AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science

FINAL PROJECT

EXTENSION OF A TOOL SUPPORTING DESIGN OF MULTISCALE APPLICATIONS

KONRAD STRACK

SUPERVISOR:
dr inż. Katarzyna Rycerz

Kraków 2014

NON-PLAGIARISM STATEMENT

I HEREBY DECLARE THAT I HAVE WRITTEN THIS THESIS COMPLETELY BY MYSELF, THAT I HAVE USED NO OTHER SOURCES OR RESOURCES THAN THE ONES MENTIONED, AND THAT I AM AWARE OF THE RULES OF THE PENAL CODE REFERRING TO CRIMINAL LIABILITY FOR PROVIDING FALSE EVIDENCE.

. . . . . .

SIGNATURE

1. Project goals and vision

The goal of the project was to extend the functionality of the Multiscale Application Designer (MAD), which is developed as part of the MAPPER project (Multiscale Applications on European e-Infrastructures) [12]. This goal has been achieved by providing additional tools such as the built-in code editor, and by improving MAD's integration with other systems.

MAD supports users in design of multiscale applications. It allows them to create multiscale applications by combining multiple single scale models. These applications are described graphically using the gMML representation of the Multiscale Modeling Language (MML) [13]. Models and their types are described in the technical documentation in section 3.1.1

Models available for designing applications in MAD are fetched from the MAPPER Memory (MaMe) [11]. MaMe provides a persistent storage for the models and whole applications. Models are stored in the component of MaMe called the MaMe Registry, whereas applications are stored in the MaMe Repository.

Applications created in MAD can be afterward imported into the GridSpace Experiment Workbench [7], and GridSpace is responsible for their execution, and for fetching the results.

In order to support these various use cases, MAD provides the possibility to export applications to different XML-based formats. The xMML format (the XML representation of MML) is used by the MaMe Repository to store applications. The GridSpace experiment format is used to pass applications to GridSpace. For the detailed description of supported formats, please refer to section 4.1 of the user documentation and section 5.4 of the technical documentation.

The project aimed to improve user experience (e.g. the code editor) of MAD's users and integration with other services (e.g. the export and import capabilities), and to verify the direction of modifications. The plan was to test the modified version of MAD and provide it to the MAPPER consortium and other creators of multiscale applications, to be used in various fields, including nano-materials, simulations of irrigation canals, fusion modeling or medical simulations, and many more.

2. Functional scope

MAD is used mostly by scientists working in different fields such as biology or nanotechnology. They do not necessarily have a deep background in computer science. Therefore, all requirements had to focus strongly on user experience provided by various features of MAD.

The most important implemented aspects of the functionality were:

- The possibility to edit models belonging to a multiscale application. Users can edit implementations and parameters of models using a tabbed text editor. The editor also handles synchronization of parameters shared between models belonging to the same tightly coupled section.

- MAD-specific extensions to the xMML representations of multiscale applications designed with MAD. This functionality allows users e.g. to save the exact state of the design by exporting the application to xMML. The exported xMML file can be used to restore it afterward with all details.
- Management of implementations of models. Now all implementations are imported from MaMe, can be edited and selected for further work in the GridSpace Experiment Workbench.
- Improved graphical representation of models in the workspace and improvements to the workspace itself. Also, the application's layout has been rewritten to accommodate the editor panel and to allow resizing of all panels.

3. Selected realization aspects

The following technologies were used during development of the project:

Java 6
: MAD is implemented in Java 6 and continuing the project in the same language was a natural choice. Additionally, Maven is used to configure the project and its dependencies.

Google Web Toolkit
: The user interface in the application is implemented entirely in the Google Web Toolkit (version 2.4) [8]. In addition to existing widgets created fully in Java, the UiBinder templates were introduced to simplify the design of new widgets and panels.

lib-gwt-svg
: This library [10] provides support for using the SVG image format. The graphical representation of models is based on the SVG support provided by this library.

gwt-dnd
: The *gwt-dnd* library [9] provides support for drag and drop events in the Google Web Toolkit. It is responsible for elementary actions in the user interface such as adding a model to the workspace or dragging it around the workspace once added.

JAXP
: JAXP stands for the Java API for XML Processing. Operations on XML files are essential to several features available in MAD, including the import and export of applications.

4. Work organization

The work organization during the project depended on various external circumstances. Two main aspects had the impact on the organization and the number of developers in the team.

From the beginning, the project was developed by one person, due to a one year long internship between two semesters during which the engineering project should be carried out. The scope

of the project was also adjusted so that it could be developed by one person in the available time.

Additionally, MAD is developed as part of the MAPPER Project and it imposed deadlines for delivering the functionality. MAD with new functionality was presented at a project meeting in Munich (10-12 October 2012) and on the review after two years of the MAPPER project in Amsterdam (27-28 November 2012). These requirements meant that the core functionality should be finished during the first semester of the project. The second semester was used mainly to improve or create missing parts of the documentation.

The project was implemented incrementally and discussed during regular meetings with Dr. Katarzyna Rycerz and Daniel Harężlak. Notes from these meetings are available as an appendix in the documentation.

Work was initially divided into several milestones, which were based on the initial backlog. One additional validation stage was planned before the last milestone. The backlog was not fixed and was rediscussed and slightly modified during consecutive milestones.

The last milestone was used to fix small issues and bugs, and to merge the code with the main development branch.

Some small fixes in the editor were also introduced later, shortly before the second semester.

Project management tools used during development included Subversion and Git with the *git-svn* plugin.

5. Project results

The product received at the end of the project is fully functional and has filled the requirements stated at the beginning of the project. MAD with the added functionality (the editor, export improvements, etc.) is deployed in production[1] and used by scientists from different fields.

MAD has been also described in multiple deliverables of the MAPPER Project [2, 5].

Additionally, MAD with the new functionality has been promoted during the 2nd MAPPER Seasonal School in Barcelona (3-4 June 2013), which had over 20 participants.

The MAPPER Project is already finished, and it is hard to predict what will happen to MAD now. Some ideas discussed during development may still be implemented in the future, such as support for filters (as defined in the Multiscale Modeling Language[13]) in multiscale applications or additional features in the editor (e.g. syntax highlighting).

Some improvements are also possible from the development perspective—for example updating the version of GWT to the latest one (along with updating the dependent libraries).

[1] `https://gs2.mapper-project.eu/mad/`

References

[1] K. R. M. B. J. Borgdoff, C. Bona-Casas et al. A distributed multiscale computation of a tightly coupled model using the Multiscale Modeling Language. *International Conference on Computational Science, ICCS 2012*, 2012.

[2] D. H. K. Rycerz, T. Gubala et al. D 8.3 second prototype with demonstration v1.1. `http://dice.cyfronet.pl/projects/details/Mapper-files/deliverables/D8.3-SecondPrototype-CYF-v1.1.pdf`, 2011.

[3] E. C. K. Rycerz, T. Gubala et al. D 8.1 description of the architecture and interfaces v2.13. `http://dice.cyfronet.pl/projects/details/Mapper-files/deliverables/d8.1-architectureinterfaces-cyf-v2.13.pdf/view`, 2011.

[4] E. C. D. H. T. G. J. M. M. P. K. Rycerz, M. Bubak. Composing, execution and sharing of multiscale applications. *Submitted to Future Generation Computer Systems, after 1st review*, 2013. Overview of the infrastructure.

[5] R. P. K. Rycerz, J. Borgdoff et al. D 8.4 final validation and integration with external modules v1.5. `http://dice.cyfronet.pl/projects/details/Mapper-files/deliverables/d8.4-final\_validation-cyf-v1.5.pdf/view`, 2013.

[6] MAD, MaMe, EW tutorial. `http://www.mapper-project.eu/web/guest/mad-mame-ew`.

[7] GridSpace tutorial. `https://gs2.mapper-project.eu/ew/tutorials`.

[8] Google Web Toolkit. `http://www.gwtproject.org/`.

[9] The gwt-dnd library. `https://code.google.com/p/gwt-dnd/`.

[10] The lib-gwt-svg library. `http://www.vectomatic.org/libs/lib-gwt-svg`.

[11] MAPPER Memory (MaMe) User Manual. `http://dice.cyfronet.pl/projects/details/Mapper-files/MaMe_User_Manual`.

[12] The MAPPER project. `http://www.mapper-project.eu/`.

[13] MML. `https://github.com/blootsvoets/xmml`.

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science

FINAL PROJECT

EXTENSION OF A TOOL SUPPORTING DESIGN OF MULTISCALE APPLICATIONS

KONRAD STRACK

SUPERVISOR:
dr inż. Katarzyna Rycerz

Kraków 2014

NON-PLAGIARISM STATEMENT

I HEREBY DECLARE THAT I HAVE WRITTEN THIS THESIS COMPLETELY BY MYSELF, THAT I HAVE USED NO OTHER SOURCES OR RESOURCES THAN THE ONES MENTIONED, AND THAT I AM AWARE OF THE RULES OF THE PENAL CODE REFERRING TO CRIMINAL LIABILITY FOR PROVIDING FALSE EVIDENCE.

. . . . . .

SIGNATURE

Contents

1. Scope of the project

1.1. Introduction

This section describes a high-level scope of the project and provides introductory information.

1.2. Glossary of terms

Conduit A connection (dependency) between two submodels.

gMML A graphical representation of MML with respective symbols assigned to all entities in MML. Represents only part of the information about the structure of a multiscale application.

GridSpace A platform that allows users to execute virtual experiments. Provides the GridSpace Experiment Workbench [4].

GridSpace experiment or **EW experiment** A single experiment consisting of multiple code snippets, represented together as one XML file.

GridSpace Experiment Workbench or **Experiment Workbench (EW)**

Implementation A representation of a model's logic as code written in one of the available programming languages; also the parameters of the model.

Loosely coupled model An acyclic model (without any cycles between its submodels).

MAD (Multiscale Application Designer) The web application under the scope of this project, used for designing multiscale applications.

MML (Multiscale Modeling Language) The abstract language that describes submodels and connections between them. It can have several representations such as the xMML and gMML.

Multiscale application or **Application** An application which implements a multiscale process.

Multiscale process Naturally existing process (e.g. in physics or biology) which consits of parts happening on multiple scales at the same time.

MUSCLE (Multiscale Coupling Library and Environment) A communication framework designed to allow multiscale modeling of simulations. In MAD, the MUSCLE framework is used in implementation of tightly-coupled sections. The coupling of a MUSCLE application is specified in Ruby in a CxA file.

Node In the context of MAD, commonly used to name a submodel or mapper.

Registry The place (in this case—MaMe [5]) where various entities such as the submodels are stored.

Repository The place (in this case—MaMe [5]) where xMML descriptions of the multiscale applications can be stored.

Submodel instance One instance of a submodel. There may be multiple instances of the same submodel in one multiscale application.

Tightly coupled model A model with a cyclic topology (containing a loop betweeen its submodels). Communication between nodes of a tightly-coupled section can be implemented with the MUSCLE framework [6].

xMML An XML representation of MML.

1.3. Description of the problem

MAD offers limited functionality in the area of building multiscale applications. Some actions, such as editing code and parameters of models, have to be performed in GridSpace [4].

The proposed solution is to add functionality that would allow users to modify the parameters and implementations of models directly in MAD, and simplify creation of multiscale applications.

1.4. Description of users

Users can use MAD to design experiments to be run in the GridSpace Experiment Workbench or exported to one or more of available formats. They can build experiments using models available in MaMe [5], by introducing them into the workspace, modifying their parameters and providing or modifying their implementation in one of the supported languages.

Users of MAD are mostly scientists, and they do not necessarily have a strong background in computer science. They require a tool that is simple and efficient to use.

1.5. Description of the product

The final product consists of MAD with newly added functionality, including:

- A node editor with the possibility to select and edit implementation and parameters of nodes.
- Modified export/import mechanism with MAD-specific extensions.
- Improvements in the overall user experience.

The product is supposed to facilitate creation of multiscale applications. This need is reflected by the scope of changes planned for this project.

1.6. Preliminary risk analysis

Following table represents preliminary risk assessment regarding the completion of the project.

