D8.1 Description of the Architecture and Interfaces

Project acronym: *MAPPER*

Project full title: Multiscale Applications on European e-Infrastructures.

Grant agreement no.: 261507

Due-Date:	M6
Delivery:	M6
Lead Partner:	Cyfronet
Dissemination Level:	Public
Status:	Final
Approved:	Quality Board, Project Steering Group
Version:	2.13

DOCUMENT INFO

Data and version number	Author	Comments
16.02.2011 v 0.1	Katarzyna Rycerz	Plan of the document
21.02.2011 v 0.2	Tomasz Gubala	Initial content for sections on result browsing, SSM repository, module registry and result management
22.02.2011 v 0.3	Tomasz Gubala	Started the bibliography section
22.02.2011 v 0.4	Katarzyna Rycerz	draft of general architecture section, update on modules metadata stored in the registry
22.02.2011 v 0.5	Joris Borgdorff	Initial information about xMML tool
23.02.2011 v 0.6	Tomasz Piontek	Transformaing xMML to QCG JobProfile
23.02.2011 v 0.7	Katarzyna Rycerz	update on general architecture section
24.02.2011 v 0.8	Tomasz Piontek	QCG Client
24.02.2011 v 0.9	Mariusz Mormonski	update on infrastructure metadata stored in the modules registry
28.02.2011 v 1.0	Stefan Zasada	AHE Client
28.02.2011 v 1.1	Eryk Ciepiela	Introduction to GridSpace Experiment Workbench
07.03.2011 v 1.2	Katarzyna Rycerz	Draft of requirements section
08.03.2011 v 1.3	Alexandru Mizeranschi	SBML Toolbox - Copasi
9.03.2011 v 1.4	Daniel Harezlak	Multiscale Application Designer - chapter draft
09.03.2011 v1.5	Grzegorz Dyk	Provenance - chapter draft
10.03.2011 v1.6	Katarzyna Rycerz	Improvements on requirements section
10.03.2011 v1.7	Joris Borgdorff	Added an MML section
14.03.2011 v1.8	Tomasz Gubala	Small changes in XMML repository section
14.03.2011 v1.9	Katarzyna Rycerz	Multiscale Application Skeleton Section
14.03.2011 v2.0	Eryk Ciepiela	Experiment Workbench section improvements
15.03.2011 v2.1	Eryk Ciepiela	Experiment Workbench and Experiment Execution Engine sections improvements
15.03.2011 v2.2	Daniel Harężlak	MAD section improvements
15.03.2011 v 2.3	Katarzyna Rycerz	Summary and Conclusions

		sections, small MML section improvements
15.03.2011 v 2.4	Katarzyna Rycerz	Formatting
16.03.2011 v 2.5	Katarzyna Rycerz, Eryk Ciepiela, Grzegorz Dyk	Minor changes, formatting
17.03.2011 v 2.6	Eryk Ciepiela	Execution engine section improvements
17.03.2011 v 2.7	Grzegorz Dyk	Provenance section improvements
17.03.2011 v 2.8	Marian Bubak, Katarzyna Rycerz	Corrections of the overall structure
17.03.2011 v2.9	Jan Meizner, Piotr Nowakowski,Marek Kasztelnik, Włodzimierz Funika	Proofread
18.03.2011 v2.10	Marian Bubak, Włodzimierz Funika, Katarzyna Rycerz	Minor corrections
30.03.2011 v 2.11	Katarzyna Rycerz	Minor changes of document structure
05.04.2011 v 2.12	Bastien Chopard, Katarzyna Rycerz	General review of the document
06.04.2011 v 2.13	Werner Dubitzky, Katarzyna Rycerz	General review of the document

TABLE OF CONTENTS

1 Executive summary 7
2 Contributors 8
3 Glossary of terms 9
4 Characteristic and requirements of multiscale applications 13
4.1 Characteristic of multiscale applications 13
4.2 Multiscale Modeling Language 15
4.2.1 Submodel execution loop 16
4.2.2 Graphical representation (gMML) 16
4.2.3 Textual representation (xMML) 18
5 Architecture of multiscale programming and execution tools layer 19
5.1 Introduction 19
5.2 Architecture of multiscale programming and execution tools 19
5.3 Current usage scenarios of programming and execution tools 22
5.3.1 Tightly coupled applications 22
5.3.2 Loosely coupled applications 24
6 Multiscale application skeleton 27
6.1 Motivation 27
6.2 Background 27
6.3 Functionality 28
6.4 Example of multiscale application skeleton 28
7 Conclusions 30
8 Annex 1. Detailed design 31
8.1 User Interfaces and visual tools 31
8.1.1 Multiscale Application Designer (MAD) 31
8.1.2 GridSpace Experiment Workbench 36
8.2 Programming Tools 40
8.2.1 XMML Repository 40
8.2.2 Registry for Application Modules 44
8.2.3 SBML toolbox 47
8.3 Execution tools 49
8.3.1 GridSpace Experiment Execution Engine 49
8.3.2 Result Management 53
8.4 Provenance 54
8.4.1 Use cases 54

8.4.2 Design....56
8.4.3 Data definition....57
8.4.4 Provenance data acquisition....58
8.4.5 Data sharing and querying....58
9 Annex 2. MAPPER Application Technical Inquiry....60
9.1 Goal....60
9.2 Instructions....60
9.3 Questions....60
10 Annex 3. The AHE client usage....63
11 Annex 4: QCG client usage....65
12 References....69

LIST OF TABLES AND FIGURES

Tab. 1 Summary of requirements and their proposed support in the WP8 architecture....21
Tab. 2 Summary of multiscale applications requirements and proposed solutions....30

Fig. 1. Example of submodel execution loop with various operators....16
Fig. 2: An example of the In-stent restenosis 3D model....17
Fig. 3. Operators in the graphical representation of Multiscale Modelling Language....17
Fig. 4. Architecture of the multiscale programming and execution tools....20
Fig. 5. Multiscale tools for tightly coupled execution scenario....23
Fig. 6. Multiscale tools for loosely coupled execution scenario....25
Fig. 7. The Structure of In-stent Restenosis Application (2D version)....29
Fig. 8. Application skeleton example based on In-stent Restenosis 2D structure....29
Fig. 9. Multiscale Application Designer use case diagram....32
Fig. 10. Design of Multiscale Application Designer....33
Fig. 11. Multiscale Modeling Language, its representations (xMML, gMML) and relation to GridSpace Experiment....34
Fig. 12. xMML to QocCosGrid Job Profile conversion - use case....36
Fig. 13. xMML to QosCosGrid Job Profile Translation Module - components and design....36
Fig. 14. Generating GridSpaceExperiment from gMML diagram....38
Fig. 15. Sample of remote directory contents after successful login to an execution machine....39
Fig. 16. Multiscale application designers will be able to store and manage XMML descriptions inside the repository....41

Fig. 17. Proper handling of XMML description files requires appropriate file storage as well as a means of capturing and persisting the accompanying metadata (author, version, date etc.). 42
Fig. 18. The Persistence Abstraction Layer provides generic file and metadata storage features as well as customization via the domain model. 43
Fig. 19. The registry of application modules serves both design and execution of multiscale applications. 44
Fig. 20. Preliminary scenario of aggregating various information and monitoring services used in different e-infrastructures. 46
Fig. 21. The module registry re-applies the persistence abstraction layer with a different domain model to store simulation model metadata. 47
Fig. 22. Functionality of GridSpace Execution Engine. 49
Fig. 23. The functionality of the MAPPER result management component from the point of view of a person running an instance of a MAPPER application. 53
Fig. 24. The result management in MAPPER is delivered both through the direct access to user's files on a target machine and through a dedicated result location registry (for other means of result persistence, like dedicated storage facilities). 54
Fig. 26. The design of Provenance system. 56
Fig. 23. Use cases for Provenance collector. 55

1 Executive summary

The aim of this deliverable is to provide specification of the architecture and interfaces of multiscale programming and execution tools layer. This deliverable describes design of an environment for composing multiscale simulations from single scale models encapsulated as scientific software components and distributed in the various European e-Infrastructures supporting the two main paradigms of multiscale computing: loosely coupled and tightly coupled. According to the MAPPER description of work the aim of the proposed environment is to:

- support composition of multiscale applications by using visual and programming tools to build multi-disciplinary and multi-scale “in silico” experiments,
- support execution of such experiments and achieve their reusability,
- integrate solutions designed for multiscale simulations’ development (such as MUSCLE communication library) with possibilities given by environments for application composition and European e-Infrastructures,
- allow interaction between software components from different e-Infrastructures in a hybrid way.

The document is organized as follows: Section 3 contains glossary of terms used in this document, In Section 4 we have described application characteristic and requirements. This Section also covers the idea of standard language for describing multiscale applications structure - Multiscale Modelling Language. The general architecture of the tools and a typical use case is described in Section 5. The tools are divided into the following groups: user interfaces and visual tools, programming, execution and provenance. In Section 6 we describe the motivation and idea of multiscale application skeleton framework. We summarize in Section 7. Detailed design of the tools can be found in Annex 1 (Section 8).

2 Contributors

Below we list the institutions and names of the contributors. Their exact role in this deliverable is depicted in the document info table at the beginning of the document.

Cyfronet: M. Bubak, E. Ciepiela , G. Dyk, T. Gubała, D. Harężlak, J.Meizner, P.Nowakowski, K. Rycerz
PSNC: M. Mamoński ,T. Piontek
UvA: Joris Borgdorff
UNIGE: Bastien Chopard, Jean Luc-Falcone
UU: Alexandru Mizeranschi
UCL: Stefan Zasada

3 Glossary of terms

The terminology used in this document is listed below. It should be noted that the terminology will evolve during the project.

Application Hosting Environment (AHE): a framework supporting running applications on Grid infrastructures hosting Globus, UNICORE or GridSAM middleware.

Conduit: see MUSCLE

Coupling: interaction between two single scale models, could be uni-directional or bi or more directional (for coupling implementation see also: modules and MUSCLE)

Coupling template: a specific coupling from the SEL-operator of one submodel to the operator of another.

Coupling topology: a graph representation of a multiscale model, having coupling templates as its edges and instances of submodels as its nodes.

CxA: Ruby-based file format that describes a MUSCLE application: (1) modules parameters (2) couplings between modules. See: MUSCLE

Experiment host: host where GridSpace experiment is executed

gMML: see MML

Grid Resource Management System (GRMS): part of QCG middleware responsible for managing resources.

GridSpace (GS): GridSpace Experiment Workbench and Execution Engine.

GridSpace Experiment Workbench (GS Experiment Workbench or **EW):** GridSpace frontend **-** the web portal facility intended to be the interface for the end-users to perform activities related to composition and running multiscale applications.

GridSpace Experiment Execution Engine (GS E3): backend of GridSpace Experiment Workbench (EW) that takes responsibility for coordination of a GS experiment run

GridSpace experiment: set of snippets in various script languages stored in XML file. This XML file can be stored in Repository.

Job Profile: see QCG Job Profile

Loosely coupled and **tightly coupled**: a collection of submodels instances is loosely coupled if there is no cycle between them in the coupling topology, and tightly coupled otherwise.

Mapper: type of scaleless module. See: module, MUSCLE. Note: this is not MAPPER project.

Metadata: data about data (e.g. link to actual file, but not file itself)

Module: an independent software module implementing certain functionality. For MAPPER purposes we distinguish two kinds of software modules:

- scaleful software modules implementing single scale models (e.g. MUSCLE kernels),
- scaleless software modules used to convert data from one scaleful module to another. The examples are: MUSCLE conduits (unidirectional) or MUSCLE kernels called mappers (bi- or more- directional). See also: MUSCLE.

Multiscale process: a natural process that acts on multiple scales at once. For example, a reaction-diffusion system has diffusion taking place on a temporal and spatial meso-scale and reactions taking place on a temporal and spatial micro-scale. The reaction-diffusion process is thus described as a multiscale process.

Multiscale model: the model of a multiscale process.

Multiscale Modeling Language (MML): the high level concept of the language that describes single scale submodels and their complicated connections (the coupling topology of a multiscale model). It is a concept for modelers and has several representations. The one described in this document are xMML and gMML:

- **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.

Multiscale Coupling Library and Environment (MUSCLE): a communication library that can be used to connect modules implementing single scale models (called kernels) into a multiscale simulation. Kernels can be joined by means of unidirectional converters (called conduits). Some kernels can also be implemented as scaleless bi- or more- directional modules (called mappers). The structure of the MUSCLE application is described in CxA file.

Number of submodel instances: the number of instances a submodel may have within the multiscale model

Services: making the software fit for use.

Single scale process or **subprocess**: a natural process that acts only on a single scale. To be more precise, its scale ranges from the finest sizes it considers up to the total size of the process. In the context of a multiscale process, a single scale process can be called a subprocess.

Single scale model or **submodel**: a model of a single scale process. In the context of a multiscale model, a submodel.

Scale Separation Map (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 a model to their judgment in a way that serve the visual goal. SSM is not meant for computational (execution) purposes.

Snippet: a piece of code in a script language.

Submodel Execution Loop (SEL): pseudocode of a single scale model, defining the operators used and the order in which they are used.

Submodel instance: an instance of a submodel within the coupling topology. Multiple instances could be created of the same submodel.

Synchronization points: points during execution that one submodel instance will need to synchronize with another (including itself), by requiring input.

System Biology Markup Language (SBML): XML-based language for representing models. It's oriented towards describing systems where biological entities are involved in, and modified by, processes that occur over time.

QosCosGrid (QCG): a resource and task management system aiming to provide supercomputer-like performance and structure to cross-cluster large-scale computations that need guaranteed level of Quality of Service (QoS).

QCG JobProfile: XML-based language describing how to execute an application using QCG middleware.

Provenance: metadata about experiment creation, usage and results.

Repository: place where multiscale applications' description files are stored and managed (e.g. xMML files)

Registry: place where information (metadata) about some entities (in our case simulation modules) are registered (but modules themselves are not stored!)**.**

Task graph: an acyclic directed graph representation of the submodel instances and their synchronization points as they unfold over time. It may include each of the operators of the SEL as nodes.

User Interface machine (UI): machine accessible directly (via ssh) by a user from which he can access other (Grid, PBS) resources.

xMML: see MML.

4 Characteristic and requirements of multiscale applications

4.1 Characteristic of multiscale applications

Multiscale applications implement models of multiscale processes [HOEKSTRA10, ENGQUIST]. We focus on such multiscale applications that can be described as a set of connected single scale modules i.e. modules that implement models of single scale processes. Moreover, one application run can consists of many instances of the same single scale module. In modeling, a multiscale application can be represented in a form of *coupling topology* - a graph having instances of single scale models as its nodes.

Usually building multiscale application is not trivial and requires a lot of effort from the application designer. There are many solutions to combine single scale codes together, e.g. the Model Coupling Toolkit [MCT] – a toolkit supporting solving parallel coupled models, HLA components approach [RYCERZ10] that joins services provided by HLA standard [HLA] with component technology, Python based approached such as AMUSE [MUSE], or MUSCLE library [MUSCLE] that wraps single scale models as Java agents and uses an agent framework to execute the coupled simulations.

Although these environments are capable to support parallel or distributed multiscale simulations, they are usually used by individual users on local clusters to solve specific multiscale problems. Due to the fact that they are not deployed on any of existing e infrastructures, they are not widely used by a multiscale simulations community and do not fully support development of general solutions and standards in that field. The goal of WP8 is to build tools that support programming and execution of such applications using technological possibilities given by available e-infrastructures. In this document we focus on the requirements for such tools.

Below we present characteristic of multiscale applications based on the information gathered by means of inquiry attached in Annex (Section 9) as well as the review described in D 4.1. The detailed information about applications can be found in D 4.1. The exact filled inquiries can be found on the MAPPER project wiki (*http://www.mapper-project.eu/web/guest/wiki*) and contain questions about application structure, implementation details and their developer's expectations.

