UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

Distributed multiscale computing

Borgdorff, J.

Publication date
2014

Link to publication

Citation for published version (APA):
Borgdorff, J. (2014). *Distributed multiscale computing*.

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)

Download date:19 Sep 2021

D
Software resulting from this thesis

For this thesis, a number of software products were developed. This appendix lists the contributors to the software and where it can be found.

D.1 xMML

The xMML format is described in detail in Chapter 2. It is described by an W3C XML Schema that was developed as part of this thesis, in collaboration with Jean-Luc Falcone and Bastien Chopard, University of Geneva, Switzerland. The latest specification can be found on `http://github.org/blootsvoets/xmml`.

D.2 jMML

The jMML library is a Java library that handles MML. Specifically, it can: read and write the XML format xMML; generate the graphical description gMML; generate a scale separation map; generate a basic MUSCLE project layout and configuration file. The jMML library is a Maven project with three modules:

1. A utilities module `jmml-util` that contains data structures such as lists and graphs, and SI unit and formula handlers.

2. A specification module `jmml-specification` converts from and to xMML using the Java Architecture for XML Binding (JAXB). It has custom classes that facilitate easy manipulation of xMML.

3. An API `jmml-api` to create a coupling topology, code skeleton, task graph or scale separation map from a xMML specification. It can also output a task graph or gMML to pdf.

The main part of the `jmml-specification` is generated from the W3C XML Schema of xMML by the `xjc` tool of JAXB. This means that when a new version of xMML is released, jMML can easily adapt. It also means that it can always read, modify and write a certain version of xMML. On top of these generated sources, verification of couplings and datatypes is performed. xMML files should be validated against the XML Schema before being passed to the `jmml-specification` module, since the module does not perform strict validation of the XML structure.

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

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

```
$ xmllint -xinclude FILE.xml > NEWFILE.xmml
```

which is part of the libxml2 package.

To manipulate xMML in Java, use Maven to import the `jmml-util` and `jmml-specification` modules. To also generate a task graph or coupling topology in Java, the `jmml-api` module can be imported. The jMML library is open source, and available from `http://github.org/blootsvoets/jmml`. It was developed by myself as a research tool as part of this thesis.

D.3 MUSCLE

MUSCLE is fully described in Chapter 3. Its current development started with the DSCL package left as part of the COAST project [65]. Since then, development by myself and Mariusz Mamonski and Bartosz Bosak, Poznan Supercomputing and Network Centre (PSNC), Poland, transformed it to a more usable, faster, and more portable package. Of the current 30,000 lines of code, 1,200 come from the original package (counting 23,000 lines of code). In collaboration with Derek Groen, University College London, UK, wide-area communications can be performed with the MPWide library [59]. MUSCLE is open source software, available at `http://apps.man.poznan.pl/trac/muscle`.

D.4 ISR3D

The in-stent restenosis model ISR3D is fully described in Chapter 4. The original code was developed as part of the COAST project [32]. As part of this thesis, I updated it to use the latest constructs in the multiscale modelling theory described in Chapter 2, e.g., fan-in and fan-out mappers, and to make use of the latest features in MUSCLE, e.g., MPI, full Fortran and C++ support, and logging. The models scientific contents were largely updated by Carles Bona Casas, University of Amsterdam, the Netherlands. The code of ISR3D is currently not published.