D 8.2 First prototype with demonstration

Project acronym: *MAPPER*

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

Grant agreement no.: 261507

Due-Date:	30 September
Delivery:	28 September
Lead Partner:	Cyfronet
Dissemination Level:	Public
Status:	Final
Approved:	QAB
Version:	1.7

DOCUMENT INFO

Data and version number	Author	Comments
25.07.2011 v0.1	Katarzyna Rycerz	Plan of the document
05.09.2011 v0.2	Katarzyna Rycerz	General architecture chapter, draft of executive summary and skeleton chapters
06.09.2011 v0.3	Tomasz Gubala	Added parts concerning MaMe, result management and browsing
07.09.2011 v0.4	Katarzyna Rycerz	Improved draft concerning skeleton, added chapter about measurement of tools efficiency
07.09.2011 v0.5	Joris Borgdorff	Added parts about jMML
07.09.2011 v0.6	Alexandru Mizeranschi	SBML toolbox chapter
08.09.2011 v0.7	Katarzyna Rycerz	Executive summary and Conclusions Chapter
09.09.2011 v0.8	Joris Borgdorff	some corrections to jMML part
09.09.2011 v0.9	Eryk Ciepiela	Iinput to sections related to Experiment Workbench, Experiment Execution Engine, MML to Experiment, QCG Intergration
09.09.2011 v1.0	Katarzyna Rycerz	Corrections to general architecture chapter
09.09.2011 v1.1	Grzegorz Dyk	First version of section about provenance added
13.09.2011 v1.2	Daniel Harezlak	Added sections about MAD, jMML and CxA
13.09.2011 v1.3	Katarzyna Rycerz	formatting
15.09.2011 v1.4	Katarzyna Rycerz	Modifications to the document structure
16.09.2011 v1.5	Bartosz Bosak, Katarzyna Rycerz	Small changes to section related to QCG
23.09.2011 v1.6	Katarzyna Rycerz	Corrections according to internal review 1
28.09.2011 v1.7	Katarzyna Rycerz	Corrections according to internal review 2

TABLE OF CONTENTS

LIST OF FIGURES

1 Executive summary

This document describes the functionalities and possible access to demonstrations for the multiscale programming and execution tools in the MAPPER project. More specifically, D8.2 is the prototype of the tools facilitating creation and execution of multiscale applications with structure described in Multiscale Modelling Language.

The presented tools support composition of multiscale applications from existing single scale submodules installed on e-infrastructures. After being composed, such applications are executed.

The first prototype contains most important programming and execution tools which includes: the application composition tool called Multiscale Application Designer (MAD), Registry for application modules description implemented as MAPPER Memory (MaMe), tools supporting high level stage of execution: GridSpace (GS) Experiment Workbench (EW) and GS Execution Engine. The execution tools interact with underlying interoperability layer to access the infrastructures. The full design of the tools was presented in D 8.1. As for the first prototype actual applications are manually programmed, the tools have been initially tested using skeleton application. We present architecture of the current implementation, detailed description of the tools, their current functionality, links to prototypes, code repositories and/or demonstration videos.

The document is organized as follows: In Section 4 we briefly describe architecture of the prototype and its relation to design in D 8.1. The detailed information about each tool prototype can be found in Section 5. The use case on the sample skeleton application used for tools' tests is described in Section 6. Section 7 we list links to prototypes, code repositories and/or demonstration videos. Section 8 outlines preliminary evaluation of efficiency of WP8 tools. We conclude in Section 9. The status of the actual MAPPER applications and their preliminary usage of the tools can be found in D 7.1.

2 Contributors

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

Cyfronet: M. Bubak, E. Ciepiela, G. Dyk, T. Gubała, D. Harężlak, K. Rycerz
PSNC: B. Bosak, M. Mamoński ,T. Piontek
UvA: Joris Borgdorff
UU: Alexandru Mizeranschi
UCL: Stefan Zasada
UNIGE: Bastien Chopard

3 Glossary of terms

In this document we will use terminology listed below. Additional glossary of terminology can be also found in Section 3 of D 8.1.

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

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) modules parameters (2) couplings between modules.

Executor: a common entity for hosts, clusters, grid brokers etc. It's anything that is capable of running software which is already installed on it (represented as Interpreters).

Experiment host: host where GridSpace experiment is executed

Filter: in MML terminology one-to-one type of connection between submodels

gMML: see MML

jMML : java library supporting MML

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

Interpreter: a software package accessible from any script language available on any infrastructure accessible by MAPPER community. An example of interpreter can be MUSCLE(See D 8.1) or LAMMPS[1] tools. We assume that the 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 submodels instances is loosely coupled if there is no cycle between them in the coupling topology, and tightly coupled otherwise.

Mapper: in MML terminology mapper is one-to-many type of connection 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 description

Multiscale Application Designer (MAD): MAPPER application composition tool

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

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

Multiscale model: the model of a multiscale process.

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

- **xMML**: the XML representation of MML that contains all information about application structure. The latest version of xMML specification can be found on http://napoli.science.uva.nl/xmml/xmml.tar.gz.

- **gMML** - the graphical representation of MML that contains only part of 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 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 script language.

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

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

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

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

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

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

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

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

xMML: see MML

4 General architecture of the first prototype.

4.1 First prototype architecture in the context of the tools design.

The general architecture of the first prototype is shown in the Fig 1, which is enhanced version of the architecture presented in D8.1 according to evolving user requirements. The first prototype contains most important multiscale programming and execution tools. As can be seen in the figure most of the modules are already present in the implementation of the first prototype. This includes: Multiscale Application Designer (MAD), Modules Registry implemented as MAPPER Memory (MaMe), GridSpace (GS) Experiment Workbench (EW) and GS Execution Engine. We have also introduced a new module (GS Interpreters Registry) that turned out to be important for the overall tools functionality as described in Subsection 4.2. There are also some less urgent modules that still exist in a design phase and are shown in the figure as dashed line boxes. We will subsequently improve the presented prototype by implementing also these modules according to evolving user requirements.