Probability here is a decimal between 0 and 1 with 0 indicating *impossible* and 1 indicating *inevitable*.

Impact is defined as a 5-point scale from 1 to 5 with 1 indicating *small disruption or delay* and 5 implicating *cancellation of the project*.

Risk description	Probability	Impact
Incompatibility between the external GWT libraries (lib-gwt-svg or gwt-dnd) with the developed functionality.	0.5	3
Development in the main branch could impact the scope of this project, and cause a change in requirements or slow down the development process.	0.3	1
Change of project requirements during the planned one-year break in project development.	0.2	2
Unsuitability of technologies already being used in MAD to perform all foreseen changes.	0.2	3
Changes (modified behavior or regressions) in supported web browsers requiring modifications in the final product.	0.1	3

2. Work organization and schedule

2.1. Introduction

This section describes the organization of work on the project and high-level plan of development (including foreseen functionality, prototyping of features, etc.).

2.2. Assumptions

In order to provide a quick feedback cycle regarding the functionality being implemented, the following assumptions about the development process have been made:

- Iterative approach to the development with regular meetings (every 2 weeks, if possible) that should serve as an assessment of work done in the previous iteration and also give possibility to adjust plans for the next iteration.

- Parts of user interface, which are not well-defined from the usability perspective, can be prototyped. Such prototypes are evaluated during meetings and accepted or discarded afterwards (possibly in favor of another solution or prototype).
- Development on the project can happen in parallel to the development in unrelated areas by other developers associated with MAD (this may include bug fixing or unrelated features). In case of such an event, codebase should be merged with the parallel changes.

2.3. Work organization

Due to the iterative approach to development and quite well defined backlog, there was no need to create formal releases and work on the project has been carried out continuously. The preliminary order of development tasks has been created in the initial backlog, but it could be and was modified along the way. For example, the order of tasks could be changed when it would positively impact the development. In some cases, it has also been extended to include ideas and tasks that were not in the initial scope of the project but served an important role in improving MAD's user interface.

MAD is being developed as part of MAPPER project, which partially enforces development schedule. MAD with new functionality was presented at a project meeting in Munich (10-12 October 2012) and on the review after two years of MAPPER project in Amsterdam (27-28 November 2012). Therefore, new functionality in MAD had to be finished before those events, creating a deadline in September 2012. Small changes and improvements (or small non-urgent bug-fixes) could be introduced also at a later stage.

2.4. Planned development

The initial milestones, each of which could theoretically span multiple iterations, were:

Milestone 1 *Analysis of existing state*
Analysis of the application and used technologies. Estimation of changes that are required in the application to allow implementation of planned features.

Milestone 2 *Refactorization of the current UI layout's implementation*
Modifications to the application's UI layout essential to introduce required components, such as an editor for code and parameters. On this early stage, a full editor is not required, but can be substituted by direct access to code and parameters of modules in simple dialog boxes provided for this purpose.

Milestone 3 *Introduction of the code editor*
Preliminary implementation of the code editor with support for parameters of modules.

Milestone 4 *Code editor for tightly-coupled sections*
Implementation of the editor widgets for tightly-coupled sections, including support for common (global) variables and merging of those variables when multiple values of the same variable are introduced into a tightly-coupled section.

Milestone 5 *Data import and export*
Import and export of the experiment to xMML and export to Experiment Workbench, where all exports and imports take into account the possibility to modify the code and parameters directly in MAD and include such modifications.

Validation Validation of implemented features.

Milestone 6 *Bug-fixing and improvements resulting from final testing*
Required changes and bug fixes for problems noticed after the main functionality has been implemented.

2.5. Summary

The main focus was concentrated around modifications to the user interface, some of which required a significant amount of time to be performed. Several components had to be redesigned using UiBinder, in order to take advantage of some of the recent features available in GWT and mainly to facilitate design of the interface.

However, all tasks related to data import and export, as well as opening the project in the GridSpace Experiment Workbench, were also not overlooked.

3. Work progress

3.1. Introduction

The technologies and application modules where well-defined from the very beginning. MAD is a multi-module Maven project built in Java with user interface created with GWT, *lib-gwt-svg* for SVG support and *gwt-dnd* for drag-and-drop support. At the time when the project started, the version of GWT used was 2.3, and possibility to upgrade to the already available version 2.4 was limited by the lack of official support for the newer version in libraries dependent on GWT.

The implementation process was preceded by an analysis phase, essential to gather necessary knowledge about the architecture of MAD, its dependencies and development practices.

3.2. Milestones

The following sections describe what has been accomplished in the consecutive milestones.

3.2.1. Milestone 1

Milestone 1 was an introductory phase used to discover application's stack of technologies and assess which areas and components of the application required modification.

Usage of additional UI libraries built on top of GWT, such as *Ext GWT* from Sencha (currently called *GXT*) or *Smart GWT*, has been proposed during this milestone. However, finally it has been decided that complexity of integration with other libraries (such as *gwt-dnd*) would be significantly increased. Such decision would also decrease general performance of the application from the users' perspective (due to higher hardware requirements needed to execute the application in their browsers).

3.2.2. Milestone 2

Most of the changes in the first milestone were needed to build a necessary background for the future development. Usage of UiBinder has been considered as a viable substitute for Java-based creation of the user interface. First efforts were made to determine a proper way of including UiBinder templates in the application.

- A new *HTMLLayout* has been added in addition to two preexisting layouts, *SimpleLayout* and *HorizontalLayout*. It has been constructed with UiBinder and based on GWT's HTMLPanel.
- Information about available implementations has been attached to models imported from the repository, in order to allow further editing of those implementations in MAD.
- Added the possibility to display dialog boxes with first available implementation and that implementation's parameters for each model.

3.2.3. Milestone 3

Second milestone concentrated mostly on the code editor in order to provide basic capability to modify implementations of models and their parameters.

Since the introduction of UiBinder in the previous milestone was successful, further integration continued also in this milestone.

- Application's layout was edited to include the editor.
- Introduction of a tabbed editor for single modules (no support for tightly-coupled sections was provided at this point) activated on a node's selection. Layout of *CodeEditor* was created using UiBinder templates.
- Highlighting nodes on the workspace when a corresponding tab in the editor is selected.
- Support for multiple implementations of models with the possibility to select implementation in the code editor.
- Refactorization of parts of codebase responsible for the creation of nodes.

3.2.4. Milestone 4

- Detection of nodes belonging to a tightly-coupled section.

- Highlighting nodes of a tightly-coupled section present in the workspace. Widget for representation of nodes was refactored to use UiBinder, in order to improve highlighting.
- Implementation of the editor for tightly-coupled sections, which is added as a separate tab in the existing editor when at least one node corresponding to a tightly-coupled section is detected.
- Automatically saving edited parameters and code on every value change in corresponding input fields.
- Support for displaying and editing global parameters of tightly-coupled sections.
- Dialog box for merging global parameters displayed every time when a new node is added to the workspace and a difference between its global parameters and global parameters of existing tightly-coupled section is detected.
- Synchronization between values of global parameters among all nodes in a tightly-coupled section.

Starting from this milestone, UiBinder became the main templating mechanism for new widgets and views in MAD.

At the end of this milestone, few previously unforeseen changes to the layout were performed to allow users to adjust application's layout if more space on the workspace is required and to improve the general user experience.

- Previously used HTMLPanel for the application layout has been replaced with SplitLayoutPanel, in order to allow resizing of the editor and workspace panels.
- Automatically adjusting editor size when the editor panel is resized.
- Possibility to scroll the workspace when nodes present in the workspace use more space than is available in the visible working area.
- Automatically activating the newly created editor when a new node is added to the workspace.
- Activating the corresponding editor when user clicks on an image of a node present in the workspace.

3.2.5. Milestone 5

Most efforts in the fourth milestone were put into exporting and importing models.

- Possibility to export edited implementations to xMML was added to the *ibuilder-xmml* module. This behavior was, however, made configurable and implementations from the repository can be used instead.
- Import of previously exported xMML-formatted files gained support for importing implementations.
- Support for exporting edited implementations to the GridSpace Experiment Workbench.

3.2.6. Validation

In general, the results of the validation phase were positive, but some minor problems were found (mostly small UI issues missed during development). All those issues were scheduled to be solved in the following milestone.

3.2.7. Milestone 6

- Menu panel was rewritten to use UiBinder.
- Small changes in the layout of the editor to correct resizing issues and improve displaying of implementations and global variables.
- Clearing editor panel when workspace is cleared.
- Automatically opening all editors for newly loaded or imported graphs.
- Various small bugfixes.

After this milestone, there was one outstanding problem with the layout of the editor. Due to the complexity of GWT, some problems arose with correct resizing of the editor and other elements (such as text fields) inside. This problem has been finally solved during the break in the project.

3.3. Testing

The chosen approach to testing depends highly on the module of the application.

3.3.1. ibuilder-gwt

All testing of user interface in *ibuilder-gwt* has been performed manually. Unit tests are used only for non-GWT components and are executed with JUnit.

Lack of automated UI testing could not be the best approach for an application with a highly sophisticated interface, but fortunately the user interface of MAD is relatively simple, and the number of actions which can be performed by users is limited. As a result manual testing becomes a quite effective approach, especially when the interface and its behavior are iteratively prototyped.

It also seems that using automation tools such as Selenium would significantly increase the time spent on development, because additional time would be needed to keep UI tests in sync with the rapidly changing interface. Moreover, addition of Selenium or similar tools would also impose additional work on other developers after the end of the project.

Also, due to the nature of GWT, unit and integration testing of interfaces built with GWT is not a straightforward task. A standard approach to unit testing with JUnit or a similar test execution suite is not possible in the majority of cases. GWT widgets often depend on native JavaScript

code and cannot be instantiated outside of a browser or a similar environment simulating a browser.

One possible approach is to use GwtTestCase which allows to unit test GWT widgets. It is done, however, by launching a simulated browser environment in which such tests are executed. Tests executed in such manner are therefore slow, and the development time with such tests may increase significantly.

Using a library such as the GwtMockito could be a better approach, but we are constrained by the version of GWT which is used by MAD (it currently is 2.4). Once GWT in MAD is updated to version 2.5.1, the possibility to use GwtMockito can be revisited. GwtMockito is built on top the powerful Mockito library and offers API to mock GWT widgets and as a result execute tests with JUnit and no performance penalty as is the case with GwtTestCase.

3.3.2. ibuilder-api and ibuilder-xmml

Those two modules can be rather easily covered by unit and integration tests. The most significant parts of code (e.g. responsible for xMML import and export) are covered, and the tests are executed with JUnit.

4. Division of work

Due to external constraints this project has been implemented by one developer. From the very beginning there was a one year long break planned between two semesters, during which this project had been scheduled to be carried out.

The annual break had a significant impact also on the order of development tasks and creation of the documentation. The requested functionality had to be implemented before specified deadlines, whereas user documentation and parts of process and technical documentation could be finished at a later stage.

Functionality described in this project was developed in parallel to normal development tasks in the main development branch. However, it had only small impact on the development process and created the need for additional testing after the branches were merged.

After the final merge of the new functionality into the main development branch, all development tasks were again carried out by the original development team.

5. Project summary

Development of MAD has been constrained by deadlines set by the MAPPER Project schedule, and external circumstances such as the one year long break in the project. Nonetheless, the project has been finished successfully and it allowed me to gather experience in many areas, including software engineering and web development, and knowledge about specific libraries such as the Google Web Toolkit.

Not every aspect of the project had been precisely specified at the beginning, and it left space for experimentation and adjustment. If some feature turned out not to be useful, it could easily be removed. If there was an idea to improve something that had not been planned earlier, it could be added to the project. The focus of the project was to improve the user experience of MAD's users, and this approach turned out to be ideal for such kind of project.

There were some obstacles along the way, such as problems with versions of the GWT Development Plugin for Firefox or the general slowness of the plugin in Chrome, which had some impact on the speed of development. However, these problems were not significant enough to have a negative impact on finishing the project.

Some aspects of creating layouts in the Google Web Toolkit also turned out to be suprisingly complex. There were many constraints and details which required a significant amount of time to understand and avoid various issues.