A typical multiscale application consists of:

- software modules simulating certain phenomena in certain time or space scale (scaleful); usually this modules are computationally intensive, could require HPC resources, often (but not always) are implemented as parallel programs,
- software modules that convert data from one scaleful module to another; usually these modules do not have demanding computational requirements; however, to avoid additional communication, they often required to be executed "close" to the scaleful modules they are connecting; they can even be implemented in the same process as one of the scaleful modules.

The communication structure of the multiscale application varies significantly. So far, we have identified following schemas:

- *master-worker* paradigm e.g. macro scale module (master) triggers micro scale simulation of a part of its domain that requires more detailed attention; these types of applications are also supported by Heterogeneous Multiscale Methods [ENGQUIST]; we can find examples outside of MAPPER - e.g. a suspension flow application described in [LORENZ]; this type usually requires dynamic trigger of modules execution, usually number of modules instances is dynamic,
- *peer to peer* type of computation where all modules are executed concurrently and exchange data in usually asynchronous fashion; example is part of MAPPER In-stent Restenosis application [CAIAZZO], Canals [THANG] and Fusion [COSTER] applications; during the course of execution, applications often pass many synchronization points (the number can be static or dynamic); therefore, this type often requires mechanism of efficient communication,
- *pipe* - modules execute one after another; example is MAPPER Nano Polymer application [SUTER],
- *hybrid* - the combination of two or more possibilities mentioned above; example is Instent Re-stenosis Application (ISR): Initial Condition module is connected to the rest of the simulation in "pipe", and then the rest of the simulation consists of modules that run concurrently.

Regarding type of modules execution one can distinguish between:

- *stateless* modules - after they finish, they return result (in a form of returned state parameter or snapshot file) and do not preserve any data from their computation; example is LAMMPS module from Nano Polymer simulation,

- *stateful* modules - after (often partial) computation they remember the state of it; often remembering the state from previous calculation can fasten next one; examples are modules of ISR or Fusion applications.

Regarding characteristic mentioned above, multiscale application can be also classified as *loosely* or *tightly* coupled. In loosely coupled simulation there is no loop in coupling topology (this could be a pipe or direct acyclic graph scheme) and the modules are stateless. In tightly coupled simulation, there is a loop in coupling topology and the modules are stateful.

Besides of the requirements coming from application structure, there are additional ones:

- some of the applications need to switch between different versions of the modules with the same functionality (e.g. Fusion application),
- some of the modules can be interactive (i.e. can require user input during application execution - e.g. Nano polymer simulation),
- modules of the same application often need access to different resources from HPC to Grid type.

4.2 Multiscale Modeling Language

The requirement of making the modeling and design of multiscale applications easier was the main motivation for an elaboration of a language that uniformly describes multiscale models and their computational implementation on abstract level. Within the MAPPER project we plan to extend the idea of Multiscale Modeling Language (MML) [FALCONE, HOEKSTRA10] which 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 [HOEKSTRA10,HOEKSTRA07] which describe multiscale coupled cellular automata. Notably, from this formalism the submodel execution loop (SEL) is re-used.

4.2.1 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.
Fig. 1 shows the example pseudo code of the SEL.

```
f := finit
t := 0
while not EC(f, t):
    Oi(f, t)
    f := S(f, t)
    f := B(f, t)
    t += theta(f)
end
Of(f, t)
```

Fig. 1. Example of submodel execution loop with various operators.

The operators shown in
Fig. 1 are: finit, Oi, B, S, and Of for initialization, intermediate observation, boundary condition calculation, solving step, and final observation respectively. Operators finit, B, and S are allowed to receive data and Oi and Of to send data. EC is the end condition for the submodel and theta is the possibly variable time step. Coupling templates are defined as couplings between the operator of one submodel to the operator of another.

4.2.2 Graphical representation (gMML)

In the graphical representation MML, UML-like icons are used to show different couplings.
Fig. 2 shows an example of the In-stent restenosis 3D model.

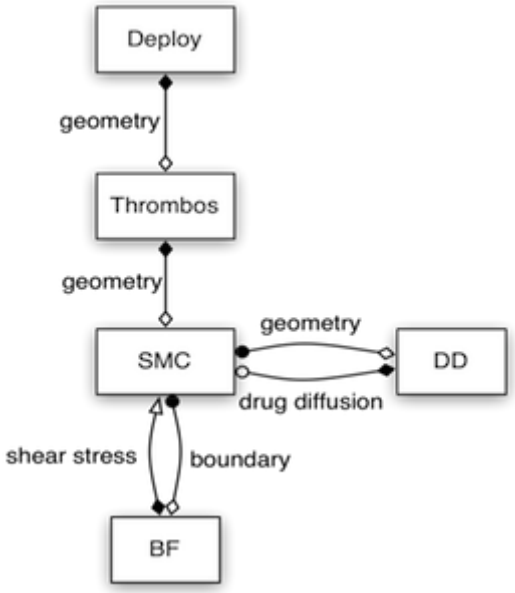

Fig. 2: An example of the In-stent restenosis 3D model.

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

Fig. 3 for which operators correspond to which tail or head icon.

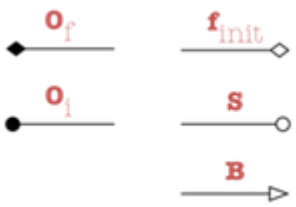

Fig. 3. Operators in the graphical representation of Multiscale Modelling Language.

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

4.2.3 Textual representation (xMML)

xMML, the textual representation of MML, is based on XML format. xMML is described on the MAPPER wiki page *(http://www.mapper-project.eu/web/guest/wiki)* in more detail. XML was chosen as a well-known, human-writable, machine-readable standard that promotes interoperability.

In this version, xMML contains a model version and description. Each of the datatypes, converters and submodels used in that model are defined. 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 necessary to run a multiscale model.

5 Architecture of multiscale programming and execution tools layer

5.1 Introduction

This section discusses the overall multiscale programming and execution tools architecture and explaines how the visual, programming and execution tools interact with each other when supporting the user with creation and execution of multiscale applications. The typical use case scenario is also included.

The tools are developed in WP8 that is a part of Joint Research Activities in the MAPPER project. However, the actions taken in this workpackage have to be synchronised with the service activities WPs (WP4, WP5 and WP6). In particular, some of the tools developed in WP8 are extension to the already existing tools beeing adapted in WP4, integrated in WP5 and operated by WP6. Therefore the rest of this Section is divided into two parts. Section 5.2 presents a planned architecture of the tools that it is going to be achieved in the end of the project. It is quite obvious that a detailed design diagram such as the one presented in Fig. 4 will evolve throughout the lifetime of the project and should be considered as a plan. Section 5.3 contains a snapshot of the current state of development of the tools that are being already adapted and integrated in Services WPs and are to be extended in this WP.

5.2 Architecture of multiscale programming and execution tools

The tools can be divided into following groups: *user interfaces and visual tools*, *programming, execution* and *provenance:*

- User interfaces and visual tools are developed in task 8.1. This group includes a tool for creating MML in a visual form as well as the GridSpace Experiment Workbench with the File Browser.

- The programming tools (developed in task 8.2) include the repository of XMML files for further reuse and the registry describing information about existing application modules. This group also includes the toolbox supporting generation of System Biology Markup Language (SBML) model representation.

- The execution tools (developed in Task 8.3) include the high level execution engine that orchestrates overall application execution and connects to the interoperability

layer by the Application Hosting Environment (AHE) and the QosCosGrid (QCG) client. This group contains also the result management facility.

- The provenance tool developed in Task 8.4 supports tracking of application execution.

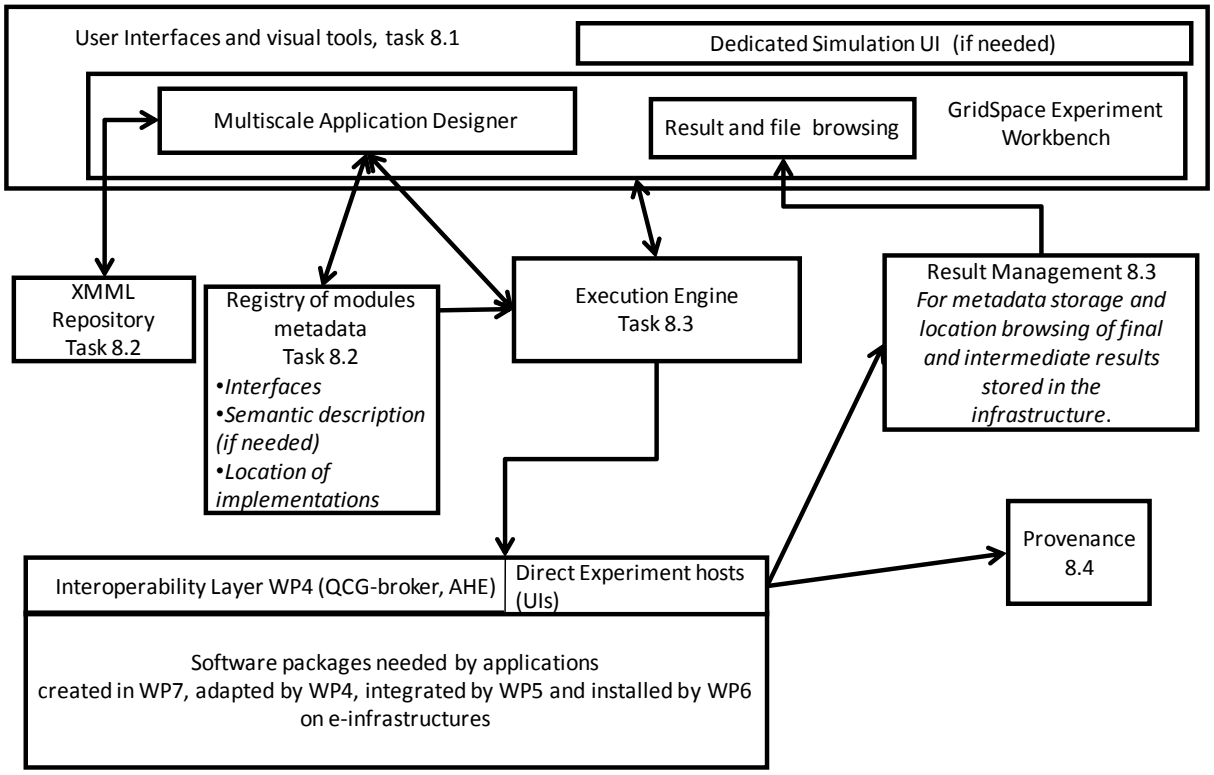

Fig. 4. Architecture of the multiscale programming and execution tools.

The typical scenario of the tools usage is:

1. A user creates MML (in a graphical or textual form) using Multiscale Application Designer tool,
2. The info about possible modules are taken from the Modules Registry,
3. The textual representation of MML (xMML) can be stored in the xMML Repository and reused from there,
4. The xMML is transformed into a set of execution instructions for execution engine (like GridSpace experiment or QCG Job Profile),
5. The simulation is executed on the e-infrastructure using appropriate interoperability layer (e.g. QCG, AHE or direct connection to User Interface machine available on the e-infrastructure). If the application is interactive (requires manual changes during the

execution), the flow goes back to the GridSpace Workbench that allows interactive experimentation,

6. Results are managed by Results Management and can be viewed in GridSpace result viewer,
7. The track of the execution is stored by Provenance Services.

The detailed design of the mentioned components can be found in Section 8 of this document. Tab. 1 below matches requirements mentioned in Section 4 with the architecture components.

facilitate designing multiscale applications	common standard - Multiscale Modeling Language (MML), tool supporting MML (MAD)
facilitate application development	interactive creation of experiments in GS Experiment Workbench
efficient execution	usage of existing e-infrastructures via interoperability layers (QCG, AHE)
interactive execution	interactive experimentation in GS EW
models reusability	store modules descriptions in a registry
applications descriptions' reusability	store application descriptions in a repository
tracking application execution, measuring tools efficiency	provenance system

Tab. 1 Summary of requirements and their proposed support in the WP8 architecture.

As stated above, the work described here is being done in cooperation with Services WPs. In this document we show how the designed tools are going to extend or use the software tools described in detail in D 4.1:

- GridSpace – the current version of this software is a general purpose tool for scientific applications. In this document we propose the extensions suitable especially for multiscale simulations. The detailed list of the new GridSpace fuctionality can be found in Section 8.3.1.1. (for Experiment Workbench) and Section 8.3.1.1 (for Execution Engine),
- MUSCLE communication library – in this document we describe how to use it from designed tools,
- QCG – In this document we describe how to use it from designed tools,
- AHE – In this document we describe how to use it from designed tools

5.3 Current usage scenarios of programming and execution tools

In this Section we describe the current state of the architecture and connections between existing software tools in the context of the whole project. In Section 5.3.1 we describe the case of tightly coupled applications and In Section 5.3.2 the case of loosely coupled applications.

5.3.1 Tightly coupled applications

The MAPPER tightly coupled applications are supported by MUSCLE environment as it is designed with MML concept in mind. MUSCLE consists of:

- a communication library to connect tightly coupled simulation modules (MUSCLE kerneles). The library allows to concurrently run all modules of the simulation that communicate directly using message passing paradigm. MUSCLE API is specifically designed for complex automata simulation model [HOEKSTRA10] and allows a user to specify connection ports (called Exits and Entrances). The MUSCLE communication is based on actor-based concurrency model i.e. asynchronous sending, synchronous receiving. Exits and Entrances are connected using external CxA formal file (see below),
- external configuration mechanims for specifying connections between modules (CxA file) and their parameters.

The cooperation between the tools for tightly coupled (MUSCLE) application execution scenario is shown in the Fig. 5. To execute the MUSCLE application, the GridSpace Experiment Workbench connects to user interface (UI) machine that stores all information and software necessary to run the application. This includes CxA description (a Ruby script that configures the connections between single-scale modules and the parameters of the whole application - connections can be viewed by the MUSCLE viewer) and the actual application implementation files. Next, the GS connects to the resource management system that could be either local and accessible from UI (e.g. PBS queue system) or grid (provided by QCG interface) to start the MUSCLE application. Once the application is started, the modules communicate using MUSCLE library. The MUSCLE application can benefit from GridSpace by integrating in one environment all steps necessary to:

1. configure application using CxA format - currently the MUSCLE connection editor and viewer is integrated into GridSpace environment. In the future the more advance composition tool is planned to be build as a part of task 8.1,

2. run application on various type of e-infrastructures supported by GridSpace: currently PBS cluster and QCG e-infrastructure. The details of low-level application execution (e.g. lauching kernels on different machines) are hidden for the user, who controls application structure in CxA file.

