D 8.4 Final validation and integration with external modules

Project acronym: *MAPPER*

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

Grant agreement no.: 261507

Due-Date:	30 september
Delivery:	
Lead Partner:	Cyfronet
Dissemination Level:	Public
Status:	After corrections from QAB
Approved:	
Version:	1.4

DOCUMENT INFO

Data and version number	Author	Comments
17.06.2013 0.1	Katarzyna Rycerz	Plan of the document
27.06.2013 0.2	Joris Borgdorff	Section about MML
2.07.2013 0.3	Katarzyna Rycerz	Section with general overview of the tools
2.07.2013 0.4	Robert Pajak	Download statistics section
3.07.2013 0.5	Katarzyna Rycerz	Evaluation of tools efficiency
22.07.2013 0.6	Tomasz Gubała	Sections about MaMe
03.09.2013 0.7	Daniel Harężlak	Sections about MAD, integration methodology and QCG integration
04.09.2013 0.8	Maciej Pawlik	Sections about AHE integration, provenance
05.09.2013 0.9	Eryk Ciepiela	Sections about GridSpace Experiment Workbench and Execution Environment
10.09.2013 1.0	Katarzyna Rycerz	Minor corrections, formatting
12.09.2013 1.1	Katarzyna Rycerz	Minor annex corrections and formatting
14.09.2013 1.2	Piotr Nowakowski	Editing and proofreading
14.09.2013 1.3	Katarzyna Rycerz	Minor corrections
20.09.2013 1.4	Katarzyna Rycerz	Corrections according to QAB
22.09.2013 1.5	Katarzyna Rycerz	Minor corrections

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Executive summary

This deliverable describes the final prototype of multiscale programming and execution tools in the MAPPER project. Specifically, D8.4 describes the tools facilitating creation and execution of multiscale applications whose structure is specified with the Multiscale Modelling Language (MML). The presented tools support composition of multiscale applications from existing single-scale submodules installed on e-infrastructures. Following composition, such applications are executed. The tools are designed so that , by applying a uniform multiscale specification, the applications and submodules are reusable, i.e. the tools support building different multiscale applications from the same modules and switching between different versions of the modules with the same scale and functionality. Applications consist of computationally intensive simulation modules requiring HPC or Grid resources, often implemented as parallel programs, with tight (cyclic), loose (acyclic) or hybrid coupling schemes.

The final prototype contains improved programming and execution tools: the application composition tool called Multiscale Application Designer (MAD), a registry for application modules descriptions (MAPPER Memory – MaMe), a repository of MML applications descriptions, as well as tools supporting high-level execution – the GridSpace (GS) Experiment Workbench (EW) and the GS Execution Engine, for executing the applications on selected resources. This documents also describes the Provenance and Result Management system. Additionally, we discuss integration of the described tools with external modules: QCG-Broker and the Application Hosting Environment (AHE), MAPPER services that interface underlying e-infrastructures. The description covers the detailed architecture of each tool along with its specific features. Links to the prototypes, code repositories and demonstrations are provided.

The document is organized as follows: Section 2 lists contributors to this deliverable while Section 3 contains a glossary of frequently used terms. In Section 4 we briefly describe the architecture of the prototype and its relation to design presented in D8.1, and implementation status presented in D8.2 and D8.3. Detailed information about each tool can be found in Section 5. In Section 6 we include links to prototypes, code repositories and/or demonstration videos. Section 7 describes the integration methodology applied in the MAPPER project. Section 8 briefly summarizes external modules integrated with presented tools. Section 9 outlines evaluation of the performance of WP8 tools. We conclude our description in Section 10. Details regarding the functionality of selected tools can be found in

the Annex (Section 12). The status of MAPPER applications and their use of the presented tools can be found in D7.3.

2 Contributors

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

Cyfronet: K. Rycerz, D. Harężlak, M. Pawlik, E. Ciepiela, T. Gubała, R. Pająk, M. Bubak, P. Nowakowski

PSNC: M. Mamoński ,T. Piontek

UvA: Joris Borgdorff

UU: Alexandru Mizeranschi

UCL: Stefan Zasada

UNIGE: M. Ben Belgacem

3 Glossary of terms

This document makes use of the following terminology:

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

Asset : an entity indicating an input or output file of the application**.**

Car-Parrinello Molecular Dynamics (**CPMD):** package containing a parallelized plane wave/pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics.

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

Executor: a common entity for hosts, clusters, grid brokers etc. – any module capable of running software which is already installed on it (represented as interpreters).

Experiment host: host where a GridSpace experiment is executed

Filter: in MML terminology, a one-to-one type of relationship between submodels

gMML: see MML

GUI: Graphical User Interface

jMML : Java library supporting MML

GridSpace experiment: a set of snippets in various script languages stored in an XML file.

Interpreter: a software package accessible from any scripting language available on any

infrastructure available to the MAPPER community. Examples of interpreters include MUSCLE[1] and LAMMPS[2] tools. We assume that this software is installed in WP4.

JobProfile: see QCG JobProfile

Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS): package supporting classical molecular dynamics simulations.

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

Mapper: in MML terminology, a mapper is a one-to-many relationship between single-scale submodels. Note the difference between mappers and the MAPPER project.

MAPPER memory (MaMe)**:** semantics-aware persistence store for MAPPER metadata based on xMML descriptions

Multiscale Application Designer (MAD): MAPPER application composition tool

Metadata: data describing other data (e.g. a link to an actual file, but not file itself)

Multiscale model: the model of a multiscale process.

Multiscale Modelling Language (MML): a high-level concept of the language that describes single-scale submodels and their connections. The connection can be realized by mappers (one-to-many connection) or filters (one-to-one connection with data filtering). It has several representations; the ones described in this document being xMML and gMML:

- **xMML**: XML representation of MML that contains all information about application structure. The latest version of the xMML specification can be found at http://napoli.science.uva.nl/xmml/xmml.tar.gz.
- **gMML:** graphical representation of MML that contains only some of the information about application structure, useful for modellers and application developers.

Multiscale Coupling Library and Environment (MUSCLE): a communication library that can be used to connect modules implementing single-scale models into a multiscale simulation. The structure of the application is described in a CxA file.

Submodel (Single-scale model): a model of a single-scale process. In the context of a multiscale model, a submodel.

Snippet: a piece of code in a scripting language.

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

[1] http://apps.man.poznan.pl/trac/muscle

[2] http://lammps.sandia.gov/

System Biology Markup Language (SBML): an XML-based language for representing models. It's oriented towards describing systems which involve biological entities and are 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 Quality of Service (QoS) guarantees.

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

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

Registry: a place where information (metadata) about certain entities (in our case, simulation modules) is registered (note that the modules themselves are not stored there).

Task graph: an acyclic directed graph representing 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): a machine accessible directly (via SSH) by users from which they can access other (Grid, PBS) resources

xMML: see MML

4 General overview of multiscale programming and execution tools

The general architecture of the multiscale programming and execution tools is shown in Fig. 1, which is an enhanced version of the architecture presented in D8.3, in accordance with evolving user requirements.

The first group – user interfaces and visual tools – developed in task 8.1, comprises the Multiscale Application Designer (MAD) – a visual tool for composing single-scale models and their mappers into full multiscale applications; the GridSpace Experiment Workbench (EW), which is a front-end for the GridSpace Execution Engine, as well as interfaces to other programming and execution tools.

The second group, i.e. programming tools (developed in task 8.2), contains MAPPER memory (MaMe) – a registry for applications building blocks, i.e. single scale models and their mappers – and an xMML repository of structural descriptions of composed applications in a special XML-based language called xMML (see subsection 5.1.1).

The third group – execution tools – developed in task 8.3, includes the GridSpace Execution Engine used for execution of applications built in MAD. GridSpace calls arbitrary software required by applications and registered in the GridSpace registry of interpreters. An important interpreter is the Multiscale Coupling Library and Environment (MUSCLE) used by tightly coupled multiscale applications. To access various e-infrastructures, GridSpace is integrated with external modules i.e. QCG-Broker and the AHE interoperability layer on the API level. QCG-Broker and AHE are MAPPER services that provide access to e-infrastructures with additional functionality required by multiscale applications like co-allocation and resource reservation (QCG) or unified access to various grids like globus or UNICORE (AHE). Direct access via SSH is also supported.

Result Management takes advantage from the Provenance system (Task 8.4) capable of saving snapshots of experiment results together with their metadata. The user is able to view experiment results and metadata using the Provenance Interface.

The following chapters contain detailed descriptions of the presented tools, together with their architecture and features, lists of changes from the second prototype (described in D8.3) and links to prototypes, code repositories and/or demos.

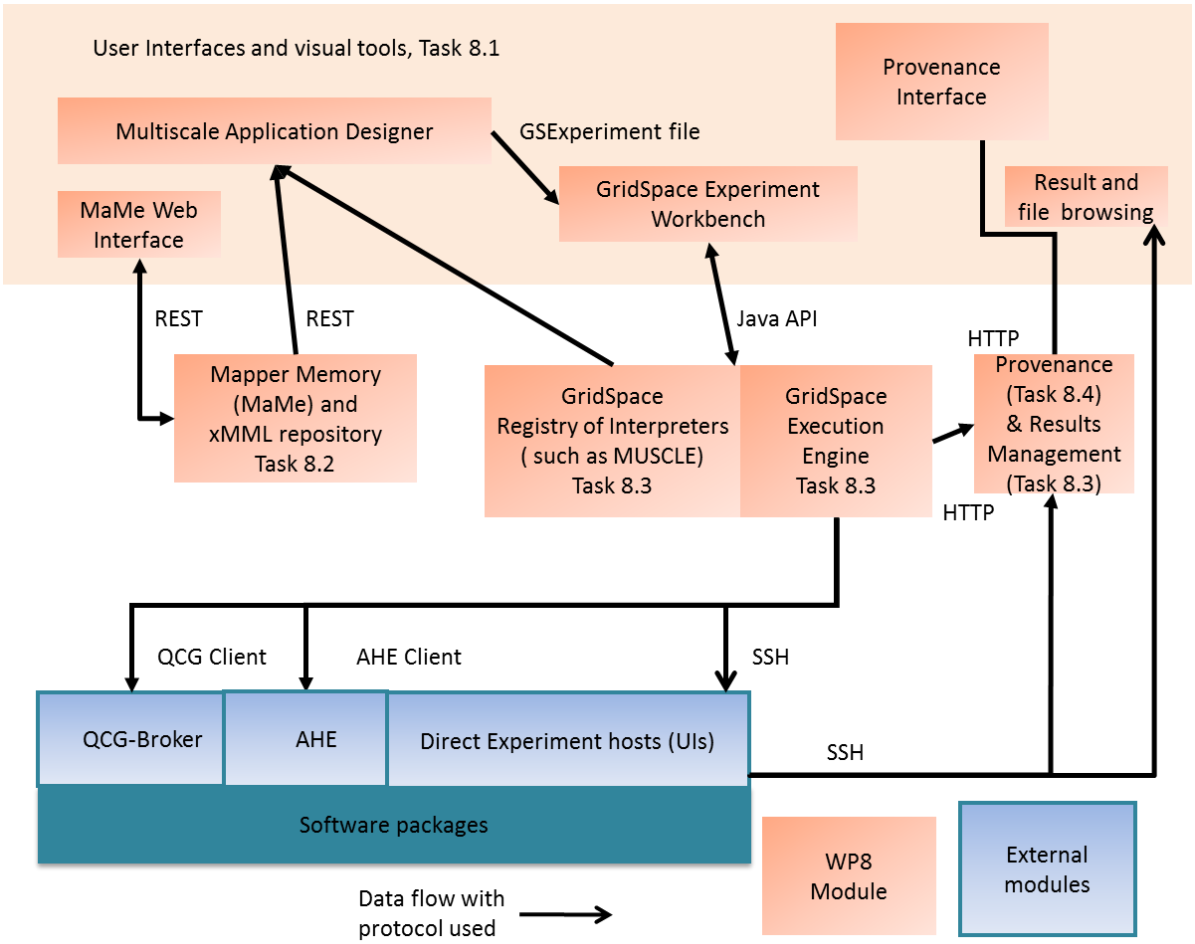

Fig. 1 Architecture of the programming and execution tools.

5 Final Report on the tools

5.1 User Interfaces and visual tools

5.1.1 Multiscale Modeling Language

The goal of MML [Borgdorff_a] is to bridge the gap between multiscale modelers and tool developers, and execution software. MML expresses a high-level specification of a multiscale model in terms of well defined elements such as single-scale models (called submodels), connections (conduits) and mappers transforming data between submodels. Due to the constraints that MML imposes, this specification is sufficient for runtime environments to execute a model and run a simulation. On the other hand, MML does not dictate which runtime environment to use, and an MML specification only describes submodels in a general way rather than forcing a particular implementation of submodels.

To make MML suitable both for users and software, MML has a graphical format called gMML and an XML format called xMML. Users can compose their model with gMML and discuss its structure. This gMML description is converted into the more specific and machine-readable xMML, which is suitable as an exchange format.

MML assumes a rigorous approach toward multiscale modeling. First, the scales of the phenomena to be modeled must be assessed, using, for instance, a scale separation map (SSM) [Hoekstra]. Next, a decomposition into single scale models called submodules, including their interactions and scale bridging methods should be defined. Submodels have a well-defined flow called the submodel execution loop (SEL) which restricts the points where submodels may interact. With this restriction, there is a limited set of possible coupling templates, based on the temporal scale relation between pairs of submodels.

In MML it is possible to change one submodel without affecting others, as long as the coupling remains valid. Each submodel has ports that send or receive at certain points in the submodel execution loop. Ports of different submodels are then coupled with a unidirectional conduit. If the data from one submodel is not in a suitable form to be processed by another submodel, so-called conduit filters or mappers may transform the data.