The MAPPER Project is already finished, and it is hard to predict what will happen to MAD now. From the development perspective, in my opinion, MAD would certainly gain from updating GWT to version 2.5.1 (which may require careful testing and updating the *gwt-dnd* and *lib-gwt-svg* libraries) and from more extensive use of UiBinder templates to better separate UI from the business logic.

From the usability perspective, it could gain from introducing even deeper integration with MaMe (e.g. managment of nodes and applications in the repository). Furthermore, the editor could also be improved by adding features such as syntax highlighting. Unfortunately, there was not enough time to work on these ideas and they were out of the scope of the project.

MAD is currently deployed in production with changes introduced by this project and is used by scientists working in various fields including fusion modeling, nano-materials, simulations of irrigation canals or medical simulations. The use of web technologies, such as the Google Web Toolkit, made MAD easily available to all interested users, without the need to install any specific software.

MAD with the new functionality was presented at a project meeting in Munich (10-12 October 2012) and on the review after two years of the MAPPER project in Amsterdam (27-28 November 2012), and has been promoted during the 2nd MAPPER Seasonal School in Barcelona (3-4 June 2013) with over 20 participants.

Appendices

A. Meeting minutes

This section lists meeting minutes from meetings held during development of the project. Dr. Katarzyna Rycerz was present at all meetings. Daniel Harężlak has also participated in most of them.

A.1. Meeting on 17/04/2012

Previous work has been presented.

Observations:

- All modules of the same type share instances of implementations. Performing a deep copy may be required.
- Global parameters should be merged when nodes are added to the workspace.

Important tasks and comments about next iterations:

- Add detection of a tightly-coupled section. Nodes belonging to a tightly-coupled section can be detected by checking the interpreter of their implementations (should be 'muscle'). Mark all nodes belonging to a tightly-coupled section.
- Add scrolling of the editor and list of parameters (perhaps just by adding a ScrollPanel?). Figure out if its reasonable to place parameters in multiple columns.
- Merge global parameters in a popup window / dialog box if there is a collision.
- Automatically save parameters and implementations (listeners?).
- Display editors for models in a tightly-coupled section in one tab, perhaps with sub-tabs.
- Use StaticController to get the editor.
- Add the possibility to choose implementation.
- Highlight the node when a corresponding tab in the editor panel is chosen.

A.2. Meeting on 08/05/2012

Previous work has been presented.

Tasks to start or continue before the next meeting:

- Check and implement export to xMML, etc. (with a configuration parameter specifying whether data should be taken from MaME or from edited nodes).

- Think about the possibility to make nodes look better (some of them have very many ports).
- Correct resizing of panels.
- Add the possibility to scroll the workspace.
- Highlight nodes in a more visible and clearer way.
- Highlight tightly-coupled sections.
- Don't activate the editor when the node is still in the toolbox.
- Try to use SplitLayoutPanel for the layout.
- Copy implementations when node is moved from the toolbox to the workspace.
- Update all global parameters after merging.

A.3. Meeting on 30/05/2012

Previous work has been presented.

Tasks to start or continue before the next meeting:

- Improve editor resizing.
- Correct resizing of the menu panel (or disable). Perhaps rewrite to UiBinder.
- Add export to EW.
- Add export to xMML (with extensions, additional parameters). Check if they can be serialized automatically.
- Remove editor when the corresponding node is removed from the workspace.
- Activate the editor when a node is dragged into the workspace.
- Think about the possibility to introduce a common hierarchy between modules (e.g. *Implementation* in *ibuilder-gwt* could extend *Implementation* in *ibuilder-api*).

A.4. Meeting on 21/06/2012

Previous work has been presented.

Main points of the meeting:

- Common hierarchy between modules will not be introduced due to time constraints.
- Next meeting will probably be the last one before the break. All functionality should be completed before that meeting.
- Continue work on export to EW.

- Rewrite menu to use UiBinder.
- Improve resizing of the editor.

A.5. Meeting on 11/07/2012

Main points of the meeting:

- A problem with export to EW (`Error Code: GridSpaceConnectionError, additional info:GridSpace connection URL: https://gs2.mapper-project.eu:8443/ew/gridspace`) was caused by self-signed certificates, which were not being accepted. The problem has been fixed by Daniel Harężlak.
- The product is considered to be feature complete.
- Further work on documentation is scheduled for next year, as well as any possible improvements to the product.
- Code will be merged to the SVN trunk.

A.6. Meeting on 01/10/2013

Main points of the meeting:

- Work on the documentation will be continued. Precise deadlines are not yet known.
- Patches to fix outstanding issues with the layout of the editor have been merged successfully.

6. References

[1] D. H. K. Rycerz, T. Gubala et al. D 8.3 second prototype with demonstration v1.1. http://dice.cyfronet.pl/projects/details/Mapper-files/deliverables/D8.3-SecondPrototype-CYF-v1.1.pdf, 2011.

[2] E. C. K. Rycerz, T. Gubala et al. D 8.1 description of the architecture and interfaces v2.13. http://dice.cyfronet.pl/projects/details/Mapper-files/deliverables/d8.1-architectureinterfaces-cyf-v2.13.pdf/view, 2011.

[3] R. P. K. Rycerz, J. Borgdoff et al. D 8.4 final validation and integration with external modules v1.5. http://dice.cyfronet.pl/projects/details/Mapper-files/deliverables/d8.4-final_validation-cyf-v1.5.pdf/view, 2013.

[4] GridSpace tutorial. https://gs2.mapper-project.eu/ew/tutorials.

[5] MAPPER Memory (MaMe) User Manual. http://dice.cyfronet.pl/projects/details/Mapper-files/MaMe_User_Manual.

[6] The MUSCLE library. http://apps.man.poznan.pl/trac/muscle.

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science

FINAL PROJECT

EXTENSION OF A TOOL SUPPORTING DESIGN OF MULTISCALE APPLICATIONS

KONRAD STRACK

SUPERVISOR:
dr inż. Katarzyna Rycerz

Kraków 2014

NON-PLAGIARISM STATEMENT

I HEREBY DECLARE THAT I HAVE WRITTEN THIS THESIS COMPLETELY BY MYSELF, THAT I HAVE USED NO OTHER SOURCES OR RESOURCES THAN THE ONES MENTIONED, AND THAT I AM AWARE OF THE RULES OF THE PENAL CODE REFERRING TO CRIMINAL LIABILITY FOR PROVIDING FALSE EVIDENCE.

. . . . . .

SIGNATURE

Contents

1. Introduction

The goal of this document is to describe the requirements of the new functionality for the Multiscale Application Designer and to put it in the context of the existing functionality and services related to MAD.

Please not that this document does not describe all technical details of MAD—described are only parts essential to place in the correct context the functionality introduced by this project.

1.1. Glossary of terms

Conduit A connection (dependency) between two submodels.

gMML A graphical representation of MML with respective symbols assigned to all entities in MML. Represents only part of the information about the structure of a multiscale application.

GridSpace A platform that allows users to execute virtual experiments. Provides the GridSpace Experiment Workbench [5].

GridSpace experiment or **EW experiment** A single experiment consisting of multiple code snippets, represented together as one XML file.

GridSpace Experiment Workbench or **Experiment Workbench (EW)**

Implementation A representation of a model's logic as code written in one of the available programming languages; also the parameters of the model.

Loosely coupled model An acyclic model (without any cycles between its submodels).

MAD (Multiscale Application Designer) The web application under the scope of this project, used for designing multiscale applications.

MML (Multiscale Modeling Language) The abstract language that describes submodels and connections between them. It can have several representations such as the xMML and gMML.

Multiscale application or **Application** An application which implements a multiscale process.

Multiscale process Naturally existing process (e.g. in physics or biology) which consits of parts happening on multiple scales at the same time.

MUSCLE (Multiscale Coupling Library and Environment) A communication framework designed to allow multiscale modeling of simulations. In MAD, the MUSCLE framework is used in implementation of tightly-coupled sections. The coupling of a MUSCLE application is specified in Ruby in a CxA file.

Node In the context of MAD, commonly used to name a submodel or mapper.

Registry The place (in this case—MaMe [8]) where various entities such as the submodels are stored.

Repository The place (in this case—MaMe [8]) where xMML descriptions of the multiscale applications can be stored.

Submodel instance One instance of a submodel. There may be multiple instances of the same submodel in one multiscale application.

Tightly coupled model A model with cyclic topology (containing a loop between its submodels). Communication between nodes of a tightly-coupled section can be implemented with the MUSCLE framework [10].

xMML An XML representation of MML.

1.2. Description of the problem

The functionality offered by MAD is limited, and some tasks are delegated to other services, such as the MaMe or GridSpace Experiment Workbench. Such division of work between multiple services makes MAD not as convenient to use, as it could be. In order for users to work more efficiently, it is essential to provide tools and solutions in MAD that would make it a central point for creation of multiscale applications. Such new solutions would loosen the dependence on other services for carrying out simple tasks such as editing implementations of models.

The proposed solution is to add functionality that would allow users to modify the parameters and implementations of models directly in MAD, and simplify creation of multiscale applications.

1.3. Description of users

Users can use MAD to design experiments to be run in the GridSpace Experiment Workbench [5] or exported to one or more of available formats. They can build experiments using models available in the MAPPER Memory (MaMe) [8], by introducing them into the workspace, modifying their parameters and providing or modifying their implementation in one of the supported languages.

Users of MAD are mostly scientists, and they do not necessarily have a strong background in computer science. They require a tool that is simple and efficient to use.

1.4. Description of the product

Multiscale Application Designer (MAD) is a web application designed to facilitate creation of multiscale applications by using predefined reusable single scale submodels and mappers.

The final product consists of MAD with newly added functionality, including:

- A node editor with the possibility to select and edit implementation and parameters of nodes.
- Modified export/import mechanism with MAD-specific extensions.
- Improvements in the overall user experience.

The product is supposed to facilitate creation of multiscale applications. This need is reflected by the scope of changes planned for this project.

1.4.1. Functional requirements

Requirements related to the user interface:

- Additional panel for the editor, which:
  - Displays tabs for each node present in the workspace.
  - Allows to edit loosely coupled models.
  - Allows to edit both implementations and parameters of nodes.
  - Allows to choose a default implementation for a particular model.
  - Allows to edit tightly-coupled sections, and groups models belonging to such sections together.
  - Handles synchronization of global parameters for tightly-coupled sections.
- Improved (better visibility) graphical representation of selected nodes and nodes belonging to a tightly-coupled section.

Requirements related to export and import of multiscale applications:

- Introduction of MAD-specific extensions to the xMML [9] export that allow to save and restore implementations and parameters of nodes.
- Possibility to choose the implementation to be used for exports to GridSpace Experiment Workbench.

1.4.2. Non-functional requirements

Requirements related to quality:

- Keep the computing power and memory footprint requirements low, so that the application can be used on less modern hardware in a satisfactory way.
- Produced user documentation should be easily understandable.

Systems and technologies:

- It has to be possible to host MAD in Linux-based environments.

- MAD should be fully functional in the following browsers:
  - Chrome - versions 20 and above
  - Firefox - versions 12 and above (including the ESR releases 17.x and 24.x)
  - Internet Explorer - versions 9 and above
- MAD should work in the aforementioned browser in the following operating systems:
  - Linux
  - Mac OS X
  - Windows (XP, Vista, 7)
- Google Web Toolkit will be the basis for all user interface development.

1.5. Risk analysis

Table 1 represents the summary of risks.

Probability is represented by a decimal between 0 and 1 with 0 indicating *impossible* and 1 indicating *inevitable*.

Impact is defined as a 5-point scale from 1 to 5 with 1 indicating *small disruption or delay* and 5 implicating *cancellation of the project*.

Risk description	Probability	Impact
Incompatibility between the external GWT libraries (lib-gwt-svg or gwt-dnd) with the developed functionality.	0.5	3
Development in the main branch could impact the scope of this project, and cause a change in requirements or slow down the development process.	0.3	1
Change of project requirements during the planned one-year break in project development.	0.2	2
Unsuitability of technologies already being used in MAD to perform all foreseen changes.	0.2	3
Changes (modified behavior or regressions) in supported web browsers requiring modifications in the final product.	0.1	3

Table 1: Summary of risks

2. Architecture

2.1. Overview

MAD is one of many applications, libraries and systems created as part of the MAPPER project and as such it communicates with many other systems and uses data provided by them.

