specification, and analysis. *J. Parallel Distrib. Comput.* **73**, 465–483. (doi:10.1016/j.jpdc.2012.12.011)
38. Chopard B, Borgdorff J, Hoekstra AG. 2014 A framework for multi-scale modelling. *Phil. Trans. R. Soc. A* **372**, 20130378. (doi:10.1098/rsta.2013.0378)
39. Nikishova A, Hoekstra AG. 2019 Semi-intrusive uncertainty propagation for multiscale models. *J. Comput. Sci.* **35**, 80–90. (doi:10.1016/j.jocs.2019.06.007)
40. Luk OO, Hoenen O, Bottino A, Scott BD, Coster DP. 2019 ComPat framework for multiscale simulations applied to fusion plasmas. *Comput. Phys. Commun.* **239**, 126–133. Elsevier. (doi:10.1016/j.cpc.2018.12.021)
41. Lakhlili J, Hoenen O, Luk OO, Coster DP. 2020 Uncertainty quantification for multiscale fusion plasma simulations with VECMA toolkit. In *Computational Science - ICCS 2020* (eds V Krzhizhanovskaya *et al.*). Lecture Notes in Computer Science, vol. 12143. Springer, Cham. (doi:10.1007/978-3-030-50436-6_53)
42. Luk OO, Lakhlili J, Hoenen O, von Toussaint U, Scott BD, Coster DP. 2021 Towards validated multiscale simulations for fusion. *Phil. Trans. R. Soc. A* **379**, 20200074. (doi:10.1098/rsta.2020.0074)
43. Nikulin, Mikhail S. 2001 *Hellinger distance*, vol. 78. Springer, NY: Encyclopedia of mathematics.
44. Kullback S, Leibler RA. 1951 On information and sufficiency. *Ann. Math. Stat.* **22**, 79–86. (doi:10.1214/aoms/1177729694)
45. Villani C. 2016 *Optimal transport: old and new*. Berlin, Germany: Grundlehren der mathematischen.
46. Groen D. 2016 Simulating refugee movements: where would you go? *Procedia Comput. Sci.* **80**, 2251–2255. (doi:10.1016/j.procs.2016.05.400)
47. Suleimenova D, Bell D, Groen D. 2017 A generalized simulation development approach for predicting refugee destinations. *Sci. Rep.* **7**, 13377. (doi:10.1038/s41598-017-13828-9)
48. Suleimenova D, Bell D, Groen D. 2017 Towards an automated framework for agent-based simulation of refugee movements. In *The Proc. of the 2017 Winter Simulation Conf., Las Vegas,*

NV (eds WKV Chan, A DAmbrogio, G Zacharewicz, N Mustafee, G Wainer, E Page), pp. 1240–1251. Piscataway, NJ: IEEE. (doi:10.1109/WSC.2017.8247870)
49. Groen D, Bell D, Arabnejad H, Suleimenova D, Taylor SJE, Anagnostou A. 2019 Towards modelling the effect of evolving violence on forced migration. In *the 2019 Winter Simulation Conf. (WSC)*, pp. 297–307. (doi:10.1109/WSC40007.2019.9004683)
50. Suleimenova D, Groen D. 2020 How policy decisions affect refugee journeys in South Sudan: a study using automated ensemble simulations. *J. Artif. Soc. Soc. Simul.* **23**, 14. (doi:10.18564/jasss.4193)
51. Gent PR, Mcwilliams JC. 1990 Isopycnal mixing in ocean circulation models. *J. Phys. Oceanogr.* **20**, 150–155. (doi:10.1175/1520-0485(1990)020<0150:IMIOCM>2.0.CO;2)
52. Heus T *et al.* 2010 Formulation of the Dutch Atmospheric Large-Eddy Simulation (DALES) and overview of its applications. *Geosci. Model Dev.* **3**, 415–444. (doi:10.5194/gmd-3-415-2010)
53. Jansson F, Edeling W, Attema J, Crommelin D. 2021 Assessing uncertainties from physical parameters and modelling choices in an atmospheric large eddy simulation model. *Phil. Trans. R. Soc. A* **379**, 20200073. (doi:10.1098/rsta.2020.0073)
54. Edeling W, Groen D. 2019 FabUQCampaign. See https://github.com/wedeling/FabUQCampaign.
55. Edeling W, Crommelin D. 2020 Reducing data-driven dynamical subgrid scale models by physical constraints. *Comput. Fluids* **201**, 1–11. (doi:10.1016/j.compfluid.2020.104470)
56. Crommelin D, Edeling W. 2020 Resampling with neural networks for stochastic parameterization in multiscale systems. (http://arxiv.org/abs/2004.01457)
57. Vassaux M, Sinclair RC, Richardson RA, Suter JL, Coveney PV. 2020 Toward high fidelity materials property prediction from multiscale modeling and simulation. *Adv. Theory Simul.* **3**, 1–18. (doi:10.1002/adts.201900122)
58. Vassaux M, Richardson RA, Coveney PV. 2019 The heterogeneous multiscale method applied to inelastic polymer mechanics. *Phil. Trans. R. Soc. A* **377**, 20180150. (doi:10.1098/rsta.2018.0150)
59. Suter J, Sinclair R, Coveney P. 2020 Principles governing control of aggregation and dispersion of graphene and graphene oxide in polymer melts. *Adv. Mater.* **32**, 2003213. (doi:10.1002/adma.202003213)
60. Chen F *et al.* 2011 The integrated WRF/urban modeling system and its applications to urban environmental problems. *Int. J. Climatol.* **31**, 273–288. (doi:10.1002/joc.215810.1002/joc.2158)
61. Prusa JM, Smolarkiewicz PK, Wyszogrodzki A. 2008 EULAG a computational model for multiscale flows. *Comput. Fluids* **37**, 1193–1207. (doi:10.1016/j.compfluid.2007.12.001)
62. Wyszogrodzki A, Miao S, Chen F. 2012 Evaluation of the coupling between mesoscale-WRF and LES-EULAG models for simulating fine-scale urban dispersion. *Atmos. Res.* **118**, 324–345. (doi:10.1016/j.atmosres.2012.07.023)
63. Wright DW *et al.* 2020 Building confidence in simulation: applications of EasyVVUQ. *Adv. Theory Simul.* **3**, 1900246. (doi:10.1002/adts.201900246)
64. Nikishova A, Veen L, Zun P, Hoekstra AG. 2018 Uncertainty quantification of a multiscale model for in-stent restenosis. *Cardiovasc. Eng. Technol.* **9**, 761–774. (doi:10.1007/s13239-018-00372-4)
65. Nikishova A, Veen L, Zun P, Hoekstra AG. 2019 Semi-intrusive multiscale metamodelling uncertainty quantification with application to a model of in-stent restenosis. *Phil. Trans R. Soc. A* **377**, 20180154. (doi:10.1098/rsta.2018.0154)
66. Ye D, Nikishova A, Veen L, Zun P, Hoekstra AG. 2020 Non-intrusive and semi-intrusive uncertainty quantification of a multiscale in-stent restenosis model. (http://arxiv.org/abs/2009.00354)
67. Saltelli A, Ratto M, Andres T, Saisana M, Tarantola S. 2008 *Global sensitivity analysis: the primer*. New York, NY: John Wiley & Sons.

Publication [P7]

Bosak, B., Kopta, P., Kulczewski, M., and Piontek, T.: *mUQSA – An online service for uncertainty quantification and sensitivity analysis.* In Paszynski, M., Barnard, A. S., and Zhang, Y. J. (eds), *Computational Science – ICCS 2025 Workshops.* Lecture Notes in Computer Science, vol. 15911. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-97570-7_6

Ministry points / conference: 140

Contribution of authors:

- **Bartosz Bosak**
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, Related work, mUQSA Functionality, mUQSA Architecture, Practical Example, Conclusion and Future Work)
- Piotr Kopta
  - Coauthorship of the text of publication (Sections: mUQSA Architecture)
- Michał Kulczewski
  - Coauthorship of the text of publication (mUQSA Functionality, Conclusion and Future Work)
- Tomasz Piontek
  - Review and editing of the paper
  - Coauthorship of the text of publication (Introduction, Conclusion and Future Work)

mUQSA - an online service for Uncertainty Quantification and Sensitivity Analysis

Bartosz Bosak[1][0000−0002−9331−3166], Piotr Kopta[1][0000−0001−8237−0969], Michał Kulczewski[1][0000−0002−1349−2927], and Tomasz Piontek[1][0000−0003−0147−3996]

Poznan Supercomputing and Networking Center
ul. Jana Pawła II 10, 61-139 Poznan, Poland
office@man.poznan.pl
https://www.psnc.pl

Abstract. Enabling uncertainty quantification (UQ) for a computational model can be a complex process, often exceeding the domain expertise of its developer or user. To address this challenge, we developed the mUQSA platform, which streamlines typical uncertainty quantification scenarios through a modern web-based environment and pre-configured automated workflows that leverage large-scale computing resources for state-of-the-art UQ techniques. mUQSA is designed with two key objectives: to support users in analyzing how input uncertainties propagate through their models and to identify the input parameters that most significantly influence outcomes through sensitivity analysis (SA). The particular strength of mUQSA lies in its accessibility: offered as a Software-as-a-Service (SaaS) solution, mUQSA eliminates the need for installation or configuration of software, enabling users to access UQ/SA methods directly from a web browser. Its user-friendly, intuitive interfaces for defining use cases and displaying results guarantee high accessibility, while a robust execution layer, supported by HPC resources, ensures mUQSA can handle even very complex computational models.

Keywords: Uncertainty Quantification · Sensitivity Analysis · multiscale · HPC

1 Introduction

Uncertainty Quantification is one of the major tools for assessing credibility of computational simulations. Based on a solid mathematical framework, UQ techniques address several problems in scientific modeling, such as certification, prediction, verification and validation of model and software, parameter estimation, data assimilation, and inverse problems [20]. However, applying uncertainty quantification (UQ) to computational models, although often essential for ensuring actionable results, represents an additional layer of effort, both logically and technologically, that extends beyond the core domain of expertise of an individual scientist or engineer. Furthermore, UQ algorithms, while potentially critical to the validity of simulation results, are supplementary to the fundamental scientific model itself. Finally, the increased number of executions needed for reliable

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

UQ imposes significantly higher computational resource demands, especially for highly complex models, such as those in multiscale simulations.

Given these factors, we leveraged the EasyVVUQ library [18], which provides core methods for efficient Verification, Validation, and Uncertainty Quantification of computational models and is a key outcome of the VECMA project [23], alongside two components of the QCG family [17]: QCG-Portal, engineered for the intuitive submission and management of computational experiments on supercomputers, and QCG-PilotJob [16], a second-level scheduler that optimizes the execution of numerous logical jobs within a single HPC resource allocation, to build the multipurpose Uncertainty Quantification and Sensitivity Analysis framework, mUQSA. To provide seamless access to the advanced computational capabilities it integrates, mUQSA has been developed to work in a Software-as-a-Service (SaaS) model.

Established under the Polish national PIONIER-LAB project [15], mUQSA is currently being refined within the European HiDALGO2 project [11] to address the specific requirements of environmental use cases and support the evolving needs of simulation developers and end users.

From a design perspective, mUQSA integrates intuitive web-based user interface, HPC processing, and state-of-the-art UQ algorithms, delivering a comprehensive and versatile environment for widely requested uncertainty related studies, including non-intrusive forward uncertainty quantification and sensitivity analysis.

That said, mUQSA is not intended to be a universal tool for Uncertainty Quantification (UQ). Instead, it targets common UQ and SA workflows, aiming to streamline their implementation. The provided UQ methods enable users to identify basic trends and compute key statistical moments for Quantities of Interest (QoIs), which are critical output values influenced by uncertain input parameters. Furthermore, the Sensitivity Analysis, based on Sobol's indices, offers a robust approach to assessing the relative importance of specific inputs. As such, the platform offers flexibility across a broad spectrum of applications, ranging from monolithic molecular dynamics codes and complex multiscale climate models to the development of digital human twins.

Thanks to the intuitiveness of the interface and the automation offered, mUQSA can also be a starting point to get practical knowledge in the fields of uncertainty quantification and sensitivity analysis for those who are not yet familiar with the concepts.

The rest of this paper is organised as follows. In Section 2 we briefly present other software available for similar purposes. Section 3 outlines main functionality provided by mUQSA and the way it is offered to users. Section 4 elaborates on the execution layer of the platform. Finally, in Section 5 we conclude and present future plans for the development of the platform.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

2 Related work

Verification, Validation, and Uncertainty Quantification (VVUQ) are critical topics addressed in numerous projects and initiatives, playing an essential role in both science and industry. To support the application of VVUQ in scientific simulations, efforts are underway, such as those of the American Society of Mechanical Engineers (ASME) [22], to establish best practices and develop standards for common VVUQ processes. SmartUQ, a leading provider of UQ software, has also highlighted the benefits of simplification, reporting that its UQ tools have saved stakeholders millions of dollars and thousands of working hours [1]. Alongside such commercial solutions, a number of smaller-scale initiatives and diverse research teams successfully develop software tailored to general and specialized VVUQ applications, much of it distributed as open source. Libraries such as EasyVVUQ [18], UncertainPy [21], Chaospy [8], and OpenTURNS [3], as well as more comprehensive toolkits and frameworks such as Dakota [6], Uranie [4], and the VECMA Toolkit [9], offer a high degree of flexibility and adaptability in a wide range of use cases. However, these tools often require substantial initial effort to set up and may encounter scalability challenges in large-scale workflows without complex configuration. Specifically, while Dakota is one of the most popular and feature-rich UQ frameworks, offering advanced capabilities like calibration and surrogate modeling, its setup process can be cumbersome. Configuring Dakota to run workflows on remote computing resources involves minimal automation, which presents significant challenges to novice users. Similar difficulties are encountered by users of other well-recognized tools, such as OpenTURNS. The situation is somewhat different with the VECMA Toolkit, where FabSim3[10], combined with EasyVVUQ and QCG-PilotJob[16], introduces several automation features to facilitate the execution of UQ processes on HPC resources. However, VECMA Toolkit is focused on command-line usage, which may still be considered too complex for some users.

3 mUQSA Functionality

The goal of the mUQSA platform is to provide an easy-to-use web environment that enables automated uncertainty quantification and sensitivity analysis of computational models on high-performance computing (HPC) resources.

This placement of the mUQSA toolkit is achieved through several means.

Intuitive wizard for the preparation of the UQ/SA scenario. The wizard is divided into several logical steps that allow one to collect all necessary information required to run UQ/SA for a selected model on computational resources. Thus, we have separate steps for the definition of sampled parameters, configuration of the method (see Figure 1), or specification of the way the model should be executed. Combined with the provided documentation, the step-by-step collection of information in the wizard streamlines the process of incorporating UQ into the model and makes it achievable even for non-experienced users.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

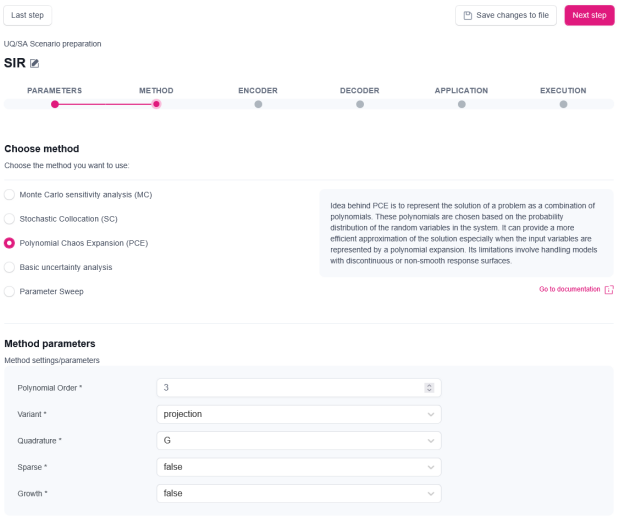


Fig. 1. Selection and configuration of a UQ method in the mUQSA wizard

Built-in interactive presentation of results With built-in support for displaying output data as interactive charts and tables in predefined formats, users can access insights from their analysis in a concise, visually engaging way, with the structure effectively showcasing key findings. Figure 2 illustrates a result presentation view for the sensitivity analysis, using the SIR model as an example (this model will be illustrated in more detail later in Section 5). If this is not sufficient, users can also access the raw results data, enabling deeper analysis when needed.

Automated execution on HPC machine One of the primary challenges in uncertainty quantification for computationally intensive models is the ability to efficiently and transparently perform the necessary, often extensive, computations on supercomputers. This is where mUQSA excels. The platform seamlessly manages the scheduling and execution of tasks required for the sampled parameters, reducing the user's role to simply initiating the experiment, while all subsequent processes run effortlessly in the background.

Support and documentation While mUQSA simplifies the definition and execution of uncertainty quantification and sensitivity analysis tasks, users with limited prior knowledge of these procedures may find the interaction with the platform challenging. Additionally, some advanced features may require more in-depth explanation beyond the brief help descriptions provided in the graphical user interfaces (GUIs). To address this, mUQSA is supported by comprehensive documentation and a set of tutorials that demonstrate its usage in detail, ensuring that users receive the necessary assistance. [14]

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

Fig. 2. Presentation of Sensitivity Analysis results in mUQSA

3.1 mUQSA workflow

Conceptually, mUQSA can be divided into two main layers: the User Interface (UI) layer, and the Execution layer.

The UI layer has two main duties:

1. definition of all input parameters and configuration settings for UQ/SA scenario,
2. presentation of results for further analysis, once the required calculations have been completed.

In turn, the Execution layer's role is to efficiently execute required computations according to the selected method and user-defined input data. Given this context, an overall picture of the execution flow in mUQSA can be visualised as in Figure 3

The Scenario Definition and Analysis steps correspond directly to the two above duties of UI. Scenario Definition's aim is to collect all the specific data on the scenario from a user, including the selection of a Method and the definition of Parameters and then to push it for fully automatic processing on an HPC machine. Analogically, the Analysis step receives results from the processing on the HPC machine and displays an interactive web page with condensed information for further analysis by a user.

The rest of steps of the workflow have a strictly computational character and belong to the Execution layer. Sampler generates sets of inputs for model evaluations according to a selected method. Then, multiple streams, each comprising an Encoder, Execution, and Decoder, are instantiated - one per required evaluation - and executed in parallel. Next, the output data from the evaluations is merged together by Collator and analysed programmatically by Analyser, again

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

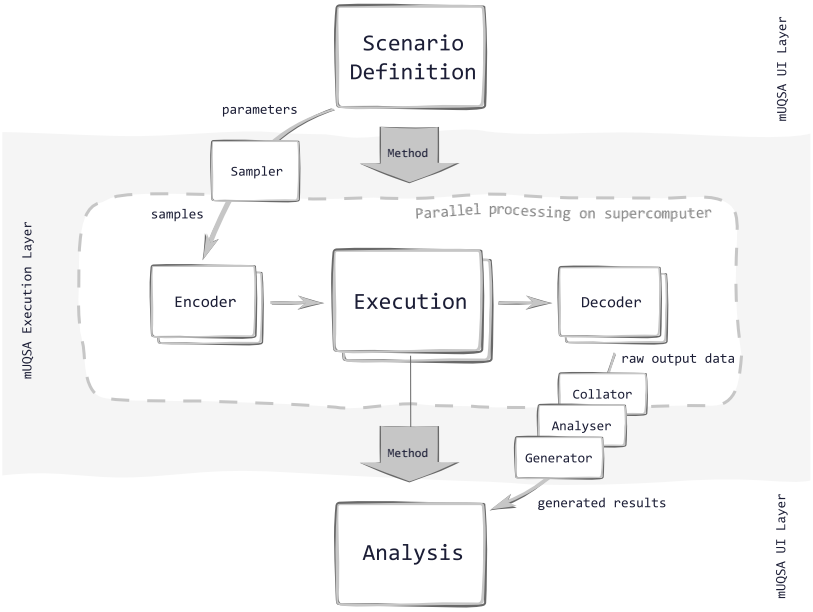


Fig. 3. High-level view on the mUQSA conceptual workflow

in accordance with the selected method. Finally, the results are converted into human-readable form by Generator.

3.2 Supported UQ/SA methods

mUQSA's core algorithmic component is derived from the EasyVVUQ library [18]. This choice of algorithmic engine for mUQSA is deliberate, as EasyVVUQ, with its well-defined API, provides a robust set of widely recognized and valued uncertainty quantification methods. mUQSA incorporates most of these techniques, including:

- **(Quasi-)Monte Carlo (MC)** - a statistical method for numerical integration or simulation that uses random sampling. Quasi-Monte Carlo improves on standard Monte Carlo by using low-discrepancy sequences to achieve faster convergence.
- **Polynomial Chaos Expansion (PCE)** - a technique that represents uncertain quantities as a series of orthogonal polynomials, providing an efficient way to approximate the response of a system to input uncertainties.
- **Stochastic Collocation (SC)** - a spectral method that approximates the solution of a problem by collocating the system at specific sample points (based on the probability distribution of the inputs) and constructing a global approximation through interpolation

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

- **Ensemble Analysis** - a basic method where multiple simulations are run with different sets of input parameters to analyze the overall system behavior. The results are aggregated to understand the variability and uncertainty.
- **Parameter Sweep** - a technique where a model is run repeatedly across a range of input values, systematically varying one or more parameters to explore how the system's output changes with respect to these inputs.

As a result, mUQSA can be viewed, with some simplification, as a web-based interface for EasyVVUQ. Specifically, it serves two key purposes: 1) provides an intuitive interface to access selected EasyVVUQ methods, and 2) is preconfigured to use a designated computational resource, ensuring efficient execution of the computations defined by these methods.

Each of the methods supported by mUQSA can handle scalar and vector Quantities of Interest (QoIs) and can be configured to match specific needs of the given scenario.

Additionally, leveraging the results of the MC, PCE, and SC methods, the platform automatically computes **Sobol's indices**. This allows for a detailed sensitivity analysis that quantifies the contribution of each input parameter (or combinations of parameters) to the output variance. The Sobol indices help identify the most influential factors driving uncertainty by decomposing the total variance into contributions from individual parameters and their interactions.

It is worth noting that the presentation of results in the platform is largely consistent across methods, enabling users to compare them, discern trends, and identify the most suitable options.

4 mUQSA Architecture

Developing a comprehensive Uncertainty Quantification system poses challenges on multiple conceptual and technological fronts. Fortunately, in development of mUQSA, we were able to reuse several already available components that allowed to build a functional system in relatively short time.

The conceptual scheme of the mUQSA workflow described in Figure 3 can be mapped to the representation focused on the concrete software components shown in Figure 4. QCG, EasyVVUQ, and QCG-PilotJob [16] are software packages adopted from previous initiatives, while all the blocks highlighted in red are newly developed components for mUQSA. The following sections will detail this flow, focusing on the specific software components involved.

Problem definition: mUQSA GUI

The mUQSA interface is built on the capabilities of the QCG software, particularly QCG-Portal, which provides core mechanisms for executing and managing computational tasks on remote resources directly from a web browser. QCG-Portal also supports interface customization for specific use cases, enabling adjustments to task input data specification and execution monitoring to meet

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

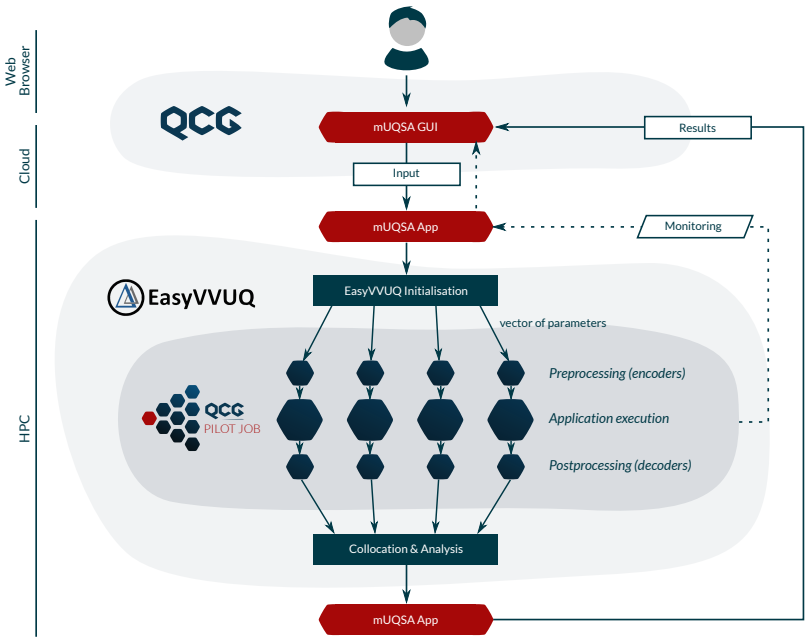


Fig. 4. High-level architecture of mUQSA platform

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

particular needs. Consequently, the mUQSA wizard is a custom JavaScript application that integrates and leverages core functionality of the portal. Once all input data is collected through the wizard, mUQSA transfers it to QCG-Portal, which submits a specific job to be executed on an HPC resource. Importantly, users can track the execution state through a dedicated mUQSA monitoring panel within the task details view in QCG-Portal (see Figure 5).