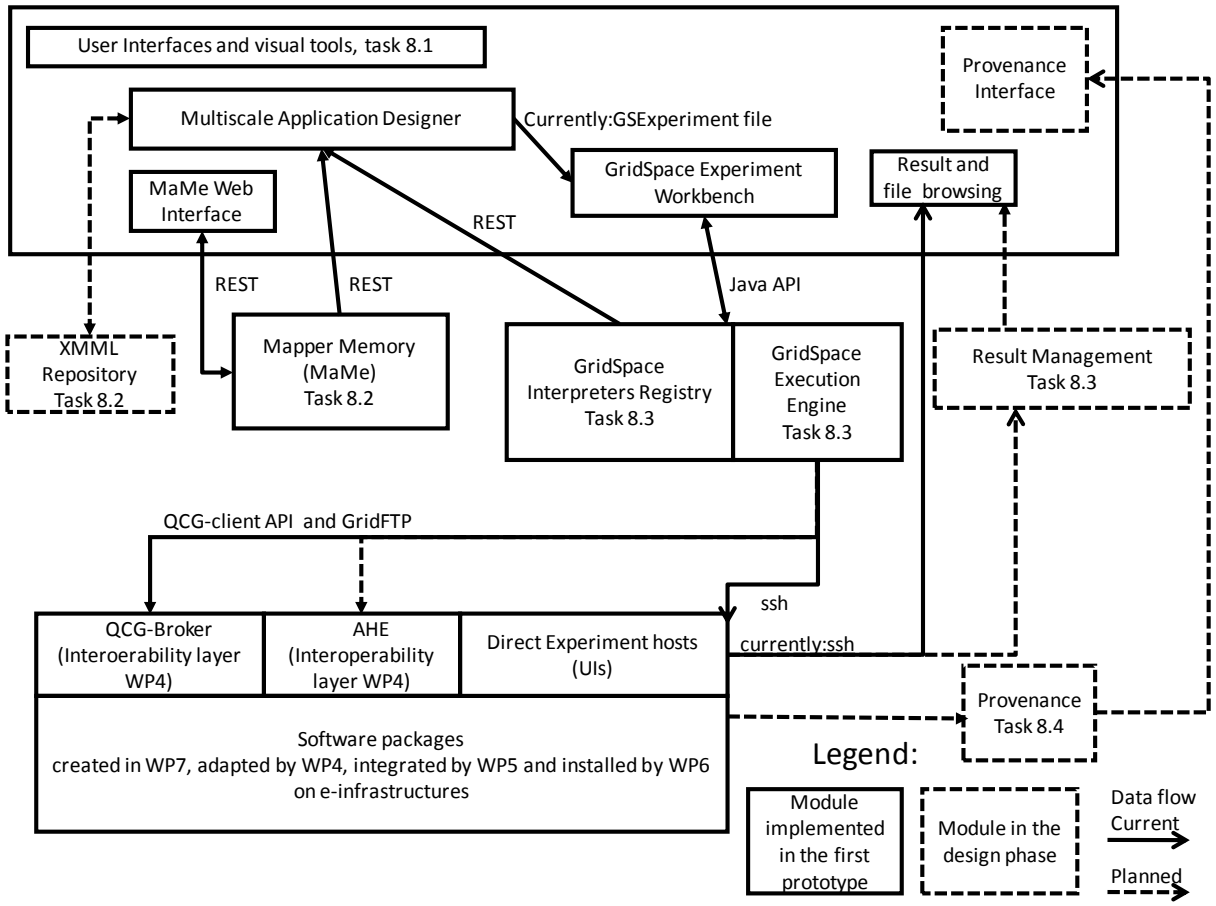

Fig. 1 Architecture of the first prototype of the programming and execution tools in the context of the full design presented in D8.1.

In Fig. 1 we also depicted protocols and interfaces between the tools. As the interoperability layer services (QCG-Broker, AHE) uses standardized protocols (GridFTP) for data flow the GS Execution Engine use them to stage-in and stage-out input and output files.
We assume that interoperability layer services are required to support gridftp protocol as they are intended to support european e-infractructures. The data flow itself will be managed by GridSpace using gridftp clients and - where possible - third party file transfer.

In the next chapters we present the detailed description of the tools and contain architecture of the current implementation, its current functionality, possible changes in comparision to the design in D8.1, links to the prototype, code repository and/or demo film. The detailed plan of vertical integration of all MAPPER software is described in D 5.2.

4.2 Altered execution model of applications and resources

Taking into account application requirements described in D 4.1 from the execution tools perspective we have categorized modules of MAPPER applications in the following groups:

- script snippets (pieces of code) of common script languages (e.g. Perl, Ruby, Python, Bash) – e.g. Nanotechnology application [GROEN]),
- script snippets for general-purpose toolkits (e.g. LAMMPS or CPMD for Nanotechnology application,
- script snippets (e.g. CxA connection file in ruby for MUSCLE applications like ISR [CAIAZZO] or Canals [THANG]) for connecting/executing compiled computational kernels of actual application,
- Additional codes and libraries (e.g. computational kernels in Fortran in Fusion applications etc.)

We have introduced the execution model of MAPPER applications that consists of snippets that are associated with the interpreters. In our model *Interpreter is a software package accessible from a script language, that is available on the infrastructure no matter where it's installed*. It describes the software itself: what input it requires, what parameters it uses etc. An example of interpreter can be MUSCLE or LAMMPS tools.

On the other hand we needed abstraction level above a range of ways the computational facilities are accessible. The tools should support ssh-accessible clusters, QCG-Broker, AHE-accessible resources and other not known in advance within or beyond the scope MAPPER project. Therefore, in our model, we introduced *executor entity i.e. a common entity for hosts, clusters, grid brokers etc*. capable of running software which is already installed on it (represented as Interpreters). The execution model assures that execution format of application (called GridSpace experiment) can be considered as portable in terms of infrastructure to be used. The Interpreter Registry was introduced to store bindings between interpreters and executors (see Subsection 5.3.2).

5 Tools - description of the first prototype

5.1 User Interfaces and visual tools

5.1.1 jMML Library

The jMML library is a Java library that handles MML [BORGDORFF] described in detail in Subsection 4.2 of D8.1. (see also Section 3). Specifically, jMML it can read and write xMML[2] (textual representation of MML) and it can generate gMML (graphical representation of MML). The relationship between gMML and xMML is explained in detail in Subsection 8.1.1.3 of D8.1. (see also Glossary of terms in D8.1). jMML is a Maven[3] project with three modules:

[2] http://napoli.science.uva.nl/xmml/xmml.tar.gz

[3] http://maven.apache.org/

- 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, it can detect which submodels need to be started initially for the model to calculate correctly by constructing the coupling topology. Second it, can output the coupling topology as a Graphviz file, which can then be converted to PDF containing MML by the dot tool of Graphviz. 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.

5.1.2 Multiscale Application Designer (MAD)

The current implementation of MAD provides a functional MAPPER application composition tool (see D8.1 for detail design). A snapshot is presented in