MAPPER Memory (MaMe) [8] contains two very important data collections: Models Registry and Experiments Repository. Models Registry contains all models available in MAD for constructing experiments. Experiments Repository allows to store and retrieve experiments in xMML format (experiments created or modified using MAD in particular) [9]. Data from both Models Registry and Experiments Repository is retrieved and saved using REST API[1] provided by MaMe.

Experiments created with MAD can be also exported as Gridspace Experiment Workbench experiments [5] or directly opened in GridSpace Experiment Workbench.

For the overview of the architecture please also see [2].

2.2. Modules

MAD's source code is represented as a Maven project named *ibuilder*. The Maven project will be, therefore, referred to as *ibuilder* in this section, but effectively *ibuilder* contains all components which add up to create MAD.

The *ibuilder* module is a parent of three other modules: *ibuilder-api*, *ibuilder-xmml* and *ibuilder-gwt*.

Figure 1 shows how MAD is divided into modules and the modules' relation with MaMe.

2.2.1. ibuilder-api

The *ibuilder-api* module contains generic implementation, definitions and structures that can be built upon to create the complete application. It serves as a core of MAD and theoretically could be reused to build another implementation of the user interface, alternative to *ibuilder-gwt*.

2.2.2. ibuilder-xmml

The *ibuilder-xmml* is responsible for handling all operations related to reading and writing xMML-formatted files. It also contains tests related to xMML marshalling and unmarshalling.

[1] For full description of provided API please refer to respective *API Help* available at `http://gs2.mapper-project.eu/mame`

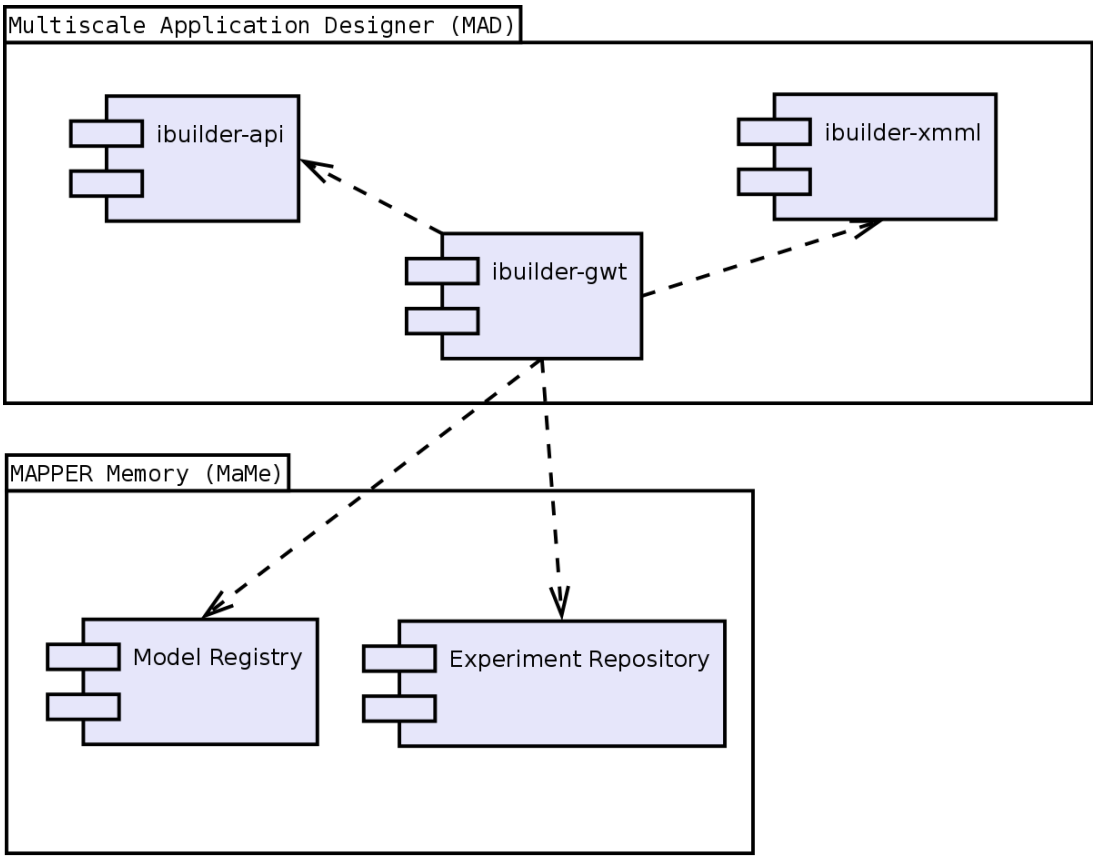

Figure 1: An overview of MAD's modules and their the relation to MaMe.

2.2.3. ibuilder-gwt

The *ibuilder-gwt* contains all logic related to the user interface. By using *ibuiler-api* and *ibuilder-xmml* it builds the UI and makes the core functionality available to users. It is also responsible for communication with MaMe and GridSpace.

This module depends on the Google Web Toolkit [6] and requires the GWT Development Plugin installed in the browser for development purposes.

3. Technology

3.1. Data representation

3.1.1. MML and its gMML representation

Multiscale Modeling Language (MML) [9] is designed to represent multiscale applications as composed of multiple single scale models with well-defined coupling between the submodels. Applications represented in such way can be easily analyzed and executed.

MML allows to specify implementation, scale and computational requirements of specific submodels.

MML defines few basic elements, which are important as far as implementation of MAD is concerned. Those elements have their respective graphical representations specified. The graphical representation of MML is known as gMML.

Submodels
Submodels are self-contained reusable components connected to other elements by their input and output ports. A single submodel may specify its implementation and computational requirements. In theory, submodels are unaware of other submodels, and user-defined filters should be applied to conduits if there are differences in data representation between connected submodels.

Submodels are represented in gMML as rectangles. Figure 2 shows a sample submodel as represented in MAD.

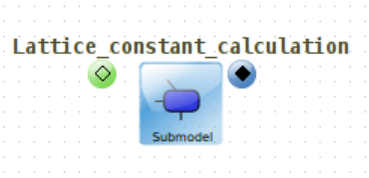

Figure 2: The graphical representation of a submodel in MAD.

Conduits
Conduits are simple unidirectional point to point connections between other elements of an MML diagram.

Representation of extremities of conduits differs based on their usage. Figure 3 shows these representations for the receiving (input) end, and Figure 4 shows representations for the sending (output) end.

In MAD those representations are permanently attached to models and depend on the port type. Creation of a coupling is, therefore, as simple as connecting two ports in the workspace. Representations of input and output ports in MAD are shown on Figure 5 and Figure 6 respectively.

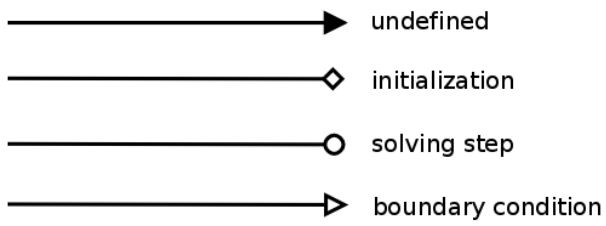

Figure 3: Symbols for extremities of conduits on the receiving side.

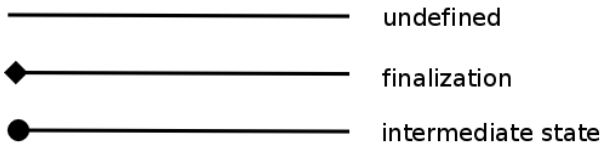

Figure 4: Symbols for extremities of conduits on the sending side.

Mappers
Mappers are simple multi-port objects which slightly resemble submodels. However, mappers only task is data transformation. They can be used to simplify and optimize coupling or to build complex connections between submodels.

There are two types of mappers defined in MML: *fan-in* and *fan-out*. *Fan-in* mappers contain multiple input ports and one output port which returns a function of the inputs. *Fan-out* mappers contain single input and multiple output ports which return respective functions of the input port.

Mappers are represented in gMML as hexagons. Figure 7 shows a sample mapper as represented in MAD.

Filters
Filters are similar to conduits and can be thought of as types of conduits which additionally perform stateful data transformations. They are used to couple submodels that cannot be coupled directly due to e.g. different data representation expectations on the input port of one of the submodels.

Filters are represented in gMML as rounded squares in the middle of conduits. Filters are currently not supported in MAD.

3.1.2. xMML

xMML is an XML format specification of the Multiscale Modeling Language. For the detailed specification of xMML, please see [9].

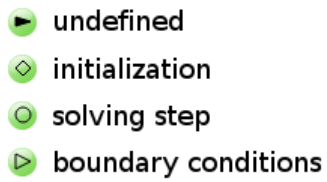

Figure 5: Representation of ports on the receiving (input) side in MAD.

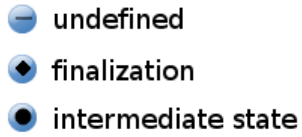

Figure 6: Representation of ports on the sending (output) side in MAD.

3.2. Libraries

MAD uses several external libraries, in order to provide its functionality. Dependence on other libraries differs significantly between modules of the application.

For example, each of the *ibuilder-api* and *ibuilder-xmml* modules contains only one such dependency. In case of the *ibuilder-gwt*, however, there are several libraries used to build the user interface.

3.2.1. Google Web Toolkit

The Google Web Toolkit library is used construct and manage the user interface of MAD. Due to version constraints enforced by usage of other libraries dependent on GWT, the version 2.4 is used currently.

There are two libraries dependent on GWT which provide important functionality not available directly in GWT—*lib-gwt-svg* and *gwt-dnd*.

Most of the new widgets added to the MAD's codebase were implemented with the use of the UiBinder [7]. For the full documentation and examples regarding the UiBinder, please visit: `http://www.gwtproject.org/doc/latest/DevGuideUiBinder.html`.

3.2.2. lib-gwt-svg

This library is responsible for displaying vector graphics (SVG) in the application. It is used mostly for displaying SVG images representing nodes, ports and conduits.

lib-gwt-svg was already chosen for MAD at the beginning of the project.

See: `http://www.vectomatic.org/libs/lib-gwt-svg`.

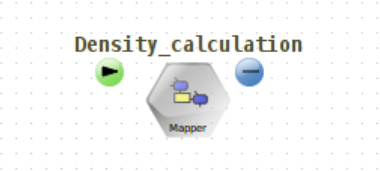

Figure 7: The graphical representation of a mapper in MAD.

3.2.3. gwt-dnd

This library is responsible for providing and handling drag and drop events. Some actions, such as adding a node to the workspace or moving nodes on the workspace, depend on it.

gwt-dnd was already chosen for MAD at the beginning of the project.

See: `https://code.google.com/p/gwt-dnd/`.

3.3. Data formats

3.3.1. SVG (Scalable Vector Graphics)

Used to provide the graphical representation of nodes, ports and conduits. Files are named with the `.svg` extension.

Requires the *lib-gwt-svg* library.

3.3.2. XML

MAD has the possibility to export and import XML files. These files are named with the `.xml` extension.

The Java API for XML Processing (JAXP) is used to handle XML files.

3.3.3. UiBinder

The UiBinder [7] from GWT describes the layout of many widgets and panels in the application's user interface. These files are named with the `.ui.xml` extension.

This format is handled automatically by the Google Web Toolkit.

3.4. Requirements

3.4.1. Hardware requirements

MAD should run on most modern hardware configurations. Following requirements assume ideal operating system conditions (no memory or CPU-heavy applications running at the same time). These estimates may be increased by requirements specific to the operating system or the web browser, which are used to run MAD.

Minimal requirements, below which the possibility to use MAD may be reduced, are:

- 256MiB of RAM.
- Integrated graphic card with at least 32MiB of memory.
- Screen resolution of at least 1024 x 768.

