D7.1 First report on adaptation of applications

Project acronym: *MAPPER*

Project full title: Multiscale Applications on European e-Infrastructures

Grant agreement no.: 261507

Due-Date:	M12
Delivery:	M12
Lead Partner:	UNIGE
Dissemination Level:	PU
Status:	Final
Approved:	Approved (QAB)
Version:	1.4

DOCUMENT INFO

Date and version number	Author	Details
21-07-2011 v0.1	B. Chopard (UNIGE)	document structure
01-09-2011 v0.1	A. Mizeranschi (UU)	contribution
01-09-2011 v0.1	J. Borgdorf (UvA)	contribution
06-09-2011 v0.1	N. Kacem (UNIGE)	contribution
09-09-2011 v0.1	D. Groen (UU)	contribution
12-09-2011 v1.0	B. Chopard (UNIGE)	integration
16-09-2011 v1.0	W. Dubitzki (UU)	internal review
21-09-2011 v1.1	J. Borgdorf (UvA)	corrections
27-09-2011 v1.0	T. Piontek (POZNAN)	internal review
28-09-2011 v1.2	B. Chopard (UNIGE)	corrections
30-09-2011 v1.4	D. Coster (IPP)	fusion additions

TABLE OF CONTENTS

List of Tables

List of Figures

Executive summary

This deliverable provides a report of deliverable D7.1. Its aim is to review the work done during the first 12 months of the MAPPER project, according to the description of WP7.
The goal of WP7 is to adapt or develop a set of selected multiscale applications to the MAPPER framework. This amounts to expressing these applications in a Multiscale Modeling Language (MML), and to adapt the submodels in order to implement their mutual couplings. From that stage, the application can be run on a distributed computing infrastructure, such as the European grid platforms.
Deliverable D7.1 is a first report on the adaptations of the selected applications, whether tightly or loosely coupled. Each of them is described with respect to its level of integration as a “multiscale distributed application”. In this first report, the objective is to build a manual adaptation to better understand the specificities of each application and use this information to build the tools that are developed in WP8.
At the end of this first year, we can conclude that the main objective is attained in a satisfactory way. Most applications are amenable to the MAPPER framework, although only two of them (nano-material and In-Stent Restenosis 3D) are fully MAPPER compliant at this stage. For the other applications, work is progressing well and we do not expect any difficulty to reach the manually programmed version within a few months.

Contributors

This deliverable is the result of contributions from all partners involved in the applications development. UNIGE, as workpackage leader has supervised the integration of all parts of this document. A detailed list of contributor is given on the first page of the document.

1 Introduction

The goal of WP 7 is to consider a portfolio of real-life multiscale applications from various fields of science and technology. These applications must then be adapted to the distributed multiscale framework developed in the MAPPER project. For what concerns WP7, this amounts to express each application in terms of the coupling of several single-scale submodels, identify the nature of the couplings that links each pair of submodels and develop the software components that implements this architecture.
The so-called Multiscale Modeling Language (MML) is the base of our proposed framework. A more detailed explanation of the concepts and notation is given in section D and the references therein. Note that some aspects of the theory are is still under development within the MAPPER project. The original approach evolves as we learn more from the selected applications.
As planned in the description of work, the adaptation of the selected application to the MML framework is first performed manually, by reprogramming some parts of each applications, or by developing new software components, including the proper communication ports, as required by the formalism. Then these elements must be integrated manually within the Grid Space engine or the MUSCLE middleware in order to obtained a distributed multiscale application ready to run on the selected platforms. This requires to write explicitly the scripts that glues the different components together. In a later stage of the MAPPER, the integration will be done automatically from the MML description, through the tools developed in WP 8.
The next section considers the applications in the selected portfolio and describes how the above concepts have been implemented for each of them.

2 Report on the adaptation of applications

This section is organized as follows. For each application, a short description is first given in order to remind the reader of the problem and to make this document more self-consistent. Also, some new decisions on the multiscale framework might be given if they differ from the choices already explained in deliverable 4.1.
After this short introductory description, the implementation of the application within the chosen framework is described. In particular, this second

section explains how far the application is from the targeted milestone: “a manually programmed version”. The discussion includes, whenever possible, the status of the refactored (or new) codes for the submodels, the status of conduits and mappers, their Integration within MUSCLE and/or GridSpace (communication ports, configuration), and the MML or full xMML description.
For several applications, additional material is given in an appendix, at the end of the document.
Note that a summary of the MML approach is given in appendix D. It explains the symbols used in many of the diagrams shown below, in particular the meaning of the shapes that terminate the conduit from one submodel to the other.
Due to a lack of manpower, the two fusion applications, “ Equilibrium Stability Workflow” and “Transport Turbulence Equilibrium” have not yet been adapted to the framework.

2.1 Simulation of Clay-polymer Nanocomposites (Nanomaterials)

2.1.1 Description

Polymer nanocomposites (PNCs) are a new range of particle filled composites with one component possessing a dimension in the nanometre range. Nanocomposites fall within the realm of the emergent area known as nanotechnology, where materials are designed and built at the atomic level, an area of science currently at the forefront of academic, industrial, health and public interest [1]. The microscopic structure and mechanisms of layered nanomaterials operate over many different length scales, ranging from nanometers to microns; each length scale needs to be properly simulated to fully understand their features. This knowledge will eventually lead to the design of novel layered mineral systems with properties tailored to their application. Several types of nanocomposite structure are possible within mineral layers separated by polymeric material including polymer molecules that wrap around large mineral particles (tactoids), polymers that enter between the layers of the clay (intercalate) or mineral layers which exfoliate and become homogeneously dispersed within the polymer matrix.
To understand how we can control which of the three scenarios is most

likely, we need to understand the thermodynamic conditions for the stability of the mineral layer dispersed state and the kinetic mechanisms of dispersion, i.e. from intercalation of the clay layers by polymer through to full exfoliation of the clay layers. To accomplish this, we develop a loosely coupled multiscale simulation model that allows us to study and design layered mineral composites. Our model is initiated at a quantum mechanical scale, followed by a fine-grained molecular dynamics submodel, in turn providing the input for a coarse-grained molecular dynamics submodel (see Fig. 1). Several post-processing scripts are run between each scale of the model to perform data conversion. These scripts may be preprogrammed, but can also (in some cases) be modified by the user between submodels. An overview of the structure of this simulation can be found in Fig. 2.
The purpose of coupling the single scale submodels is to produce a coarse-grained nanomaterials simulation which exhibits statistically accurate behavior on finer levels (e.g., on the resolution of clay-polymer interactions). The fine-grained and quantum mechanical submodels have multiple instances (typically one or two dozen), while we run a single instance of the coarse-grained submodel in the final stage of the computation. Because the simulations do not run concurrently, and the frequency (and required performance) of data exchange between subcodes is limited, we use GridSpace to perform the coupling between the submodels [2]. An overview of the application requirements can be found in Table 1.

2.1.2 Implementation

The loosely coupled nanomaterials application is scheduled to be deployed and working by the fall of 2011, and our efforts have been primarily aimed towards achieving that goal. We have initially investigated the quantum mechanical aspects of clay polymer interactions and have obtained the expertise to integrate the submodel on the quantum mechanical level into our simulations. Furthermore, we have made considerable progress in the deployment, as we have gained access to the PRACE and EGI resources and already deployed and tested the individual submodels for all the resources described in Fig. 2. We are currently working towards establishing the code coupling using GridSpace 2, and testing the integration between GridSpace 2 and AHE for our application.

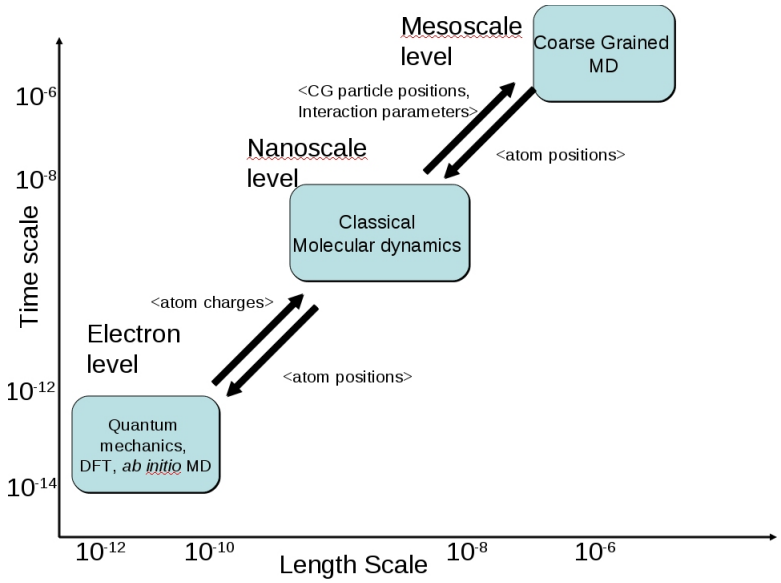

Figure 1: *Scale Separation Map of the nanomaterial application. Labels between '<>' indicate the contents of the interaction*

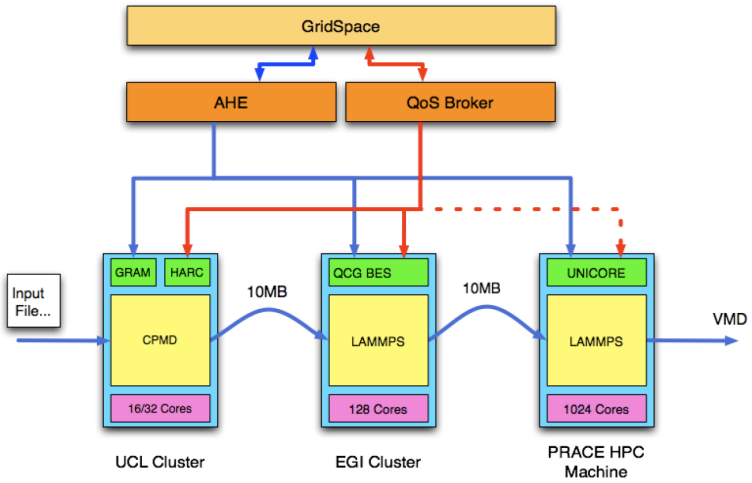

Figure 2: *Deployment overview of a loosely coupled distributed multiscale nanomaterials application within MAPPER. Here the CPMD submodel runs on a small cluster at University College London, while the fine-grained and coarse-grained MD submodels run respectively on a PL-Grid EGI cluster in Poland and the Huygens supercomputer at SARA in Amsterdam.*

The works towards implementing the nanomaterial application have not yet led to revisions in the conceptual properties of our application, which can be found in Fig. 1 and Fig. 3.