Fig. 2

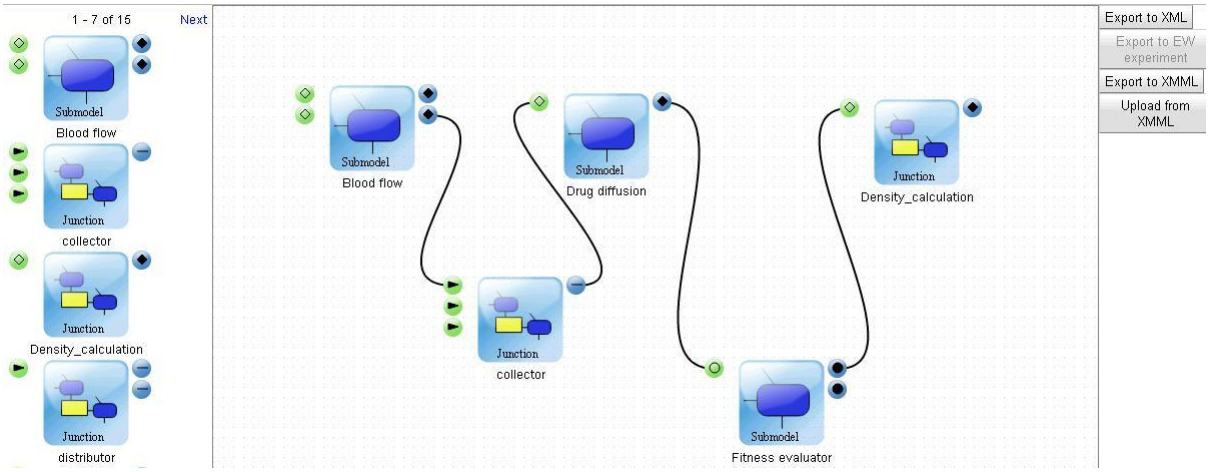

Fig. 2 A snapshot of the current implementation of MAD showing sample multiscale application composed from single scale submodules.

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 are in the middle of the view. There, application composition can be performed by connecting the 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 to GS experiment which can be executed inside GS experiment workbench, exporting to extended xMML format and finally loading a graph from the extended xMML file. As current multiscale applications are complicated more in terms of submodules connections then their number, the current version does not support advanced zooming features. This, however, is subject to change according to actual applications requirements.

Submodel and mapper items are generated according to the contents of the MaMe registry which is one of the external dependencies of MAD. The xMML repository support is still under implementation and will allow users to share their applications.

Usage of jMML

The jMML library provides MAD with basic data structures of the xMML notation, as well as with auxiliary algorithms to detect tightly-coupled sections of application described by xMML (see also Subsection 8.1.2 and Fig. 14 of D8.1). The detection is used during the generation of GS experiments (see below) to produce one snippet per one tightly-coupled section. As tightly coupled application are MUSCLE-based, the snipet is in CxA file format.

The common code base ensured by using the jMML library speeds up the development and keeps the tools synchronised. Staying up to date with the jMML development is carried out by using Maven dependencies. For external use XML Schema can be used.

Transforming MML to GS Experiment

The first prototype of the MAD tool supports exporting MML diagrams both to xMML format (for describing high level application structure) and GridSpace experiment format (for execution in GS Experiment Workbench as described in Subsection 5.1.3). In particular, the experiment can contain CxA snippets for MUSCLE application that can be executed on various infrastructures (e.g. QCG). This is described in detail later in this section.

Importantly, prior to export multiscale application as GridSpace experiment, the MAD tool needs supplementary information from the Interpreters Registry described in Subsection 5.3.2. The registry tells what software can realize submodels and which infrastructure

elements can run such software. This information is intentionally not included in MML description in order to keep it independent on software that realizes submodules. MAD tool inquires MaMe (see Subsection 5.2.1.1) for interpreters that realize given sub-models, and then inquires the Interpreters Registry about what parameters and codes are required by the interpreters. Therefore, MAD user can choose the interpreter that suits his or her the best and provide all parameters and codes it requires. That combo of information is then enough to construct GridSpace experiment.

Despite experiment designates interpreters that are to be used to realize submodes it is still ambiguous what infrastructure or sites are to be used to execute the experiment. The selection of infrastructure to be used is left for the multiscale application execution step that is carried out though the GridSpace Experiment Workbench tool. As a result, business-logically the same experiment can be run in many different ways by harnessing different infrastructures to run it.

Support for QCG Job Profile
In D 8.1 we have described possible ways of transformation between xMML to QCG JobProfile languages. According to GS model of computational resources (see Subsection 4.2.), the QCG is considered as one of the executors and the transformation is performed on the level of Execution Tools (see Subsection 5.3.3).