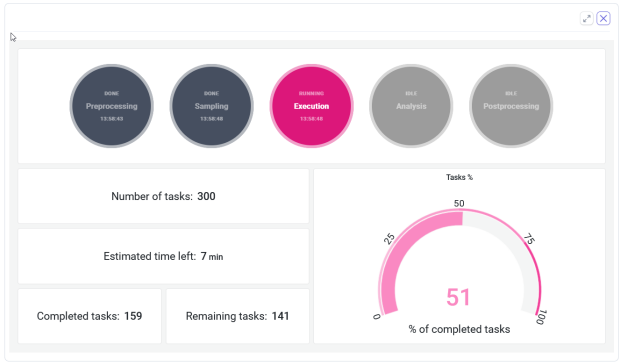


Fig. 5. mUQSA monitoring dashboard in QCG-Portal

Execution preparation: mUQSA App

When the job submitted from QCG-Portal is launched on an HPC resource, the specifically crafted mUQSA App is started and takes care of establishing the entire execution. In the first step, it converts input configuration (stored in a JSON document) to the EasyVVUQ's Campaign object to enable seamless execution of the scenario with the selected method on allocated resources.

Execution: EasyVVUQ and QCG-PilotJob

The configured campaign is executed by EasyVVUQ through a series of steps, several of which can be effectively parallelized. After method-specific sampling, the resulting parameter vectors are used to evaluate the model, which also involves encoding and decoding phases. Using EasyVVUQ's integration with QCG-PilotJob, these evaluations are dynamically scheduled and executed in parallel, optimizing resource utilization and improving computational efficiency. In particular, since QCG-PilotJob supports the execution of parallel codes, individual evaluations can be applications parallelized with MPI or OpenMP. Moreover, beyond evaluating pre-installed codes, mUQSA enables the execution of models provided as Apptainer containers [2], relieving users from the often challenging task of compiling and installing code on a computing resource. Throughout the

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

process, custom monitoring scripts relay updates on the execution stage and progress to centralized QCG monitoring services. Upon completion, EasyVVUQ aggregates data from all evaluations and performs a method-specific analysis.

Results preparation: mUQSA App

The raw output data generated by EasyVVUQ is then accessed by the mUQSA App, which transforms it into interactive and visually appealing web pages summarizing the analysis performed. Although the primary use case for mUQSA involves the GUI, the underlying software can also be used without it. Since the input data is in JSON format and web page generation occurs on the cluster, command-line usage is also possible.

Results visualisation: mUQSA GUI

Finally, the generated web page is presented to the user within QCG-Portal, enabling analysis of the results through zoomable and draggable charts, filtered tables, and, if needed, the option to export data into widely recognized formats for further exploration in an external program of choice.

5 Practical Example

This section demonstrates mUQSA's functionality through a case study of the SIR compartment model [24], a framework for simulating infections by categorizing a population into Susceptible (S), Infectious (I), and Recovered (R) groups [24]. The purpose of this section is to provide a brief overview, while more detailed showcases are available as tutorials on the mUQSA web page [14].

The SIR model starts with a defined state of S, I and R groups, and then it is iteratively updated using differential equations based on two key parameters:

β : Infection rate, the average number of contacts per person per time unit resulting in transmission from Susceptible to Infected.
γ : Recovery rate, the rate at which Infected individuals transition to Recovered (the inverse of infection duration).

The objective is to evaluate the output uncertainty of the model given uncertain inputs β and γ and analyse the model's sensitivity to their variability. The process is described below.

5.1 Accessing mUQSA

To use mUQSA, users require access to a resource where the framework is preconfigured. This may be a private, self-hosted deployment of QCG-Portal and mUQSA software or, more commonly, a public HPC system, such as the Proxima cluster at PSNC, with mUQSA and its dependencies pre-installed. Once access is granted, users can interact with mUQSA directly via a web browser,

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

eliminating the need for local software installation or configuration. By logging into QCG-Portal (e.g. https://qcg.pcss.plcloud.pl) with the credentials provided, users select mUQSA from the available applications - often the default in single-application setups - and launch the mUQSA scenario definition wizard.

5.2 Configuring UQ/SA Scenario

In the mUQSA wizard, users can choose to start a completely new scenario or load a previously prepared one and adjust its settings. The wizard consists of the following steps:

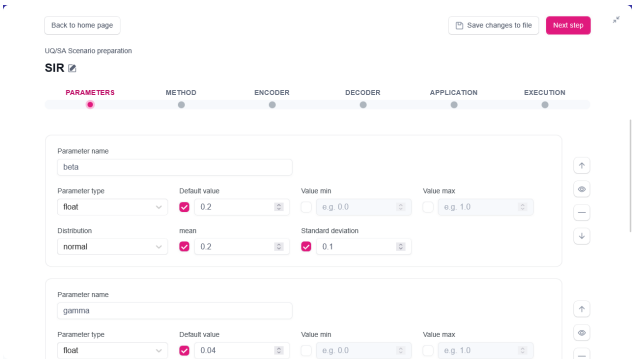


Fig. 6. Definition of parameters in the mUQSA wizard

Step 1 - Parameters : Allows users to define input parameters for analysis, specifically uncertain parameters, their distributions, and constraints. This guides UQ algorithms on how to sample parameter combinations for model evaluation. For our example analysis of the SIR model, β (infection rate) is assigned a uniform distribution, and γ (recovery rate) follows a normal distribution (Figure 6). The rest of parameters, such as the initial number of infected persons, have static values and would not be sampled.

Step 2 - Method : Enables the selection and configuration of the uncertainty quantification (UQ) method for the specified scenario. The wizard provides brief descriptions of the methods available to guide selection, with comprehensive documentation linked for new users seeking a deeper understanding. For our analysis, we chose the PCE method with a polynomial order of 3 (see Figure 1), although other methods can also be explored.

Step 3 - Encoder : Focuses on specifying how the generic input data from mUQSA, processed via EasyVVUQ, is encoded into a format compatible with the model. Our code relies on simple text file inputs, which can be generated using available encoders such as GenericEncoder. This encoder inserts sampled parameters into designated placeholders within a provided template

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

to produce the final input file. Simple templates can be specified directly in the portal, while more complex templates can be referenced from the file system of the computing resource. For highly specialized cases, custom encoders can be developed to meet specific requirements.

Step 4 - Decoder : Converts model-specific outputs (quantities of interest, QoI) into a generic format for mUQSA (EasyVVUQ) analysis. For the SIR model's CSV output, we use one of already available decoders, which is CSVDecoder. In our case, we analyze the Infected (I) QoI, though Susceptible (S) or Recovered (R) could also be examined. Custom decoders can be provided for complex cases.

Step 5 - Application : Selects the model for evaluation, which can be a preinstalled application or an Apptainer image. In both cases, mUQSA assumes that the model is available on a computing resource before the UQ workflow is launched. The SIR model, since it is bundled with mUQSA, is chosen by pointing to its location in the mUQSA installation directory on the Proxima cluster.

Step 6 - Execution : Specifies execution settings, including whether evaluations run serially or in parallel and how many evaluations run concurrently. For the SIR model, we configure serial execution with multiple evaluations running in parallel. To enhance user experience, the wizard estimates the required wall clock time and CPU hours based on the chosen UQ algorithm and specified execution settings.

Once configured, the scenario is ready to be submitted for processing on the remote computing resource.

5.3 Execution

The submitted mUQSA workflow is placed in a queue in the computing cluster, waits for free resources, and is then launched and executed. This process is fully automatic. Users can track the state and progress of the workflow using the built-in QCG-Portal features and the mUQSA monitoring panel provided (see Figure 5 for reference).

5.4 Analysis

When the calculations are complete, the results are displayed on a customised webpage accessible via QCG-Portal (see Figure 2). The output includes graphs and tables showing key statistics (e.g., mean, percentiles) and Sobol indices, tailored to scalar or vector QoIs. For the SIR model vector QoI (Infected, I), a line graph displays the number of infected individuals over time; scalar QoIs would use a pie chart. If the default presentation is insufficient, users can download raw data for further analysis in external tools.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

6 Conclusion and Future Work

Developing versatile, scalable and user-friendly software for uncertainty quantification can be extremely challenging if not infeasible. For this reason, in the initial phase of the development of mUQSA, we chose to temper expectations with respect to versatility. Our focus was to implement a curated set of widely recognized methods and deliver them through an intuitive web portal, ensuring efficient and transparent execution of required computations on supercomputers. The software is currently deployed on the PSNC's infrastructure and is readily accessible to interested users. To date, it has supported several application use cases (for example, [13]) and was made available to participants of a hackathon jointly organized by HiDALGO2, CIRCE, and SEAVEA projects [12]. At this point we already can confirm validity of our approach and its usefulness for the individuals and communities searching for easily-applicable UQ.

Our future plans include widely understood simplification and hardening of the platform, so it will be even more accessible, predictable, and adaptable to individual use-cases. As a first step, we aim to enhance its resilience against failures and introduce the ability to save and load EasyVVUQ Campaign objects at various stages of the workflow execution. These changes are essential for large use cases, where repetition of evaluations may be extremely inefficient.

Although our current focus is not on expanding the number or variety of supported UQ methods, we plan to extend this aspect in the future. Among other enhancements, we are considering adding Markov Chain Monte Carlo (MCMC) to address inverse problems, alongside exploring methods to enable the automatic construction of surrogate models. The implementation of these or other new techniques will depend largly on the expressed needs of the scientists and engineers.

Ultimately, the scope and direction of mUQSA's evolution will hinge on the extent to which experts from various branches of UQ science contribute to its development. To foster this, our planned efforts include promoting and disseminating mUQSA to facilitate outreach and forge new collaborations. We anticipate that the expertise of HiDALGO2, SEAVEA [19], and possibly other UQ-focused initiatives, such as DATAHYKING [5] and ELLIS [7], will advance the functionality and usability of the mUQSA framework, enabling it to address the requirements of transformative applications, such as AI-driven models.

Acknowledgments. Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (EuroHPC JU) and Poland, Germany, Spain, Hungary, France and Greece under grant agreement number: 101093457 (HiDALGO2 project). The development of mUQSA platform received partial funding from the PIONIER-LAB project (POIR.04.02.00-30-A005/16-00).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

References

1. 3545 University Ave Madison: SmartUQ About. https://www.smartuq.com/about/ (2025), [Online; accessed 22-April-2025]
2. Apptainer: Apptainer webpage. https://apptainer.org/ (2025), [Online; accessed 22-April-2025]
3. Baudin, M., Dutfoy, A., Iooss, B., Popelin, A.L.: OpenTURNS: An Industrial Software for Uncertainty Quantification in Simulation, p. 1–38. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-11259-6_64-1
4. Blanchard, Jean-Baptiste, Damblin, Guillaume, Martinez, Jean-Marc, Arnaud, Gilles, Gaudier, Fabrice: The uranie platform: an open-source software for optimisation, meta-modelling and uncertainty analysis. EPJ Nuclear Sci. Technol. **5**, 4 (2019). https://doi.org/10.1051/epjn/2018050, https://doi.org/10.1051/epjn/2018050
5. DATAHYKING: Marie curie doctoral network funded by the european commission under grant agreement no. 101072546, https://datahyking.eu/, [Online; accessed 22-April-2025]
6. Eldred, M., Bohnhoff, W., Hart, W.: Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, sensitivity analysis, and uncertainty quantification (07 1999)
7. ELLIS: European lab for learning & intelligent systems, https://ellis.eu/, [Online; accessed 22-April-2025]
8. Feinberg, J., Langtangen, H.P.: Chaospy: An open source tool for designing methods of uncertainty quantification. Journal of Computational Science **11**, 46–57 (2015). https://doi.org/https://doi.org/10.1016/j.jocs.2015.08.008, https://www.sciencedirect.com/science/article/pii/S1877750315300119
9. Groen, D., Arabnejad, H., Jancauskas, V., Edeling, W.N., Jansson, F., Richardson, R.A., Lakhlili, J., Veen, L., Bosak, B., Kopta, P., Wright, D.W., Monnier, N., Karlshoefer, P., Suleimenova, D., Sinclair, R., Vassaux, M., Nikishova, A., Bieniek, M., Luk, O.O., Kulczewski, M., Raffin, E., Crommelin, D., Hoenen, O., Coster, D.P., Piontek, T., Coveney, P.V.: Vecmatk: a scalable verification, validation and uncertainty quantification toolkit for scientific simulations. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **379**(2197) (Mar 2021). https://doi.org/10.1098/rsta.2020.0221, http://dx.doi.org/10.1098/rsta.2020.0221
10. Groen, D., Arabnejad, H., Suleimenova, D., Edeling, W., Raffin, E., Xue, Y., Bronik, K., Monnier, N., Coveney, P.V.: Fabsim3: An automation toolkit for verified simulations using high performance computing. Computer Physics Communications **283**, 108596 (2023). https://doi.org/https://doi.org/10.1016/j.cpc.2022.108596, https://www.sciencedirect.com/science/article/pii/S0010465522003150
11. HiDALGO2: HiDALGO2 project webpage. https://hidalgo2.eu (2025), [Online; accessed 22-April-2025]
12. HiDALGO2, CIRCE, SEAVEA: Hackathon on verification, validation, and uncertainty quantification (vvuq) for global challenge applications. Hybrid event organized by HiDALGO2, CIRCE, and SEAVEA projects, held at High-Performance Computing Center Stuttgart (HLRS), Germany, and online (jun 2024), https://www.hidalgo2.eu/hackathon-by-hidalgo2-circe-and-seaveavvuq-projects-5-to-7th-of-june-2024/, held from 5–7 June 2024, focusing on the SEAVEA toolkit (SEAVEAtk) and mUQSA integration; registration details at https://forms.gle/4ytFD4TNEWFcA35u9

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

13. Kulczewski, M., Bosak, B., Kopta, P., Szeliga, W., Piontek, T.: Fostering uncertainty quantification in global challenges with muqsa toolkit. In: Wyrzykowski, R., Dongarra, J., Deelman, E., Karczewski, K. (eds.) Parallel Processing and Applied Mathematics. pp. 35–46. Springer Nature Switzerland, Cham (2025)
14. mUQSA: mUQSA documentation webpage. https://muqsa.multiscale.pionier-lab.pionier.net.pl (2025), [Online; accessed 22-April-2025]
15. PIONIER-LAB: PIONIER-LAB project webpage. https://pionier-lab.pionier.net.pl (2025), [Online; accessed 22-April-2025]
16. QCG: QCG-PilotJob webpage. https://qcg-pilotjob.readthedocs.io (2025), [Online; accessed 22-April-2025]
17. QCG: QCG webpage. https://qcg.psnc.pl (2025), [Online; accessed 22-April-2025]
18. Richardson, R., Wright, D., Edeling, W., Jancauskas, V., Lakhlili, J., Coveney, P.: Easyvvuq: A library for verification, validation and uncertainty quantification in high performance computing. Journal of Open Research Software **8**(1), 1–8 (Apr 2020). https://doi.org/10.5334/JORS.303
19. SEAVEA: Software environment for actionable vvuq-evaluated exascale applications, https://www.seavea-project.org/, [Online; accessed 22-April-2025]
20. Sullivan, T.J.: Introduction to uncertainty quantification. Texts in Applied Mathematics **63** (2015). https://doi.org/10.1007/978-3-319-23395-6, a beginner-friendly text explaining UQ concepts, including aleatory and epistemic uncertainties, with examples.
21. Tennøe, S., Halnes, G., Einevoll, G.T.: Uncertainpy: A python toolbox for uncertainty quantification and sensitivity analysis in computational neuroscience. Frontiers in Neuroinformatics **12** (2018). https://doi.org/10.3389/fninf.2018.00049, https://www.frontiersin.org/journals/neuroinformatics/articles/10.3389/fninf.2018.00049
22. The American Society of Mechanical Engineers: ASME VVUQ standards. https://www.asme.org/codes-standards/publications-information/verification-validation-uncertainty (2025), [Online; accessed 22-April-2025]
23. VECMA: VECMA project webpage. https://www.vecma.eu (2025), [Online; accessed 22-April-2025]
24. Wikipedia: Compartmental models in epidemiology. https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology (2025), [Online; accessed 22-April-2025]

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_6

Publication [P8]

Wright, D. W., Richardson, R. A., Edeling, W., Lakhlili, J., Sinclair, R. C., Jancauskas, V., Suleimenova, D., **Bosak, B.**, Kulczewski, M., Piontek, T., Kopta, P., Chirca, I., Arabnejad, H., Luk, O. O., Hoenen, O., Węglarz, J., Crommelin, D., Groen, D., and Coveney, P. V.: *Building confidence in simulation: Applications of EasyVVUQ.* Advanced Theory and Simulations, 3(8):1900246 (2020). https://doi.org/10.1002/adts.201900246

Ministry points / journal: 20

Contribution of authors:

- David W. Wright, Robert A. Richardson
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, EasyVVUQ, Goals, Background in UQ Techniques, Discussion and Conclusions)
- Peter V. Coveney
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, Goals, Discussion and Conclusions)
- Jan Węglarz
  - Review and editing of the paper
  - Coauthorship of the text of publication (Sections: Introduction)
- Jalal Lakhlili
  - Coauthorship of the text of publication (Sections: EasyVVUQ, Goals, Background in UQ Techniques, Example 3-Fusion)
- Wouter Edeling
  - Coauthorship of the text of publication (Sections: EasyVVUQ, Background in UQ Techniques)
- Vytautas Jancauskas
  - Coauthorship of the text of publication (Sections: EasyVVUQ, Discussion and Conclusions)
- **Bartosz Bosak**, Tomasz Piontek, Piotr Kopta
  - Coauthorship of the text of publication (Sections: EasyVVUQ, Goals)

- Robert C. Sinclair, Irina Chirca
  - Coauthorship of the text of publication (Section: Example 1-Materials)
- Daan Crommelin
  - Coauthorship of the text of publication (Section: Example 2-Ocean Circulation)
- Onnie O. Luk, Olivier Hoenen
  - Coauthorship of the text of publication (Section: Example 3-Fusion)
- Derek Groen, Diana Suleimenova, Hamid Arabnejad
  - Coauthorship of the text of publication (Sections: Example 4-Forced Migration)
- Michał Kulczewski
  - Coauthorship of the text of publication (Section: Example 5-Urban Air)

ADVANCED THEORY AND SIMULATIONS
www.advtheorysimul.com

Building Confidence in Simulation: Applications of EasyVVUQ

*David W. Wright, Robin A. Richardson, Wouter Edeling, Jalal Lakhlili, Robert C. Sinclair, Vytautas Jancauskas, Diana Suleimenova, Bartosz Bosak, Michal Kulczewski, Tomasz Piontek, Piotr Kopta, Irina Chirca, Hamid Arabnejad, Onnie O. Luk, Olivier Hoenen, Jan Węglarz, Daan Crommelin, Derek Groen, and Peter V. Coveney**

Validation, verification, and uncertainty quantification (VVUQ) of simulation workflows are essential for building trust in simulation results, and their increased use in decision-making processes. The EasyVVUQ Python library is designed to facilitate implementation of advanced VVUQ techniques in new or existing workflows, with a particular focus on high-performance computing, middleware agnosticism, and multiscale modeling. Here, the application of EasyVVUQ to five very diverse application areas is demonstrated: materials properties, ocean circulation modeling, fusion reactors, forced human migration, and urban air quality prediction.

1. Introduction

In order for the results of computational science to become widely accepted components of decision making processes, such as in medicine and industry, it is essential that we quantify the trust one can have in the model in question. Confidence can only be gained by ensuring not only that simulation codes are solving the correct governing equations (validation), but they are solving them correctly (verification) and we have a comprehensive estimate of the uncertainties in the result uncertainty quantification).[1,2] Collectively the processes involved in evaluating our level of trust in the results obtained from models are known as VVUQ. While the need for rigorous model assessment is widely acknowledged, it is far from being universally implemented within the scientific literature. The reasons for this are wide ranging, but include lack of specialist knowledge of VVUQ techniques and, until recently, the difficulty in obtaining sufficient computational power to perform the necessary sampling in large scale simulations.

We have recently developed EasyVVUQ,[3] a package designed to help leverage recent advances in the scale of computational resources to make state of the art VVUQ algorithms available and accessible to a wide range of computational scientists. EasyVVUQ is a component of the VECMA open source toolkit (http://www.vecma-toolkit.eu), which provides tools to facilitate the use of VVUQ techniques in multiscale, multiphysics applications.[4]

In order to enable straightforward computations of EasyVVUQ scenarios on HPC resources, the tool has been designed to work with a variety of middleware technologies, such as FabSim3[5] or

Dr. D. W. Wright, Dr. R. A. Richardson, Dr. R. C. Sinclair, I. Chirca, Prof. P. V. Coveney
Centre for Computational Science
Department of Chemistry
University College London
London WC1H 0AJ, UK
E-mail: p.v.coveney@ucl.ac.uk

Dr. W. Edeling, Prof. D. Crommelin
Centrum Wiskunde & Informatica
Science Park 123, Amsterdam 1098 XG, The Netherlands

Dr. J. Lakhlili, Dr. O. O. Luk, Dr. O. Hoenen
Max-Planck Institute for Plasma Physics, Garching
Boltzmannstraße 2, Garching bei München 85748, Germany

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/adts.201900246

DOI: 10.1002/adts.201900246

B. Bosak, M. Kulczewski, T. Piontek, P. Kopta
Poznań Supercomputing and Networking Center
ul. Jana Pawła II 10, Poznań 61-139, Poland

Dr. D. Suleimenova, Dr. H. Arabnejad, Dr. D. Groen
Brunel University London
Uxbridge UB8 3PH, UK

Prof. D. Crommelin
Korteweg-de Vries Institute
University of Amsterdam
Science Park 105-107, Amsterdam 1098 XG, The Netherlands

Prof. P. V. Coveney
Informatics Institute
University of Amsterdam
Amsterdam 1090 GH, Netherlands

Dr. V. Jancauskas
Leibniz Supercomputing Centre
Boltzmannstraße 1, Garching bei München 85748, Germany

Prof. J. Węglarz
Institute of Computing Science
Poznan University of Technology
Piotrowo 2, Poznań 60-965, Poland

QCG.[6] The integration with pilot job mechanisms, in particular with QCG-PilotJob[7] and Dask JobQueue, allowed us to bypass limitations of regular queuing systems related to the scheduling of workloads composed of a very large number of relatively short tasks.

In this paper, we showcase the use of EasyVVUQ in a selection of applications chosen to have highly varied computational and VVUQ requirements. The examples come from a wide range of domains; materials science, climatology, fusion physics, forced population displacement, and environmental science. All of the examples come from active research projects and were chosen to highlight the range of capabilities of EasyVVUQ:

1. Materials—a simple parameter sweep performed using a computationally expensive molecular dynamics simulation;
2. Ocean circulation— estimation of Sobol sensitivity indices using stochastic collocation in a differential equation based model;
3. Fusion—estimation of Sobol sensitivity indices using the polynomial chaos expansion in a multiscale simulation workflow;
4. Forced migration—estimation of Sobol sensitivity indices in an agent based model;
5. Environmental—estimating uncertainties using stochastic collocation in a model forecasting urban air quality.

2. EasyVVUQ

EasyVVUQ is a Python library, developed within the VECMA project, designed to simplify the implementation of VVUQ workflows in new or existing applications. The library is designed around a breakdown of such workflows into four distinct stages; sampling, simulation execution, result collation (or aggregation), and analysis. In the sampling stage, the uncertainty on the inputs of the model are defined, for instance, by specifying independent probability density functions $p(\xi_i)$ for each model parameter ξ_i. This leads to a sampling plan, that is, a collection of points in the input space $\boldsymbol{\xi}$ where the model must be executed. This execution stage is deemed beyond the remit of the package (it can be handled for instance by Dask JobQueue, FabSim3,[5] QCG-PilotJob[8], RADICAL Cybertools,[9] etc.) but EasyVVUQ does provide some functionality to address it. The final collation and analysis stages, which are handled by EasyVVUQ, deal with post processing the simulation outcomes into mean predictions, uncertainty estimates, and sensitivity measures.