The jMML library, a Java library that handles MML is described in the Annex (Section 12.1).

5.1.2 Multiscale Application Designer

The Multiscale Application Designer (MAD) is a graphical tool enabling easy multiscale application composition out of individual components described using MML. The tool is available to users through a web browser and facilitates convenient drag-and-drop techniques to build applications. An example application designed in MAD is shown in Fig. 2.

In the left column a list of graphical representations of submodels and mappers (see MML terminology) is available. The items can be dragged onto the workspace in the central part of the view. There, application composition can be performed by connecting ports to produce the final arrangement. The menu on the right contains a list of actions which can be called by the user. These include saving the graph in an internal XML format, exporting it to a GS experiment which can be executed inside the GridSpace EW (see subsection 5.1.3), exporting the graph to an extended xMML format or loading a graph from the extended xMML file.

For the sake of convenience an option is provided to open an experiment directly in GridSpace EW. Finally, separate repository and reservation views are available from the menu. In the bottom part of the main view a property editor is displayed, which enables users to edit module parameters.

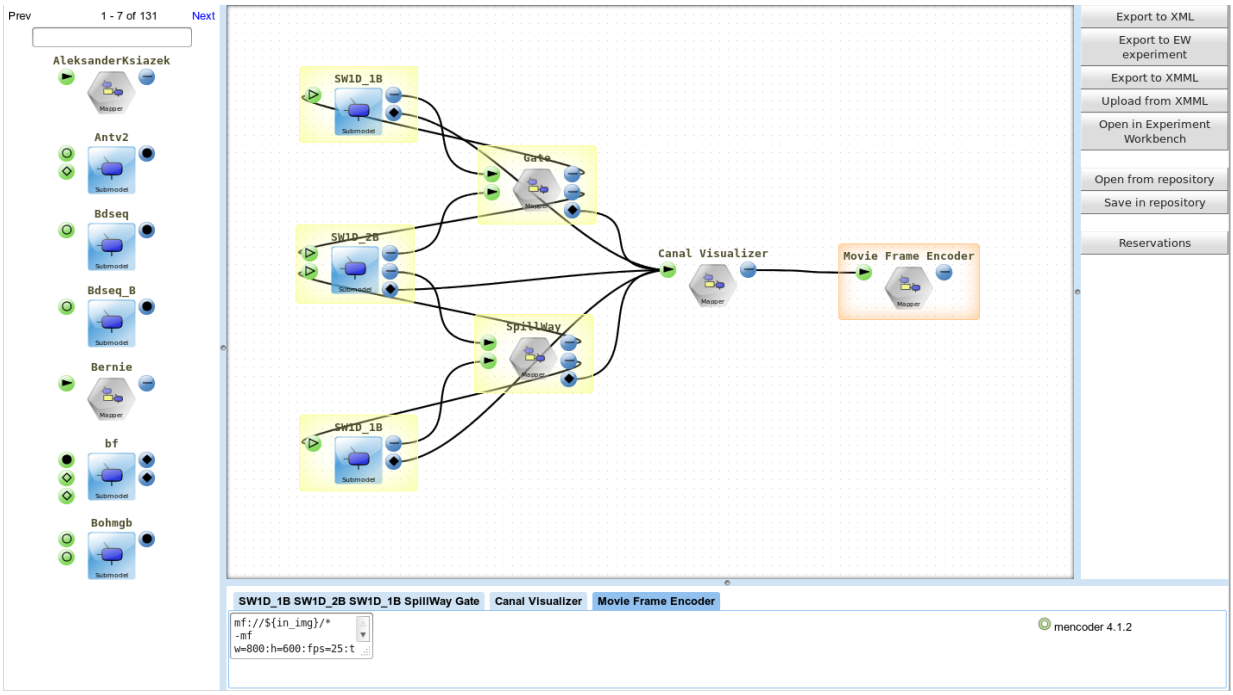

Fig. 2 A snapshot of MAD's main view, showing a sample multiscale application composed from single scale submodules. The application is modeled using MML.

Submodel and mapper items are generated according to the contents of the MaMe registry which is one of the external dependencies of MAD. As there are many modules registered in the repository (over a hundred), a filter box was added which limits the displayed items as the user types in the query. Key features of MAD and its integration with other MAPPER components are described in detail in the Annex (Section 12.2).

5.1.3 GridSpace Experiment Workbench

In order to handle execution of applications, we have chosen the *in silico* experimentation approach embodied by a virtual laboratory platform called GridSpace [Ciepiela]. During the course of the MAPPER project GridSpace was modified and extended according to multiscale applications requirements. The applications executed by GridSpace, called GridSpace experiments, consist of a number of arbitrary executable pieces of code (called snippets) and their input and output files (assets). Each snippet can support its own type of application execution (e.g. a tightly-coupled connection scheme specified in MUSCLE CxA [Borgdorff_c], a Python or Ruby script performing some user-defined calculations, or a script using a specific scientific package such as LAMMPS[3]). As a result the GridSpace experiment concept fits into the requirements of general multiscale application description. An example relation between a high-level MML description and a GridSpace experiment is shown in Fig. 3.

The GridSpace architecture is composed of a web-based UI (the GridSpace Experiment Workbench – EW), and the GridSpace Execution engine backend. GridSpace EW serves experiments and their input and output files through web layer. Experiment parts (i.e. code snippets) are executed by the GridSpace Execution Engine using appropriate software (i.e. interpreters) such as scripting language interpreters or specific software (e.g. LAMMPS, CPMD[4] or MUSCLE). Executors are used to dispatch computational tasks to e-Infrastructures using direct shell access or MAPPER services in the interoperability layer (i.e. QCG-Broker or the Application Hosting Environment). Since the previous release of MAPPER prototypes, thoroughly described in D8.2 and D8.3, considerable effort has been directed towards further development of the GridSpace platform in order to accommodate requirements of multiscale applications both from the MAPPER portfolio and external sources. In particular, the GridSpace Experiment Workbench (EW) has been refactored and enriched as described in Annex (Section12.3.1).

[3] http://***lammps***.*sandia.gov/*

[4] http://www.cpmd.org/

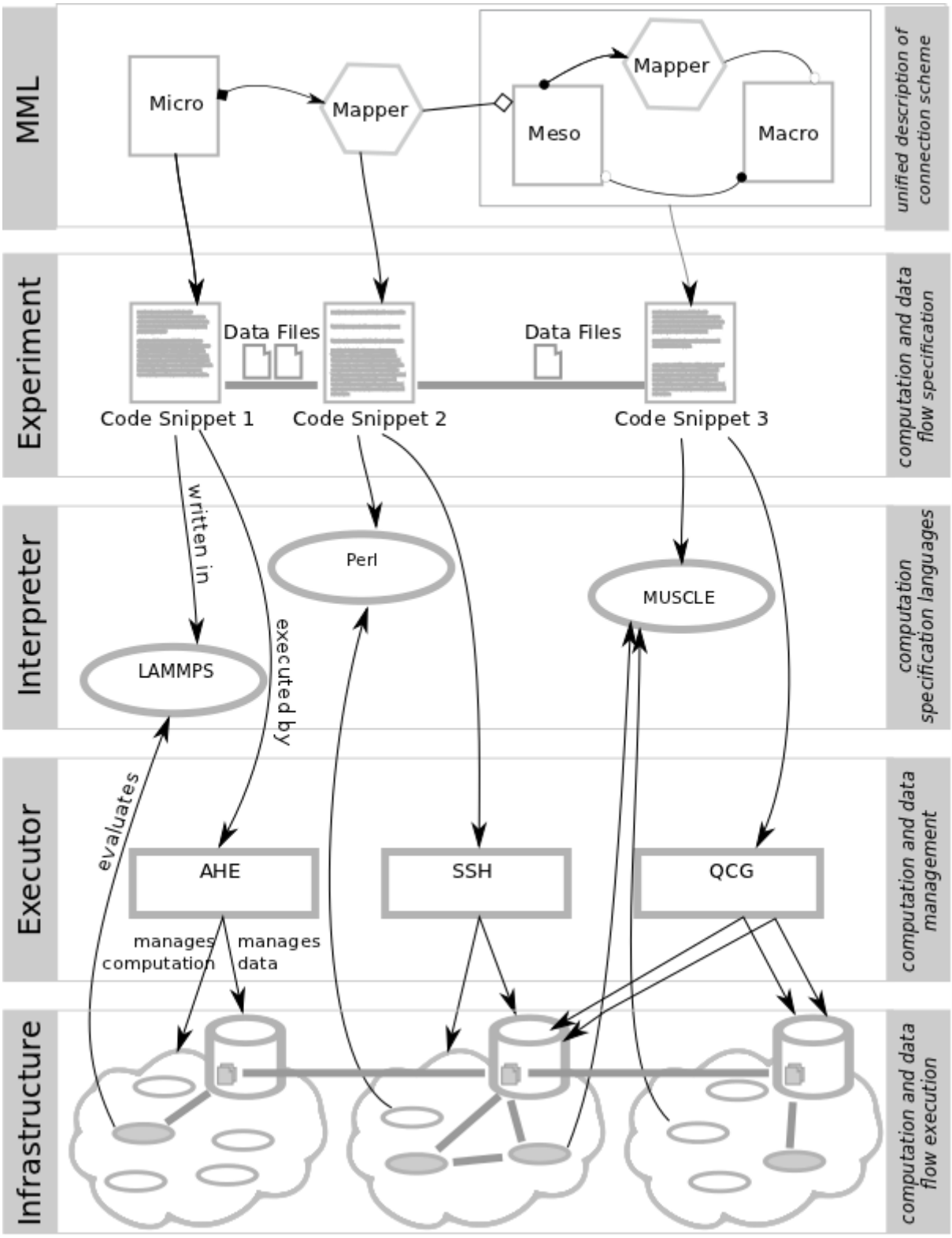

Fig. 3 Relations between MML description, GridSpace experiment and execution environment. The high-level MML description is translated into an experiment that can later be executed on specific infrastructures using MAPPER Services (AHE, QCG) [Rycerz_b]

5.2 Programming tools

5.2.1 MAPPER MEmory and MML repository

The main task of MAPPER Memory (MaMe) is to provide a rich, semantics-aware persistence store for other components to *record* and *share* information. The MaMe registry delivers its functionality based on a well-defined domain model which includes all important elements of MAPPER metadata: **scale modules**, **mappers** and **filters**, together with their **ports**, **implementations** and other constitutive elements and attributes. Thanks to MaMe other MAPPER tools are able store, publish and share a common registry of such elements throughout the entire installation. MaMe is based on the idea of Semantic Integration [Gubala], designed and developed at ACC Cyfronet AGH.

MaMe consists of two main building blocks, each performing a distinct role. These are *MAPPER Modules Registry* and *MAPPER xMML Repository*. The primary purpose of the module metadata registry is to persistently store and publish descriptions of scale models, mappers and filters developed for MAPPER applications. The main objective of the MAPPER xMML Repository is to provide MAPPER MAD users with shareable storage for constructed multiscale application descriptions in the form of an xMML schema. A detailed description of both tools can be found in Annex (Section 12.4).

5.3 Execution Tools

5.3.1 GridSpace Execution Engine

The GridSpace Execution Engine constitutes a backend facility for submission of experiments developed and run through the GridSpace Experiment Workbench (EW) previously described in Section 5.1.3. Detailed features of the GridSpace Execution Engine are described in the Annex (Section 12.3.2).

To achieve effective integration with MAPPER external services GridSpace introduces a level of abstraction above access to computational facilities, i.e. SSH-accessible clusters, QCG brokers, AHE-accessible resources and other infrastructures not known in advance, but potentially required within or beyond the scope of MAPPER. According to that abstraction, GridSpace experiment snippets are associated with interpreters, regardless of where (and whether) they are installed. The interpreters are then deployed on a set of computational resources, binding them to executors (see Fig. 3). In the MAPPER project we have developed three executors: the SSH executor (for integration with basic SSH-accssible resources), the QCG executor (for integration with QCG Broker) and the AHE executor (for

integration with AHE). Details on these executors can be found in the Annex (section 12.3.2). Details on GridSpace-AHE and GridSpace-QCG integration can also be found in deliverables related to service activities (i.e. D4.2 and D5.2).

5.3.2 Result Management

In MAPPER, result management involves three aspects:

- storing, reading and browsing the file created as the result of experiment execution,
- storing metadata that describes experiment output,
- browsing results and searching with the associated metadata.

Physical file handling is provided by a file browser in GridSpace, enabling access to files that are present at a location related to the specific experiment executor using SSH and GridFTP. A provenance tool aids the user in this process. It stores copies of experiment input and output files, and versioning them. This way experiment results are persisted even if they are changed or deleted on the original machine. Copying is disabled for files that are too large to be transferred over the network. Two protocols are used for transferring files:

- SSH – in this case the GridSpace server works as an intermediary between the executor storage and versioning storage. This is required since establishing a direct SSH connection between machines via a third party is usually not possible
- GridFTP – if the machine that runs the experiment supports this protocol, GridSpace uses the third-party-copy feature and files are copied directly between the source and target machines.

Besides file metadata, the provenance system collects detailed information on each GridSpace experiment execution, including what results (files) were created, when and by which execution steps (snippets). It also provides links to result copies, facilitating easy access and browsing. More details are available in subsection 5.4.

Basic browsing features are provided by file browser of GridSpace. GridSpace enables users to browse the directory tree present on the infrastructure that runs the experiment. Additionally, GridSpace allows to monitor standard output of the running experiments on-line. The more sophisticated browser is QUATRO, provided for provenance. It is a web-based application that allows users to construct queries for the RDF database containing provenance data. The results of such queries may include copies of files produced by an experiment. As a result, the provenance browser can be used as an experiment result browser.