Recommended hardware

- At least 512MiB of RAM.
- Integrated graphic card with at least 64MiB of memory.
- Screen resolution of at least 1920 x 1080.

Please note the importance of the high screen resolution. Despite the workspace's ability to scroll its contents, building complex applications is much less pleasant when not all nodes of the application can be displayed at the same time.

3.4.2. Software requirements

MAD runs in a web browser, and it should be possible to use it in most modern browsers and most operating systems, even if they are not officially supported.

MAD has been tested on Linux and Windows (Windows Vista and Windows 7) platforms.

Supported browser are:

- Chrome / Chromium - due to Chrome's frequent automatic updates, only three last versions are supported at any given time, but theoretically any version from 20 upwards should be able to run it.
- Firefox - only three last versions from regular releases of Firefox are supported. Additionally, MAD should work in the Extended Support Releases 17.x and 24.x.
- Internet Explorer - versions 9 and above.

3.4.3. Libraries and dependencies

There are no specific requirements regarding libraries or other dependencies that should be provided by the user. MAD runs as an application in a web browser, and therefore all required

dependencies are downloaded to the browser when the application is accessed.

3.4.4. Knowledge and skills

Usage of MAD itself doesn't require any specific skills. However, knowledge about designing multiscale applications may be required, in order to create a working application.

Some programming knowledge may be needed to modify implementations of nodes. Programming languages used depend on a particular application and may include Java, C, C++, Fortran, Bash, and more.

3.4.5. Requirements for developers

Development of applications based on GWT requires significant amounts of RAM and CPU power. A machine with less than 4GiB of RAM and a mid-range CPU may not be suitable.

Usage of the GWT Development Plugin can cause different problems in different browsers. In Firefox, it provides quite fast page reloading, but may end up with an OutOfMemory error after the page is refreshed a few times. In Chrome, page reloading times are noticeably worse, but it does not cause the OutOfMemory error.

Recommended hardware for GWT development would include at least 8GiB of RAM and a fairly fast 4-core CPU.

This, of course, concerns only development of the *ibuilder-gwt* module. There are no specific requirements for development of the *ibuilder-api* and *ibuilder-xmml*.

3.5. Safety and security

3.5.1. Safety

MAD has been tested on the reference hardware with parameters not worse than the minimal requirements. Under such conditions, it should not cause any harm to the hardware. Running MAD on hardware worse than specified in the minimal requirements may cause unexpected behavior.

However, the software is provided as is, and the authors shall not be held liable for any damages or problems resulting from the usage of MAD.

3.5.2. Security

MAD doesn't include any specific security mechanism. The URL at which MAD is available is guarded by a 128-bit SSL certificate. However, MAD is a completely open solution with no access restrictions.

Other services such as GridSpace provide their own security and authorization mechanisms.

4. Testing

Because this project only adds on the previous functionality of MAD, all existing testing mechanisms were in place and could be used against the new code. During the project, no tests were performed on parts of the application not directly connected to the current development. Of course, the application was tested for possible regressions.

4.1. The *ibuilder-api* module

The *ibuilder-api* module was almost unaffected by the development. The only change was a small API change related to the layout of the application. This module has been tested only manually and indirectly, as part of tests of the *ibuilder-gwt* and *ibuilder-xmml*.

4.2. The *ibuilder-xmml* module

The *ibuilder-xmml* module contains integration tests of marshalling and unmarshalling of the xMML representation of an application. These tests have been modified to handle properly the presence of MAD-related extensions to the xMML (addition of the implementations and ports).

This module has been also tested indirectly by testing export and import capabilities of MAD via the graphical interface provided by the *ibuilder-gwt* module.

4.3. The *ibuilder-gwt* module

Testing of the *ibuilder-gwt* module is the most challenging part. There is no easy way to create unit tests for GWT widgets.

Use of the GwtTestCase is one of the possible solutions, but the tests are, in fact, run in a simulated browser environment. The execution speed of such tests is much worse than in case of normal JUnit tests. GwtTestCase was considered as an option for complex widgets, but in the end, due to its drawbacks, it has not been used anywhere in the project.

Some solutions—the GwtMockito project, for example—were not available at the time of development. GwtMockito is still not suitable for MAD, due to its requirement of GWT 2.5.1.

Solutions for automation of UI tests, such as Selenium, were also excluded. Good as they are, such tools would significantly impact the development speed (due to the need of keeping tests in sync with the interface), and the project was carried out by only one developer. Usage of Selenium is also not that simple with the Google Web Toolkit. GWT creates complex hierarchies of elements in the DOM tree, and these hierarchies also vary, even between similar widgets. Addition of Selenium or similar tools would also impose additional work on other developers after the end of the project.

In the end, all tests of the user interface were carried out manually after every milestone of the project.

The main testing areas included:

- Nodes:
  - Verify that nodes are loaded from MaMe.
  - Verify that nodes can be added and deleted from the workspace.
  - Check if ports can be correctly joined by conduits and if conduits can be removed.
- Editor (after the corresponding functionality has been added):
  - Verify that adding a node to the workspace opens a corresponding editor tab.
  - Check if nodes from a tightly-coupled section are grouped under one tab.
  - Check if edited implementations and parameters can be edited.
  - Check if global parameters are synchronized.
  - Verify that selecting a tab selects the corresponding node in the workspace.
  - Verify that deleting a node from the workspace, removes the corresponding tab from the editor.
- Import and export:
  - Check the correctness of the export to XML.
  - Check the correctness of the export to xMML.
  - Check the correctness of the export to EW experiment file.
  - Check that the application can be opened in the GridSpace Experiment Workbench via menu.
- Compatibility with the *gwt-dnd* library:
  - Test if nodes can be moved from the node panel to the workspace.
  - Check if nodes can be dragged around on the workspace.
  - Check that dragging actions do not take precedence over the click actions.
- Compatibility with the *lib-gwt-svg* library:
  - All images for nodes, ports and conduits should be visible.
  - Images should not cover other elements in a way that would create an unclickable area (e.g. nodes covering parts of conduits).

5. Implementation

MAD is developed entirely in Java 6. To see all major libraries used in MAD, please refer to the list of libraries in section 3.2.

This section does not describe the whole scope of MAD. It concentrates on ares of the application developed during this project.

5.1. The main application layout

There were two layouts implemented for the application. The `SimpleLayout` and the `HorizontalLayout`. The first layout is based on a `FlowPanel` with directly set sizes (in percents) of widgets attached to this layout. The second layout is just a simple wrapper for the `HorizontalPanel`.

These two layouts have been replaced by the new `HTMLLayout`, which uses a UiBinder template, and therefore it is much more configurable than previous layouts. The template for this class can be found in `resources/cyfronet/gs/ibuilder/client/layout/HTMLLayout.ui.xml`.

The layout is based on a `SplitLayoutPanel`. The choice of `SplitLayoutPanel` over the `DockLayoutPanel` is intentional and allows to resize respective panels added to the `HTMLLayout`.

5.2. The editor

The editor is the biggest single feature added in this project. It uses UiBinder templates whenever it is possible and useful.

The `EditorViewImpl` class extends `ResizeComposite` from GWT in order to provide correct resizing to its children. It can contain multiple editors, one per tab. When the first node belonging to a tightly-coupled section is added to the workspace, a `TightlyCoupledEditor` is added to the `EditorViewImpl` and a `SimpleEditor` corresponding to that node is added to the `TightlyCoupledEditor`.

The inner `GlobalParameter` class is used for synchronization of parameters between all nodes belonging to one tightly-coupled section. It is achieved with the GWT event mechanism.

Figure 8 presents the class hierarchy of the most important classes related to the editor.

5.3. UiBinder

Usage of UiBinder greatly facilitates designing layouts. Several widgets have been rewritten to use UiBinder templates during the development process. Some examples are

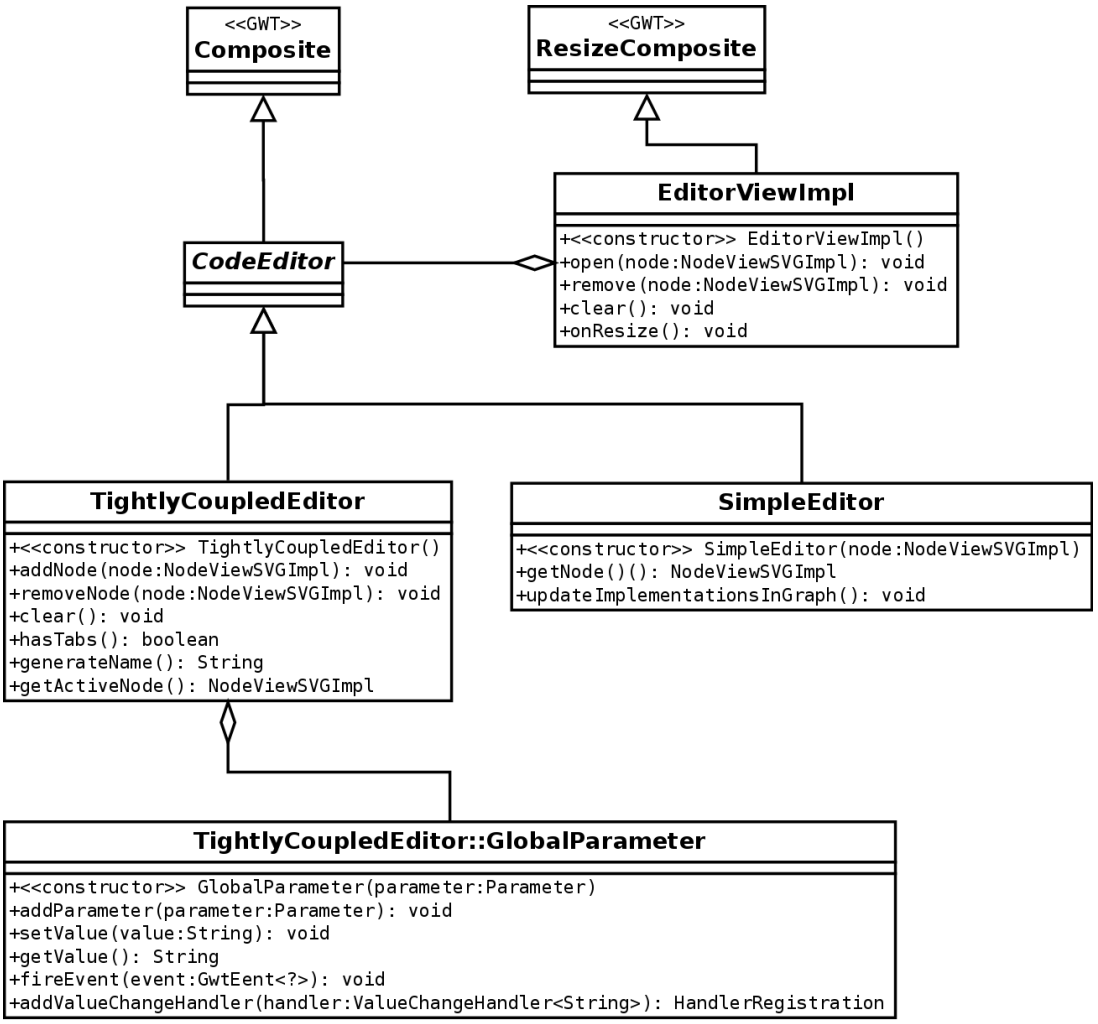

Figure 8: A class diagram with classes related to the editor.

`NodeViewSVGImpl` (which represents the layout of a single node) and `MenuView` (which represents the menu on the right).

The following snippet shows the template for a very simple panel (in this case, the `EditorViewImpl.ui.xml`):

```
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'
             xmlns:g='urn:import:com.google.gwt.user.client.ui'>

    <ui:style>
        .tabPanel {
            margin: 2px;
        }
    </ui:style>

    <g:TabLayoutPanel barUnit="PX" barHeight="26"
    addStyleNames="{style.tabPanel}"/>

</ui:UiBinder>
```

The `<ui:style>` blocks are used widely in the application, in order to facilitate applying CSS styles (styles are placed in UiBinder templates instead of adding them manually in the Java code).