Table 1: *Overview of the requirements of each submodel in the nanomaterials application. It lists the current resources used by the submodel and the required resources to use the submodels full potential.*

Coarse-grained molecular dynamics

Requirement	State of the art	Required capabilities
# of cores	256	8192
memory per core [MB]	200	200
runtime	2-6 hours	21 days
coupling frequency	1/run (at the start)	1/hour
max. data size (coupling)	1MB	32MB
max. data size (output)	1GB	32GB

Fine-grained molecular dynamics

Requirement	State of the art	Required capabilities
# of cores	256	8192
memory per core [MB]	200	200
runtime	2-6 hours	2-6 hours
coupling frequency	1/run (at the end)	1/hour
max. data size (coupling)	1MB	32MB
max. data size (output)	1MB	32MB

Quantum mechanical simulation

Requirement	State of the art	Required capabilities
# of cores	16	64
memory per core [MB]	200	200
runtime	1 hour	1 hour
coupling frequency	1/run (at the end)	1/hour
max. data size (coupling)	1MB	32MB
max. data size (output)	1MB	32MB

2.2 In-stent Restenosis 3D (Physiology)

2.2.1 Description

The multiscale three-dimensional In-stent Restenosis model (ISR3D) allows 3D simulation of a stent deployment in a coronary artery and subsequent processes. The objective of the model is to help understand restenosis and to indicate improvements in stent design. An extended multiscale model, in terms of an SSM is described by [4]. A simplified version as well as a detailed description of the implementation of the submodels and the coupling with MUSLE is provided by [3, 5]. Some detailed two-dimensional

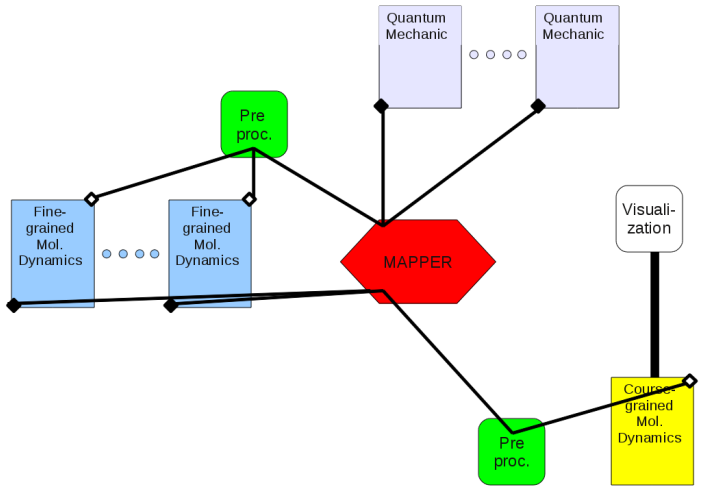

Figure 3: *Multiscale Modelling Language graph of the nanomaterial application.*

simulations are reported by [6].

2.2.2 Implementation

The ISR3D model consists of five submodels: stent deployment as initial conditions (IC), blood flow (BF), drug diffusion of a drug eluting stent (DD), smooth muscle cell proliferation (SMC), and thrombus formation (Blob). Except IC, these are shown as a Scale Separation Map in Figure 4. First, IC initializes the model by placing a stent in an artery. These initial conditions are sent to SMC, which calculates cell dynamics and proliferation. Then for each iteration of SMC, Blob optionally calculates where thrombus should be formed given the blood circulation. This information, along with the cell positions, are given to DD and BF, which are calculated in parallel. Both DD and BF then return their calculated values to SMC. For performance reasons BF keeps track of its last state, simplifying subsequent flow calculations.
The computational requirements of the submodels are listed in Table 2.

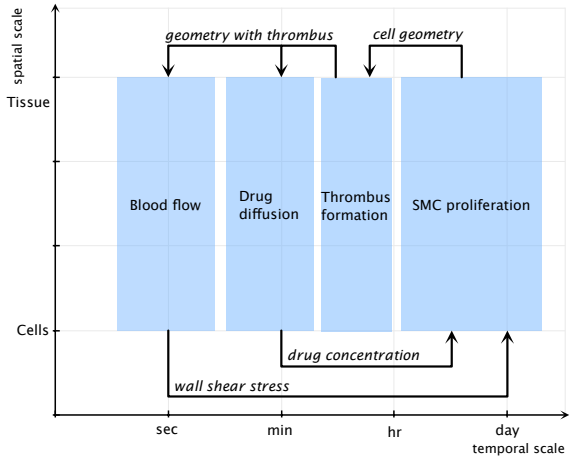

Figure 4: *A SSM of the ISR3D model, excluding initial conditions.*

Table 2: *Runtime statistics of submodels of ISR3D. See text for the definition of the submodels. First is the test environment for the figures listed, then the runtime per iteration, memory, data transferred per iteration and finally the total amount of data transferred.*

Submodel	Test env.	Cores	Runtime/iter.	Memory	DT/iter.	Total DT
IC	Pentium 4	1	20 min.	200 MB	150 MB	150 MB
SMC	Pentium 4	1	20 min.	100 MB	100 MB	300 GB
BF	NEC SX-8	64	15 min.	200 MB	50 MB	150 GB
DD	Pentium 4	8	5 min.	100 MB	50 MB	150 GB
Blob	Pentium 4	1	5 min.	100 MB	50 MB	150 GB
ISR3D	mixed	1 month	700 MB	400 MB	120 GB	

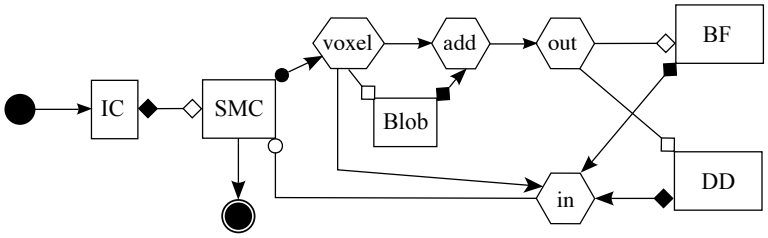

Figure 5: *The gMML of the ISR3D model described in Section 2.2.1. SMC transfers to a fan-out mapper 'voxel' that converts the particle based representation to a regular grid. This grid is given to Blob and the 'add' mapper, and the mapping from cells to grid is given to the 'in' mapper. After Blob has calculated a thrombus, it sends it to the fan-in mapper 'add', which combines it with the grid that 'voxel' sent. The combined grid is sent to the fan-out mapper 'out', which sends it on to DD and BF. The results of DD and BF are collected by the fan-in mapper 'in', which maps their results on a grid back to the particles of SMC and sends this result to SMC.*

The coupling templates used are (see apendix D for definitions)

- $O_f \rightarrow f_{init}$ from IC to SMC,
- $O_i \rightarrow f_{init}$ from SMC to Blob,
- $O_f \rightarrow f_{init}$ from Blob to DD and BF,
- $O_f \rightarrow S$ from DD and BF to SMC.

Each of the submodels is implemented with MUSCLE, as are the fan-in and fan-out mappers to facilitate data transfers from one submodel to the other. The data transfers now happen as visualized in Figure 5.
The model has run at an EGI test site in Poznan. Except SMC, each of the submodels have run on a DEISA resource, the Huygens supercomputer at SARA computing centre in the Netherlands. The SMC submodel could not be run there due to a library dependency.
Although the Blood Flow code runs, we are still trying to exploit the full extent of parallelism of the submodel using MPI. So far, the combination of MUSCLE, Java, Java Native Interface (JNI), and MPI, have not been able to work for us, although there has been progress. To solve this, we are working on using the Palabos Lattice Boltzman solver [1]. This solver already uses MPI and can run as a standalone application, so it does not need JNI.

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

The MML was visualized Figure 5 but the full xMML is listed in Appendix A.1.

2.3 Transport Turbulence Equilibrium (Fusion)

2.3.1 Description

This project is concerned with describing the behaviour of the core plasma of a tokamak. The codes used for this are

1. a 1D transport codes that advances (in time) profiles of density (multiple ion species), temperature (multiple ion species and electron), plasma magnetic flux and rotation profiles (multiple ion species) as a function of a radial coordinate (ρ) subject to externally provided metric coefficients ("geometry"), sources and transport coefficients [ETS, [7]]

2. an equilibrium code which, given information calculated by the transport code, calculates the metric coefficients needed by the transport and turbulence codes [HELENA, [8]]

3. a turbulence code that, given plasma profiles by the transport code and metric information from the equilibrium code, calculates transport coefficients to be used by the transport code [GEM, [9]]

This is a reduction of the schematic workflow shown in Figure 6, to Figure 7. The physics in a tokamak involves a wide scale separation in time, space, and dimensionality (with descriptions ranging from 1D to 6D), as shown in Figure 8. For the reduced description used initially in this project, a scale separation map is shown in Figure 9.
Estimates for the computational requirements are shown in Table 3.

Table 3: *Estimated computational requirements for the transport-equilibrium-turbulence fusion workflow showing the runtime per iteration, memory, data transferred per iteration and finally the total amount of data transferred, all for modern Intel/AMD processors.*

Submodel	Cores	Runtime/iter.	Memory	DT/iter.	Total DT
HELENA	1	5 min	100 MB	10 MB	10 GB
GEM	4*8 – 16*256	25 min	2 GB	1 MB	1 GB
ETS	1	0.01 s	100 MB	1 MB	1 GB

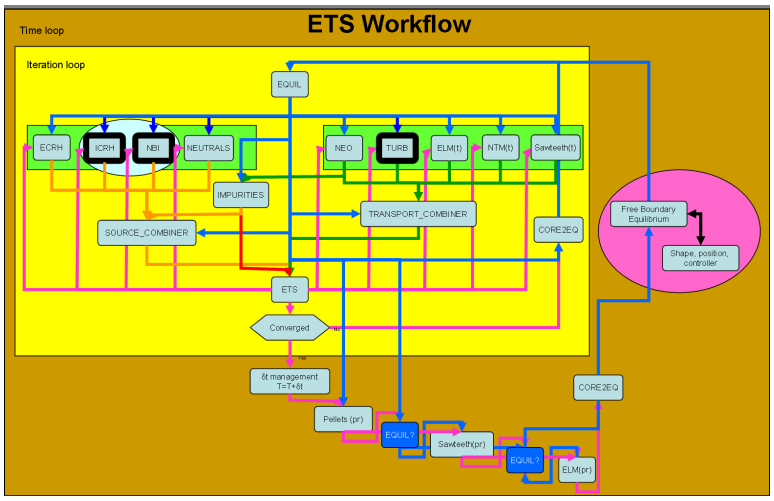

Figure 6: *Schematic workflow describing the evolution of the core plasma in a tokamak.*

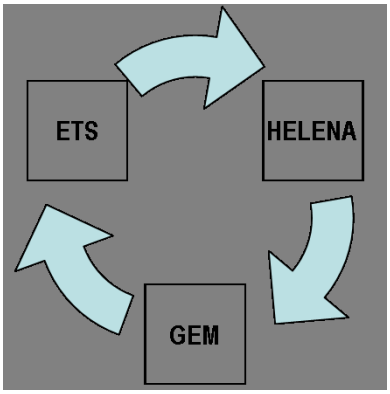

Figure 7: *Reduced workflow showing threee key ingredients: ETS, a 1D core transport code; HELENA, an equilibrium code; and GEM, a turbulence code*