5.4 Provenance

As described in subsection 5.3.2, the MAPPER provenance system provides the functionality of tracking, storing, browsing and querying metadata about execution of multiscale application.

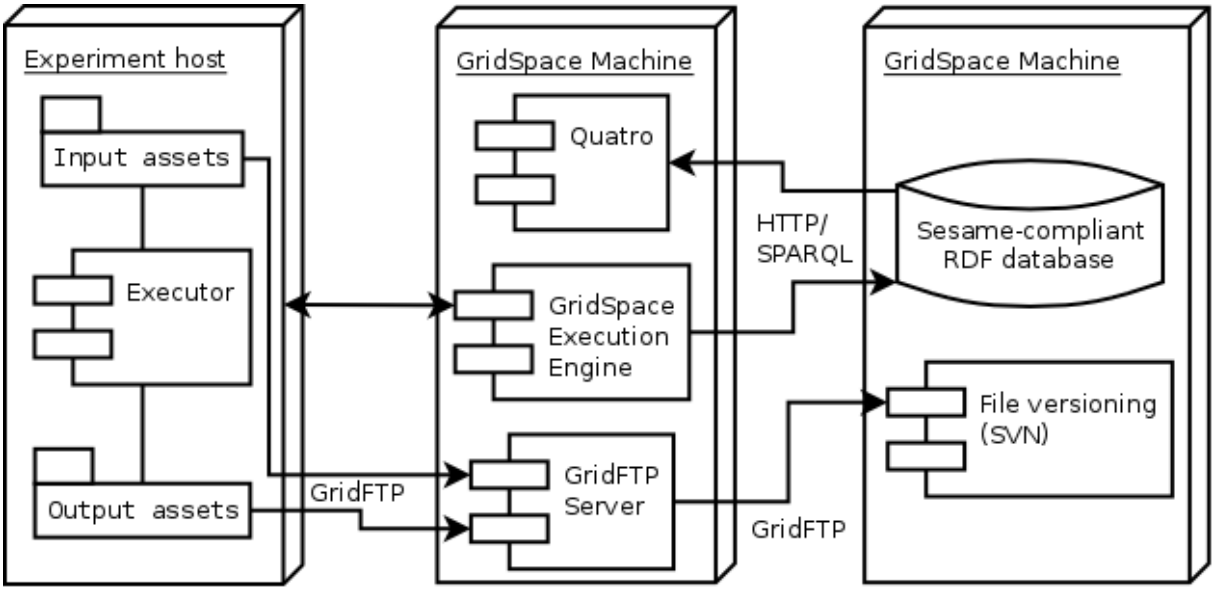

Fig. 4 Architecture of the provenance system, including relations between an experiment executed on an experiment host, the GridSpace execution engine, the provenance database and the provenance interface (QUATRO).

The architecture of the provenance system is shown in Fig. 4. The Experiment Host is a host that runs the experiment using input and output files (input and output assets). The GridSpace Machine hosts the GridSpace Execution Engine described earlier on in this deliverable, together with a GridFTP server to enact file transfers from the experiment host, required to store and register experiment results. The GridSpace Execution Engine supplies event information to the provenance database. Provenance data can be browsed by QUaTRO (provenance interface). DataStore keeps provenance-related data and snapshots of experiment input and output files. Details of the provenance data format are described in the Annex (Section 12.7).

6 Prototype availability

In general, the user and administrator manuals and movies demonstrating WP8 tools can be found at http://dice.cyfronet.pl/projects/details/Mapper.
Tutorials are available at:

- http://www.mapper-project.eu/web/guest/84
- http://www.mapper-project.eu/web/guest/mad-mame-ew

Below we present a detailed list of links for each tool.

6.1 MML and jMML library

The MML language is described in [Borgdorff_a].
The xMML specification is available on http://github.com/blootsvoets/xmml as an open source project.
The jMML library is available on http://github.com/blootsvoets/jmml as an open source project.

6.2 MAD

MAD is available as a production service under the following URL:
https://gs2.mapper-project.eu/mad
User and administration manuals are available under the following URLs:

- http://dice.cyfronet.pl/projects/details/Mapper-files/MAD_User_Manual
- http://dice.cyfronet.pl/projects/details/Mapper-files/MAD_Administrator_Manual

The code is open source and can be retrieved from the following SVN repository:
https://gforge.cyfronet.pl/svn/gs2-utils/ibuilder
Anonymous repository access is possible by using anonsvn as login and anonsvn as password.

6.3 MAPPER Memory and MML repository

MAPPER Memory (MaMe) registry is available at: http://gs2.mapper-project.eu/mame. The source code of the prototype is available at:

- https://gforge.cyfronet.pl/svn/sint/trunk/mame (the MaMe tool)
- https://gforge.cyfronet.pl/svn/sint/trunk/sintmodel_mapper (the semantic MAPPER data model)

and is open for read-only access to any interested party (SVN client software is needed to check out the project).

A detailed user manual that explains all the functions of MaMe, is available at:
http://dice.cyfronet.pl/projects/details/Mapper-files/MaMe_User_Manual
For users interested in the internal mechanisms of MaMe and/or wishing to perform an independent installation of MaMe, the administrator's manual, which explains these matters, is provided at:
http://dice.cyfronet.pl/projects/details/Mapper-files/MaMe_Administrator_Manual

6.4 GridSpace Experiment Tools

An installation of the Experiment Workbench is continuously available at https://gs2.mapper-project.eu/ew/ in its the most recent stable version. The associated interpreter registry is accessible for external tools (such as MAD) through a REST endpoint at https://gs2.mapper-project.eu/ew/gridspace.
Experiment Workbench subpages include comprehensive tutorials and user manuals. Sample use cases for GridSpace are provided at http://dice.cyfronet.pl/projects/details/Mapper
The source code of GridSpace (including the Experiment Workbench and Execution Environment) is maintained as a git repository at http://dev.cyfronet.pl/gitlab/gs/gridspace.git

6.5 Provenance and Results Management

QUaTRO2 – the tool used for browsing provenance data and experiment results is available at the following address:
https://gs2.cyfronet.pl/quatro/
A movie presenting how it can be used is available at:
http://dice.cyfronet.pl/projects/details/Mapper
Source code is available from the following locations:
http://dev.cyfronet.pl/gitlab/gs/gridspace.git – source code of provenance embedded in GridSpace, the module is located in the "provenance" directory;
https://gforge.cyfronet.pl/svn/quatro2/trunk/ – most recent version of the QUaTRO browser.

7 Integration and validation methodology

Integration of MAPPER components is complex and comprises of many layers, from abstract application composition to execution. Additionally, the infrastructures and overlaying middleware used to run MAPPER applications often are subject to updates and configuration changes which may degrade the stability of the MAPPER software stack or make its current configuration obsolete. In order to discover such inconsistencies a testing methodology was introduced which, in an automated fashion, checks whether the production setup of all the tools works fine and, in case the tests do not pass, informs the support team straight away. The main goal of the methodology is to perform integration tests of tools on the production platform.

The testing procedure assumes that a set of test cases is available and includes a sample application producing results which can be tested for validity. A continuous integration tool such as Jenkins (http://jenkins-ci.org) periodically runs the applications and checks whether valid results are produced. The process is fully automated and a notification is sent only when the tests fail (this can be configured independently). Individual test consists of the following steps:

- xMML structure validation – given application sample stored as an xMML file is checked against the current production xMML schema,
- GS experiment structure – the xMML application sample is transformed into a GS experiment and its structure is validated,
- experiment execution – the resulting experiment is executed by providing a predefined set of parameters which should produce valid results.

The transformation and execution are run on the production instances of MAPPER tools. To make this possible, each tool exposes a dedicated REST interface to perform testing tasks. This methodology has proven useful and several problems were quickly resolved in this manner (e.g. computational site downtime, module reorganization on one of the sites, certificate expiration, etc.)

8 Tools integration with external modules

8.1 Integration with MAPPER services

Below, we summarize MAPPER services integrated into GridSpace by executor mechanism. The detailed integration was described in D 4.2 and D 5.2. The detailed description of GridSpace AHE and QCG executors can be found in subsection 12.3.2 of this document.

executor	Description and availability
Application Hosting Environment (AHE)	The Application Hosting Environment, AHE, provides interfaces to run applications on resources provided by national and international grids (UNICORE, Globus and QCG grids) in addition to local departmental and institutional clusters meaning that a user can use a single AHE installation to access resources for example from the UK NGI, Polish NGI and PRACE. More details about AHE can be found in D 4.2 and D 5.2 AHE is available at http://www.mapper-project.eu/web/guest/ahe
QCG-Broker (QCG)	QCG-Broker is a meta-scheduling system that manages the whole process of remote job submission and advance reservation to various batch queueing systems and subsequently to underlying clusters and computational resources. QCG-Broker deals with various meta-scheduling challenges e.g., co-allocation, load-balancing among clusters, remote job control, file staging support or job migration. More details about QCG can be found in D 4.2 and D 5.2 QCG is available on http://www.qoscosgrid.org/trac/qcg-broker

8.2 Integration with scientific software

Below we summarize the most important software required by multiscale application integrated into GridSpace by interpreter mechanism in the MAPPER project

Interpreter	**Used by application**	**description**
The Multiscale Coupling Library and Environment (MUSCLE)	In-stent restenosis, Irrigation Canals, Gene Regulatory Networks	a portable framework to do multiscale modeling and simulation on distributed computing resources available at http://apps.man.poznan.pl/trac/muscle

CPMD	Nanomaterial	a parallelized implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics. Available at http://www.cpmd.org/
CPMD2CUBE	Nanomaterial	A companion program to CPMD and is used to convert CPMD produced data to format understood by other packages (e.g. GAUSSIAN).
LAMMPS	Nanomaterial	Large-scale Atomic/Molecular Massively Parallel Simulator - available on http://lammps.sandia.gov/
MSI2LMP_POT	Nanomaterial	Post processing tool of LAMMPS
Canal Visualizer	Irrigation Canals	Visualization tool for irrigation canals simulation
Mencoder	Irrigation Canals, Fusion	General video decoding, encoding and filtering tool used in MAPPER for visualisation of simulation results
Helena	Fusion	Plasma equilibrium codeused in fusion modelling application
Ilsa	Fusion	Plasma equilibrium codeused in fusion modelling application
Abaqus	CrushBox	software for finite element analysis and computer-aided engineering

9 Evaluation of tool efficiency

According to the description of Task 8.5 in the DoW we measure the efficiency of the WP8 tools by comparing the effort of coupling applications by hand with the effort of coupling them using the tools. The tools support the process of creation of multiscale applications by enabling users to share created modules along with descriptions of their connections, and reusing modules in different configurations. The connection scheme can be created by the user on conceptual level using visual tools. Having stored all required data, it is possible to automate production of an executable from the resulting connection schema. Actual execution is supported by the ability to choose required resources, then start and monitor execution from a single web-based entry point. Following execution, the output data should be fetched and presented to the user. In Tab. 1, we describe the actions that can be facilitated by WP8 tools.

Tab. 1 Actions involved in developing and execution of multiscale applications that can be facilitated by WP8 tools.

Action	Tool Support
description of single-scale models	for regular applications the description is registered in MaMe by using an interactive user interface
design and implementation of single-scale modules implementing models	can be done using specific frameworks such as MUSCLE
design of connection schema among singe-scale modules	interactive visual design in MAD; the previously designed connection schemes can also be loaded from an xMML repository; automatic generation of connection schema
preparation of executable application from the connection schema	automatic generation of experiment by MAD (several milliseconds) – this assumes that implementations of single-scale modules are already available
mapping modules to (possibly different) external services that access e-infrastructures; setting parameters of these services	done by the user via a unified web interface (GridSpace EW); interactive process

execution of modules	initialization of execution is done from a unified web interface by pressing the run button;
monitoring standard output and error	for SSH-accessible resources, standard error and output can be monitored in the GridSpace EW, enabling users to interact with running modules
using interoperability layer	convenient usage of features of underlying services (QCG-Broker, AHE)
fetching results	automatically fetched and visible in GridSpace EW by means of standard protocols – SSH and GridFTP
viewing results	visible in GridSpace EW; additionally the QUaTRO browser can search for copies of experiment results
viewing provenance results	QUaTRO browsing and searching provenance data

To evaluate the efficiency of tools, we have used the following metrics:

User experience with new MAPPER tools measured by: feedback forms. During the second seasonal MAPPER school http://www.mapper-project.eu/web/guest/second-seasonal-school we have measured the usability of MaMe, MAD and GridSpace Experiment Workbench tools based on [Brooke]. The obtained average SUS score for the tools was 70 points (out of 100 possible) . This average was based on answers provided by the most active 15 participants.

Mean time required to train a new user to use MAPPER tools measured during Seasonal Schools in Task 2.4. The tutorial conducted during the second MAPPER seasonal school was successful and consisted of a 30-minute presentation and 60 minutes of hands-on exercises available at http://www.mapper-project.eu/web/guest/84. There were no major problems in using the tools by school participants according to their responses in the SUS survey; e.g. for the question “I think the system was easy to use” the answers were: 33% – fully agree (5/5 possible points), 47% – agree (4/5 points), 20% OK (3/5 points). There were no answers lower than 3 points

Statistics of successful execution of complete multi-scale simulations measured on the basis of data obtained by provenance services created in Task 8.4; We have used

the provenance system to track execution of individual simulation stages. Overall, users completed 2326 snippet runs which can be interpreted as executions of a single simulation stage. Additionally, we can count other actions performed by users indicative of tool usage:

- the GridSpace login operation was performed successfully 1563 times,
- users converted their MAD MML designs to GridSpace experiments 232 times,
- GridSpace experiments were created (including those based on MML) 553 times.

