D5.2 MAPPER vertical integration plan

Project acronym: *MAPPER*

Project full title: Multiscale Applications on European e-Infrastructures

Grant agreement no.: 261507

Due-Date:	M24
Delivery:	M24
Lead Partner:	PSNC
Dissemination Level:	PU
Status:	Draft
Approved:	
Version:	2.2

DOCUMENT INFO

Date and version number	Author	Details
06-09-2011 v0.1	Bartosz Bosak	Initial version
07-09-2011 v0.2	Stefan Zasada	Added section about AHE
07-09-2011 v0.3	Katarzyna Rycerz	Added section about GridSpace
09-09-2011 v0.4	Joris Borgdorff	Added section about MUSCLE
14-09-2011 v0.5	Stefan Zasada	Extended part about AHE
14-09-2011 v1.0	Bartosz Bosak	Corections to the general structure
15-09-2011 v1.1	Katarzyna Rycerz	Modifications related to WP8 components
15-09-2011 v1.2	Bartosz Bosak	Minor corrections
23-09-2011 v1.3	Bartosz Bosak	Addressed Katarzyna Rycerz's comments - minor corrections
29-09-2011 v1.4	Bartosz Bosak	Addressed Ilya Saverchenko's comments
30-09-2011 v1.5	Bartosz Bosak	Language correction
24-08-2012 v2.0	Bartosz Bosak	Initial version of second release
07-09-2012 v2.1	Krzysztof Kurowski	Final editing
19-09-2012 v2.2	Bartosz Bosak	Addressed internal reviewers comments

TABLE OF CONTENTS

List of Tables

List of Figures

1 Executive summary

This deliverable is a living document presenting plans and the current status of vertical software and e-Infrastructure services integration in support of large scale multiscale computing defined by the MAPPER user communities. The aim of this document is to ensure a consistent integration between components laying on different levels in the technical MAPPER architecture to satisfy specific application scenarios' needs defined in particular by WP4. Additionally, this deliverable aims to follow e-Infrastructure polices and best practices required for productive-quality software deployment and support presented in WP6, especially in the context of PRACE (PaRtnership for Advanced Computing in Europe) and with EGI InSPIRE (Integrated Sustainable Pan-European Infrastructure for Researchers in Europe). This document focuses on particular relations between multiscale applications provided by the MAPPER user communities, the dedicated MAPPER software for efficient and flexible distributed execution of such applications, and finally the existing, improved or new e-Infrastructure-level middleware services and tools. The vertical integration plans have been driven by two defined general scenarios targeted on two classes of multiscale applications, namely loosely-coupled and tightly-coupled. As each group of applications requires advanced and in some places also different capabilities, the MAPPER middleware services and tools must provide a wide range of new features dedicated for multiscale computations

Various aspects of vertical integration of MAPPER components are discussed in subsequent sections of the deliverable, starting from the general requirements for MAPPER and its general architecture, through the complete analysis of the required functions for the two defined types of applications, the implementation details of scenarios, ending with the productive software tests and deployment of scenarios on e-Infrastructures, in particular on PRACE and EGI sites.

The current version of the document describes several services and tools classified in Description of Work - Anex I [1] as ”fast-track” and ”deep-track” components, namely: MUSCLE, GridSpace, QosCosGrid, AHE. In order to provide the most useful information, during the whole project lifetime the deliverable will be periodically updated and adjusted to available at this moment project's results.

2 Contributors

- PSNC: Bartosz Bosak, Krzysztof Kurowski, Mariusz Mamonski, Tomasz Piontek
- UvA: Joris Borgdorff
- Cyfronet: Katarzyna Rycerz, Eryk Ciepiela
- UCL: Stefan Zasada, Derek Groen
- LMU: Ilya Saverchenko

3 List of abbreviations

Item	Description
AHE	Application Hosting Environment
AR	Advance Reservation
BES	Basic Execution Service
BF	blood flow
CPMD	Car-Parrinello Molecular Dynamics
DD	drug diffusion
DEISA	Distributed European Infrastructure for Supercomputing Applications
DRMAA	Distributed Resource Management API
EGEE	Enabling Grids for E-sciencE
EGI	European Grid Initiative/Infrastructure
GRAM	Grid Resource Allocation and Management
GRN	gene-regulatory networks
GS2	Grid Space 2
HPC	High-Performance Computing
JSDL	Job Submission Description Language
LRMS	Local Resource Management System
MAD	MUSCLE Application Designer
MaMe	MUSCLE Memory
MHD	magnetohydrodynamics
MML	Multiscale Modeling Language
MPI	Message Passing Interface
MTO	MUSCLE Transport Overlay
MUSCLE	Multiscale Coupling Library and Environment
NGS	National Grid Services
PBS	Portable Batch System
PRACE	Partnership for Advanced Computing in Europe
PL-Grid	Polish National Grid Initiative
QCG	QosCosGrid
SMC	smooth muscle cell
UNICORE	Uniform Interface to Computing Resources
xMML	XML representation of MML

Table 1: Abbreviations

4 High-level vertical integration plan

A general view on vertical integration plan of MAPPER is presented in Figure 1. Although the plan was established on the basis of the expert knowledge and experiences of the partners, it may be slightly altered during the project. The components highlighted in green are being developed and extended to fulfill certain needs of distributed multiscale computations in MAPPER, whilst the violet components are already productively exploited software components and will not be changed or modified in the MAPPER project.

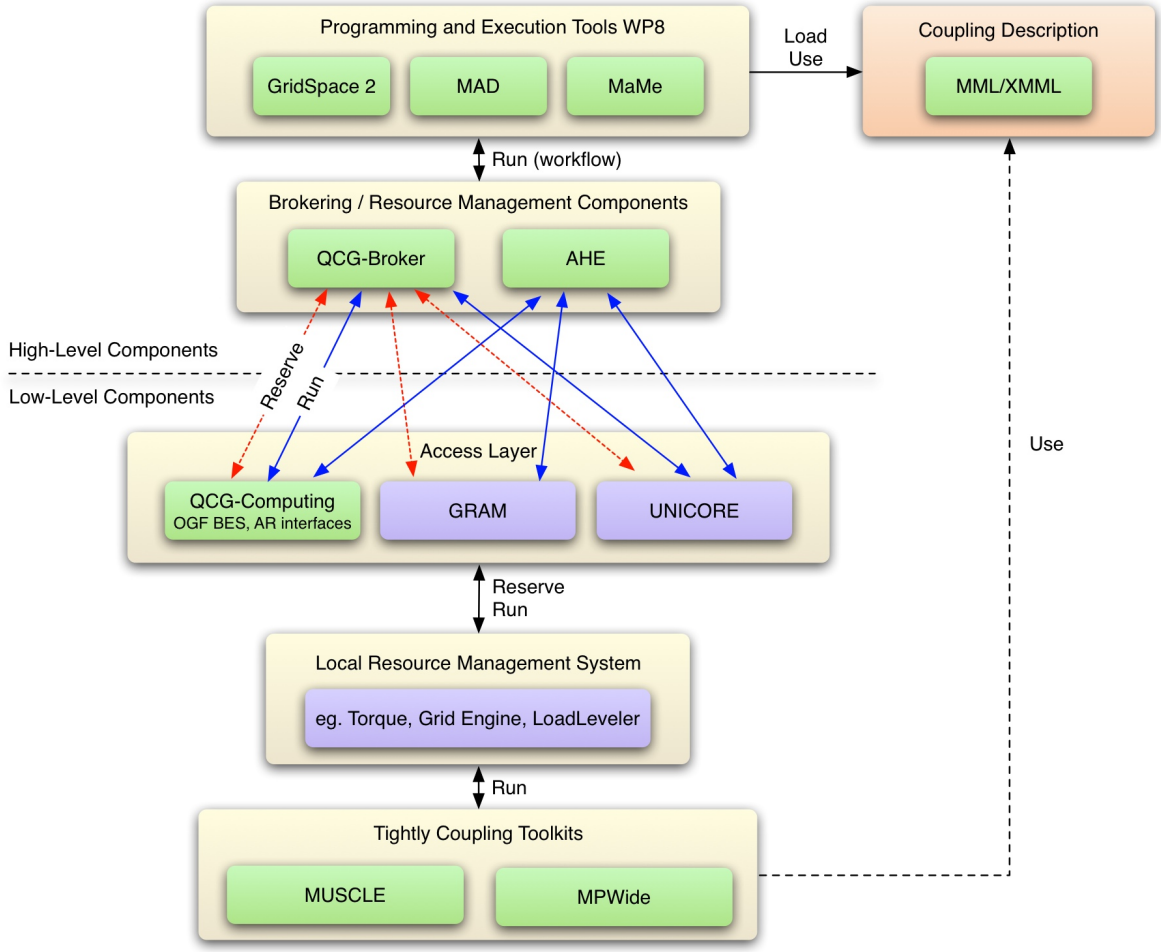


Figure 1: General vertical integration plan

Two levels of components separated in the figure by the dashed line represent a natural distinction between low-level components that must be installed locally at the resource providers side and high-level components, specific for distributed computing infrastructures, e.g. Grid or Cloud environments. The first of the top-placed components in the proposed architecture is GridSpace 2 platform belonging to the group of programming and execution tools described in details in D8.2 [32]. Within the MAPPER project GridSpace 2 plays a role of user interface and workflow execution engine. This second feature is particularly important in scope of MAPPER as it is responsible for the processing of loosely-coupled computations. Multiscale Application Desinger (MAD) and Mapper

Memory (MaMe) extend the high-level functionality by providing multiscale application composition tool and semantical storage for MAPPER data, respectively. The Brokering and Resource Management components, AHE and QCG-Broker, create an interoperability layer between GridSpace 2 and low-level components. QCG-Broker offers functions for co-allocating resources which are highly desirable for multi-cluster application runs. The access layer located at the top of the low-level part of the architecture consists of the three components: QCG-Computing BES/AR, Grid Resource Allocation and Management (GRAM) and UNICORE, offering remote interfaces to underlaying Local Resource Management Systems (LRMS) such as Torque [21], PBS Pro [22], Sun Grid Engine [19] or LoadLeveler [26].
The elementary (single-scale) applications are run by selected LRMS and may use various techniques for parallel execution, e.g. OpenMPI, OpenMP, CUDA. In turn, the possibility of running tightly-coupled multiscale scenarios, composed of many single-scale models is provided by two platforms: MUSCLE and MPWide. Owing to the support for advance reservation and co-allocation functions, cooperating single-scale models may be run simultaneously on several sites.
It was decided to use XML representation for the description of coupling for both MAPPER scenarios. A new XML document type, called xMML is based on Multiscale Modeling Language (MML) described in details in [6] as well as deliverables D4.1 [29] and D8.1 [31]. Nevertheless, in the current phase of the project, the abstract xMML description with respect to loosely-coupled scenarios is transformed into form of GridSpace scripts, whilst in tightly-coupled applications it is replaced by the CxA files being the main input file for the MUSCLE framework.
It is noteworthy that in the proposed architecture both defined MAPPER scenarios, i.e. loosely-coupled and tightly-coupled, may be combined in real use-cases. In other words, depending on the xMML definition, the particular multiscale application may be executed as a workflow in GridSpace and simultaneously use MUSCLE toolkit on the lowest level. The detailed information how it is realized is included in D8.2 [32].