A common object, the Campaign, contains information on the application being analyzed alongside the runs mandated by the sampling algorithm being employed, and is used to transfer information between each stage. All applications outlined below share a similar Campaign creation step, up until the point where a specific sampler and input uncertainties are selected. This general procedure consists of creating an EasyVVUQ Campaign object, defining the parameter space and code outputs, and selecting an encoder, decoder, and collation element. The following code can be used as a generic template for all applications we consider (up to sampler selection), where variables indicated by $< \cdot >$ have to be replaced with application-specific values.

```
# import EasyVVUQ
import easyvvuq as uq

# Set up a fresh EasyVVUQ campaign
my_campaign = uq.Campaign(name="<campaign_name>", work_dir=<"path_work_directory">)

# Define parameter space for the uncertain parameters (2 in this case)
params = {
    "param_1": {
        "type": "<type_param_1>",   #e.g. "float"
        "min": <min_param_1>,
        "max": <max_param_1>,
        "default": <default_value_param_1>},
    "param_2": {
        "type": "<type_param_2>",
        "min": <min_param_2>,
        "max": <max_param_2>,
        "default": <default_value_param_2>},
    "out_file": {
        "type": "string",
        "default": "<outfile_filename>"}}

```

```
# The name of the output file, and the columns containing
# the quantities of interest (qois)
output_filename = params["out_file"]["default"]
output_columns = ["<qoi_1>", "<qoi_2>", "<qoi_3>", "<qoi_4>"]

# Create an encoder, decoder and collation element
encoder = uq.encoders.GenericEncoder(
    template_fname= "<path_to_input_template>",
    delimiter='$',
    target_filename="<input_filename>")
decoder = uq.decoders.SimpleCSV(target_filename=output_filename,
                                output_columns=output_columns,
                                header=0)
collater = uq.collate.AggregateSamples()

# Add the app (automatically set as current app)
my_campaign.add_app(name="<app_name>",
                    params=params,
                    encoder=encoder,
                    decoder=decoder,
                    collater=collater)
```

Most such variables are self explanatory, hence we only highlight

- `"<path_to_input_template>"`: This is the path to the input template of a particular application. Essentially, this is just the standard input file that the application uses, except that the value of uncertain variables (those in `params`), must be flagged by a delimiter (`$` in this case, e.g., `param_1=$param_1`), such that they will change their values for each sample.
- `"<input_filename>"`: This is the file name that will be given to each realisation of the input template.

Note also that the parameter space definition (in Listing 1) has optional specification of the type and minimum/maximum allowed values. EasyVVUQ's Cerberus dependency uses this information to apply verification of input variables such as type, range, and conditional checks.[10] EasyVVUQ additionally provides version checking for the library (and each of the component VVUQ elements) so that the user is made aware when a given element they have been using in the past may now have a new algorithm/behavior. This functionality, along with detailed logging of element application and "fail-early" checks, is intended to aid the user in verifying that a VVUQ workflow is doing what was intended.

For more information on the various EasyVVUQ elements, we refer to the software release publication.[3] The following sections detail the use of EasyVVUQ as applied to a variety of different application domains. All can be considered to have gone through the Campaign creation process as described above, hence we will not repeat this setup code. Only the assignment of input distributions, sampler selection, and post-processing will be described for each example application. Relevant information, code, and output data for the following example applications may be found in Supporting Information.[11]

3. Goals

We intend, through the five following example sections, to demonstrate how EasyVVUQ can be used to augment existing applications with VVUQ features or capabilities, notably:

A) In a non-intrusive manner (all solvers may be used as "black boxes", with no changes to their internals). This applies to all five example applications.
B) Favouring consistency and interoperability between approaches (a particular UQ approach may be painlessly swapped for another due to EasyVVUQ's standard interface for VVUQ elements). In this work, we demonstrate a basic parameter sweep (Section 5), stochastic collocation (Sections 6, 8, and 9), and polynomial chaos expansion (Section 7), showing a similar pattern of application.
C) Combining VVUQ elements together into single elements (to create complex) behavior easily using small, existing parts). This is demonstrated with Encoders in the Fusion (Section 7) and UrbanAir (Section 9) applications.
D) Allowing execution of generated runs in any order, using any desired middleware (of particular importance to HPC applications, where job submission and execution patterns are key to performance and highly dependent on the computing resources). This design principle is demonstrated by the mix of execution methods used in the five example applications, ranging from local or manual execution through to dynamic pilot job schedulers.

The focus is not on the scientific results of each section, but on the consistency of the approach when applied to different techniques, for different solvers from different scientific domains. EasyVVUQ seeks to abstract out both the underlying model (with its application specific inputs and execution needs) and the

implementation of VVUQ (particularly UQ) algorithms. These algorithms, which may be custom implementations in EasyVVUQ or sourced from existing libraries such as chaospy or SAlib, all interact via standardized interfaces, such that the user should not have to worry about the provenance of the underlying implementation, but rather about connecting the operations together (or swapping them for others).

4. Background in UQ Techniques

This section gives a brief background in the various UQ techniques which are used in the example applications.

4.1. Stochastic Collocation

Once an input distribution is defined, the output quantities of interest (QoIs) become random variables. The stochastic collocation (SC) method creates a polynomial approximation of a quantity of interest q in the stochastic space $\boldsymbol{\xi} \in \mathbb{R}^d$ via the following expansion:

$$q(\boldsymbol{\xi}) \approx \sum_{j=1}^{N_p} q_j(\boldsymbol{\xi}_j) L(\boldsymbol{\xi}) \tag{1}$$

Here, the stochastic space $\boldsymbol{\xi}$ is the space of the uncertain code input parameters, for which independent, user-specified, probability density functions (pdfs) must be provided: $\xi_i \sim p(\xi_i)$, $i = 1, \cdots, d$. Furthermore, q_j are the code samples which are computed on a structured multi-dimensional grid, and N_p is the total number of collocation points, that is, the total number of code evaluations. The samples q_j are interpolated to an arbitrary point within the stochastic space $\boldsymbol{\xi}$ by means of Lagrange interpolation polynomials $L(\boldsymbol{\xi})$. For interpolation in multiple dimensions ($d > 1$), $L(\boldsymbol{\xi})$ is built as a tensor product of 1D Lagrange polynomials. The SC method, and similarly the polynomial chaos expansion (PCE) method (described briefly in the next section), are well-known and we refer to ref. [12] for more details on these techniques. Suffice it to say that the tensor product construction yields an exponential increase in N_p with the number of uncertain variables d and the chosen polynomial order, an example of the familiar "curse of dimensionality." However, for moderate values of d, the SC and PCE methods can display exponential convergence with N_p, thereby outperforming Monte Carlo sampling.[12]

There are three main uses of the SC expansion (1). First, the N_p code samples q_j can be used to estimate the first two moments of q in the stochastic space, giving a mean prediction and an estimate of the output uncertainty due to the prescribed distributions on the inputs. Second, (1) acts as a computationally inexpensive surrogate model for the code. Using the Lagrange polynomials, the code samples q_j (evaluated at a specific parameter values $\boldsymbol{\xi}_j$), can be interpolated to an unsampled location $\boldsymbol{\xi}$. Finally, the SC expansion is amenable to variance-based global sensitivity analysis. Estimates of the well-known Sobol sensitivity indices can be obtained from Equation (1) as a post processing step, (which is outlined in Section 4.3).

4.2. Polynomial Chaos

The PCE method is an expansion technique that is closely related to SC method presented in Section 4.1. Whereas in SC we build Lagrange interpolation functions for known coefficients, in PCE we estimate coefficients for known orthogonal polynomial basis functions. Here, we can approximate the quantity of interest q with the following expansion:

$$q(\boldsymbol{\xi}) \approx \sum_{j=1}^{N_p} c_j P_j(\boldsymbol{\xi}) \tag{2}$$

In this equation, $\boldsymbol{\xi}$, c_j, and N_p are the uncertain parameter, expansion coefficients, and number of expansion factors,[13,14] respectively. The polynomials P_j are chosen such that they are orthogonal to the input distributions, which differ from the SC expansion in Equation (1).

To compute the c_j coefficients, two variants have been implemented: spectral projection and linear regression. In the spectral projection variant, we project the response against each basis function (composed of the polynomials set (P_j)) and we exploit their orthogonality properties to extract each coefficient. In the linear regression variant (also known as point collocation), we use a least squares method that minimizes a normed difference between the PC expansion and the output for a set of samples; the coefficients c_j are then the solution of the resulted linear system.[12]

By using the PCE method, like the SC method, we can obtain the statistical moments (mean, standard deviation, variance, and $(100 - x)^{\text{th}}$ percentile) of the quantities of interest, and we can also provide a global sensitivity analysis in the form of Sobol indices (which is outlined in the next section).

4.3. Sobol Indices

Sobol indices are variance-based sensitivity measures of a function $q(\boldsymbol{\xi})$ with respect to its inputs $\boldsymbol{\xi} \in \mathbb{R}^d$.[15] As in the case of the SC method, an independent probability density function $p(\xi_i)$ is assigned to each input ξ_j, which makes this a global method. Local sensitivity methods, on the other hand, measure the sensitivity of q at some point $\boldsymbol{\xi}_0$ in the domain, and are uninformative away from this point. Another advantage of global methods is that they can capture the sensitivity due to higher-order interactions (several parameters changing at once).

Sobol indices are derived from the analysis of variance (ANOVA) decomposition of $q(\boldsymbol{\xi})$. This decomposes q into a sum of basis functions of increasing input dimension, which in long forms reads as:

$$\begin{aligned} q(\boldsymbol{\xi}) = {} & q_\emptyset + q_1(\xi_1) + \cdots + q_d(\xi_d) + q_{12}(\xi_1, \xi_2) + q_{13}(\xi_1, \xi_3) \\ & + \cdots + q_{d-1\,d}(\xi_{d-1}, \xi_d) + \cdots + q_{1\ldots d}(\xi_1, \cdots, \xi_d) \end{aligned} \tag{3}$$

A more concise notation is

$$q(\boldsymbol{\xi}) = \sum_{u \subseteq \mathcal{F}} q_u \tag{4}$$

where u is a multi-index and $\mathcal{F}$ is the power set of $\mathcal{U} := \{1, 2, \cdots, d\}$. Let us define ξ_u as $\xi_u := \{\xi_i | \forall i \in u\}$, that is, the set of all inputs with an index in u. Furthermore, u' is the complement of u, that is, $u \cup u' := \mathcal{U}$ and $u \cap u' = \emptyset$.

In the ANOVA decomposition, the basis functions q_u satisfy the following properties:

$$\int q_u(\xi_u)\mathrm{d}p(\xi_u) = 0, \quad \text{if } u \neq \emptyset$$

$$\int q_u(\xi_u)q_v(\xi_v)\mathrm{d}p(\xi_u \cup \xi_v) = 0, \quad \text{if } u \neq v \tag{5}$$

that is, they have zero mean and are orthogonal when integrated over the distributions. These properties hold when the basis functions are defined as

$$q_\emptyset = \int q(\boldsymbol{\xi})\mathrm{d}p(\boldsymbol{\xi})$$

$$q_u = \int q(\boldsymbol{\xi})\mathrm{d}p(\xi_{u'}) - \sum_{w \subset u} q_w(\xi_w) \tag{6}$$

It is perhaps more clear to write this in terms of conditional expectations:

$$q_\emptyset = \mathbb{E}[q]$$

$$q_i = \mathbb{E}[q \mid \xi_i] - q_\emptyset$$

$$q_{ij} = \mathbb{E}[q \mid \xi_i, \xi_j] - q_i - q_j - q_\emptyset$$

$$\cdots \tag{7}$$

Hence, $q_\emptyset$ represents the mean of $q(\boldsymbol{\xi})$, and the q_i basis functions represent the effect of varying a single parameter ξ_i, minus the mean. Basis functions such as q_{ij} capture the effect of changing ξ_i and ξ_j simultaneously, minus all lower-order interactions, etc.

Therefore, the variances of these basis functions are the sensitivity measures we aim to approximate. Since the q_u have a zero mean, these are defined as

$$D_u := \mathbb{V}\mathrm{ar}[q_u] = \int q_u^2 \mathrm{d}p(\xi_u), \quad u \neq \emptyset \tag{8}$$

Using the orthogonality property of the basis functions, (8) can be rewritten as

$$D_u = \int \left(\int q(\boldsymbol{\xi})\mathrm{d}p(\xi_{u'}) \right)^2 \mathrm{d}p(\xi_u) - \sum_{w \subset u} D_w \tag{9}$$

Expression (9) allows us to compute all D_u in increasing order, if we can compute the first integral on the right-hand side. The authors of ref. [16] developed a method to approximate this integral using the SC expansion (1) for $q(\boldsymbol{\xi})$, and similar techniques exist for the PCE method.[17] It is out of the scope of the current paper to go into detail, and we refer the interested reader to refs. [16, 17] for the mathematical details. Essentially, once all the code samples are obtained, the Sobol indices, which are defined as

$$S_u := \frac{D_u}{D} \tag{10}$$

can be approximated in a post-processing step. Here, $D := \mathbb{V}\mathrm{ar}[q] = \sum_{u \subseteq \mathcal{U}} D_u$.[15] Note that all D_u are positive, and that the sum of all possible S_u equals 1. Each D_u measures the amount of variance in the output q that can be attributed to the parameter combination indexed by u.

5. Example 1—Materials

5.1. Application Outline

Molecular dynamics (MD) simulations are often used to investigate the properties of materials,[18] including as part of multiscale material prediction applications.[19] Here, we take a well understood soft-matter system and study how calculations of its Young's modulus (stiffness) using MD can vary with the system size and starting configuration.

The system under consideration (**Figure 1**) is an epoxy resin—epoxy tetraglycidyl methylene dianiline (TGMDA) cured with polyetheramine (PEA) in a 1:1 ratio. Epoxies are thermosetting polymers. Small reactant monomer molecules have several reactive sites which create strong covalent bonds between several other molecules, forming a dense network of crosslinks. The resulting polymer network is very strong and epoxies are widely used in manufacturing, in the aerospace industry, as adhesives, and as multipurpose insulators.

MD simulations in the condensed phase are almost always periodic, which means that we only simulate a comparatively small simulation cell to approximate the bulk properties. The size of this simulation cell has many implications for computational cost and, more importantly, the scientific results it furnishes. Finite size effects, self-interaction across periodic boundary boundaries, and thermal fluctuations in small systems can all affect the simulation's outcome. We measure the Young's modulus (YM) of an epoxy system by measuring the pressure exerted along one axis before and after a small strain.

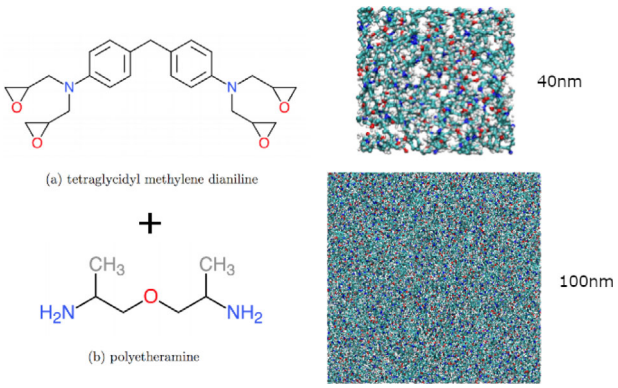


Figure 1. We test the effect of simulation size on the Young's modulus of an epoxy resin made from a 1:1 mixture of the monomers shown here.

5.2. VVUQ Algorithm

Since the instantaneous pressure of a molecular dynamics simulation can fluctuate by several GPa, it is necessary to average this value over a long sample period to measure the change in pressure due to an applied strain. The YM could also be affected by starting velocities of the atoms in the system, and the configuration of the epoxy network. To measure the system size dependence of all of these potential sources of variance, we design an EasyVVUQ Campaign that will take samples across each variable. Then, bootstrap analyses will measure their effect on the YM. A closer look at the variance due to each variable will show which is the most significant.

5.3. Execution Pattern

This application makes use of the BasicSweep sampler in EasyVVUQ, which recursively carries out a parameter sweep across the range of allowed values specified for each input variable. The system size is limited by computational cost at the high end, and the system stability at the lower. In this case, we know these approximate limits beforehand, so choose the specific range we want to sample using this method. The sampler is set up like this:

```
    sweep = {
        "box_size": [30, 40, 60, 80, 100, 150],
        "structure_no": list(range(1, 11)),
        "replica_no": list(range(1, 11)),
    }
    sampler = uq.sampling.BasicSweep(sweep=sweep)
```

In the above, we create a sampling element to sample across 6 simulation sizes, build 10 epoxy networks (structures) at each size, then measure the YM for each network starting from 10 different snapshots. These numbers are somewhat arbitrary, and more parameters could be swept depending on availability of computational resources.

Building the epoxy networks is done with an in-house developed script,[20] used in ref. [21] Simulating the epoxy network is accomplished using LAMMPS.[22] The execution of the system building procedure and measurement simulations are submitted on a remote computing resource. The "restart campaign" functionality of EasyVVUQ is required here, as the sampling and analysis stages were performed in separate python scripts. This ability to restart a campaign from a different script is useful in cases where, for example, the runs are expected to take a long time on a remote computing resource, and the user cannot or does not wish to have an EasyVVUQ script running locally, waiting for such jobs to finish.

Each replica generates three values for the YM, measured by separately straining along each principle axis of the polymer simulation. So that we can treat each of these values as equivalent measurements, we change the results pandas DataFrame format to have one row for each value; all YM values are in one column which makes some analysis more straightforward. This mode may be set using

```
campaign.set_collated_dataframe_format('one_row_per_var').
```

5.4. Results and Analysis

The system can be characterized by simply employing a boostrap analysis of the campaign.

```
bootstrap = uq.analysis.EnsembleBoot(groupby=["box_size"], qoi_cols=["Value"])
campaign.apply_analysis(bootstrap)
bootstrap_results = campaign.get_last_analysis()
```

The results of the above analysis are shown in **Figure 2**, along with a histograms of all measured YMs associated with each box size. We can clearly see that the average YM is independent of simulation size, above 4 nm. There is approximately a 20% increase in YM for a system size of 3 nm. We can safely say that for this system the characteristic length is therefore less than 4 nm.

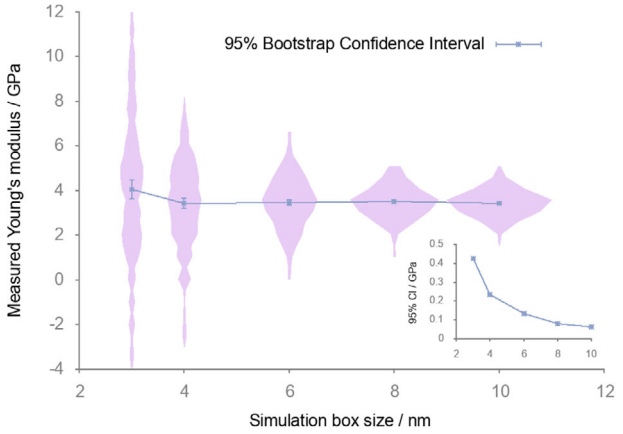


Figure 2. Young's modulus of an epoxy resin measured with different simulation sizes. Each point is the average of 300 simulations, which make up the pink histograms for each box size. The 95% bootstrap confidence interval for each simulation size is shown clearly in the bottom right insert.

We would like to know if the structure of an epoxy network has a significant bearing on the YM of a system, that is, if there is a large variation in the expected YM given an epoxy network. We approach this with the law of total variance

$$\mathrm{Var}(\mathrm{YM}) = \mathrm{Var}[\mathrm{E}(\mathrm{YM}|\#)] + \mathrm{E}[\mathrm{Var}(\mathrm{YM}|\#)] \tag{11}$$

where "#" is used to denote a specific network of cross-links. We can calculate the first and third terms of this law (the total variance in YM, and the expected variance given a specific structure) by some straightforward manipulation of the campaign results DataFrame.

```
# Retrieve the results DataFrame so we can manipulate here
results = campaign.get_collation_result()[["box_size","structure_no","Value"]]

total_var                 = results.groupby("box_size").var()["Value"]
var_per_structure         = results.groupby(["box_size","structure_no"]).var()
expected_var_given_struct = var_per_structure.groupby("box_size").mean()["Value"]
var_due_to_structure      = total_var - expected_var_given_struct
```

Detailed results are shown in Supporting Information;[11] however, the analysis shows that $\mathrm{Var}[\mathrm{E}(\mathrm{YM}|\#)] \ll \mathrm{E}[\mathrm{Var}(\mathrm{YM}|\#)]$. Therefore, the epoxy network structure has no significant effect on the YM. The variance is due to the inefficient sampling of MD. We studied low strains (0.5%) because epoxies are often brittle above these strains, but simulating further (into plastic deformation) could resolve the dependence on the network structure. Chaotic dynamical systems may manifest a pathology of IEEE floating point arithmetic which was hitherto unknown,[23] providing a potentially interesting overlap between uncertainty quantification and verification in affected systems.

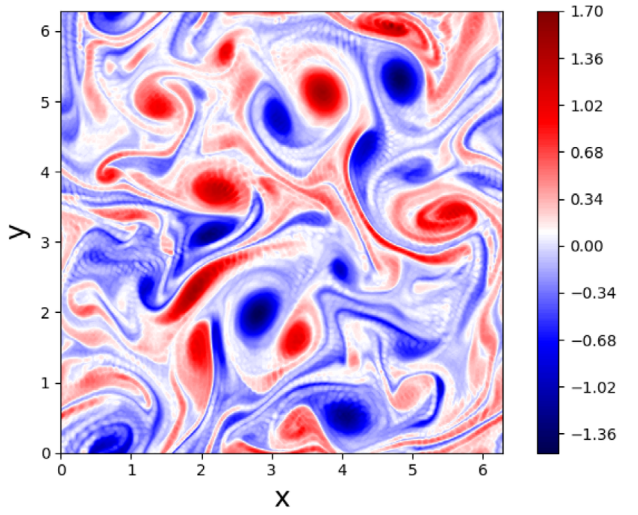


Figure 3. A snapshot of vorticity contours from Equation (12) with fully periodic boundary conditions, solved on a numerical grid of 256×256 points.

6. Example 2—Ocean Circulation

6.1. Application Outline

In this section, we consider the forced-dissipative vorticity equations for 2D incompressible flow (as described in Verkley et al.[24]), used as a simplified study for the general circulation in the oceans. The governing equations are

$$\frac{\partial \omega}{\partial t} + J(\Psi, \omega) = \nu \nabla^2 \omega + \mu(F - \omega)$$
$$\nabla^2 \Psi = \omega \tag{12}$$

Here, ω is the vertical component of the vorticity, defined from the curl of the velocity field $\mathbf{V}$ as $\omega := \mathbf{e}_3 \cdot \nabla \times \mathbf{V}$, where $\mathbf{e}_3 := (0, 0, 1)^T$. The stream function Ψ relates to the horizontal velocity components by the well-known relations $u = -\partial\Psi/\partial y$ and $v = \partial\Psi/\partial x$. The non-linear advection term is defined as

$$J(\psi, \omega) = \frac{\partial \Psi}{\partial x}\frac{\partial \omega}{\partial y} - \frac{\partial \Psi}{\partial y}\frac{\partial \omega}{\partial x} \tag{13}$$

This system generates flow fields such as those shown in **Figure 3**, which depicts a snapshot of the vorticity ω.