All xMML of applications are stored in the xMML repository and are available through MAD at https://gs2.mapper-project.eu/mad (press the "open from repository" button). A summary of the usage of tools by applications is described in Tab. 2. A detailed report on the applications can be found in D7.3 and their integration is described in D5.2.

Tab. 2 Using WP8 tools in MAPPER applications

Application name	Field	Status of using WP8 tools
In-stent restenosis	physiology	modules registered in MaMe, connected in MAD, executed in EW using SSH and QCG executors; successfully applied and described in [Borgdorffb]
Irrigation canals	hydrology	modules registered in MaMe, connected in MAD, executed in EW using SSH and QCG executors; successfully applied in [Belgacem_a, Belgacem_b], used as a basis for the tutorial at the first seasonal MAPPER school in London http://www.mapper-project.eu/web/guest/mad-mame-ew
clay-polymer nanocomposites	Nano-material science	modules registered in MaMe, connected in MAD, executed in EW by SSH executor and AHE executor successfully applied and described in [Rycerz_a]
reverse engineering of gene-regulatory networks	Computational Biology)	modules registered in MaMe, connected in MAD, executed in EW by SSH and QCG executors

equilibrium-stability	fusion	modules registered in MaMe, connected in MAD, executed in EW; parameter study achieved using subsnippets mechanism; Tutorial example shown during the second seasonal MAPPER school in Barcelona http://www.mapper-project.eu/web/guest/second-seasonal-school
transport turbulence equilibrium workflows	fusion	modules registered in MaMe, connected in MAD, executed in EW
Heme LB	physiology	Modules registered in MaMe and connected in MAD
Crushbox (external)	metallurgy	Executed in EW; parameters study achieved using the subsnippet mechanism;

The number of single-scale models incorporated and used within the MAPPER infrastructure, measured by taking information from the models' registry developed in Task 8.2, is as follows: 45 single-scale models, 42 mappers and two filters are already registered in the model registry (MaMe), representing almost all MAPPER applications, in addition to a test application. This number can be monitored online at `http://gs2.mapper-project.eu/mame`

Number of new scientific results from applications created by MAPPER tools measured by number of publications in well recognized journals/conferences; the results were published in the following publications:

- using WP8 tools with the In-Stent Restenosis Application [Borgdorff_b]
- using WP8 tools with the Clay-Polymer Nanocomposites application [Rycerz_a, Groen_b, Suter]
- using WP8 tools with the Irrigation Canals application [Belgacem_a, Belgacem_b]
- paper on WP8 tools [Rycerz_b]
- A demo on how to use WP8 tools using the fusion loosely-coupled equilibrium-stability application example, was shown at the 2nd seasonal school.

- Results of using WP8 tools for an external metallurgical application were presented as a poster ("Efficient Execution of Grid-based Multiscale Applications for Metallurgical Industry") by B. Wilk et al. at the Summer School on Grid and Cloud Workflows and Gateways, 1-6 July 2013, Budapest, Hungary.

Statistics from monitoring MAPPER web pages

All statistics presented below have been calculated for the 1 January 2013 – 30 June 2013 period using Google Analytics.

Definitions

Page Views – total number of pages viewed. Repeated views of a single page are counted.

Unique Page Views – number of visits during which the specified page was viewed at least once. A unique page view is counted for each page URL + page title combination.

Visit – a group of interactions that take place on the website within a given timeframe. For example, a single visit can contain multiple page views, events, social interactions, custom variables and e-commerce transactions.

Pages/Visit (Average Page Depth) – the average number of pages viewed during a visit to the website. Repeated views of a single page are counted.

Avg. Visit Duration – the average duration of a session.

Tool Statistics

As the MaMe, MAD and GridSpace EW are all web-based tools, it is possible to monitor their usage as web pages. All tools are accessible from a single entry point, i.e. the `http://gs2.mapper-project.eu` web site. Below, we present statistics (pageviews and visits) for that site as a whole, and for each tool separately.

Pageviews

Tab. 3 Number of page views for the http://gs2.mapper-project.eu web site

Page Views	Unique Page Views
1976	990

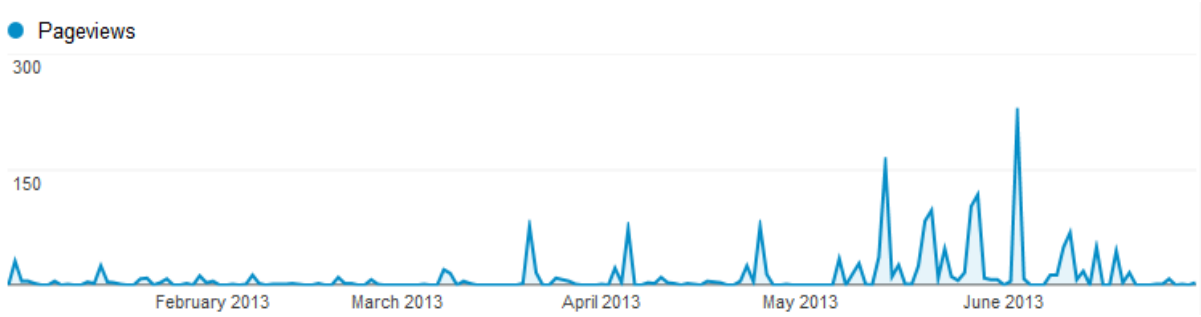

Fig. 5 Number of page views for the http://gs2.mapper-project.eu web site (i.e. MAD, MaMe and GridSpace EW in total)

Tab. 4 Number of page views for MAD, MaMe and GridSpace EW tools separately.

Tool name	Page Views	Unique Page Views
MAD	1503	700
MaMe	1000	435
GridSpace EW	1392	633

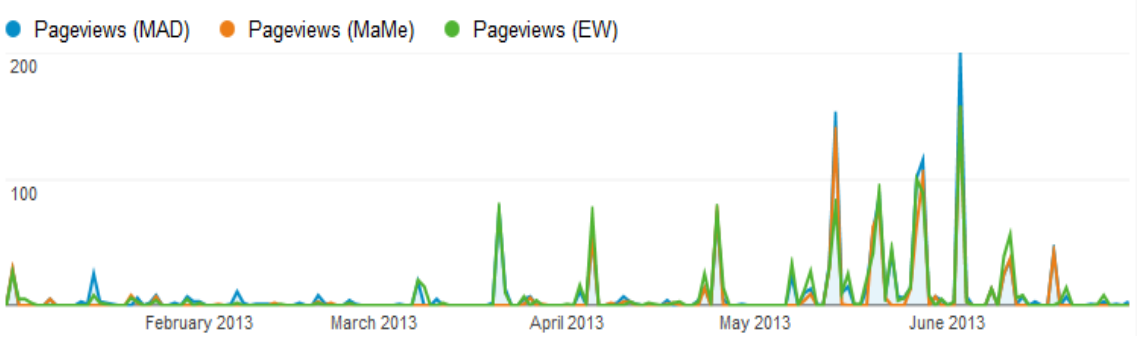

Fig. 6 Number of page views for MAD, MaMe and GridSpace EW tools.

Visits

Tab. 5 Number of visits for the http://gs2.mapper-project.eu web site

Visits	Pages/Visit	Avg. Visit Duration
437	4.52	00:09:56

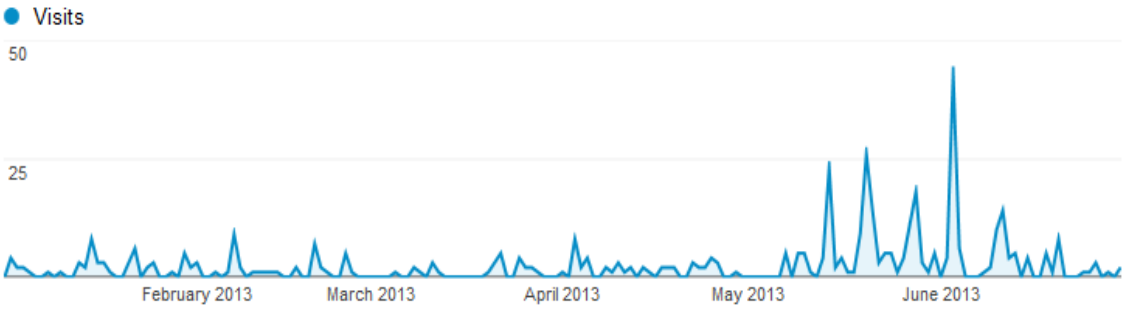

Fig. 7 Number of visits for the http://gs2.mapper-project.eu web site (i.e. MAD, MaMe and GridSpace EW in total)

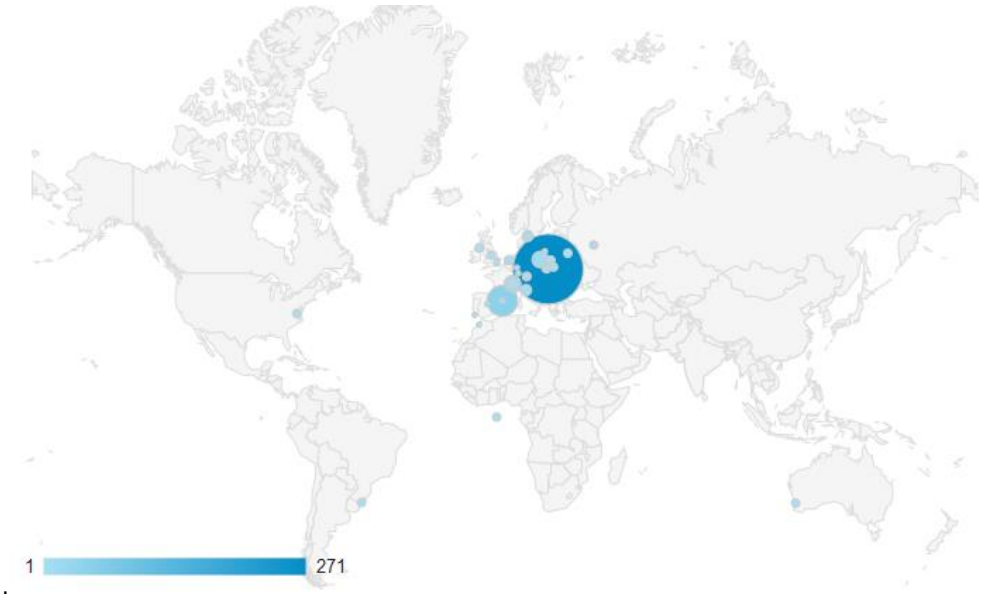

Fig. 8 Location of visitors (cities) for the http://gs2.mapper-project.eu website.

Tab. 6 Number of visits for the MAD, MaMe and GridSpace EW tools separately.

Tool name	Visits	Pages/Visit	Avg. Visit Duration
MAD	274	5.49	00:13:10
MaMe	112	8.93	00:19:02
GridSpace EW	212	6.57	00:15:06

Fig. 9 Number of visits for MAD, MaMe and GridSpace EW separately

Statistics from monitoring manuals web pages

Below, we present statistics for tool manuals available from the `http://dice.cyfronet.pl/projects/details/Mapper` website.

Page Views

Tab. 7 Number of page views for tool manuals

Manual name	Page Views	Unique Page Views
All	**160**	**107**
MAD User Manual	39	31
MAD Administrator Manual	30	18
MaMe User Manual	33	26
MaMe Administrator Manual	30	18
GridSpace2 Experiment Workbench – Administrator Manual	28	14

Visits

Tab. 8 Number of visits for tools manuals

Manual name	Visits	Pages/Visit	Avg. Visit Duration
MAD User Manual	13	24.69	00:22:27
MAD Administrator Manual	14	27.14	00:23:07

MaMe User Manual	26	12.38	00:10:36
MaMe Administrator Manual	11	25.36	00:22:29
GridSpace2 Experiment Workbench – Administrator Manual	13	26.77	00:26:42

Fig. 10 Location of visitors (cities) for the tool manuals available at http://dice.cyfronet.pl/projects/details/Mapper website.

Download statistics of the tool packages

Below we present download statistics of packages available at `http://dice.cyfronet.pl/projects/details/Mapper`. Our tools are easily accessible via a web browser, at `http://gs2.mapper-project.eu`, so users do not need to install them on their own computers. This explains the disparity between the total number of user visits on the tools' pages (presented in the previous section), and the number of direct tool downloads (shown in the table below).

GridSpace2, MAD and MaMe are all provisioned in the SaaS model and most users use these tools online, through the `http://gs2.mapper-project.eu` website.

Tab. 9 Download statistics for the tool packages.

Tool name	Number of downloads
ibuilder-gwt-0.2.0.war	8
ew-2.5.14.war	6
sint_upper_model-0.1.8.gem	1
sintmodel_mapper-0.1.5.gem	1
All	**16**

10 Summary and Conclusions

This deliverable presents the final prototype of multiscale programming and execution tools, their architecture, functionality and implementation according to the design presented in D8.1., along with changes with respect to the second prototype described in D8.3.

The presented tools are used to validate the proposed approach to unified description and execution of multiscale simulations on e-infrastructures. To achieve this goal we had to find solutions to issues such as multiscale model languages, resource-independent descriptions of applications, transformation of the former into the latter and, finally, execution of the resulting realization in a distributed environment. These problems were addressed via appropriate choice of the modeling language, its transformation into an executable experiment (independent of the actual realization of the application), and tools which enable distributed execution of said experiment.

The presented approach allows for composability, reusability and sharing of multiscale simulations, and is implemented in the framework of the presented tools. The steps required to create such an application include: modeling of single-scale phenomena, describing models in MML and designing the corresponding software module. Subsequently, by using descriptions, individual modules can be connected to form a multiscale application. The connections used by the application can be stored for reuse. The connection schema is translated into an executable and infrastructure-independent multiscale *in silico* experiment which can be executed by the user after providing information about required resources – in fact, every module can be executed by a different part of the distributed infrastructure. Access can be either direct (using SSH) or indirect, by external services such as AHE or QCG that provide additional features, including unified access to resources, reservations and co-allocation. The introduced environment facilitates (or even automates) most of the above mentioned steps.