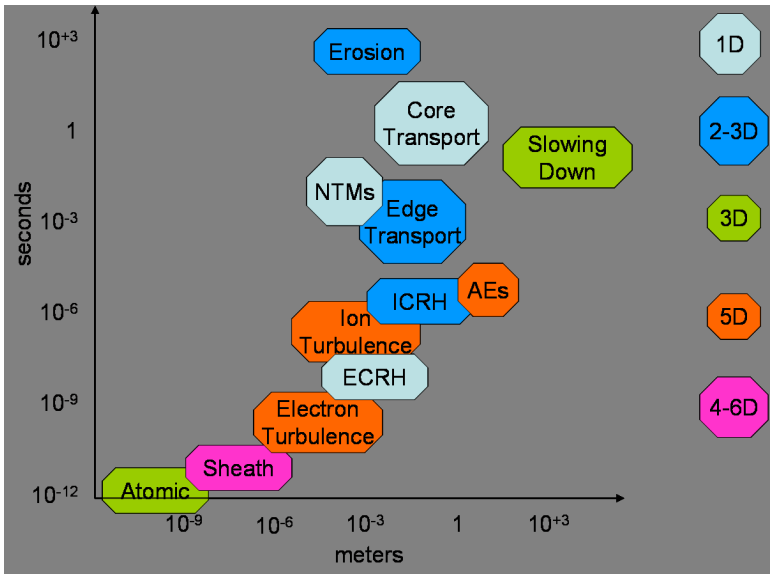

Figure 8: *A large variation in the space- and time-scales of the physics describing the behaviour is found. In addition, the modules differ in the number of treated dimensions, from 1–6,*

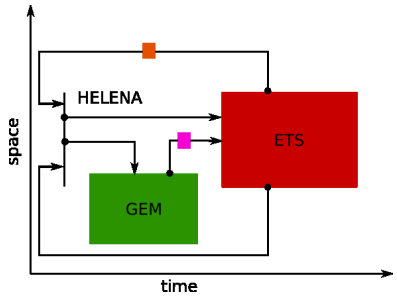

Figure 9: *Scale separation map for the reduced transport-equilibrium-turbulence workflow.*

2.3.2 Implementation

All of the modules are functional. Work is being done on serializing the data that is transferred between the codes from the current format (based on a software environnment developed within the Integrated Tokamak Modelling Task Force of the European Fusion Development Agreement (ITM-TF-EFDA), [10]) to that required within MAPPER.

2.4 HemeLB (Physiology)

2.4.1 Description

We are concerned with performing neurovasculature blood flow simulations in support of clinical neurosurgery. Cerebral blood flow behavior plays a crucial role in the understanding, diagnosis and treatment of cardiovascular disease; problems are often due to anomalous blood flow behavior in the neighborhood of bifurcations and aneurysms within the brain. Simulation offers the possibility of performing patient-specific, virtual experiments to study the effects of courses of treatment with no danger. For this work, we will use our lattice-Boltzmann model, HemeLB [12, 11], designed to simulate fluid flow in the sparse topologies of the human brain. The code can create visualizations from within a running simulation and send them to a viewing client on a workstation situated in, eventually, the hospital. The clinician can then steer the parameters of the simulation.
This work uses the HemeLB code, which is well-suited to describe fluid behavior within sparse systems such as neurovasculatures. The code is written in C/C++ and parallelized using MPI. It employs a number of algorithmic optimizations and techniques to efficiently compute and communicate flow data in sparse and complex geometries. The domain decomposition, using the ParMETIS graph partitioning library, ensures that computational domains are well balanced. Efficient layout of memory and memory access patterns further optimizes the fluid simulation. HemeLB has built in real-time rendering and steering capabilities such that the time-varying behavior of the blood flow can be visualized live by a parallel, in situ rendering algorithm. Each frame is transmitted over the network to a workstation which visualizes the frame in real time. The visual and physical parameters can be interactively modified through steering capabilities. The analysis and

visualization, particularly in real-time, is essential to this successful use of this code in clinical environments.
Away from the region of interest, the accuracy demanded to the hydrodynamic simulation is lower and so can be simulated at a lesser computational expense at a lower resolution, without significantly affecting the accuracy. Further still away, the rest of the circulatory system can be abstracted to a network model of the vascualure and a pump, i.e. the heart. We will couple this hierarchy of submodels together to form a multiscale model.
The submodels are all tightly coupled, each providing boundary conditions to the concurrently-executing, adjacent submodels. The frequent communication between models places demanding requirements on the latency of the coupling library, although in the first instance the volume of data to be exchanged is low. The coupling from the network model to HemeLB will require construction of a flow profile (typically a parabolic Poiseuille flow profile) and the reverse coupling will require computation of the the average pressure and velocity. Coupling of the different resolution HemeLB simulations will require resampling of the underlying lattice-Boltzmann distribution functions between resolutions. The number of submodels and the coupling between them is determined during simulation setup.
Software requirements:

- MPI
- ParMETIS graph partitioning library.

2.4.2 Implementation

HemeLB has been subject to major code improvements over the last year. The performance on a per-process basis has increased by a few percent and the parallel performance improved by an order of magnitude. The improvements have been made with constant regard to good software engineering and numerous parts have been rewritten in a modular fashion.
This has allowed us to begin the programming to allow coupling between models. As a first stage, infrastructure to broadcast boundary condition information to all processes that require it has been implemented and tested for that case of one process reading pressure data from a file. This could be adapted (by reimplementing one method) to accept data from either the network or another MPI process running concurrently.

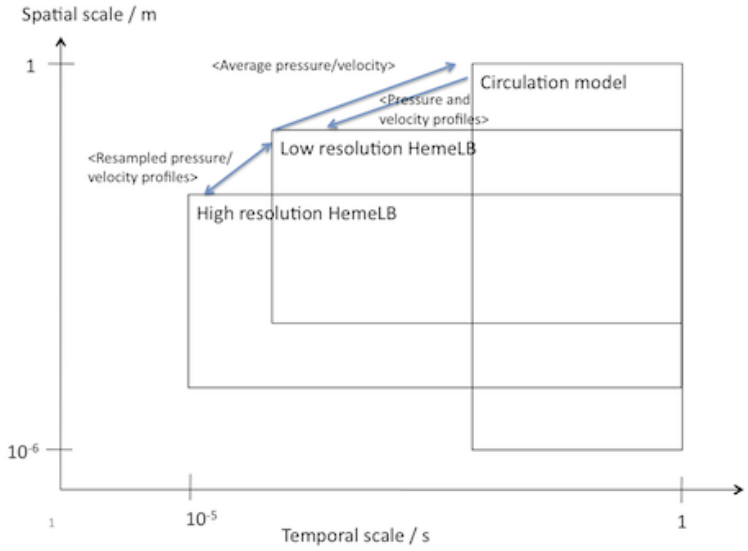

Figure 10: *HemeLB Scale Separation Map*

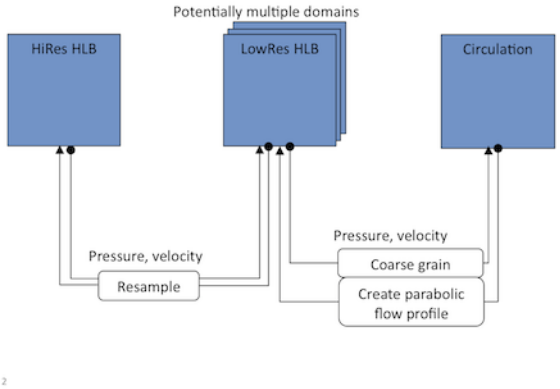

Figure 11: *HemeLB: characterization of the application coupling template*

Table 4: *Computational requirements for the HemeLB scenario. It lists the current resources used by the submodel and the required resources to use the submodels full potential.*

	HemeLB	
Requirement	State of the art	Required capabilities
# of cores	1000	20000
memory per core [MB]	100	100
runtime	30 min	30 min
coupling frequency	1/time step	1/time step
max. data size (coupling)	1 kB	100 kB
max. data size (output)	1GB	10GB

	Network flow model	
Requirement	State of the art	Required capabilities
# of cores	1	100
memory per core [MB]	10	100
runtime	minutes	?
coupling frequency	1/time step	1/time step
max. data size (coupling)	1 kB	100 kB
max. data size (output)	10 MB	100GB

For the case of coupled HemeLB simulations of different resolutions, we plan to implement the coupling based on the full lattice-Boltzmann distribution functions, instead of the hydrodynamic variables (pressure, velocity) as this involves the least loss of information.

2.5 Irrigation Canals (Hydrology)

2.5.1 Description

Canals or rivers are a central infrastructure in all populated areas. They ensure an adequate supply of water for agriculture and are a key component of electricity production or transportation. An optimal management and the control of such resources can be a critical issue for long term planning or to react to natural hazards. In a recent collaboration with ESISAR (Ecole nationale suprieure des systmes avancs et rseaux) at Grenoble INP, France, UNIGE has developed multiscale models for management of a network of irrigation canals. The problem to be solved is to define appropriate actions (e.g. opening and closing gates) that need to be taken to always guarantee an adequate water supply throughout the canal system, whatever the exter-

nal demands or perturbations can be, and respecting constraints such as water height. Our simulation example is a model of the canal de la Bourne irrigation network. This network was built in the late 19th century to irrigate the plains around Valence in France. It is still in use nowadays and its fine modelling and control has become a new challenge. Indeed, the demand of water considerably increased over the last few decades as the constraints on the quantity of water which may be withdrawn from the up-stream natural river La Bourne became more and more limited.
Details about the components of the canal network have been already presented in deliverable 4.1 of the MAPPER project [18].

2.5.2 Implementation

Four main submodels have been identified going from simple one and two-dimensional shallow water flow to three-dimensional free surface and sedimentation process.

Shallow Water 1D (SW1D) The one-dimensional shallow water equation can be used to describe long canal sections. In [23], a model has already been developed based on the D1Q3 lattice Boltzmann (LB) shallow water equation, analyzed in detail and compared with other numerical schemes. The D1Q3 LB model has been implemented with Java and numerical simulations were performed for validation.

Shallow Water 2D (SW2D) It describes branching regions or pools in which the water height varies from the left to the right side. 2D LB-SW models have been considered in several papers [24, 25, 26]. The D2Q9 LB model has been coded in Java and numerical simulations were performed for validation.

