MesoRD User's Guide

Johan Hattne

David Fange

Johan Elf

MesoRD User's Guide
by Johan Hattne, David Fange, and Johan Elf

User's Guide version 0.3 for MesoRD Edition

This product includes software developed by Roldan Pozo, Karin Remington, and Andrew Lumsdaine. (http://math.nist.gov/sparselib++/ (http://math.nist.gov/sparselib++/)).

Table of Contents

List of Tables

Welcome to the Wonderful World of MesoRD

MesoRD is experimental software. This means that if it does something you did not expect, maybe its developers would not have expected it either. It does *not* mean that you should not use it. On the contrary: please use MesoRD and send any bug reports or feature requests to us, and we will see what can be done.

Chapter 1. Introduction

What is MesoRD?

MesoRD is a tool for stochastic and deterministic simulation of chemical reactions and diffusion in 3D space.

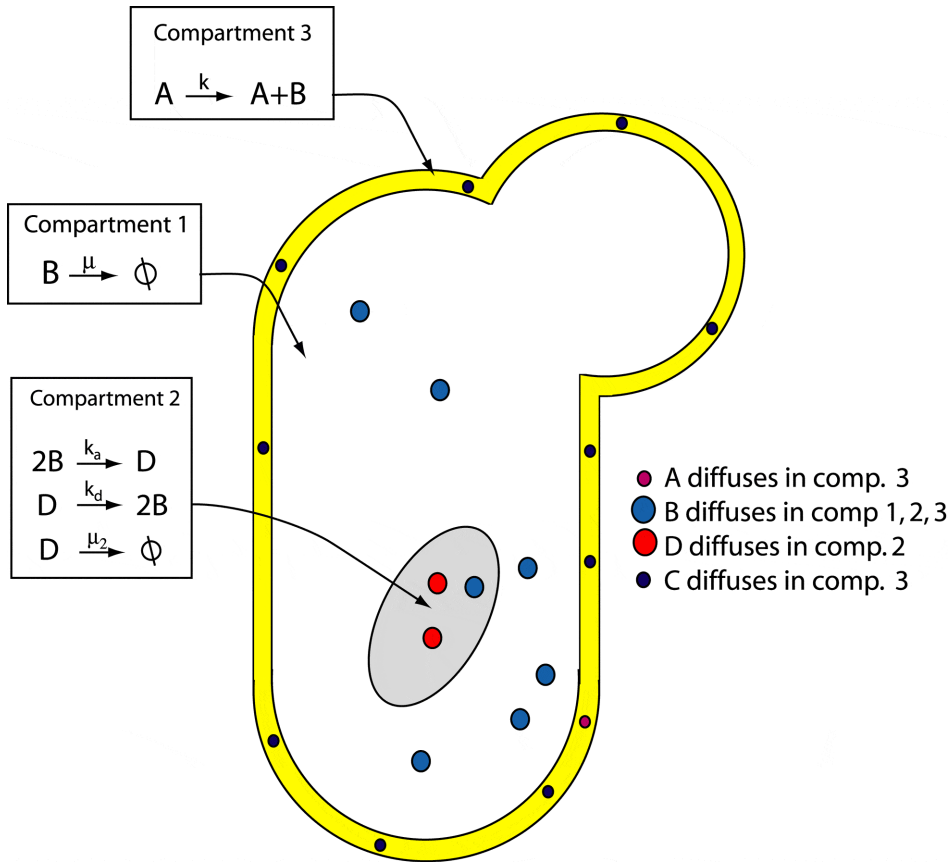
The description of the system that you want to simulate is written in the SBML file format Finney and Hucka, 2003 . The SBML file contains all information about the species, reactions, reactions rates, compartment geometries, diffusion constants etc. In addition to the SBML file, MesoRD will require information about how the simulation should be executed, such as spatial discretisation of the reaction volume, duration of the simulation, visualisation and output options, and for deterministic simulations also choice of integration method. These parameters are given through the MesoRD user interface. The output files from MesoRD are intended for external data analysis and visualisation packages, for instance the freely distributed MesoRD Matlab toolbox available from the MesoRD website.

In both the deterministic and the stochastic mode of simulation the reaction volume is discretised into a large number of small subvolumes and the state of the system is given by the number of molecules per subvolume. In the stochastic simulation the number of molecules per subvolume are discrete. Furthermore, the reaction and diffusion events that change the number of molecules are probabilistic, in the sense that the next event in the systems is sampled from a distribution function. In the deterministic simulation the state is assumed to be continuous and the change in the number of molecules per time unit is given by the average change as defined from the stochastic model. A more detailed description of the differences between the stochastic and deterministic way of modelling is given in the theory section Chapter 6.

An introductory Example

One example of a system that can be easily modelled in MesoRD is the one illustrated in the figure below. B-molecules are synthesised by a membrane bound A-molecule. The B-molecules diffuse into the cytoplasm and are degraded or modified in a first order reaction. Some B-molecules reaches an intracellular compartment, where they dimerize into a D molecule, which also can be degraded.

Figure 1-1. Example of a three compartment geometry.