In conclusion, we believe that the proposed method and environment have proven to be very successful in enabling scientists to build multiscale applications. The use of unified descriptions facilitates development of different applications from a single set of modules and switching between different versions of modules offering specific features. It can also be applied to validate the created simulations against different realizations. The accessibility of web-based tools enables applications to be shared among scientists working in the same area. Additionally, support for interactivity (i.e. user interaction during application execution) is provided.

Efficiency evaluation results (described in Section 8) show that the proposed approach is successful and that it can be used for multiscale applications in various research fields.

11 References

[Alder] Alder, G.: Design and implementation of the JGraph swing component, Technical Report, 2001

[Ambrozinski] M. Ambrozinski, K. Bzowski, L. Rauch, M. Pietrzyk: Application of statistically similar representative volume element in numerical simulations of crash box stamping, Archives of Civil and Mechanical Engineering, Volume 12, Issue 2, June 2012, Pages 126–132

[Belgacem_a] M. Ben Belgacem, B. Chopard, J. Borgdorff, M. Mamonski, K. Rycerz, D. Harezlak: Distributed Multiscale Computations using the MAPPER framework Procedia Computer Science, Volume 18, 2013, Pages 1106-1115

[Belgacem_b] M. Ben Belgacem, B. Chopard and A. Parmigiani: "Coupling method for building a network of irrigation canals on distributed computing environment" to be published in Proceedings of 10th International Conference on Cellular Automata for Research and Industry, ACRI 2012, Santorini Island, Greece, September 24-27, 2012. Series: Lecture Notes in Computer Science, Vol. 7495

[Borgdorff_a] J. Borgdorff, J.-L. Falcone, E. Lorenz, C. Bona-Casas, B. Chopard, and A. G. Hoekstra: "Foundations of distributed multiscale computing: Formalization, specification, and analysis," Journal of Parallel and Distributed Computing, vol. 73, pp. 465–483, 2013.

[Borgdorff_b] J. Borgdorff, C. Bona-Casas, M. Mamonski, K. Kurowski, T. Piontek, B. Bosak, K. Rycerz, E. Ciepiela, T. Gubala, D. Harezlak, M. Bubak, E. Lorenz, A. G. Hoekstra: A Distributed Multiscale Computation of a Tightly Coupled Model Using the Multiscale Modeling Language. Procedia CS 9: 596-605 (2012)

[Borgdorff_c] J. Borgdorff, M. Mamonski, B. Bosak, D. Groen, M. Ben Belgacem, K. Kurowski, A. G. Hoekstra: Multiscale Computing with the Multiscale Modeling Library and Runtime Environment Computer Science, Volume 18, 2013, Pages 1097–1105

[Brooke] John Brooke Usability evaluation in industry, SUS - a quick and dirty usability scale (CRC Press, Boca Raton, FL), pp 189–194 (1996)

[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 (October 2010),

[Groen_a] D. Groen, J. Borgdorff, C. Bona-Casas, J. Hetherington, R. W. Nash, S. J. Zasada, I. Saverchenko, M. Mamonski, K. Kurowski, M. O. Bernabeu, A. G. Hoekstra and P. V. Coveney: fidelity multiscale biomedical simulations, Interface Focus 2013 3, 20120087, published 21 February 2013

[Groen_b] D. Groen, J. Borgdorff, S. Zasada, C. Bona-Casas, J. Hetherington, R. Nash, A. Hoekstra, P. Coveney: A Distributed Infrastructure for Multiscale Biomedical Simulations, accepted by the Virtual Physiological Human Conference 2012

[Hoekstra] Hoekstra, A.G., Lorenz, E., Falcone, J.-L., Chopard, B. Toward a Complex Automata Formalism for MultiScale Modeling, in: International Journal for Multiscale Computational Engineering 5, 6, 491--502.

[Gubala] T. Gubala, M. Bubak, P. M. A. Sloot: Semantic Integration of Collaborative Research Environments. In: M. Cannataro (ed.) Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare, Chapter 26, pp. 514-530, Information Science Reference, IGI Global (2009)

[Rycerz_a] K. Rycerz, E. Ciepiela, G. Dyk, D. Groen, T. Gubala, D. Harezlak, M. Pawlik, J. Suter, S. Zasada, P. Coveney, M. Bubak: Support for Multiscale Simulations with Molecular Dynamics Procedia Computer Science, Volume 18, 2013, Pages 1116-1125

[Rycerz_b] K. Rycerz et al.: Composing, Execution and Sharing of Multiscale Applications, submitted to Future Generation Computer Systems, in review

[Mendes] Mendes, P., Sha, W., Ye, K.: Artificial gene networks for objective comparison of analysis algorithms, Bioinformatics 2003, 19(90002): 122-129, 2003.

[Savageau] Savageau, M.A.: Biochemical systems analysis: A study of function and design in molecular biology, Addison-Wesley, Reading, Mass., 1976.

[Suter]J. Suter, D. Groen, L. Kabalan and P. Coveney: Distributed Multiscale Simulations of Clay-Polymer Nanocomposites, Materials Research Symposium, San Francisco, United States of America, April 2012.

[Vohradsky] Vohradský, J.: Neural network model of gene expression. The FASEB Journal 15, 846, 2001.

[Voit] Voit, E.O. & Schwacke, J.H.: Understanding through modeling: A historical perspective and review of biochemical systems theory as a powerful tool for systems biology. Konopka, A.K. (editor), Systems biology: Principles, methods and concepts, pp. 28-77, 2007.

12 Annex: Detailed description of tools

12.1 JMML library

JMML library handles MML. Specifically, it can read and write xMML and it can generate gMML. It is a Maven project with three modules:

- A utilities module *jmml-util* that contains data structures such as lists and graphs, and SI unit handlers.
- A specification module *jmml-specification* that converts from and to xMML using the Java Architecture for XML Binding (JAXB). It has custom classes that facilitate easy manipulation of xMML.
- An API *jmml-api* to create a coupling topology, task graph or scale separation map from a xMML specification. It can also output a task graph or gMML to pdf.

The main part of the jmml-specification is generated from the W3C XML Schema of xMML by the xjc tool of JAXB. This means that when a new version of xMML is released, jMML can easily adapt. It also means that it can always read, modify and write a certain version of xMML. On top of these generated sources, additional verification of couplings and datatypes is performed. This happens at any time that the xMML specification is changed in jMML. Once the code is generated, xMML is no longer validated using the XML Schema when an xMML file is read. Instead, xMML should be validated before it is passed to the jmml-specification module.

The jmml-api module can analyze the xMML specification in several ways. First of all, it can detect which submodels need to be started initially for the model to calculate correctly by constructing the coupling topology. It can output the coupling topology as a Graphviz file, which can then be converted to PDF by the dot tool of Graphviz. The resulting PDF then contains gMML. It can also generate the task graph of a specification, assuming that the time scales of all submodels are regular. If a model will run into a deadlock, given the xMML specification, this will be detected and reported on the command line and, if converted to PDF, in the resulting PDF file. The task graph algorithm is memory-efficient and a task graph can be manipulated after it is generated. Graphviz and PDF viewers, on the other hand, can not always handle graphs that are as large as a full task graph. Finally, it can detect the scales of the different submodels, and whether they are separated or not, and make a scale separation map. The map can be viewed as a window or converted to SVG using the Batik SVG converter.

The jmml-api includes a command-line tool that can generate a coupling topology, task graph, or scale separation map from a given xMML file. The only prerequisite is that such an xMML must have all XInclude files processed in advance.

Anyone that wants to manipulate xMML can import the first two models into their Java project. If they want to generate a task graph or coupling topology the third module can be used.

The jMML library can also output a MUSCLE configuration file for a given xMML file and generate a directory structure with preliminary code and base code already filled in based on the xMML file.

To install jMML, Maven must be installed. Then, download jmml and go the directory. There you can run

```
bin/jmml
```

which will show the options, for example, running

```
bin/jmml path/to/xml.xmml --cxa isr.cxa.rb
```

will generate a MUSCLE configuration file.

12.2 MAD

The following sections contain the description of MAD's main features. Following publication of the second prototype (D8.3), work on integrating a reservation management view was performed which included interfacing with the MAPPER reservation portal and the QCG broker. Additionally, many UI improvements were introduced in the final version.

Transforming gMML to GS experiment

The MAD tool supports exporting MML diagrams both to the xMML format (for describing high-level application structure) and to the GridSpace experiment format. In particular, the experiment can contain CxA snippets for MUSCLE applications that can be executed on various infrastructures (e.g. QCG).
Importantly, prior to exporting a multiscale application as a GridSpace experiment, the MAD tool needs supplementary information from the Interpreter Registry provided by the Experiment Workbench. The registry tells what software can implement particular submodels and which infrastructure elements can run such software. This information is intentionally not included in the MML description in order to keep it independent of software that implements

submodules. The MAD tool inquires MaMe for interpreters that implement given submodules, and then inquires the Interpreter Registry about what parameters and codes are required by the interpreters. Therefore, MAD users can choose the interpreter that suits their needs and provide all parameters and codes it requires. Such combination of information is then sufficient to construct a GridSpace experiment.

While each experiment designates interpreters that are to be used to implement submodules, the selection of infrastructure elements or sites to be used to execute the experiment remains ambiguous and is deferred until the application execution step carried out through the GridSpace Experiment Workbench tool. As a result, the same experiment can be run in many different ways by harnessing different infrastructures.

CxA configuration support

In the process of building the final GS experiment from the gMML representation, the CxA configuration comprises a single code snippet. The snippet is generated by filling in a common template with appropriate sections which include classpath, kernels, parameters and connections. Data required to compose individual sections is retrieved from the MaMe registry. The current template of the CxA configuration is given in

Fig. 11

```
# configuration file for a MUSCLE CxA
abort "this is a configuration file for to be used with the
MUSCLE bootstrap utility" if __FILE__ == $0
# add build for this cxa to system paths (i.e. CLASSPATH)
m = Muscle.LAST
m.add_classpath "${classpath}"
m.add_libpath "/user/lib"
# cxa configuration section
cxa = Cxa.LAST
cxa.env["cxa_path"] = File.dirname(__FILE__)
# declare kernels
${kernels}
# parameters
${parameters}
# configure connection scheme
cs = cxa.cs
${connections}
```

Fig. 11 CxA configuration template filled in by MAD according to the gMML contents.

The CxA classpath section holds the locations of .jar (Java Archive) files containing implementations of the relevant MUSCLE kernels. The locations are specific for particular MAPPER submodules and are provided during the process of registration in the MaMe registry. For version 2 of MUSCLE, this is modified by using cross-infrastructure modules; hence the Java classpath is configured locally on the computation site. This gives more generic control over local site configuration which becomes transparent to the application.
The kernels section defines a list of kernels taking part in the computation. Each submodule and mapper from the gMML description is assigned a kernel. The order in which kernels are defined is irrelevant in the CxA configuration. Each kernel can be configured with an independent list of parameters in the parameters section. MAD takes parameter names and values from the MaMe registry.
Tha final CxA configuration section defines connections between kernels. It is built according to the graphical representation of the application.

Module property editor

The bottom part of the interface is occupied by a property editor (see Fig. 12) which allows users to change properties (including simulation parameters) of MML components. Editor tabs correspond to individual nodes in the workspace area, unless a tightly-coupled section is present in which case a single tab combines the section components' properties. The editor fills in the property forms with default values coming from the MaMe registry. If present, different MML component implementations can be chosen in the property editor. Another feature of the property editor is handling of global parameters which are defined in MaMe in an MML component namespace. This may introduce conflicts if the same global property is imported by a few components. Such cases are discovered and a proper warning is produced. The user may pick a global parameter value of one of the conflicting components or specify a new one. When an application is saved all changes applied in the editor are saved and restored when the application is being imported into MAD.

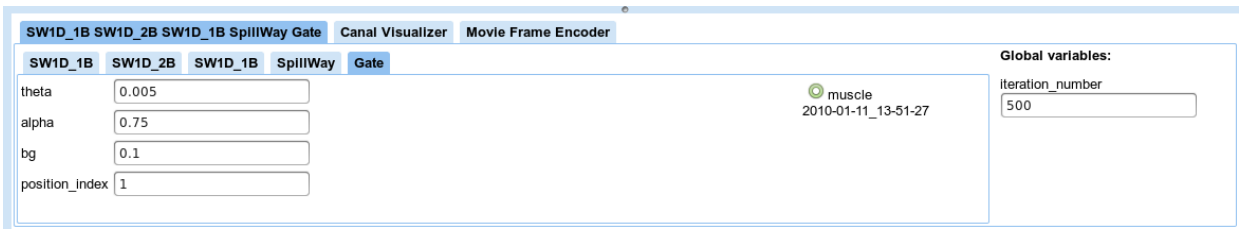

Fig. 12 Property editor allowing for submodule parameter modification

xMML repository

The xMML repository is made available as part of the MaMe registry and allows for storing and retrieving of multiscale applications written in the xMML format. MAD utilizes this by

providing xMML repository widgets (see Fig. 13). This allows users to compose their applications and afterwards save and share them with others. Each stored application is annotated with a name and a short description. If an application with the same name already exists, it is overwritten and placed in the archive for later recovery. Saved applications cannot be removed directly from MAD but it can be done through MaMe by providing valid credentials. To access the repository view the relevant menu items should be used. The presented application list is straightforward and shows all the saved items without any access restrictions.