Free surface 3D (FS3D) This submodel focuses on the details of the water flow around gates, spillways etc... A meaningful numerical investigation of this problem requires the correct calculation of vertical structure of the flow and the interaction between the liquid and the solid obstacles. A submodel, based on the Free Surface Lattice Boltzmann algorithm (FSF-

LB) [17] has been programmed in the PALABOS[2] environment. It allows for an explicit 3D solution for the interaction between the gate/water system without having to simulate a bi-fluid problem (water-air) [13, 14, 15] or by simplifying the problem using a shallow water approach [16]. 3D massive parallel lattice Boltzmann computations of Free Surface fluid flow have been performed numerically through a gate (see Fig. 12) and results were compared with empirical relationship derived from experiments performed on a laboratory micro-canal facility (LCIS laboratory [22]). Parameters such as the shape of the gate, the initial water level and the velocity of the opening gate system were numerically modified in order to extend the validity range of the known empirical relation. Dam breaking simulations have been also performed using the same model and they showed intresting results as shown in appendix B.1. The Palabos software uses MPI, which may con-

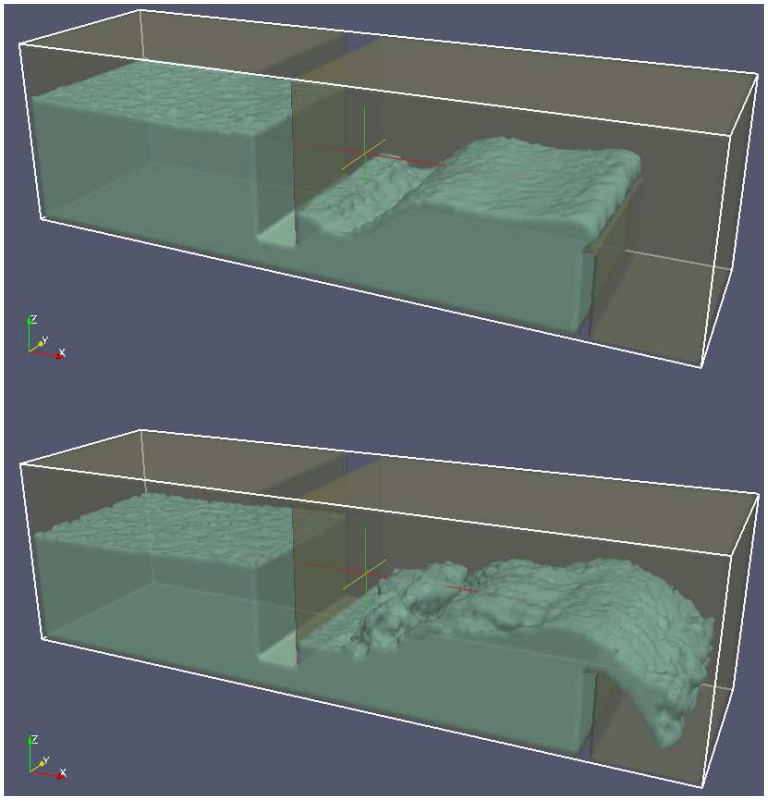

Figure 12: *Lattice Boltzmann simulations of a 3D free surface flow around a gate.*

[2] www.palabos.org

flict with MUSCLE. A Java binding is being developed to make PALABOS compatible with the tools used in MAPPER.

Sediment transport Sedimentation strongly influences the water flow, i.e. reduces the efficiency of the irrigation network. Sediment can be defined as a fragmented material from rocks that has been formed by different physical and/or chemical process and may be transported in three ways: bedload, saltation and suspension. A model of sedimentation has been developed based on a bi-fluid lattice Boltzmann approach. In its 1D version, it uses a simple shallow water 1d model in order to estimate the shear stress between the water and the bed of the canal. The particles on both lattices are updated with respect to the shear stress value. This model has been coded in C++ but not yet validated with respect to experimental investigations. The next steps will consist of using the same approach to generate the 2D sedimentation model and eventually a 3D model based on the developed free surface model.

Coupling Implementation As a proof of concept for the tightly coupled aspect of the irrigation canal application, we considered for instance a connection between two gates of 5 m long and two shallow water 1D of 1 km long. The gate is a simple submodel based on an analytical expression which needs at each time step the state variables in the boundaries of each canal in order to compute the updated values of the missing variables needed by the SW1D models. Hence, the coupling operators used are f_{init} and O_f.

In its complex form, the gate model can be transformed into a 3D free surface model when it is needed. Particularly, it will be the case when the flow around the gate becomes turbulent and as a consequence, small details of the fast water dynamics around the gate influence significantly the slow dynamics of the long reach canal, i.e. they modify the 1D shallow water submodel.

In this case, the grid sizes are 2 m for the SW1D and 1 cm for the FS3D which is translated into a space overlap on the SSM as shown in Fig 13. We assume that the flow takes 10 hours to reach the steady state which is the same time scale for both submodels due to the continuity of the flow. Moreover, the time step sizes are 10 min for the SW1D and 0.1 min for the

FS3D which is translated into a time overlap on the SSM.

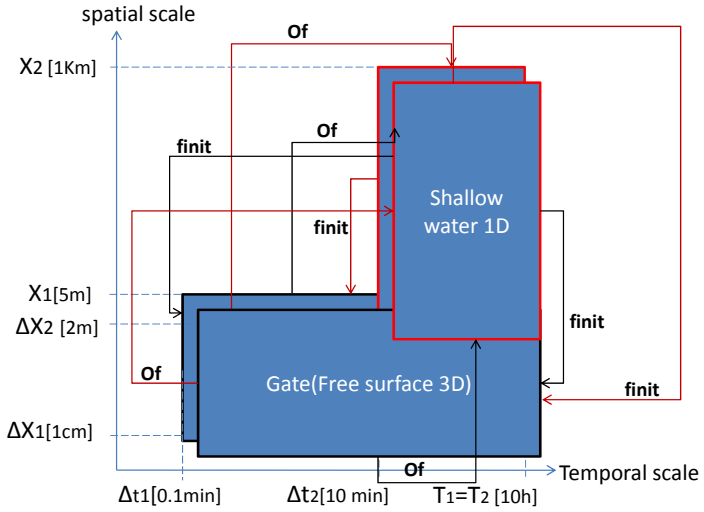

Figure 13: *Scale Separation Map.*

In practice and as a first step, a simple coupling example has been performed between two instances of a 1D canals. the submodels and portals have been defined in MaMe (mapper memory) [19]. Then, the multiscale application designer (MAD) [20] is used in order to explicitly couple the instances of the considered submodels. In the simulated example, each SW1D has two inputs and four outputs and each gate has four inputs and two outputs. Once the coupling between all portals (inputs and outputs) has been done, a CxA file (see appendix B.2) is automatically generated and then used through MUSCLE to run the simulation. The application has been successfully run using GridSpace2 with MUSCLE as an interpreter and simulations were performed on a machine hosted by Poznan Supercomputing and Networking Centre.

In a second step, the coupling example described by the SSM of Fig. 13 has been implemented in the same way successfully and in this case, two instances of SW1D and two instances of a gate are used. As depicted in the Fig. 14, each SW1D has two inputs and four outputs and each gate has four inputs and two outputs. Similary, the coupling is described by a cxa file (see appendix B.3), while the Java based coupling is done using the MUSCLE classes. Therefore, as an example, our SDW1 kernel implementation is written mainly in Java (see appendix B.3).

Finally, a stand-alone version of the second example of coupling has been

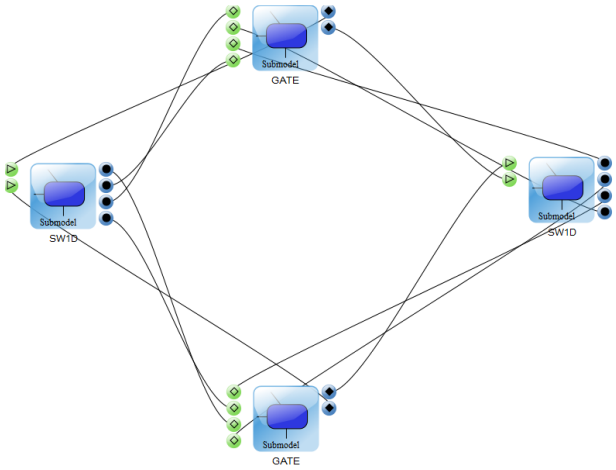

Figure 14: *coupling diagram generated by the MAD tool from WP 8. For the meaning of the symbols, see appendix D.*

implemented in Java directly in order to easily handle the output data of our numerical simulations and for validation. This implementation uses the Java *CxALite* package [21] and allows to run the coupling schema on the same machine for debugging and validation purposes (see appendix B.4) .

2.6 Reverse-engineering of gene-regulatory networks (Computational Biology)

2.6.1 Description

Due to the limited availability of published data for the bile acid and xenobiotic system (BAXS), we have decided to temporarily shift our focus towards investigating reverse-engineering of gene-regulatory networks (GRNs). We chose this, as gene regulation is an important dimension in biology and, most importantly, plenty of data is publicly available.

Regulation of gene expression (or gene regulation) refers to processes that cells use to create functional gene products (RNA, proteins) from the information stored in genes (DNA). These processes range from DNA-RNA transcription to the post-translational modification of proteins. Gene regulation is essential for life as it increases the versatility and adaptability of an organism by allowing it to express protein when needed. After the com-

pletion of such large-scale efforts as the Human Genome Project[3] and recent advances in microscopy and biological imaging techniques, the scientific community is acquiring data of unprecedented detail about the building blocks and internal structures of living organisms. The stage has thus been set for mathematical modelling and computational simulation of complex, large-scale GRNs.

The idea behind reverse-engineering of GRNs is to discover an optimal set of parameters for a computational model of the network that is able to adequately simulate the behaviour described by the gene expression data sets. One first issue we encounter is that there are multiple methods one can choose from to represent the GRN. Three of the most important of these methods have been compared by Swain et al. [33]. These mathematical models require a number of parameters to be fine-tuned in order for the models to accurately simulate real biological network behaviour. This is a combinatorial optimization problem that often requires considerable computational resources depending on the size and complexity of the network being investigated.

Due to the limited number of time points at which gene expression measurements are typically made, the reverse-engineering of GRNs usually constitutes an under-determined problem (having more parameters to estimate than the number of measurements [28]. Although hundreds of gene activities can be measured simultaneously with microarray experiments, the number of (time-dependent) data points established for each gene is typically small, leading to highly under-determined systems. As a consequence, reverse-engineering can yield solutions which are able to fit the available data very well, yet which are very weak in their ability to predict dynamic activity under different conditions to those initially explored [35].

In addition, depending on the mathematical model used, very small variations in a single parameter value can have quite a dramatic effect on the overall network behaviour. There may therefore be a great variation in the quantity of computing power required to optimize parameter values in different networks and in different mathematical models. We distinguish between two sets of case-studies for our application: in the beginning we will focus on simple, monolithic GRNs, consisting of typically up to 10-20 genes which are part of a single biological function. Then, we will address

[3]The Human Genome Project: http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml

more complex, modular GRNs, consisting of tens or hundreds of genes for which a process/functional decomposition was investigated in the biological system. As an example of a simple GRN case scenario we mention the work of Cantone et al. [27], who synthetically engineered a GRN containing 5 genes in yeast in order to facilitate an *in vivo* assessment of various reverse-engineering and gene network modelling approaches, including approaches based on ordinary differential equations (ODEs). We have acquired two datasets, containing 16 and 21 time points respectively. For the optimization itself, we will initially use the particle swarm optimization (PSO) technique. PSO is a population-based stochastic optimization technique, developed by Kennedy & Eberhart [30] and useful in the field of numerical optimization (e.g. minimizing or maximizing a given function). It is inspired from animal social behaviours such as bird flocking or fish schooling and shares many similarities with evolutionary computation techniques such as genetic algorithms (GA). As with GA, in PSO the system is initialized with a population of random solutions, called particles, and searches for optima by updating these particles through generations. Unlike GA, however, PSO does not have any evolution operators such as crossover or mutation, resulting in a fewer number of parameters for the particles. In PSO, the potential solutions “fly” through the solution space by “following” the current optimum particles.

In our implementation, we combine two approaches of evolutionary algorithms: the cellular and the insular approach. See Appendix C.1 for a more thorough description of the PSO technique and details of our implementation. The insular aspect of this technique is what gives a first level of multiscaleness to our application. Each island can be seen as a submodel and thus the coupling will consist of the communication between the islands (the periodic migration of particles). Figure 15 shows the coupling template we use for simulating the simple GRN case-studies. More details can be found in the following section.