The system has a non-trivial geometry, it is non-homogeneous, and it may have some interesting stochastic properties. It is therefore suitable for simulation in MesoRD. How this model is described in SBML for MesoRD is described further in Chapter 7.

Where we go from here

The rest of the introduction includes the Section called *Mission Statement*, the Section called *System Requirements*, the Section called *Project Status*, the Section called *Citing use of MesoRD*, and the Section called *Funding*, but maybe you want to start making models.

Constructing models for MesoRD is mainly a question about writing, or generating as it may be, valid SBML files. However, there are some special considerations one has to keep in mind when working with MesoRD.

- Chapter 3 contains information on how to write SBML files for MesoRD. Note that this documentation by no means is a replacement for the SBML language documentation. If you are not familiar with SBML, Finney and Hucka, 2003 may be a good starting point.

- The construction of geometries is discussed separately in Chapter 4. The geometry specification is the largest MesoRD secific addition to the model-files compared to standard SBML. Currently SBML has no support for geometry definitions.
- Chapter 5 describes the user interfaces available to control the launching of simulations. It covers the parameters that are not included in the SBML model definitions and the output file formats.
- Chapter 6 contains information on how stochastic and deterministic reaction-diffusion systems are treated in MesoRD.
- Chapter 7 is a small tutorial, giving a full description of how to translate the example from the Section called *An introductory Example* into valid SBML for MesoRD.

All this assumes that MesoRD has been successfully installed. Installation is covered in Chapter 2. The impatient may as well skip ahead to any of the above chapters.

System Requirements

The software requirements for installing MesoRD are detailed in Chapter 2. MesoRD runs on several different operating systems. We have tested Linux/Alpha, Linux/x86, Mac OS X/PPC, NetBSD/x86, Solaris/Sparc32 and Windows XP. The main platforms for simulations has been Windows XP and Linux/x86, using the Intel C++ compiler version 8.1. Lately a lot of the development of MesoRD has been carried out using Mac OS X/x86.

Reaction-diffusion simulation in 3D is computationally demanding and MesoRD can therefore be a resource hog as far as CPU time, disk space and memory is concerned.

- MesoRD will use whatever CPU time it is assigned.
- MesoRD is multithreaded on a high level in the sense that the simulator and the visualiser run in separate threads. If the visualiser is enabled, MesoRD has the capability to significantly burden *both* processors on a dual CPU system. In addition to dual processors a good 3D graphics card is necessary for smooth real-time visualisation while running large (>10000 subvolumes) simulations [1]. When running large simulations on a one-CPU machine we strongly recommend that the visualiser is turned off, since it will only slow things down. Running on more than two processors should not improve performance significantly. If the processors do not share memory, one is almost guaranteed to cause performance loss.
- How much disk space is needed depends on how often the system is checkpointed. Checkpoint files can become huge, especially if the system is big and checkpointing is done frequently. The size of the checkpoint files should scale linearly with the number of species and the number of sub volumes in the model. The number of sub volumes depends on the geometry specified and the spatial discretisation. If the *sparse* option is chosen in stochastic simulations the file sizes are dramatically decreased, since only data for non-empty subvolumes are stored.
- Memory consumption should scale linearly with product of the number of subvolumes and the number of species. The deterministic simulation has been observed to require about 5 times more memory then the stochastic simulations. [2]

Mission Statement

MesoRD has been designed to have a number of features. Whether we have succeeded or not in our mission is not entirely clear.

Useful

MesoRD should be useful for scientist who wants to simulate reaction diffusion kinetics. It should be more convenient, but equally reliable, to use MesoRD as compared to writing your own code. To reach this goal the source code is freely exchanged so that is can be independently validated.

User-friendly

User-friendliness was interpreted to mean that one should not have to recompile the code in order to run a simulation with different parameters or different reactions. (This goal was met already in version 0.1)

Maintainable

The code should be readable and understandable by people who were not involved in the early stages of development. This is one of the reason for choosing C++ as the implementation language, because it is a common denominator of everybody currently involved in development. We try to follow good coding practise as advertised in GNU Coding Standards and Stroustrup, 1997. We use source-level documentation which can be transformed to a number of useful formats using doxygen.

Portable

The implementation should run on as many different platforms as possible. Where possible, we want to be able to use compilers optimised for the platform in question. In particular, this means we want to follow standards and we do not want to critically rely on platform specific features such as DirectX on Windows, or the `fork()` call on Unices.

Efficiency

The implementation should make the best use of whatever resources available. This may actually clash with the previous design goals. This is also really difficult.

Project Status

A number of bugs has been found and corrected since the last release of MesoRD. This does not mean that the code is error free in any way. *Please report any errors you encounter, be they in the source code or the documentation.* Suggestions for improvements are welcome as well. You can report bugs using the web interfaces at SourceForge, http://sourceforge.net/projects/mesord (http://sourceforge.net/projects/mesord). The SourceForge pages allow for posting feature requests as well.

What is new in MesoRD 0.3

- Adherence to the new SBML standard (http://precedings.nature.com/documents/2715/version/1) for unit definitions and compartment size unit.
- Rewritten CSG module.
- Support for the libsbml-3.*
- Command-line support on Windows platforms.

Known Bugs

Some of the items here may be of interest mainly to developers. This list is most likely *not* complete!