5 Loosly coupling platform and GUI - GridSpace 2

The loosely-coupled type of applications, in which an entire application consists of several steps in foreseeable order, is well matched the GridSpace operation model. GridSpace 2 is based on the notion of exploratory programming where each experiment consists of a number of snippets creating a workfow. Each snippet may be written in a different programming language and executed independently. In this way, time-consuming experiments do not have to be started from scratch each time a modification is made during the development. The web portal of the Experiment Workbench assists users in the iterative development of simulation of workflows with the use of popular scripting languages. The Experiment Execution Environment, which is a layer underneath the Experiment Workbench, evaluates snippets and executes them on remote sites when required. The GS2 has been integrated with AHE and QCG-Broker services to offer advanced capabilities for job execution across many sites in Europe. The detailed description of running MUSCLE jobs through QCG services is presented in the Appendix A.
Moreover, GS2 provides a rich set of interpreters to develop glue code with, what is important for a user having to perform various analysis of the intermediate results between subsequent steps of

a multi-model simulation. The Perl interpreter is supporter, as is the bash shell to run the actual computations, but other interpreters like Ruby, Python or AWK (depends on the availability of the packages on the target machine) might also be provided if needed. All of them may be used in combination in a single computational experiment (like a workflow of tools in a pipeline). GS2 supports running jobs through PBS using PBS gems (applications), which, when compiled on the target machine, allow submitting computational runs directly from script languages like Ruby, Perl or Python. Alternatively, whole interpreter process can be executed wrapped as a PBS job.
The additional requirements for GS useful for the loosely-coupled applications are the multi-machine login capabilities and the multi-site file transfer. The former allows being simultaneously present on two different machines (required e.g. to perform the loosely-coupled scenario consisting of a full 3-step nano computation for the polymer study) in a single Experiment Workbench window. The other, when the multi-machine login is present, will allow to transfer files between these machines in a simple drag-and-drop manner. This features was presented in the first prototype of WP8 tools (see details in D8.2 [32]).

6 Tightly coupling platforms

6.1 MUSCLE

The MUSCLE coupling library and environment [7, 13] was selected to execute tightly-coupled multiscale models on distributed resources in MAPPER.
MUSCLE offers support for scales and it has two base elements: the kernel, which contains the submodel code, and the conduit, which transfers data. When communication within a submodel occurs, it is regulated by the model developer. Conduit filters may filter data that is transferred through conduits, and a developer may implement them or reuse a pre-defined one. Operations on data that need multiple inputs or multiple outputs but do not qualify as a submodel are called mappers, but are implemented as a kernel. [6]
Within kernels, parallel code can be used, for instance by using Java threads or MPI technique.
A MUSCLE model can be executed by starting an instance of MUSCLE on each machine that should be used and starting submodel instances on those MUSCLE instances. If a single machine setup is needed, just one MUSCLE instance needs to be started, otherwise they can communicate using TCP/IP. Machines that have no TCP/IP connections to others that run MUSCLE can only be accessed if there is an interactive node with a TCP/IP connection that controls the executable on that machine.
When a MUSCLE model is executed, each of the submodels may be started separately. A tightly-coupled scenario is enabled by having loops within the submodels that wait for messages from other submodels at certain points. When no more messages can arrive, for instance when the submodel that should send the message quits, the waiting submodel can also quit.

6.2 MPWide

The second platform that supports the tightly-coupled computations in MAPPER is MPWide communication library for distributed message passing [8]. MPWide is trivial to install and allows

distributed applications to obtain superior wide area communication performance, compared to approaches such as cross-site MPI or single-stream TCP communication libraries. MPWide is currently used to enable the coupling in a multiscale scenario of intracerebral bloodflow, where the performance and responsiveness of the coupling communications is particularly important. One of the first steps that was taken here is implementing MPWide APIs for FORTRAN and Python applications. Further more, the MPWide is going to be integrated into the MUSCLE Transport Overlay (MTO), thereby it will greatly enhance the cross-site communication performance of MUSCLE.

7 Middleware e-Infrastructure services

7.1 QosCosGrid

QosCosGrid (QCG) [2] is a set of middleware services and tools dedicated for dealing with computationally intensive large-scale, complex and parallel simulations that are often impossible to run within a single computing cluster. Basing on QosCosGrid, resources of different administrative domains may be virtually welded via Internet into one powerful computing resource. Applications running on the basis of QosCosGrid, in particular GridSpace platform, may use transparently a variety of common technologies, such as OpenMPI [3] or ProActive[4], typically used for parallel execution only on a single machine. Within the MAPPER project, the QosCosGrid stack has been integrated with the MUSCLE coupling library enabling flexible execution of multiscale applications. Since the common multiscale simulations require simultaneous execution, the QosCosGrid capabilites for advance-reservations and co-allocation of various types of resources which are unavailable in gLite or UNICORE, provide a good opportunity to run demanding multiscale scenarios on many clusters.

7.1.1 QCG-Broker

QCG-Broker [2] is a meta-scheduling system, which in principle allows developers to build and deploy resource management systems for large-scale distributed, multi-cluster computing infrastructures. The main goal of QCG-Broker was to manage the whole process of remote job submission and advance reservation to various batch queueing systems and subsequently to underlying clusters and computational resources. QCG-Broker deals with various meta-scheduling challenges efficiently, e.g., co-allocation, load-balancing among clusters, remote job control, file staging support or job migration.
The QCG-Broker service has been designed as an independent component which can take advantage of various low-level core and grid services and existing technologies, such as QCG-Computing or QCG-Notification, as well as various grid middleware services such as gLite, Globus or UNICORE.
The XML-based job definition language supported by QCG-Broker is Job Profile. The Job Profie schema makes it relatively easy to specify the requirements of large-scale parallel applications together with the complex parallel communication topologies. Consequently, application developers and end users are able to run their experiments in parallel over multiple clusters as well to

perform various benchmark-based experiments as alternative topologies are taken into account during meta-scheduling processes in QCG-Broker.

7.1.2 QCG-Computing

Another key service in the MAPPER integration plan is QCG-Computing assigned to the low-level components. The QCG-Computing service (previously SMOA-Computing) [2] is an open implementation of SOAP Web Service for multi-user access and policy-based job control routines by various Distributed Resource Management systems. It uses Distributed Resource Management Application API (DRMAA) [5] to communicate with the underlying DRM systems. The flexible QCG-Computing design and implementation support different plugins and modules for external communication, authentication, authorization as well as interactions with accounting infrastructures and other external services. QCG-Computing service is compatible with the OGF HPC Basic Profile [16] specification, which serves as a profile over the Job Submission Description Language (JSDL) [17] and OGSA Basic Execution Service [18] Open Grid Forum standards. It also offers remote interface for Advance Reservations management, and a support for basic file transfer mechanisms. Additionally, in order to provide co-allocation support, QCG-Computing may be integrated with QCG-Broker - such a solution is exploited in MAPPER scenarios.
So far, the service has been successfully tested with many Distributed Resources Management systems, i.e.: Sun (Oracle) Grid Engine (SGE) [19], Platform LSF [20], Torque/PBSPro [21], PBS Pro [22], Condor [23], Apple XGrid [24] and Simple Linux Utility for Resource Management (SLURM) [25]. The Advance Reservations capabilities were exposed for SGE, LSF and Maui [27] (a scheduler that is typically used in conjunction with Torque) systems.

7.2 AHE

The Application Hosting Environment, AHE [9], developed at University College London, provides simple desktop and command line interfaces, to run applications on resources provided by national and international grids, in addition to local departmental and institutional clusters. It does not display the details of the underlying middleware in use by the grid to a user. In addition, a mobile interface for Windows Mobile based PDAs is available, and an iPhone interface is in development. The AHE is able to run applications on UNICORE, Globus and QCG grids, meaning that a user can use a single AHE installation to access resources for example from the UK NGI, Polish NGI and PRACE. Development of an EGI connector for AHE is currently conducted.
The AHE is designed to allow scientists to run unmodified, legacy applications on grid resources quickly and easily, manage the transfer of files to and from the grid resource and monitor the status of the application. The philosophy of the AHE is based on the fact that very often a group of researchers will want to access the same application, but not all of them will possess the skill or inclination to install the application on remote grid resources. In the AHE, an expert user installs the application and configures the AHE server, so that all participating users can share the same application. This community model draws a parallel with the modus operandi of numerous scientific application communities.
The AHE client is easily installed on an end users machine, requiring only that they have a Java

installation and an X.509 certificate for the grid, which they want to access. The client package contains both GUI and command line clients which interoperate, allowing jobs launched with the GUI client to be manipulated with the command line tools and vice versa, and application workflows to be easily constructed.
Within the MAPPER AHE is connected to the GridSpace workflow management system, in order to allow AHE hosted applications to be orchestrated as workflow components using GridSpace. AHE is used to hide such aspects of the complexity of running applications on the grid as where application binary is located, what environment variables need to be set to run the application, and even the back end interface used to communicate with grid resource. This simplifies the process of creating and running a workflow. The workflow designer does not need to be aware of where the application resides or how to launch it, and they simply issue a command to AHE to run the tool. GridSpace contains an extension that makes use of the AHE's client API in order to launch and monitor applications. Further more, AHE is able to make use of reservations created by QCG-Broker and also submit jobs via the QCG-Computing service.

