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

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.; Hoekstra, A.G.

Published in:
Procedia Computer Science

DOI:
10.1016/j.procs.2012.04.064

Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 00 (2012) 1–10
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, Faculty of Science, University of Amsterdam, Amsterdam, the Netherlands*
[b] *Poznań Supercomputing and Networking Center, Poznań, Poland*
[c] *Instutute of Computer Science, AGH Univerisity of Science and Technology, Krakow, Poland*
[d] *CYFRONET, AGH Univerisity of Science and Technology, Kraków, Poland*

Abstract

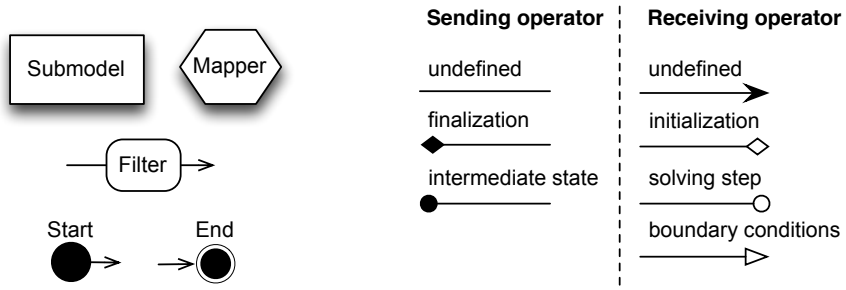
Nature is observed at all scales; with multiscale modeling, scientists bring together several scales for a holistic analysis of a phenomenon. The models on these different scales may require significant but also heterogeneous computational resources, creating the need for distributed multiscale computing. 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)



(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.

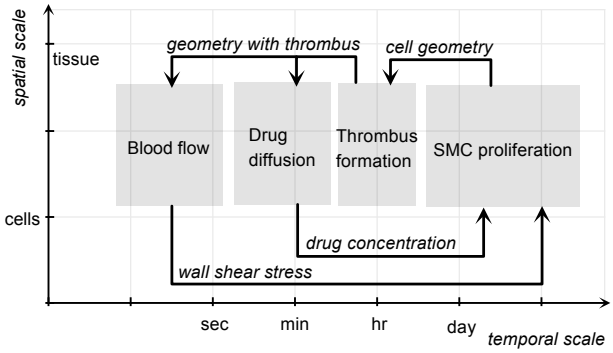


Figure 2: The scale separation map (SSM) of the ISR model, containing four submodels: blood flow, drug diffusion, thrombus formation and smooth muscle cell proliferation. The vertical axis is the logarithmic spatial size, the horizontal axis the logarithmic temporal size.

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`

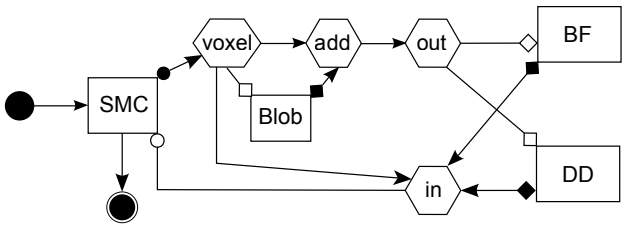


Figure 3: The gMML of the ISR3D model described in Section 3. The meanings of the elements are described in Figure 1. The in and out hexagons are fan-in and fan-out mappers, distributing and collecting data to and from DD and BF.

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).

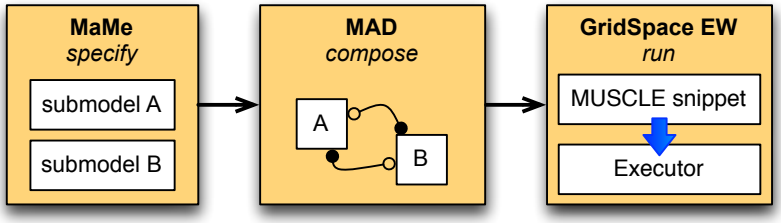


Figure 4: Multiscale programming and execution tools. MAPPER Memory (MaMe) registers information about MML submodels and mappers; Multiscale Application Designer (MAD) supports a user in composing simulation from those submodels and transforms MML into an executable experiment executed in GridSpace Experiment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

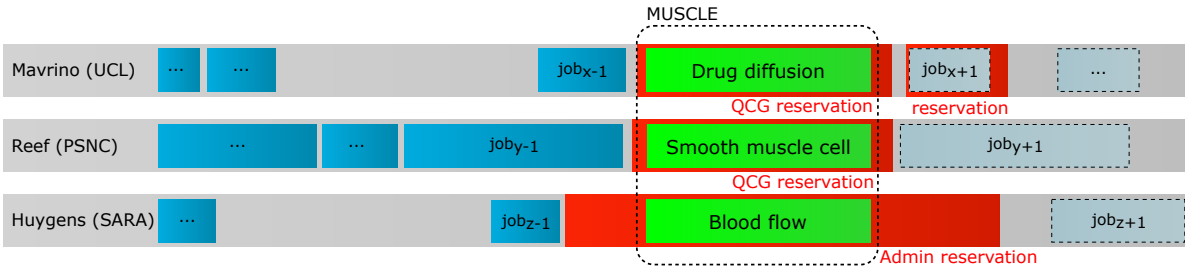


Figure 5: An example of resources co-allocation at three sites

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.

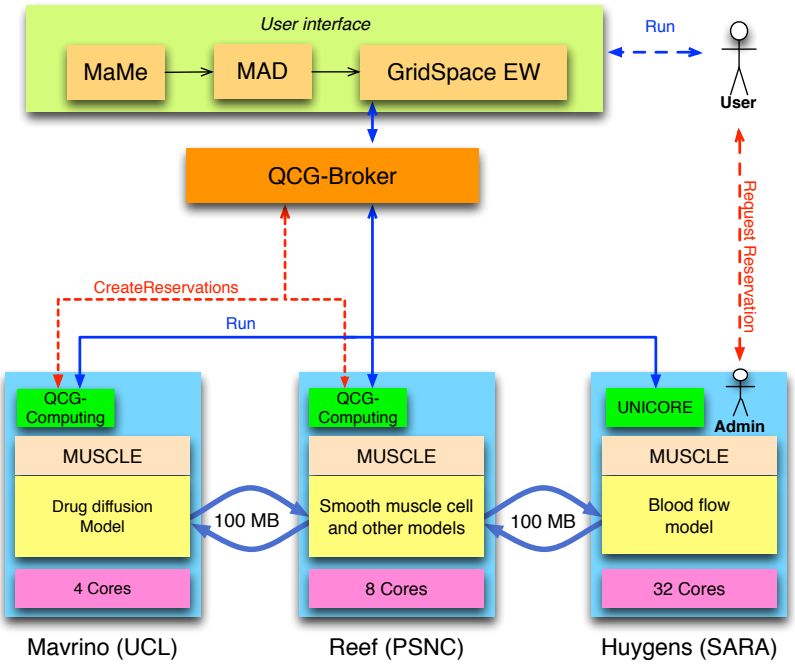


Figure 6: Overall architecture of deployment used during MAPPER demonstration

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.