5.4. MAD-specific data in exported files

In order to import and export from MAD all details about a multiscale application, MAD-specific information is added to the xMML file.

The following snippet shows a very simple application, exported to the xMML format.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:model
    id="xmml-1389284816594"
    name="MAD generated XMML"
    xmml_version="0.4"
    xmlns:ns3="http://www.mapper-project.eu/xmml"
    xmlns:mad="http://www.mapper-project.eu/mad">

    <ns3:description>This document was generated by MAD.
      To change it directly edit this file</ns3:description>
    <ns3:definitions>
        <ns3:datatype id="file"/>
        <ns3:datatype id="directory"/>
        <ns3:datatype id="double_array"/>
        <ns3:mapper id="mapper-1" interactive="no" name="coreprofPlot" type="fan-out">
            <ns3:description>Plot coreprof CPO (requires also
              data containing in equilibrium CPO)</ns3:description>
            <ns3:ports>
                <ns3:in datatype="directory" fixedName="false" id="port-id-in_data"/>
                <ns3:out datatype="directory" fixedName="false" id="port-id-out_img"/>
            </ns3:ports>
        </ns3:mapper>
        <ns3:submodel id="submodel-2" interactive="no" name="Antv2">
            <ns3:timescale delta="1 min" total="1 min"/>
            <ns3:spacescale delta="1 cm" total="1000 cm"/>
            <ns3:ports>
                <ns3:out datatype="double_array" fixedName="false"
                  id="port-id-ok" operator="Oi"/>
                <ns3:in datatype="double_array" fixedName="false"
                  id="port-id-otherDistance" operator="S"/>
                <ns3:in datatype="file" fixedName="false"
                  id="port-id-in_file" operator="finit"/>
            </ns3:ports>
        </ns3:submodel>
        <ns3:mapper id="mapper-3" interactive="no" name="Elephant2ant" type="fan-out">
            <ns3:ports>
                <ns3:in datatype="double_array" fixedName="false" id="port-id-inDistance"/>
                <ns3:out datatype="double_array" fixedName="false" id="port-id-outDistance"/>
            </ns3:ports>
        </ns3:mapper>
    </ns3:definitions>
    <ns3:topology>
        <ns3:instance id="id-001" mapper="mapper-1">
            <ns3:extra>
                <mad:implementations>
                    <mad:implementation intVersion="2.7.2" interpreter="Python">
                        <mad:bundleLocation></mad:bundleLocation>
                        <mad:class></mad:class>
                        <mad:configFile>import ual
import ascii_cpo
import os
import matplotlib
import re

(...)
                        </mad:configFile>
                        <mad:parameters/>
```

```
            </mad:implementation>
        </mad:implementations>
        <mad:portMapping>
            <mad:portMapping id="id-002" name="in_data"/>
            <mad:portMapping id="id-003" name="out_img"/>
        </mad:portMapping>
        <mad:position x="-258" y="0"/>
    </ns3:extra>
</ns3:instance>
<ns3:instance id="id-020" submodel="submodel-2">
    <ns3:extra>
        <mad:implementations>
            <mad:implementation intVersion="2010-01-11_13-51-27"
              interpreter="muscle">
                <mad:bundleLocation>
                  grass1.man.poznan.pl:antv2.jar
                </mad:bundleLocation>
                <mad:class>mask.example.Antv2</mad:class>
                <mad:parameters>
                    <mad:parameter id="initial_distance"
                      type=":number" value="0"/>
                    <mad:parameter id="in_file"
                      type="asset:string" value="density.txt"/>
                    <mad:parameter id="out_file"
                      type="asset:string" value="step_log.txt"/>
                    <mad:parameter id="value_time_delta"
                      type="dependsOnTimescale:number" value="1"/>
                    <mad:parameter id="value_time_max"
                      type="dependsOnTimescale:number" value="10"/>
                    <mad:parameter id="value_space_x_delta"
                      type="dependsOnSpacescale:number" value="1"/>
                    <mad:parameter id="value_space_x_max"
                      type="dependsOnSpacescale:number" value="1000"/>
                </mad:parameters>
            </mad:implementation>
        </mad:implementations>
        <mad:portMapping>
            <mad:portMapping id="id-021" name="otherDistance"/>
            <mad:portMapping id="id-022" name="in_file"/>
            <mad:portMapping id="id-023" name="ok"/>
        </mad:portMapping>
        <mad:position x="503" y="320"/>
    </ns3:extra>
</ns3:instance>
<ns3:instance id="id-024" mapper="mapper-3">
    <ns3:extra>
        <mad:implementations>
            <mad:implementation intVersion="2010-01-11_13-51-27"
              interpreter="muscle">
                <mad:bundleLocation>
                  grass1.man.poznan.pl:elephant2ant.jar
                </mad:bundleLocation>
                <mad:class>mask.example.Elephant2ant</mad:class>
                <mad:parameters>
                    <mad:parameter id="stop" type=":number" value="10"/>
                    <mad:parameter id="step" type=":number" value="1"/>
                </mad:parameters>
            </mad:implementation>
        </mad:implementations>
        <mad:portMapping>
            <mad:portMapping id="id-025" name="inDistance"/>
            <mad:portMapping id="id-026" name="outDistance"/>
        </mad:portMapping>
        <mad:position x="221" y="218"/>
    </ns3:extra>
</ns3:instance>
<ns3:coupling
```