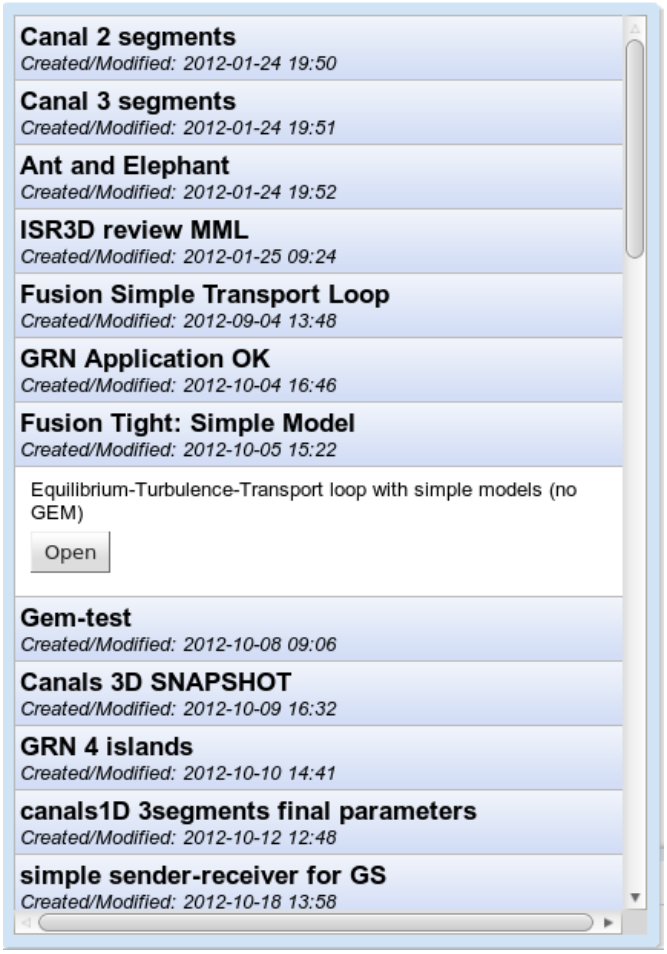

Fig. 13 Application list of the xMML repository

Module management

The first prototype of MAD generated an executable form of tightly-coupled sections by using (among others) low-level properties which were specific for an execution environment (infrastructure site) and were defined in MaMe. This posed interoperability problems on the modeling level. Additionally, the feature of executing particular parts of tightly-coupled sections on different execution sites brought extra difficulties in generating proper mapping descriptions. In the second prototype mapping between parts of tightly-coupled sections and execution sites was moved to the GridSpace Experiment Workbench (EW) which enabled

the modeling layer to operate on abstracted dependencies. This, however, required the EW experiment generation process to be modified. The modification is in place and passes correspondences between MML components belonging to a tightly-coupled section and abstract module names. This enables different mapping strategies to be implemented in EW in order to optimally use the available infrastructure and keep the application descriptions abstract.

Reservation support

One of the prominent functionalities of the MAPPER platform is handling of computational resource reservations. In MAD this is supported when exporting an application to its executable form. Reservations can be mapped to individual submodules comprising the application by using the dedicated user interface shown in Fig. 14.

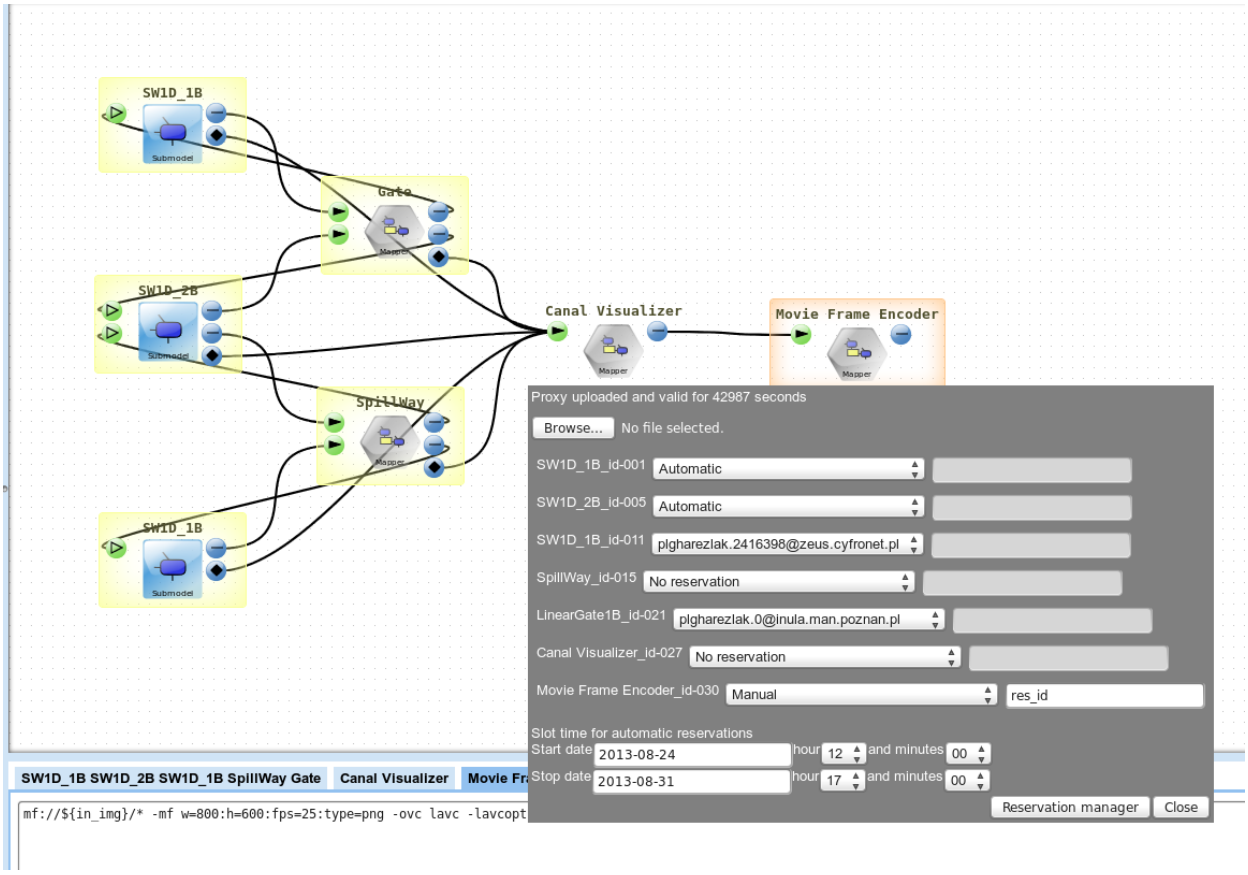

Fig. 14 Mapping among reservations and application submodules

Each submodule can be assigned different types of reservations which are described below:

- no reservation – given submodule is not assigned a reservation,
- automatic – reservations are automatically scheduled within a specified time window,
- manual – reservations are scheduled manually and a corresponding reservation identifier is provided by the user,

- reservation identifier – one of the available reservations is assigned (chosen from a list).

The reservation creation interface provided by the QCG component was embedded into MAD to offer better user experience and allow for managing the reservations from just one tool. Besides GUI integration, MAD also utilizes a QCG client library to obtain the list of available reservations.

12.3 GridSpace

Below we describe all features and improvements that were implemented in GridSpace during the MAPPER project. From the software engineering point of view, the architecture of EW was changed and decomposed into several separate Java projects. The **cyfronet.gs2.ew** module is the EW web user interface, while the remaining modules are server-side libraries which we refer to later as the Execution Engine. In the following subsections we describe changes in both the GridSpace Experiment Workbench and the GridSpace Execution Engine.

- **cyfronet.gs2.experiment** – classes and APIs enabling construction of experiments and serializing them to the XML format through JAXB technology, used both by the Experiment Workbench and MAD that exports the multiscale application description in the experiment format;
- **cyfronet.gs2.executor** – classes and APIs defining the contract between GridSpace and computational facilities;
- **cyfronet.gs2.?-executor** – specific implementation of an executor, e.g. SSH;
- **cyfronet.gs2.core** – engine that carries out experiment execution;
- **cyfronet.gs2.ew** – web interface and server backend that incorporate all the above-listed modules.

12.3.1 Features added to GridSpace Experiment Workbench during the MAPPER project

One-click opening of MAD-generated experiments. The GridSpace Experiment Workbench (EW) can now open experiments generated by external applications, most notably the Multiscale Application Designer (MAD), using a dedicated HTTP endpoint (/workbench/open). Making HTTP POST request to such URLs results in opening the experiment provided as the content of the request. If the user is not yet authenticated, the login form is displayed instead. Following successful authentication, EW is opened with the

experiment that has been passed from the external application. Thanks to this feature, MAD can now offer one-click EW redirections and open the experiment generated on the fly by MAD.

Support for long-running experiments. One of the major deficiencies of GridSpace Experiment Workbench in the first prototype was lack of proper handling of long-running experiments. EW didn't provide any means to run experiments in an unattended way, so the users had to be logged into EW throughout the experiment run time, keeping the session with EW open. In the second prototype, EW handles experiment execution on the server side thus allowing users to log out from EW and re-log again later to check the status of execution, which is carried out continuously "in the background". Technically speaking, this was achieved by decoupling user sessions with the EW server from the underlying execution service that can operate without the user being logged in. When the user logs out of EW, all their experiment executions are detached from the user session and kept "parked" on the server side, which happens without interrupting the computations. Once the user logs back in, experiment executions are brought back from the server-side "parking" and reattached to the user's session. As a result, the user's workbench becomes persistent in such a way that the user can log in and out at any time, and even when the session expires the workbench state is persisted on the server side and can be retrieved when the user logs in again. Moreover, users are notified when the session expiration time is about to be reached and asked if the session has to be prolonged.

Improved URL handling. Following up with improvements aimed at supporting long-running experiments, the URL handling mechanism has been reworked in order to preserve the EW user interface state. Currently, the URL encodes the state of the user interface; thus subsequent references to a given URL retrieve the EW user interface in exactly the same state. That enables users to save interface state as an URL and retrieve it later by navigating to that URL. What is more, web browser history management is now supported as "back" and "refresh" buttons work in the expected way.

Pluggable openers for visualizing and editing data files. In order to deliver visualization and editing of complex data files, the so-called openers have been introduced. An opener is a web application powered by e.g. an applet, a javascript library, a flash component, or any other rich internet application technology. Openers are served by EW and run within users' web browsers sharing a secure session with EW. Openers are pluggable and can be added to EW at any time by their provider. As a result, openers can be developed and delivered

independently. In particular, existing web-based tools can be reused and adopted to fit the EW architecture. Openers interoperate with EW using its RESTful interface dedicated for openers: each data file stored on an executor is assigned an HTTP URL and can be fetched and saved using HTTP GET and PUT requests respectively. Openers are then able to GET a data file in a given (supported) file format and deal with it in specific ways, e.g. visualizing or opening it for edition. The modified file can be then PUT back on the server and stored in its original location.

Configuration Management. EW offers an interface through which administrators can manage all configuration of EW and underlying GridSpace libraries. This properly secured panel provides an easy way to change the configuration via a web browser, with changes applied immediately during runtime. This brings more dynamicity in managing EW and releases administrators from having to recompile or restart EW.

Interface for GridSpace Execution Engine Features: EW provides a user interface for the GridSpace Execution engine features described in the following subsection**.**

12.3.2 Futures added to GridSpace Execution engine during the MAPPER project

In order to support the features offered by the GridSpace EW and other MAPPER tools, the following functionality has been provided.

Abstraction layer for executors and portability of experiments. The computational resource model was altered in order to introduce a layer of abstraction above the vast range of ways in which computational facilities are made accessible – including SSH clusters, QCG brokers, AHE-accessible resources and others, not known in advance. According to the new model, the GridSpace experiment contains snippets that are associated with abstract interpreters (regardless of where they're actually deployed). Interpreters are then installed on a set of computational facilities, binding them with executors. GridSpace incorporates the Interpreter Registry that stores these bindings. The additional notion of the Executor Descriptor was introduced in order to supplement the experiment with instructions needed during the execution phase concerning executors which are to handle individual snippets. Consequently, the experiment itself became a portable format in terms of being more high-level, as the information about which infrastructure components should be used in its

execution is provided separately on the execution stage, in the Execution Descriptor data structure through the Experiment Workbench UI.

SSH-Executor. As a consequence of the new abstraction layer of executors, the implementation of SSH-Executor was decoupled from the base code of GridSpace and provided as a standalone pluggable and interoperable module.

QCG-Executor. Integration with QCG middleware is ensured by using a dedicated Java client exposing the QCG broker's API. The client is used in both Multiscale Application Designer (MAD) and GridSpace tools. The former uses the client to retrieve reservation information used during application composition. The latter uses it to submit computation tasks to the broker. In both cases authentication is performed by passing the user's proxy certificate when invoking broker operations.

In MAD, reservation information is presented to the user to build a mapping between individual reservations and application modules. This information is passed from MAD to GridSpace EW to be included in the job profile, which is a description of the tasks submitted to the QCG broker. Reservations can also be created in MAD by using an embedded view (Flash application) served by the broker. In this way the user is faced with an integrated system, capable of controlling the entire lifecycle of the application.

The QCG client is integrated with MAD and GridSpace by using the Maven build system; hence updating to new versions is a straightforward process.

AHE-Executor. Integration with AHE is provided on the basis of a modified AHE Client written in Java. The primary functions of this executor are user authentication and running jobs. Modifications were introduced in the AHE Client so that it is easily embeddable in other Java applications. Below we present a detailed list of functions exposed and provided by the AHE Executor:

- login – users are able to authenticate themselves; two methods of authentication are supported (in both cases the executor establishes a GridFTP connection to the file staging server that the runner machine will use for staging input and output).
- using a proxy certificate – the executor sends it to the AHE MyProxy server with help from the AHE Client. The proxy is protected by GridSpace-managed temporary credentials.
- user name and password – this mode assumes that the user has manually uploaded his/her proxy to MyProxy. The Executor expects a valid username and password and

then attempts use them to authenticate with MyProxy and download the proxy certificate.