- The Annotation class is unstable on the Windows and may cause memory access violations.
- The visualiser class is unstable with respect to optimisations on Windows, at least when compiled with icc and vc++-8.0. Whether this is due to bad code, bad compiler or an evil operating system is yet unknown.
- The CSG module seems to suffer from numerical instabilities under icc optimisation. It seems the problem is related to near-singular matrices. The implementation can certainly be made to be more robust.
- Running deterministic simulations for models with many compartments with many different molecule kinds will result in a lot of unnessecary computations.

One final note. I am not Bjarne Stoustrup. Neither is anybody else in the MesoRD development team [3]. If you think that the code looks stupid and could implemented better, you are most likely right. We encourage any improvements. We appreciate your help!

Todo

- We would like to support SBML in full, with events and functions and everything.
- The gtkmm GUI for Unices should be completed.
- Support for more kinds of subvolumes, like tetrahedrons. This work has been started but is yet not completed.
- We would like to be able to simulate 1D and 2D geometries embedded in 3D and to treat diffusion limited reactions correctly in these geometries. (This is not very easy.)
- Can we implement other algorithms that gives complementary information, such as the GFRD (Greens function reaction diffusion) algorithm?
- We should make an efficient implementation for some parallel architecture.

Citing use of MesoRD

Please use reference Hattne et al., 2005 when citing MesoRD and reference Elf and Ehrenberg, 2004 when citing the Next Subvolume Method.

Funding

The development of MesoRD has been funded by grants from the Swedish Research Council (Vetenskapsrådet) to Måns Ehrenberg, Uppsala University. We are very grateful for this support.

If you use MesoRD in a commercial project, please consider supporting the development. Your contribution will be gratefully acknowledged.

Notes

1. One should also be aware that the visulisation done by MesoRD may not be the actual simulation result.
2. MesoRD has a much too generous memory usage, i.e. it is not at all minimal. This will be a major concern in the next version.
3. I know this for a fact, because there are only two developers at the time of writing. (This is still true for version 0.2 and 0.3, although we a are now three developers.)

Chapter 2. Installation

MesoRD Dependencies

Please note that MesoRD currently have stability problems on the Windows. We are working to settle this problem.

MesoRD depends on libSBML, which requires an xml library: either Xerces-C++, expat or xml2. libSBML is supported on Unix as well as on Windows. Binary Windows libraries of libSBMLis provided by the developers http://sbml.org/Software/libSBML. The Windows distribution of libSBML currently ships with precompiled binaries of either Xerces-C++, expat or xml2.

MesoRD contains a visualiser which requires OpenGL, GLU and, on X-window systems, GLX. On Unices, building the visualiser is optional; on Windows platforms it is required. On Mac OS X this means that you need to install the X11 package to build MesoRD with visualiser support. The visualiser does however not introduce any new *critical* dependencies on *any* system, since Windows ships with OpenGL as well as GLU. If you are on a Windows platform, you may want to make sure that you have performance enhanced drivers for your video card installed. Such drivers are typically available from the particular video card vendor.

Table 2-1. MesoRD Dependencies

Package	Version	Description
libsbml	>= 3.0	The libsbml library allows for easy parsing and validation of SBML files. Note that versions prior to 3.0 will not work.
OpenGL	N/A[a]	The open graphics library which provides easy-to-use functions for three-dimensional graphics. NetBSD users may want to avoid the XFree OpenGL in favour of for instance Mesa.

Package	Version	Description
Pre-emptive threads implementation	N/A	MesoRD requires a pre-emptive threads implementation in order to run the different components concurrently. Note that MesoRD is multithreaded, irrespective of the number of processors on the host. On Unices the threads implementation should be POSIX compliant[b]. On Windows, MesoRD will make use of native NT threads. The thread requirement may limit the number of Windows versions that can be used for running MesoRD.
Notes: a. On Unices with the X windowing system you may choose between the implementation that comes with the X server and stand-alone alternatives such as Mesa. There may be issues with some version combinations: on NetBSD, we had strange problems when linking to the XFree86 implementation of OpenGL, while the Mesa implementation worked without any problems at all. On Windows, you should use performance enhanced drivers from your video card vendor, provided such drivers exist. b. Note however that the GNU pth library will not do, as it is not pre-emptive. We have been using unproven-pthreads, which is a hacked version of MIT Pthreads, from the NetBSD `pkgsrc` tree.		

Table 2-2. To build the MesoRD documentation

Package	Version	Description
docbook	NA	The MesoRD documentation is written in docbook.

Installing a Binary Distribution

On Windows, MesoRD ships as an executable installer. Note that the setup utility will copy over a few sample model definitions to wherever the binary was installed. The binay was build using Visual Studio 2005 with the following set of flags:

If there are problems running a binary distributions, you may want to ensure that the shared libraries are found. On Unices, you can use the ldd tool to see what libraries the binary will link against during run time. If the libraries listed are not the ones you wish to use (e.g. if you want to change which OpenGL implementation to use), you can alter the run-time linkage using the

`LD_LIBRARY_PATH` environment variable [1]. On Windows, the DLL:s need to be either in a standard location or in the same directory as the executing binary.