8 Application scenarios

In order to provide scientists the most useful environment for developing and running multiscale simulations, the MAPPER consortium defined two general use-case scenarios corresponding to the two basic coupling types (tightly- and loosely-coupled). In the first phase of the project two representative applications were selected and deployed on the MAPPER infrastructure: the loosely-coupled application which utilize CPMD and LAMMPS codes to proceed nano-materials simulations and the tightly-coupled one, ISR3D, based on MUSCLE library from the physiology science. Both applications were demonstrated during the first project review on PM12. During the next period, the set of applications adapted to the multi-scale scenarios has iteratively grown and shortly MAPPER will support all 7 selected applications from 5 science fields. Thanks to the gained knowledge and established good-practices, we hope the further applications, external to the project, will be supported in near future.
In the context of this section, we wish to introduce the deployment schemas of particular applications on the MAPPER infrastructure and thus present also the details of two basic scenarios. The detailed description of the applications is out of scope of this document and is a part of deliverable D4.1 [29]. In turn, the adaptation of applications to the MAPPER environment is presented in a deliverable D7.1 [30].

8.1 Nano-materials

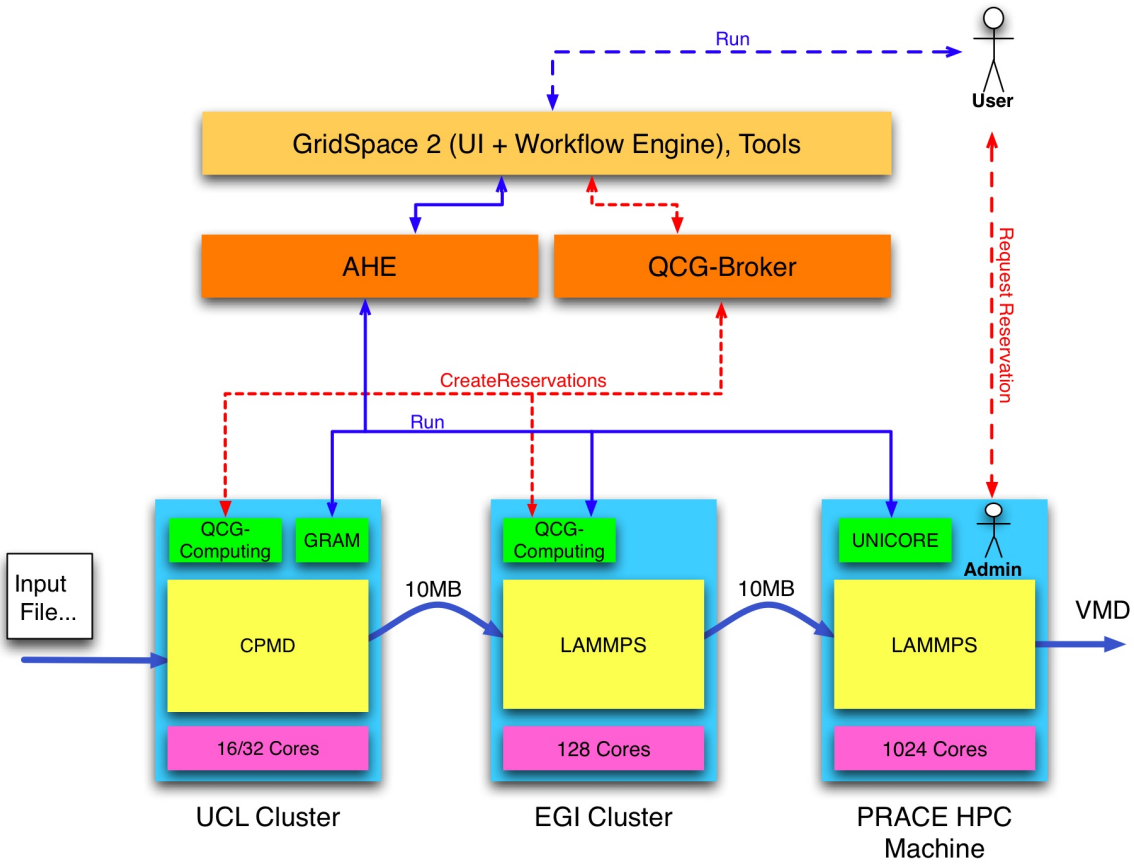


Figure 2: Nano-materials application scenario

The first application is taken from the field of materials science and implements loosely-coupling paradigm of multi-scale computing. The loosely-coupled scheme consists of three levels of simulation. The lowest level simulates the electronic degrees of freedom, using the Car-Parrinello Molecular Dynamics (CPMD) code [11]. The atomic charges calculated at this step are passed to the initial models simulated using LAMMPS classical molecular dynamics code [12]. Finally, Coarse-Grained Molecular Dynamics (CGMD) simulations are performed using data received from previous step, again with the use of LAMMPS code.

The three components of the loosely-coupled application scenario are deployed on resources appropriate to their requirements. As it is presented on Figure 2, first, the CPMD application is executed on small cluster taking as an input a file located on a filesystem. The data processed at this step is transformed by GridSpace scripts into a form of input to the LAMMPS MD code and transferred to an EGI resource running LAMMPS. Once the first LAMMPS model has been completed, its output is similarly processed into a form that can be used by the second LAMMPS model, and the data transferred to a PRACE HPC resource. At the end, the result of computations may be visualized by a VMD software [10]. The size of the transfered data between the core models is not large, approximately reaching 10MB.

As it was mentioned, the sequential execution of the different scale models is managed by the

GridSpace workflow execution engine. The Application Hosting Environment is an interoperability layer offering an access to various types of computing resources. In turn, QCG-Broker is responsible for co-reserving resources to ensure a proper order of the execution of the submodels. This task is critical since a reservation of PRACE resource for processing of the last submodel must be created significant advance — as the first one; thus the reservations periods of the other two submodels must follow parameters of the PRACE reservation. It can also be noticed that the reservation of a PRACE machine is currently established outside the MAPPER components stack. It is believed that a further interconnected development of MAPPER and PRACE will allow removing this limitation.
The detailed sequence diagram showing interactions in the planned implementation of the loosely-coupled scenario for the first MAPPER review is presented in the Appendix B. The core part of the computation consists of the three steps corresponding to the main application phases and invokes one after the other. The order of creating advance reservations is important: in general, the period of reservation of HPC resource, which is created at the beginning, determines the start times and end times of the other two reservations.

8.2 Instent Restenosis 3D

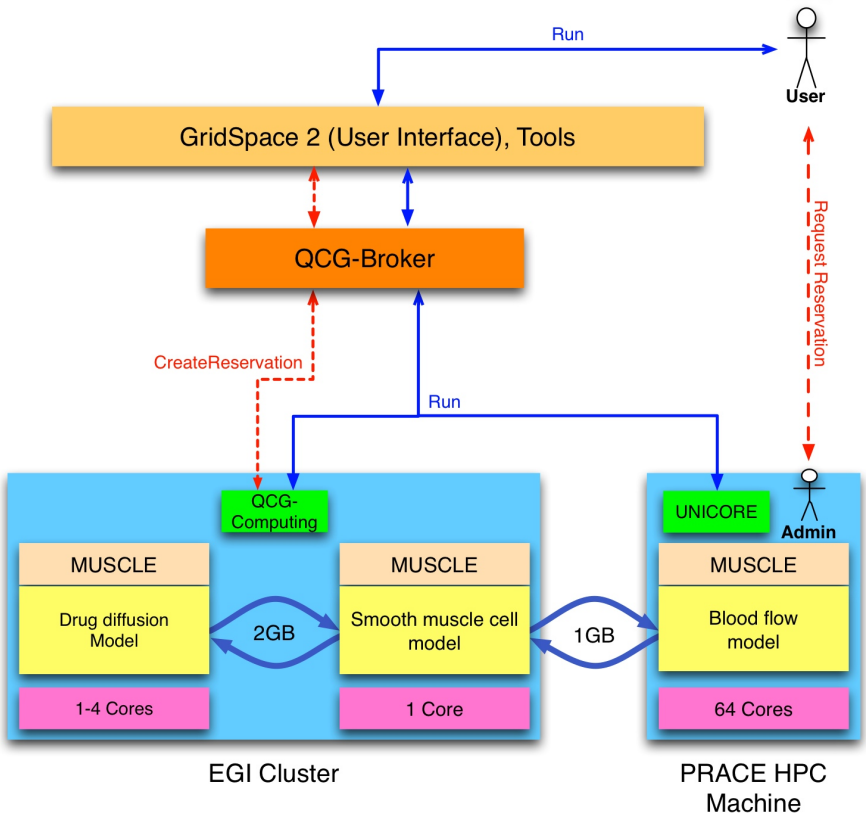


Figure 3: Instent Restenosis 3D application scenario

The three-dimensional In-stent Restenosis model (ISR3D) belongs to the group of MAPPER tightly-coupled applications. ISR3D allows 3D simulation of a stent deployment in the coronary artery

and subsequent processes. The objective of the model is to help understand restenosis and to indicate improvements in the stent design. The simplified ISR3D model considered in this scenario consists of the three submodels: blood flow (BF), drug diffusion (DD) and smooth muscle cell proliferation (SMC) (For details see D4.1 [29]). After the initialization routine, which is done in a loosely-coupled fashion, the model enters a tightly-coupled loop, where SMC, BF and DD are simultaneously computed. At some points of the compound calculations, the submodels exchange and synchronize data. In a typical run it may be about 1-2 GB of information.

The current version of the integration scheme of ISR3D application with the MAPPER components is presented in the Figure 3. In the opposite to the loosely-coupled scenario, where the coupling is realized at the highest level by GridSpace which executes a workflow, in this scenario the routines responsible for submodels synchronization come from the MUSCLE library and are integrated at the programing level with the codes of the particular submodels' kernels. All interactions with the Access Layer services, including operations needed to reserve and co-allocate resources, are performed by the QCG-Broker service. Note that the currently presented integration scheme slightly differs from the scheme described in the previous release of the the document and now the DD and SMC modules are deployed on a single machine. This deployment seems to be more optimal in practice.