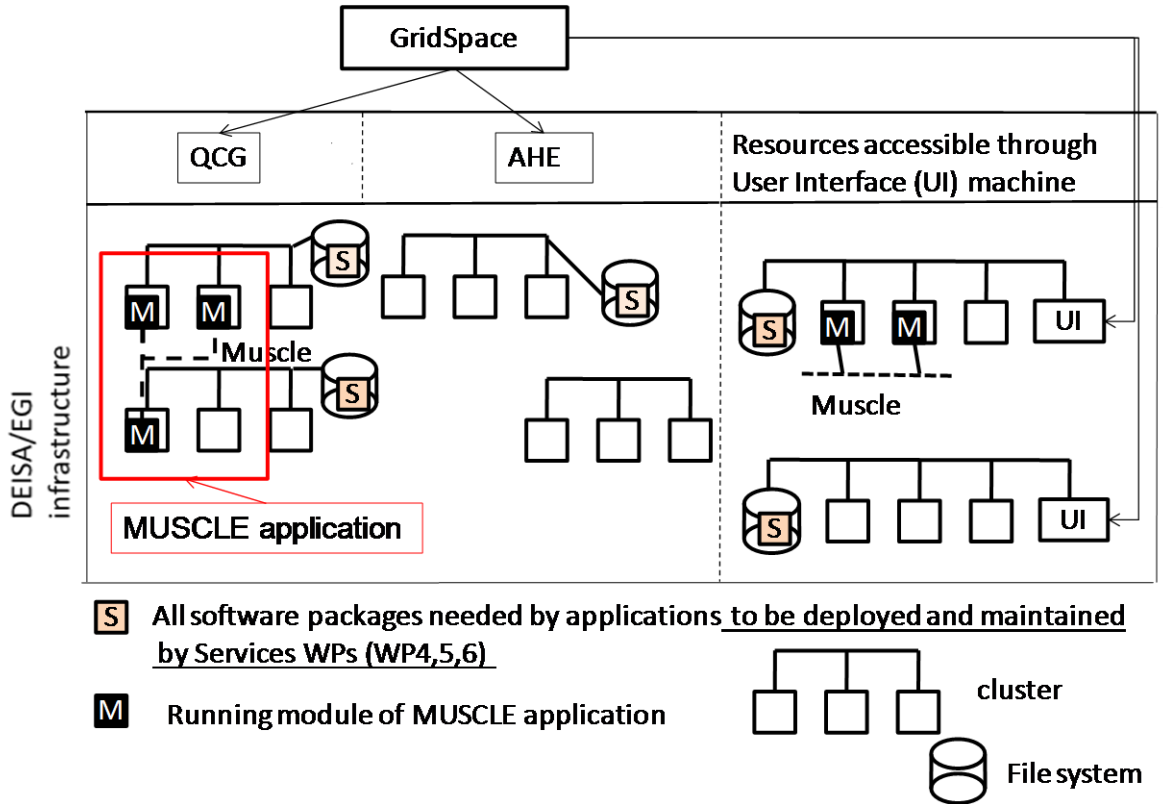

Fig. 5. Multiscale tools for tightly coupled execution scenario.

5.3.1.1 MUSCLE from GridSpace perspective

Below we present some aspect of MUSCLE from GridSpace as a programming and execution tool perspective.

- From a programming tool perspective, the idea of external configuration of modules connections is very promising as it naturally leads to simulation composability that MAPPER programming tools should support. Furthermore, as CxA file is a Ruby script, it is directly supported by GridSpace (designed to build so-called experiments from script snippets). Taking into account requirements of Multiscale Modelling Language, the information stored in the current CxA format need to be further extended. Additionally, running the application on any e-infrastructure should require a mechanism that allows a user to specify his preferences about mapping

MUSCLE kernels to actual resources - e.g. a user may want some kernels to be run on the same machines (see: ISR application).

- From an execution tool perspective, the actual execution of MUSCLE application requires (1) symultanous run of the kernels on various resources, (2) availability of direct communication between the kernels. On cluster architecture, this can be achieved by access to PBS mechanisms (supported by GridSpace). On cross-cluster architecture this can be achieved using various other mechanisms used in MAPPER project proposed by QCG or HARC.

Drawbacks of MUSCLE from developers' perspective:

- MUSCLE environment consists of the library API and visualisation tool. The design of the environment is not modular - it does not allow to easily separate the tool from the library, therefore adaptation of the visualisation tool to the GridSpace required deep level of understanding of MUSCLE implementation.
- installation of MUSCLE is not automatic, requires third-party libraries and installing additional ruby gem, which makes it a bit tricky without administrative priviliegies

Drawbacks of MUSCLE from application designers' perspective:

- MUSCLE does not allow to dynamically add kernels during runtime, which is useful for e.g. master-worker type of applications or applications that need to run modules in sequential order as a workflow (see e.g. "Initial Conditions" module of ISR application)
- The actual converters (conduits) are unidirectional. If it is required to have bi- or more- directional converters (like in ISR application), a normal MUSCLE kernels have to be used (such kernels are called mappers). This approach may be confusing as there is no mechanism to distinguish between kernels implementing scaleful models and scaleless mappers.

5.3.2 Loosely coupled applications

The possible cooperation between the tools for loosely coupled applications is shown in the Fig. 6. GridSpace Experiment Workbench connects directly to the machines where loosely coupled modules can be launched either directly or by local management system (e.g. PBS). A user can benefit from interactive exploratory programming feature of GridSpace that enables the user to make a decission how to execute module B after seeing output of

module A. As for tightly coupled applications, GS can benefit from QCG or AHE solutions (such as advanced reservation) to execute loosely coupled modules on the e-infrastructures.

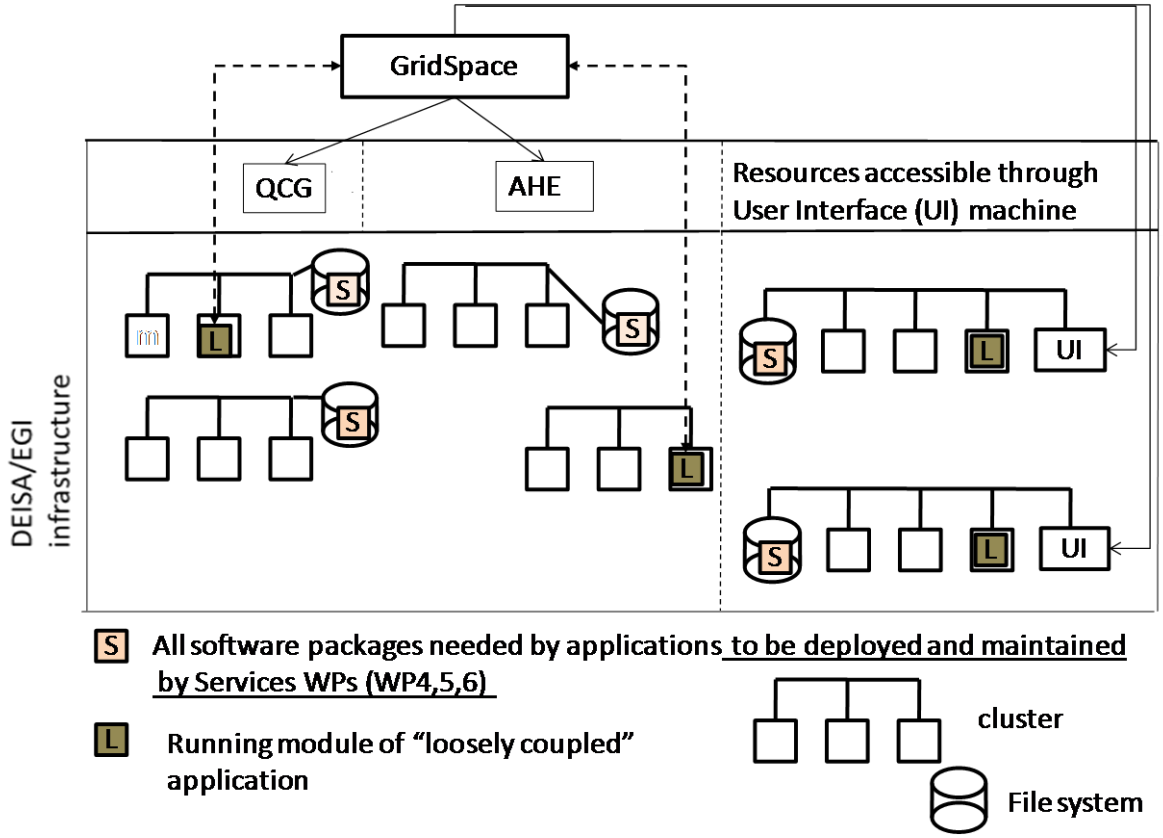

Fig. 6. Multiscale tools for loosely coupled execution scenario.

The well defined example of a loosely-coupled application in MAPPER is the Nano Polymer application that suits well the GridSpace operation model (see GridSpace description in D 4.1). First of all, the main model of the interaction with computational resources and data files is through the SSH protocol, also vastly used in GS. Since the Experiment Workbench (the main part of the GS tool) works entirely on the top of SSH it allows for easy switch from current terminal-based mode of work to the browser based solution. The user is able to log onto a computational resource (i.e. the user access node, precisely) perform the operations required to run the simulations using the provided set of interpreters.

Moreover, GS may provide 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 supported, 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). GS support running jobs through PBS natively through a

PBS gem - when compiled on the target machine it allows to submit computational runs directly from script languages like Ruby, Perl or Python.

The features which are not present in GS at the moment yet are foreseen to be useful for this application is the multi-machine login capability and the external SCP file transfer. The first will allow to be simultaneously present on two different machines (required to perform the 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 transferring files between these machines in a simple drag-and-drop manner. For the time being the users may still use two different browser tabs to access different machines and the file copy mechanism may be easily written in a simple shell script.

6 Multiscale application skeleton

6.1 Motivation

Application analysis conducted during preparation of this deliverable have shown that MAPPER applications vary significantly and it is difficult to identify a common pattern of requirements, essential for tools design. In Section 3 we described the characteristics of multiscale applications (types of modules, communication structures, etc.), as can be seen, quite a wide spectrum of types is covered.

Moreover, real applications are quite difficult to test as their installation is time consuming - it often requires third party libraries and special compilers (e.g. Fortran). The execution time is also quite long (e.g. for In-stent Restenosis it was ca. three days). Therefore, we propose to develop a Multiscale Application SKeleton framework (MASK). The framework will enable to build easy to test prototypes of multiscale applications. In particular, groups of users will benefit from MASK :

- application designers will be able to perform early tests of a chosen decoupling technique modeled using XMML,
- tool developers will be able to identify common patterns of multiscale simulation requirements,
- tool developers will be able to early test the programming and execution tools,
- services managers will be able to test the infrastructure.

6.2 Background

In general, algorithmic skeletons (called also Parallelism Patterns) are quite an old idea [COLE]. A recent survey of algorithmic skeleton frameworks can be found in [GONZALES]. The most common patterns include master-worker, pipe, divide and conquer, map-reduce. These patterns are general and can be applied to multiscale simulations. In the MAPPER project we aim to investigate how to extend this idea to specifically support multiscale simulations. When designing the framework we will have to define:

- a basic skeleton set based on the types of modules and communication patterns described in Section 4,
- the skeleton's capability of joining and nesting,
- the interface with which programmers code their skeleton applications either based on XMML or scripting languages,
- the language in which the skeleton applications are compiled and run,
- a communication library (e.g. MUSCLE),

- the capability to access and manipulate input/output files.

6.3 Functionality

The aim of the MASK framework is to support the creation of “empty” multiscale applications possessing the same structure as a real one: the same number and type of modules and the same communication structures: master-worker, p2p, pipe or hybrid (the types of modules and communication structures are described in Section 3 of the current Document). Also MASK will support parametrizing such an “empty” application with:

- execution time,
- amount of exchanging data,
- execution mode – batch, interactive,
- topology (e.g. a converter should be close to its source or sink modules).

MASK will allow the user to fill the skeleton in a customized way: by adding a real code or well known computational kernels (e.g. from benchmark suites) for comparison.

6.4 Example of multiscale application skeleton

To illustrate the idea of the application skeleton, we use an example the Instent Restenosis Application (ISR) 2-D version. From the execution tools’ perspective ISR fits a tightly coupled paradigm as its modules run concurrenly and communicate directly during runtime (a p2p communication structure). The current implementation of ISR uses MUSCLE as a communication library.

As shown in Fig. 7 the application consists of three modules of different time scale: simulation of blood flow, simulation of muscle cells, and drug diffusion. The application includes also scale-less transformation modules connecting ones which feature a scale (scaleful). The scaleful modules and two-way transformation modules are implemented as MUSCLE kernels. One-way converters are implemented as MUSCLE conduits.

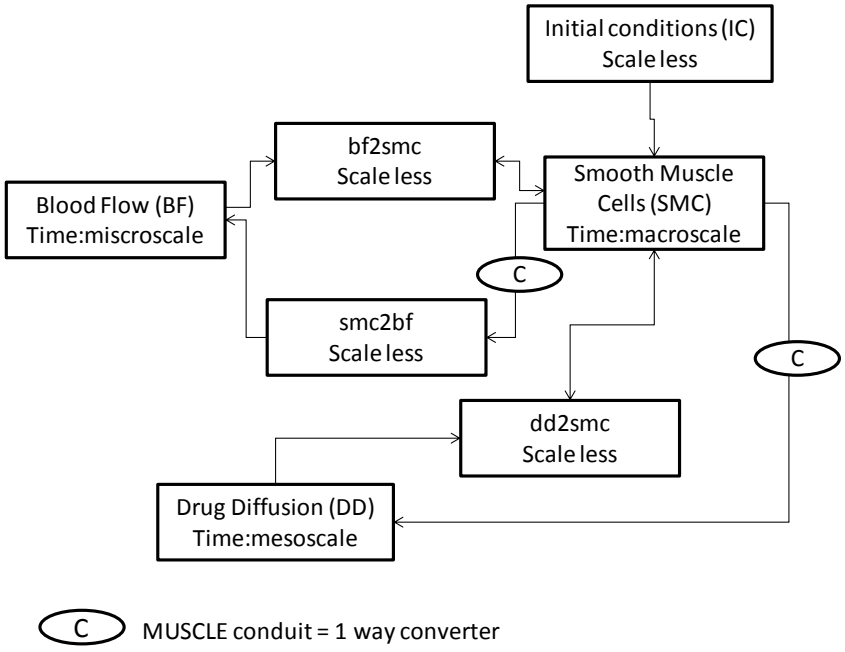

Fig. 7. The Structure of In-stent Restenosis Application (2D version).

Fig. 8 shows an example skeleton (structure of application prototype) based on ISR 2D structure; one can distinguish between different types of modules:

- initial condition (needed only at the beginning of the simulation),
- stateful modules containing actual simulation models,
- state-less modules being converters (either: 2-way ones - implemented as MUSCLE kernels, or 1-way ones - implemented as MUSCLE conduits).

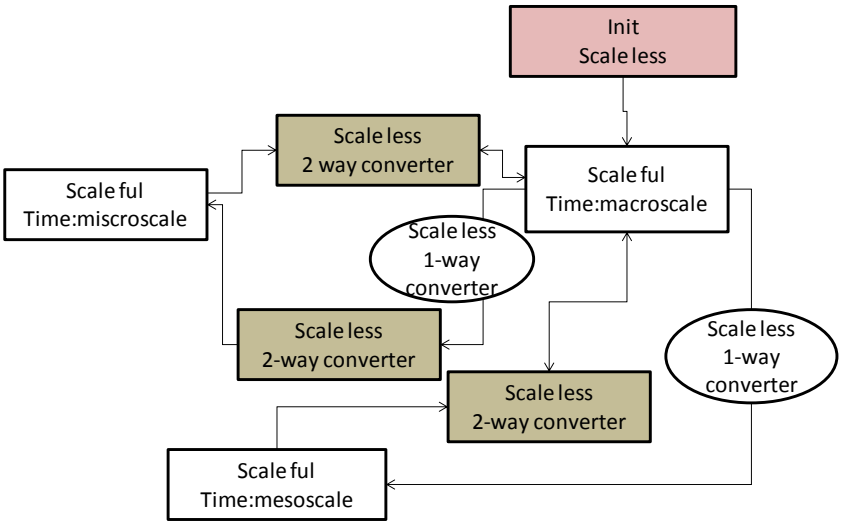

Fig. 8. Application skeleton example based on In-stent Restenosis 2D structure.

MASK will enable the user to create and execute such a skeleton without actually filling the computational loops of the models.