Building MesoRD from Source on Unix

Make sure you have libSBML installed. You will need version 3.0 or later. Recent versions of libsbml allow for replacing Xerces-C++ with expat or xml2 so you can choose the one that fits your setup best.

To build MesoRD move into the unpacked source file directory (e.g. `mesord-0.3`) and type:

```
% ./configure
% make
% make install
```

Should any required components remain un-found, you may need to adjust the compiler's include and library search paths using the `CFLAGS` and `CXXFLAGS` environment variables.

Working the GNU AutoTools

If you are compiling a Unix binary from SVN sources, you may need to reconfigure the sources and regenerate the make files. This is done using the GNU AutoTools. Note that, in addition to autoconf, you will need the `The GNU Autoconf Macro Archive`, which is available for download from the GNU Autoconf web site. The following steps assume that the current directory is the root directory of the distribution.

1. Make sure the source tree is clean. Do this by running **make clean**. If command fails because there are no make files, then you may safely assume that the source tree is clean.
2. Recreate `aclocal.m4`. Note that we distribute some custom macros to detect the particular non-standard libraries needed by MesoRD. Therefore, you need to inform aclocal to look in the **m4/** directory for additional macro files: **aclocal -I ./m4**. If you need macro files placed elsewhere, just append more `-I` arguments to the command line. You need to include `The GNU Autoconf Macro Archive`, and also macro files which are inluceded with the CppUnit package.
3. Recreate template header files. Do this by simply running **autoheader**.
4. Recreate the configure script. Run **autoconf**.
5. Generate the make files. Run **automake -a -c**. Note that with these options, automake will install any missing standard files.

Now you may resume with configuring and compiling the source using the generated files. This process was detailed the Section called *Building MesoRD from Source on Unix* previously.

Building MesoRD from Source on OS X

To build MesoRD on OS X you need to install the developer tools form the OS X installation DVD. After the installtion is complete you can follow the instructions given in the Section called *Building MesoRD from Source on Unix*.

If you want to use the MesoRD visualizer on OS X you will also have to install the X11 server and the X11 developer libraries.

Building MesoRD from Source on Windows

Please note that MesoRD currently have stablity problems on the Windows platform. We are working to settle this problem.

Building MesoRD from Source using Visual Studio 2005

1. Download libsbml and install the distribution. You will need version 3.0 or later of libsbml. The libSBML binary distribution ships with either Xerces-C++ or Expat so there is no need to install these libraries separately.

MesoRD provides a project and solution file for Microsoft Visual C++. These files are found in `woe32/woe32.vcproj` and `woe32/woe32.sln` of the MesoRD distribution root. Opening any of these files will get you started with Visual Studio. Next you need to configure the compiler to find the necessary include and library files.

1. In order for the compiler to find the header files of libSBML, select Options... from the Tools menu, choose the VC++ Directories under Projects and Solutions. Select Include files in the Show directories for dropdown. Add the directories containing the libsbml header files. The headers are typically found in `C:\Program Files\SBML\libsbml-3.X.Y-xerces\win32\include`.
2. The compiler will also need to know the location of the libsbml DLL. Similarly add the directories containing the libsbml DLL to Library files. These are typically found in `C:\Program Files\SBML\libsbml-3.X.Y-xerces\win32\bin`.
3. Finally, we need to make sure that the binary we are about to compile will be linked against the above libraries. Select win32 Properties... from the Project menu. Expand the Linker section and select Input. Under Additional Dependencies make sure that `libsbml.lib` is in the list.
4. You may have to make sure that MesoRD is linked against OpenGL and GLU. Make sure that `glu32.lib` and `opengl32.lib` are in the Additional Dependencies list. Similarly, we need code from the versioning library, so you will need to add `version.lib` as well.
5. MesoRD is an object oriented program, and parts of it make extensive use of Run-Time Type Information, RTTI. This may have to be explicitly turned on: select win32 Properties... from the Project menu. Expand the C/C++ section, chose the Language category and make sure that Enable Run-Time Type Information is set to yes.

Building MesoRD from Source using Visual Studio 6.0.

Please note that we are currently not using Visual Studio 6.0. The instructions below were valid for the 0.2 realase, but have not been updated or tested since.

Note that theses instructions assume that you are using Microsoft Visual C++. However, you can currently not expect the Microsoft compiler to work. You will need to install something else in its place, for instance icc, see the Section called *How to use Intel's icc compiler with Microsoft Visual C++*.

1. Download libsbml and install the distribution. You will need version 3.0 or later of libsbml. The libSBML binary distribution ships with either Xerces-C++ or Expat so there is no need to install these libraries separately.

MesoRD provides a workspace file for Microsoft Visual C++. This file is found in `woe32/woe32.dsw` of the MesoRD distribution root. Next you need to configure the compiler to find the necessary include and library files.