The way of the execution of tightly-coupled application differs significantly from the previous use-case. This difference is clearly shown in the Appendix C presenting a sequence diagram of the tightly-coupled scenario proposed for the first review of the project. One can note that the main part of the computations - marked by the dashed line - consists of the three MUSCLE-based models running in parallel and exchanging information among themselves. In contrast to the loosely-coupled scenario, the coupling is realized on the application level, not in the GridSpace workflow engine, thus the vertical interaction steps between the components are limited significantly.

8.3 Irrigation Canals

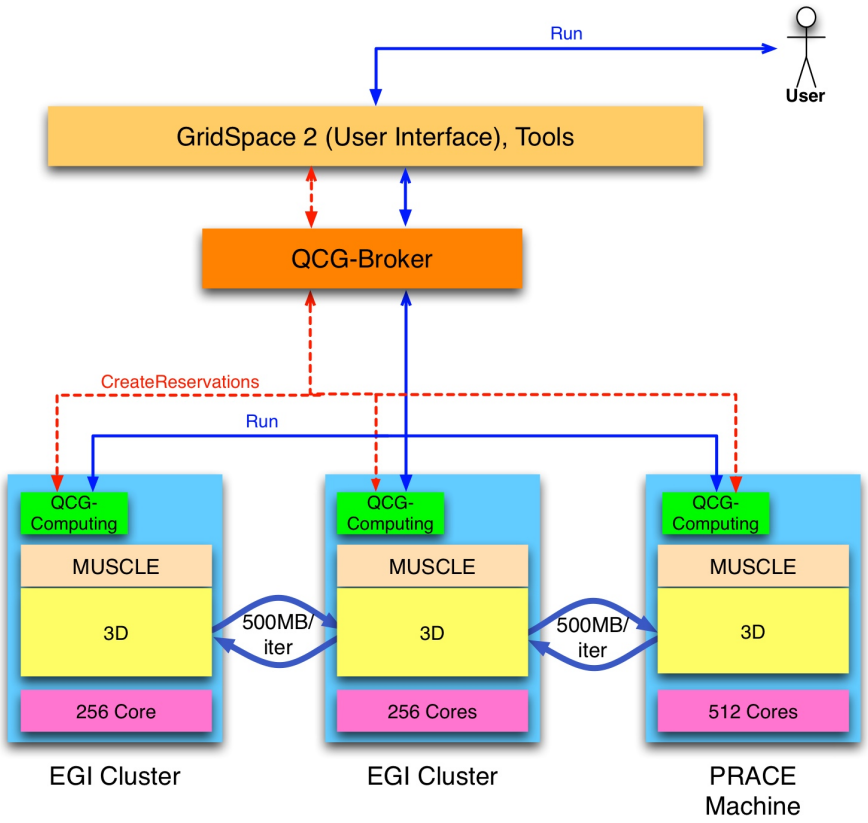


Figure 4: Irrigation Canals application scenario

Modeling of a network of irrigation canals is a topic of the next MAPPER application scenario. The problem to be solved by the application 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 external demands or perturbations can be, and respecting constraints such as water height.

The target Irrigation Canals scenario consists of four single-scale modules coupled in a tightly way. The first module is based on the 1-dimensional shallow water equation and is used to describe long canal sections. The second module, based on 2-dimensional shallow water, describes branching regions or pools in which the water height varies from the left to the right side. To describe in details the flow around gates or to describe the transport of sediments that can deposit along the canal the Free Surface 3D model is used in third module. The last module is responsible for modeling of transport of sedimentations that strongly influences the water flow, i.e. reduces the efficiency of the irrigation network.

The currently available version of Irrigation Canals integrates many 3D modules that deals with disjunctive sections of analyzed canal (4). The coupling is based on the MUSCLE library and utilize MTO, thus the computations may be executed in parallel on many physical machines.

Because of the maturity as well as advanced multiscale methodology included in the application, the Irrigation Canals has been selected to detailed demonstration during the second (PM24) MAP-

PER review. The sequence diagram included in Appendix D presents plan for this evaluation.

8.4 Fusion - Equilibrium Stability Workflow

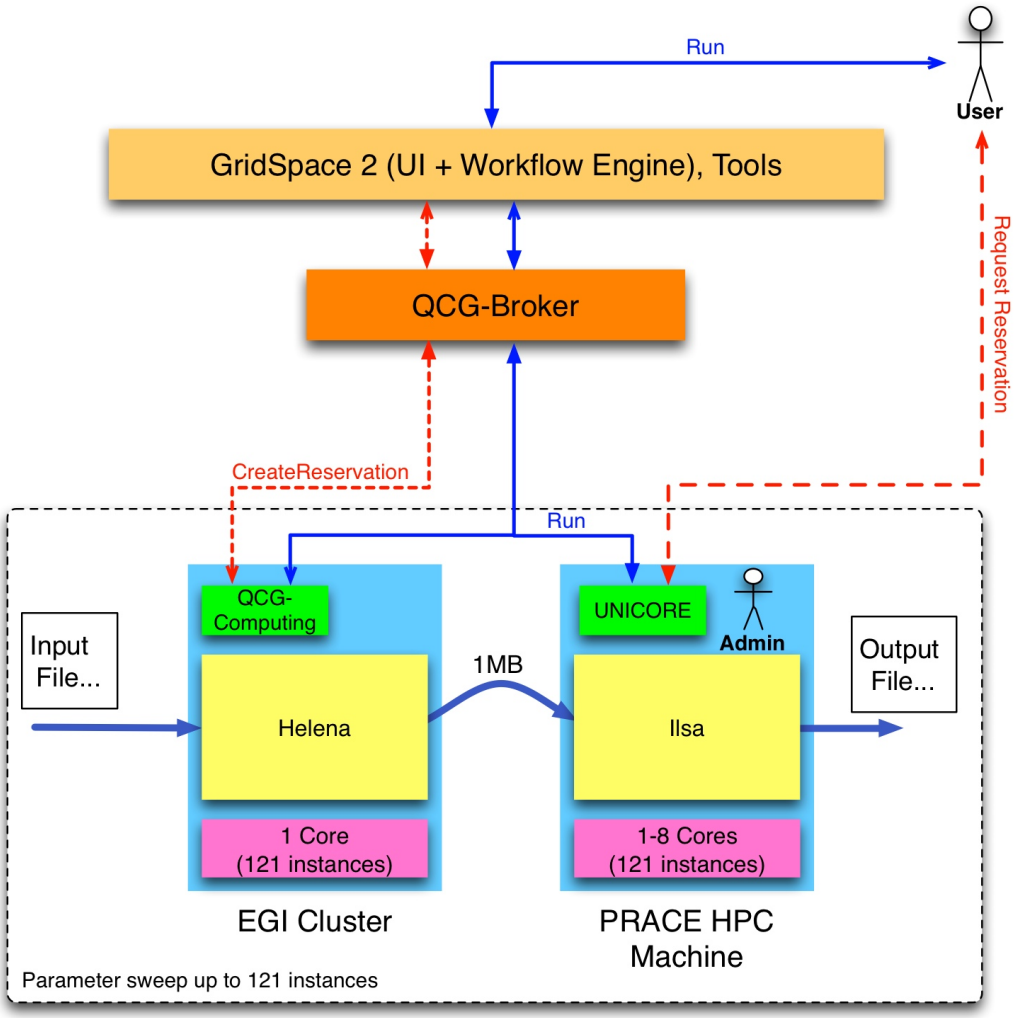


Figure 5: Equilibrium Stability Workflow application scenario

The equilibrium stability workflow application is one of the scenarios used to simulate aspects of nuclear fusion processes. The application consists of two loosely-coupled subcodes: a magnetohydrodynamics (MHD) equilibrium code (HELENA) and a linear MHD stability code (ILSA). The resource requirements to run the codes are limited, however in a real environment, to improve the statistical accuracy of simulations, the workflow should be executed simultaneously with up to 121 instances.

The Equilibrium Stability calculations fall into group of MAPPER loosely-coupled applications (see Figure 5). As so, the workflow execution will be be steered by GridSpace 2 which will move input/output files from one module to another. The creation of resource reservations as well as running of co-allocated tasks will be delegated to QCG components and UNICORE (in case of

PRACE resources).
The two machines were selected for the initial deployment of the scenario, i.e. EGI cluster for HELENA and the PRACE HPC machine for ILSA. Taking into account the restrictions of PRACE policy, there will be a need to independently, in advance, create a reservation on HPC resource.
In the current phase of the project there are ongoing works to integrate GridSpace 2 with QCG-Broker.