7 Conclusions

This deliverable describes architecture and design of programming and execution tools layer supporting the development and execution of multiscale applications. The key building blocks of the architecture have been identified and their internal structure and functionality described. Their description is based on the application, software and infrastructure review described in D 4.1. In Tab. 2 we summarize the application requirements and indicate solutions proposed in the Document.

Requirement	**Solution**	**Proposed tool**	**Section of D 8.1**
facilitate the design of multiscale applications	to develop a common standard - Multiscale Modelling Language (MML)	Multiscale Application Designer tool for supporting creating applications in MML	4.2 and 8.1.1
facilitate application development	exploratory programming	GridSpace Experiment Workbench	8.1.2
efficient execution	usage of the existing e-infrastructures, resource brokering, allocation	Execution Engine and clients to interoperability layer tools (QCG, AHE)	8.3.1
interactive execution	exploratory programming	GridSpace Experiment Workbench	8.1.2
models reusability	store module descriptions in a registry	Models' Registry	8.2.2
applications descriptions' reusability	store application descriptions in a repository	Repository of textual representation of MML	8.2.1
tracking application execution, measuring tools' efficiency	to build a provenance system	Provenance tool	8.4
easy testing and verifying of tools	multiscale application skeleton	Multiscale Application SKeleton framework	6

Tab. 2 Summary of multiscale applications requirements and proposed solutions.

The development of complex modern software is an iterative process. The presented design of the tools will be subject to inevitable changes as user requirements evolve and new circumstances emerge. However, the design presented in this document provides a solid basis for the implementation of the first prototype of the tools.

8 Annex 1. Detailed design

This Section describes components of the architecture presented in Section 5.2 in more detail. Following subsections contains use cases and design diagrams of the tools: Section 8.1 presents user interfaces and visual tools (Task 8.1), Section 8.2 described programming tools (Task 8.2), Section 8.3 – execution tools (Task 8.3), Provenance is described in Section 8.4 (Task 8.4).

8.1 User Interfaces and visual tools

8.1.1 Multiscale Application Designer (MAD)

8.1.1.1 Use cases

The Multiscale Application Designer (MAD) will be a MAPPER's tool allowing users to build multiscale applications in a graphical environment by using graphical multiscale modeling notation (gMML). As explained in Section 4.2, gMML is a graph of submodels (graph vertices) coupled by a set of connectors (graph edges). The MAD tool will assist users in composing gMML graphs by providing a list of available modules from which suitable ones will be placed on the tool's working space by using drag-and-drop techniques. The modules can then be connected by appropriate connectors. If enough information is provided a gMML graph can be visualized as a Scale Separation Map (SSM).

As depicted in the use case diagram below the Draw gMML use case includes three other use cases. These are straightforward and represent MAD's ability to store, load and update gMMLs. The format used to store the graphs is xMML, which is an XML implementation of the graphical Multiscale Modeling Language.

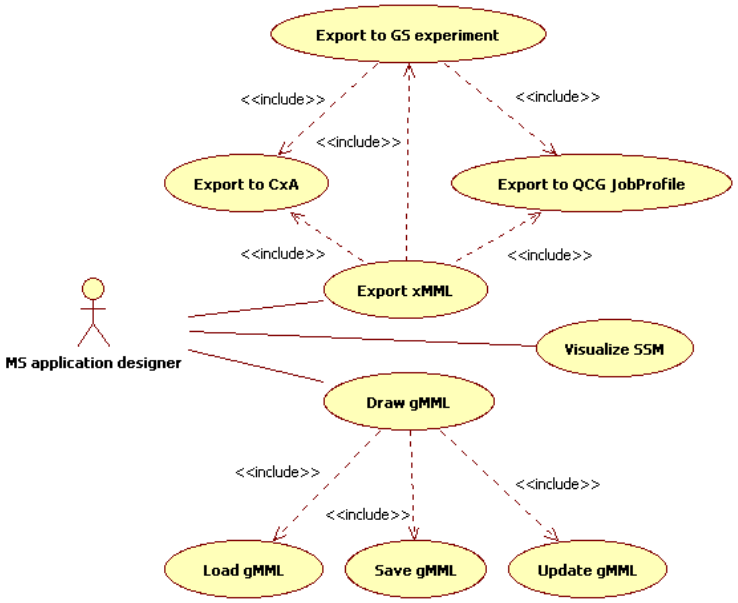

Fig. 9. Multiscale Application Designer use case diagram.

Because gMML is merely an abstraction of the actual application which, to run on the MAPPER infrastructure, needs concrete execution descriptions (namely CxA or QCG Job Profile scripts) the MAD tool provides suitable export facilities. The one worth mentioning here is the GS experiment exporter which includes two other exporters. This is dictated by the fact that GS can use both CxA and Job Profile descriptions for application execution.

The Visualize SSM use case produces a Scale Separation Map only in a read-only form. The final map is build by using submodels' attributes describing its space and time constraints. If a requirement of editable SSMs emerges it could be handled by MAD after specifying dynamic relations between these notations.

8.1.1.2 Design

Multiscale Application Designer will provide a graphical editor which by employing drag-and-drop techniques will enable users to construct gMML diagrams. Also, if required, each element of the graph will have an option to be manually edited by the user. As depicted in the picture below the editor will use external facilities such as XMML Repository, Module Registry and export utilities.

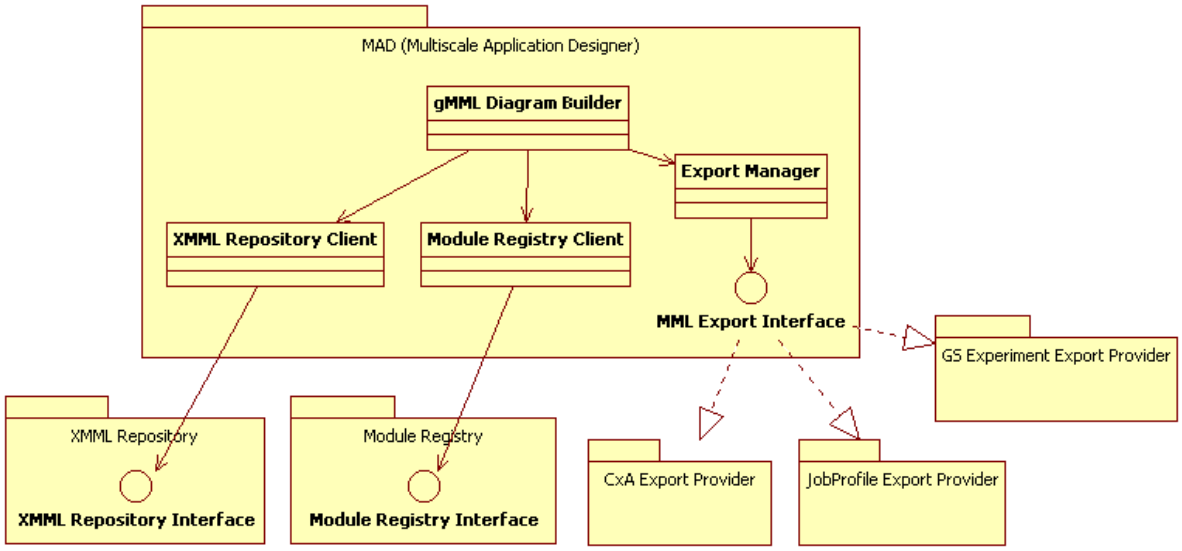

Fig. 10. Design of Multiscale Application Designer.

The XMML Repository component is used to store xMML documents together with all necessary metadata, which are used to draw gMML graphs. The API provided by the component will enable MAD to implement the CRUD (create-read-update-delete) set of operations for users building gMML diagrams.

The Module Registry component provides information about existing modules which can be part of the gMML diagrams. A set of such modules will be visualized by MAD for the user to compose gMMLs by coupling them with suitable connectors. The registry will store extra information about individual modules (provided during the process of registering a module) which will be necessary to create concrete application descriptions (e.g. script templates, infrastructure host details, etc.).

gMML diagrams will be stored by using xMML notation enriched by MAD's metadata holding information about graphical representation of individual modules. Due to the fact that xMML is an XML-based notation a separate namespace can be utilized to add these details.

MAD will also provide a utility for xMML export to more concrete application descriptions which can be used to execute them on MAPPER's infrastructures. To make this a plugin-like functionality an xMML Export Interface will be provided. Each export provider compliant with the interface will be exposed in the user interface and to other providers. Example export providers are to: Muscle CxA, QCG Job Profile and GridSpace Experiment.

8.1.1.3 Transforming xMML to GridSpace Experiment

Generally speaking, GridSpace Experiment Workbench role is to support creation and running of multiscale application expressed in Multiscale Modeling Language (MML). The language itself is abstract so it needs to have representations: XML-based (xMML), and graph-like (gMML) as shown in Fig. 11. xMML can represent complete MML (section A and B in Figure 2) while gMML can represent only a basic skeleton of multiscale application (section B). On the other hand gMML also models the graphical and layout information about the application model (section C) that reaches beyond MML language but improve visual modeling of applications. Aside visual modeling it is always indispensable to fill the high-level description of application with more details (section A) to make the application complete. Therefore, gMML editor needs to be backed with xMML editor where all this information has to provide. However, high-level MML description of application is still not enough to make it executable. Lacking parameters and code fragment need to be filled and stored in experiment snippets. That's why additional schema (section D) needs to be intertwined with xMML .

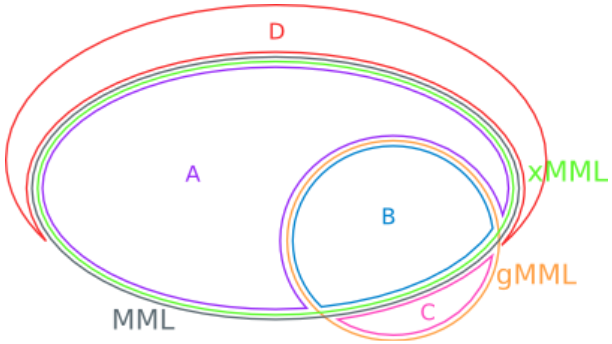

Fig. 11. Multiscale Modeling Language, its representations (xMML, gMML) and relation to GridSpace Experiment.

Regarding storage aspect of MML, it can be serialized as xMML document with or without graphical information extension of gMML (section C) and with or without experiment detail (section D). Using dedicated XML schemas for MML information and additional graphical information and experiment information, XML document can be a composite but still keeping those aspects explicitly decoupled. Thus, MAD would read both these aspects to render graphical representation of application, while GridSpace Experiment Execution Engine would parse MML and experiment aspect of the application.

8.1.1.4 Transforming xMML to CxA

The CxA file contains all configuration needed to run a model using MUSCLE. The xMML format already specifies this entire configuration except for parameters. Export provider will

be created, taking an xMML file and optionally a parameters file and it can generate the CxA file.

8.1.1.5 Transforming xMML to QCG Job Profile

The QosCosGrid middleware (QCG) is a resource and task management system aiming to provide supercomputer-like performance and structure to cross-cluster large-scale computations that need guaranteed level of Quality of Service (QoS). QCG offers end users efficient and secure access to dynamically discovered computational resources in grid environment with a requested QoS. From the user perspective the main goal of the QCG services is to manage the whole process of computational experiment in the way that satisfies Users (Job Owners) and their applications requirements as well as constraints and policies imposed by other stakeholders, i.e. resource owners and Grid or Virtual Organizations administrators. The QCG services were integrated with MUSCLE library to support multi-scale applications' use cases requiring co-allocation of heterogeneous resources synchronized with the use of advance reservation mechanism. The description of experiment to be submitted and controlled by QCG services must be expressed in the formal XML schema based language called Job Profile describing as well the application itself (executable, topology, arguments, input/output files and directories, etc.) as its requirements in terms of resources and execution time.

Use Cases

The Job Profile is the e-Infrastructure level description and its complexity can be difficult to understand for problem oriented scientists that expect the access to computation infrastructure to be as easy and intuitive as possible without necessity to use anything else then the xMML description of their model and application. The main motivation for the xMML to JobProfile Transformation Module is to provide simple way to do automatic transformation of the xMML description into the Job Profile, what will release the user from necessity to know both description languages and will allow him to focus only on the problem itself. User will have to provide the xMML description (template) of the application he wants to execute on MAPPER e-Infrastructure with additional run specific parameters and the module will transform this xMML document into the ready to execute Job Profile. The generated Job Profile can be in next step submitted directly to QCG services to be executed on project infrastructure or passed to any tool (for example GridSpace) to be validated or modified before submission.

Fig. 12. xMML to QocCosGrid Job Profile conversion - use case.

Design

Because both documents (xMML and Job Profile) are in XML format the key component of the module will be the XSLT transformation. Additional advantage of the XSLT technology is its independence from programming language. There are XSLT interpreters for mostly all of popular programming languages. The java based API will be designed and implemented for integration of xMML to JobProfile Transformation module with GridSpace EW. In case of necessity of use of the module functionality with other then java language the direct use of XSLT transformation with programming language specific XSLT interpreter is recommended.

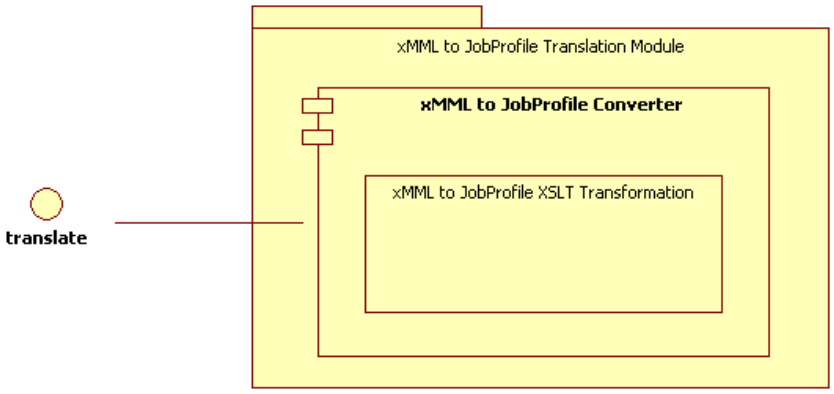

Fig. 13. xMML to QosCosGrid Job Profile Translation Module - components and design.

8.1.2 GridSpace Experiment Workbench

GridSpace Experiment Workbench (EW) [CIEPIELA] is the web portal facility intended to be the interface for the end-users to perform activities related to composition and running multiscale applications.

EW constitutes a single-sign-on entry-point for accessing whole MAPPER framework functionality where all users' tools and facilities are integrated in order to provide single well-equipped workbench. Therefore, EW is densely connected with other elements of the architecture presented in Section 5 of this document.

8.1.2.1 Typical use case

The typical use case scenario involves EW in the following way:

1. User logs in to the EW by providing login credentials to the selected subset of Experiment Hosts (User Interface head machines of the clusters) or credentials used by grid interoperability layer tools (e.g. login/password pair, GSI certificates). At least one credential must be provided on logging in, however additional ones can be provided any time when being logged in EW.
2. User creates mutliscale applications' gMML diagrams using MAPPER Application Designer MAD graphical tool. Behind the scenes EW connects to the xMML Repository (for storing and reading MMLs represented as XML documents for reuse) and to the registry of modules metadata (for getting info about modules available in the MAPPER environment).

The resulting diagram document is then transformed into GridSpace Experiment - a sequence of steps (called Snippets, containing complete executable code) to execute by GridSpace Execution Engine that backs the EW. The xMML document is first recognized as a task directed acyclic graph (DAG) of loosely coupled elements (steps) as depicted in Fig. 14. Single step can be a composite of modules which are tightly coupled with each other other with MML dependencies, or elementary - involving just one module. In the case of elementary module it is mapped directly to a snippet. A composite of tightly-coupled modules - in turn - is transformed into a snippet containing Job Profile document which is to be interpreted by a dedicated interpreter in GridSpace Execution engine (see Section 8.3.1). Job Profile specifies which modules, how coupled are with which parameters are to be executed.