1. In order for the compiler to find the header files of libSBML, select Options... from the Tools menu, choose the Directories tab and show directories for Include files. Add the directories containing the libsbml header files. The headers are typically found in `C:\Program Files\SBML\libsbml-3.X.Y-xerces\win32\include`.
2. The compiler will also need to know the location of the libsbml DLL. Similarly add the directories containing the libsbml DLL to Library files. These are typically found in `C:\Program Files\SBML\libsbml-3.X.Y-xerces\win32\bin`.
3. Finally, we need to make sure that the binary we are about to compile will be linked against the above libraries. Select Settings... from the Project menu. Pick the Link tab and `libsbml.lib` to Project Options.
4. You may have to make sure that MesoRD is linked against OpenGL and GLU. Select Settings... from the Project menu. Pick the Link tab and add `glu32.lib` and `opengl32.lib` to Project Options. Similarly, we need code from the versioning library, so you will need to add `version.lib` as well.
5. MesoRD is an object oriented program, and parts of it make extensive use of Run-Time Type Information, RTTI. This may have to be explicitly turned on: select Settings... from the Project menu. Pick the C/C++ tab, chose the C++ Language category and check the Enable Run-Time Type Information (RTTI).

How to use Intel's icc compiler with Microsoft Visual C++

First you need to install the Intel C++ compiler. We have used version 8.0 and found it to work well in most cases. You will need a license, which can be obtained directly from Intel. The next step is to configure Microsoft Visual C++ to use the freshly installed compiler. This is done by selecting Intel(R) C++ Compiler Selection Tool from the Tools menu in Visual C++. Check the box labelled Intel(R) C++ Compiler and click OK.

Building a Distribution

On Unix, this is accomplished by running **make dist** in the MesoRD distribution root. For Windows, we include a script file for Inno Setup in `woe32/woe32.iss`. The script was generated using ISTool.

Generating the Documentation

The documentation you are currently reading is written in DocBook. In order to generate formatted versions from the provided XML files, you will need a tool chain. Such tool chains are currently, to the best of our knowledge, only available on Unix platforms. If you live in Windows-land, you can install Cygwin. The tools we use are probably not feature-complete, as we occasionally find it difficult to obtain the desired result. If you know about any real good open source of free tools for producing HTML and PDF output from DocBook, pleas let us know. We would really want to be able to use MathML.

To build the documentation you need a set of tools installed. Latex, dvi2bitmap, fig2dev, xsltproc, jade (or openjade), and also the dbtexmath set of script.

You need to add dbtexmath to your path (in bash: export PATH=$PATH:path_to_dbtexmath). You also need to add the flags --with-chunck.xsl and --with-xml-dcl to the configure script. To build the documentation type:

```
% ./configure --with-chunk-xsl=path_to_chunk --with-xml-dcl=path_to_xml
% cd doc
% make html
% make pdf
```

There is a source-level documentation as well, which can be generated using doxygen. First, make sure you are in the root of the MesoRD distribution, which contains the `Doxyfile` file. Then run:

```
% doxygen
```

A Note on C++ Compilers

In order to compile MesoRD, you will need a C++ compiler. While some effort is made to adhere to applicable standards, some compilers are found to work better than others. Currently, there *are* compilers that are not supported by MesoRD. Since certain compilers do not even implement the standards correctly, any effort to follow standards would be wasted if we chose to support such systems. The following compilers are tested, and found to work with MesoRD.

- Intel's icc version 8.0 for Windows and Linux has been tested and found to work. Versions prior to 8.0 are untested, but may very well be able to compile working versions of MesoRD.
- The GNU Compiler Collection, gcc, version 3 and above. Versions prior to 3 do not fully implement the C++98 standard. MesoRD makes use of several features not present in the gcc version 2 series compilers.
- Sun's WorkShop 6 update 2. You may need to add the `-pto` to your `CXXFLAGS` prior to configuring the source. Note that Xerces-C++ is easily installed on this setup, while libsbml may be a little problematic (add `-lc` and `-lCrun` to `LDFLAGS`).

The freely available Visual Studio 2005 Express ships with Visual C++ version 8.0, which is able to build MesoRD, atleast on a 32-bit Windows XP system. These binaries were however not stable on a Windows XP 64-bit system, neigther on a Windows 7 64-bit system. Compiling 32-bit MesoRD binaries with Visual Studio 2005 Profesional 64-bit edition, on a Windows XP 64-bit system did generate usable binaries on Windows XP 64-bit and ... Microsoft's Visual C++ version 6 is not able to build MesoRD.

Source from SVN

We do have a SVN repository for anonymous read access hosted at SourceForge. You can browse the repository by visiting http://sourceforge.net/projects/mesord and following the appropriate links. Anonymous checkouts are allowed; the above mentioned page contains links to all the necessary information to do so. You may need to know that the module name of the application is `mesord`, without any capital letters.

Notes

1. Or whatever environment variable is equivalent to `LD_LIBRARY_PATH` on your system. On OS X it is `DYLD_LIBRARY_PATH`.

Chapter 3. Writing SBML for MesoRD

Systems Biology Markup Language

General SBML Structure

The basic structure of an SBML model definition is given in Example 3-1. The leading `?xml` tag should alway be included, as the character encoding for SBML is always UTF-8. The outermost enclosing `sbml` tag should used verbatim in any models used with MesoRD, since MesoRD only supports SBML level 2. Not all of the features built into SBML are utilised in MesoRD. Similarly, there are some features required by MesoRD which are not provided in SBML. The departures in SBML files for use with MesoRD with respect to standard level 2 are briefly listed in the Section called *Departures from Standard SBML Level 2*.