```
            mad:connectionId="id-027"
            from="id-024.outDistance"
            mad:fromPortId="id-026"
            to="id-020.otherDistance"
          mad:toPortId="id-021"/>
    </ns3:topology>
</ns3:model>
```

MAD uses the `mad` namespace for its extensions to xMML.

Each `<mad:implementations>` element contains all implementations available for the corresponding node. Each implementation is further represented by a single `<mad:implementation>`.

`<mad:parameters>` describes parameters of a specific implementation, if such parameters are available. Each parameter is described by a single `<mad:parameter>` element.

`<mad:portMapping>` defines names and IDs of ports.

`<mad:position>` describes node's position on the workspace. This particular information is used to restore the application in MAD to exactly the same state as before the export.

5.5. Directory structure

The directory structure of MAD's codebase is quite simple—it follows the Maven standards.

In the main project directory, there is a `pom.xml`, which describes Maven configuration for the parent project. Except for the configuration file, there are three directories:

- `ibuilder-api` - contains the *ibuilder-api* module
- `ibuilder-gwt`- contains the *ibuilder-gwt* module
- `ibuilder-xmml` - contains the *ibuilder-xmml* module

Each of these modules has its own `pom.xml` and a `src` directory which contains all of module's source code.

One level deeper, there are directories `main` and `test`, which contain the source code of the application and tests for this code, respectively.

One level deeper, there are either of the following three directories:

- `java` - the source code written in Java
- `resources` - any `.properties` files, images (e.g. `.svg`) and UiBinder templates (`.ui.xml`)
- `webapp` - configuration of the web application

6. References

[1] D. H. K. Rycerz, T. Gubala et al. D 8.3 second prototype with demonstration v1.1. http://dice.cyfronet.pl/projects/details/Mapper-files/deliverables/D8.3-SecondPrototype-CYF-v1.1.pdf, 2011.

[2] E. C. K. Rycerz, T. Gubala et al. D 8.1 description of the architecture and interfaces v2.13. http://dice.cyfronet.pl/projects/details/Mapper-files/deliverables/d8.1-architectureinterfaces-cyf-v2.13.pdf/view, 2011.

[3] R. P. K. Rycerz, J. Borgdoff et al. D 8.4 final validation and integration with external modules v1.5. http://dice.cyfronet.pl/projects/details/Mapper-files/deliverables/d8.4-final_validation-cyf-v1.5.pdf/view, 2013.

[4] MAD, MaMe, EW tutorial. http://www.mapper-project.eu/web/guest/mad-mame-ew.

[5] GridSpace tutorial. https://gs2.mapper-project.eu/ew/tutorials.

[6] Google Web Toolkit. http://www.gwtproject.org/.

[7] Google Web Toolkit, UiBinder documentation. http://www.gwtproject.org/doc/latest/DevGuideUiBinder.html.

[8] MAPPER Memory (MaMe) User Manual. http://dice.cyfronet.pl/projects/details/Mapper-files/MaMe_User_Manual.

[9] MML. https://github.com/blootsvoets/xmml.

[10] The MUSCLE library. http://apps.man.poznan.pl/trac/muscle.

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science

FINAL PROJECT

EXTENSION OF A TOOL SUPPORTING DESIGN OF MULTISCALE APPLICATIONS

KONRAD STRACK

SUPERVISOR:
dr inż. Katarzyna Rycerz

Kraków 2014

NON-PLAGIARISM STATEMENT

I HEREBY DECLARE THAT I HAVE WRITTEN THIS THESIS COMPLETELY BY MYSELF, THAT I HAVE USED NO OTHER SOURCES OR RESOURCES THAN THE ONES MENTIONED, AND THAT I AM AWARE OF THE RULES OF THE PENAL CODE REFERRING TO CRIMINAL LIABILITY FOR PROVIDING FALSE EVIDENCE.

. . . . . .

SIGNATURE

Contents

1. Introduction

1.1. What is the Multiscale Application Designer?

MAD (Multiscale Application Designer) allows you to create multiscale applications by combining multiple single scale models (in MAD they can also be referred to as *nodes*). These applications are described graphically using the gMML representation of the Multiscale Modeling Language (MML) [5]. Models available in MAD are described in section 3.1. For the description of all models and their types as defined in MML, please refer to section 3.1.1 of the technical documentation.

Models available for designing applications in MAD are fetched from the MAPPER Memory (MaMe) [4]. MaMe provides a persistent storage for the models and whole applications. Models are stored in the component of MaMe called the MaMe Registry, whereas applications are stored in the MaMe Repository.

Applications created in MAD can be afterward imported into the GridSpace Experiment Workbench [3], and GridSpace is responsible for their execution, and for fetching the results.

In order to support these various use cases, MAD provides the possibility to export applications to different XML-based formats. The xMML format (the XML representation of MML) is used by the MaMe Repository to store applications. The GridSpace experiment format is used to pass applications to GridSpace. For the detailed description of supported formats, please refer to section 4.1.

MAD provides you with a simple graphical interface designed to help you with building multiscale applications. It includes:

- The workspace where all parts of an application are composed.
- A list of nodes that you can use to add nodes to the workspace with a simple drag and drop operation.
- An editor, which allows you to edit parameters of nodes in the workspace.
- The menu, which contains various actions you can perform on the whole application.

The interface is described in more detail in section 2.

1.2. Requirements

MAD runs in a web browser which allows it to run easily on Linux, Windows and Mac OS X. Supported browsers include:

- Chrome - version 20 and above
- Firefox - version 12 and above, also Extended Support Releases 17.x and 24.x
- Internet Explorer - version 9 and above

MAD doesn't have high hardware requirements. The minimal reasonable requirements are:

- 256MiB of RAM.
- Integrated graphic card with at least 32MiB of memory.
- Screen resolution of at least 1024 x 768.

It may also work on hardware which does not conform to these requirements. However, in such case it can be noticeably slower, and various problems may appear.

The recommended requirements are:

- At least 512MiB of RAM.
- Integrated graphic card with at least 64MiB of memory.
- Screen resolution of at least 1920 x 1080.

Please note the importance of the screen resolution—it is much more convenient when the whole multiscale application is visible on the screen, than when you need to scroll to see different parts of the application.

2. Overview

Figure 1 presents an example screenshot of the application. We can distinguish four main parts, each one serving a different purpose.

The huge area in the center is the *workspace area*. It contains all nodes and connections between them that together form the application you are designing. The *node panel* on the left contains a list of available nodes, which can be dragged to the workspace every time you want to add a node. The *editor panel* on the bottom displays parameters and source code of the nodes visible in the workspace. Finally, the *menu panel* on the right contains a list of buttons representing possible actions, e.g. ones related to import and export.

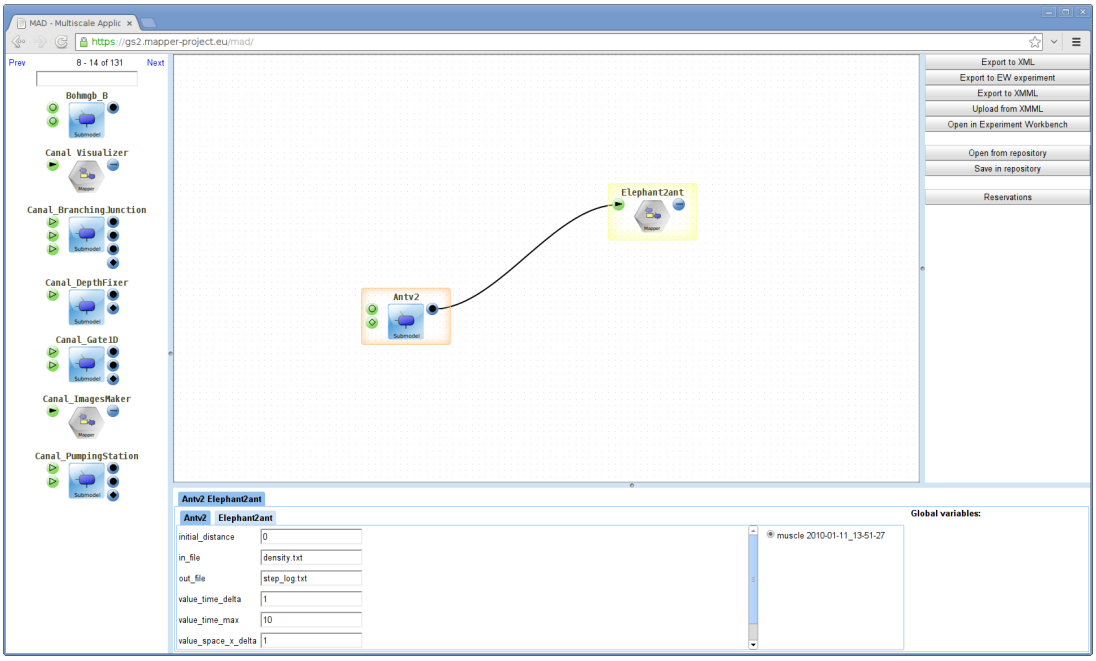

Figure 1: An example view of MAD.

You can resize each panel by dragging the corresponding blue bar on the side of the panel. The content inside the panel will be resized to fit the new size. The workspace will occupy the remaining space.

2.1. Workspace

Workspace shows the graphical representation of your application and contains all nodes and connections between them.

Nodes are placed on the workspace by dragging them with a mouse pointer from the node panel. Figure 2 shows such action: an instance of a node is dragged from the node panel to the workspace.

When a node is added to the workspace, two things happen:

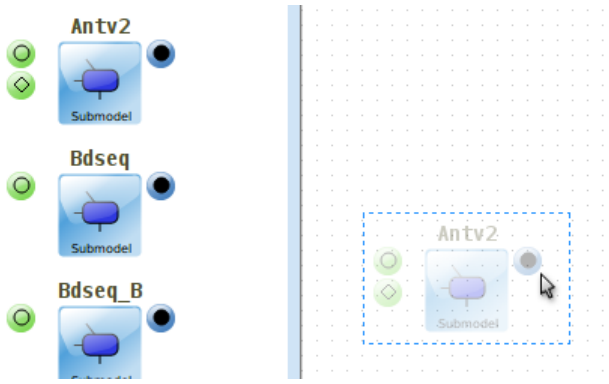

Figure 2: Adding a node to the workspace.

1. An editor in the editor panel is opened for that node.
2. The node is marked as selected.

For a detailed description of the behavior of nodes when they are selected and the graphical representation of nodes in various states, please refer to section 3.3.

A node can be removed from the workspace by selecting it and pressing the **Delete** key on the keyboard.

2.2. Node panel

The node panel contains all nodes available for usage in the multiscale application. It presents an alphabetical list of nodes with submodels and mappers intermixed. Nodes are displayed exactly as on the workspace—with their name and all ports.

2.2.1. Browsing nodes

As can be seen in Figure 3, the number of available nodes is displayed on the top of the panel, along with the range of nodes being displayed currently. Two links—**Prev** and **Next**—allow you to go to the previous and the next page when there are more nodes than can be displayed in the panel. Those links are active (such state is represented by blue color) only when the possibility to switch the page exists.

2.2.2. Searching for nodes

The search field visible in Figure 3 allows you to search in the list of nodes. In order to search, start typing in the search field. Results will be updated after every keystroke. The search is case-insensitive, and the typed-in text has to match some part of the name in order to yield results.

Example 1 If you search for `ant`, results may include *Antv2*, *Elephant* and *Elephant2Ant*.

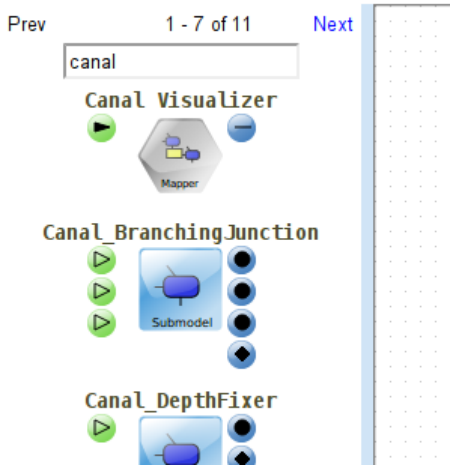

Figure 3: Controls of the node panel.

Example 2 If you search for `phant`, results may include *Elephant* and *Elephant2Ant*.

2.3. Editor panel

Editor panel consists of multiple tabs with each tab corresponding to a node in the workspace or a group of nodes in case of tightly-coupled sections. However, tabs for tightly-coupled sections include multiple sub-tabs, one tab per node belonging to the section.

In the beginning, the editor is empty. A tab is created every time you add a new node to the workspace and is removed every time you delete a node from the workspace.

Corresponding tabs are activated when a node is selected in the workspace. Inversely, when a tab is chosen, the corresponding node becomes selected in the workspace.

2.3.1. Single nodes

The largest part of each tab (the part on the left) may contain a text editor or a list of input fields corresponding to the parameters of the submodel or mapper. An example text editor can be seen on Figure 4, whereas Figure 5 shows a list of parameters. Parameters are saved automatically while typing without any additional action from the user.

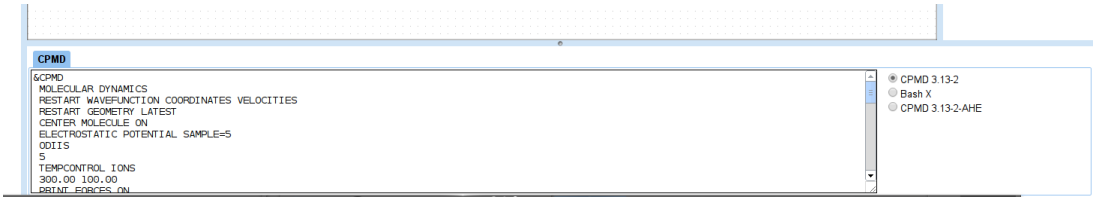

Figure 4: A simple editor with source code.

Nodes can have several implementations assigned to them. Checkboxes on the right side of the tab allow you to choose between different implementations. The chosen implementation will be used as a default one when the model is exported.

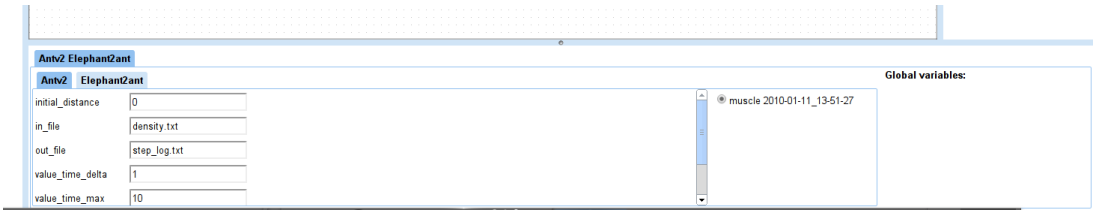

Figure 5: An editor for tightly-coupled sections.

2.3.2. Tightly-coupled sections

There are slight differences between tabs for normal nodes and tabs for nodes belonging to a tightly-coupled section. All nodes in the tightly-coupled section are grouped under one tab. The name of the tab contains names of all nodes in the section. Communication between nodes of a tightly-coupled section can be implemented with MUSCLE [6].

Such tab consists of two parts:

- Another panel with tabs—each tab corresponds to one node in the tightly-coupled section.
- List of global parameters analogous to the list of node's parameters in tabs for single nodes. These parameters are shared between all nodes belonging to the tightly-coupled section.

Global parameters are saved automatically while typing, and they are synchronized between all nodes without any additional action from the user.

2.4. Menu

MAD's menu consists of several buttons:

- 5 related to import and export of the multiscale application to and from MAD
- 2 related to the application repository
- 1 related to reservations

2.4.1. Import and export

This section describes actions hidden behind buttons present in the menu. For more precise description of export and import formats supported by MAD, please refer to section 4.

Export to XML
: This option allows you to export the application to the XML format. An `.xml` file will be created as a result of this operation.

Export to EW experiment
: This option will export the application to the GridSpace Experiment Workbench experi-

ment. It will create an `.xml` file, which can later be imported in the GridSpace Experiment Workbench.

Export to xMML
This option creates an `.xml` file with application exported to the xMML format. Please note that though xMML is represented in XML the data contained conforms to the specific xMML format requirements. Implementations, parameters and positions of all nodes in the workspace are saved too.

Import from xMML
This option imports an xMML file and overwrites current workspace with a model imported from this file.

Open in Experiment Workbench
Directly opens the application in Experiment Workbench. Equivalent to exporting to EW experiment (as an `.xml` file) and loading it in Experiment Workbench.

2.4.2. Repository