3. The experiment run is coordinated by the GridSpace Execution Engine but is actually executed on the e-infrastructure using appropriate interoperability layer (e.g. QCG, AHE or direct connection to User Interface machines). If the application is interactive (requires manual changes, feedback or decision during the execution), the flow goes back to the EW.
4. The experiment execution status is traced and visualized on the application diagram in MAD.
5. Obtained experiment results are put in the Results Management service and can be viewed from within the EW using result viewers.

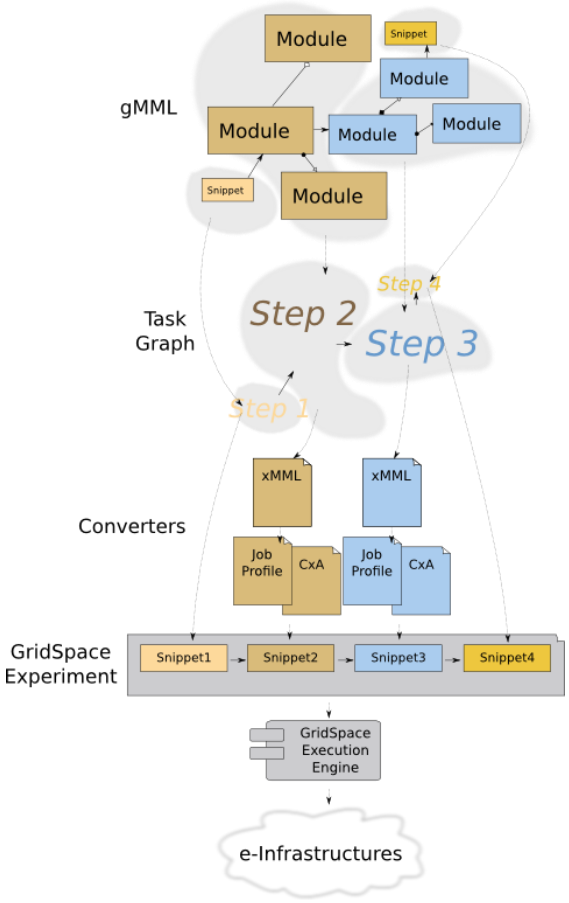

Fig. 14. Generating GridSpaceExperiment from gMML diagram.

8.1.2.2 New features imposed by MAPPER's requirements

Being constantly under improvement and development process, EW needs to meet new requirements that emerged during analysis of multiscale applications. Below, the major ones are enumerated:

- Multiscale applications in its entirety can take time of days. It means that EW has to support user logout during the course of application run. Even when user is offline EW needs to continuously trace the run, buffer all data related to it. User has to be given with a feature to re-log into the session started beforehand. The corresponding mechanism of notification would be beneficial to let users know about the progress in application run so he or she can re-log in and manage further execution.
- MAD tool needs to be seamlessly integrated with existing tools for creating experiments.

- EW may need to proxy ordinary User Interface machines and expose GSI-HTTPS server that would enable access to the data from other remote modules involved in application run.

8.1.2.3 Files and Results Browsing in GS Experiment Workbench

The Experiment Workbench enables you to manage the files which you need for the MAPPER applications to run (inputs) and to view the files which are produced by these applications (results). There are basically two mechanisms of storing such documents planned in MAPPER (see Section 8.3.2 for the detailed design of the result management component). Both mechanisms will be provided to the user of the Workbench in the form of a file (item) browser (see Fig. 15).

The current functionality of EW allows one to view the contents of one's home directory. This directory can be used to store experiments as well as any arbitrary files one may seem useful while working with EW.

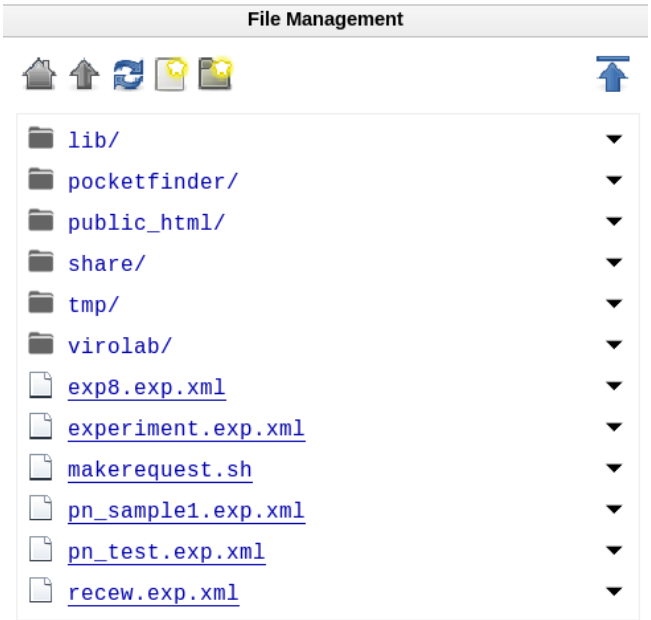

Fig. 15. Sample of remote directory contents after successful login to an execution machine.

Using the actions toolbar one is able to perform the following operations in the browser: return to the home directory, navigate to parent directory, refresh directory contents (useful after application completed), create a new file of directory and upload files from your computer. Moreover, a pull-down operations menu is available for each listed file. One can

access it by clicking the small arrow next to the selected filename. This menu contains the usual options one expects for file handling: delete, rename, view in web browser (which commands the immediate download of the file). Apart from these usual functions one may also:

- Open (or run) experiment: Opens the file as an experiment in the Workbench (only for experiments) and, if requested, immediately executes it.
- Use path in snippet: Pastes the file path in the currently active snippet window.
- Use contents in snippet: Pastes the file contents in the currently active snippet window.
- Open with: Brings up another menu which enables you to open the selected file using a specific application. The list of available applications corresponds to the file extension: GridSpace2 provides a selection of customized *openers* for popular file formats (such as Jmol and JQplot). Of special importance for MAPPER is the CxA files viewer, which allows to see the modules and connections of an application conforming to the CxA description standard.

8.2 Programming Tools

8.2.1 XMML Repository

8.2.1.1 Use Cases

The main motivation behind the repository of multiscale application descriptions is to provide users with the ability to save, load and update descriptions of multiscale applications designed within the MAPPER framework. While the exact set and layout of multiscale application descriptions is not yet fully decided upon, it is foreseen that each application will be described using one or more MML (Multiscale Modeling Language) files, possibly in the XML notation (called xMML).

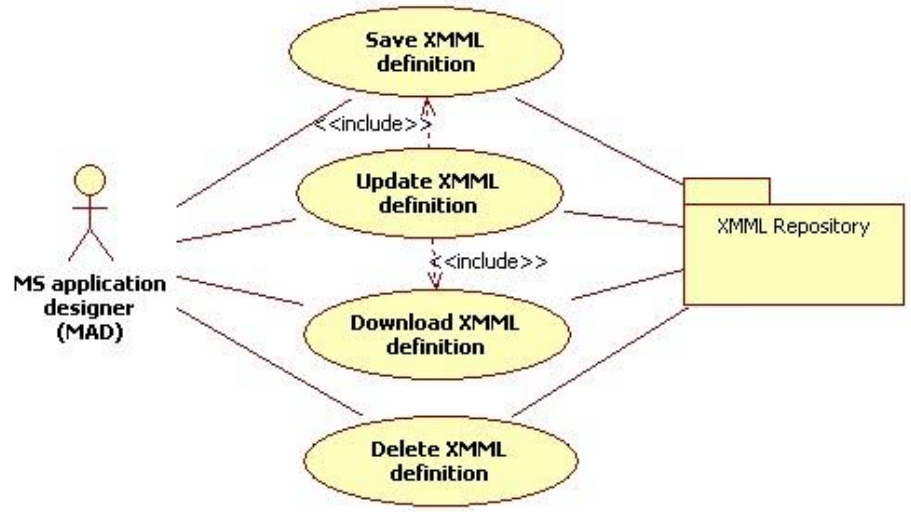

Fig. 16. Multiscale application designers will be able to store and manage XMML descriptions inside the repository.

Regardless of the exact notation and layout of the files, the main use cases of the repository are straightforward to define (see Fig. 16). Using the MAD designer tool which assists in the process of XMML definition, the authors are able to store the created description in the remote repository. This description can be downloaded in order to adjust, extend, or remove it (if no longer needed). The update process is actually a combination of the load-save pair of functions.

8.2.1.2 Design

In order to deliver the functionality of XMML file storage, the repository is split into two main building blocks (see Fig. 17). The former is responsible for storing the actual files in the repository. Since the files will be rather small and not very numerous, this part of the repository will use the underlying file system.

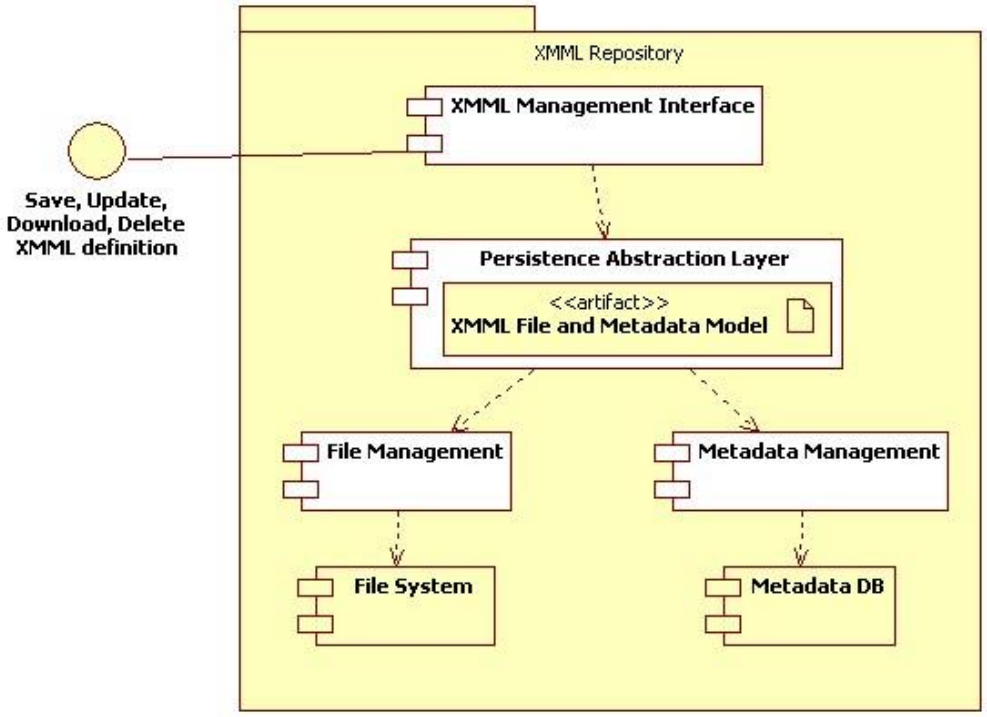

Fig. 17. Proper handling of XMML description files requires appropriate file storage as well as a means of capturing and persisting the accompanying metadata (author, version, date etc.).

However, it is also highly likely that MAPPER users will require further information to be stored along with the files. Information regarding authorship, ownership, versioning and tagging might be handy for the designers of multiscale applications. To this end, the repository will also be enhanced with metadata storage capability, based upon a dedicated database.

On top of both modules, the persistence abstraction layer will provide a unified interface to the storage mechanisms and a dedicated REST-like API for external tools (mainly the MAD creation tool in the MAPPER portal) enabling other components to access the repository. The abstraction layer and the storage mechanisms are described in detail in the following section.

8.2.1.3 Persistence Abstraction Layer

This abstraction layer was extracted from the design of the XMML repository for reuse. Owing to its genericity, the functions of this layer may be used by different elements of the MAPPER framework. This includes the XMML repository, the scale module registry and the result management tool (the latter two are described in other sections of this design document).

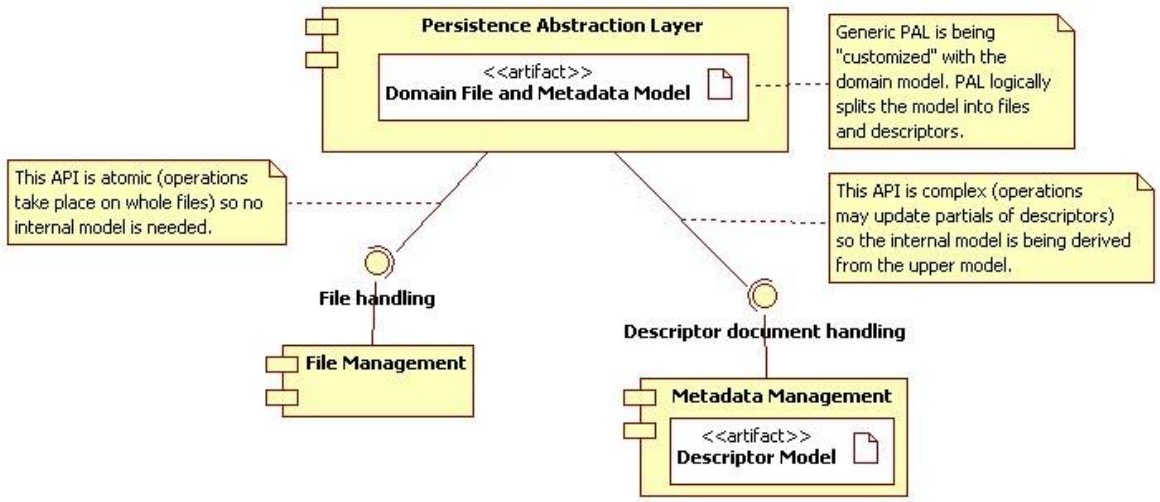

Fig. 18. The Persistence Abstraction Layer provides generic file and metadata storage features as well as customization via the domain model.

In order to be able to serve specific purposes in spite of its generic nature, the abstraction layer is parametrized with a domain model (see Fig. 18). This is based on the idea of *semantic integration* [SEMINT] and works as follows:

- for each specific usage area the abstraction layer is outfitted with a model which described this area;
- depending on the use cases, the model consists of metadata (descriptor documents with structure and key-value pairs), and, optionally, file model (not all usage scenarios require storing files);
- an underlying schemaless (non-relational) DB server is used to manage the persistence of descriptors;
- the abstraction layer ensures that the descriptor and file store remain in sync with each other.

By using this general mechanism we are able to deliver the various features of the MAPPER framework (XMML repository, result management, module registry). This makes the framework less fragile (fewer design elements) and simpler to develop and deploy. Moreover, should future design extensions become necessary (for instance to facilitate provenance gathering, experiment storage and similar) the semantic integration technology in place will help deliver them in shorter time.

8.2.2 Registry for Application Modules

8.2.2.1 Use Cases

The main function of the registry of application (scale) modules is to deliver the required information thereon to the designer of the application (at design time) and to the runtime MAPPER components (at runtime) (see

Fig. 19. The registry of application modules serves both design and execution of multiscale applications. We will discuss individual use cases in chronological order.