As in ref. [24], the forcing term is chosen as the single Fourier mode $F = 2^{3/2}\cos(5x)\cos(5y)$. The system is fully periodic in the x and y directions over a period of $2\pi L$, where L is a user-specified length scale, chosen as the Earth's radius ($L = 6.371 \times 10^6[m]$). The inverse of the earth's angular velocity Ω^{-1} is chosen as a time scale, where $\Omega = 7.292 \times 10^{-5}[s^{-1}]$. Thus, a simulation time period of a single "day" can now be expressed as $24 \times 60^2 \times \Omega \approx 6.3$ non-dimensional time units. Given these chosen length and time scales, we non-dimensionalize (12) and solve by using a spectral method with the second-order accurate AB/BDI2 time-stepping scheme.[25]

The viscosity ν and the forcing term coefficient μ are tunable parameters, and are typically set to a value such that the build up of grid-scale noise at the smallest resolved scale is prevented. In our example code, their values are computed such that a Fourier mode at this scale is exponentially damped with a user-specified e-folding time scale, that is, a time scale over which a decay of 63 % occurs ($1 - e^{-1}$). This leads to the following expressions for ν and μ:

$$\nu = \frac{1}{24 \cdot 60^2}\,\frac{1}{\Omega}\,\frac{1}{K^2}\,\frac{1}{\xi_1} \quad \text{and} \quad \mu = \frac{1}{24 \cdot 60^2}\,\frac{1}{\Omega}\,\frac{1}{\xi_2} \tag{14}$$

Here, K is the highest resolved wave number in our spectral method, which is fixed at 85. More important for our current discussion are ξ_1 and ξ_2, that is, the aforementioned damping time scales (expressed in days), which we treat as uncertain. We use EasyVVUQ to estimate the effect of this uncertainty on certain measures derived from the solution of (12). Our focus will be on the (time-dependent) energy E and enstrophy Z, defined as

$$E(t) := \frac{1}{2}\left(\frac{1}{2\pi}\right)^2 \int_0^{2\pi}\int_0^{2\pi} \mathbf{V}\cdot\mathbf{V}\,dx\,dy \quad \text{and}$$

$$Z(t) := \frac{1}{2}\left(\frac{1}{2\pi}\right)^2 \int_0^{2\pi}\int_0^{2\pi} \omega^2\,dx\,dy \tag{15}$$

Specifically, we are interested in the time-averaged statistical moments of the energy E and enstrophy Z; for example, our quantities of interest q take the form of

$$q = \int_{T_0}^{T} E(t)\,dt =: \overline{E} \quad \text{or} \quad q = \int_{T_0}^{T} \left(E(t) - \overline{E}\right)^2 dt \tag{16}$$

and likewise for the enstrophy. The integration interval $[T_0, T]$ will be defined later.

6.2. VVUQ Algorithm

For this particular problem, we will use the stochastic collocation method, as outlined in Section 4.1. In addition to the statistical moments of the aforementioned energy and enstrophy, the Sobol sensitivity indices of our damping time scales will serve as our QoIs as well. Specific implementation details are given next.

6.3. Execution Pattern

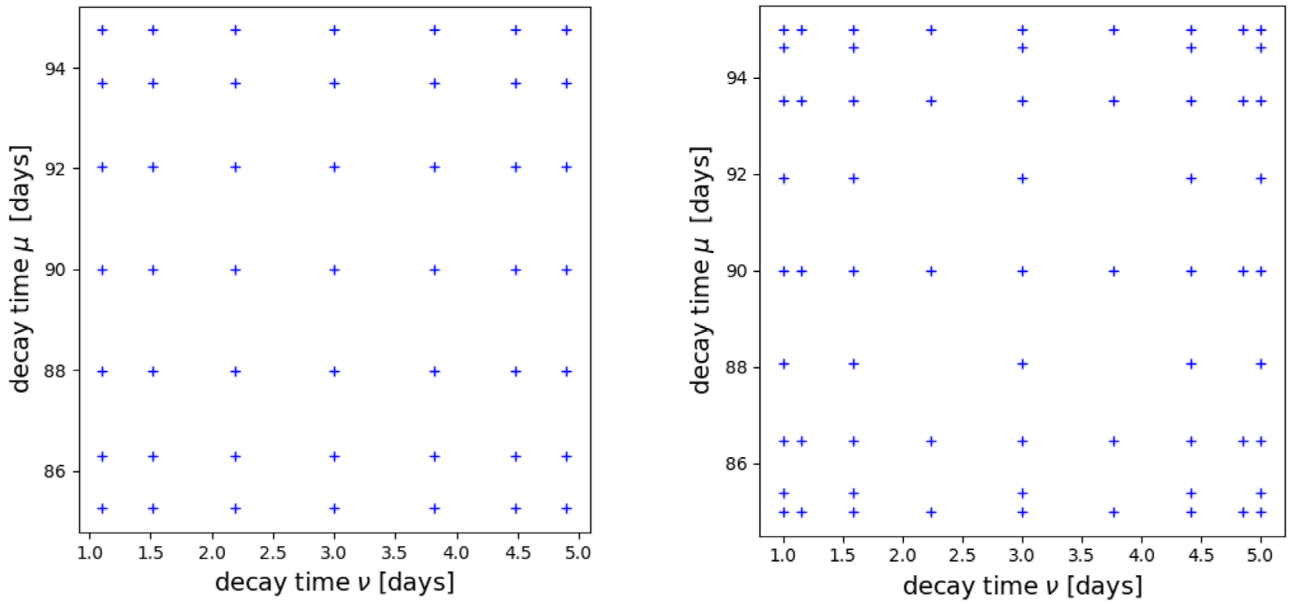
EasyVVUQ is designed to work with the Chaospy library,[26] for the specification of the input distributions. We will assume the following distributions for the uncertain decay times associated with ν and μ:

```
import chaospy as cp
vary_ocean = {"decay_time_nu": cp.Uniform(1.0, 5.0),
              "decay_time_mu": cp.Uniform(85.0, 95.0)}
```

That is, we assume that the viscous term ($\nu\nabla^2\omega$) in Equation (12) has a uniformly distributed uncertain decay time at the smallest retained scale between 1 and 5 days, whereas our forcing term is damped somewhere between 85 and 95 days. We then select the stochastic collocation sampler via

```
my_sampler = uq.sampling.SCSampler(vary=vary_ocean, polynomial_order=6),
```

By selecting a polynomial order of 6, a seven-point quadrature rule for each uncertain dimension is created. Hence, since we have two uncertain variables, we obtain a tensor grid of 49 points in the stochastic space, see **Figure 4**a. At each point, we have to evaluate the code solving (12). Instead of directly creating a full tensor product of the seven-point 1D quadrature rule, we can also construct a sparse grid (see Figure 4b), which uses a linear combination of tensor products of quadrature rules of different orders.[12] By using carefully chosen 1D quadrature rules, many points in the different tensor products will coincide, leading to a more efficient sampling plan in high dimensions. To switch to a sparse grid, one might use



(a) A full tensor grid. (b) A sparse grid.

Figure 4. Two stochastic collocation grids generated by EasyVVUQ. Each symbol is a point in the stochastic space at which the code solving Equation (12) must be evaluated.

```
my_sampler = uq.sampling.SCSampler(vary=vary_ocean, quadrature_rule="C",
                polynomial_order=3, sparse=True, growth=True)
```

Here, `quadrature_rule="C"` denotes the use of 1D Clenshaw–Curtis quadrature rules, which are a common choice in sparse grid constructions. Furthermore, `growth=True` selects an exponential growth rule, which ensures that the Clenshaw–Curtis rules are "nested" such that a quadrature rule of the next order contains all points of the previous order, leading to the aforementioned more efficient sampling plan in high dimensions. However, since we just have two uncertain variables here, we will use the full tensor product construction. Depending on the spatial resolution of the computational grid (in our case, we employ a 2D grid of 256×256 points), the cost of sampling Equation (12) at all collocation points $\boldsymbol{\xi}_j$ can be high. Moreover, since we are interested in the time-averaged statistics as in Equation (16), we must run each sample until convergence in these statistics can be safely demonstrated. We use FabSim3[5] to facilitate the execution of these samples in parallel on the Eagle supercomputer at the Poznan Supercomputing and Networking Center (PSNC). A FabSim3 plugin "FabUQCampaign" has been created to execute the ensemble run of EasyVVUQ samples on a remote resource, with minimal change in the code that is executed on the localhost. For a tutorial on the setup of FabUQCampaign, see ref. [27].

```
import fabsim3_cmd_api as fab
fab.run_uq_ensemble(my_campaign.campaign_dir, "ocean", machine="eagle_vecma")
```

The `fab` module is a wrapper around FabSim3 command-line instructions, such that these can be executed from within Python. Furthermore, `machine` specifies the name of the remote HPC resource (the PSNC Eagle cluster in our case), and `campaign_dir` is the directory containing the EasyVVUQ campaign. Finally, `"ocean"` is the name of the script which executes a single run of our model (12); see the tutorial[27] for more details. Once the ensemble run has completed, the results can be retrieved through:

```
fab.get_uq_samples(my_campaign.campaign_dir, machine="eagle_vecma")
```

If one wishes to run a (small) local ensemble for testing or debugging purposes, specifying `machine="localhost"` will make sure that everything is executed locally. Note that FabSim3 is not the only available execution interface between EasyVVUQ and HPC clusters. EasyVVUQ-QCGPilotJob is a lightweight integration code that simplifies usage of EasyVVUQ with a QCG-PilotJob execution engine; see ref. [7] for a tutorial.

6.4. Results and Analysis

In our example, $d = 2$ (ν and μ), and our quantities of interest are time-averaged moments of Equation (16) of the energy and enstrophy. For each sample of the ensemble run, Equation (12) is simulated for 11 years, and the last 10 years are used to compute the time-averaged E and Z moments. To perform the post-processing analysis of these samples, an `SCAnalysis` object is created:

```
#evaluate the energy surrogate at xi
sc_analysis.surrogate("E", xi)
```

```
sc_analysis = uq.analysis.SCAnalysis(sampler=my_sampler,
                                     qoi_cols=output_columns)
my_campaign.apply_analysis(sc_analysis)
results = my_campaign.get_last_analysis()
```

The `results` dictionary contains the statistical moments and the Sobol indices of the quantities of interest, the latter of which are given below for this particular case:

```
========================
Sobol indices E_mean
S(nu) = 0.6649
S(mu) = 0.3256
S(nu, mu) = 0.0096
========================
Sobol indices Z_mean
S(nu) = 0.7073
S(mu) = 0.2823
S(nu, mu) = 0.0103
========================
Sobol indices E_stdev
S(nu) = 0.4661
S(mu) = 0.0881
S(nu, mu) = 0.4458
========================
Sobol indices Z_stdev
S(nu) = 0.4971
S(mu) = 0.0775
S(nu, mu) = 0.4254
```

A value close to one means that this variable, or combination of variables, explains most of the variance in the selected output. Clearly, ν is the more influential parameter for both the time-averaged energy E and enstrophy Z. However, for the corresponding standard deviations (stdevs), μ does play an important role in the second-order Sobol index, indicating a significant interaction between ν and μ for these QoI.

In order to use the SC expansion as a surrogate model for the real code, we can draw random samples from Equation (1) via

Here, `xi` is an array containing a random sample from the input distributions of ν and μ. The surrogate is far cheaper than the original model, such that we can use it to evaluate the output probability density function via a kernel-density estimate (KDE). **Figure 5** shows the KDE of E, evaluated using 5×10^4 samples from (1).

7. Example 3—Fusion

7.1. Application Outline

Thermonuclear fusion is potentially a solution to the provision of base load electricity, which is carbon free and not subject to geopolitical problems. Understanding the mechanisms of heat

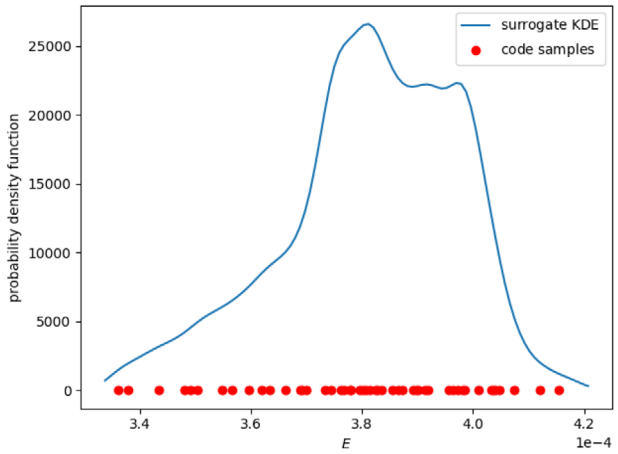


Figure 5. The kernel-density estimate of the time-averaged energy E, computed from 50 000 samples of a SC surrogate of polynomial order 6. The 49 code samples used to build the surrogate are also shown.

and particle transport in hot fusion plasma is one of the keys to obtain a cost-efficient reaction in the fusion devices. Our present understanding of the problem is that turbulence at small scales is responsible for much of this transport, but the profiles of temperature and density evolve over much larger scales.

A wide standardization effort toward integrated modeling[28] for fusion plasmas has allowed us to build modular applications in the form of a workflow. The code-to-code coupling is done via standardized data-objects[29] (referred to hereafter as CPO files), while specific parameters are stored in XML. This setup allows users to swap codes with others of different complexity. Based on this effort, a multi-scale application is developed to study the turbulence effects on plasma transport at larger scales.[30] However, much remains to be done on the validation of such simulations as well as on the control of their uncertainties. In this work, we present an early validation pattern we uncovered by extracting and comparing experimental and UQ simulation output distributions. In our application, these uncertainties originate from applied heating sources (extrinsic) and/or from the noisy, chaotic nature of the turbulence (intrinsic). We focus here on quantifying extrinsic uncertainties for the heating source as well as boundary conditions for both electron and ion temperatures. The heat source (energy per unit time) for each species is a Gaussian function with respect to the radial (or toroidal flux) coordinate ρ_{tor}, and it is characterized by its amplitude, width, and position. The boundary conditions refer to the initial temperatures at the plasma edge for both species; the edge is positioned at the maximum ρ_{tor} value, or at normalized $\rho_{tor} = 1.0$.

7.2. VVUQ Algorithm

The EasyVVUQ library provides both quasi-Monte Carlo (QMC) and polynomial Chaos expansion (PCE) methods (described in Section 4.2) that we can select from to conduct UQ and SA in the multi-scale fusion workflow.[31] In the work we present here, the PCE method was selected because it can carry out the calculations much faster than the QMC method. However, this is only valid if the number of uncertain parameters remain relatively low.

7.3. Execution Pattern

Similar to the ocean circulation example, we specify the input distributions using chaospy through EasyVVUQ. In addition, to fully benefit from the standardized interface for each code within our multi-scale workflow, we extended the EasyVVUQ base encoder with a new domain specific CPOEncoder (for boundary conditions of electron and ion temperature profiles) and a generic XMLEncoder (for electron and ion heating sources approximated by the amplitude, position and width of a Gaussian function). These format-bound encoders allow us to update real data and parameter files without having to create a template, which in turn gives us more flexibility. Since we are interested in uncertainties driven by both the heating sources and the boundary conditions for electrons and ions temperatures, we need to combine these two encoders with the MultiEncoder provided by EasyVVUQ. Therefore, the encoder creation from listing 1 is modified with the following snippet of code:

```
# Extended encoder for XML template
encoder_xml = XMLEncoder(template_filename="<path_to_input_xml_template>",
                         target_filename="<input_xml_file>",
                         common_dir=common_dir,
                         uncertain_params=uncertain_params)
# Extended encoder for CPO template
encoder_cpo = CPOEncoder(template_filename="<path_to_input_cpo_template>",
                         target_filename="<input_cpo_file>",
                         common_dir=common_dir,
                         uncertain_params=uncertain_params)
# Encoder to be used by Campaign object
encoder = uq.encoders.MultiEncoder(encoder_cpo, encoder_xml)
```

- `common_dir` is a folder that contains all required input files.
- `uncertain_params` is a python dictionary specified by the user, and it contains the list of parameters with their probability distributions types followed by the chaospy glossary.

In addition, the new encoders have a specific function that provides two dictionaries containing the names and types of all parameters to be varied and their corresponding distributions.

```
# Get parameters: vary = vary_1 + vary_2 and params = params_1 + params_2
params_1, vary_1 = encoder_xml.draw_app_params()
params_2, vary_2 = encoder_cpo.draw_app_params()
```

We set up the PCE sampler using a polynomial of order 4 to ensure good accuracy:

```
# Create the sampler
sampler = uq.sampling.PCESampler(vary=vary, polynomial_order=4)
```

The output of the application is composed of several CPO format files, so the same kind of modification is done for the creation of the decoder, which uses our domain-specific CPODecoder.

```
# Create a specific decoder for CPO output files
decoder = CPODecoder(target_filename="<output_cpo_file>",
                     output_columns=<quantities_of_interest_list>)
```

Finally, to generate all samples needed for the analysis, we can either call the function `ExecuteLocalexecute` provided directly by EasyVVUQ or resort to a wrapper enabling the execution using the QCG-PilotJob mechanism.[7,8]

7.4. Results and Analysis

To perform a post-processing analysis on the generated samples, we use the `PCEAnalysis` object from EasyVVUQ. For the results, as in the ocean circulation example, we use `analysis_results`, the output dictionary of the campaign object's `get_last_analysis()` method, in which the statistical moments, and the Sobol indices of the Quantities of Interest are stored.

The current version of the fusion workflow uses an analytical turbulence code, with four uncertain parameters (amplitude, width, position of heating source, and boundary condition). We assumed each of these parameters has a normal distribution in the range of ±20% around its original value, and as the number of samples is determined by the uncertain parameter number and polynomial degree in the PCE method, the number of runs required for this example is 1296.

The uncertainty quantification of the fusion workflow is shown in **Figures 6** and **7**. The quantities of interest are the electron and ion temperature profiles, spanning from the radial position of plasma core ($\rho_{tor} = 0$) to the edge (normalized $\rho_{tor} = 1.0$). The standard deviation indicates that the ion temperature varies weakly since the uncertainties are carried by the electrons sources. The sensitivity analysis reveals that the variance in the electron and ion temperatures is mainly due to the uncertainty from three parameters: the position and amplitude parameters of the sources at core region of the plasma and, as expected, boundary condition parameter at the edge region. The parameter width has no direct effect on the variance of the two quantities, so according to ref. [32], this parameter can be neglected and then the number of samples can be reduced while keeping the same variance behavior.

In addition to uncertainty quantification, the fusion application performs validation on the simulation results by comparing the distribution of the QoI to the distribution from experimental measurements (first results are shown in the **Figure 8**). Specifically, we create the `ValidationSimilarity` object to determine the similarity between two distribution functions:

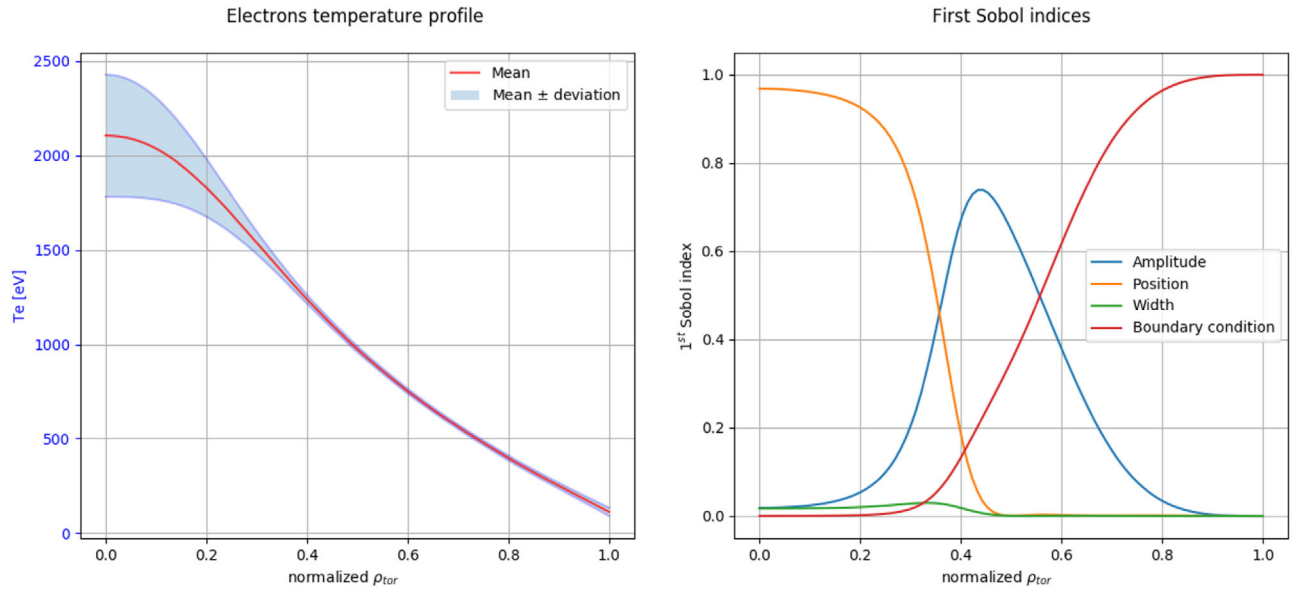


Figure 6. Descriptive statistics and sensitivity analysis of UQ example for the electron temperature.

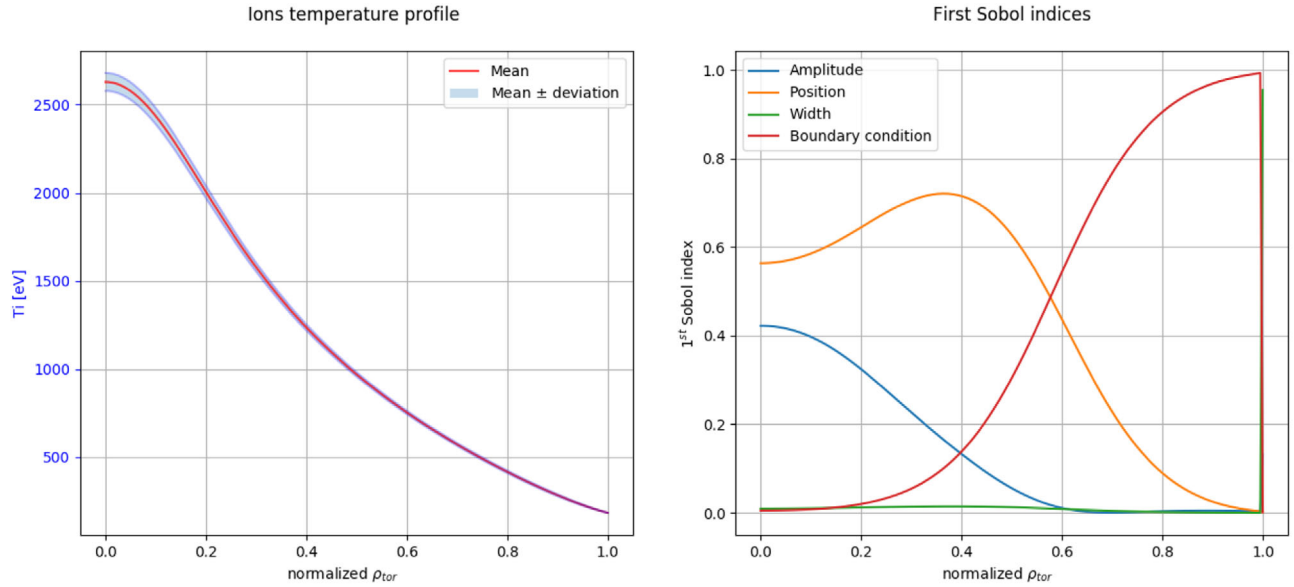


Figure 7. Descriptive statistics and sensitivity analysis of UQ example for the ion temperature.

```
# Create a specific Validation object
validation = uq.comparison.ValidationSimilarity()
validation_results = validation.compare(dataframe1=<exp_dist_values>,
                                        dataframe2=<sim_dist_values>)
```

- `exp_dist_values` is a list of approximated distributions of the given experimental samples. The samples obtained from fusion experiments contain the mean, lower, and upper threshold values; these lower and upper threshold values do not necessarily equal to each other. Therefore, we treat each sample as a two-piece normal distribution.