8.5 Fusion - Transport Turbulence Equilibrium

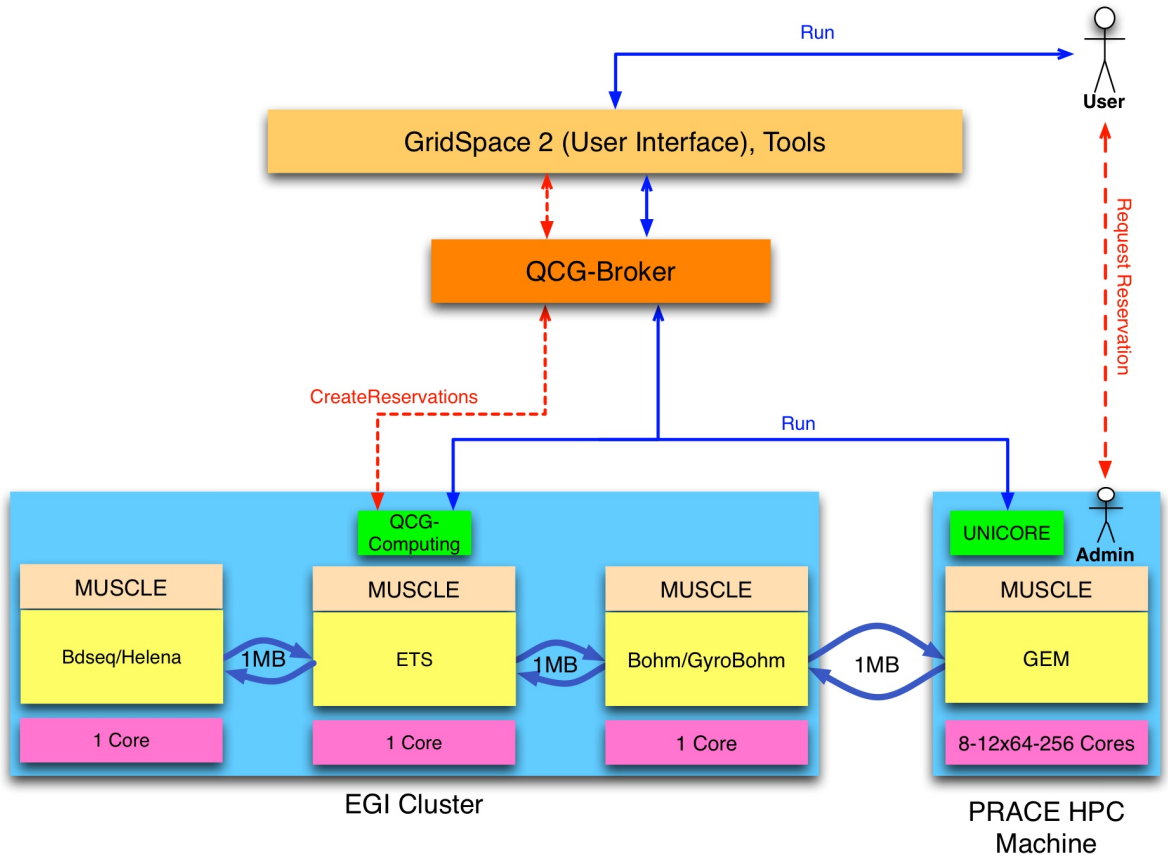


Figure 6: Transport Turbulence Equilibrium application scenario

The next application from fusion community simulates the simplified and approximate full fusion core in a nuclear fusion reactor. The current version of simulation is composed of four main modules: 2D equilibrium solver - Bdesq or Helena, 1D transport code - ETS, Bohm/GyroBohm code and most demanding 3D gyrofluid turbolence code - GEM. All the selected components are coupled in a tight fashion using the MUSCLE library.
The proposed integration plan of the scenario is presented in Figure 6. Consequently, the execution of the simulation will be initiated by GridSpace 2, which will employ the QCG components to create reservations of resources and run tightly-coupled modules in a coordinated way. Since the first three modules are computed only on one core each, they will be deployed on a single EGI cluster.

The fourth module, GEM, will be deployed on a more powerful PRACE machine and will utilize even as much as 3072 cores per run. Again, the limitation of this scenario is a fact that the reservation of PRACE resource must be established prior to the submission of the simulation in GridSpace 2.

8.6 Patient-specific whole brain blood flow simulations

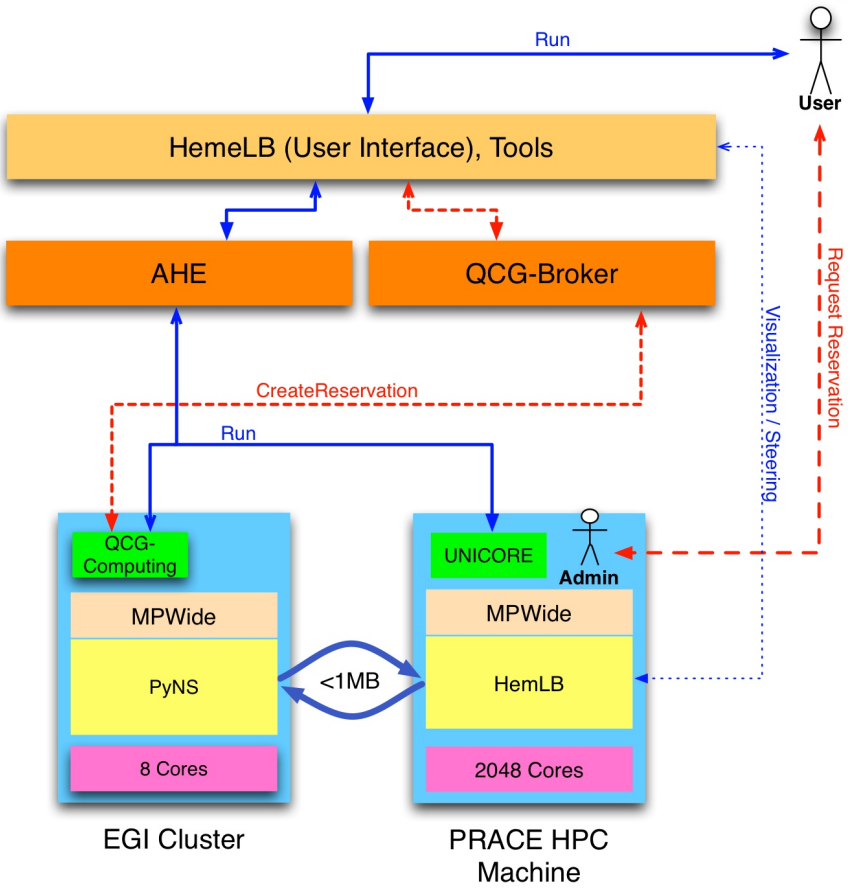


Figure 7: Patient-specific whole brain blood flow simulations application scenario

Neurovascular blood flow simulations that can support clinical neurosurgery are the topic of the next MAPPER scenario. The main simulation model is here the lattice-Boltzmann code - HemeLB [14] that simulates fluid flow in the sparse topologies of the human brain. The core model employes PyNS tool to simulate blood flow outside of the region of direct interest at a coarse-grained resolution. Both models are tightly-coupled using the MPWide library. Since the support for live visualization of the blood flow in the brain is crucial in clinical environments, the HemeLB offers built in real-time rendering and steering capabilities.

In context of the for-coming second MAPPER review, the application will be deployed on two sites as it is presented on Figure 7: the HemLB code will be deployed on one of PRACE HPC machines and the PyNS tool will be deployed on a smaller EGI cluster. The blood flow simulations is currently not triggered by GridSpace 2, but with usage of the dedicated tools. This is because it is currently using high performance MPWide communication library, which is planned to be integrated with

MUSCLE (supported by GridSpace) in the third year. As a result, usage of GridSpace 2 is expected in third year. Nevertheless, the QCG as well as AHE services are already employed in the scenario to reserve resources and run tasks on the e-Infrastructures.

8.7 Reverse-engineering of gene-regulatory networks

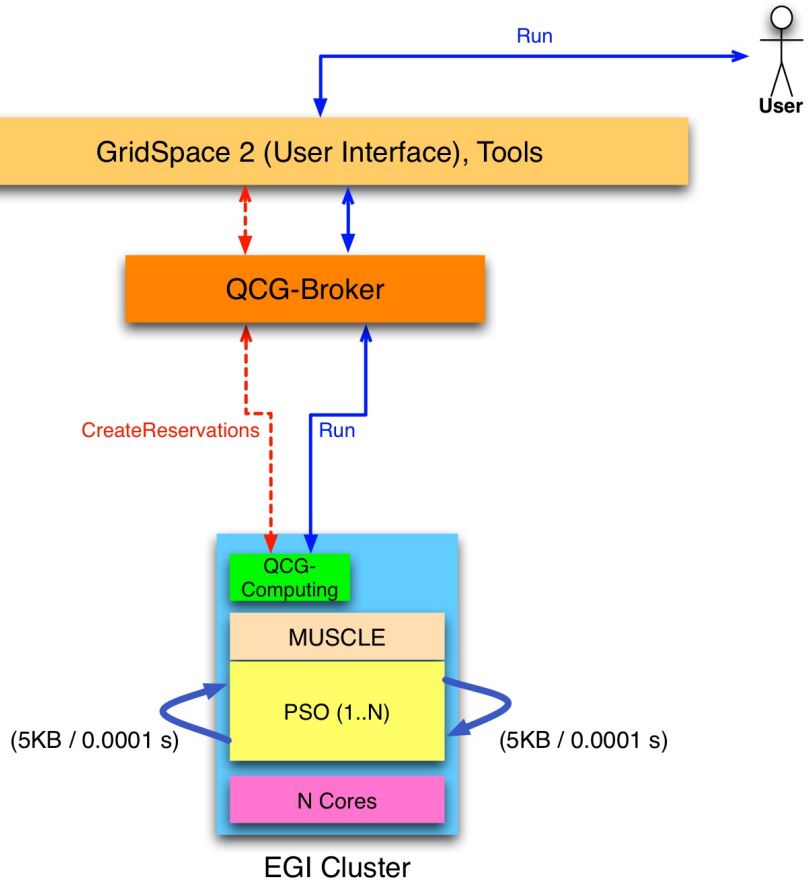


Figure 8: Reverse-engineering of gene-regulatory networks application scenario

The application that was selected initially by the computational biology community to multi-scale adaptation was the modeling of the bile acid and xenobiotic system. However, due to limited availability of published data, this application was temporarily abandoned and the efforts were shifted to reverse-engeenering of gene-regulatory networks (GRNs).

The GRNs application is built upon the reverse engineering strategy and employes the particle swarm optimization (PSO) [15] to find the best model that is able to adequately simulate the behaviour described by the gene expression data sets. The application utilizes currently only one single-scale model, proceeding particle swarm optimization, but it is going to be extended by further module in near future. The PSO module was already successfully integrated with the MUSCLE library and executed in a distributed environment.

Figure 8 presents plans regarding the application deployment and execution in its current form. The application will be started by a user within GridSpace 2 that indirectly utilize QCG components

to reserve resources and run the code on e-Infrastructure. The single instance of GRN application will be run in parallel and utilize numerous (N) cores.

9 Conclusions

The MAPPER vertical integration plan presented in this document incorporates the two basic coupling types of multiscale applications. The first type describes a concept of loosely-coupled computations. The key role in this model is performed by the GridSpace 2 platform, which executes a workflow of consecutive, essentially independent tasks. The second type is defined as a tightly-coupled as the multiscale coupling logic is moved to the lowest level of the MAPPER architecture and embedded using MUSCLE or/and MPWide in the application. When needed, both elementary coupling templates may be easily integrated within the MAPPER to use in more complex multiscale applications.