Applications created in MAD can be saved in the repository called MaMe Repository [4]. If you click on the **Open from repository** button, a list will be shown containing all applications available in the repository. Such list can be seen in Figure 6. In order to load an application from the repository, select it from the list and click **Open**. This list can also be accessed directly in MaMe at: `http://gs2.mapper-project.eu/mame/experiments_list`.

After you create an application you may want to save it in the repository instead of exporting it to e.g. xMML format so that it can also be accessed by other people. In order to save the application, click on the **Save in repository** button. After you click, a dialog box will appear with a simple form, such as the one in Figure 7. Afterward, fill in the name and description of the application and click **Save**.

Please note that once saved, the application can be removed only directly in MaMe. MAD doesn't support removal of applications from the repository.

2.4.3. Reservations

MAD allows you to directly manage time reservations for execution of tightly-coupled sections. If your application contains a tightly-coupled section, you can do this by clicking the **Reservations** button. A dialog box will appear with a form in which you need to provide a valid user proxy. The form allows you to choose reservation types for each of tightly-coupled nodes.

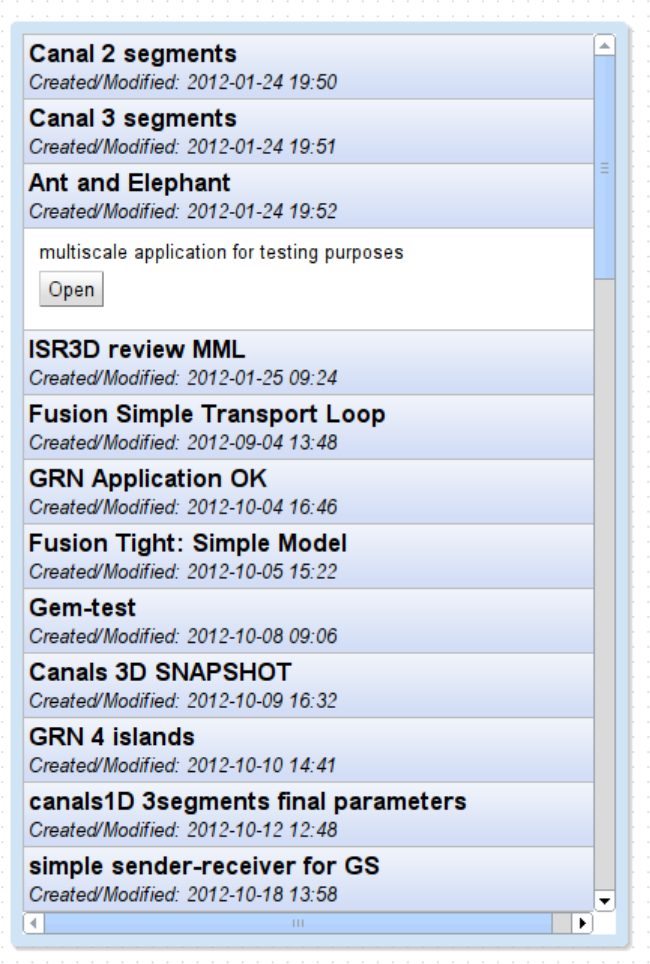

Figure 6: List of applications available in the repository.

3. Nodes

In order to work effectively with MAD, it is essential to know available types of nodes and ports belonging to these nodes. Ports are used as endpoints for connections between nodes.

3.1. Types of nodes

There are three types of nodes defined in MML: *submodels*, *filters* and *mappers*. MAD, however, supports two of those types: *submodels* and *mappers*.

Submodels

Submodels are self-contained reusable components with specified implementation and computational requirements. Theoretically, they should not be aware of other submodels and connect to them only by conduits (a conduit is a unidirectional point to point connection between two elements of an MML diagram). Submodels are represented graphically as squares. An example submodel can be seen on Figure 8.

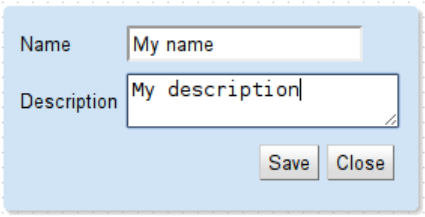

Figure 7: The form used to save an experiment in the repository.

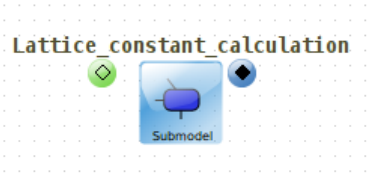

Figure 8: A simple submodel.

Mappers

Mappers are similar to submodels, but their only task is data transformation. They may be used to optimize and simplify coupling between submodels or to build complex connections between them. Mappers are represented graphically as hexagons. Figure 9 shows an example mapper.

3.2. Types of ports

Each node may contain input and output ports. Input ports are green and are placed on the left side of the node. Output ports are blue and are placed on the right side.

There are several types of ports with their specific names and symbols. We distinguish 4 types of input ports, namely:

- undefined
- initialization (f_{init})
- solving step (S)
- boundary conditions (B)

Names in brackets may be encountered instead of full names e.g. in MaMe. Figure 10 shows graphical symbols, which represent respective types.

There are also 3 types of output ports:

- undefined
- finalization / final observation (O_f)
- intermediate state / intermediate observation (O_i)

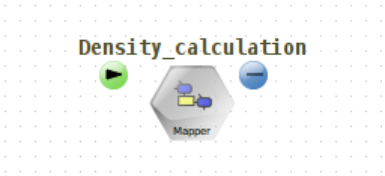

Figure 9: A simple mapper.

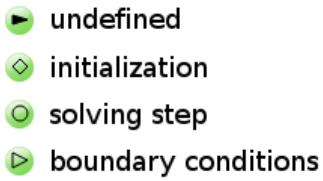

Figure 10: Graphical symbols for input ports in MAD.

As with input ports, the names in brackets may be encountered e.g. in MaMe. Figure 11 shows graphical symbols assigned to these types.

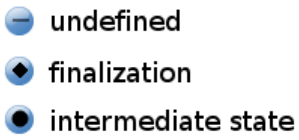

Figure 11: Graphical symbols for output ports in MAD.

Ports have their respective names used in nodes' implementations. Knowing these names may greatly help to distinguish between multiple ports of the same type, and therefore, connect nodes properly. In order to display the name of some port hover the mouse pointer over its symbol. A small tooltip will be displayed with the name of this port.

3.3. Operations on nodes and ports

There are several operations that can be performed on all nodes and ports, irrespective of their types. These operations include, among others: adding a node to the workspace, removing a node from the workspace, adding and removing a connection (a conduit) between two ports, selecting nodes, etc. Following sections will shortly describe actions needed to perform these operations.

Furthermore, nodes in the workspace are displayed in a slightly different way depending on whether they are currently selected and whether they belong to a tightly-coupled section or not. When a node is not selected it is displayed on a white background, like in Figure 12. When it is selected (either by clicking directly on the node, or by selecting the corresponding tab in the editor panel) it is displayed on a slightly red background, like in Figure 13.

When a node belongs to a tightly-coupled section, it has a yellow-beige background, like the node in Figure 14. Please note that if a node belonging to a tightly-coupled section is selected, it

is displayed like any other selected node, with no indication that it belongs to a tightly-coupled section.

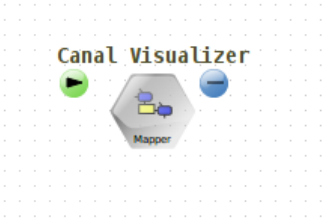

Figure 12: A node, which is not selected.

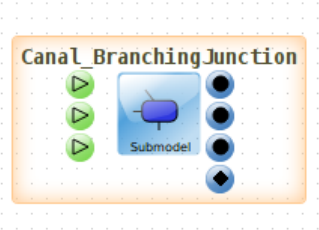

Figure 13: A selected node.

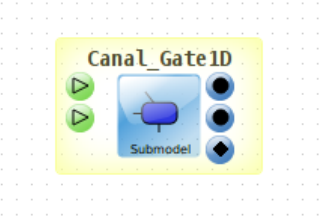

Figure 14: A node belonging to a tightly-coupled group.

3.3.1. Adding a node to the workspace

To add a node to the workspace, first you need to find it in the node panel. For instructions on how to browse and search for nodes in the node panel, please refer to section 2.2. After you have chosen a node:

1. Click and hold the mouse button on the node's symbol (square for submodels and hexagon for mappers).
2. While still holding the node, drag it to the workspace and drop it in a desired location by releasing the mouse button.

After you have added the node, a new tab will be automatically opened in the editor panel, and it will immediately become active.

3.3.2. Removing a node from the workspace

To remove a node from the workspace:

1. Click on the node's symbol (square for submodels and hexagon for mappers).
2. Press **Delete**.

Please note that when you remove a node from the workspace all existing connections to and from this node will also be removed.

3.3.3. Adding a connection (conduit) between two ports

To create a conduit between two ports, first click on any port from the pair and then click on the node you wish to connect it to. The order in which you click nodes is not important. When you successfully connect two ports a black curve representing the conduit will be added to indicate the connection. Note that you cannot create a conduit between two input ports nor between two output ports.

3.3.4. Removing a conduit

To remove a conduit from the workspace:

1. Click on the conduit you wish to remove. It will change its color from black to red to indicate the selection.
2. Press **Delete**.

4. Import and export

MAD provides multiple possibilities of exporting and importing applications, and interacts with other services such as the Experiment Workbench or the MaMe repository. This section provides an overview of available import and export options and interaction with external services.

4.1. Formats

The three formats that MAD understands and allows you to export the application to are: XML, EW experiment, xMML. These all formats are in fact XML-based, but they structure information differently and contain different amount of details.

4.1.1. XML

MAD allows you to export an application by serializing all of its nodes and connections between them to XML. Important thing to be aware of is that this is a very simple representation. E.g. it doesn't include positions of nodes on the workspace and types of ports.

4.1.2. xMML

xMML is the XML representation of the Multiscale Modeling Language (MML). This representation describes the application in high detail and includes MAD-specific information about implementations and parameters (for details please refer to section 5.4 in the technical documentation). Applications exported into this format can later be imported into MAD. An application is then restored to the exactly same state as in the moment of exporting.

4.1.3. GridSpace Experiment Workbench format

EW experiment format is also an XML-based format. It describes the application as an experiment which can be imported into the GridSpace Experiment Workbench. Note that an application can be opened directly from MAD in the Experiment Workbench, without the need to first export manually to the correct format. Please refer to section 2.4.1 for details on how to directly open an application in the Experiment Workbench.

4.2. Repository and models registry

MAD uses an external service—MAPPER Memory (MaMe)—in order to load and store definitions of nodes and full applications. It consists of two services:

Models Registry
: Responsible for storing all submodels and mappers. These submodels and mappers are

imported as nodes into the node panel (see section 2.2) and are consequently available for creating applications in MAD.

Experiments Repository

Responsible for storing full applications created in MAD. It stores applications along with all values changed in implementations when an application is being designed in MAD. For more details on how to use the repository from MAD, please refer to section 2.4.2.

MaMe is available at the following URL: `http://gs2.mapper-project.eu/mame/`. It provides additional tools for managing and modifying models and applications (including deletion of applications, which is not possible in MAD).

5. Additional information

Additional sources of information about MAD and its relation to MaMe and the GridSpace Experiment Workbench can be found online.

For more information about the MAPPER Project, please visit: `http://www.mapper-project.eu/`.

If you want to learn more about the possibilities of MAD, MaMe and the Experiment Workbench, please visit: `http://www.mapper-project.eu/web/guest/mad-mame-ew`.

Detailed information about the MUSCLE library can be found at: `http://apps.man.poznan.pl/trac/muscle`.

6. Glossary of terms

Conduit A connection (dependency) between two submodels.

gMML A graphical representation of MML with respective symbols assigned to all entities in MML. Represents only part of the information about the structure of a multiscale application.

GridSpace A platform that allows users to execute virtual experiments. Provides the GridSpace Experiment Workbench [3].

GridSpace experiment or **EW experiment** A single experiment consisting of multiple code snippets, represented together as one XML file.

GridSpace Experiment Workbench or **Experiment Workbench (EW)**

Implementation A representation of a model's logic as code written in one of the available programming languages; also the parameters of the model.

Loosely coupled model An acyclic model (without any cycles between its submodels).

MML (Multiscale Modeling Language) The abstract language that describes submodels and connections between them. It can have several representations such as the xMML and gMML.

Multiscale application or **Application** An application which implements a multiscale process.

Multiscale process Naturally existing process (e.g. in physics or biology) which consits of parts happening on multiple scales at the same time.

MUSCLE (Multiscale Coupling Library and Environment) A communication framework designed to allow multiscale modeling of simulations. In MAD, the MUSCLE framework is used in implementation of tightly-coupled sections. The coupling of a MUSCLE application is specified in Ruby in a CxA file.

Node In the context of MAD, commonly used to name a submodel or mapper.

Registry The place (in this case—MaMe [4]) where various entities such as the submodels are stored.

Repository The place (in this case—MaMe [4]) where xMML descriptions of the multiscale applications can be stored.

Submodel instance One instance of a submodel. There may be multiple instances of the same submodel in one multiscale application.

Tightly-coupled model A model with a cyclic topology (containing a loop betweeen its submodels). Communication between nodes of a tightly-coupled section can be implemented with the MUSCLE framework [6].

xMML An XML representation of MML.

7. References

[1] E. C. K. Rycerz, T. Gubala et al. D 8.1 description of the architecture and interfaces v2.13. http://dice.cyfronet.pl/projects/details/Mapper-files/deliverables/d8.1-architectureinterfaces-cyf-v2.13.pdf/view, 2011.

[2] R. P. K. Rycerz, J. Borgdoff et al. D 8.4 final validation and integration with external modules v1.5. http://dice.cyfronet.pl/projects/details/Mapper-files/deliverables/d8.4-final_validation-cyf-v1.5.pdf/view, 2013.

[3] GridSpace tutorial. https://gs2.mapper-project.eu/ew/tutorials.

[4] MAPPER Memory (MaMe) User Manual. http://dice.cyfronet.pl/projects/details/Mapper-files/MaMe_User_Manual.

[5] MML. https://github.com/blootsvoets/xmml.

[6] The MUSCLE library. http://apps.man.poznan.pl/trac/muscle.

List of Figures