- execute – this allows users to run code snippets with the GridSpace web interface. The modified AHE Client creates and submits AHE Job Objects to the AHE runner machine. Each job object contains all the information about the job, including the name of the application, its parameters as well as any other input and output. Following submission the AHE Client periodically polls the job runner for the job status. Once the job finishes, all involved components are notified, and the output is presented to the user. The AHE Client used by this operation does not stage any input or output (this is different from the behavior of the standalone client). The reason for this is that all resources needed for job execution are managed by GridSpace directly on the staging server and therefore are already in place. Since the last report, the AHE Executor has been extended with reservation handling. The user is able to reserve resources beforehand, and supply a reservation identifier along with the jobs to be executed. The execution process can be interrupted, and cancelled at any stage. AHE handles such interruptions according to its procedures. Interpreters that are relevant for running sample multiscale applications have been configured – in particular the LAMMPS and CPMD tools installed on AHE machines are available through GridSpace. Additionally, the configuration was extended so it is possible to run AHE jobs on all infrastructure nodes that support AHE Execution by themselves.
- logout – the connection to the staging server is closed. All temporary objects and authentication information is removed or discarded.

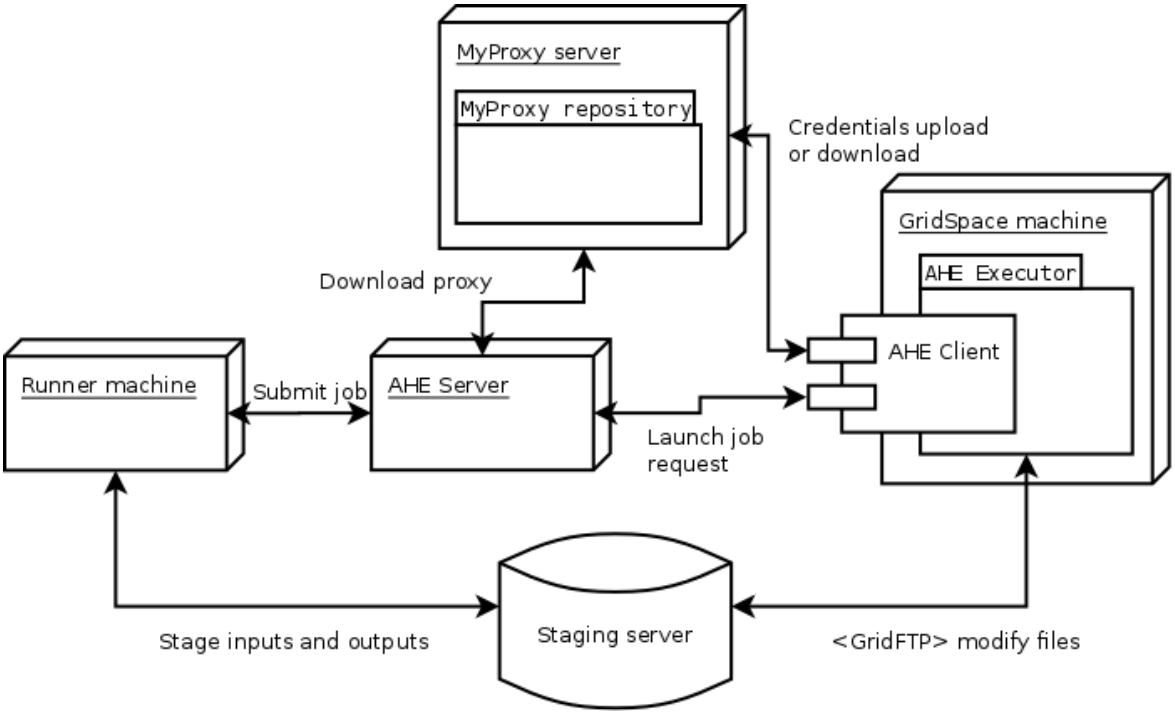

Fig. 15 The connections between individual elements in the GridSpace and AHE.

Fig. 15 shows the connections between individual elements in the GridSpace-AHE integration. The AHE Executor, operating on the GridSpace machine, manages authentication information by communicating with MyProxy server through the AHE Client. Input data is sent to the staging server via a GridFTP client, and then execution is performed by signaling the AHE Server. Once the execution initiates, appropriate runner machines stage input, execute applications and process data. Output is stored on the staging server. Following execution, any output can be retrieved with the GridSpace file manager.
A new AHE-Executor has been developed in accordance with the new abstraction schema described above.

Login with proxy certificate generated on the fly. Since some executors (such as AHE and QCG) use grid certificate proxies for authentication and authorization purposes, a new method of authentication is now provided in EW. Users are no longer required to generate a proxy on their own, manually or using specific tools. Instead, the EW login form can accept a grid certificate and the corresponding certificate key, and generate a certificate proxy on the fly within web browser using a dedicated built-in applet. In this way, users' sensitive personal credentials are not sent outside their computers: the proxy is generated within the web browser and subsequently used by EW for authentication and authorization.

Use of multiple executors in a single experiment run. EW supports concurrent sessions with more than one executor. This means that users can log into multiple executors, manage files from different sites and run experiments that combines many executors.

Data flow between executors. As a consequence of using different executors within the experiment run, mechanisms for staging input and output files between executors have been implemented.

Support for long-lasting connections with executors. Support for long-running experiments requires long-lasting connections with executors that rely on connection-oriented protocols (e.g. SSH, GridFTP). For this purpose the GridSpace Execution Engine periodically triggers dedicated “keepalive” calls against all connected executors. Depending on the specific implementation of an executor, the keepalive call may be implemented in various, executor-specific ways. This call must be handled by the executor in order to avoid session expiration and to check connectivity with external systems the executor relies on, if any.

Isolated executor client modules. Since GridSpace is meant to support an unrestricted range of executors, mismatches and software conflicts between executor client modules are expected to occur and have indeed occurred. AHE and QCG client modules proved incompatible with each other; however this was remedied by introducing isolation between executor client modules enabled by the Java class loader mechanism. Executors share only the minimum amount of classes that are loaded by EE, while the rest is loaded by the executors independently, using isolated and separated class loaders. That avoids name conflicts of in Java classes induced by version conflicts of modules that multiple executors depend on. Such low-level isolation creates a robust foundation which strengthens the extendability of GridSpace in terms of the number of executors that can coexist in a single GridSpace instance.

Compound assets. Some multiscale application modules require entire input directories (rather than single files). Therefore, GridSpace experiments had to support directories as their input/output assets. As a result, application developers can now specify a path to an input/output directory using a terminating slash character. GridSpace treats with such input/output definitions as describing a directory. Moreover, a given directory can be specified as a compound set consisting of individual input/output files. Copying single files to and from compound assets is handled transparently by GridSpace.

Interpreter Registry. GridSpace configuration includes a file that lists available interpreters and their installations on executors. This file is XML-formatted and made available remotely via a RESTful API (used e.g. by the Multiscale Application Designer).

Parameterized interpreter arguments. Following the first prototype release it was discovered that when using scientific software packages (configured as interpreters in GridSpace) on different computing sites (configured as executors in GridSpace) some interpreter- or executor-specific parameters have to be set when submitting computations. For example, a queue name in the PBS job submission system is an executor-specific parameter, specific to a given computing site. The configuration file for a given scientific software package is an interpreter-specific parameter that needs to be provided on whatever site such software is to be run. For this purpose, GridSpace allows the user to specify a program invocation command that contains interpreter- and executor-specific parameters to be provided in the experiment and experiment execution descriptor respectively.

Experiment variable management. As GridSpace experiments are fully portable and execution environment-agnostic, they need to be supplemented with additional information about where and with what execution environment-related parameters they are to be run. This information is referred to as *experiment variables*. They can be easily added to the experiment, saved as a file (outside of the experiment file), or fetched from a file. This allows for persisting and managing multiple run configurations for experiments.

Support for subsnippets. In order to enable complex control flow in experiments, GridSpace introduces the subsnippet mechanism. It enables nesting code snippets – the nesting snippet can initiate execution of the nested snippet anywhere in its own code. This paves the way towards execution of nested snippets in loops and conditional statements for nesting snippet, which, in turn, facilitates experiments that follow popular application patterns such as parameter sweep.

GridSpace execution engine exposes an HTTP RESTful API for execution of subsnippets that is callable from within snippet processes on target executors. Calls to this API are performed using the snippet's interpreter language. However, in order to release experiment authors from dealing directly with REST requests these invocations are generated automatically using configurable code templates for specific interpreters. Such templates are included in the Interpreter Registry.

GridSpace Directory Library. GridSpace EW configuration management relies on the GridSpace Directory library that enables reading, storing and changing configuration files during runtime. Based on the Apache Virtual File System project, it can handle various types of remote storage so that GridSpace configuration can be stored in a remote location, or even shared by multiple instances of GridSpace. It also ensures that changes in configuration files are applied immediately.

12.4 MAPPER Memory

The following sections describe two parts of MAPPER Memory: Registry of Module Metadata and xMML repository.

12.5 Registry of Module Metadata

The primary purpose of the module metadata registry is to persistently store and publish descriptions of scale models, mappers and filters developed for MAPPER applications. In order to deliver such functionality, MaMe employs a three-layer architecture, with a persistent database, a semantic domain model and external user/API interfaces (see Section 8.2.2.3 in D8.1). It supports several modes of interaction:

Browsing registered elements. The user is able to see all the registered *scale models*, *mappers* and *filters* as the basic building blocks for any MAPPER multiscale application (Fig. 16). The defined ports and implementations of the elements are also stored in MaMe, yet due to the growth in the number of stored models and mappers, the element list was made more concise. The main view only presents the most crucial information to the user, allowing him or her to click for further details, as needed (see Fig. 17).

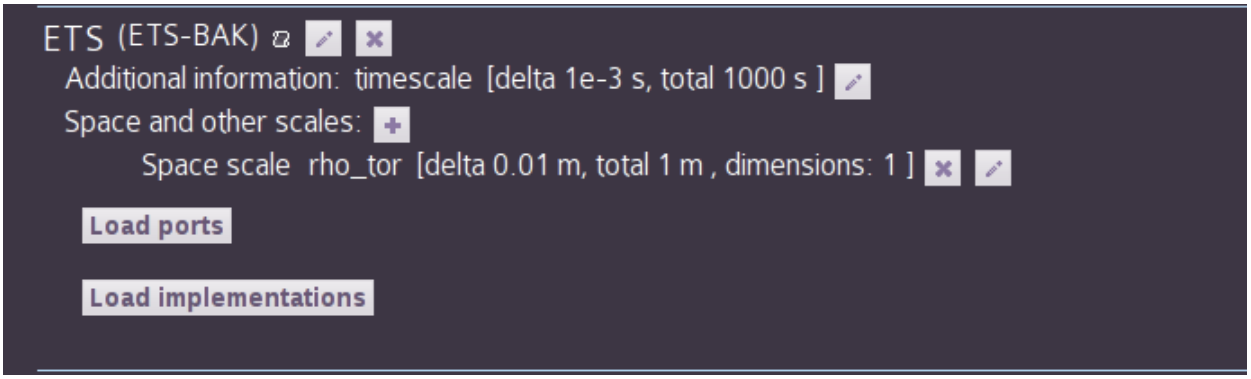

Fig. 16 A single-scale model presented by the MaMe Model Registry. Only the most general metadata is displayed when browsing the list of available models, mappers and filters. Any additional information and dialog boxes (ports, implementations) are loaded on demand with asynchronous AJAX calls.

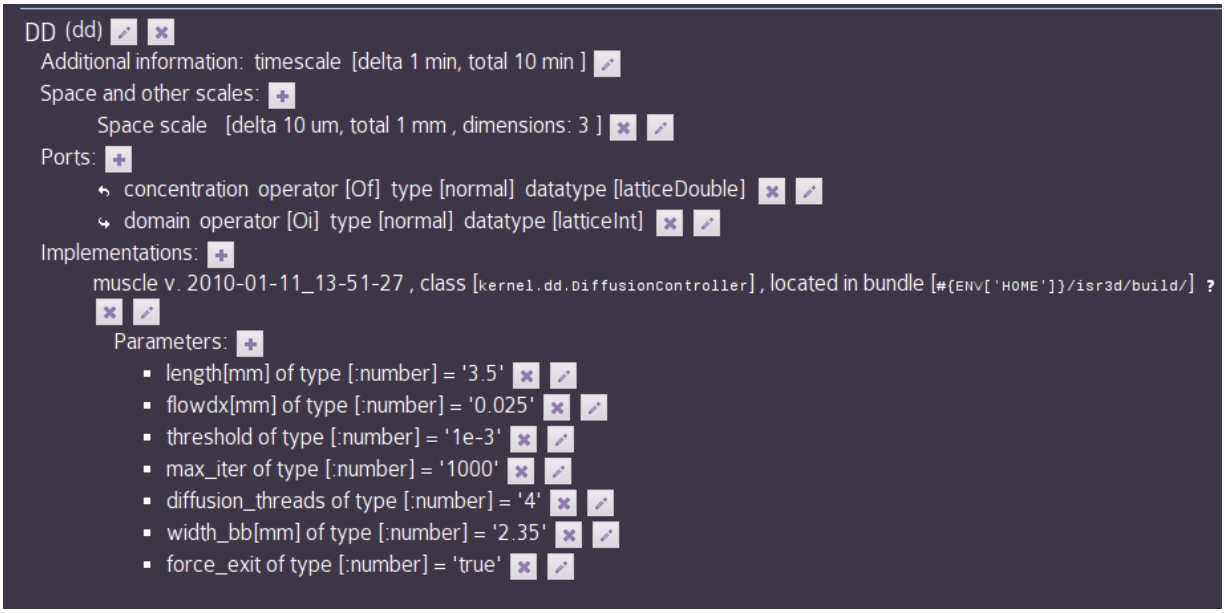

Fig. 17 Any element of a MAPPER application, be it a scale model, a mapper or a filter, may have its *implementations* registered in MaMe. In this way other MAPPER components (MAD, GS etc.) are able to learn how to execute a given computation. Moreover, scale models and mappers may have *ports*, which play a crucial part in xMML notation and which provide the means for putting together complex applications into a single workflow processing.