Support for CxA snippets
During the process of building the final GS experiment from the gMML representation 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 needed to compose individual sections is retrieved from the MaMe registry. The current template of the CxA configuration is given below in Fig. 3

```
# 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. 3 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 particual Mapper submodules and are provided during the process of registration in the MaMe registry.

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 the 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 paramater names and values from the MaMe registry.

Tha last CxA configuration section defines connections between kernels. It is built according to the graphical representation of the application. The first prototype of MAD does not make any connection validation.

5.1.3 GridSpace Experiment Workbench

The first prototype the GridSpace Experiment Workbench offers a set of additional features that adopt it to the MAPPER Framework.

First, GridSpace was adapted to a new execution model presented in Subsection 4.2 consisting of interpreters (independent on infrastructure), executors (types of infrastructure). The additional notion of Executor Descriptor was introduced in order to supplement the experiment with instructions needed on the execution phase about selected executors to handle particular snippets. Executors are implemented as a GridSpace plug-ins that handle communications with computational facilities like the ones mentioned above. For the first prototype we support SSH Executor and QCG Executor. Consequently, the experiment itself became a portable format in terms of being more high-level, as the information about what infrastructure to execute it are provided separately on execution stage in the Execution Descriptor data structure through the user interface of Experiment Workbench.

Secondly, the first prototype of Experiment Workbench supports simultaneous sessions with more than one executor. It means that users can log in to multiple executors, manage files from different sites and the same time, run experiments that involve many executors.

The current status of Experiment Workbench allows for creation of experiments from scratch and running it over the computational resources of MAPPER. Moreover it is prepared to open experiments previously exported from MAD tool.

5.1.4 MAPPER Application Result Browsing

Subsection 8.1.2.3 of D8.1 describes one of the extensions to the GridSpace Experiment Workbench - the remote file browser widget. The widget is planned to be further extended to suit the needs of MAPPER user.

The first extension to the file browsing capability, which was introduced up to date, is the ability to browse files in a cross-site manner. This is especially helpful for user of multiscale applications, whose elements are bound to run on different machines and clusters. That means, the input files, the intermediary and the final results are usually stored on different file systems within different computing installations (even in different cities or countries).

The current implementation of the file browsing allows the user, who has access rights to the respective computing elements, to access all these resources and be able to switch the file browser view from one machine to another. This capability is connected with the general implementation of multi-machine login feature of the GridSpace. As mentioned in Subsection 8.1.2.3 of D8.1, the widget will also be extended in the future to accommodate browsing results from the result store along with their provenance information, if available.

5.2 Programming tools

5.2.1 MaMe: MAPPER Memory

MAPPER Memory (MaMe) main responsibility is to provide rich, semantics-aware persistence store for other components to *record* information. The MaMe registry is meant to deliver its functionality based on a well-defined domain model which includes all important elements of MAPPER metadata defined in MML: **scale modules**, **mappers** and **filters**, together with their **ports**, **implementations** and other constituting elements and attributes. Thanks to MaMe, other MAPPER tools may store, publish and share common registry of such elements throughout the entire Project and its Consortium.
Currently MaMe is in its first, early prototype, and provides most of the functionality of registry for modules metadata (cf. Subsection 8.2.2 in D8.1). It also provides the initial part of the functionality of xMML description repository (cf. Subsection 8.2.1 in D8.1). The following subsections give more details on the current state of this prototype.

5.2.1.1 Registry of Modules 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. To deliver such functionality, MaMe employs three-layer architecture, with persistent database, semantic domain model and external user and API interfaces, as was planned and described in Subsection 8.2.2.3 in D8.1.
The prototype currently provides a set of possibilities 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. The defined ports and implementations of the elements are also presented. See Fig. 4 for example of the scale modules presented in MaMe and Fig. 5 to see a mapper with a defined implementation.

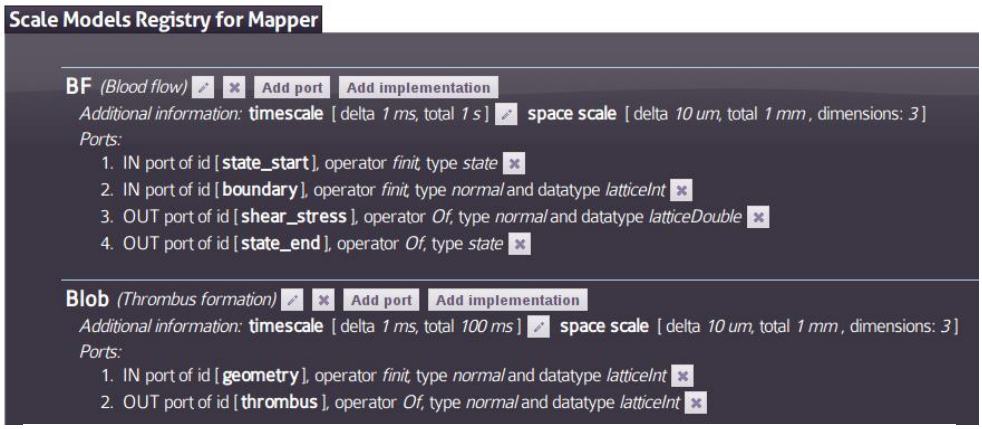

Fig. 4 A sample of MaMe application elements list with two scale modules

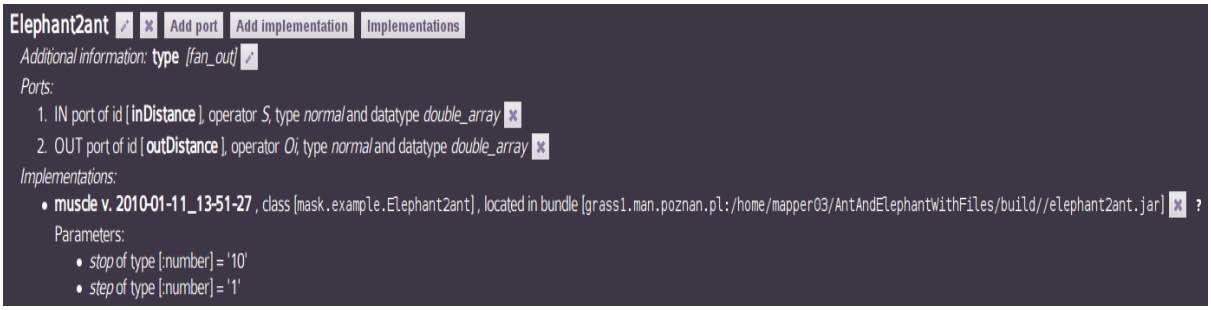

Fig. 5 MaMe also store information on current implementations of application elements - here, a mapper definition with a single implementation.

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. The users are able to delete whole or part of any element definition and they are able to alter the description (metadata) of such elements. Fig. 6 and Fig. 7 present two examples of web forms provided for easy and effective update and creation of registered metadata.

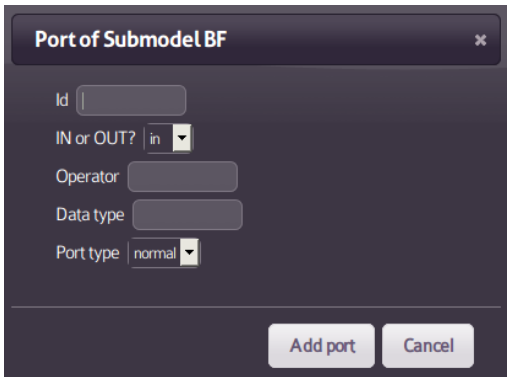

Fig. 6 A simple MaMe web form for altering application element description by adding a new port

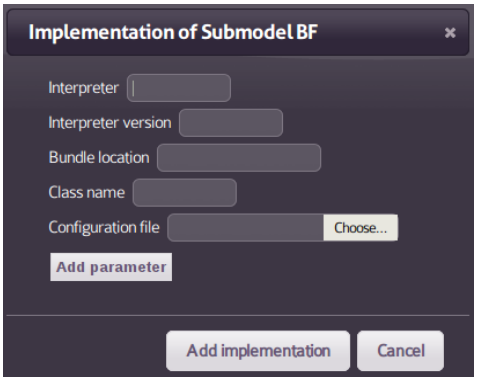

Fig. 7 While the codebase of MAPPER applications grows, the administrator may use MaMe forms like this one to add new implementations of registered application modules.

Apart from its persistence capabilities and its user-oriented web interface for metadata browsing and maintenance, MaMe also provides API interface for other MAPPER tools to connect with the registry and to publish or retrieve stored information. Currently, MaMe API provides the following remote operations (all based on the REST principles):

- /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).

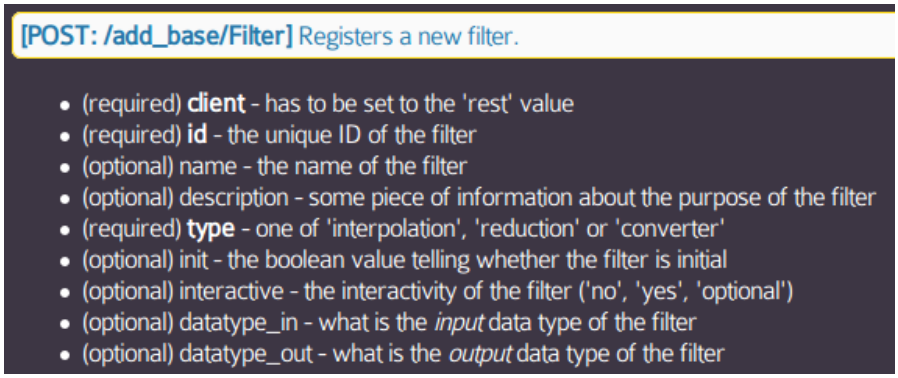

Fig. 8 All the REST operations, that MaMe exposes as its API for other MAPPER tools in use for interactions, are described online. Please use the API Help button on the MaMe main menu to see documentation on each of the API operations, similar to the one in the picture.

The API is documented online so each mapper developer has access to it at anytime (see

Fig. 8 to see a sample of API description in MaMe). At the moment two other MAPPER tools actively use MaMe API interface for sharing metadata:

- MASK skeleton generator stores information about elements of skeleton MAPPER application obtained through code generation capability (see Section 6).
- MAD application design tool retrieves the available elements of multiscale applications in order to provide the user all possibilities of building application structure (see Subsection 5.1.2).

The current MaMe implementation supports almost all use cases described in Subsection 8.2.2.1 of D8.1 (Modules Registry). Of course, MaMe will evolve with time according to future changes in xMML schema definition. Also, the API interface capabilities are expected to grow since more tools may need to access MaMe in the future and more elaborated interaction patterns may be involved.