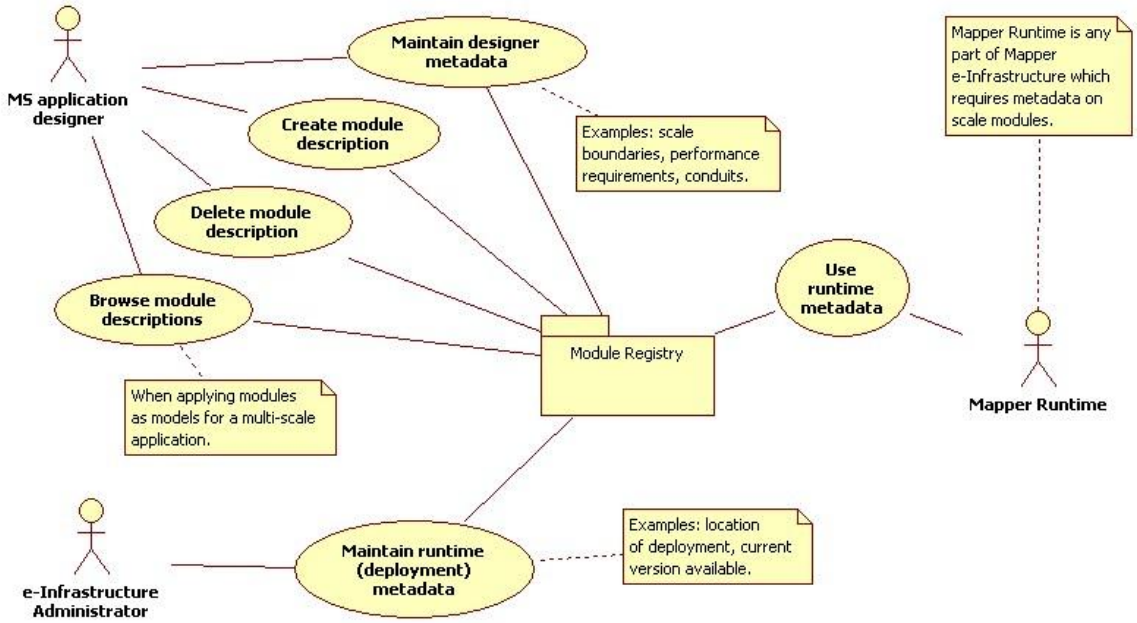

Fig. 19. The registry of application modules serves both design and execution of multiscale applications.

First of all, the designer needs to register the modules the application will use. The developer of a module might potentially also act as the designer of the multiscale application. Upon module registration, design-related information might be provided. This, apart from the usual authorship and licensing data, includes details on how the module fits inside an XMML description of an application (please see the following section for a detailed list of metadata). Once the module is registered and properly described, the designer may use it inside the MAD creation tool (see Section 8.1.1) when building a new application.

When a newly designed application is executed in the MAPPER infrastructure, runtime-related information (such as deployment and availability data of the module) becomes

important. This information is usually provided by the administrator of the e-infrastructure and is automatically applied by the execution mechanisms within the MAPPER middleware to find and run the particular simulation module.

8.2.2.2 Module Descriptor Information

This section lists the initial set of metadata that needs to be stored in order for a simulation module to be useful for multiscale application designers and to be executable at runtime.

The module metadata required by application designers should comprise information stored in the XMML file, in the “submodel” section:

- name (which serves as a unique identifier), version;
- information whether module is initial, stateful or interactive;
- scale information: time, space, user-defined;
- optional model properties, important for connecting to other submodules (e.g. size of data items exchanged using ports);
- input and output ports: what kind of MML operator they are assigned, the data type exchanged using each port;
- implementation details: size, runtime, memory, cores, platform, language, required libraries.

In addition, module metadata should cover infrastructure-related data – specifically, the list of sites where the module is installed coupled with information on how to run it, e.g.:

- the home directory of the module;
- the environment module name which sets up the environment for the (scale) module.

In the first version of the Module Registry this information will be provided and maintained by the e-infrastructure administrators (as depicted in Fig. 19). In the final version we may consider an approach based on gathering information by polling various information and monitoring services already present in the e-infrastructure, e.g.: Berkeley Database Information Index (BDII), Common Information Service (CIS), Nagios or Inca (as shown in Fig. 20).

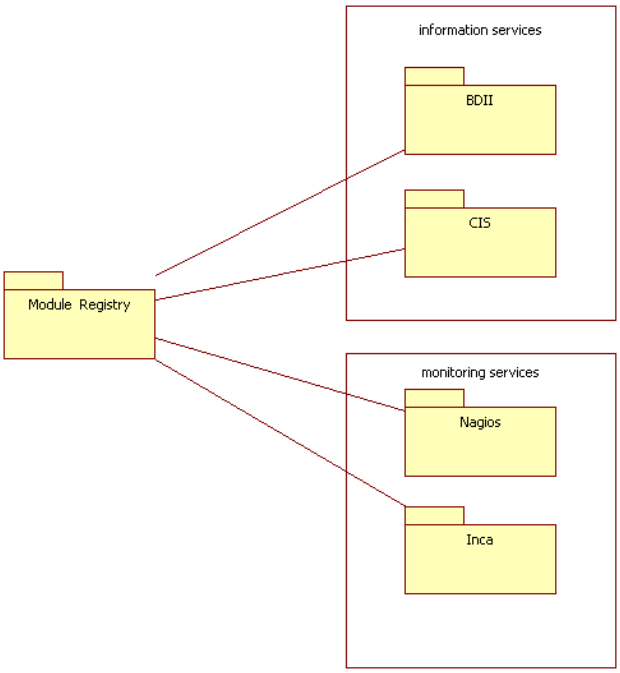

Fig. 20. Preliminary scenario of aggregating various information and monitoring services used in different e-infrastructures.

8.2.2.3 Design

The design of the registry follows the design pattern established for the XMML repository component (see Fig. 17). It uses the persistence abstraction layer to store the required metadata about simulation modules of MAPPER applications. Please consult Section 8.2.1.3 for a detailed design of this abstraction layer. Note that, contrary to the XMML repository component, the module registry does not require file storage. This is due to the fact that no module itself is stored here – all modules are assumed to be deployed in the e-infrastructure and the registry only stores metadata about each module.

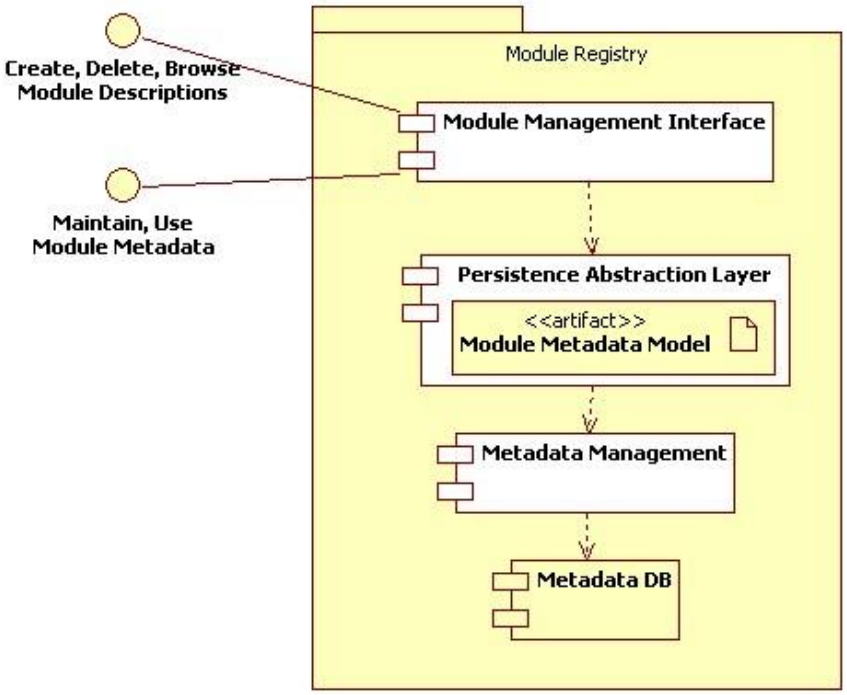

Fig. 21. The module registry re-applies the persistence abstraction layer with a different domain model to store simulation model metadata.

8.2.3 SBML toolbox

COPASI [HOOPS] is a software application for simulation and analysis of biochemical networks and their dynamics. It is a standalone program that supports models in the SBML standard and can simulate their behavior using ODEs or Gillespie's stochastic simulation algorithm; arbitrary discrete events can be included in such simulations. COPASI carries out several analyses of the network and its dynamics and has extensive support for parameter estimation and optimization. Additionally, it provides means for visualizing data in customizable plots, histograms and animations of network diagrams.

COPASI can be used in two different executable versions: a graphical user interface (CopasiUI) and a command line version (CopasiSE) which only contains the calculation engine. CopasiSE is intended for situations in which the user is not expected to interact with the software. The following use cases are examples of situations in which it would be used:

- when third-party programs manipulate COPASI files, call CopasiSE to produce results, and then inspect and continue generating other COPASI files depending on results;

- to run simulations “in the background”, which is useful when the run takes a long time;
- as a simulation engine for specialized front-ends that may be created by others.

Key features of Copasi that interest us:

- Imports and exports SBML files;
- User-friendly GUI;
- ODE solver using LSODA [PETZOLD];
- Several optimization algorithms for minimizing an objective function: genetic algorithms, particle swarm, random search, simulated annealing etc.;
- Several APIs available in C++, Java and Python through which the programmer can directly use Copasi’s internal routines;
- A CLI executable is available, useful for running batch jobs;
- Available under the open-source Artistic license 2.0.

We aim to use COPASI, among other tools, to build a “package” with our data and software routines which can then be used in MAPPER. Depending on how we will integrate COPASI, two usage scenarios are possible:

1. Our first option is to use COPASI only for its user interface, in order to create the initial models and generate the data sets. In order to enable support for running the reverse engineering and sensitivity analysis, we would need to provide several other features, such as an ODE solver, our own implementations of optimization algorithms and a script that would integrate all of these components and enable them to run on MAPPER’s middleware layer.
2. Our second option would free us from having to provide our own implementations for optimization and algorithms or use 3rd party software, by gaining access to COPASI’s internal routines. The existing APIs provide many important features, the most important ones being:
    - Creating and saving a model;
    - Loading and processing a model;
    - Running a timecourse simulation;
    - Running a parameter scan over a timecourse simulation;
    - Running an optimization task.

8.3 Execution tools

8.3.1 GridSpace Experiment Execution Engine

GridSpace Experiment Execution Engine (E3) backs GridSpace Experiment Workbench (EW) and takes responsibility for coordination of experiment run. As shown in Fig. 22, it submits exepriment parts (snippets) to Experiment Hosts (either User Interface machines or to meta-brokers). Each snippet would have assigned Experiment Host where it needs to be run, an interpreter that is installed on the Experiment Host that is in charge to evaluate snippet code. Snippet would be executed on one of many Experiment Hosts where user is granted to connect. Snippet is interpreted by exactly one interpreter. Interpreter would be of two kinds:

- regular interpreter - it maps one to one to the single executable program (possibly distributed) on e-Infrastructure, so the code of the snippet is interpreted by the program. Interpreter is run submitted to be via Experiment Hosts that support,
- pseudo-interpreter - doesn't correspond to any program but to a meta-scheduler. Then, the code of the snippet is rather job specification containing information on executable programs to run along with their run parameters.

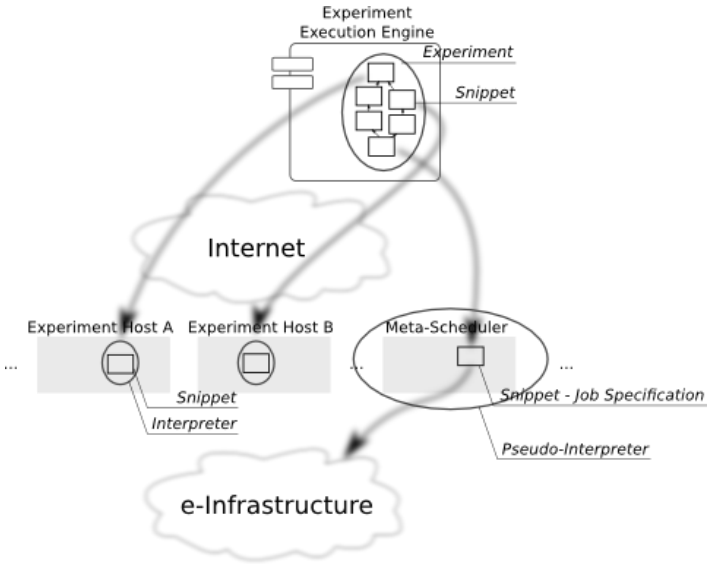

Fig. 22. Functionality of GridSpace Execution Engine.

Currently, it's configurable in E3 what interpreters are available. In the case of regular interpreters it's a simple entry in E3 configuration that specifies on which resource, which executable is be considered as an interpreter. For MAPPER's use this configuration needs to be synchronized with the modules registry. Regarding pseudo-interpreters they are plugged

in the E3 as dedicated software modules through extension point of Interpreter Provider Interface which is defined by a set of classes and interfaces. This is the way QosCosGrid GRMS and AHE clients will be incorporated in E3.

8.3.1.1 New features imposed by MAPPER's requirements

Being constantly under improvement and development process, EW needs to meet new requirements that emerged during analysis of multiscale applications. Below, the major ones are enumerated:

- Multiscale applications are likely to span over a number dispersed and heterogeneous clusters and HPC resources. Therefore, snippets, although in the scope the same experiment, have to be enabled to be targeted to different Experiment Hosts or meta brokers.
- Experiment Hosts and meta brokers involved in the application run differ in connectivity and user authentication methods. Therefore, EW has to support a number of methods starting from SSH and GSI-SSH to GSI-enabled meta brokers.
- Since multiscale applications may extensively process and exchange data, EW has to ensure effective staging of data sources. Again, the data management depends on the underlying e-infrastructure capabilities and mechanisms and EW need to seamlessly integrate with them. For example, EW may proxy ordinary User Interface machines and expose GSI-HTTP ot GSI-FTP server that would enable access to the data from other remote modules involved in application run.
- EW has to support fine-grained execution of experiments including check-pointing and partial re-execution. It means EW need to track the artifacts being produced and consumed in course of the experiment run by the respective snippets in order to avoid unnecessary recomputing of already computed results. Artifact-based record of check-points would enable partial re-execution.

8.3.1.2 GridSpace Experiment Execution Engine - AHE plugin

AHE (Application Hosting Environment) [ZASADA, ZASADA2] is a frontend for running applications on Grid infrastructures hosting Globus, UNICORE or GridSAM middleware. It also supports cross-site MPIg applications and HARC reservations. To accomplish this AHE delivers a set of stateful web services (implemented with the WSRF:Lite toolkit) which can be contacted by a command line or GUI client (Java API extraction is also possible for programmatic calls).

After authenticating with a suitable credential (e.g. proxy certificate) the client may be used to execute one of the available applications. Application-specific parameters and input files are

automatically transferred to the execution node. During execution the same client is used to monitor the state of execution and after its finished result files are transferred back to user's machine.

From the GridSpace perspective such frontend can be used to launch applications on Grid as part of a loosely-coupled MAPPER application independently of the underlying middleware. Input and output files, as well as, proxy delegation is handled by AHE. Also, if such need emerges, it is possible to register additional applications/scripts hosted by AHE. However, this would require deploying software on the underlying Grid infrastructures.

AHE client details

AHE provides scientists with application specific services to utilize grid resources in a quick, transparent manner with the scientific objective as the main driver of the activity. The AHE provides resource selection, application launching, workflow execution, provenance and data-recovery.

The AHE client is designed to be easily installed on an end user's machine, requiring only that the user has 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.

The AHE client can also be used as an API, making it easy for other tools to launch remote grid based applications via AHE. AHE can submit to a variety of back end resource managers, including GridSAM, Unicore 6 and Globus GRAM 4. A proxy certificate, stored on a MyProxy server, is used by AHE to submit jobs on a user's behalf. The detailed description of AHE client usage can be found in the Annex (Section 10).