Registering new elements. MaMe provides a set of web forms to define new elements of multiscale applications – scale models, filters and mappers – along with their attributes and subelements (e.g. ports or scales).

Updating existing elements. Users are able to delete any element definition (or part thereof) and they are able to alter the description (metadata) of such elements. Fig. 18 and Fig. 19 present two examples of web forms provided for easy and effective management of registered metadata.

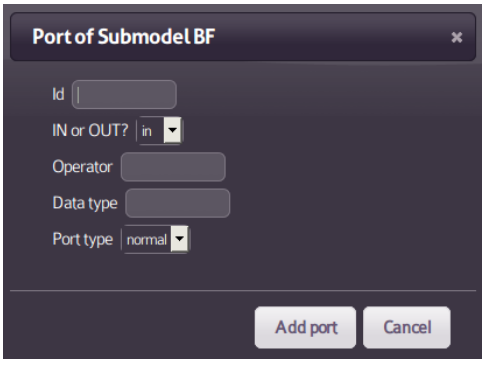

Fig. 18 A simple MaMe web form for altering application element description by adding a new port; the administrator or the developer is able to add and remove any part of module metadata. In addition, any element can be modified *in-situ*.

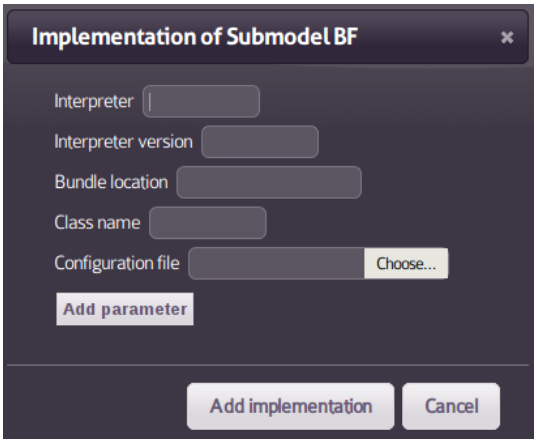

Fig. 19 As the codebase of MAPPER applications grows, the administrator may use MaMe forms such as this one to add new implementations of registered application modules.

Apart from its persistence capabilities and its user-oriented web interface for metadata browsing and management, MaMe also provides an API for other MAPPER tools to connect with the registry and publish or retrieve stored information. Currently, the MaMe API provides the remote operations, based on the REST principles, as shown in Tab. 10

Tab. 10 MaMe API interface for other MAPPER tools to connect with the registry and to publish or retrieve stored information.

/models-list	lists all the registered models, mappers and filters with their full descriptions
/add_base/Submodel	registers a new scale model
/add_base/Mapper	registers a new mapper
/add_base/Filter	registers a new filter
/add_implementation/(Mapper|Submodel|Filter)/'id'/	adds an implementation of a given type of element (either scale model, mapper of filter)

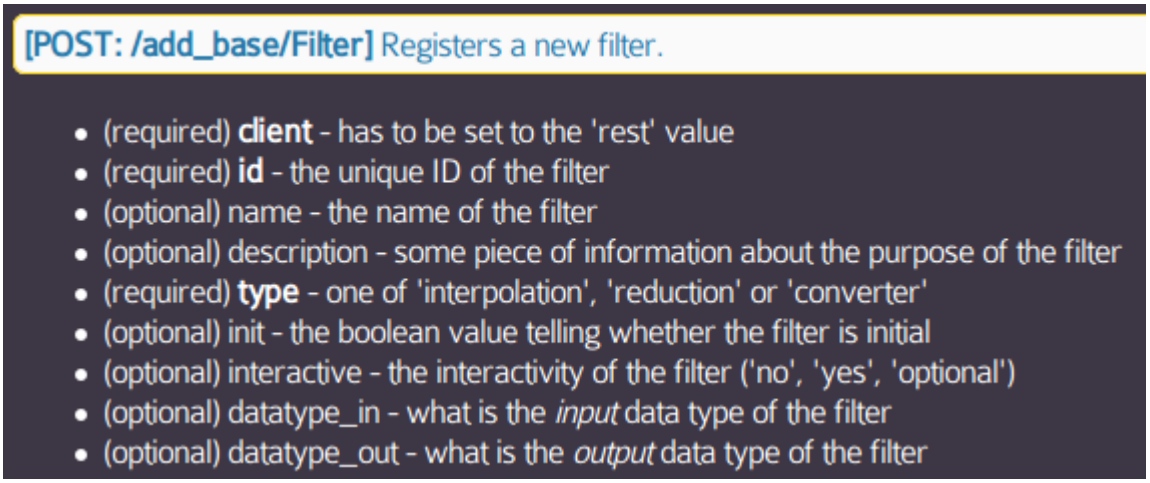

Fig. 20 All REST operations, that MaMe exposes as its API for other MAPPER tools in use for interactions, are described online. Users can click the API Help button in the MaMe main menu to view documentation on each API operation, similar to the one in the picture.

The API is documented online so each MAPPER developer has access to it at anytime (see Fig. 20 for a sample API method description in MaMe). This API is actively used by MAD. Since the API is both public and open, there are no obstacles to adding further tools which interface with MaMe's xMML metadata.

12.6 Repository of xMML Descriptions

The second component of MaMe is a repository of xMML application (or experiment) descriptions. Its main objective is to provide MAPPER MAD users with a persistent and shareable storage for multiscale application descriptions in the form of an xMML schema.

As presented in Fig. 21, the Experiment Repository holds descriptions of multiscale applications. These descriptions may be written by users by hand, but the preferred (and arguably better) method of planning applications is to use the MAD tool to design a new multiscale application in a user-friendly way. MAD is able to load stored xMML descriptions and save new ones in the repository.

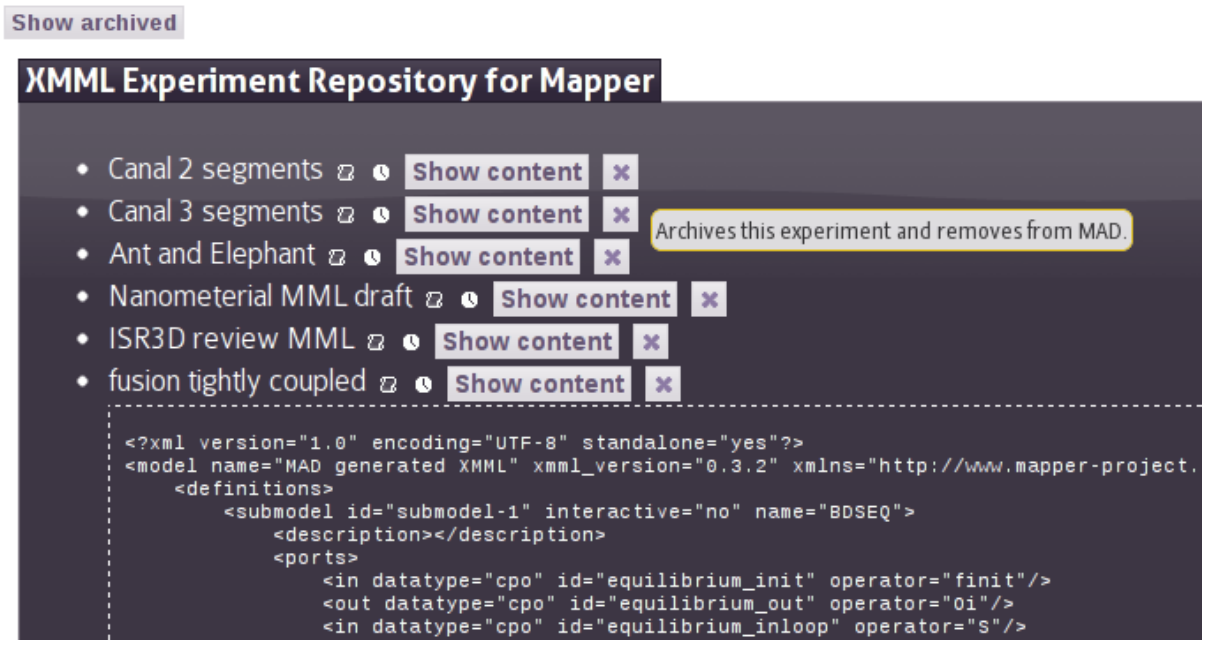

Fig. 21 Main view of the Experiment (xMML) Repository part of MaMe, showing the list of recorded experiment (application) descriptions, each represented by an xMML document in the specific version of this notation, along with a list of application elements and their interconnections.

In order to provide a method for removing unnecessary, older versions of experiments, an experiment archive has been implemented – removing any xMML experiment description from MaMe will move it to the archive (instead of deleting it completely). Such archived experiments do not clutter MaMe and MAD displays but are easy to retrieve with the use of the Reload from Archive capability of MaMe.

In order to accommodate MAD and similar tools using MaMe internally, a HTTP/REST interface was developed for the xMML Repository. Currently it provides three operations listed in Tab. 11.

Tab. 11 HTTP/Rest interface of the xMML repository

/experiments_list	returns the full list of available (i.e., not archived) experiments and their metadata, in JSON format
/experiment_content	returns the full xMML document for a specific experiment saved in MaMe
/save_experiment	saves a given experiment if no such experiment yet exists, or updates its content if it has been already been recorded by MaMe

The user is also able to upload his/her description written manually in the xMML XML notation. If the parsing process is successful and the description conforms to the latest xMML schema definition, MaMe will extract all the *model*, *mapper* and *filter* information from the document and register them, rendering them available for browsing and updating in the MaMe user interface, and for building new application descriptions in MAD.

12.7 Provenance System

The provenance system uses an RDF database to collect following events:

- experiment has started – contains date, experiment name and metadata;
- snippet execution has started – contains date, experiment for which this snippet is run, snippet input (snapshots are created) and snippet code;
- snippet execution has been canceled – same as above, but creates snapshots of snippet output instead of input. Additionally, the output is linked with the snippet that produced it;
- experiment execution has finished – contains date and reference to the same experiment metadata as the experiment start event.

The ontology used for describing provenance is based on the Open Provenance Model Vocabulary (OPMV - http://open-biomed.sourceforge.net/opmv/ns.html). Key elements are shown in Fig. 22.

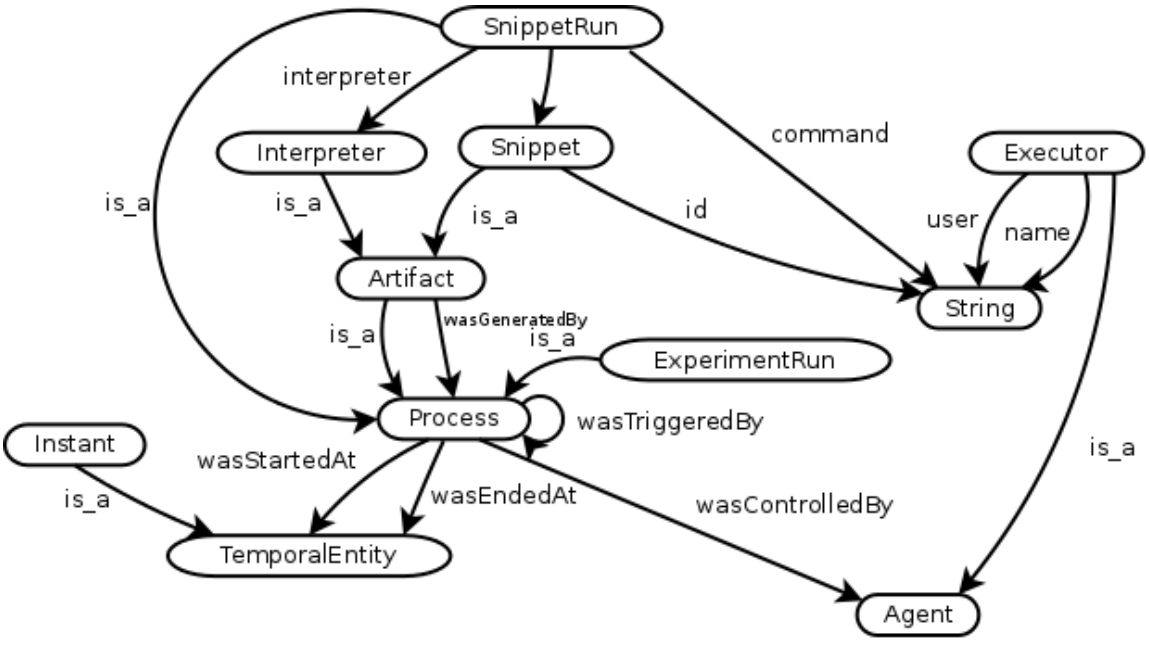

Fig. 22 The ontology used by the MAPPER provenance system

The system uses a 4store RDF database for keeping provenance metadata and an SVN repository for versioning input and output files of experiment steps (snapshots). Additionally, creating script snapshot is supported. During the final reporting period the provenance data model was validated and extended with interpreter artifacts which enable easier experiment tracking.