Regarding the underlying metadata schema, the current prototype of MaMe module registry is based on xMML **0.3.2** schema definition.

5.2.1.2 Repository of xMML Descriptions

Another component of MaMe registry that was recently developed concerns the repository of xMML application descriptions. The main objective of this effort is to deliver the MAPPER tool users a persistent and shareable storage for constructed multiscale application descriptions in the form of xMML schema. As the development of the first MaMe prototype was focused mainly on delivering the registry of modules metadata, the stage of implementation of the xMML description repository is in its infancy.

Currently, as shown in Fig. 9, the user is able to upload her or his description in xMML XML notation. Then, if the parsing process goes properly 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 inside the internal database, so they are available for browsing and updating in MaMe user interface and for building new application descriptions in MAD.

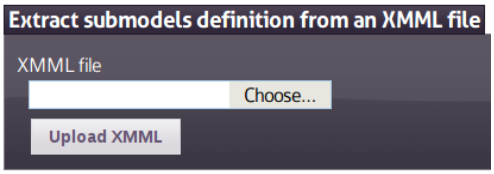

Fig. 9 The MaMe xMML upload form for extraction of valuable information on models, mappers and filters constituting a multiscale application.

MaMe internal domain model is currently not aware of application structure and may only store the building blocks, or elements, of such a structure. Further development will introduce additional extensions of the model to cover for different relations and connections between modules, mappers and filters. Thus, it will be possible to store the full structure of an application inside MaMe and, furthermore, user tools like MAD will be able to offer Consortium-wide **save** and **load** capabilities for created xMML descriptions.

5.2.2 The SBML toolbox

During the implementation of the systems biology application for MAPPER (which was described in deliverables D4.1 and D7.1) we encountered the need for extra functionality that tools such as COPASI[4] [HOOPS] could not provide on their own. We decided to go one step further from what was mentioned in deliverable D8.1 and build our own SBML toolbox, thus ensuring that we can incorporate any existing tools that provide the functionality we need and not rely only on COPASI.

SBML toolbox facilitates the conversion to and from SBML[5] and several other important file formats and tools used within the field of systems biology. SBML is becoming a widely-used standard for storing biological models, with an existing online database[6] storing validated models written in this format. Also, a rather new standard which we decided to adopt is SBRML[7] [DADA]. It was developed recently in order to address the issue of storing experimental and simulation data and meta-data regarding the way this data was produced

For our purposes, we have identified the necessary tools to enable the fo flow of information as shown in Fig. 10.

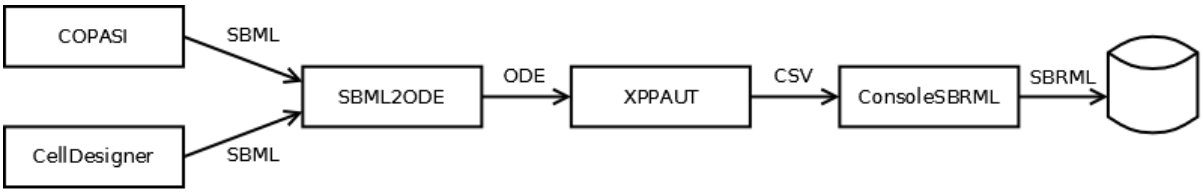

Fig. 10 Flow of information in a SBML toolbox

For generating SBML files, the user has several choices among pre-existing tools in systems biology. Tools such as the previously mentioned COPASI or CellDesigner[8] [FUNANHASHI] have been in continuous development for several years, reaching a suitable level of maturity.

---

[4] COPASI home page: http://www.copasi.org/

[5] The systems biology markup language: http://sbml.org

[6] The Biomodels online database: http://www.ebi.ac.uk/biomodels-main/

[7] The systems biology results markup language: http://www.comp-sys-bio.org/tiki-index.php?page=SBRML

[8] CellDesigner home page: www.celldesigner.org/

Both tools provide means to import and export SBML files, as well as additional features for simulating biological systems. Also, both tools provide an easy to use graphical interface, thus facilitating a user-friendly means of generating SBML files.

After generating the SBML files, we focus on a simple usage scenario: artificial data generation and storage (specifically, an artificial time-series experiment or simulation of the model represented in the SBML file). Although many existing simulation packages in systems biology (including COPASI and CellDesigner) provide this feature, our aim was to provide a simple toolset facilitating batch processing of SBML files, ready to be integrated in more complex applications. The result was a series of Java classes which provide the functionality we needed for our systems biology application.

To enable the data generation from the SBML file, the first aspect we need to investigate was using an ODE solver to integrate the differential equations specifying the reaction kinetics in the SBML file. For this, we chose the XPPAUT[9] ODE solver (Ermentrout, 2002). To enable this tool, however, we needed a way to generate an input file for XPPAUT from the initial SBML file. SBML2XPP[10] is a simple executable which can create .ODE files for XPPAUT from SBML files. We have successfully integrated it within our toolbox and it was our initial choice for an SBML converter while developing the systems biology application.

During the development, however, we encountered some optimization issues related to the execution time of the SBML2XPP converter. A much faster alternative for us was to program our own converter, by calling the routines that the libSBML[11] library provides. We use the Java version of the libSBML library to read from and write to SBML models and to construct an input file for XPPAUT. XPPAUT outputs its results in a CSV (comma separated values) file, which is then translated to SBRML by using a tool called ConsoleSBRML[12].

We have created simple Java library for managing and orchestrating each of the previously described tools. The functionality of our library includes:

- Using libSBML to create ODE files from SBML files containing a special pattern (we made assumptions about the SBML file structure, based on the problem we were investigating at the time).
- Creating and running a process of the SBML2Xpp tool for supporting any type of SBML file.
- Creating and running a process of the XPPAUT tool.
- Analysing the output of XPPAUT
- Converting an SBRML file to a CSV file. The functionality can easily be extended to provide the backwards conversion, as the ConsoleSBRML tool supports it.

[9] XPPAUT, a tool for solving stochastic, differential, and difference equations: http://www.math.pitt.edu/~bard/xpp/xpp.html

[10] The SBML2XPP tool: http://www.ebi.ac.uk/compneur-srv/sbml/converters/SBMLtoXPP-Aut.html

[11] The libSBML library: http://sbml.org/Software/libSBML

[12] The ConsoleSBRML tool: http://sbrml.sourceforge.net/SBRML/Welcome.html

5.3 Execution Tools

5.3.1 Execution Engine First Prototype

Experiment Execution Engine is primarily a back-end for the Experiment Workbench described in Subsection 5.1.3 that offers experiment execution capabilities. It uses executors in order to support computing infrastructures involved in MAPPER.

Execution Engine incorporates the Interpreters Registry which keeps data about available interpreters and executors in the MAPPER Framework and binding between them telling which executors support which interpreters. This information is needed in order to dispatch execution of snippets written in some interpreter to supporting executor.

On the other hand the same Interpreters Registry is used by MAD tool though the REST API. As explained in previous sections MAD needs this information to construct complete GridSpace experiment.

Therefore, Experiment Execution Engine along with Interpreters Registry backs Experiment Workbench and MAD tool.

In the first prototype Experiment Execution Engine support new features of Experiment Workbench and requirements imposed by MAD tool as follows:

- Support for new model of resources (including interpreters and executors) that allows for portable experiments, and late assignment of executors to execute the experiment.
- Support for multi-executor execution of experiment. Depending on user demands snippets can be dispatch to execute on different executors.
- As a consequence of using different executors within the experiment run we support mechanism for staging input and output files between executors.
- Interpreters Registry made remotely accessible via REST API.
- Implementation of executors including SSH Executor and QCG Executor.

5.3.2 Interpreters Registry First Prototype

Interpreters Registry is delivered as a part of Experiment Execution Engine, but being functionally independent and used by other elements of MAPPER Framework (namely MAD tool) it is considered as a stand-alone functional component of the architecture.

It's main role is to keep information about the resources available in the infrastructure made available to the MAPPER community including:

- Interpreters and Executors – see Subsection 4.2
- Binding between Interpreters and Executors - telling which Interpreters are available on which Executors. Includes additional data about how to access particular Interpreter on particular Executor.

Interpreters Registry exposes remote REST API for the use of MAD tool, which needs description of Interpreters in prior to export complete GridSpace experiment.

5.3.3 Connection with QCG Client First Prototype

As explained earlier, the support for computing infrastructures is realized through dedicated executors. Technically it means that GridSpace provides a dedicated implementation of the Executor interface that is a client for QCG-Broker. In the first prototype we provide such an Executor that is able to construct a JobProfile document that corresponds to the GridSpace experiment snippet and takes care of creation and staging of all the files involved including main script, input and output files. Executor for QCG-Broker is implemented as cyfronet.gs2.grms-executor module, which realized the contact specified in cyfronet.gs2.executor module's interfaces.

The executor given with a snippet to execute performs the following steps:

- The Executor checks all the input files required by the snippet and put corresponding input entries in the QCG Job Profile document. If the input file is stored in location that doesn't support GridFTP protocol the files are staged on dedicated GridFTP Server.
- Executor checks all the output files expected to be produced by the snippet and put corresponding input entries in the QCG Job Profile document. The output files are stored on the site of computation.
- The code of the snippet is copied to the main script file, which is also staged in the dedicated GridFTP Server, and the corresponding main script entry is put in the QCG Job Profile document.
- The JobProfile is sent to the QCG-Broker.
- QCG-Broker is periodically inquired for the status of the job.
- After the job is finished the location of produced files is stored for further use in case the subsequent snippets use them as input files.
- The input files and main script files previously staged-in on dedicated GridFTP server are deleted. It's reasonable as inputs are stored in their original location, and main script is kept in the experiment as code of the snippet.

More information about integration with QCG can be found in D 5.2.

5.3.4 Browsing Results of MAPPER Applications

The multiscale application outputs and results management, according to its design in Subsection 8.3.2 of D8.1, has two counterparts. The first one, storing and retrieving files on remote computing machines, is already realized through the GridSpace file browsing tool and its capability of performing file transfers to and from the target execution host with the ssh protocol.

The second part of the result management functionality, is related to storing metadata on computation outputs not accessible directly on computing hosts file systems. In such a case, the persistence layer of MaMe registry will be used to store relevant description of the output along with important information on its whereabouts. Then, the GridSpace file browsing widget will be extended to accommodate such an information. Also, the provenance component of MAPPER toolbox will be able to attach more provenance-related metadata to the results, if available and needed.

The first step to deliver this functionality was performed in the implementation of the first MaMe prototype. This assumed development and deployment of the common persistence layer (see Subsection 8.2.1.3 in D8.1 for details) for such metadata. Further work will consist of defining proper domain data model to describe the needed information and extending the REST API of MaMe to make the result metadata transfer there and back possible.

5.4 MProv - MAPPER Provenance data collector and storage

5.4.1 Functionality

The main responsibility of MProv tool is collecting, storing and publishing provenance information about the application execution process. This tool is still under development. Target functionality include:

- tracking multiscale application execution within the GridSpace. MProv communicates with GS and detects execution of each snippet to get its input, output, snippet name, experiment name, execution host etc. Development status: advanced
- storing gathered provenance information. This includes metadata (who did what, where, how was asset generated) and asset snapshots (called artifacts). Each asset version is kept and can be retrieved to rerun experiments with old arguments. Development status: to be deployed - individual components are third party, free software; more details in further sections
- providing a RDF data browser that enables user to input SPARQL query and search for particular provenance entries and relationships. Development status: finished.

5.4.2 Architecture overview

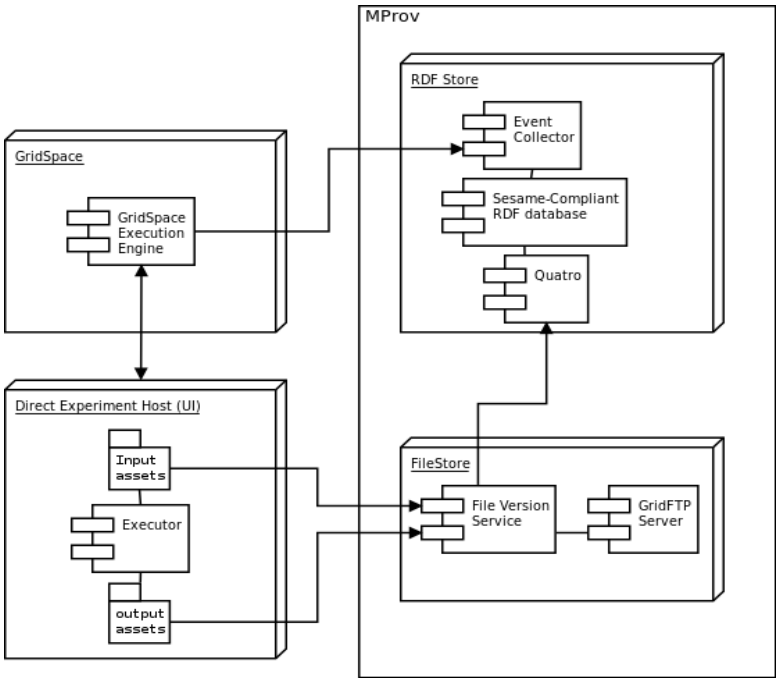

Fig. 11 MProv system architecture showing interactions between GridSpace Execution Engine and MProv tool.

MProv architecture is shown in Fig. 11 in the context of cooperation with GridSpace execution engine described in Subsection 5.3.1. The Event Collector exposes a REST interface that accepts provenance events sent from GS2. Information about these events is stored in RDF database. The content of the database is presented to the user by Quattro - a web application for browsing RDF data.

A separate host is responsible for storing file snapshots that take part in application execution. Files kept in this repository can be accessed directly with HTTP URLs allowing the user to get them directly from web browser running Quattro.

5.4.3 Provenance data acquisition

MProv's Event Collector is a REST-based service that accepts provenance events and stores them in RDF database. The events are specified in MProv's event format (MEF) as triples containing description of *action*, entity that invoked that action (named *who*) and the subject of that action (named *what)*. MEF is based on the concept taken from the Open Provenance Model Vocabulary (OPMV)[13] ontology: agent, process and artifact (see the specification for details). Examples include:

- who: process, artifact, supervisor
- what: activity target. May be a file, other process, variable defining a state etc.
- action: create (an artifact), start, stop (a process), updateParameter, read/use (a file, input, parameter), etc.

Two additional elements, *when* and *cause* are optional. The former specifies time at which event occurred (*now* by default) while the latter defined the cause of the action (event that triggered it).

An example of MEF event written in XML format is shown in