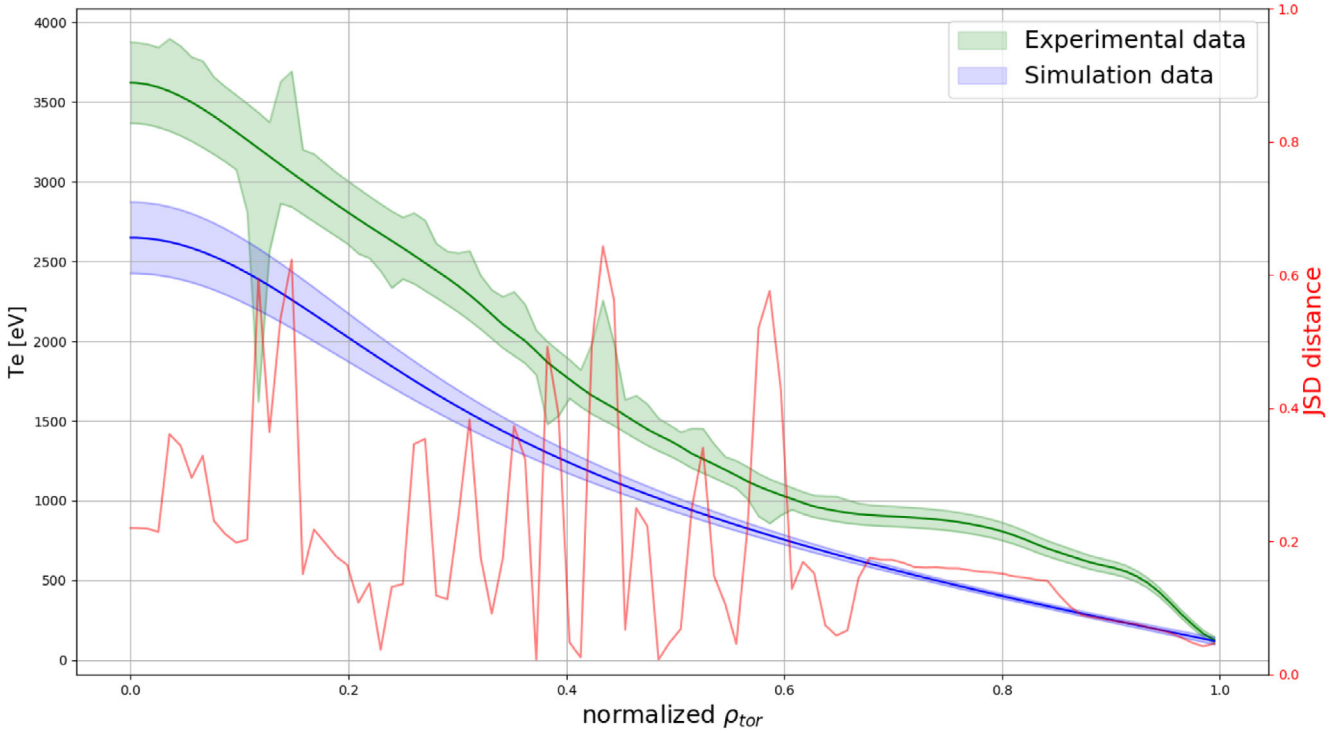


Figure 8. Validation using comparison between experimental and simulation data for electron temperatures. On the left hand-side axis, the expected values and standard deviations, and on the right hand-side axis, the Jensen–Shannon distance measuring the similarity between distributions with respect to the normalized toroidal flux coordinate ρ_{tor}.

- `sim_dist_values` is a list of output distributions given by Analysis results: `analysis_results[''output_distributions'']["QoI"]`. In EasyVVUQ, we use a function that constructs a kernel density estimator (KDE) for each polynomial by sampling it.

For the similarity measure, we use the Jensen–Shannon distance (JSD), which is a symmetrized and smoothed version of the Kullback–Leibler divergence.[33,34] It is defined by

$$JSD(P,Q) = \frac{1}{2}\sum_{i=1}^{Ns}\left(P_i \log(\frac{P_i}{M_i}) + Q_i \log(\frac{Q_i}{M_i})\right) \quad (17)$$

Here, Ns is the number of samples, P and Q are defined as two discrete probability distributions, and $M = \frac{1}{2}(P+Q)$. As presented in Figure 8, Jensen–Shannon distance takes values in the range $[0,1]$. The values closer to 0 indicate a smaller "distance" between the two distributions and therefore a stronger similarity. Two other measures based on the Hellinger and Wasserstein distances[33] are also available in EasyVVUQ. These measures were also tested on the current example, and they give equivalent results as the Jensen–Shannon distances.

8. Example 4—Forced Migration

8.1. Application Outline

Forecasting forced displacement is of considerable importance since 70.8 million people are today being forcibly displaced worldwide, a record level.[35] It is also challenging as many forced population data sets are small and incomplete, and data sources have too little information.[36] Nevertheless, forced population predictions are essential to save the lives of such migrants, to investigate the effects of policy decisions and to help complete incomplete data collections on forced population movements.

Through the use of computational approach, namely the FLEE agent-based simulation code, we predict the distribution of forced population arrivals to potential destinations as governments and NGOs can efficiently allocate humanitarian resources and provide protection to vulnerable people.[37] We represent forcibly displaced people as individual agents, combining simple rulesets for individuals to allow complex movement patterns to emerge and validate simulation results against real data. We are also able to systematically explore the possible impact of policy decisions using the FabSim3-based FabFlee toolkit while accounting for the sensitivity to a subset of parameters and assumptions in the model, such as the probability of migrants making specific moves. In **Figure 9**, we present a simulation instance of the Mali conflict, in which a number of insurgent groups began a fight for the independence of the Azawad region resulting in an increasing number of forcibly displaced people since January 16, 2012.[36,37]

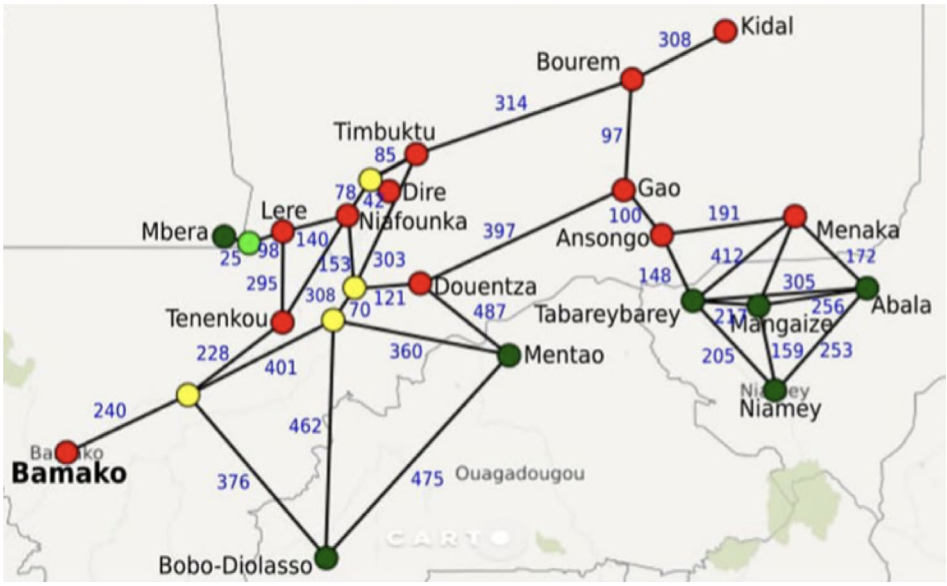


Figure 9. Overview of geographic network model for Mali, which includes conflict zones (red circles), camps (dark green circles), forwarding hub (light green circle), and other major towns (yellow circles) interconnected with straight-lines that represent roads and their length in kilometers with adjacent blue numbers.

8.2. VVUQ Algorithm

FabFlee uses the EasyVVUQ library to facilitate VVUQ for simulation analysis. It allows us to automate parameter exploration analysis and explore essential one-at-a-time input uncertainty quantification. Importantly, uncertainty quantification and sensitivity analysis are required in multiscale migration studies to understand in what regimes and scenarios our simulation approach performs well. FabSim3, EasyVVUQ, QCG-PilotJob, and other QCG components can be combined in a variety of ways, enabling users to combine their added values while retaining a limited deployment footprint. As previously mentioned, EasyVVUQ can use FabSim3 to facilitate automated execution. Users can convert their EasyVVUQ campaigns to FabSim3 ensembles using a one-line command, and the FabSim3 output is ordered such that it can be directly moved to EasyVVUQ for further decoding and analysis.

8.3. Execution Pattern

We use similar approach as described in the ocean circulation example for sensitivity analysis of forced migration application. In particular, we analyze the probability of parameters when agents move from their current location to a different one on a given day. These probabilities depend on the type of locations, namely conflict zone, camp, or other location, where agents reside.[37] We adjust these parameters to understand the importance of our assumptions in regard to the validation results. In **Figure 10**, we provide the overall workflow of forced displacement application for sensitivity analysis.

To provide the input distributions, for instance, we specify a uniformly distributed move chance probabilities for camps between 0.0001 and 1.0, as well as for conflict locations between 0.1 and 1.0 as illustrated below.

```
import chaospy as cp
vary = {"camp_move_chance": cp.Uniform(0.0001, 1.0),
        "conflict_move_chance": cp.Uniform(0.1, 1.0)}
```

Then, we set up the stochastic collocation (SC) sampler using

```
# Create the sampler
my_sampler = uq.sampling.SCSampler(vary=vary, polynomial_order=3)
```

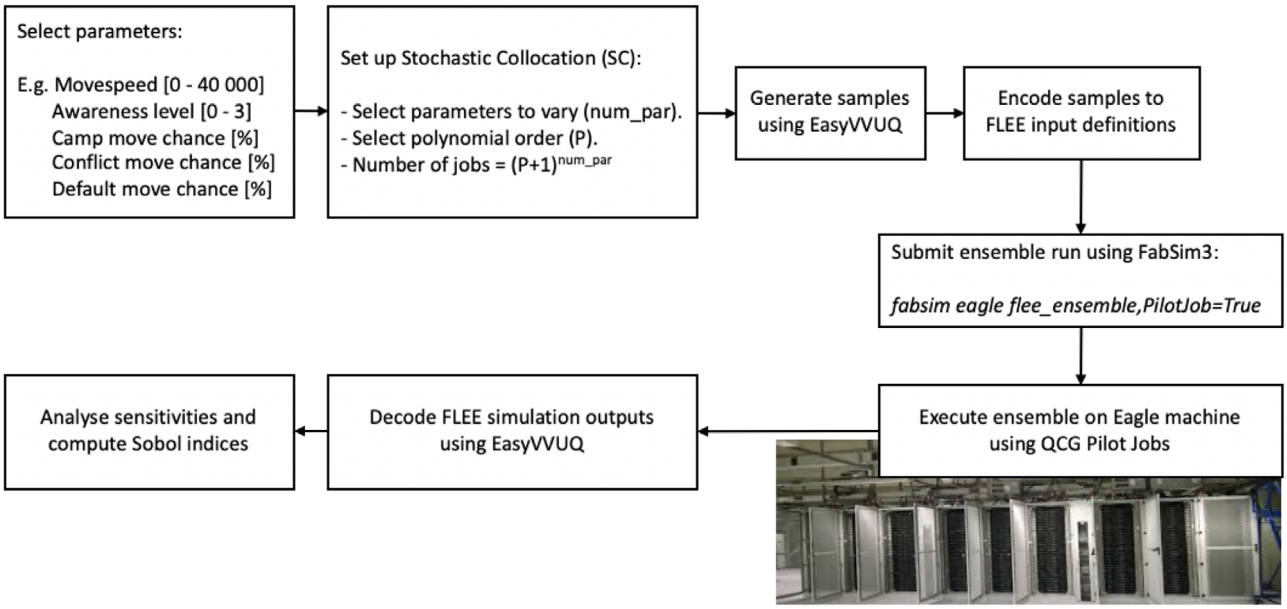


Figure 10. Overview of the FLEE workflow, where we use FabSim3 in conjunction with EasyVVUQ and QCG Pilot Job Manager.

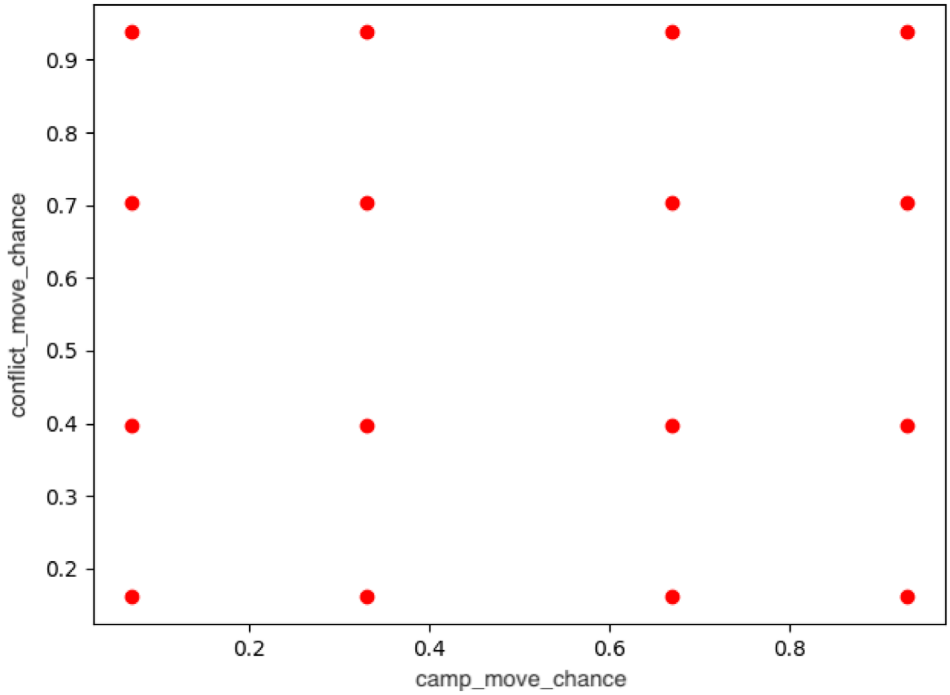


Figure 11. A stochastic collocation grid generated by EasyVVUQ for migration application parameters.

where a polynomial order is 3 in this instance. In turn, it creates a four-point quadrature rule for each move chance parameter (see **Figure 11**). EasyVVUQ encodes the generated samples to FLEE input definitions for specific conflict simulations and submits all ensemble runs for execution using FabSim3 to Eagle machine where QCG-PilotJobs schedule submitted ensemble runs and pre-reserved resources.

8.4. Results and Analysis

We apply the same SC-based Sobol index method[16] as in the ocean circulation example above to the Mali conflict, and obtain the results illustrated in **Table 1** for two parameters. We draw our own distinction on the Sobol indices by accepting parameters with values below 0.05 while identifying parameters with higher

Table 1. FabFlee and EasyVVUQ input parameter exploration results for multiscale migration application, where we vary two parameters.

Parameters	Sobol indices
camp_move_chance (0)	0.9497191
conflict_move_chance (1)	0.04978418
Combination of parameters (0, 1)	0.00049672

values as sensitive to output results. The camp_move_chance parameter is more sensitive in our model compared to the other parameter, namely conflict_move_chance, since camps are primary destination locations for forcibly displaced people fleeing from conflict locations. We also find that our models are not sensitive to the combination of these parameters.

9. Example 5—UrbanAir

9.1. Application Outline

The UrbanAir application concerns the modeling and forecasting of the concentration and dispersion of pollutants. It is a 3D multiscale model that combines a numerical weather prediction (NWP) model, running at larger scale (e.g., mesoscale), with a city-scale geophysical flow solver for accurate prediction of contaminant transportation through the street corridors, over buildings and obstacles.

The NWP model is based on the community Weather Research and Forecasting (WRF) model,[38] while the city-scale problem is solved using the EULAG model.[39] EULAG is a numerical solver for all-scale geophysical flows, with many proven scenarios, for example, flows around buildings[40] with comparison against wind tunnel experiments.[41] The coupling between WRF and EULAG model has been evaluated in ref. [42]. Typically, an emergency response situation requires fast and accurate tools. However, the use of more complex and expensive models is dictated by the need for accurate prediction of peak concentrations and plume temporal evolution.

With increased model resolution, small-scale flow characteristics are becoming more essential for prediction, and general urban parameterization coming from the NWP model is not enough. The WRF output is used as the initial and lateral boundary conditions for the EULAG simulation, along with terrain data (terrain elevation, road network, buildings shapes, and height) and emission data. **Figure 12** presents the general workflow of the application. The IMB approach is used in EULAG to explicitly resolve complex building structures, accounting for different urban aerodynamic features, such as channeling, vertical mixing, and street-level flow. The pollutant dispersion is simulated using passive tracer equations.

The NWP model may be supplemented with an additional chemistry module, to simulate chemical transportation and mixing over larger scales.[43,44]

In order to accurately simulate at small scales (grid resolutions up to 1 metre), HPC resources are required. EULAG is proven to scale up to thousands of CPU cores to support such resolution and to decrease overall time-to-solution.[45] The key problem in providing accurate forecasts is the lack of complete, well-known emission sources. Contaminants—such as NO, NO_2, PM2.5 (particulate matter under 2.5 μm in size), and PM10 in particular—are emitted by point sources (e.g., industrial chimneys), line sources (e.g., road transportation), and area

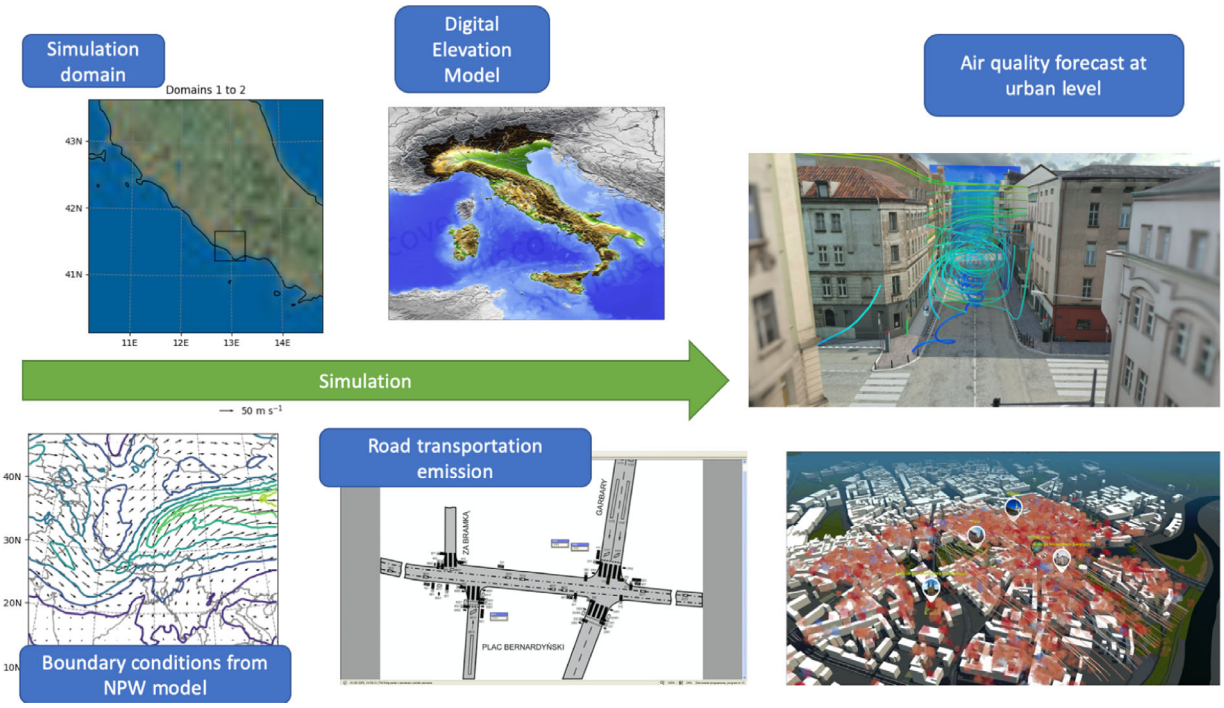


Figure 12. High-level UrbanAir application workflow.

sources (e.g., heat appliances). The uncertainty comes from unknown emission details. Taking road transportation as an example, there is a set of parameters that need to be estimated: these include the ratio of cars using gasoline to diesel fuel, fuel usage, emission index, percentage of cars that cold-started, and so on. Through the use of computational ensemble simulations, we can address these issues using statistical data, such as by combining the number of cars passing a given road section within 1 h with previously estimated parameter values.

9.2. VVUQ Algorithm

In order to assess the influence of unknowns in the emission sources, we have designed an EasyVVUQ campaign that samples across each of the input variable. It allows us to assess input uncertainty quantification and sensitivity analysis, though we concentrate on the former at the moment. The uncertainty may additionally stem from weather boundary conditions, heights of buildings, etc. To facilitate uncertainty quantification for this computationally demanding application, The QCG-PilotJob is used to choreograph the execution of the ensembles.

9.3. Execution Pattern

Currently, we focus on quantifying uncertainty coming from parameters related to NO_2 emission attributed to road transportation. The simulations require input data regarding, for example, NO_2 index from gasoline engines, fuel usage, density, and ratio of gas to diesel cars. The input distribution is specified using chaospy via EasyVVUQ. Here, we focus only on parameters related to petrol-powered vehicle, while a similar setup is needed for diesel vehicle analysis.

```
vary_gas_no2 = {
 "gas_engine": cp.Uniform(0.1, 0.9),
 "gas_usage": cp.Uniform(3.0, 13.0),
 "gas_density": cp.Uniform(0.001, 0.9),
 "gas_no2_index": cp.Uniform(0.01, 0.1),
}
```

Next we set up different samplers for different input parameters we want to be sampled.

```
emis_encoder = EmisEncoder(
    template_fname=jobdir + '/' + TEMPLATE,
    delimiter='$',
    target_filename=ENCODED_FILENAME)

wind_encoder = uq.encoders.GenericEncoder(
    template_fname = jobdir + '/' + WIND_TEMPLATE,
    delimiter='$',
    target_filename='wind.dat')

scalars_encoder = uq.encoders.GenericEncoder(
    template_fname = jobdir + '/' + SCALARS_TEMPLATE,
    delimiter='$',
    target_filename='scalars.dat')

fort13_encoder = uq.encoders.GenericEncoder(
    template_fname = jobdir + '/' + FORT13_TEMPLATE,
    delimiter='$',
    target_filename='fort.13')
```

A custom encoder is used, EmisEncoder, whose goal is to use the values of the sampled parameters as components to calculate the correct value of road transportation emissions.

```
#gas_engine - number of gasoline cars
no2_gas_emis = gas_engine * vehicles
#length - road length
#gas_usage - fuel usage per 100km
no2_gas_emis *= length * gas_usage
no2_gas_emis /= 100
#gas_density - gasoline density
#gas_no2_index - gasoline NO2 emission index
no2_gas_emis *= gas_density * gas_no2_index
#and make the emission per second
no2_gas_emis /= 3600
```

We setup a stochastic collocation sampler

```
my_sampler = uq.sampling.SCSampler(vary=vary_gas_no2,polynomial_order=1)
```

and use the multiencoder for our campaign.

```
encoders = uq.encoders.MultiEncoder(
    emis_encoder,
    wind_encoder,
    scalars_encoder,
    fort13_encoder)
```

To facilitate running ensembles, each of which requires hundreds of cores, we use an integrator between QCG-PilotJob and EasyVVUQ called EasyVVUQ-QCGPJ.[7,8]

```
qcgpjexec = easypj.Executor()
#use 32 nodes equipped with 24 cores each
qcgpjexec.create_manager(dir=my_campaign.campaign_dir, resources='32:24')
#encoding will use 1 core, one for each of the generated ensemble
qcgpjexec.add_task(Task(
    TaskType.ENCODING,
    TaskRequirements(cores=Resources(exact=1))
))
#each ensemble will run on 16 nodes
qcgpjexec.add_task(Task(
    TaskType.EXECUTION,
    TaskRequirements(nodes=Resources(exact=16),cores=Resources(exact=24)),
    application='mpirun.sh 384'
    ))

#the execution patterns is to generate ensembles, then to run simulation
#there should be 2 ensembles simulation in parallel since one requires
#16 nodes, and we requested 32 nodes for the whole campaign
qcgpjexec.run(
    campaign=my_campaign,
    submit_order=SubmitOrder.PHASE_ORIENTED)
```

9.4. Results and Analysis

In this simulation example, a $2 \times 2 \times 2$ metre grid resolution has been used, and the same resolution has been applied to the output results, which contain NO_2 concentration for each given point in 3D space. The output is then transformed into x^*y columns with z NO_2 values, that is, for each point in 2D-space, there is a list of NO_2 concentration at different heights. Such data is then processed and analyzed using the SCAnalsysis object from EasyVVUQ.

```
my_analysis = uq.analysis.SCAnalysis(sampler=my_sampler,
qoi_cols=["vertical_NO2_concentrations_at_some_point_in_2D_space"])
my_campaign.apply_analysis(my_analysis)
results = my_campaign.get_last_analysis()
stats = results['statistical_moments']
        ['vertical_NO2_concentrations_at_some_point_in_2D_space']
```

The goal of the analysis is to provide us with the mean concentration (and associated uncertainty) for the whole domain at different heights, and to study how the final result may vary due to the incomplete emissions data. While at present the analysis is performed for a given point in 2D space for different heights above street level, future analyses will concern the entire 3D space.