The first two applications described in the document, namely Nano-materials and ISR3D, were demonstrated during the first MAPPER review. A detailed description of this two presentations is depicted in a form of sequence diagrams attached in appendixes B and C.

The remaining applications, with accordance to the integration schemes described in this document, will be presented during the second review. The particular attention will be dedicated to the Irrigation Canals application that was selected to be demonstrated in details. The representative sequence diagram presenting coupling as well as deployment plan for this application is shown in appendix D.

Since the integration work as well as implementation of elementary applications are still ongoing the particular scenarios may be presented in a partially modified form, including also use of different resources. Nevertheless, any of the possible modifications will not influence significantly on the general vertical integration plan. All possible changes to the information presented in the current version of document will be included in its future releases.

A Generic support for existing and potential future MUSCLE applications in the MAPPER framework

A.1 Glossary

- (sub)module - abstract software component. Referenced by XMML. May be implemented by kernel(s)
- kernel - a basic MUSCLE application component. May be bundled as Java ARchive (JAR) jar or dynamically loadable library. May implement a (sub)module
- kernel repository - points to location with kernel bundles (jars or .so)
- MUSCLE - Multiscale Coupling Library and Environment
- MaMe - MApper MEmory - MAPPER registry service that store global information related to (sub)modules and their available implementations, but without site-specific information like class or library paths.
- MAD - Multiscale Application Designer
- GS2 - GridSpace 2 Experiment Workbench
- Environment Module[33] - Popular System tool that facilitate management of user shell environment. Widely used within EGI and PRACE e-infrastructures. Exploited in MAPPER to store site specific kernel environment like local class or library paths.
- QCG - QosCosGrid Middleware stack

A.2 Introduction

MAPPER users tools (MaMe, MAD, GS2) together with QosCosGrid services address MAPPER concept to allow for ad-hoc composition of multiscale applications from building blocks of MML entities that are registered and made available for application designers. MaMe, MAD and GS2 already collaborate with each other in order to be able to generate an arbitrary MUSCLE application in a form of GS2 experiment. Generic mapping of an arbitrary GS2 experiments to corresponding QCG QCG JobProfiles has been worked out in order to enable execution of all existing and potential future MUSCLE applications through QosCosGrid stack.

A.3 Integration description

Given with a real-world example of Canals Applications the respective GS2 experiment generated by MAD using data registered in MaMe looks as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<experiment>
    <metadata>
        <name>Canals Application</name>
```

```
        <author>Eryk Ciepiela</author>
        <creationDate>Fri Feb 10 21:30:15 CET 2012</creationDate>
    </metadata>
    <snippet interpreter="muscle-2010-01-11_13-51-27" id="a1">
        <output>a5</output>
        <output>a6</output>
        <output>a7</output>
        <output>a8</output>
        <output>a9</output>
        <code># configuration file for a MUSCLE CxA

# add build for this cxa to system paths (i.e. CLASSPATH)
m = Muscle.LAST

# configure cxa properties
cxa = Cxa.LAST

cxa.env[&quot;cxa_path&quot;] = File.dirname(__FILE__)

# declare kernels
cxa.add_kernel('SW1D_1B_001', 'com.unige.irigcan.kernel.d1.SW1D_1B_kernel')
cxa.add_kernel('SW1D_2B_005', 'com.unige.irigcan.kernel.d1.SW1D_2B_kernel')
cxa.add_kernel('SW1D_1B_011', 'com.unige.irigcan.kernel.d1.SW1D_1B_kernel')
cxa.add_kernel('SpillWay_015', 'com.unige.irigcan.junction.SpillWay_kernel')
cxa.add_kernel('LinearGate1B_021', 'com.unige.irigcan.junction.Gate_kernel')



# parameters
cxa.env['GLOBAL:iteration_number']=100
cxa.env['SW1D_1B_001:length']=7.0
cxa.env['SW1D_1B_001:width']=0.1
cxa.env['SW1D_1B_001:dx']=0.05
cxa.env['SW1D_1B_001:dt']=0.025
cxa.env['SW1D_1B_001:water_level']=0.1
cxa.env['SW1D_1B_001:borderconnection']='BEGIN'
cxa.env['SW1D_1B_001:is_GAUSS']=true
cxa.env['SW1D_1B_001:segment_index']=0
cxa.env['SW1D_1B_001:out_data']='SW1D_1B_out_data_001'
cxa.env['SW1D_2B_005:length']=7.0
cxa.env['SW1D_2B_005:dx']=0.05
cxa.env['SW1D_2B_005:dt']=0.025
cxa.env['SW1D_2B_005:width']=0.1
cxa.env['SW1D_2B_005:water_level']=0.1
cxa.env['SW1D_2B_005:borderconnection']='BOTH'
cxa.env['SW1D_2B_005:is_GAUSS']=false
cxa.env['SW1D_2B_005:segment_index']=0
cxa.env['SW1D_2B_005:out_data']='SW1D_2B_out_data_005'
cxa.env['SW1D_1B_011:length']=7.0
```

```
cxa.env['SW1D_1B_011:width']=0.1
cxa.env['SW1D_1B_011:dx']=0.05
cxa.env['SW1D_1B_011:dt']=0.025
cxa.env['SW1D_1B_011:water_level']=0.1
cxa.env['SW1D_1B_011:borderconnection']='BEGIN'
cxa.env['SW1D_1B_011:is_GAUSS']=true
cxa.env['SW1D_1B_011:segment_index']=0
cxa.env['SW1D_1B_011:out_data']='SW1D_1B_out_data_011'
cxa.env['SpillWay_015:Ls']=0.1
cxa.env['SpillWay_015:Rs']=0.75
cxa.env['SpillWay_015:hs']=0.1
cxa.env['SpillWay_015:width']=0.1
cxa.env['SpillWay_015:position_index']=0
cxa.env['SpillWay_015:out_data']='SpillWay_out_data_015'
cxa.env['LinearGate1B_021:theta']=0.005
cxa.env['LinearGate1B_021:alpha']=0.75
cxa.env['LinearGate1B_021:bg']=0.1
cxa.env['LinearGate1B_021:position_index']=0
cxa.env['LinearGate1B_021:out_data']='Gate_out_data_021'

# configure connection scheme
cs = cxa.cs
cs.attach('SW1D_1B_001' =&gt; 'LinearGate1B_021') {
  tie('f_out', 'left_in')
}
cs.attach('LinearGate1B_021' =&gt; 'SW1D_1B_001') {
  tie('left_out', 'f_in')
}
cs.attach('SW1D_2B_005' =&gt; 'LinearGate1B_021') {
  tie('left_out', 'right_in')
}
cs.attach('LinearGate1B_021' =&gt; 'SW1D_2B_005') {
  tie('right_out', 'left_in')
}
cs.attach('SW1D_2B_005' =&gt; 'SpillWay_015') {
  tie('right_out', 'left_in')
}
cs.attach('SpillWay_015' =&gt; 'SW1D_2B_005') {
  tie('left_out', 'right_in')
}
cs.attach('SW1D_1B_011' =&gt; 'SpillWay_015') {
  tie('f_out', 'right_in')
}
cs.attach('SpillWay_015' =&gt; 'SW1D_1B_011') {
  tie('right_out', 'f_in')
}
</code>
    </snippet>
```

```
    <inputOutput fileName="Gate_out_data_021" name="out_data" id="a5">
        <description></description>
        <producedBySnippet>a1</producedBySnippet>
    </inputOutput>
    <inputOutput fileName="SpillWay_out_data_015" name="out_data" id="a6">
        <description></description>
        <producedBySnippet>a1</producedBySnippet>
    </inputOutput>
    <inputOutput fileName="SW1D_2B_out_data_005" name="out_data" id="a7">
        <description></description>
        <producedBySnippet>a1</producedBySnippet>
    </inputOutput>
    <inputOutput fileName="SW1D_1B_out_data_001" name="out_data" id="a8">
        <description></description>
        <producedBySnippet>a1</producedBySnippet>
    </inputOutput>
    <inputOutput fileName="SW1D_1B_out_data_011" name="out_data" id="a9">
        <description></description>
        <producedBySnippet>a1</producedBySnippet>
    </inputOutput>
</experiment>
}
```

The experiment contains one snippet to be interpreted by MUSCLE. The snippet code is hence a CXA code. The snippet is expected to produce 5 files.
In GS2 Experiment Workbench user can pick QCGExecutor as the one to execute the snippet. QCGExector takes care of creating all needed files on User Interface (UI) host (as configured in QCGExecutor) using GridFTP and generate appropriate QCG JobProfile. QCG JobProfile contains entries respective to all output files with GridFTP URI as their storage destination.
After QCG JobProfile submission QCG-Broker distributes the computations over sites. After computation is done QCG-Broker attempts to transfers all the output files from all sites back to the UI host. As the output file names are unique QCG-Broker will find individual output file on at most one site so no overwriting will occur. Similarly input files can be handled: QCG-Broker can upload each input file to all sites and still no overwriting will occur. The example Job Profile would therefore look as follows:

```
<qcgJob appId="16512">
    <task persistent="true" taskId="muscle">
        <requirements>
            <topology>
                <processes processesId="SW1D_1B_001:SW1D_1B_005:LinearGate1B_009">
                    <processesMap slotsPerNode="2">
                        <processesPerNode>1</processesPerNode>
                        <processesPerNode>1</processesPerNode>
                    </processesMap>
                    <quirements>
                        <resourceRequirements>
                            <computingResource>
```

```
                        <hostParameter name="module">
                            <stringValue value="canals/PM20"/>
                        </hostParameter>
                    </computingResource>
                </resourceRequirements>
            </requirements>
        </processes>
    </topology>
</requirements>