Example 3-1. The basic structure of an SBML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
  <model id="My_Model">
    <listOfEvents>
        ...
    </listOfEvents>

    <listOfFunctions>
        ...
    </listOfFunctions>

    <listOfUnitDefinitions>
        ...
    </listOfUnitDefinitions>

    <listOfCompartments>
        ...
    </listOfCompartments>

    <listOfSpecies>
        ...
    </listOfSpecies>

    <listOfParameters>
        ...
    </listOfParameters>

    <listOfRules>
        ...
    </listOfRules>

    <listOfReactions>
```

```
      ...
    </listOfReactions>
  </model>
</sbml>
```

Note that none of the components in the skeletal SBML file above are actually mandatory: if there are no parameters in a model, it makes no sense to include an empty `listOfParameters`. The following sections discuss each part in turn, focusing on where the user has to take special care when writing model definitions for MesoRD.

Departures from Standard SBML Level 2

This section briefly presents the key topics where MesoRD and SBML disagree as far as the format of input files is concerned. It would seem that by introducing these changes, we are breaking the major purpose of SBML: interchangeability. This is entirely true. It is also the reason why we have attempted to make this section as small as possible. On the other hand: if we didn't reinterpret SBML a little, we could not have achieved what we wanted.

- Compartments for use in MesoRD must have the default setting of `spatialDimensions="3"`. Furthermore, compartments must have `constant="true"`.
- Species must be defined once for every compartment where they can be found. [1] Each definition must include diffusion constant for diffusion from the compartment of definition to all other compartments in the model.
- Reactions in MesoRD are single-compartment. Reactions must include `reversible="false"`. Using `stoichiometryMath` is completely unsupported.

Note that in cases where MesoRD requires additional information in form of SBML annotations, these elements are defined in the name space identified by `http://www.icm.uu.se`. Note also that the URI reference in arguments such as `xmlns:MesoRD="http://www.icm.uu.se"`, is not directly usable; it does not identify an actual, retrievable document or resource on the Internet. It is simply intended to make the name space unique.

Unit

Figure 3-1. The SBML definition of `UnitDefinition`

UnitDefinition	
id	SId
name	string { use=”optional” }
unit	Unit[1..*]

Figure 3-2. The SBML definition of `Unit`

Unit	
kind	UnitKind
exponent	integer { use="optional" default="1" }
scale	integer { use="optional" default="0" }
multiplier	double { use="optional" default="1" }

SBML gives the user the possibility to define a name for a unit. The defined name can subsequently be used in the expression of quantities in a model. Novel units in SBML are composed of a set of predefined units. These predefined units are listed in Finney and Hucka, 2003. A new unitdefinition is constructed as follows: $u_{new} = \{multiplier\}*(10^{\{scale\}} * u)^{\{exponent\}}$, where u is one of predefined units. Units are constructed through multiplication of unitdefintions as $U = u_1 * ... * u_N$, where U is the constructed unit and $u_{1...N}$ are unitdefinitions. For instance, some of the sample model files included in the distribution define a unit `um` for micrometre. This unit is then used when defining the geometry of the system. *Please note that the relation between scale and exponent has changed compared to old versions of MesoRD*

Example 3-2. Unit Definition.

```
<listOfUnitDefinitions>
  <unitDefinition id="um">
    <listOfUnits>
      <unit kind="metre" scale="-6"/>
    </listOfUnits>
  </unitDefinition>
</listOfUnitDefinitions>
```

It may be interesting to know that MesoRD internally only uses basic SI units, as defined in for instance Physics Handbook. Any units are converted to products of these units when parsing the SBML. Thus you could for instance define a unit `fulmeter` as one Joule per Newton, and specify all your lengths in `fulmeter` in the unit definitions.

Finney and Hucka, 2003 defines default units for several entities in the model. The specification also says that one can redefine these default units throughout the model. *You can not rely on default units in MesoRD, why you instead should define units explicitly, whenever possible.*

Compartment

Figure 3-3. The SBML definition of `Compartment`

Compartment	
id	SId
name	string { use=”optional” }
units	string { use=”optional” }
spatialDimensions	integer { maxInclusive=”3” minInclusive=”0” use=”optional” default=”3”}
size	double { use=”optional” }
units	SId { use=”optional” }
outside	SId { use=”optional” }
constant	boolean { use=”optional” default=”true” }

In SBML, a compartment is a container of finite size for substances. Note that compartments do not necessarily have to correspond to actual structures inside or outside of a cell [2]. It is important to remember that we cannot have a substance without having a *unique* compartment to contain it. This is the source of some trouble when it comes to species that can cross compartment boundaries. MesoRD's approach to resolve the problem is discussed in the Section called *Species*. When using a concentration for defining the initial concentration of species the units tag has to be defined. See the Section called *Species* for more information

The most important modification of compartments for use in MesoRD compared to their definition in SBML, is that *every compartment needs a geometry definition*. Exactly how these geometries are defined is discussed in Chapter 4. Suffice it here to say that the model excerpt below will result in ellipsoid spanning two micrometres in the x-direction, and one micrometre in the y- and z-direction respectively.

Example 3-3. Compartment Defintion.