8.3.1.3 GridSpace Experiment Execution Engine - QosCosGrid GRMS plugin

QosCosGrid [KUROWSKI] exposes an interface for submitting, monitoring and managing jobs - considered as a collection of computation tasks with dependencies between them - through GRMS service/ This service is implemented as a web service-accessible resources using Globus Toolkit. Aside other trivial operations on jobs, the most vital part of this interface is about describing jobs using XML-based - so called - Job Profile document.

There are several points of linkage between QocCosGrid and GridSpace identified, namely:

- **Submitting job.** GridSpace Experiment Workbench (portal) needs to be enabled to submit and monitor jobs through GRMS. In particular, Experiment Workbench has to

generate propoer Job Profile descriptors basing on the user-provided business-logic code as well as other remarks on how the job is supposed to be run. (e.g. what resources involve).

- **Staging data in and out.** GridSpace needs to ensure accessibility of data files used by jobs running on the QosCosGrid infrastructure. The scope of this problem reach beyond GridSpace and QosCosGrid, but also relates to the way GridSpace and QosCos Grid deals with a storage infrastructure of the MAPPER framework.
- **Delegation of security credentials.** GridSpace Experiment Workbench needs to pass security credentials to GRMS service which, on behalf of the GridSpace (precisely: on behalf of a user being logged in in the GridSpace Experiment Workbench), accesses data and computational resources. This also means that resources accessed by task running on QosCosGrid need to support delegated security credentials.

QCG client details

For the integration purposes with other MAPPER services like GridSpace2 or Application Hosting Environment (AHE) and for direct use by end-users the API and java based command-line client to QCG-Broker service was implemented. The QCG-Broker has been implemented as federations of WS-I compliant web services using a WS-Addressing approach as a standardized way of including message routing data within SOAP headers. The security model assumes usage by clients X509 proxy certificates signed by trusted Certificate Authority (CA) for authentication and authorization purposes. The integration between QCG services and other tools and services developed in MAPPER project can be done using either the java API or the command-line client taking internally advantage of the aforementioned API. As it was stated the QCG-Broker has a WebService interface described formally by WSDL format document what makes it accessible from any language supporting the web services technology. The QCG-Broker client is delivered as package containing set of jar-type files and lunching script. The detailed description of QCG client usage can be found in the Annex (Section 11).

8.3.1.4 Access to TeraGrid resources.

TeraGrid resources are accessible via a regular remote shell login. This E3 communicate with the resources through (GSI-)SSH protocol. Behind the scene the cluster uses Sun Grid Engine (version 6.2) as the batch job submission system so in order to use it smoothly from GS2 one might need a compiled SGE Gem (see D4.1).

8.3.2 Result Management

8.3.2.1 Use Cases

The usage scenarios of the result management component are quite simple (see Fig. 23). The functionality will be delivered through a conceptually simple interaction of browsing, uploading and downloading files on a remote server. While the notion of an *application result* might be considered more general than just files, the practical approach is to handle such objects as files (mainly due to large size of the results in comparison to available operational memory).

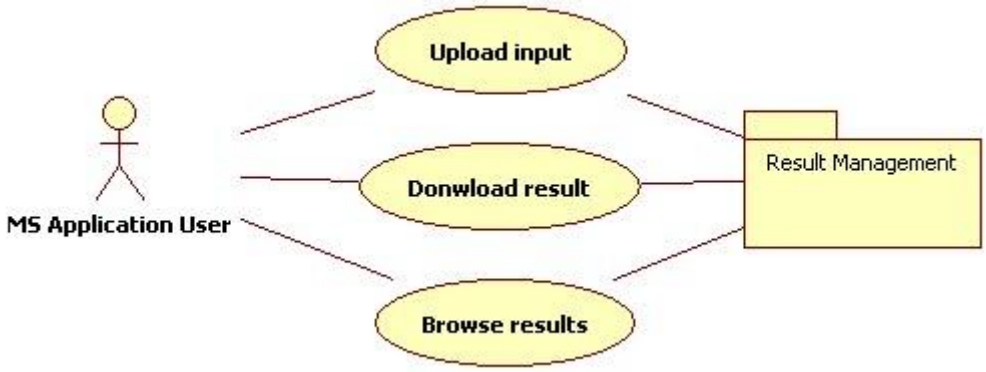

Fig. 23. The functionality of the MAPPER result management component from the point of view of a person running an instance of a MAPPER application.

The user of a MAPPER application will be able to upload the files onto a server (where they are available for the application simulation modules as the input), browse what output was produced by application runs and, when desired, download that output to the local computer.

8.3.2.2 Design

The design of the result management component takes into consideration two possibilities for the result storage mechanism (see Fig. 24). The first possibility is relatively straightforward - the user accesses the remote machine (through the Experiment Workbench portal) and is able to browse the contents of his or her home directory on the remote file system. This is done with the help of the ssh/scp protocol being employed to relay the user communication through the GridSpace server (which hosts the Experiment Workbench web application) up to the target login machine, See Section 8.1.2 for the details. Therefore, whenever the user uploads or downloads a file, the file is eventually being handled by the file system mechanism on the login host.

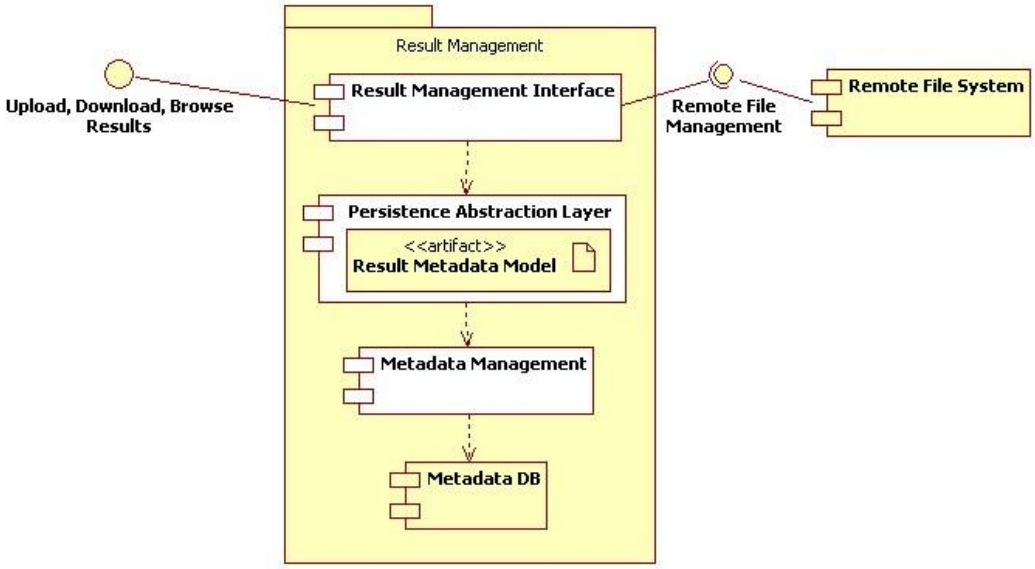

Fig. 24. The result management in MAPPER is delivered both through the direct access to user's files on a target machine and through a dedicated result location registry (for other means of result persistence, like dedicated storage facilities).

Another possibility of file storage mechanism is being delivered through the reuse of the persistence abstraction layer (see Section PAL-SECTION). This approach will be needed when some dedicated result storage facilities (like e.g. gridftp servers or LFC file catalogues) are employed within the MAPPER framework. In such a case the result management will store the relevant information about the location and the retrieval method for a particular application execution result (for instance, a gridftp URL or an LFC logical file name).

Please note that, in contrary to the design of the XMML repository (see Section 8.2.1), the result management is not planned to use the the built-in file management component to handle application execution result files. This is due to the expected large amount and size of such data - the file storage mechanism built in the persistence layer is rather suited for smaller documents.

8.4 Provenance

8.4.1 Use cases

The central issue connected to the provenance relates to collecting information about how a given experiment was carried out step-by-step along with the associated parameters, inputs and outcomes. In particular, provenance will be used to answer the following questions:

- who conducted the experiment,

- where was the experiment carried out (a particular execution environment: AHE, QCG etc.),
- how a given outcome was obtained (i.e. whether it was created by a process being part of the experiment); this concerns the final outcome as well as any intermediate values (e.g. parameters passed from one process to another one, partial results etc.),
- which processes were involved in experiment execution.

The aforementioned information can be used to:

- find possible mistakes and errors in an experiment design or execution; for example an experiment designer may regard a given outcome as invalid. Using provenance data he/she may start to examine intermediate results to find the one that was invalid and a process that returned it,
- improve the experiment to be more efficient or accurate; provenance may be helpful to figure time-consuming processes and flaws in the experiment design (i.e. multiple invocations of the same process).

Use cases for Provenance collector (a component responsible for collecting and providing provenance data) are shown in Fig. 25.

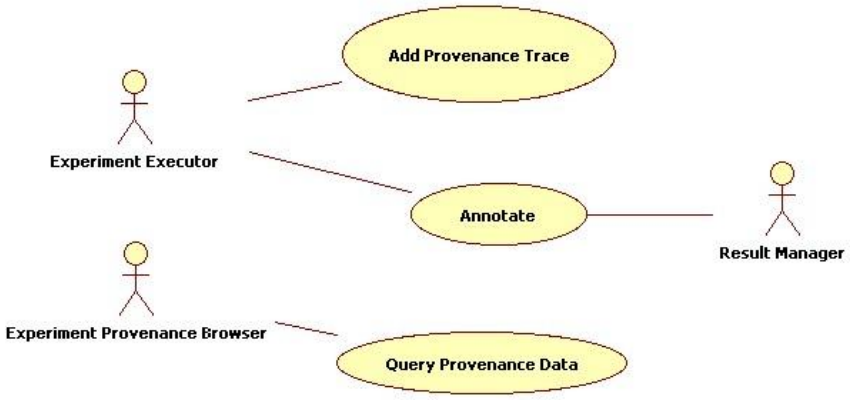

Fig. 25. Use cases for Provenance collector.

The Experiment Executor is responsible for executing a whole experiment. Since it has the most complete knowledge of entities of the experiment its responsibility is to insert the traces of experiment execution into the provenance subsystem.

Annotation is enriching raw provenance data with a proper context of experiment (semantics regarding variables - names, values as well as processes etc.). Again, the Experiment Executor is suitable for this task. Moreover, the Result manager is also helpful as it may contain valuable data about experiment results that also have to be annotated.

Finally, provenance collector enables external components to query for collected data using special query languages.

8.4.2 Design

Fig. 26 presents a high-level view on the provenance subsystem. The component will have two well-defined interfaces to communicate with other components: query and trace storing. The former may be used by a query browser or other entity to request provenance data. The latter is used by components that possess any knowledge on the experiment execution process or information about the experiment itself (such as the Experiment Executor or Result Manager) to notify the provenance subsystem about significant events.
The exact implementation of these interfaces will be based on popular, well-known interfaces like REST.

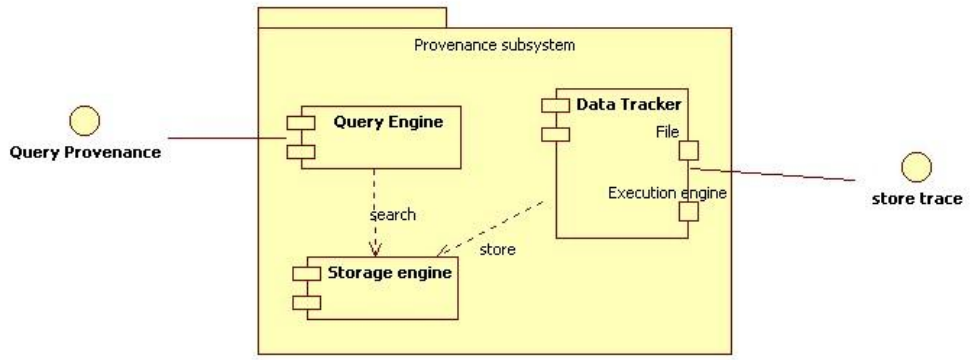

Fig. 26. The design of Provenance system.

There are three main components of the Provenance system:

Query Engine is used to accept queries from other entities regarding the collected provenance data. It verifies a request and processes data in the Storage Engine, accordingly. A result is sent back to the entity that has issued the request. A typical use would be a web application sending a request using REST protocol, receiving a response written in JSON format and displaying it in a proper way to the user.

Data Tracker is a component that keeps track of the events that occur when running an experiment. This can be done in two modes. The first one is passive, meaning the tracker awaiting traces from external components on well defined interfaces (as mentioned before). The second one is active which means observing changes in the experiment execution

environment: log files, creating outputs etc. Each trace input format (files, execution engine traces etc.) will have its own entry point installed as plugin.

Storage Engine is used to provide the persistence of the gathered provenance data. The exact format and implementation depends on the assumed provenance data representation (please see Section 8.4.5). For RDF format a triple storage is the best way. An alternative may be a Sint storage for Semantic Integration (please see Section 8.4.5).

8.4.3 Data definition

Provenance data can be defined as a direct acyclic graph as in Open Provenance Model *(http://openprovenance.org/).* As it is meant to be used in scientific and business workflows it should be suitable for the considered system.
Each node represents a single entity: process, artifact, and agent. Artifact represents a piece of state of any "thing" present in the experiment environment: file, variable, input data etc. Artifacts cannot be changed, they only can be produced by processes. Agents represent the context of a process execution (why it was invoked, who did it and where).
Artifact examples:

- input data,
- experiment results,
- user inserted parameters,
- any parameters exchanged by processes taking part in a workflow.

Process examples:

- any process executed on UI.

Agent examples:

- a workflow,
- an execution engine.

These entities are connected with dependencies. These are the edges of the provenance graph. The thorough set of dependencies is yet to be discussed. Examples include:

- process uses an artefact,
- process created an artefact,
- process invoked a process,
- artifact was derived from another artefact,

- agent controls a process.

The detailed semantics of these relations can be found in the OPM specification.

8.4.3.1 Annotations

The aforesaid data structure does not provide any semantic value (is domain agnostic). For example, the following information that is very usable is not provided:

- timestamps of process start and finish,
- parameter names and values,
- artifact types (file, user input) and semantic (star age, protein type etc.),
- user names.

In order to provide these information annotations will be introduced into the graph. Each node or edge may have an arbitrary number of annotations. Typical annotations are:

- timestamps of process start and finish,
- parameter names and values.

Annotating is a complex task and requires cooperation of several components, e.g. of the Execution Engine or Result Ranager. Exact annotation sources and annotation types are yet to be defined.

8.4.4 Provenance data acquisition

One of the main issues of provenance is provenance data acquisition. Each process invocation or finish, artifact creation, or agent action has to be reported. This should be done by the Execution engine and the UIs involved. Therefore, an instrumentation of process executors is required. Some UIs will have their own provenance capabilities (i.e. GridSpace) that can be used for the discussed purpose.
Similarly, some data can be derived from the experiment in an non-invasive way. Below there are some examples:

- XMML files: they contain some initial parameters, model identifiers (both annotate processes); it is a valuable source of annotations,
- process execution log files (start and finish timestamps).

8.4.5 Data sharing and querying

Provenance data has to be accessible to users or other components in a way that enables answering questions mentioned in Section 8.4.1. Moreover, it should be possible to retrieve

all the information (properties) about any specific provenance entity (process, artifact, agent). This can be done using several approaches. Two of them are especially noteworthy:

- Provenance data will be published using RDF. It is a well established format often used to share scientific data even between completely different systems or execution environments. All querying operations can be performed using a dedicated query language based on SPARQL. This solution requires an efficient storage for RDF triples as they will be read and searched for extensively.
- Semantic Integration [GUBALA] - this approach has already been used in some experiment execution environments (ViroLab) for exchanging information between experiments from different domains. Moreover, it uses a storage solution that is used in GridSpace to ease integration. However, this solution has not been used for provenance purposes therefore it has to be checked for typical provenance queries support.