The uncertainty quantification of the UrbanAir workflow for an arbitrary point in 2D-space is shown in **Figure 13**. Since the NO_2 concentrations is attributed to road transportation, it tends to decrease with increasing height above road level. Note that the interpolation of NO_2 concentration in between every 2 m is here due only to the plotting software. The standard deviation indicates how much uncertainty of input parameters (currently only four are taken into account) is reflected in the air quality predictions. In the forthcoming work, a sensitivity analysis will be conducted to select the most important uncertainty input parameters.

10. Discussion and Conclusions

In this work, we have applied EasyVVUQ to five diverse application areas, in order to extract information on sensitivity or

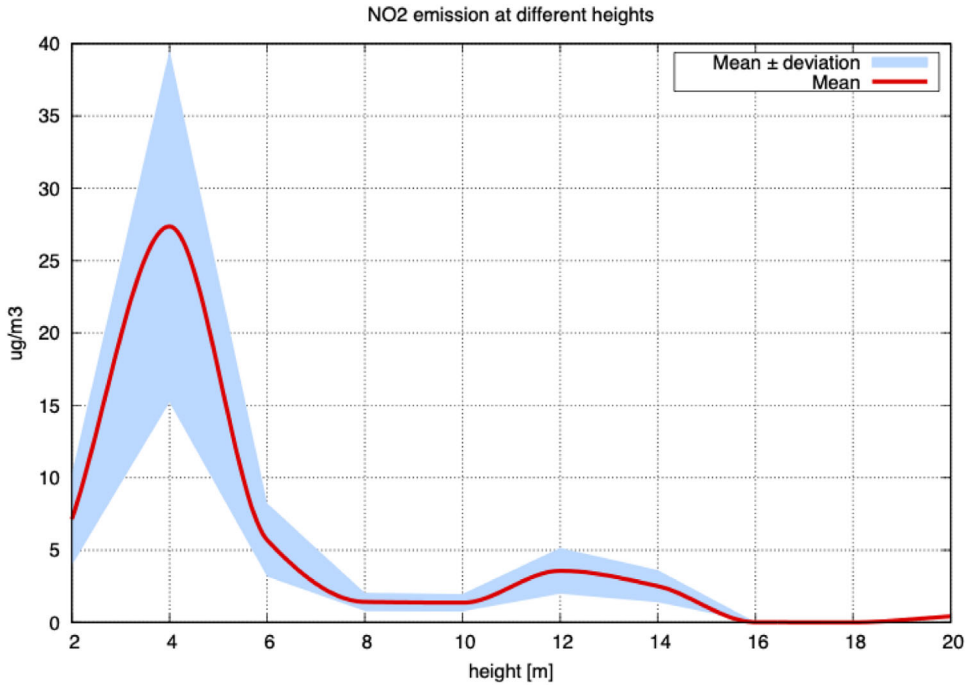


Figure 13. Emissions of NO_2 at different heights above street-level from road transportation, with the mean (red line) and standard deviation (blue region) calculated using the EasyVVUQ campaign.

uncertainty in these pre-existing models, without the need for intrusive modifications to the code. EasyVVUQ provides the tools necessary for computational scientists to add state of the art VVUQ algorithms to their simulation workflows without modifying the underlying codebase.

The library is intentionally execution-method agnostic, providing the base VVUQ workflow elements to allow for different execution patterns (such as Pilot Jobs) facilitated by any choice of middleware solutions. The agnosticism to choice of middleware (including using no middleware at all), and restartability of the workflow, provide the flexibility necessary for EasyVVUQ to be applied to many workflows in the HPC domain. For example, the Fusion application above uses the PSNC Pilot Job Manager to manage job execution, whereas the Ocean Circulation and Migration applications rely on FabSim3. Execution of the materials application, meanwhile, is handled manually by the user. Other middleware solutions may be used, such as RADICAL Cybertools,[9] Dask JobQueue,[46] or cloud submission tools.

The encoding and decoding steps of a standard EasyVVUQ script ensure that application-specific information is abstracted from the rest of the VVUQ workflow. This keeps the UQ algorithms in the sampling elements entirely generic. As such, multiple sampling elements may be chained or combined into more complex sampling elements (such as via use of the `MultiSampler` element). Complex encoding may also be achieved through combining multiple encoders into a single `MultiEncoder` element.

This generic approach is intended to accommodate switching to different UQ methods at no development cost to the user, allowing users to easily try out a variety of UQ approaches. It is intended that many more UQ algorithms will be integrated into this framework over time.

Acknowledgements

D.W.W. and R.A.R. authors contributed equally to this work. The authors are grateful to the VECMA consortium, its Scientific Advisory Board, and the VECMAtk alpha users for their constructive discussions and input around this work. The authors acknowledge funding support from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement 800925 (VECMA project, www.vecma.eu), and the UK Consortium on Mesoscale Engineering Sciences (UKCOMES, http://www.ukcomes.org), EPSRC reference EP/L00030X/1. This work was supported by the Netherlands eScience Center. The calculations were performed in part at the Poznan Supercomputing and Networking Center.

Note: In the originally published version, the URL for the VECMA open source toolkit was repeated in paragraph 3 of Section 1. This was corrected on August 10, 2020, removing the duplicate text.

Conflict of Interest

The authors declare no conflict of interest.

Keywords

high-performance computing, multiscale simulations, uncertainty quantification

Received: December 13, 2019
Revised: April 24, 2020
Published online: June 15, 2020

[1] W. L. Oberkampf, C. J. Roy, *Verification and Validation in Scientific Computing*, Cambridge University Press, Cambridge **2010**.
[2] W. L. Oberkampf, S. M. DeLand, B. M. Rutherford, K. V. Diegert, K. F. Alvin, *Reliab. Eng. Syst. Safe.* **2002**, *75*, 333.

[3] R. A. Richardson, D. W. Wright, W. Edeling, V. Jancauskas, J. Lakhlili, P. V. Coveney, *J. Open Res. Softw.* **2020**, *8*, 11.
[4] D. Groen, R. A. Richardson, D. W. Wright, V. Jancauskas, R. Sinclair, P. Karlshoefer, M. Vassaux, H. Arabnejad, T. Piontek, P. Kopta, B. Bosak, J. Lakhlili, O. Hoenen, D. Suleimenova, W. Edeling, D. Crommelin, A. Nikishova, P. V. Coveney, in *Computational Science – ICCS 2019*, Springer International Publishing, New York **2019**, pp. 479–492.
[5] D. Groen, A. P. Bhati, J. Suter, J. Hetherington, S. J. Zasada, P. V. Coveney, *Comput. Phys. Commun.* **2016**, *207*, 375.
[6] T. Piontek, B. Bosak, M. Ciżnicki, P. Grabowski, P. Kopta, M. Kulczewski, D. Szejnfeld, K. Kurowski, *J. Grid Comput.* **2016**, *14*, 559.
[7] B. Bosak, J. Lakhlili, EasyVVUQ-QCGPJ, https://github.com/vecma-project/easyvvuq-qcgpj (accessed: May 2020).
[8] P. Kopta, B. Bosak, QCG-PilotJob, https://github.com/vecma-project/QCG-PilotJob (accessed: May 2020).
[9] V. Balasubramanian, S. Jha, A. Merzky, M. Turilli, *arXiv preprint arXiv:1904.03085*, **2019**.
[10] Cerberus: Lightweight, extensible data validation library for python, https://github.com/pyeve/cerberus (accessed: April 2020).
[11] Supplementary information/code repository for this paper, https://github.com/vecma-project/EasyVVUQApplicationsSupplementary (accessed: December 2019).
[12] M. Eldred, J. Burkardt, in *47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, American Institute of Aeronautics and Astronautics, Reston, VA **2009**, p. 976.
[13] D. Xiu, *Numerical Methods for Stochastic Computations: A Spectral Method Approach*, Princeton University Press, Princeton, NJ **2010**.
[14] R. Preuss, U. von Toussaint, in *AIP Conf. Proc.*, Vol. 1756, AIP Publishing, New York **2016**, p. 060001.
[15] I. Sobol, *Math. Comput. Simulat.* **2001**, *55*, 271.
[16] G. Tang, G. Iaccarino, M. Eldred, in *51st AIAA/ASME/ASCE/AHS/ASC Structures*, Structural Dynamics, and Materials Conf., Orlando, FL **2010**, pp. 1–13.
[17] B. Sudret, *Reliab. Eng. Syst. Safe.* **2008**, *93*, 964.
[18] M. Vassaux, R. C. Sinclair, R. A. Richardson, J. L. Suter, P. V. Coveney, *Adv. Theory Simul.* **2019**, 1900122.
[19] M. Vassaux, R. Richardson, P. Coveney, *Philos. Trans. R. Soc., A* **2019**, *377*, 20180150.
[20] R. C. Sinclair, Epoxy polymerisation code, https://github.com/velocirobbie/epoxy_polymerisation (accessed: August 2019).
[21] M. Vassaux, R. C. Sinclair, R. A. Richardson, J. L. Suter, P. V. Coveney, *Adv. Theory Simul.* **2019**, *2*, 1800168.
[22] S. Plimpton, *J. Comput. Phys.* **1995**, *117*, 1.
[23] B. M. Boghosian, P. V. Coveney, H. Wang, *Adv. Theory Simul.* **2019**, 1900125.
[24] W. Verkley, P. Kalverla, C. Severijns, *Q. J. R. Meteorol. Soc.* **2016**, *142*, 2273.
[25] R. Peyret, *Spectral Methods for Incompressible Viscous Flow*, Springer Science & Business Media, New York **2013**.
[26] J. Feinberg, H. P. Langtangen, *J. Comput. Sci.* **2015**, *11*, 46.
[27] W. Edeling, D. Groen, FabUQCampaign, https://github.com/wedeling/FabUQCampaign (accessed: May 2020).
[28] G. L. Falchetto, D. Coster, R. Coelho, B. D. Scott, L. Figini, D. Kalupin, E. Nardon, S. Nowak, L. L. Alves, J. F. Artaud, V. Basiuk1, J. P. S. Bizarro, C. Boulbe, A. Dinklage, D. Farina, B. Faugeras, J. Ferreira, A. Figueiredo, Ph. Huynh, F. Imbeaux, I. Ivanova-Stanik, T. Jonsson, H.-J. Klingshirn, C. Konz, A. Kus, N. B. Marushchenko, G. Pereverzev, M. Owsiak, E. Poli, Y. Peysson, et al., *Nucl. Fusion* **2014**, *54*, 043018.
[29] F. Imbeaux, J. Lister, G. Huysmans, W. Zwingmann, M. Airaj, L. Appel, V. Basiuk, D. Coster, L. G. Eriksson, B. Guillerminet, D. Kalupin, C. Konz, G. Manduchi, M. Ottaviani, G. Pereverzev, Y. Peysson, O. Sauter, J. Signoret, P. Strand, *Comput. Phys. Commun.* **2010**, *181*, 987.
[30] O. O. Luk, O. Hoenen, A. Bottino, B. D. Scott, D. P. Coster, *Comput. Phys. Commun.* **2019**, *239*, 126.
[31] J. Lakhlili, O. Hoenen, O. Luk, D. Coster, Multiscale Fusion Workflow, https://github.com/vecma-ipp/MFW (accessed: May 2020).
[32] A. Nikishova, A. G. Hoekstra, *J. Comput. Sci.* **2019**, *35*, 80.
[33] G. M. Venturini, *PhD Thesis*, Universidad Carlos III de Madrid (Spain) **2015**.
[34] J. Lin, *IEEE Trans. Inf. Theory* **1991**, *37*, 145.
[35] UNHCR, Figures at a glance, https://www.unhcr.org/figures-at-a-glance.html (accessed: May 2020).
[36] D. Groen, *Procedia Comput. Sci.* **2016**, *80*, 2251.
[37] D. Suleimenova, D. Bell, D. Groen, *Sci. Rep.* **2017**, *7*, 13377.
[38] F. Chen, H. Kusaka, R. Bornstein, J. Ching, C. Grimmond, S. Grossman-Clarke, T. Loridan, K. Manning, A. Martilli, S. Miao, D. Sailor, F. Salamanca, M. Taha, H. abd Tewari, X. Wang, A. Wyszogrodzki, C. Zhang, *Int. J. Climatol.* **2011**, *31*, 273.
[39] J. M. Prusa, P. K. Smolarkiewicz, A. Wyszogrodzki, *Comput. Fluids* **2008**, *37*, 1193.
[40] P. K. Smolarkiewicz, R. Sharman, in *8th GMU Conf. on Transport and Dispersion Modeling*, George Mason University, Fairfax, VA **2004**.
[41] P. K. Smolarkiewicz, R. Sharman, J. Weil, S. G. Perry, D. Heist, G. Bowker, *J. Comput. Phys.* **2007**, *227*, 633.
[42] A. Wyszogrodzki, S. Miao, F. Chen, *Atmos. Res.* **2012**, *118*, 324.
[43] A. Kumar, R. Jimenez, L. Belalcazar, N. Rojas, *Aerosol Air Qual. Res.* **2015**, *16*, 12.
[44] J. Karlický, P. Huszár, T. Halenka, *Adv. Sci. Res.* **2017**, *227*, 181.
[45] Z. P. Piotrowski, A. Wyszogrodzki, P. K. Smolarkiewicz, *Acta Geophys.* **2011**, *59*, 1294.
[46] M. Rocklin, in Proc. of the 14th Python in Science Conf., Citeseer, **2015**, pp. 130–136.

Publication [P9]

Kulczewski, M., **Bosak, B.**, Kopta, P., Szeliga, W., and Piontek, T.: *Fostering Uncertainty Quantification in Global Challenges with mUQSA Toolkit.* In Wyrzykowski, R., Dongarra, J., Deelman, E., and Karczewski, K. (eds), *Parallel Processing and Applied Mathematics. PPAM 2024.* Lecture Notes in Computer Science, vol. 15581. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-85703-4_3

Ministry points / conference: 20

Contribution of authors:

- Marcin Kulczewski
  - Authorship of the idea underlying the paper
  - Coauthorship of the text of publication (Sections: Introduction, Motivation, Applying Uncertainty Quantification to RES, Summary)
- **Bartosz Bosak**
  - Coauthorship of the text of publication (Section: mUQSA, Summary)
- Piotr Kopta
  - Coauthorship of the text of publication (Section: mUQSA / QCG-Portal, muQSA / QCG-PilotJob)
- Tomasz Piontek
  - Review and editing of the paper
  - Coauthorship of the text of publication (Section: mUQSA / QCG-Portal, muQSA / QCG-PilotJob)
- Wojciech Szeliga
  - Coauthorship of the text of publication (Section: Motivation, Applying Uncertainty Quantification to RES)

Fostering uncertainty quantification in Global Challenges with mUQSA toolkit

Michał Kulczewski[1][0000−0002−1349−2927], Bartosz Bosak[1][0000−0002−9331−3166], Piotr Kopta[1][0000−0001−8237−0969], Wojciech Szeliga[1][0000−0002−0768−9141], and Tomasz Piontek[1][0000−0003−0147−3996s]

Poznan Supercomputing and Networking Center
Jana Pawla II 10, 61-139, Poznan, Poland
https://psnc.pl

Abstract. This paper delves into the application of Uncertainty Quantification (UQ) and Sensitivity Analysis (SA) to address complex, multiscale Global Challenges. Using a Renewable Energy Sources case study, we demonstrate various approaches to incorporate UQ and SA. UQ helps mitigate uncertainties in models and input data, leading to more reliable results. SA identifies the significant influence of specific input parameters on model outputs, aiding in resource allocation and problem-solving. By reducing the number of required parameters, SA can optimize computational resources and accelerate time-to-solution. Additionally, we showcase how UQ and SA can directly contribute to addressing Global Challenges. The paper concludes by discussing the multiscale Uncertainty Quantification and Sensitivity Analysis platform (mUQSA) and its underlying tools, which streamline the implementation of UQ and SA for Global Challenges.

Keywords: Renewable Energy Sources · Uncertainty Quantification · mUQSA · multiscale · HPC.

1 Introduction

The scientific consensus firmly establishes the burning of fossil fuels as a primary driver of climate change. A widely adopted strategy to address this issue centers on transitioning from fossil fuels to renewable energy sources like wind and solar power. The ongoing geopolitical instability in Eastern Europe has further underscored the urgency of this shift. Recent years have witnessed remarkable advancements in renewable energy technology and industry growth. In 2016, wind power accounted for 10.4% of the EU's electricity demand. In 2022, the share of energy consumed in the EU from renewable energy sources was 23%[1]. Projections indicate that wind power will supply 30% of the EU's total power needs by 2030. However, to mitigate climate change, reduce the emission of air

[1] https://ec.europa.eu/eurostat/databrowser/view/nrg_ind_ren/

pollutants and improve energy security, the revised Renewable Energy Directive[2] increases the binding target from 32% to a 42.5% share of renewables in the EU energy consumption, with the aim of achieving 45%.

Accurately understanding, controlling, and predicting energy production from wind and solar plants necessitates a deep comprehension of underlying physical phenomena. Recent advancements in HPC towards exascale computing enable more sophisticated modeling, offering the industry unprecedented opportunities to study these phenomena in detail. Detailed weather forecasts are invaluable to plant operators, as they allow for estimating future energy production. This knowledge is critical not only for Renewable Energy Source (RES) plant owners but also for Distribution System Operators (DSOs) to maintain grid stability and optimize energy market trading. Precisely predicting energy production is essential for grid stabilization and responding to meteorological extreme events that can significantly impact energy generation. Many wind energy DSOs currently rely on statistical approaches and general weather forecasts for energy production predictions. While statistical analysis of energy production and weather correlations can provide insights, general weather forecasts often suffer from coarse grids that overlook local topography and conditions affecting energy production, resulting in inaccurate predictions.

Renewable Energy Sources (RES) is a multiscale modeling toolkit designed to empower Distribution System Operators (DSOs) with advanced prediction capabilities. RES leverages two powerful models, WRF and EULAG:

- WRF: One of the most widely used mesoscale weather prediction systems globally, WRF provides a strong foundation for capturing regional weather patterns.
- EULAG: This all-scale geophysical flow solver brings versatility to RES, enabling simulations across various scales.

The effectiveness of these models has been demonstrated in numerous applications. Studies have shown WRF's accuracy in predicting wind flow in complex terrain [1] and solar irradiation [2]. Similarly, EULAG's capabilities have been validated for wind flow analysis [3] and solar scenarios [4]. The general flow in RES is as follows: data obtained from WRF mesoscale forecast model is transferred into a separate numerical model (EULAG) suitable for predictions at the local scale. Initial data with resolution of several kilometres is supplemented with topography data and information about buildings' shape and height, which influences the wind flow. For wind energy, the RES-wind allows for the collection of separate wind data for each wind turbine, including the vertical wind component. In the final stage, an estimate is made to predict the amount of energy that will be produced. For solar energy, RES-solar module is used. Last but not least, to assess the probability of extreme weather event impact on electrical infrastructure or renewable energy installations, the RES-damages component is used, which is under development in HiDALGO2 project [3]. By integrating these

[2] https://energy.ec.europa.eu/topics/renewable-energy/renewable-energy-directive-targets-and-rules/renewable-energy-directive_en

[3] https://hidalgo2.eu

various modules, RES offers a comprehensive and high-resolution approach to energy production prediction, enabling DSOs to optimise grid management and market operations.

2 Motivation

Verification, Validation, and Uncertainty Quantification (VVUQ) are undeniably cornerstone elements of contemporary application development. The credibility of simulation results, directly rooted in VVUQ processes, is paramount across diverse scientific and engineering domains. Moreover, several ancillary aspects of VVUQ indirectly yet substantially influence the quality of computational experiments. Sensitivity Analysis (SA), which investigates the impact of input uncertainty on model output, is one such critical component. Model calibration and surrogate generation are additional key elements.

Consequently, there's a global movement to support application developers and users in navigating the intricacies of VVUQ. This effort spans from established initiatives, like those led by the American Society of Mechanical Engineers (ASME [5]), which seek to unify practitioners and develop VVUQ standards, to smaller teams employing VVUQ techniques with open-source toolkits like Dakota [6], EasyVVUQ [7], and Uncertainpy [8]. Furthermore, there's a growing trend towards user-friendly graphical environments for VVUQ, such as quoFEM [9].

In Global Challenges, uncertainty quantification (UQ) is indispensable. UQ's primary objective is to evaluate the reliability of predictions, accounting for variability, randomness, and model misspecifications, thereby enhancing solution credibility. Uncertainty can arise from various sources, including input parameters, physical phenomena definitions, discretisation methods, and coupling between models. This paper focuses on uncertainties stemming from input parameters and model parametrisation. Sensitivity analysis (SA) is a pivotal approach within UQ. SA identifies influential input parameters, enabling the reduction of uncertainties and the number of required ensembles. This optimization leads to decreased computational resource demands and shorter time-to-solution. Furthermore, UQ analysis facilitates the identification of the most probable outcomes across all ensembles, providing users with a clearer understanding of the likely results.

The coupled models employed in RES can be adapted to other environmental applications, such as urban air quality. In fact, uncertainty quantification was previously applied to an urban air quality scenario within the UrbanAir application [10]. This application leverages the EasyVVUQ toolkit [7] and QCG tools [11] for automation in HPC environments. UrbanAir is a tool designed to assess and forecast air quality within urban environments, specifically at the district and street levels. It incorporates a comprehensive range of emission sources:

- point sources, attributed to emission from chimneys;
- line sources, attributed to road transportation;
- area sources, attributed to heat appliances.

In the UrbanAir application, uncertainty quantification and sensitivity analysis studies primarily focused on line sources of emission. Road transportation modeling is inherently fraught with uncertainties, such as the number of vehicles passing through a street, engine types, and their associated NOx emission parameters. This study facilitated ensemble simulations that considered a wide range of parameter values. Several outcomes emerged from this analysis. First, a comprehensive ensemble of possible conditions enabled the generation of an average air quality forecast. Second, the probability of specific air quality levels was calculated, allowing for the presentation of most likely results to users. Third, sensitivity analysis identified the least influential parameters, leading to a reduction in the number of ensembles and computational requirements for daily forecasts. These positive outcomes encourage the application of UQ to Renewable Energy Sources models.

3 Applying Uncertainty quantification to RES

In RES applications, UQ and SA are applied, in addition to the aforementioned reasons, to investigate the extent to which increasing spatial or temporal resolution leads to improved results, and to explore the potential benefits of replacing certain components of the EULAG model with surrogates to accelerate computations. Beyond its core applications, uncertainty quantification (UQ) can also deliver more application-oriented results. An example is the RES-damages module, which estimates the potential impact of extreme weather events on overhead electrical networks. A case study was conducted in a Polish city, focusing on excessive wind speeds. The workflow employed three nested domains, with the outermost domain covering the entire Poland at a resolution of 3.6 km, middle domain covering voivodeship with 800m resolution, the innermost domain discretised at a detailed 100-meter spacing covering the whole city, and the 33 eta vertical levels. A custom overhead electrical network was created for this study, based on the real electrical network coverage. This high level of detail enabled the simulation to generate meteorological results for each individual component of the infrastructure.

In RES-damages, uncertainty quantification (UQ) is applied by the means of mUQSA toolkit to analyze the impact of complex terrain on existing or underconstruction overhead electrical networks. In a simplified analysis, uncertainties for ensemble generation and subsequent analysis include various weather conditions, such as wind speed and direction. In this study, five distinct electrical lines located in the outskirts and city center were analysed. Ensembles were generated to simulate various wind conditions, including direction and speed the means of the mUQSA toolkit The wind sector was divided into eight directions: N, NE, E, SE, S, SW, W, NW, while the wind speeds were assigned uniformly from 1 to 20 m/s. The initial conditions were taken fromt the GFS forecast data, and then only wind speed and direction were updated with the generated values.