```
<listOfCompartments>
  <compartment id="CompartmentOne">
    <annotation xmlns:MesoRD="http://www.icm.uu.se">
      <MesoRD:scale MesoRD:x="2"
                    MesoRD:y="1"
                   MesoRD:z="1">
        <MesoRD:sphere MesoRD:radius="1"
                       MesoRD:units="um"/>
      </MesoRD:scale>
    </annotation>
  </compartment>
</listOfCompartments>
```

Notable differences between SBML and MesoRD compartments are listed below. At present, one should not rely on MesoRD reporting any violations against these new requirements. If the program silently breaks, it may be worthwhile verifying that the below conditions are satisfied.

- Compartments for use in MesoRD *must* have the default setting of `spatialDimensions="3"`. MesoRD's idea of spatiality is three-dimensional, and since all compartments are eventually

built from three-dimensional subvolumes, it makes absolutely no sense to set `spatialDimensions` to anything other than three.

- Although it is not fatally wrong to specify the size of an compartment, it is unnecessary. Because MesoRD will discretise the compartment from fixed sized sub volumes, it is difficult to say beforehand exactly how large a compartment will be. Note that it is the internally computed size that will be used if the amount of any given species is specified in terms of concentrations, as opposed to absolute numbers of molecules.
- Currently, MesoRD only supports the default `constant` compartments. This means that one should *not* set `constant="false"`. The size of the individual subvolumes which compose a compartment is fixed throughout the entire simulation. Although it would be possible to change the size of compartment during a simulation by dynamically adding or removing sub volumes, this is currently not implemented.
- There is no need for the `outside` attribute to be specified, as far MesoRD is concerned. The topology of the different compartments in the model is completely covered by the geometry description.
- Note that the computed volume of a compartment is recorded as global parameter with the id `volume_subVolume`. If one wishes to include the volumes of a compartments in kinetic rate expressions, the correct procedure is to define parameters with the id and name that are the same as the compartment id. The value of a particular parameter is irrelevant. When simulating MesoRD will replace the value, in litres, with the volume as computed.

The most common problem during early development of MesoRD turned out to be that the geometry of the compartment was empty. It is not very difficult to inadvertently accomplish such a feat. If one for instance were to model a box, the shortest side of which is 10 nanometre, using cubic subvolumes of side length 100 nanometre, the resulting geometry may not contain any subvolumes, even though the specified volume *is* in fact three dimensional.

Species

Figure 3-4. The SBML definition of `Species`

Species	
id	SId
name	string { use=”optional” }
compartment	SId
initialAmount	double { use=”optional” }
initialConcentration	double { use=”optional” }
substanceUnits	SId { use=”optional” }
hasOnlySubstanceUnits	boolean { use=”optional” default=”false” }
boundaryCondition	boolean { use=”optional” default=”false” }
charge	integer { use=”optional” }
constant	boolean { use=”optional” default=”true” }

SBML defines species as substances or entities that take part in one or more reactions. Clearly, ATP and glucose are species. However, large complexes such as ribosomes fit the SBML definition of species as well. In SBML, species are defined in exactly one compartment. We shall call this compartment the ”home compartment” of a species for the remainder of this section. The existence of a unique home compartment leads to problems when we have species that can cross compartment boundaries. The way the problem is solved in MesoRD is that a species that can occur in several compartment is defined several times, once for every compartment in which it can occur. To prevent violation of the uniqueness criteria for SBML ID:s, every definition must have a unique ID. In order to keep track of which species are actually the same, such species must share the same name. One may think that this is *extremely* redundant. However, the next paragraph explains why it is in fact only *very* redundant. [3]

Please note that speciesSizeUnit has been removed from the SBML standard. This is now represented by the unit on the `Compartment`*, see the Section called Species*

Another requirement that MesoRD introduces that is absent in SBML is the need for diffusion rate constant. Every species definition must define a rate constant for diffusion from the home compartment into every other compartment, including the diffusion rate within the home compartment itself. Thus we see that this partially offsets the redundancy introduced by having to define the same species several times: it need not be the case the diffusion constant for diffusion from compartment one to compartment two is equal to the diffusion constant from compartment two to compartment one, which could be useful to model active transport. The diffusion rate constant must be defined with a unit and we recommend using centimetres squared per second.

To set the initial number to a fixed value regardless of the compartment size use the initialAmount, and set the hasOnlySubstranceUnits to `true`. To set the initial concentration of a molecule in a compartment set the hasOnlySubstranceUnits to `false`, set the initialConcetration to either `mole` or `item` and finally set the units in the Compartment corresponding compartment to a unit as defined in the Section called *Unit*. The unit needs to have a volume-unit, ie a unit that can be reduced to m^3. *Note: the unit in the compartment is the equivavlent of the earler spatialSizeUnit definition in specis.*

The example below defines an hypothetical species `A` in a two-compartment system. Note that the initial amount in `CompartmentOne` is given in number of molecules since `hasOnlySubstanceUnits="true"`, but that the initial amount in `CompartmentTwo` is given as a concentration in units of `mole` per `litre`. Furthermore, note that the diffusion constants are different for diffusion within two compartments.

Example 3-4. Species Definition.