In the second set of case studies, the multiscale aspect will be represented by the modular nature of the GRNs we investigate. This will be reflected in the modular aproach we will employ for the reverse-engineering, where each submodel will represent a GRN module. It is our assumption that we will need to use multiscale modelling and simulation methodologies and technologies for such a complex scenario. As an example, we have ac-

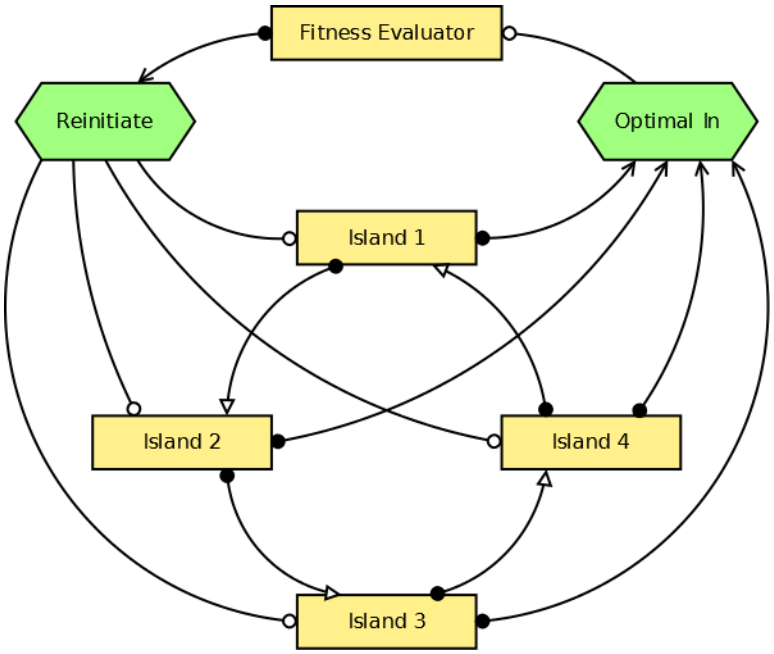

Figure 15: *The coupling template for the simple GRN scenarios*

quired a dataset published by Materna et al.[31], containing 48 time points for 176 genes. The authors investigated gene expression in the early development stages of a marine animal called a sea urchin. Figure 16 shows part of the GRN they describe, where each coloured box represents a module of the GRN.

2.6.2 Implementation

We have finished the implementation of a first prototype of the GRN application, which allows us to run local experiments with a GRN model consisting of 5 genes and 40 parameters (a typical simple GRN scenario, implementing the recurrent neural network (RNN) method for modelling GRNs [34], based on the data provided by Cantone et al. [27]). As previously stated, each island is a separate module implemented as a Java class. The islands run independently of each other, except for occasions in which they will exchange particles. See Appendix C.2 for more details about the run-time behaviour of our prototype.
As dictated also by the MAPPER project's goal, we try to make use of as many pre-existing tools and standards as possible. Systems biology stan-

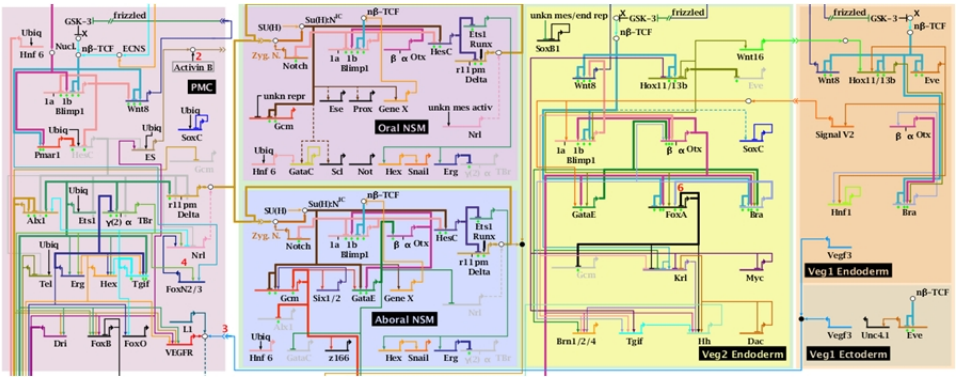

Figure 16: *Part of a GRN snapshot in early sea urchin development*

dards such as SBML and SBRML are used to represent our GRN model (which consists of a system of ODEs) and the experimental and simulated data, respectively. We use the XPPAUT[4] ODE solver to generate data from our GRN model for parameter fitting. We have identified the necessary tools to enable the flow of information between the following standards and formats:

$$SBML - XPPAUT - CSV - SBRML$$

We use the Java version of the libSBML[5] library to read from and write to SBML models and to construct an input file for XPPAUT. For the latter step we also have the option of using a special tool called SBML2XPP[6], however in practice this proved to take more computing time than directly using the libSBML API routines. XPPAUT outputs its results in a CSV (comma separated values) file, which is then translated to SBRML by using a tool called ConsoleSBRML[7].

To address the need for high-performance computing, we will deploy the GRN application on the MAPPER infrastructure. We have set up the computing architecture to use some of the special standards and internal tools which are being developed for MAPPER. Namely, we use MUSCLE to manage the coupling between our islands and we will run the application on the

[4] http://www.math.pitt.edu/~bard/xpp/xpp.html
[5] http://sbml.org/Software/libSBML
[6] http://www.ebi.ac.uk/compneur-srv/sbml/converters/SBMLtoXPP-Aut.html
[7] http://sbrml.sourceforge.net/SBRML/Welcome.html

GridSpace2[8] simulation framework. To achieve this, we will register the components that MUSCLE needs to run into a web interface called MaMe (MAPPER Memory) Registry and we will define the coupling topology of our application in the MAD (Multiscale Application Designer) tool, by using the MML standard.
In conclusion, all the components described in the coupling topology are fully implemented and running, using the MUSCLE library for communication. Following work will be on finalizing the MML description of the application and deploying it on MAPPER using the GridSpace environment.

3 Conclusions

This document is a first report on the adaptation of the chosen applications to the MAPPER infrastructure and to the theoretical framework. Therefore, it is not yet expected that all of them are ready to run in a production mode. However, at this stage of the project, we can observe that a significant progress has been made for most of the selected applications. The proposed multiscale formalism could be applied rather well to all of them despite their different computational structure.
Some applications could follow the proposed approach very closely and are in an advanced stage of development. This is often related to the familiarity of the developers with the proposed formalism and the degree of maturity of the application. From the description of the previous section, it is clear that there is not yet a convergence of the methodology and vocabulary among the different teams. This is easily explained by (1) the diversity of the application groups, (2) the fact the theoretical framework is still adjusting as the project progresses and (3) that some teams are more involve in the MML development than others.
Although there are still several pieces of code that need to be produced before all applications can be run “manually” within the chosen runtime frameworks (MUSCLE, GridSpace), we can conclude that the work done in WP7 during this first year is in accord with our plan and confirms the feasibility and potential of the MAPPER approach for a wide range of applications.

[8] http://dice.cyfronet.pl/gridspace/

A Details on implementation of the ISR-3D application

A.1 The xMML description of In-stent Restenosis 3D (Physiology)

The following is the xMML description of ISR3D.

```
<?xml version="1.0"?>
<model xmlns="http://www.mapper-project.eu/xmml" xmlns:xi="http://www.w3.org/2001/X
  <description>
    A model of the process that occurs in the artery after stenting.
  </description>

  <definitions>
    <datatype id="latticeInt" size_estimate="x*y*z*sizeof(int)"/>
    <datatype id="latticeDouble" size_estimate="x*y*z*sizeof(double)"/>
    <datatype id="agentLocations" size_estimate="4*n*sizeof(double)"/>
    <datatype id="agentDouble" size_estimate="n*sizeof(double)"/>
    <datatype id="agentFull" size_estimate="4*n*sizeof(double)+n*sizeof(double)"/>
    <datatype id="latticeMetadata" size_estimate="2*sizeof(int)+4*sizeof(double)"/>

    <mapper id="distributor" type="fan-out">
      <description>
           Maps a number of agents or particles to a lattice.
      </description>
      <ports>
        <in id="geometry" datatype="latticeInt"/>
        <out id="latticeBF" datatype="latticeInt"/>
        <out id="latticeDD" datatype="latticeInt"/>
      </ports>
    </mapper>
    <mapper id="collector" type="fan-in">
      <ports>
        <in id="cell_mapping" datatype="agentFull"/>
        <in id="latticeBF" datatype="latticeDouble"/>
        <in id="latticeDD" datatype="latticeDouble"/>
```

```
      <out id="cell_bf" datatype="agentDouble"/>
    </ports>
  </mapper>
  <mapper id="voxelizer" type="fan-out">
      <ports>
        <in id="cell_positions" datatype="agentLocations"/>
        <out id="geometry" datatype="latticeInt"/>
        <out id="cell_mapping" datatype="agentFull"/>
      </ports>
  </mapper>
  <mapper id="gridCombine" type="fan-in">
      <ports>
        <in id="geometrySmc" datatype="latticeInt"/>
        <in id="geometryBlob" datatype="latticeInt"/>
        <out id="geometry" datatype="latticeInt"/>
      </ports>
  </mapper>

  <filter id="coarsenGrid" type="reduction" dimension="spatial"/>

  <submodel id="IC" name="Initial cell conditions">
    <timescale delta="1 ms" total="100 ms"/>
    <spacescale delta="10 um" total="1 mm" dimensions="3"/>

    <ports>
      <out id="cells" operator="Of" datatype="agentLocations"/>
    </ports>
  </submodel>

  <submodel id="Blob" name="Thrombus formation">
    <timescale delta="1 ms" total="100 ms"/>
    <spacescale delta="10 um" total="1 mm" dimensions="3"/>

      <ports>
        <int id="geometry" operator="finit" datatype="latticeInt"/>
        <out id="thrombus" operator="Of" datatype="latticeInt"/>
```

```
    </ports>
  </submodel>

  <submodel id="BF" name="Blood flow" stateful="optional">
    <timescale delta="1 ms" total="1 s"/>
    <spacescale delta="10 um" total="1 mm" dimensions="3"/>

    <ports>
      <in id="state_start" operator="finit" type="state"/>
      <in id="boundary" operator="finit" datatype="latticeInt"/>
      <out id="shear_stress" operator="Of" datatype="latticeDouble"/>
      <out id="state_end" operator="Of" type="state"/>
    </ports>
  </submodel>

  <submodel id="SMC" name="Smooth muscle cells">
    <timescale delta="1 hr" total="4 weeks"/>
    <spacescale delta="10 um" total="1 mm" dimensions="3"/>
    <param id="n" value="500000"/>

    <ports>
      <in id="initial_positions" operator="finit" datatype="agentLocations"/>
      <in id="shear_stress" operator="S" datatype="agentDouble"/>
      <in id="drug_concentration" operator="S" datatype="agentDouble"/>
      <out id="cell_positions" operator="Oi" datatype="agentFull"/>
    </ports>
  </submodel>

  <submodel id="DD" name="Drug diffusion">
    <timescale delta="1 min" total="10 min"/>
    <spacescale delta="10 um" total="1 mm" dimensions="3"/>

    <ports>
      <in id="boundary" operator="finit" datatype="latticeInt"/>
      <out id="drug_concentration" operator="Of" datatype="latticeDouble"/>
    </ports>
```

```
    </submodel>
  </definitions>

  <topology>
    <instance id="ic" submodel="INIT" domain="artery"/>
    <instance id="blob" submodel="Blob" domain="artery.blood"/>
    <instance id="bf" submodel="BF" domain="artery.blood"/>
    <instance id="dd" submodel="DD" domain="artery.tissue"/>
    <instance id="smc" submodel="SMC" domain="artery.tissue"/>

    <instance id="voxel" mapper="voxelizer"/>
    <instance id="add" mapper="gridCombine"/>
    <instance id="out" mapper="ditributor"/>
    <instance id="in" mapper="collector"/>

    <coupling from="ic.cells" to="smc.initial_positions"/>
    <coupling from="smc.cell_positions" to="voxel.cell_positions"/>
    <coupling from="voxel.geometry" to="blob.geometry"/>
    <coupling from="voxel.cell_mapping" to="in.cell_mapping"/>
    <coupling from="voxel.geometry" to="add.geometrySmc"/>
    <coupling from="blob.thrombus" to="add.geometryBlob"/>
    <coupling from="add.geometry" to="distributor.geometry"/>
    <coupling from="distributor.latticeBF" to="bf.boundary"/>
    <coupling from="distributor.latticeDD" to="dd.boundary"/>
    <coupling from="dd.drug_concentration" to="in.latticeDD"/>
    <coupling from="bf.shear_stress" to="in.latticeBF"/>
    <coupling from="in.cell_dd" to="smc.drug_concentration"/>
    <coupling from="in.cell_bf" to="smc.shear_stress"/>
  </topology>
</model>
```

B Details on implementation for the hydrology application

B.1 Dam break simulations

This section illustrates the 3D Free Surface flow submodel in case of the so-called dam-break simulation. Three successive snapshots of the evolution are shown.