Figure 1 illustrates the wind speeds at which electrical lines are impacted under various conditions. The x-axis represents the five electrical lines studied,

while the y-axis indicates wind speed in m/s. The 90th percentile means that in 90% of the ensembles, the wind speed impacting the electrical line is less than or equal to the given line. In other words, for 90% of the generated conditions, the maximum wind speed is indicated by this line. The mean wind speed is relatively low, especially for two sites, indicating their greater invulnerability to wind conditions. However, the remaining three sites are more susceptible to the impact of wind speed and direction. Only one site shows minimal impact, while the others are more vulnerable. Electrical line number 5 is particularly endangered due to the surrounding terrain, which amplifies the wind speed beyond the initial boundary conditions. Sensitivity analysis (SA) investigates how

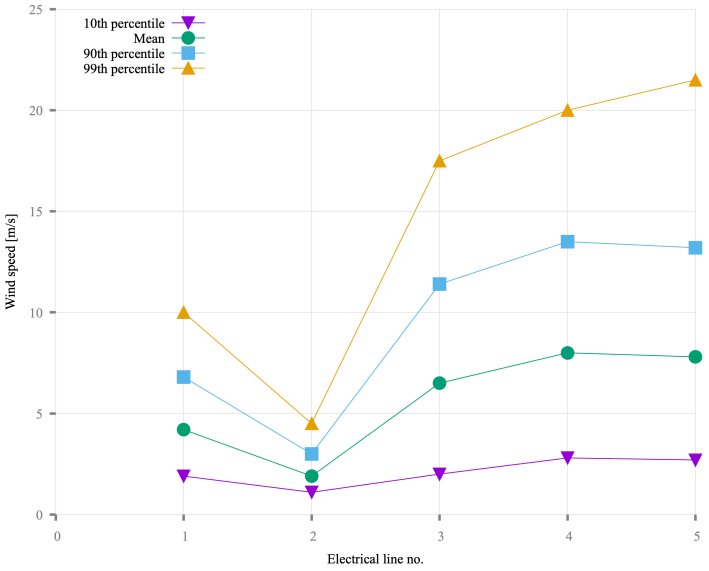


Fig. 1. Wind speed at different sites

analyzed uncertainty parameters influence simulation results. In RES-damages, SA is used to determine whether wind direction or wind speed has a greater impact on wind speed at different sites. Figure 2 illustrates the influence of wind speed and direction. Overall, wind speed emerges as the most significant factor for all five sites. However, the analysis reveals variations in influence. At some sites, such as site number 3, wind direction is nearly as influential as wind speed, while at others, like site number 5, wind direction has minimal impact on the probability of damage. As expected, higher wind speeds increase the vulnerability of electrical lines to damage, regardless of wind direction. With a certain margin of error, further analysis could potentially exclude wind direction from the uncertainties, as it has a moderate to minor impact on the final wind speed at different sites.

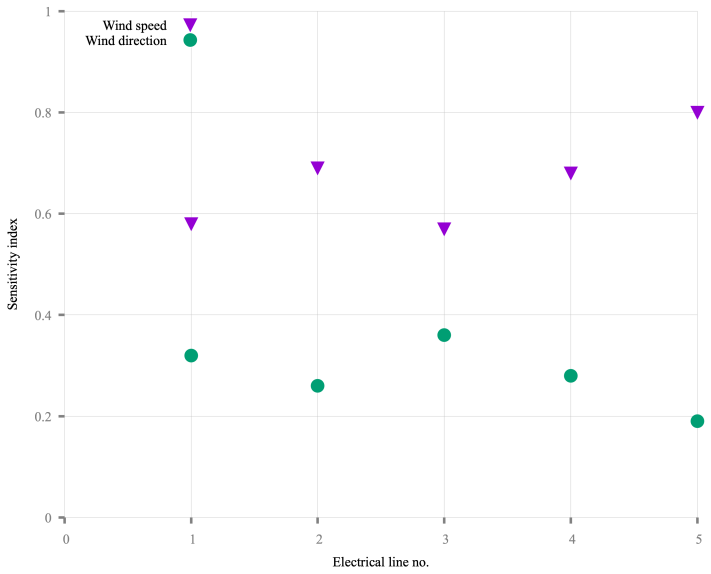


Fig. 2. Wind speed and direction impact at different sites

The studies presented demonstrate how UQ and SA can support renewable energy source owners and electrical network operators in preparing for extreme weather events. By identifying the most vulnerable sites, particularly those susceptible to excessive wind speeds, these tools provide valuable insights. Ongoing and future studies plan to incorporate parameters related to electrical cables and other infrastructure into the UQ process, enabling a more comprehensive assessment of damage probability. The uncertainty quantification and sensitivity analysis are conducted in a fully non-intrusive manner using the mUQSA platform and its underlying tools, which are detailed in the following sections.

4 mUQSA

Uncertainty Quantification (UQ) for scientific applications presents multifaceted challenges. Users often grapple with the complex methodologies underlying UQ, hindering the appropriate application of algorithms to their specific problems. While existing software offers a comprehensive suite of UQ methods, practical implementation can be cumbersome, particularly for inexperienced users. Another challenge associated with versatile non-intrusive UQ methods, such as those used in RES, is ensuring efficient and scalable execution on demanding computational models. These methods inherently rely on multiple model evaluations, which can be resource-intensive when individual simulations require significant computational power. The Multipurpose Uncertainty Quantification and Sensitivity Analysis (mUQSA) platform [12] was developed to address these

two primary challenges. As illustrated in Figure 3, mUQSA is built upon three key components

- QCG-Portal for the user-centric web-based integration with computing clusters,
- QCG-PilotJob for the efficient resource usage,
- and EasyVVUQ for algorithmic part.

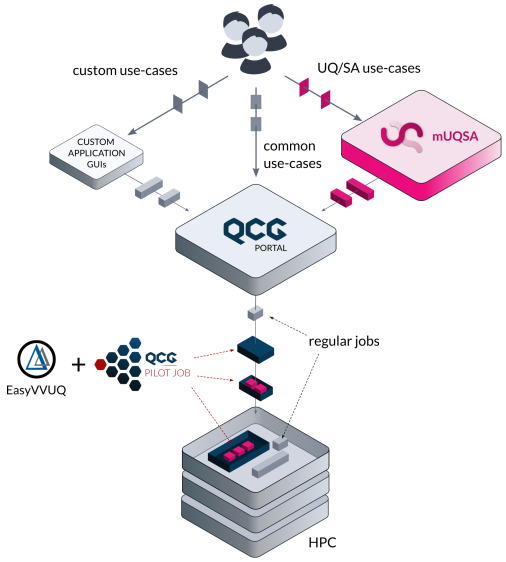


Fig. 3. High-level architecture of mUQSA: mUQSA web-interface is built on top of QCG-Portal, while both EasyVVUQ and QCG-PilotJob are used to ensure efficient execution of UQ/SA algorithms on HPC infrastructure

Consequently, mUQSA offers a comprehensive environment supporting Uncertainty Quantification and Sensitivity Analysis. Its intuitive web interface facilitates both scenario preparation (Figure 4) and results analysis (Figures 1 and 2), providing access to a validated suite of UQ/SA methods, including Monte Carlo, Polynomial Chaos Expansion, and Stochastic Collocation. Furthermore, it automates the execution of these methods on HPC infrastructures. The mUQSA platform's high-level abstractions facilitate a relatively straightforward application of advanced UQ/SA algorithms to diverse scenarios. These abstractions have demonstrated their effectiveness in UQ and SA for RES applications and are believed to be adaptable to a broad range of scientific and industrial domains.

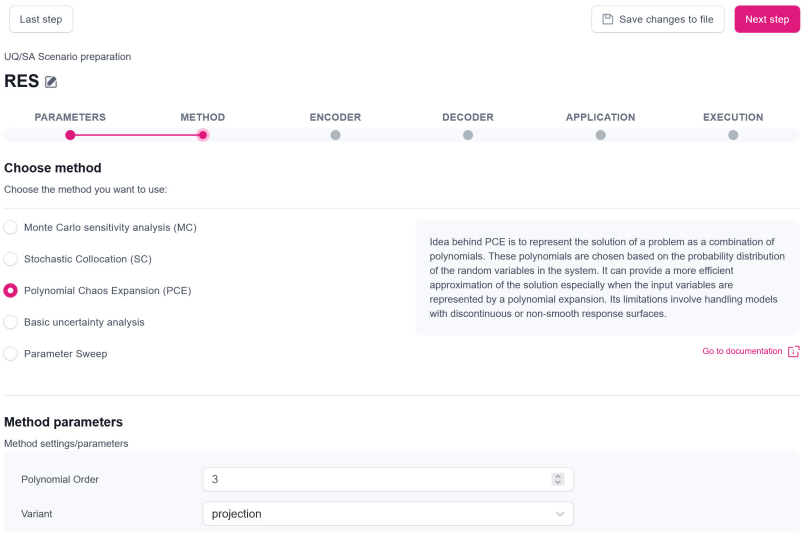


Fig. 4. mUQSA wizard for the definition of UQ/SA scenario launched on top of QCG-Portal

4.1 QCG-Portal

Even for applications with moderate computational demands, comprehensive UQ analysis can quickly necessitate large-scale computing resources. The traditional command-line interface approach to raw HPC environments can be time-consuming and cumbersome for researchers prioritizing domain-specific work over technical aspects. Recognizing this, mUQSA was developed with a strong focus on user experience (UX) leveraging modern web technologies. Fortunately, mUQSA could build upon existing mechanisms provided by QCG-Portal, a comprehensive software stack designed to simplify access to computing infrastructure through a web browser. QCG-Portal supports the development of custom services using a SaaS paradigm (see Figure 5). The mUQSA platform leverages QCG-Portal to offer the following key features:

- Slurm Cluster Integration: mUQSA tasks can be seamlessly submitted from the web browser to the Slurm queue.
- Task Monitoring and Management: Submitted tasks can be monitored, canceled, or resubmitted using the same interface.
- Data Management: QCG-Portal handles all data management aspects, including automatic input and output data transfer between simulations and storage.
- Security: QCG-Portal supports OAuth2 and OIDC protocols, ensuring high security standards through a Single-Sign-On mechanism.

– Customization: Task and custom application templates enable easy integration of dedicated interfaces for preparing input and analysing results of mUQSA scenarios.

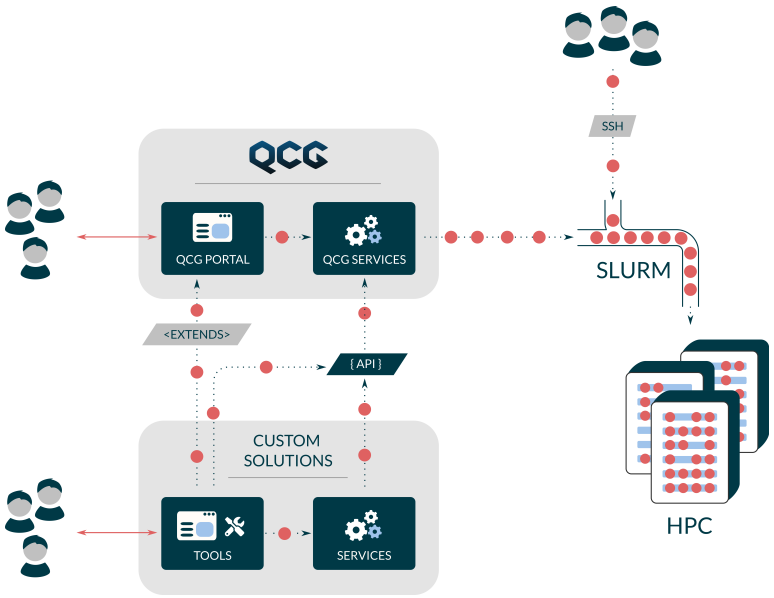


Fig. 5. High-level concept of QCG-Portal: custom solutions extend core functionality provided by QCG-Portal and other QCG services to deliver new use-case-specific functionalities

4.2 EasyVVUQ

Modern Uncertainty Quantification (UQ) algorithms, when combined with adequate computational resources, offer significant opportunities for model developers and users by providing tangible insights from systematically measured uncertainty. While various UQ methods, such as Monte Carlo, Polynomial Chaos Expansion, and Stochastic Collocation, yield meaningful and accurate results, they exhibit distinct characteristics that may be crucial for specific application scenarios. To address this, mUQSA was designed to provide users with a diverse portfolio of algorithms and an extensible architecture that facilitates the integration of new methods. From a programmatic perspective, the EasyVVUQ library, developed with contributions from mUQSA developers, addresses these aspects. By building mUQSA as a web interface on top of EasyVVUQ, we effectively present the state-of-the-art techniques offered by the EasyVVUQ library in a user-friendly manner.

4.3 QCG-Pilotjob

UQ methods inherently demand multiple model evaluations to generate high-quality results. To address this, during the VECMA project, EasyVVUQ was integrated with the QCG-PilotJob library, providing it as an execution backend tailored for demanding HPC model executions. By implementing the Pilot Job paradigm, QCG-PilotJob efficiently realizes second-level scheduling (in contrast to the first-level scheduling provided by queuing systems). This approach enables the flexible execution and management of numerous logically distinct tasks within a single standard job in a queuing system. QCG-PilotJob effectively addresses limitations on the number of tasks or the frequency of task submissions, which are common in HPC environments. In order to address typical execution scenarios, QCG-PilotJob provides two main mechanisms for job submission presented in Figure 6, namely:

1. Batch submission of predefined workflow.
2. Dynamic submission with API (python or network).

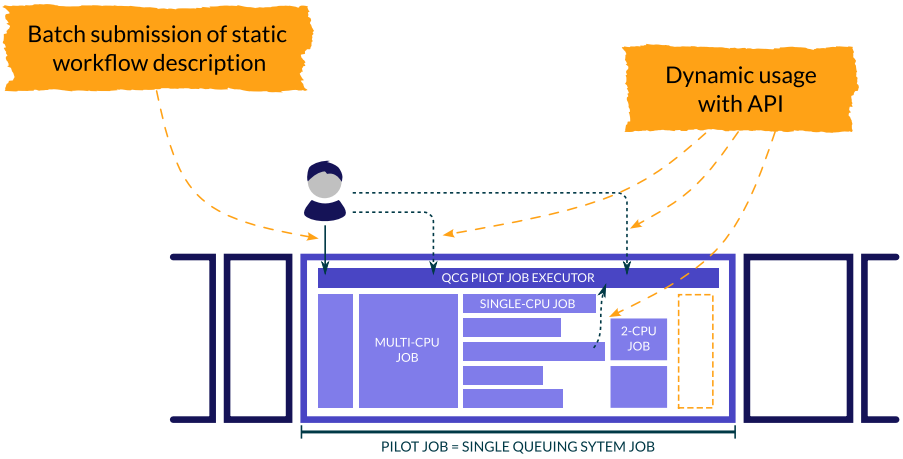


Fig. 6. Two main types of QCG-PilotJob's tasks execution schemes: batch submission of static workflow and dynamic submission with the provided API. The API interface is used by EasyVVUQ/mUQSA

Given EasyVVUQ's need for high flexibility in submitting and managing evaluation tasks, the second option was implemented using the `QCGPJPool` class, integrated within the EasyVVUQ package. `QCGPJPool` is the default executor for mUQSA tasks. Importantly, typical mUQSA users do not need to concern themselves with the executor's configuration, as mUQSA administrators handle all necessary setup and maintenance.

5 Summary

This paper explores the applicability of uncertainty quantification (UQ) and sensitivity analysis (SA) to Global Challenges. UQ can be employed to directly support the resolution of Global Challenges, as exemplified by the Renewable Energy Sources case. This paper also introduces the mUQSA platform, a user-friendly web interface for performing non-intrusive UQ and SA. Future plans include incorporating additional parameters related to electrical cables and infrastructure into the UQ analysis to assess the probability of damages to renewable energy sources or traditional infrastructure.

Another future goal is to develop a surrogate model to reduce the computational demands of the application. A surrogate model can replace a complex and time-consuming model with an approximate solution that maintains satisfactory results. This approach is particularly valuable in emergency scenarios where a slight reduction in quality is acceptable for faster results.

Acknowledgments. Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking and Poland, Germany, Spain, Hungary, France and Greece under grant agreement number: 101093457. This publication expresses the opinions of the authors and not necessarily those of the EuroHPC JU and Associated Countries which are not responsible for any use of the information contained in this publication. The results presented in the article were partially performed on resources provided by the Poznan Supercomputing and Networking Center.

The mUQSA platform's development received partial funding from the PIONIER-LAB project (POIR.04.02.00-30-A005/16-00). We extend our gratitude to the PIONIER-LAB Multiscale Laboratory for their invaluable support in adapting the mUQSA platform to the requirements of the RES use-case.

Additionally, we appreciate the PRACE-LAB project (POIR.04.02.00-00-B001/18) for providing the opportunity to conduct experiments using PSNC's HPC computing resources.

References

1. Solbakken, Kine, Birkelund, Yngve: Evaluation of the Weather Research and Forecasting (WRF) model with respect to wind in complex terrain. Journal of Physics: Conference Series (2018). https://doi.org/1102. 012011. 10.1088/1742-6596/1102/1/012011
2. J. -H. Kim et al: The WRF-Solar Ensemble Prediction System to Provide Solar Irradiance Probabilistic Forecasts. IEEE Journal of Photovoltaics **12**(1), 141–144 (2022) https://doi.org/10.1109/JPHOTOV.2021.3117904
3. M. Z. Ziemiański, M.J. Kurowski, Z.P. Piotrowski, B. Rosa, O. Fuhrer: Toward very high horizontal resolution NWP over the Alps: Influence of increasing model resolution on the flow pattern. Acta Geophysica **59**, 1205–1235 (201)
4. García, Roberto, Camaño, Juan, Barras, Rosa: Toward very high horizontal resolution NWP over the Alps: . Implementation of Two Different Shadow Models into EULAG Model: Madrid Case Study. 316–323 (2011) https://doi.org/10.1007/978-3-642-29843-1_36

5. ASME Homepage, https://www.asme.org/codes-standards/publications-information/verification-validation-uncertainty, last access 2024/05/17
6. Adams, B.M., Bohnhoff, W.J., Dalbey, K.R., Ebeida, M.S., Eddy, J.P., Eldred, M.S., Hooper, R.W., Hough, P.D., Hu, K.T., Jakeman, J.D., Khalil, M., Maupin, K.A., Monschke, J.A., Ridgway, E.M., Rushdi, A.A., Seidl, D.T., Stephens, J.A., Swiler, L.P., and Winokur, J.G., "Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.15 User's Manual," Sandia Technical Report SAND2020-12495, November 2021.
7. D. Groen , H. Arabnejad , V. Jancauskas , W. N. Edeling , F. Jansson , R. A. Richardson , J. Lakhlili , L. Veen , B. Bosak , P. Kopta , D. W. Wright , N. Monnier , P. Karlshoefer , D. Suleimenova , R. Sinclair , M. Vassaux , A. Nikishova , M. Bieniek , Onnie O. Luk , M. Kulczewski , E. Raffin , D. Crommelin , O. Hoenen , D. P. Coster , T. Piontek and P. V. Coveney: VECMAtk: a scalable verification, validation and uncertainty quantification toolkit for scientific simulations. Phil. Trans. R. Soc. A (2020) https://doi.org/10.1098/rsta.2020.0221
8. Tennøe S, Halnes G, and Einevoll GT (2018) Uncertainpy: A Python Toolbox for Uncertainty Quantification and Sensitivity Analysis in Computational Neuroscience. Front. Neuroinform. 12:49. https://doi.org/10.3389/fninf.2018.00049
9. Frank McKenna, Sang-ri Yi, Aakash Bangalore Satish, Adam Zsarnoczay, Kuanshi Zhong, Michael Gardner, and Wael Elhaddad. (2023). NHERI-SimCenter/quoFEM: Version 3.5.0 (v3.5.0). https://doi.org/10.5281/zenodo.10443180
10. Wright, D.W., Richardson, R.A., Edeling, W., Lakhlili, J., Sinclair, R.C., Jancauskas, V., Suleimenova, D., Bosak, B., Kulczewski, M., Piontek, T., Kopta, P., Chirca, I., Arabnejad, H., Luk, O.O., Hoenen, O., Węglarz, J., Crommelin, D., Groen, D. and Coveney, P.V: . Building Confidence in Simulation: Applications of EasyVVUQ. Adv. Theory Simul. **3** (2020) https://doi.org/10.1002/adts.201900246
11. B. Bosak, T. Piontek, P. Karlshoefer, E. Raffin, J. Lakhlili, P. Kopta: Verification, Validation and Uncertainty Quantification of Large-Scale Applications with QCG-PilotJob. In: ICCS 2021, Lecture Notes in Computer Science, vol. 12746, pp. 1–13. (2021) https://doi.org/0.1007/978-3-030-77977-1_39
12. mUQSA Homepage, https://muqsa.multiscale.pionier-lab.pionier.net.pl/, last access 2024/10/16

Bibliography

[Abraham et al., 2024] Abraham, M., Alekseenko, A., Basov, V., Bergh, C., Briand, E., Brown, A., Doijade, M., Fiorin, G., Fleischmann, S., Gorelov, S., Gouaillardet, G., Grey, A., Irrgang, M. E., Jalalypour, F., Jordan, J., Kutzner, C., Lemkul, J. A., Lundborg, M., Merz, P., Miletic, V., Morozov, D., Nabet, J., Páll, S., Pasquadibisceglie, A., Pellegrino, M., Santuz, H., Schulz, R., Shugaeva, T., Shvetsov, A., Villa, A., Wingbermuehle, S., Hess, B., and Lindahl, E. (2024). Gromacs 2024.3 manual. *Zenodo*. Full documentation for the GROMACS 2024.3 release version.

[Adams et al., 2024] Adams, B. M., Bohnhoff, W. J., Dalbey, K. R., Ebeida, M. S., Eddy, J. P., Eldred, M. S., Hooper, R. W., Hough, P. D., Hu, K. T., Jakeman, J. D., Khalil, M., Maupin, K. A., Monschke, J. A., Prudencio, E. E., Ridgway, E. M., Robbe, P., Rushdi, A. A., Seidl, D. T., Stephens, J. A., Swiler, L. P., and Winokur, J. G. (2024). Dakota 6.21.0 documentation. Technical report sand2024-15492o, Sandia National Laboratories, Albuquerque, NM.

[Agullo et al., 2011] Agullo, E., Coti, C., Herault, T., Langou, J., Peyronnet, S., Rezmerita, A., Cappello, F., and Dongarra, J. (2011). Qcg-ompi: Mpi applications on grids. *Future Generation Computer Systems*, 27(4):357–369.

[Allcock et al., 2005] Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., and Foster, I. (2005). The globus striped gridftp framework and server. *SC Conference*, 0:54.

[Alowayyed et al., 2017] Alowayyed, S., Groen, D., Coveney, P. V., and Hoekstra, A. G. (2017). Multiscale computing in the exascale era. *Journal of Computational Science*, 22:15–25.

[Alowayyed et al., 2018] Alowayyed, S., Piontek, T., Suter, J., Hoenen, O., Groen, D., Luk, O., Bosak, B., Kopta, P., Kurowski, K., Perks, O., Brabazon, K., Jancauskas, V., Coster, D., Coveney, P., and Hoekstra, A. (2018). Patterns for high performance multiscale computing. *Future Generation Computer Systems*, 91.

[Alowayyed et al., 2019] Alowayyed, S., Piontek, T., Suter, J., Hoenen, O., Groen, D., Luk, O., Bosak, B., Kopta, P., Kurowski, K., Perks, O., Brabazon, K., Jancauskas, V., Coster, D., Coveney, P., and Hoekstra, A. (2019). Patterns for high performance multiscale computing. *Future Generation Computer Systems*, 91:335–346.

[ASME, 2025] ASME (2025). Verification, validation and uncertainty quantification (vvuq). `https://www.asme.org/codes-standards/publications-information/verification-validation-uncertainty`. Accessed: 2025-10-02.

[Baudin et al., 2015] Baudin, M., Dutfoy, A., Iooss, B., and Popelin, A.-L. (2015). *OpenTURNS: An Industrial Software for Uncertainty Quantification in Simulation*, page 1–38. Springer International Publishing.

[Belgacem et al., 2013] Belgacem, M. B., Chopard, B., Borgdorff, J., Mamonski, M., Rycerz, K., and Harezlak, D. (2013). Distributed multiscale computations using the mapper framework. *Procedia Computer Science*, 18:1106–1115.

[Ben Belgacem et al., 2012] Ben Belgacem, M., Chopard, B., and Parmigiani, A. (2012). Coupling method for building a network of irrigation canals on a distributed computing environment. In Sirakoulis, G. C.

and Bandini, S., editors, *Cellular Automata*, pages 309–318, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Bensoussan et al., 1978] Bensoussan, A., Lions, J.-L., and Papanicolaou, G. (1978). *Asymptotic Analysis for Periodic Structures*. North-Holland Pub. Co., Amsterdam.