<execution type="mapper">
    <executable>
        <application name="muscle2"/>
    </executable>
    <arguments>
        <value>16512.cxa.rb</value>
    </arguments>
    <stdout>
        <directory>
            <location type="URL">
              gsiftp://qcg.man.poznan.pl/~plggdyk/16512.output</location>
        </directory>
    </stdout>
    <stderr>
        <directory>
            <location type="URL">
              gsiftp://qcg.man.poznan.pl/~plggdyk/16512.error</location>
        </directory>
    </stderr>
    <stageInOut>
        <file name="16512.cxa.rb" type="in">
            <location type="URL">
              gsiftp://qcg.man.poznan.pl/~plggdyk/canals.cxa.rb</location>
        </file>
        <file name="SW1D_1B_out_data_001" type="out">
            <location type="URL">
              gsiftp://qcg.man.poznan.pl/~plggdyk/SW1D_1B_out_data_001</location>
        </file>
        <file name="Gate_out_data_009" type="out">
            <location type="URL">
              gsiftp://qcg.man.poznan.pl/~plggdyk/Gate_out_data_009</location>
        </file>
        <file name="SW1D_1B_out_data_005" type="out">
            <location type="URL">
              gsiftp://qcg.man.poznan.pl/~plggdyk/SW1D_1B_out_data_005</location>
        </file>
    </stageInOut>
</execution>
```

```
        <executionTime useReservation="false">
            <executionDuration>P0Y0M0DT5H0M</executionDuration>
        </executionTime>
    </task>
</qcgJob>
```

Inputs or outputs can be single files or directories. In latter case QCGExecutor will generate `<directory>` elements in QCG JobProfile instead of `<file>` elements. Such approach overcomes the problem of application-specific pre- and post-processing scripts needed in order to prepare input and output files prior to transfer.
The crucial part of the QCG JobProfile, in context of the multiscale applications, is the `<topology>` section which describes how the application should be spawned over multiple sites. The every `<processes>` element describes different process group. Two different process group can be run on two different sites concurrently. The process group description consist of:

- list of MUSCLE kernels to be started (kept in `processesId`),
- resource requirements (e.g. installed module, requested memory),
- candidate host - optional,
- local reservation identifier - optional.

The other problem is to ensure that all java JARs and native library files are available in the QCG job environment. This is solved with the help of Environment Modules[1]. Application developer, using QosCosGrid Community Modules[2], creates and registers module files for locally installed kernel. The example module file can look as follow:

```
#%Module1.0#####################################################################
proc ModulesHelp { } {
        global version

        puts stderr "\tThis module sets environment variables for Canals3d"
}

module-whatis "sets environment variables for Canals3d "



module add muscle2
module add openmpi/1.4.5-gcc-4.1.2-ib

set CANALS_HOME "/home/plgrid-groups/plggmuscle/canals3d"
setenv CANALS_HOME $CANALS_HOME

prepend-path PATH "${CANALS_HOME}"
```

[1] http://modules.sourceforge.net/
[2] http://www.qoscosgrid.org/trac/qcg-computing/wiki/ComunityModules

```
prepend-path MUSCLE_CLASSPATH "${CANALS_HOME}"
prepend-path MUSCLE_LIBPATH "${CANALS_HOME}"

set     version      "PM20"
```

The `MUSCLE_LIBPATH` and `MUSCLE_CLASSPATH` environement are interpreted by the MUSCLE2 command (similar like the `CLASSPATH` variable is interpreted by Java). Thus the CXA generated by MAD does not has to provide site specific information like: `m.add_classpath ...`, and `m.add_libpath`. Instead MaMe serves as a database of mappings between kernel and environment modules names. This information is used to add appropriate requirements section to the QCG JobProfile, e.g.:

```
...
<requirements>
    <resourceRequirements>
        <computingResource>
            <hostParameter name="module">
                <stringValue value="canals/MM"/>
            </hostParameter>
        </computingResource>
    </resourceRequirements>
</requirements>
...
```

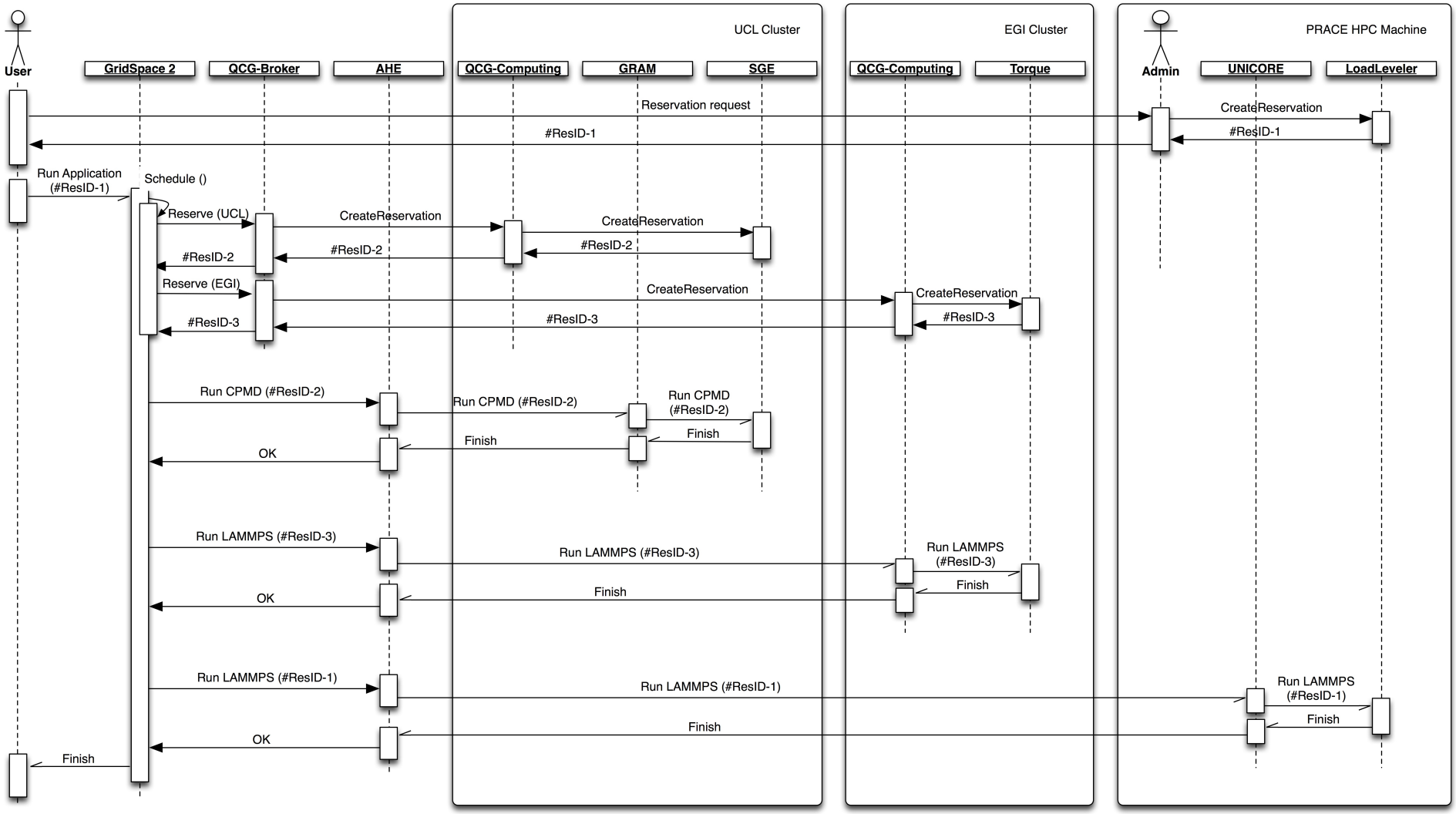
The QosCosGrid stack automatically selects site where the given module is installed and loads upon job startup.

A.4 Summary

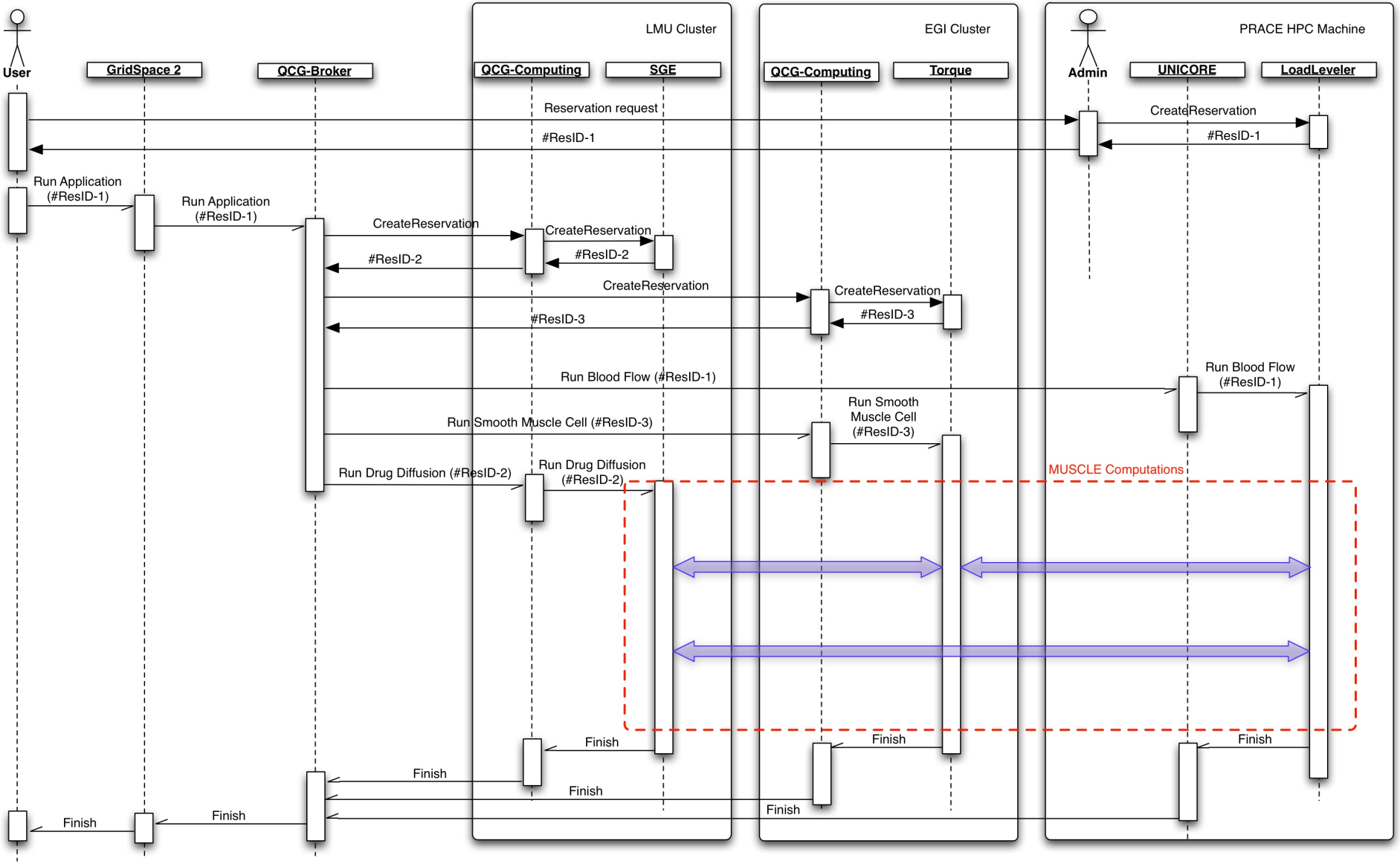
The presented solution of integration between components of MaMe, MAD, EW, QCG-Broker and QCG-Computing covers coherently all aspect of managing MUSCLE multiscale application lifecycle, namely: development, deployment, registration, discovery and submission. Moreover it meets the following requirements:

- site specific (i.e. local) information is stored only locally,
- global (sub)models meta-data is stored globally in MaMe registry thus preventing inconsistency,
- the information about locally installed modules is discoverable via the QCG-Computing service,
- QCG-Broker can automatically consider for scheduling only the sites where the given kernel is installed,
- environment modules that setup application environment can facilitate both local and remote simulation runs.

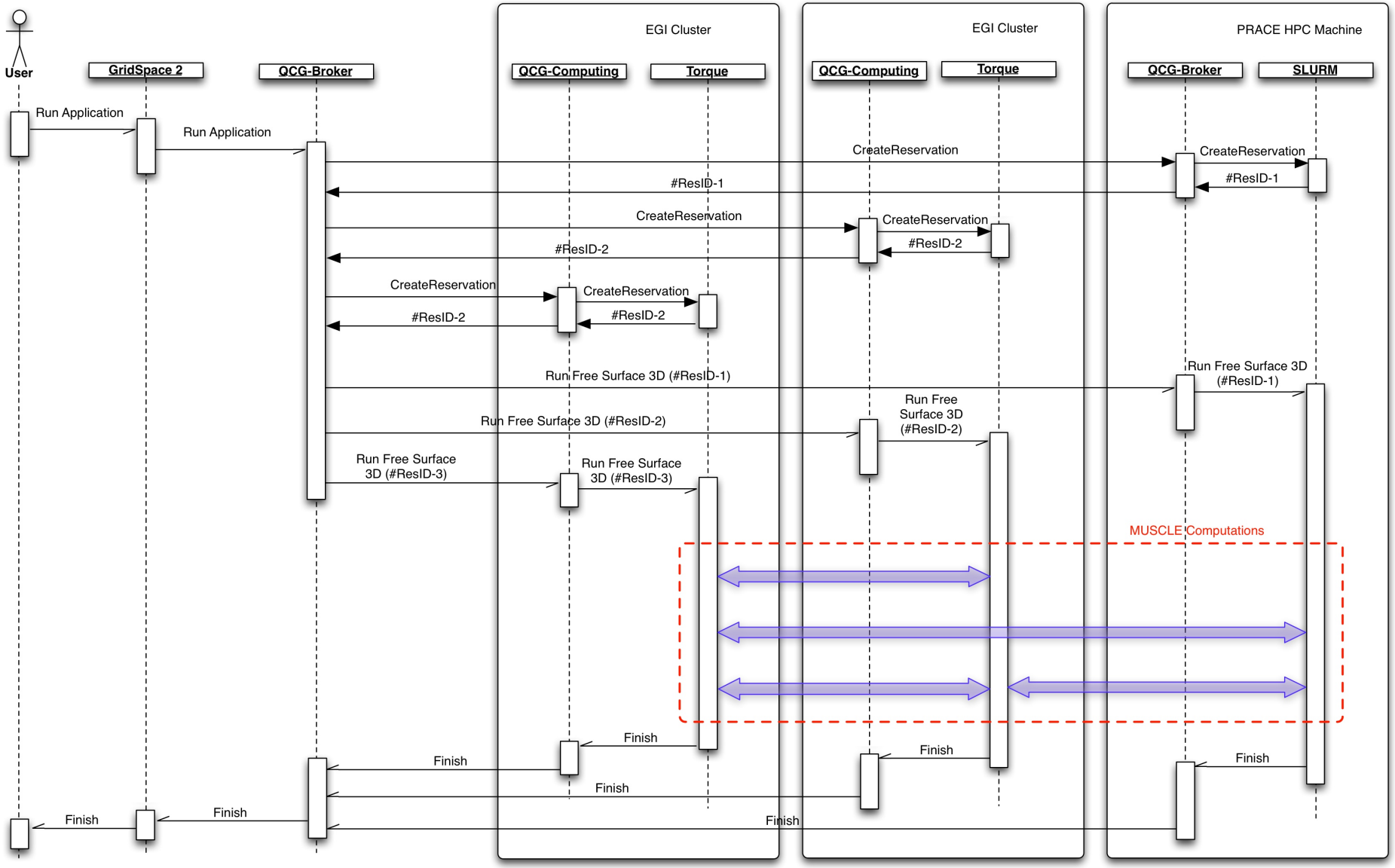
B Nano-materials sequence diagram



C Instent Restenosis 3D sequence diagram



D Irrigation Canals sequence diagram



References

[1] Multiscale Applications on European e-Infrastructures (Mapper) Project – Annex I, ”Description of Work”

[2] Krzysztof Kurowski, Tomasz Piontek, Piotr Kopta, Mariusz Mamonski, Bartosz Bosak. Parallel Large Scale Simulations in the PL-Grid Environment. Computational Methods in Science and Technology, 47-56 (2010)

[3] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett and Andrew Lumsdaine, et al. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. Recent Advances in Parallel Virtual Machine and Message Passing Interface. pp. 353-377 (2004).

[4] Caromel, D., Delbe, C., di Costanzo, A. and Leyton, M.: ProActive: an Integrated Platform for Programming and Running Applications on Grids and P2P systems. Computational Methods in Science and Technology, vol. 12, no. 1, pp. 69-77 (2006).

[5] Peter Troeger, Hrabri Rajic, Andreas Haas. Standardization of an API for Distributed Resource Management Systems. Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid07), Rio De Janeiro, Brazil, May 2007.

[6] Jean-Luc Falcone, Bastien Chopard, Alfons Hoekstra. MML: towards a Multiscale Modeling Language, Procedia Computer Science (2010), Volume: 1, Issue: 1, Publisher: Elsevier, Pages: 819-826.

[7] Jan Hegewald, Manfred Krafczyk, Jonas Tlke, Alfons Hoekstra and Bastien Chopard. An Agent-Based Coupling Platform for Complex Automata. Lecture Notes in Computer Science, 2008, Volume 5102/2008, 227-233.

[8] D. Groen, S. Rieder, P. Grosso, C. de Laat and P. Portegies Zwart. A light-weight communication library for distributed computing. Computational Science and Discovery vol. 3, no. 015002 (Aug 2010).

[9] P. V. Coveney, R. S. Saksena, S. J. Zasada, M. McKeown and S. Pickles, ”The Application Hosting Environment: Lightweight Middleware for Grid-Based Computational Science”, Comp. Phys. Comm., 176, 406-418 (2007).

[10] Humphrey, W., Dalke, A. and Schulten, K., “VMD - Visual Molecular Dynamics” J. Molec. Graphics 1996, 14.1, 33-38.

[11] The CPMD Consortium page, `http://cpmd.org/`.

[12] LAMMPS Molecular Dynamics Simulator, `http://lammps.sandia.gov/`.

[13] Jan, Hegewald. The Multiscale Coupling Library and Environment (MUSCLE), `http://muscle.berlios.de/` (2010).

[14] Mazzeo and Coveney. HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries. Computer Physics Communications (2008) vol. 178 (12) pp. 894 - 914.

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

[16] GFD 114-HPC Basic Profile, Version 1.0, http://www.ogf.org/documents/GFD.114.pdf.

[17] GFD 56 - Job Submission Description Language (JSDL) Specification, Version 1.0 - http://www.gridforum.org/documents/GFD.56.pdf.

[18] GFD 108 - OGSA Basic Execution Service Version 1.0, http://www.ogf.org/documents/GFD.108.pdf.

[19] Oracle Grid Engine, http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html.

[20] Platform Load Sharing Facility, http://www.platform.com/workload-management/high-performance-computing.

[21] Torque Resource Manager, http://www.clusterresources.com/pages/products/torque-resource-manager.php.

[22] PBS Professional, http://www.pbsworks.com/Product.aspx?id=1.

[23] Condor High-Throughput Computing System, http://www.cs.wisc.edu/condor/.

[24] Apple Xgrid, www.apple.com/server/macosx/technology/xgrid.html.

[25] Simple Linux Utility for Resource Management (SLURM), https://computing.llnl.gov/linux/slurm/.

[26] IBM Tivoli Workload Scheduler LoadLeveler, http://www-03.ibm.com/systems/software/loadleveler/.

[27] Maui Scheduler, http://www.clusterresources.com/products/maui/.

[28] Java Agent DEvelopment Framework, http://jade.tilab.com/ (2011). http://iopscience.iop.org/1749-4699/3/1/015002

[29] MAPPER Deliverable D4.1: Review of Applications Users, Software and e-Infrastructures, http://www.mapper-project.eu/documents/10155/23578/D4.1+-+Review+of+Applications%2C+Users%2C+Software+and+e-Infrastructures.pdf?version=1.0

[30] MAPPER Deliverable D7.1: First report on adaptation of applications.

[31] MAPPER Deliverable D8.1: Architecture and Interfaces, http://www.mapper-project.eu/documents/10155/23479/D8.1+-+Architecture+and+Interfaces.pdf

[32] MAPPER Deliverable D8.2: First Prototype with Demonstration, http://www.mapper-project.eu/documents/10155/23479/D8.2+-+First+Prototype+with+Demonstration

[33] Furlani, J.L., Modules: Providing a flexible user environment, Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V), 141–152,(1991).