```
<event>
   <who type="process" name="snippetA"
/>
   <what type="file" name="output.txt" />
   <action type="create" />
   <cause description="backup store" />
   <when time="2011-09-06 18:04" />
</event>
```

Fig. 12 An example of event description in MProv

MEF will be mapped to various serialization formats: XML, JSON, ProtoBuf. Currently only XML is supported.

5.4.4 Provenance data storage

There are two types of provenance data gathered by MProv: metadata (relationships between entities - result of events received in MEF format) and snapshots of files created by experiment execution.

The metadata is kept in store of triples (triplestore). We want to use 4store[14] database as it is efficient and scalable. The triplestore will be accessed by the data collector using well defined Sesame framework API. Such abstraction facilitates access to the database using Java and makes it easier to replace the database backend (for example with native Sesame). We've already managed to make 4store work together with Sesame. We also managed to use the aforementioned OPMV ontology with slight modifications and extensions for storing provenance relationships.

File snapshots will be stored in a file versioning system. We want to use SVN as it works considerably well with large text and binary files. Additionally, host containing SVN server will also have a GridFTP server to enable uploading files from experiments run on e-

[13] http://open-biomed.sourceforge.net/opmv/ns.html

[14] http://4store.org/

infrastructures. The files will be uploaded to the file server directly from experiment execution host. Both input and output files will be sent.

5.4.5 Data publishing and querying

MProv will expose a user interface for querying and browsing provenance data in RDF format. This tool, called Quatro, is already implemented and allows user to graphically compose a query for RDF data and view results using standard web browser.

6 Multiscale Application Skeleton Prototype

6.1 MASK Functionality

As described in D 8.1 we have developed additional auxiliary Multiscale Application Skeleton (MASK) tool for creating multiscale applications skeletons - i.e. “empty” multiscale applications with the same structure as real ones (number and type of modules, execution type etc.) The motivation for MASK was described in D 8.1. The functionality of the skeleton is placed between the structure description (MML) and the implementation of the application. For the point of view of multiscale programming and execution tools, the created skeleton is a running application. We decided to separate language used to create the skeleton from language in which skeleton is executed (as in most skeleton frameworks [GONZALEZ]).
In the first prototype we have decided to use metaprogramming techniques of modern scripting languages (Ruby) to design a Domain Specific Language (DSL) for writing multiscale skeletons. Such DSL is independent on executing language (e.g. java, LAMMPS script etc) and independent on communication (e.g. MUSCLE, plan files, etc.). MASK is compatible with MML and the tools described in the previous section.
The current functionality includes:

- generating MUSCLE application from skeleton described in MASK DSL
- generating LAMMPS and Perl snippets from skeleton described in MASK DSL
- support for parametrising execution time, amount of exchanging data
- support for including legacy Java, LAMMPS and Perl code into the skeleton structure

6.2 Sample application

We have used MASK to create a simple multiscale application to test our tools. The application was used as a sample use case during meeting of WP7 (applications) and WP8 (tools). The status of tools usage by the real MAPPER applications is described in D7.1.
The application consists of three scaleful single scale modules and two mapper modules used for connections between scale ones. The application simulates very simple behaviour of an ant and an elephant. The structure of the application is hybrid - the loosely coupled part is followed by a tightly coupled one as shown in the
Fig. 13.

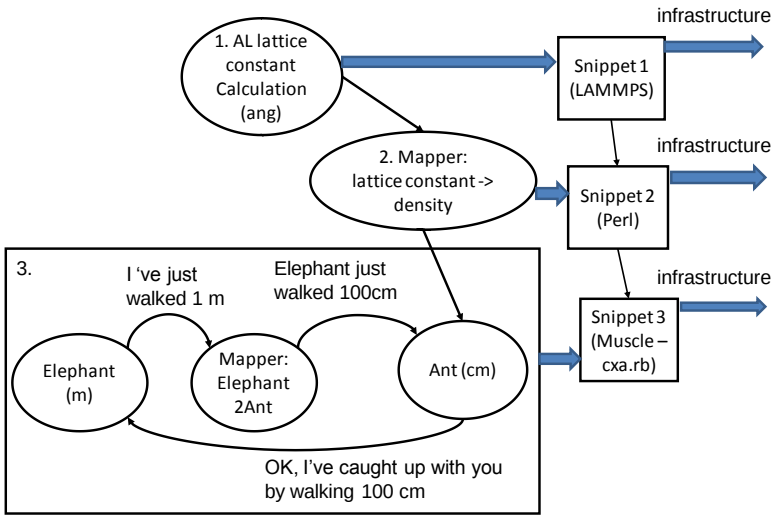

Fig. 13 Ant and elephant test application generated by MASK

The scenario of the application is as follows:

1. The module in nano scale calculates the lattice constant of aluminium atoms.
2. The scaleless mapper transforms the nano scale information into g/cm scale (density of aluminium)
3. The module called Ant calculates the weight of aluminium it carries from density obtained in the previous step (scale:centymeters)
4. The Ant walks after the Elephant in a tightly coupled manner:
    a. The Elephant walks a 1 m/step and sends information back to the Ant (scale:meters).
    b. The Ant walks 100cm and confirms its progress to the Elephant.

7 Prototype availability

Below we present details concerning availability of prototypes of each tool.

7.1 jMML library

The latest version is accessible to project members and can be downloaded with git from http://napoli.science.uva.nl/git/jmml.git.

7.2 MAD

MAD prototype is available on http://gs2.mapper-project.eu:18080/mad.
MAD development is supported by Maven software management tool. The source code is available through an SVN repository under https://gforge.cyfronet.pl/svn/gs2-utils/ibuilder. The project is divided into api and gwt modules. The first contains data model and high-level application logic definitions and the second module implements the presentation layer. In this case Google Web Toolkit[15] together with drag-and-drop[16] and SVG[17] libraries were used to implement this layer. Full and snapshot releases of the MAD tool are present in the Maven

[15] Google Web Toolkit, http://code.google.com/webtoolkit

[16] Drag-and-Drop Library for Google-Web-Toolkit, http://code.google.com/p/gwt-dnd

[17] Library to add SVG graphics to GWT applications, http://code.google.com/p/lib-gwt-svg

repository at http://dev-gs.cyfronet.pl/mvnrepo. The artifacts can be used as Maven dependencies in other projects or the final war (Web Application Archive) can be used to deploy the final application.

7.3 GridSpace Experiment Tools

From the software engineering point of view, the architecture of GridSpace Experiment Tools (i.e. GS Experiment Workbench, GS Execution Engine and GS Interpreter Registry) was changed and decomposed into several separate Java projects:

- cyfronet.gs2.experiment - the classes and APIs enabling construction of experiments and serializing it to XML format through JAXB technology, used both by Experiment Workbench and MAD that exports the multiscale application description in the experiment format.
- cyfronet.gs2.executor - the classes and APIs defining the contract between GridSpace and computational facilities.
    - cyfronet.gs2.ssh-executor - the implementation of the executor supporting SSH-accessible sites
    - cyfronet.gs2.grms-executor - the implementation of the executor supporting QCG-Broker.
- cyfronet.gs2.core - the Execution Engine that carries out experiment execution incorporated with Interpreter Registry
- cyfronet.gs2.ew - the web interface and server backend that incorporates all above-listed modules.
- The cyfronet.gs2.ew module is the Experiment Workbench web user interface, while the other ones are server-side libraries which we refer to later as Execution Engine.

The source code is kept in the SVN repositories with anonymous read-only access (login: anonsvn, password: anonsvn):
https://gforge.cyfronet.pl/svn/gs2-utils/module-name
The binary bundles of these libraries are stored in Maven repository:
http://dev-gs.cyfronet.pl/mvnrepo/.
The production installation of GridSpace Experiment Workbench is already available at https://gs2.mapper-project.eu.
The demonstration video on using Workbench with In-stent restenosis application can be found on http://www.youtube.com/watch?v=3S9-kljyXIw

7.4 MaMe

The MaMe prototype is currently deployed at: http://gs2.mapper-project.eu:1234/
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).

The sources in this early prototype are not yet prepared for external developers to build and deploy (as, e.g., another MaMe instance) on their own, yet they are open for read-only access to any interested party. One only needs a SVN client software for project checkout.

7.5 SBML toolbox

The codes are available in the svn repository
https://apps.man.poznan.pl/svn/sbml-toolbox/GRNApplication/src/

7.6 MASK and the test application

The codes of the MASK prototype are freely available on site https://github.com/kzajac/mask. An example of the DSL for the simple application can be found at https://github.com/kzajac/MASK/tree/experimental/examples.
MASK was also deployed in the current version of the GridSpace Experiment Workbench http://gs2.mapper-project.eu as one of the available interpreters. The demonstration videos on how to use the skeletons are available at http://www.youtube.com/user/dicecyfronetpl

8 Evaluation of efficiency of WP8 tools

According to the Description of Work, the first implementation of the project does not require us to couple applications using tools (only manually programmed version was promised), however certain metrics still can be used:

- We have set up tracking systems for MaMe, MAD and the GS Experiment Workbench tool, when we gather user requests. The number of requests can be viewed on http://chomik.cyfronet.pl/trac/ proving that tools are tested by MAPPER application developers
- There is a number of single-scale models already registered in models registry (MaMe) see http://gs2.mapper-project.eu:1234 proving that tool started to be used
- The preliminary papers presenting results from applications using tools were accepted on DMC2011 - Workshop on Distributed Multiscale Computing 2011 (held in conjunction with E-science 2011 conference) for:
    - ISR 3D with MUSCLE [BERNSDORF],
    - ISR 3D and MML [BORDORFF],
    - Nanotechnology application in MAPPER [GROEN],
    - ISR 2D with GridSpace and MUSCLE [RYCERZ].

  See also: http://www.computationalscience.nl/dmc2011/.

9 Conclusions

This deliverable presents the first prototype of multiscale programming and execution tools. It shows the current status of implementation according to the design presented in D 8.1. The functionality of the tools has been tested on a sample skeleton of multiscale applications created basing on real applications structures. Preliminary evaluation of the tools efficiency shows that the tools have already started to be used within the project by MAPPER applications. The current application status can be found in D 7.1. In the next years of the project the presented prototype will be enhanced to provide full functionality described in D 8.1.

10 References

[BERNSDORF] Joerg Bernsdorf, Guntram Berti, Bastien Chopard, Jan Hegewald, Manfred Krafczyk, Eric Lorenz, Alfons Hoekstra and Dinan Wang Towards Distributed Multiscale Simulation of Biological Processes, accepted by Workshop on Distributed Multiscale Computing 2011 in conjunction with the 7th IEEE e–Science conference.

[BORGDORFF] Joris Borgdorff, Jean-Luc Falcone, Eric Lorenz, Bastien Chopard and Alfons G. Hoekstra A principled approach to distributed multiscale computing, from formalization to execution, accepted by Workshop on Distributed Multiscale Computing 2011 in conjunction with the 7th IEEE e–Science conference.

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

[DADA] Dada, J.O., Spasić, I., Paton, N.W. & Mendes, P.: SBRML: a markup language for associating systems biology data with models. *Bioinformatics* 26, 932, 2010.
Ermentrout, B.: Simulating, Analyzing, and Animating Dynamical Systems: A Guide To Xppaut for Researchers and Students. *Society for Industrial and Applied Mathematics*, Philadelphia, PA, USA, 2002.

[FUNAHASHI] Funahashi, A., Morohashi, M., Kitano, H. & Tanimura, N.: CellDesigner: a process diagram editor for gene-regulatory and biochemical networks. *BIOSILICO* 1, 159 – 162, 2003.

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

[GROEN] Derek Groen, James Suter and Peter Coveney Modelling distributed multiscale simulation performance: an application to nanocomposites, accepted by Workshop on Distributed Multiscale Computing 2011 in conjunction with the 7th IEEE e–Science conference.

[HOOPS] Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P. & Kummer, U.: COPASI – a COmplex PAthway Simulator, Bioinformatics, 22(24):3067–3074, 2006.

[RYCERZ] Katarzyna Rycerz, Marcin Nowak, Pawel Pierzchala, Marian Bubak, Eryk Ciepiela and Daniel Harezlak Comparision of Cloud and Local HPC approach for MUSCLE-based Multiscale Simulations, accepted by Workshop on Distributed Multiscale Computing 2011 in conjunction with the 7th IEEE e–Science conference.

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