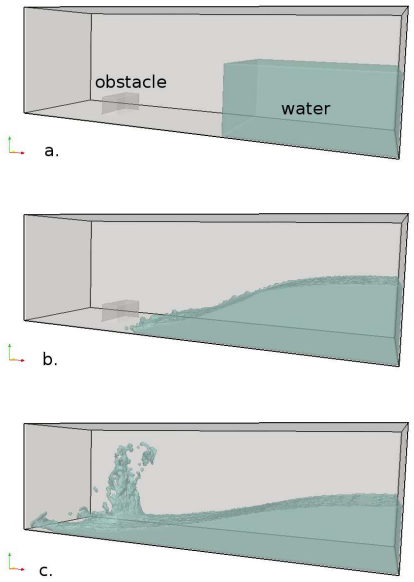

Figure 17: *Lattice Boltzmann simulations of a dam break.*

B.2 cxa file of a simple coupling between two 1D canals

```
# configuration file for a MUSCLE CxA
abort "this is a configuration file for to be used with the MUSCLE bootstrap utility
if __FILE__ == $0

# add build for this cxa to system paths (i.e. CLASSPATH)
m = Muscle.LAST
m.add_classpath "/home/IrigCan.jar"
```

```
# configure cxa properties
cxa = Cxa.LAST

cxa.env["cxa_path"] = File.dirname(__FILE__)

# declare kernels
cxa.add_kernel('sw1', 'irigcan.SW1D_kernel')
cxa.add_kernel('sw2', 'irigcan.SW1D_kernel')

# parameters
# configure connection scheme
cs = cxa.cs
cs.attach('sw1' => 'sw2' ) {
  tie('f1_out', 'f1_in')
  tie('f2_out', 'f2_in')
}
cs.attach('sw2' => 'sw1') {
  tie('f1_out', 'f1_in')
  tie('f2_out', 'f2_in')
}
```

B.3 cxa file of a coupling between two 1D canals and two gates

```
# configuration file for a MUSCLE CxA
abort "this is a configuration file for to be used with the MUSCLE bootstrap utility
if __FILE__ == $0

# add build for this cxa to system paths (i.e. CLASSPATH)
m = Muscle.LAST
m.add_classpath "$HOME/IrigCan.jar:$HOME/IrigCan.jar:$HOME/IrigCan.jar:$HOME/IrigCan

# configure cxa properties
cxa = Cxa.LAST
cxa.env["cxa_path"] = File.dirname(__FILE__)
```

```
# declare kernels
cxa.add_kernel('SW1D_001', 'irigcan.SW1D_kernel')
cxa.add_kernel('SW1D_002', 'irigcan.SW1D_kernel')
cxa.add_kernel('GATE_001', 'irigcan.Gate_kernel')
cxa.add_kernel('GATE_002', 'irigcan.Gate_kernel')

# parameters

# configure connection scheme
cs = cxa.cs
cs.attach('SW1D_001' => 'GATE_002') {
  tie('f0_right', 'f0_right')
  tie('f1_out', 'f1_in')
}
cs.attach('GATE_002' => 'SW1D_001') {
  tie('f2_out', 'f2_in')
}
cs.attach('GATE_002' => 'SW1D_002') {
  tie('f1_out', 'f1_in')
}
cs.attach('SW1D_002' => 'GATE_002') {
  tie('f0_left', 'f0_left')
  tie('f2_out', 'f2_in')
}
cs.attach('SW1D_001' => 'GATE_001') {
  tie('f0_left', 'f0_left')
  tie('f2_out', 'f2_in')
}
cs.attach('GATE_001' => 'SW1D_001') {
  tie('f1_out', 'f1_in')
}
cs.attach('GATE_001' => 'SW1D_002') {
  tie('f2_out', 'f2_in')
}
cs.attach('SW1D_002' => 'GATE_001') {
  tie('f0_right', 'f0_right')
```

```
  tie('f1_out', 'f1_in')
}
```

B.4 The MUSCLE coupling Java code between 1D canal and two gates

```
1
2  public class SW1D_kernel extends muscle.core.kernel.CAController {
3
4      private int sizeX;
5      private int dx;
6      private double a;
7
8      @Override
9      public Scale getScale() {
10         DecimalMeasure dt = DecimalMeasure.valueOf(new BigDecimal(1), SI.SECOND);
11         DecimalMeasure dx = DecimalMeasure.valueOf(new BigDecimal(1), SI.METER);
12         return new Scale(dt, dx);
13     }
14     private ConduitEntrance<Double> f1_out;
15     private ConduitEntrance<Double> f2_out;
16     private ConduitEntrance<Double> f0_left;
17     private ConduitEntrance<Double> f0_right;
18     private ConduitExit<Double> f1_in;
19     private ConduitExit<Double> f2_in;
20
21         @Override
22     public Scale getScale() {
23         DecimalMeasure dt = DecimalMeasure.valueOf(new BigDecimal(1), SI.SECOND);
24         DecimalMeasure dx = DecimalMeasure.valueOf(new BigDecimal(1), SI.METER);
25         return new Scale(dt, dx);
26     }
27     private ConduitEntrance<Double> f1_out;
28     private ConduitEntrance<Double> f2_out;
29     private ConduitEntrance<Double> f0_left;
30     private ConduitEntrance<Double> f0_right;
31     private ConduitExit<Double> f1_in;
32     private ConduitExit<Double> f2_in;
33
34     @Override
35     protected void addPortals() {
36         f1_out = addEntrance("f1_out", 1, double.class);
37         f2_out = addEntrance("f2_out", 1, double.class);
38         f0_left = addEntrance("f0_left", 1, double.class);
39         f0_right = addEntrance("f0_right", 1, double.class);
40         f1_in = addExit("f1_in", 1, double.class);
41         f2_in = addExit("f2_in", 1, double.class);
42     }
```

```
43
44      @Override
45      protected void execute() {
46
47          sizeX = 5;
48          dx = 1;
49          SW1D can1 = new SW1D(sizeX, dx);
50          can1.initLattice();
51          int nbriteration = 10;
52          String kernel_name = getKernelBootInfo().getName();
53          System.out.println(kernel_name + "----------Starting: ");
54          for (int j = 0; j < nbriteration; j++) {
55              can1.observation();
56              can1.test();
57              can1.propagation();
58              //Sending and receiving data
59              f0_left.send(can1.getf0(0));
60              f0_right.send(can1.getf0(sizeX - 1));
61              f1_out.send(can1.getf1(0));
62              f2_out.send(can1.getf2(sizeX - 1));
63              can1.setf1(0, f1_in.receive());
64              can1.setf2(sizeX - 1, f2_in.receive());
65              can1.printHH(kernel_name, j);
66          }
67          System.out.println(kernel_name + "----------End ");
68      }
69  }
```

This code is related to the SWD1 submodel. All the connexions between this SWD1 kernel and the two gates are decalred in lines 14-19 and must instantiate the Java class *ConduitEntrance*. Lines 27-32 declare the type of data to be exchanged, namely that *f0_left*, *f0_right*, *f1_out*, *f2_out* are the outputs of the kernel SWD1 and *f1_in* and *f2_in* are its two inputs. The data type in this example is the *double* Java primitive data type. It's worth reminding here that all the methods of the *"muscle.core.kernel.CAController"* class should be implemented in order to perform a computation on the *GridSpace2* platform, namely, the kernel execution code should be implemented in the method *execute()* (line 45). There, we can see the SEL and the data sending and receiving instructions. In line 50, the SWD1 object "can1" is instantiated and it implements the methods *Observation()*, *Collision()*, *Propagation()*, *BoundaryCondition()* and the other needed parameters. The methods send()/receive() allow to send/receive data to/from a declared conduit entrance. In one hand, lines 59-62 show how data are sent from SWD1 to the two gates' instances using the method send(). In

the other hand, lines 63-64 are waiting to receive data from the two inputs *f1_in* and *f2_in*.

B.5 The CxALite Java code of a coupling between two 1D canals and two gates

```
1  public static void main( String[] args ) throws InterruptedException {
2          CxA cxa = new CxA( "SW1D Simulation" );
3          // Create Kernels' Instances
4          cxa.addKernel(GateLightKernel.class, "Gate1" );
5          cxa.addKernel(GateLightKernel.class, "Gate2" );
6          cxa.addKernel(SW1DLightKernel.class, "SW1D1" );
7          cxa.addKernel(SW1DLightKernel.class, "SW1D2" );
8
9          //Setup entrance connections
10         cxa.connect("SW1D1.f1_out_right").to( "Gate2.f1_in_right").with( "conduit1" );
11         cxa.connect("SW1D2.f2_out_left").to( "Gate2.f2_in_left").with( "conduit2" );
12         cxa.connect("SW1D1.f2_out_left").to( "Gate1.f2_in_left").with( "conduit3" );
13         cxa.connect("SW1D2.f1_out_right").to("Gate1.f1_in_right").with( "conduit4" );
14         cxa.connect("Gate1.f1_out").to("SW1D1.f1_in").with( "conduit5" );
15         cxa.connect("Gate1.f2_out").to("SW1D2.f2_in").with( "conduit6" );
16         cxa.connect("Gate2.f2_out").to("SW1D1.f2_in").with( "conduit7" );
17         cxa.connect("Gate2.f1_out").to("SW1D2.f1_in").with( "conduit8" );
18
19         // execute
20         cxa.execute();
21     }
```

The kernels of the coupling schema are declared in lines 4-7. Similary to MUSCLE, all kernels must implememt a specific CxALite Java class. Lines 10-17 prepare the entrances connections between all the kernels. There, for sake of simplicity and optimization, the two outputs of the SWD1 submodel, connected to the same gate, are merged into one output. The execution of the computation is done in line 20 on the current machine.