9 Annex 2. MAPPER Application Technical Inquiry

9.1 Goal

The goal of this inquiry is to start collecting technical information about applications to be supported in the scope of the MAPPER project. Complete and detailed application descriptions are crucial to successfully address MAPPER objectives of adaptation of existing software tools and emergence of new ones facilitating multiscale application development and operation. We believe that this inquiry is the best way to get technical Work Packages familiarized with applications and a good starting-point for further cooperation.

9.2 Instructions

As it's always very hard to approach all the individual cases with generic inquiry, feel free to include all the information you consider vital even if not foreseen by the questions below. If any of the questions doesn't concern your particular case – please omit it. Please send filled out inquiry to *katarzyna.rycerz@agh.edu.pl*. As soon as official MAPPER document repository is launched, the inquiries will be filed there.

9.3 Questions

- What is the name of the application?
- Who is the contact person in technical matters concerning the application?
- Which are the most relevant publications describing the application? Are there any user or developer manuals?
- Is the application software freely available as open source or as a binary?
- What is current status of the application (concept, design, development of first prototype, first prototype, further development and version, release, deployment)? Are all the computing steps or components of equal maturity, or do some require further development to be more stable/usable/deployable/reusable?
- What is the concept of the application? How would you depict the structure of the application? How the application is composed? What are the components?
- What are the scales covered by each simulation module of the application?
- What are the differences between application runs? Do you only change parameters and input data, or does the structure of the application change between runs (can e.g. some steps be skipped or replaced)?

- Where do the computations take place? Where do the respective components operate? On which computational resources they run (clusters, high-performance computers, single computer - desktop, grid)? What is the architecture of the computers used (multicore, symmetric multiprocessing, massively parallel processing) ?
- How do components deal with information exchange?
    - Do they communicate through message passing? If so, how large and how frequent the messages are?
    - Do they share data (in memory, database, files or other)?
    - Where are the input and output data stored?
    - How do the scales correspond - i.e. what is the direction of data feed from one module to another and whether there is a feedback or not?
    - How large are these data feeds, i.e. how large a data chunk needs to be conveyed between models every communication step?
    - Is that data a binary or a plain text document?
    - Do you need the data passed from one model to another converted or adjuste d?
- What external (not developed by yourselves) libraries, software modules, frameworks, services your application makes use of?
- How long a typical production run on a target (suitable) computing platform of a module would take?
- Which of the GridSpace functionality presented during Kick-off meeting (see agenda and be there) do you find potentially useful for your application:
    - script-based application building support
    - direct SSH connection to target machines and working with your home directory files through a web interface
    - components supporting time management between simulation modules
    - iBuilder (graphical application building support)
    - registry for intermediate and final results of the application runs, to be browsed and viewed later on.

- result provenance - to be able to see metadata information about produced past results (origin, process of production, timestamp, etc.) and to be able to query by such metadata
- other presented - please specify
- other not presented - please specify.

- Are you familiar with CellML (http://www.cellml.org/) or SBML (http://sbml.org/Main_Page) concepts of storing and exchanging computer-based mathematical models? If yes, do you find them useful for your application? Why? Do you think you will need a special language describing your models to share them with other potential users?

10 Annex 3. The AHE client usage

Operation	Description
ahe-listapps -e endpoint -a application -help	Lists the applications available in the AHE application registry. Upon successful submission the command displays the applications available from this AHE server installation, with job factory endpoints.
ahe-destroy -s simname -i index -r -e endpoint -help	Destroys the simulation from the AHE job registry. Upon successful submission the command reports that the job has been destroyed.
ahe-prepare -RMVirtualMemory virtualmemory -RMMemory memory -e endpoint -RMIP ipaddress -help -s simulationName -wallTimeLimit time -RMArch arch -RMDisk disk -app application -RMType NGSorTeragrid - RMCommonName rmname -RMOpSys opsys -RMCPUCount cpucount	Creates a stateful resource on the AHE server to manage the job, and returns a list of potential machines to run the job. Optional arguments can be used to constrain the list of machines returned. Upon successful submission the command displays a list of the potential target machines that the job could be run on.
ahe-start -s simname -i index -n cpucount -wallTimeLimit time -config file -RM rmname -e endpoint -help	Processes the job configuration file to discover job input and output files, stages those files to the intermediate webdav server, and submits the job to the specified target machine. Upon successful submission the command reports the status of the job.
ahe-monitor -s simname -i index -e endpoint -help	Reports the status of job. Upon successful submission the command reports the AHE status of the job.
ahe-getoutput -s simname -i index -d -e endpoint -l localdir -help	Retrieves a job's output files to the local machine. By default files will be staged back to the location specified in the job's configuration file. Note that output can only be retrieved when the job's status is AHE DONE (see ahe-monitor). Upon successful

	submission the command will retrieve the job's output files to the specified location.
ahe-getproperties -s simname -i index -e endpoint -help	Upon successful submission the command displays the properties associated with the specified job, including the job's input and output files and status. Lists the applications available in the AHE application registry.
ahe-list -e endpoint -help	Lists the jobs owned by the submitting user in the AHE job registry, and caches the result locally. Upon successful submission the command displays the jobs contained in the registry and updates the local job cache.
ahe-terminate -s simname -i index -e endpoint - help	Terminates a running job. Upon successful submission the command reports the job has been terminated.
ahe-refresh -e endpoint -help	Upon successful submission the command updates the local job cache.

An AHE application can have one of the following states:

- AHE_PREPARING - The AHE Application instance is being constructed
- AHE_FILES_STAGED - Files have been staged to the intermediate file staging area
- AHE_JOB_BUILT - The AHE application instance has been constructed.
- AHE_PENDING - The AHE application instance is being scheduled.
- AHE_STAGING_IN - Data is being staged to the remote resource.
- AHE_STAGED_IN - Staging data to the remote resource is complete.
- AHE_STAGING_OUT - Data is being staged from the remote resource.
- AHE_STAGED_OUT - Staging data from the remote resource is complete.
- AHE_ACTIVE - The application is running.
- AHE_EXECUTED - The application execution has completed.
- AHE_FAILED - The application failed.
- AHE_DONE - The application instance is complete.
- AHE_UNDEFINED - An undefined state has occurred.
- AHE_TERMINATING - The application instance is being terminated.
- AHE_TERMINATED - The application instance has terminated

11 Annex 4: QCG client usage

The command-line java based client to QCG Broker can operate in two modes:

- batch mode – that executes single operation with arguments passed directly to the client during its invocation. The batch mode allows to use the client in any kind of scripts mostly in cases when the processing of output is needed to steer the experiment,
- console mode – that works similar to shell console in which user can type in lines with operations and arguments to be executed by service. The console mode gives additional useful features like aliases, history accessible by arrows-keys, creation and management of user proxy, help functionality.

The usage of the QCG client depends on the mode:

- for batch mode: “grms-client OPRATION [ARG1 .. ARGn]”
- for console mode: “grms-client -console” and then user is prompted to type in lines in format “OPERATION [ARG1 .. ARGn]” to be processed by client.

Regardless from the mode the QCG-Broker java based command-line client supports following list of operations:

Operation	**Description**
submit_job <desc_file> [GRMS | JSDL]	submits a job to be executed. The description of job can be expressed either in native QCG-Broker language or if it is possible in JSDL one. If the description is valid client returns to the user a globally unique job identifier, which unambiguously identifies the job in the system. QCG defines jobs as a sets of dependent tasks that constitute a logical whole (workflow). Each task is executed by system only if all tasks it depends on are in specified by the user states.
list_jobs [<limit>] [<status>]	lists jobs belonging to the user. It is possible either to limit number of jobs or to display only ones in given state. All possible states are listed below the table.

list_user_jobs [<limit>] [<status>] <user>	lists jobs belonging to the given user. The functionality is destined for administrative purposes.
test_description<desc_file> [GRMS | JSDL]	validates job description
translate_description<desc_file> JSDL	translates job description to native QCG-Broker one
job_info <jobId> [showJobDesc]	return complex information about the given job. If the showJobDesc is false the job description is not shown
cancel_job <jobId>	cancels execution of the given job
commit_job <jobId>	allows to approve the job submitted with two phase commit mechanism to be processed by the system. The two phase commit mechanism can be used to register notifications before the processing of the job will be started by broker.
list_tasks <jobId> [<status>]	lists tasks belonging to given job. Optionally it is possible to specify the task's status. Possible task statuses are listed below the table.
tasks_statuses <jobId> <summary>	lists tasks constituting the given job with their statuses
register_job_notification<jobId> <url>	registers notification consumer for the given job
list_job_notifications<jobId>	lists notifications registered for the given job
register_tasks_notification<jobId> <url>	register notification for all tasks of the given job
monitor_job <jobId> <interval>	monitors status changes of tasks belonging to given job
monitor_task <jobId> <taskId> <interval>	monitors status changes of allocations belonging to the given task
task_info <jobId> <taskId> [showDesc [limit]]	displays information about the given task. If the showDesc is false the task description is not shown. If the limit is specified the history of the task is limited to given value.
register_task_notification<jobId> <taskId> <url>	registers task's notification consumer
list_task_notifications<jobId> <taskId>	lists task's notifications

cancel_task <jobId> <taskId>	cancels execution of the given task
commit_task <jobId> <taskId>	commits the given task to be processed by the system
reserve_resources[<taskId>]<job_desc> [GRMS | JSDL]	reserve resources that meet either the wole job or given task requirements. The reservation identifier is returned
reservation_info<reservationId>	return complex information concerning the given reservation: list of reserved resources, local identifiers of reservations, reservation time slot
cancel_reservation<reservationId>	releases reserved resources

List of Job statuses:

- UNCOMMITTED - the job was submitted with two phase commit option and waits to be committed,
- SUBMITTED – the job was submitted to the system and is executed by the system,
- SUSPENDED – the job was suspended,
- ACTIVE – the job is active, at least one task is processed,
- FINISHED – the job was completed,
- FAILED – the job (at least one crucial task belonging to the job) failed
- CANCELED – the job was canceled by the user,
- BROKEN - one or more of crucial tasks failed, system waits until active tasks will finish and change the status of the job to FAILED.|

List of Task statuses:

- UNSUBMITTED – the task cannot be started because of dependencies,
- UNCOMMITED - the task waits to be committed,
- QUEUED – the task was put into the queue and waits for execution,
- PREPROCESSING – system makes some actions needed to start the task (looks for the resource, stages in files),
- PENDING – the task is pending in the queueing-system,
- RUNNING – the task is active,
- STOPPED – the task was finished or was checkpointed, but system did not start staging out files,

- POSTPROCESSING – system makes some actions needed to complete the task, for example stages out files, cleares working environment, etc.,
- FINISHED – the task was completed,
- SUSPENDED – the task was suspended,
- FAILED – the task failed,
- CANCELED – the task was canceled by the user.

12 References

[CAIAZZO] Alfonso Caiazzo, David Evans, Jean-Luc Falcone, Jan Hegewald, Eric Lorenz, Bernd Stahl, DinanWang, Jorg Bernsdorf, Bastien Chopard, Julian Gunn, Rod Hose, Manfred Krafczyk, Patricia Lawford, Rod Smallwood, Dawn Walker, and Alfons G. Hoekstra. Towards a Complex Automata Multiscale Model of In-Stent Restenosis; ICCS 2009, Part I, LNCS 5544, pp. 705–714, 2009

[CIEPIELA] E. Ciepiela, D. Harezlak, J. Kocot, T. Bartynski, M. Kasztelnik, P. Nowakowski, T. Gubała, M. Malawski, M. Bubak; Exploratory Programming in the Virtual Laboratory, in Proceedings of the International Multiconference on Computer Science and Information Technology pp. 621–628

[COLE] Murray Cole. "Algorithmic Skeletons: structured management of parallel computation" MIT Press, Cambridge, MA, USA, 1989

[COSTER] David Coster. Plasma Physics: Scientific and Computational Challenges: Fusion, EFDA, ITM and EUFORIA. 9th International Conference on Science, Arts and Culture. Veli Lošinj (Croatia). 2009.

[ENGQUIST] W, Engquist B, Li X, et al. Heterogeneous Multiscale Methods: A Review. Commun. Comput. Phys. 2007; 2: 367-450.

[FALCONE] J.-L. Falcone, B. Chopard and A.G. Hoekstra. MML: towards a Multiscale Modeling Language. Procedia Computer Science (2010) vol. 1 (1) pp. 819-826

[GONZALEZ] Horacio González-Vélez and Mario Leyton "A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers" Software: Practice and Experience Volume 40, Issue 12, pages 1135-1160, November/December 2010

[GUBALA] T. Gubala, M. Bubak, P.M.A. Sloot: Semantic Integration of Collaborative Research Environments, Chapter XXVI,in: M. Cannataro (Ed.), Chapter XXVI, Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare, Information Science Reference, 2009, IGI Global ISBN: 978-1-60566-374-6

[MUSCLE] Jan Hegewald, Manfred Krafczyk, Jonas Tölke, Alfons G. Hoekstra, and Bastien Chopard. An agent-based coupling platform for complex automata. In Marian Bubak, G. Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, ICCS (2), volume 5102 of Lecture Notes in Computer Science, pages 227–233. Springer, 2008.
[HLA] High Level Architecture IEEE standard 1516

[HOEKSTRA10] Hoekstra et al. Complex automata: multi-scale modeling with coupled cellular automata. Simulating Complex Systems by Cellular Automata (2010)

[HOEKSTRA07] Hoekstra et al. Towards a complex automata framework for multi-scale modeling: formalism and the scale separation map. ICCS 2007 (2007)

[HOOPS] Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P. & Kummer, U.: COPASI – a COmplex PAthway Simulator, Bioinformatics, 22(24):3067–3074, 2006.
[KUROWSKI] Kurowski K, Kravtsov V, Schuster A, et al. Grid-enabling complex system applications with QosCosGrid: An architectural perspective. In (eds). The Int'l Conference on Grid Computing and Applications (GCA'08), vol. 2008,

[MCT] .Larson, R. Jacob, E. Ong "The Model Coupling Toolkit: A New Fortran90 Toolkit for Building Multiphysics Parallel Coupled Models." 2005: Int. J. High Perf. Comp. App.,19(3), 277-292

[PETZOLD] Petzold, L.: Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. SIAM J. Sci. Stat. Comput.. 4. 136 - 148. 1983.

[RYCERZ10] K. Rycerz, M.Bubak, Collaborative Environment For HLA Component-Based Distributed Multiscale Simulations accepted by: W. Dubitzky et al "Large Scale Computing Technologies for Complex System Applications", Wiley&Sons.

[MUSE] S. Portegies Zwart, S. McMillan, at al. A Multiphysics and Multiscale Software Environment for Modeling Astrophysical Systems, New Astronomy, volume 14, issue 4, year 2009, pp. 369 - 378

[SUTER] Suter JL, Anderson RL, Greenwell HC, et al. Recent advances in large-scale atomistic and coarse-grained molecular dynamics simulation of clay minerals. Journal of Materials Chemistry 2009; 19: 2482-2493.

[THANG] 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.

[ZASADA] S. J. Zasada and P. V. Coveney, "Virtualizing Access to Scientific Applications with the Application Hosting Environment", Computer Physics Communications,180, (12), 2513-2525, (2009), DOI: 10.1016/j.cpc.2009.06.008.

[ZASADA2] S. J. Zasada and P. V. Coveney, "From campus resources to federated international grids: bridging the gap with the application hosting environment", Proceedings of the 5th Grid Computing Environments Workshop, Article No.: 10, (2009), ISBN:978-1-60558-887-2