[Blanchard, Jean-Baptiste et al., 2019] Blanchard, Jean-Baptiste, Damblin, Guillaume, Martinez, Jean-Marc, Arnaud, Gilles, and Gaudier, Fabrice (2019). The uranie platform: an open-source software for optimisation, meta-modelling and uncertainty analysis. *EPJ Nuclear Sci. Technol.*, 5:4.

[Boehm, 1976] Boehm, B. W. (1976). Software engineering. *IEEE Transactions on Computers*, 25(12):1226–1241.

[Boehm, 1979] Boehm, B. W. (1979). Guidelines for verifying and validating software requirements and design specifications. In *Proceedings of the EURO IFIP 79*, pages 711–719.

[Borgdorff, 2014] Borgdorff, J. (2014). *Distributed Multiscale Computing*. PhD thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands. PhD Thesis.

[Borgdorff et al., 2012] Borgdorff, J., Bona-Casas, C., Mamonski, M., Kurowski, K., Piontek, T., Bosak, B., Rycerz, K., Ciepiela, E., Gubala, T., Harezlak, D., Bubak, M., Lorenz, E., and Hoekstra, A. G. (2012). A distributed multiscale computation of a tightly coupled model using the multiscale modeling language. *Procedia Computer Science*, 9:596–605. Proceedings of the International Conference on Computational Science, ICCS 2012.

[Borgdorff et al., 2013a] Borgdorff, J., Falcone, J.-L., Lorenz, E., Bona-Casas, C., Chopard, B., and Hoekstra, A. G. (2013a). Foundations of distributed multiscale computing: Formalization, specification, analysis and execution. *Journal of Parallel and Distributed Computing*, 73(4):465–483. Published online January 2013.

[Borgdorff et al., 2011] Borgdorff, J., Lorenz, E., Hoekstra, A. G., Falcone, J.-L., and Chopard, B. (2011). A principled approach to distributed multiscale computing, from formalization to execution. In *2011 IEEE Seventh International Conference on e-Science Workshops*, pages 97–104.

[Borgdorff et al., 2013b] Borgdorff, J., Mamonski, M., Bosak, B., Groen, D., Belgacem, M. B., Kurowski, K., and Hoekstra, A. G. (2013b). Multiscale computing with the multiscale modeling library and runtime environment. *Procedia Computer Science*, 18:1097–1105. 2013 International Conference on Computational Science.

[Borgdorff et al., 2014] Borgdorff, J., Mamonski, M., Bosak, B., Kurowski, K., Ben Belgacem, M., Chopard, B., Groen, D., Coveney, P., and Hoekstra, A. (2014). Distributed multiscale computing with muscle 2, the multiscale coupling library and environment. *Journal of Computational Science*, 5(5):719–731.

[Bosak et al., 2012] Bosak, B., Komasa, J., Kopta, P., Kurowski, K., Mamonski, M., and Piontek, T. (2012). *New Capabilities in QosCosGrid Middleware for Advanced Job Management, Advance Reservation and Co-allocation of Computing Resources – Quantum Chemistry Application Use Case*, pages 40–55.

[Bosak et al., 2025] Bosak, B., Kopta, P., Kulczewski, M., and Piontek, T. (2025). muqsa – an online service for uncertainty quantification and sensitivity analysis. In Paszyński, M., Barnard, A. S., and Zhang, Y. J., editors, *Computational Science – ICCS 2025 Workshops*, volume 15911 of *Lecture Notes in Computer Science*, pages 57–70. Springer, Cham.

[Bosak et al., 2014] Bosak, B., Kopta, P., Kurowski, K., Piontek, T., and Mamoński, M. (2014). *New QosCosGrid Middleware Capabilities and Its Integration with European e-Infrastructure*, pages 34–53. Springer International Publishing, Cham.

[Bosak et al., 2021] Bosak, B., Piontek, T., Karlshoefer, P., Raffin, E., Lakhlili, J., and Kopta, P. (2021). Verification, validation and uncertainty quantification of large-scale applications with qcg-pilotjob. In Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, a. V., Dongarra, J. J., and Sloot, P. M., editors, *Computational Science – ICCS 2021*, pages 495–501, Cham. Springer International Publishing.

[Ciepiela et al., 2010] Ciepiela, E., Harżlak, D., Kocot, J., Bartyński, T., Kasztelnik, M., Nowakowski, P., Gubała, T., Malawski, M., and Bubak, M. (2010). Exploratory programming in the virtual laboratory. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 621–628.

[Ciżnicki et al., 2012] Ciżnicki, M., Kierzynka, M., Kopta, P., Kurowski, K., and Gepner, P. (2012). Benchmarking data and compute intensive applications on modern cpu and gpu architectures. *Procedia Computer Science*, 9:1900–1909. Proceedings of the International Conference on Computational Science, ICCS 2012.

[Council, 2012] Council, N. R. (2012). *Assessing the Reliability of Complex Models: Mathematical and Statistical Foundations of Verification, Validation, and Uncertainty Quantification*. The National Academies Press, Washington, DC.

[E et al., 2003] E, W., Engquist, B., and Huang, Z. (2003). Heterogeneous multiscale method: A general methodology for multiscale modeling. *Physical Review B*, 67(9):092101.

[Edeling et al., 2021] Edeling, W., Arabnejad, H., Sinclair, R., and et al. (2021). The impact of uncertainty on predictions of the covidsim epidemiological code. *Nat Comput Sci*, 1:128–135.

[Eldred and Burkardt, 2009] Eldred, M. and Burkardt, J. (2009). 47th aiaa aerospace sciences meeting including the new horizons forum and aerospace exposition. page 976, Reston, VA. American Institute of Aeronautics and Astronautics.

[Evans et al., 2008] Evans, D., Lawford, P., Gunn, J., Walker, D., Hose, D., Smallwood, R., Chopard, B., Krafczyk, M., Bernsdorf, J., and Hoekstra, A. (2008). The application of multiscale modelling to the process of development and prevention of stenosis in a stented coronary artery. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1879):3343–3360.

[Falcone et al., 2010] Falcone, J.-L., Chopard, B., and Hoekstra, A. (2010). Mml: towards a multiscale modeling language. *Procedia Computer Science*, 1(1):819–826.

[Feinberg and Langtangen, 2015] Feinberg, J. and Langtangen, H. P. (2015). Chaospy: an open source tool for designing methods of uncertainty quantification. *Journal of Computational Science*, 11:46–57.

[Ferguson et al., 2020] Ferguson, N., Nedjati-Gilani, G., and Laydon, D. (2020). Covidsim: Covid-19 microsimulation model. `https://github.com/mrc-ide/covid-sim`. Accessed: 2025-10-06.

[Goodale et al., 2003] Goodale, T., Allen, G., Lanfermann, G., Masso, J., Radke, T., Seidel, H., and Shalf, J. (2003). The cactus framework and toolkit: Design and applications. In *VECPAVector and Parallel Processing R'2002, 5th International Conference*, page 2565. Springer.

[Greenfeld, 1967] Greenfeld, A. R. (1967). A statistical approach to motor vehicle accident data analysis. Research memorandum rm-5387-rc, RAND Corporation.

[Gregersen et al., 2007] Gregersen, J., Gijsbers, P., and Westen, S. (2007). Openmi: Open modelling interface. *Journal of Hydroinformatics*, 9.

[Groen et al., 2021a] Groen, D., Arabnejad, H., Jancauskas, V., Edeling, W. N., Jansson, F., Richardson, R. A., Lakhlili, J., Veen, L., Bosak, B., Kopta, P., Wright, D. W., Monnier, N., Karlshoefer, P., Suleimenova, D., Sinclair, R., Vassaux, M., Nikishova, A., Bieniek, M., Luk, O. O., Kulczewski, M., Raffin, E., Crommelin, D., Hoenen, O., Coster, D. P., Piontek, T., and Coveney, P. V. (2021a). Vecmatk: a scalable verification, validation and uncertainty quantification toolkit for scientific simulations.

Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 379(2197):20200221.

[Groen et al., 2021b] Groen, D., Arabnejad, H., Jancauskas, V., Edeling, W. N., Jansson, F., Richardson, R. A., Lakhlili, J., Veen, L., Bosak, B., Kopta, P., Wright, D. W., Monnier, N., Karlshoefer, P., Suleimenova, D., Sinclair, R., Vassaux, M., Nikishova, A., Bieniek, M., Luk, O. O., Kulczewski, M., Raffin, E., Crommelin, D., Hoenen, O., Coster, D. P., Piontek, T., and Coveney, P. V. (2021b). Vecmatk: a scalable verification, validation and uncertainty quantification toolkit for scientific simulations. *Phil. Trans. R. Soc. A*, 379:20200221.

[Groen et al., 2014] Groen, D., Zasada, S. J., and Coveney, P. V. (2014). Survey of multiscale and multiphysics applications and communities. *Computing in Science Engineering*, 16(2):34–43.

[Groen, 2019] Groen, D. e. a. (2019). Introducing vecmatk - verification, validation and uncertainty quantification for multiscale and hpc simulations. In Rodrigues, J. e. a., editor, *Computational Science – ICCS 2019*, volume 11539 of *Lecture Notes in Computer Science*. Springer, Cham.

[He et al., 2025] He, W., Jiang, Z., Xiao, T., Xu, Z., and Li, Y. (2025). A survey on uncertainty quantification methods for deep learning.

[Hetherington et al., 2007] Hetherington, J., Bogle, I., Saffrey, P., Margoninski, O., Li, L., Rey, M. V., Yamaji, S., Baigent, S., Ashmore, J., Page, K., Seymour, R., Finkelstein, A., and Warner, A. (2007). Addressing the challenges of multiscale model management in systems biology. *Computers Chemical Engineering*, 31(8):962–979. 7th World Congress of Chemical Engineering.

[HiDALGO2, 2023] HiDALGO2 (2023). Hidalgo2: Hpc and big data technologies for global challenges. `https://www.hidalgo2.eu/`. Accessed: 2025-09-30.

[Hill et al., 2004] Hill, C., DeLuca, C., Balaji, Suarez, M., and Da Silva, A. (2004). The architecture of the earth system modeling framework. *Computing in Science Engineering*, 6(1):18–28.

[Hill, 2008] Hill, D. (2008). Nuclear energy for the future. *Nature materials*, 7:680–2.

[HiPEAC, oing] HiPEAC (ongoing). Hipeac — international conference on high performance and embedded architectures and compilers. Annual conference focusing on computer architecture, compilers, and parallel computing.

[Hoekstra et al., 2014] Hoekstra, A., Chopard, B., and Coveney, P. (2014). Multiscale modelling and simulation: a position paper. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2021):20130377.

[Hoekstra et al., 2007] Hoekstra, A., Lorenz, E., Falcone, J.-L., and Chopard, B. (2007). Towards a complex automata framework for multi-scale modeling: Formalism and the scale separation map. pages 922–930.

[Hoekstra et al., 2019] Hoekstra, A. G., Chopard, B., Coster, D., Portegies Zwart, S., and Coveney, P. V. (2019). Multiscale computing for science and engineering in the era of exascale performance. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 377(2142):20180144.

[ICCS, oing] ICCS (ongoing). International conference on computational science (iccs). Annual conference covering advances in computational science, modelling, and simulation.

[IEEE, 1986] IEEE (1986). Ieee standard for software verification and validation plans. Ieee std 1012-1986, Institute of Electrical and Electronics Engineers.

[ISC, oing] ISC (ongoing). Isc high performance — international supercomputing conference. Annual conference on high performance computing, networking, and storage.

[Jancauskas et al., 2019] Jancauskas, V., Piontek, T., Kopta, P., and Bosak, B. (2019). Predicting queue wait time probabilities for multi-scale computing. *Phil. Trans. R. Soc. A*, 377:20180151.

[Jiao et al., 2024] Jiao, L., Song, X., You, C., Liu, X., Li, L., Chen, P., Tang, X., Feng, Z., Liu, F., Guo, Y., Yang, S., Li, Y., Zhang, X., Ma, W., Wang, S., Bai, J., and Hou, B. (2024). Ai meets physics: a comprehensive survey. *Artificial Intelligence Review*, 57(256).

[Kastrati et al., 2000] Kastrati, A., Hall, D., and Schömig, A. (2000). Long-term outcome after coronary stenting. *Current Controlled Trials in Cardiovascular Medicine*, 1(1):48–54.

[Kimpton et al., 2025] Kimpton, L. M., Paun, L. M., Colebank, M. J., and Volodina, V. (2025). Challenges and opportunities in uncertainty quantification for healthcare and biological systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 383(2292):20240232.

[Knap et al., 2016] Knap, J., Spear, C., Leiter, K., Becker, R., and Powell, D. (2016). A computational framework for scale-bridging in multi-scale simulations: A computational framework for scale-bridging in multi-scale simulations. *International Journal for Numerical Methods in Engineering*, 108.

[Krzhizhanovskaya et al., 2015] Krzhizhanovskaya, V., Groen, D., Bozak, B., and Hoekstra, A. (2015). Multiscale modelling and simulation workshop:12 years of inspiration. *Procedia Computer Science*, 51:1082–1087. International Conference On Computational Science, ICCS 2015.

[Kulczewski et al., 2024] Kulczewski, M., Bosak, B., Kopta, P., Szeliga, W., and Piontek, T. (2024). Fostering uncertainty quantification in global challenges with muqsa toolkit. In Wyrzykowski, R., Dongarra, J. J., Deelman, E., and Karczewski, K., editors, *Parallel Processing and Applied Mathematics (PPAM 2024)*, volume 15581 of *Lecture Notes in Computer Science*, pages 35–46. Springer, Cham.

[Kurowski et al., 2011] Kurowski, K., Kulczewski, M., and Dobski, M. (2011). Parallel and gpu based strategies for selected cfd and climate modeling models. *Environmental Science and Engineering (Subseries: Environmental Science)*, 3.

[Luk et al., 2021] Luk, O. O., Lakhlili, J., Hoenen, O., von Toussaint, U., Scott, B. D., and Coster, D. P. (2021). Towards validated multiscale simulations for fusion. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2197):20200074.

[Ma et al., 2023] Ma, H., Liu, J., Shang, H., Fan, Y., Li, Z., and Yang, J. (2023). Multiscale quantum algorithms for quantum chemistry. *Chemical Science*, 14(12):3190–3205.

[Maassen and Bal, 2007] Maassen, J. and Bal, H. E. (2007). Smartsockets: solving the connectivity problems in grid computing. In *Proceedings of the 16th International Symposium on High Performance Distributed Computing*, HPDC '07, page 1–10, New York, NY, USA. Association for Computing Machinery.

[Mahmood et al., 2022] Mahmood, I., Arabnejad, H., Suleimenova, D., Sassoon, I., Marshan, A., Serrano-Rico, A., Louvieris, P., Anagnostou, A., Taylor, S. J. E., Bell, D., and Groen, D. (2022). Facs: A geospatial agent-based simulator for analysing covid-19 spread and public health measures on local regions. *Journal of Simulation*, 16(4):355–373.

[Mieczkowski et al., 2025] Mieczkowski, G., Szpica, D., and Borawski, A. (2025). Comprehensive analysis of elastic–plastic behavior in hybrid metal matrix composites with varied reinforcement geometry. *Materials*, 18(12):2763. Open Access.

[MMS, 2025] MMS (2025). (credible) multiscale modelling and simulation — thematic track. Track of the International Conference on Computational Science (ICCS).

[Moustapha et al., 2001] Moustapha, A., Assali, A. R., Sdringola, S., Vaughn, W. K., Fish, R., Rosales, O., Schroth, G., Krajcer, Z., Smalling, R. W., and Anderson, H. (2001). Percutaneous and surgical interventions for in-stent restenosis: long-term outcomes and effect of diabetes mellitus. *Journal of the American College of Cardiology*, 37(7):1877–1882.

[Multiscale Laboratory, 2023] Multiscale Laboratory (2018–2023). Pionier-lab: Multiscale simulation laboratory / project infrastructure. `https://multiscale.pionier-lab.pionier.net.pl/`. National platform for research infrastructure; multiscale simulation laboratory.

[Nathan Barton et al., 2007] Nathan Barton, L., Richard Becker, L., Robert Chen, L., Richard Hornung, L., Jaroslaw Knap, L., Gary Kumfert, L., James Leek, L., John May, L., Miller, P., Morrone, C., et al. (2007). Co-op.

[Nowakowski et al., 2011] Nowakowski, P., Ciepiela, E., Harężlak, D., Kocot, J., Kasztelnik, M., Bartyński, T., Meizner, J., Dyk, G., and Malawski, M. (2011). The collage authoring environment. *Procedia Computer Science*, 4:608–617. Proceedings of the International Conference on Computational Science, ICCS 2011.

[Oberkampf et al., 2002] Oberkampf, W. L., DeLand, S. M., Rutherford, B. M., Diegert, K. V., and Alvin, K. F. (2002). Error and uncertainty in modeling and simulation. *Reliability Engineering & System Safety*, 75(3):333–357.

[Oberkampf, 2003] Oberkampf, W. L. e. a. (2003). Verification, validation, and predictive capability in computational engineering and physics. Technical report.

[PCSS, 2023] PCSS (2023). Qcg — poznań supercomputing and networking center. `https://qcg.psnc.pl`. Accessed: 2025-10-06.

[Pelupessy, F. I. et al., 2013] Pelupessy, F. I., van Elteren, A., de Vries, N., McMillan, S. L. W., Drost, N., and Portegies Zwart, S. F. (2013). The astrophysical multipurpose software environment. *AA*, 557:A84.

[Phillips et al., 2020] Phillips, J. C., Hardy, D. J., Maia, J. D. C., Stone, J. E., Ribeiro, J. V., Bernardi, R. C., Buch, R., Fiorin, G., Hénin, J., Jiang, W., McGreevy, R., Melo, M. C. R., Radak, B. K., Skeel, R. D., Singharoy, A., Wang, Y., Roux, B., Aksimentiev, A., Luthey-Schulten, Z., Kalé, L. V., Schulten, K., Chipot, C., and Tajkhorshid, E. (2020). Scalable molecular dynamics on cpu and gpu architectures with namd. *The Journal of Chemical Physics*, 153(4):044130.

[PIONIER-LAB Project, 2022] PIONIER-LAB Project (2022). Pionier-lab: National platform for the integration of research infrastructures with innovation ecosystems. `https://pionier-lab.pionier.net.pl/`. Accessed: 2025-09-30.

[Piontek et al., 2016] Piontek, T., Bosak, B., Ciżnicki, M., and et al. (2016). Development of science gateways using qcg — lessons learned from the deployment on large scale distributed and hpc infrastructures. *J Grid Computing*, 14:559–573.

[Powers et al., 2017] Powers, J. G., Klemp, J. B., Skamarock, W. C., Davis, C. A., Dudhia, J., Gill, D. O., Coen, J. L., Gochis, D. J., Ahmadov, R., Peckham, S. E., Grell, G. A., Michalakes, J., Trahan, S., Benjamin, S. G., Alexander, C. R., Dimego, G. J., Wang, W., Schwartz, C. S., Romine, G. S., Liu, Z., Snyder, C., Chen, F., Barlage, M. J., Yu, W., and Duda, M. G. (2017). The weather research and forecasting model: Overview, system efforts, and future directions. *Bulletin of the American Meteorological Society*, 98(8):1717 – 1737.

[PPAM, oing] PPAM (ongoing). Ppam — international conference on parallel processing and applied mathematics. Biennial conference on parallel computing and applied mathematics.

[Qu et al., 2011] Qu, Z., Garfinkel, A., Weiss, J. N., and Nivala, M. (2011). Multi-scale modeling in biology: How to bridge the gaps. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 3(2):193–206. Discussion of methods to link molecular, cellular and tissue scales.

[Radecki, 2014] Radecki, M. e. a. (2014). Reservations for compute resources in federated e-infrastructure. In Bubak, M., Kitowski, J., and Wiatr, K., editors, *eScience on Distributed Computing Infrastructure*, volume 8500 of *Lecture Notes in Computer Science*, pages –. Springer, Cham.

[Richardson et al., 2020] Richardson, R. A., Wright, D. W., Edeling, W., Jancauskas, V., Lakhlili, J., and Coveney, P. V. (2020). Easyvvuq: A library for verification, validation and uncertainty quantification in high performance computing. *Journal of Open Research Software*.

[Rook, 1986] Rook, P. (1986). Controlling software projects. *IEEE Software Engineering Journal*, 1(1):7–16.

[Roy and Oberkampf, 2011] Roy, C. J. and Oberkampf, W. L. (2011). A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing. *Computer Methods in Applied Mechanics and Engineering*, 200(25–28):2131–2144.

[Saltelli et al., 2008] Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S. (2008). *Global Sensitivity Analysis. The Primer*. Wiley.

[Schlesinger, 1979] Schlesinger, S. (1979). Terminology for model credibility. *Simulation*, 32:103–104.

[SEAVEA Project, 2021] SEAVEA Project (2021). Seavea toolkit: Software environment for actionable & vvuq-evaluated exascale applications. `http://www.seaveatk.org/`. Accessed: 2025-09-30.

[Sfiligoi, 2008] Sfiligoi, I. (2008). glideinwms—a generic pilot-based workload management system. *Journal of Physics: Conference Series*, 119:062044.

[Sloot and Hoekstra, 2009] Sloot, P. M. and Hoekstra, A. G. (2009). Multi-scale modelling in computational biomedicine. *Briefings in Bioinformatics*, 11(1):142–152.

[Smith, 2013] Smith, R. C. (2013). *Uncertainty Quantification: Theory, Implementation, and Applications*. Society for Industrial and Applied Mathematics.

[Sobol, 2001] Sobol, I. (2001). Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and Computers in Simulation*, 55(1):271–280. The Second IMACS Seminar on Monte Carlo Methods.

[Tadmor et al., 1996] Tadmor, E. B., Ortiz, M., and Phillips, R. (1996). Quasicontinuum analysis of defects in solids. *Philosophical Magazine A*, 73(6):1529–1563.

[Thompson et al., 2022] Thompson, A. P., Aktulga, H. M., Berger, R., Bolintineanu, D. S., Brown, W. M., Crozier, P. S., in 't Veld, P. J., Kohlmeyer, A., Moore, S. G., Nguyen, T. D., Shan, R., Stevens, M. J., Tranchida, J., Trott, C., and Plimpton, S. J. (2022). Lammps — a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*, 271:108171.

[Veen and Hoekstra, 2020] Veen, L. E. and Hoekstra, A. G. (2020). Easing multiscale model design and coupling with muscle3. In Krzhizhanovskaya, V. e. a., editor, *Computational Science – ICCS 2020*, volume 12142 of *Lecture Notes in Computer Science*, pages –. Springer, Cham.

[Wright et al., 2020] Wright, D. W., Richardson, R. A., Edeling, W., Lakhlili, J., Sinclair, R. C., Jancauskas, V., Suleimenova, D., Bosak, B., Kulczewski, M., Piontek, T., Kopta, P., Chirca, I., Arabnejad, H., Luk, O. O., Hoenen, O., Węglarz, J., Crommelin, D., Groen, D., and Coveney, P. V. (2020). Building confidence in simulation: Applications of easyvvuq. *Advanced Theory and Simulations*, 3(8):1900246.

[Ye et al., 2021] Ye, D., Veen, L., Nikishova, A., Lakhlili, J., Edeling, W., Luk, O. O., Krzhizhanovskaya, V. V., and Hoekstra, A. G. (2021). Uncertainty quantification patterns for multiscale models. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2197):20200072.

[Ye et al., 2022] Ye, D., Zun, P., Krzhizhanovskaya, V., and Hoekstra, A. G. (2022). Uncertainty quantification of a three-dimensional in-stent restenosis model with surrogate modelling. *Journal of The Royal Society Interface*, 19(187):20210864.

[Zwart et al., 2013] Zwart, S. P., McMillan, S., van Elteren, A., Pelupessy, I., and de Vries, N. (2013). Multi-physics simulations using a hierarchical interchangeable software interface. *Computer Physics Communications*, 184(3):456–468.

Poznan University of Technology
Faculty of Computing
Institute of Computing Science

Typeset using LaTeX in Computer Modern.

BibTeX:

```
@mastersthesis{ key,
    author = "Bartosz Bosak, M.Sc.  Eng.",
    title = "{Methods for modelling and assessing confidence of heterogeneous multiscale
simulations in high-performance computing environments}",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2025",
}
```