C Details on implementation: Comptational biology

C.1 Particle swarm optimization

The particle swarm optimization was chosen for the optimization process in our GRN application. PSO is a population-based stochastic optimization technique [30]. It is inspired from the social behaviour of animals such as bird flocking or fish schooling. It has many similarities with evolutionary

computation techniques, such as genetic algorithms (GA), such as the fact the both algorithms are initialized with a population of random solutions and search for an optimum by updating generations. Unlike GA, however, PSO does not have any evolution operators such as crossover or mutation, resulting in a fewer number of parameters for the particles. In PSO, the potential solutions "fly" through the solution space by "following" the current optimum particles.

A simple example scenario could be bird flocking: a group of birds are randomly looking for food in an area. There is only one piece of food in the area being searched (a unique minimum/maximum in the solution space). None of the birds initially knows where the food is, however it is assumed that each bird knows how far it is to the food at each time. The best strategy to find the food in this scenario would be to follow the bird which is the closest to the food.

To simulate this, each particle has a position in the search space and a velocity. At every iteration, each particle is updated by following two or three "best" values. The first one is the best solution (fitness) it has achieved so far. This value is called *pbest*. Another "best" value that is tracked by the particle swarm optimizer is the best value obtained so far by any particle in a population. This best value is a global best and is called *gbest*. When a particle takes part of the population as its topological neighbours, the best value is a local best and is called *lbest*. After finding these values, the particle updates its velocity and positions as such:

$$v = v + c_1 rand()(pbest - present) + c_2 rand()(lbest - present) + c_3 rand()(gbest - present) \quad (1)$$

$$present = present + v \quad (2)$$

where v is the particle velocity, $present$ is the current particle position (current solution), $pbest$, $lbest$ and $gbest$ are defined as stated before, $rand()$ is a random number in the interval $(0, 1)$ and c_1, c_2 and c_3 are learning factors; usually $c_1 = c_2 = c_3 = 2$.

A simplified pseudocode of a typical PSO implementation is the following:

```
For each particle
    Initialize particle
End

Do
    For each particle
        Calculate fitness value
        If the fitness value is better than the pbest
            Set current value as the new pbest
    End

    Choose the particle with the best fitness value as the gbest
    For each particle
        Calculate particle velocity
        Update particle position
    End
While maximum iterations or minimum error criteria is not attained
```

We distinguish between two approaches inspired from evolutionary computing: insular evolutionary algorithms and cellular evolutionary algorithms:

- Insular evolutionary algorithms are based on a spatial or topological organization in which a genetic population is divided into subpopulations (islands, regions) that are optimized semi-independently from each other. In this approach [32] individuals periodically migrate between island subpopulations in order to overcome the problems associated with a single population becoming stuck in a local minimum and thus failing to find the global minimum.
- Cellular evolutionary algorithms are based on a spatially distributed population in which genetic interactions may take place only in the closest neighbourhood of each individual [29] [32]. Here, individuals are usually set up in a lattice-like topology structure.

Both of these approaches can be seen in Figure 18 where we have four

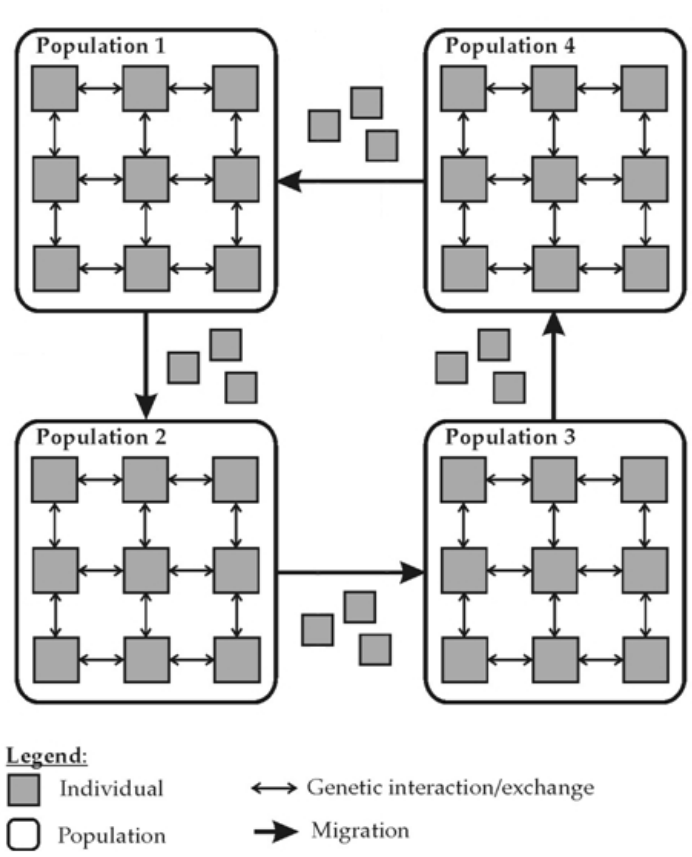

Figure 18: *PSO running with four islands*

islands, each containing a lattice of particles (each particle has four neighbours). The islands exchange particles in a ring topology.

C.2 Run-time view of the first GRN prototype

First, a short reminder on the terminology: each particle in the PSO implementation will contain a set of parameters, so it will be a potential solution to our optimization problem. In MMS terms, our islands will be submodels, so the coupling issue will only apply to how the communication across our islands will occur. We employ a number of mappers and conduits for enabling the communication between our submodels (see Figure 15 for the coupling topology). We list the content of the *cxa.rb* for a simulation running two islands:

```
abort "this is a configuration file for to be used with the MUSCLE utility"
  if __FILE__ == $0

cxa = Cxa.LAST

cxa.add_kernel('island1', 'Island')
cxa.add_kernel('island2', 'Island')
cxa.add_kernel('optimalIn', 'OptimalIn')
cxa.add_kernel('fitEval', 'FitnessEvaluator')
cxa.add_kernel('reinitiate', 'Reinitiate')

cs = cxa.cs

cs.attach('island1' => 'island2'){
  tie("sendParticlesFromIsland", "getParticlesInIsland")}
cs.attach('island2' => 'island1'){
  tie("sendParticlesFromIsland", "getParticlesInIsland")}

cs.attach('island1' => 'optimalIn'){
  tie("sendGbestFromIsland", "getGBestInOptimalIn")}
cs.attach('island1' => 'optimalIn'){
  tie("sendContinueSimulationFromIsland", "getContinueSimulationInOptimalIn")}

cs.attach('island2' => 'optimalIn'){
  tie("sendGbestFromIsland", "getGBestInOptimalIn")}
cs.attach('island2' => 'optimalIn'){
  tie("sendContinueSimulationFromIsland", "getContinueSimulationInOptimalIn")}

cs.attach('optimalIn' => 'fitEval'){
  tie("sendGBestsFromOptimalIn", "getGBestsInFitnessEvaluator")}

cs.attach('fitEval' => 'reinitiate'){
  tie("sendContinueSimulationFromFitnessEvaluator", "getContinueSimulationInReiniti
cs.attach('fitEval' => 'reinitiate'){
  tie("sendGBestFromFitnessEvaluator", "getGBestInReinitiate")}
```

```
cs.attach('reinitiate' => 'island1'){
  tie("sendGBestFromReinitiate1", "getGbestInIsland")}
cs.attach('reinitiate' => 'island1'){
  tie("sendContinueSimulationFromReinitiate1", "getContinueSimulationInIsland")}

cs.attach('reinitiate' => 'island2'){
  tie("sendGBestFromReinitiate2", "getGbestInIsland")}
cs.attach('reinitiate' => 'island2'){
  tie("sendContinueSimulationFromReinitiate2", "getContinueSimulationInIsland")}

m = Muscle.LAST
m.env["Xmx"] = "768m"
m.env["Xms"] = "16m"
m.env["Xss"] = "16m"
m.add_classpath File.dirname(__FILE__)+"/../build"
m.add_libpath File.dirname(__FILE__)+"/../build"
m.add_classpath File.dirname(__FILE__)+"/../build/libsmblj.jar"
m.add_classpath File.dirname(__FILE__)+"/../build/JavaPlot.jar"
m.add_classpath File.dirname(__FILE__)+"/../build/xpp3_min-1.1.4c.jar"
m.add_classpath File.dirname(__FILE__)+"/../build/xstream-1.3.1.jar"
```

Inside the islands, each particle communicates with its immediate neighbours at each iteration. Particles are grouped in a lattice-like structure, where each particle will have 4 neighbours. If the groups of particles in the islands are too big, the communications can be set to be less frequent. So, for example, every iteration there is a send/receive between each particle and its neighbours, then every 100 iterations particles send their best value and receive the global best within the whole island, and then every 300 iterations the global for all islands is communicated to every individual.
At the start of the simulation, each island initializes its particles with a random position. It does so by reading the content of an SBML file, which contains a skeleton model of the GRN we want to reverse-engineer, and extracting the list of parameters. It then starts executing the PSO loop (updating the velocities and positions of the particles). After the required number of steps has been achieved, the islands exchange particles among themselves. For this, the particles that will be sent will are represented as

String objects and sent via MUSCLE's routines.
After this, the islands send their *gbest* values to the *Fitness Evaluator* submodel via the *Optimal In* mapper to evaluate whether the simulation can stop (if any of the *gbest* values has a smaller fitness than the desired error). There is also a maximum number of simulations defined. If this number was reached, the *Fitness Evaluator* will signal that the simulation will stop. Otherwise, if none of the optimal particles has a small enough fitness, it will send the best of these particles as the new *gbest* to the islands, through the *Reinitiate* mapper.
The only input that the system needs is the SBML file containing the equations that need to be optimized and the experimental data used as comparison in the fitting step. The output is an SBML model containing the optimized values of the parameters in our equations, which in our case will be the global best value among all islands. The program will stop either when a preset maximum number of iterations has been reached (no output will be given in this case, or the output will be the report of a failure), or when the fitness of the global best is under a preset threshold value, in which case the program will also output this value.

D Description of the Multiscale Modeling Language (MML)

D.1 Introduction

The idea behind the multiscale modeling language (MML), to have a language to uniformly describe multiscale models and their computational implementation on abstract level, was first proposed by Falcone et al. [36]. Within the MAPPER project this idea can be tuned and expanded given the example applications of the project participants.
Two representations have been selected for a multiscale modeling language: a graphical one, simply denoted by gMML, and a textual one, using XML, called xMML. gMML can capture a large part of the model description, however, for a complete and exact description xMML is also necessary.
Both gMML and xMML have their roots in the Complex Automata formalism [37, 38] which describes multiscale coupled cellular automata. Notably, from this formalism the submodel execution loop (SEL) and the scale sep-

aration map (SSM) are re-used.

D.2 Terminology

In this section we will use a following terminology:

MML the high level concept of the language that describes single scale submodels and their couplings. It is a concept for modelers and has several representations.

xMML the XML representation of MML that contains all information about application structure.

gMML the graphical representation of MML that contains only part of information about application structure, useful for modelers and application developers.

SSM a graphical scale separation map aids visual inspection of scales used and the separation between them in a multiscale model. SSM is meant for modelers that should be able to present an model to their judgement in a way that serve the visual goal. SSM is not meant for computational (execution) purposes.

D.3 Multiscale modeling language elements

D.3.1 Submodel execution loop

The submodel execution loop (SEL) regulates and unifies the execution flow within submodels. It is formed from the basis that all submodels will have an initialization, possibly multiple iterations of solving and finalization. Moreover, during each of these phases we can define whether the submodels may send or receive data from other submodels. In pseudocode, the SEL is as follows:

```
t:= t_0
f:= f_init(t)
WHILE{t - t_0 < T}
  O_i(f, t)
  t := t + Delta_t
  f := S(f, t)
```

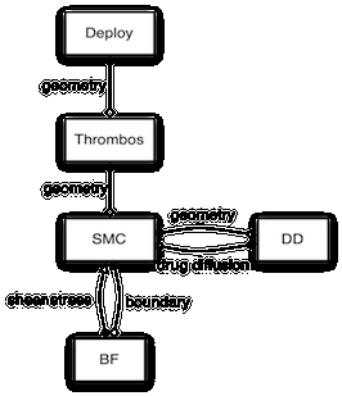

Figure 19: *A simplified example of the gMML of ISR3D.*

```
  f := B(f, t)
ENDWHILE
```

Operators are $\mathbf{f}_{\text{init}}$, $\mathbf{O_i}$, $\mathbf{B}$, $\mathbf{S}$, and $\mathbf{O_f}$ for initialization, intermediate observation, boundary condition calculation, solving step, and final observation respectively. Operators $\mathbf{f}_{\text{init}}$, $\mathbf{B}$, and $\mathbf{S}$ are allowed to receive data and $\mathbf{O_i}$ and $\mathbf{O_f}$ to send data. T is the end time for the submodel and theta is the time step. Coupling templates are defined as couplings between the operator of one submodel to the operator of another.

D.4 Graphical representation (gMML)

In the graphical representation MML, UML icons are used to show different couplings. An simplified example of the In-stent restenosis 3D model (ISR3D) is shown in Figure 19.
First of all, a submodel instance is shown as a rectangular box with its name inside. If there are multiple instances of the same submodel each instance should have its own unique name and have a suffix between angled brackets of the submodel name, like so: instanceName¡SUBMODEL¿.
A coupling between two submodels is shown as an connector with a tail and head styled differently given the operators of the coupling template. See Figure 20 for which operators correspond to which tail or head icon.
A label can be added to a connector to show what data is transferred in this coupling.
Originally the coupling had to be placed on a certain side of the submodel

Figure 20: *The icons head and tail icons of gMML.*

[36]. Due to difficulty in reading, this constraint has been lifted.

D.5 Textual representation (xMML)

The XML format xMML is currently at version 0.3.2. XML was chosen as a well-known, human-writable, machine-readable standard that promotes interoperability. It is described by both a W3C XML Schema and a Document Type Definition.
In this version, xMML contains information about the model such as a version and a description. Then each of the datatypes, converters and submodels used in that model are defined, where each denote one implementation of a its type.
The submodel definition consists of a description, its scales, ports and implementation details. The scales are specified per dimension and give an indication of the scale separation involved. The ports are sending or receiving and coupled to a specific SEL operator, and send a specific datatype. Implementation details may give a scheduler hints at where to schedule different submodels.
When the submodels are defined, the coupling topology may be created, defining first submodel instances and then couplings between those instances. Submodel instances may override the scales that were given during submodel definition. Couplings are defined with the sending port of one submodel and the receiving port of the other. As the datatypes sent over the couplings have a defined size, a communication cost can be estimated for each of the couplings.
The xMML format thus specifies the entire multiscale model and contains almost all information to be able to run a multiscale application.

References

[1] B. Chen, J. R. G. Evans, H. C. Greenwell, P. Boulet, P. V. Coveney, A. A. Bowden, and A. Whiting. A critical appraisal of polymer–clay nanocomposites. Chem. Soc. Rev., 37(3):568–594, 2008.

[2] Distibuted computing environments - gridspace technology, 2011.

[3] Alfonso Caiazzo, David JW Evans, Jean-Luc Falcone, Jan Hegewald, Eric Lorenz, Bernd Stahl, Dinan Wang, Jörg Bernsdorf, Bastien Chopard, Julian Gunn, Rod Hose, Manfred Krafczyk, Pat Lawford, Rod Smallwood, Dawn Walker, and Alfons G Hoekstra. A Complex Automata approach for In-stent Restenosis: two-dimensional multiscale modeling and simulations. Journal of Computational Science, 2(1):9–17, March 2011.

[4] David J W Evans, Patricia V Lawford, Julian Gunn, D Walker, D R Hose, R H Smallwood, B Chopard, M Krafczyk, J Bernsdorf, and Alfons 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, 366:3343–3360, 2008.

[5] Eric Lorenz. Multi-scale Modeling with Complex Automata: In-stent Restenosis and S PhD thesis, Universiteit van Amsterdam, August 2010.

[6] Hannan Tahir, Alfons G Hoekstra, Eric Lorenz, Patricia V Lawford, D Rodney Hose, Julian Gunn, and David JW Evans. Multiscale simulations of the dynamics of in-stent restenosis: impact of stent deployment and design. Interface Focus, 1(3):365–373, April 2011.

[7] David P. Coster, Vincent Basiuk, Grigori Pereverzev, Denis Kalupin, Roman Zagorksi, Roman Stankiewicz, Philippe Huynh, and Frédéric Imbeaux. The European Transport Solver. IEEE Transactions on Plasma Science 38:2085-2092, 2010.

[8] G.T.A. Huysmans, J.P. Goedbloed and W.O.K. Kerner. Isoparametric Bicubic Hermite Elements for the solution of the Grad-Shafranov Equation. CP90 Conf. on Comp. Physics Proc, World Scientific Publ. Co, 371, 1991.

[9] B. Scott. GEM – An Energy Conserving Electromagnetic Gyrofluid Model. arXiv: physics.plasm-ph/0501124, 2004.

[10] G Manduchi, F Iannone, F Imbeaux, G Huysmans, J Lister, B Guillerminet, P Strand, L Eriksson, and M Romanelli. A universal access layer for the Integrated Tokamak Modelling Task Force. Fusion Engineering and Design 83: 462-466, 2008.

[11] Marco D Mazzeo and Peter Coveney. HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries. Comput. Phys. Commun., 178(12):894–914, 2008.

[12] Marco D Mazzeo, Steven Manos, and Peter Coveney. In situ ray tracing and computational steering for interactive blood flow simulation . Comput. Phys. Commun., 181:355–370, 2010.

[13] O. Marcou and S.E. Yacoubi and B. Chopard. A bi-fluid lattice Boltzmann model for water ow in an irrigation channel. Cellular Automata **4173**:373-382 (2006).

[14] O. Marcou and B. Chopard and S.E. Yacoubi. Modeling of Irrigation Channels - a Comparative Study. Int. J. of Modern Phys. C **18**(4):739-748 (2007).

[15] O. Marcou and B. Chopard and S.E. Yacoubi. Lattice Boltzmann Model for the Simulation of Flows in Open Channels with Application to Flows in a Submerged Sluice Gate . J. Irrig. and Drain. Engrg. **136**(12):809 (2010).

[16] P. van Thang and B. Chopard and L. Lefèvre and D.A. Ondo and E. Mendes. Study of the 1D lattice Boltzmann shallow

water equation and its coupling to build a canal network. J. Comput. Phys. **229**(19):7373-7400 (2010).

[17] C. Körner and M. Thies and T. Hofmann and N. Thürey and U. Rüde. Lattice Boltzmann Model for Free Surface Flow for Modeling Foaming. J. Stat. Phys. **121**:179 (2005).

[18] Multiscale APPlications on European e-infRastructures. For more information look at *http://www.mapper-project.eu/web/guest/hydraulics*.

[19] *http://gs2.mapper-project.eu:1234/models_list*.

[20] *http://gs2.mapper-project.eu:18080/mad/*.

[21] *https://github.com/paradigmatic/CxALite*.

[22] Laboratoire de Conception et d'Intégration des Systèmes, Grenoble-EISAR Institute of Technology, Valence. For more information look at *http://www.grenoble-inp.fr/grenoble-institute-of-technology-9224.kjsp*.

[23] Pham van Thang, Bastien Chopard, Laurent Lefvre, Diemer Anda Ondo, and Eduardo Mendes. Study of the 1d lattice boltzmann shallow water equation and its coupling to build a canal network. Journal of Computational Physics, 229(19):7373 – 7400, 2010.

[24] Salmon R. The lattice boltzmann method as a basis for ocean circulation modeling. Journal of Marine Research, 57:503–535(33), 1 May 1999.

[25] Paul J. Dellar. Nonhydrodynamic modes and a priori construction of shallow water lattice boltzmann equations. Phys. Rev. E, 65(3):036309, Feb 2002.

[26] J.G. Zhou. Lattice Boltzmann Methods for Shallow Water Flows. Springer, 2004.

[27] I. Cantone, L. Marucci, F. Iorio, M. A. Ricci, V. Belcastro, M. Bansal, S. Santini, M. di Bernardo, D. di Bernardo, and

M. P. Cosma. A yeast synthetic network for in vivo assessment of Reverse-Engineering and modeling approaches. Cell, 137(1):172–181, April 2009.

[28] P. D'haeseleer, S. Liang, and R. Somogyi. Genetic network inference: from co-expression clustering to reverse engineering. Bioinformatics, 16(8):707–726, 2000.

[29] M. Gorges-Schleuter. ASPARAGOS an asynchronous parallel genetic optimization strategy. In International Conference on Genetic Algorithms, page 422–427, 1989.

[30] J. N. Kennedy and R. C. Eberhart. Particle swarm optimization. IEEE Internet Computing, 1995.

[31] S. C. Materna, J. Nam, and E. H. Davidson. High accuracy, high-resolution prevalence measurement for the majority of locally expressed regulatory genes in early sea urchin development. Gene Expr Patterns, 10(4-5):177–184, June 2010.

[32] Y. Shi, H. Liu, L. Gao, and G. Zhang. Cellular particle swarm optimization. Information Sciences, 2010.

[33] M. T. Swain, J. J. Mandel, and W. Dubitzky. Comparative study of three commonly used continuous deterministic methods for modeling gene regulation networks. BMC bioinformatics, 11(1):459, 2010.

[34] J. Vohradský. Neural network model of gene expression. The FASEB Journal, 15(3):846, 2001.

[35] L. F. A. Wessels, E. P. Van Someren, and M. J. T. Reinders. A comparison of genetic network models. In Pacific Symposium on Biocomputing, page 508–519, 2001.

[36] Jean-Luc Falcone, Bastien Chopard, and Alfons G Hoekstra. MML: towards a Multiscale Modeling Language. Procedia Computer Science, 1(1):819–826, 2010.

[37] Alfons G Hoekstra, Alfonso Caiazzo, Eric Lorenz, and Jean-Luc Falcone. Complex automata: multi-scale modeling with

coupled cellular automata. In Alfons G Hoekstra, J Kroc, and Peter M A Sloot, editors, Simulating Complex Systems by Cellular Automata, pages 29–57. Springer-Verlag Berlin, Heidelberg, 2010.

[38] Alfons G Hoekstra, Eric Lorenz, Jean-Luc Falcone, and Bastien Chopard. Toward a Complex Automata Formalism for MultiScale Modeling. International Journal for Multiscale Computational Engineering, 5(6):491–502, 2007.