```
<listOfCompartments>
    <compartment id="CompartmentOne"
   units="litre">
      <annotation xmlns:MesoRD="http://www.icm.uu.se">
        <MesoRD:translation MesoRD:x="0"
                            MesoRD:y="1"
                            MesoRD:z="0"
                            MesoRD:units="um">
          <MesoRD:box MesoRD:x="0.4"
                      MesoRD:y="0.4"
```

```
                    MesoRD:z="0.4"
                    MesoRD:units="um"/>
        </MesoRD:translation>
      </annotation>
    </compartment>
    <compartment id="CompartmentTwo">
      <annotation xmlns:MesoRD="http://www.icm.uu.se">
        <MesoRD:difference>
          <MesoRD:box MesoRD:x="0.4"
                      MesoRD:y="4.0"
                      MesoRD:z="0.4"
                      MesoRD:units="um">
            <MesoRD:pbc MesoRD:x="false"
                        MesoRD:y="true"
                        MesoRD:z="false"/>
          </MesoRD:box>
          <MesoRD:compartment MesoRD:id="CompartmentOne"/>
        </MesoRD:difference>
      </annotation>
    </compartment>
  </listOfCompartments>

<listOfSpecies>
  <species id="A1"
           compartment="CompartmentOne"
           hasOnlySubstanceUnits="true"
           initialAmount="10000"
           name="A"
           substanceUnits="item">
    <annotation xmlns:MesoRD="http://www.icm.uu.se">
      <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                        MesoRD:rate="6e-8"
                        MesoRD:units="cm2ps"/>
      <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                        MesoRD:rate="6e-8"
                        MesoRD:units="cm2ps"/>
    </annotation>
  </species>

  <species id="A2"
           compartment="CompartmentTwo"
           hasOnlySubstanceUnits="false"
           initialConcentration="1e-5"
           name="A"
           substanceUnits="mole"
           substanceVolume="litre">
    <annotation xmlns:MesoRD="http://www.icm.uu.se">
      <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                        MesoRD:rate="6e-8"
                        MesoRD:units="cm2ps"/>
      <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                        MesoRD:rate="3e-8"
                        MesoRD:units="cm2ps"/>
```

```
    </annotation>
  </species>
<listOfSpecies>
```

Other things to keep in mind when defining species are listed below.

- No matter how the initial amount of a species is declared, MesoRD internally uses numbers of molecules. If the initial amount is given as a concentration, the volume of the home compartment, *not* the entire system, is used in determining the amount of the species.
- There is no support for `boundaryCondition`, and its default value must not be changed by setting `boundaryCondition="true"`. This has to do with the fact that MesoRD does not implement SBML rates. Species can however be declared to be `constant`.
- The species `charge` is completely ignored.
- Though it is not incorrect to only specify the species in the compartments in which they physically can exist it is recommended (at least for now) to specify all species in all compartments. Why? The only reason is that the output from the program depends on the names of the species. If a species is not defined in a compartment it will take the name of a species in another compartment which might be confusing to interpret if you are looking at the output.

Parameter

Figure 3-5. The SBML definition of `Parameter`

Parameter	
id	SId
name	string { use=”optional” }
value	double { use=”optional” }
units	SId { use=”optional” }
constant	boolean { use=”optional” default=”true” }

A parameter in SBML associates a quantity with a symbolic name. The symbols may be used in mathematical formula which in turn define kinetic rate laws that control the reactions. Note also that reactions can have local parameters, as well as access to the global ones. Note that all parameters are effectively constant, since the lack of SBML rules means that there is no way to update a parameter value. The units tag needs to be defined as defined in the Section called *Unit*

- No matter what, MesoRD will *always* (re)create some additional parameters in the model. One of these is `volume_subVolume`, which gives the volume and unit of each individual sub volume. Another is `one_molecule_molar`, which gives the concentration of one molecule in a subvolume. Furthermore, for every compartment, MesoRD records a parameter with ID `COMPARTMENTID`, where `COMPARTMENTID` is the ID of the compartment. These latter parameters give the volume

and unit of the respective compartments. If one wants to use these parameters in the model, one should define them in the parameter section of the definition first. Before simulation begins, MesoRD will replace the values and units with values as calculated from the geometry.

Reaction

Figure 3-6. The SBML definition of `Reaction`

Reaction	
id	SId
name	string { use=”optional” }
reactant	SpeciesReference[0..*]
product	SpeciesReference[0..*]
modifier	ModifierSpeciesReference[0..*]
kineticLaw	KineticLaw { minOccurs=”0” }
reversible	boolean { use=”optional” default=”true” }
fast	boolean { use=”optional” }

Figure 3-7. The SBML definition of `SpeciesReference`

SpeciesReference	
species	SId
stoichiometry	double { use=”optional” default=”1” }
stoichiometryMath	StoichiometryMath { use=”optional” }

Figure 3-8. The SBML definition of `ModifierSpeciesReference`

ModifierSpeciesReference	
species	SId

Figure 3-9. The SBML definition of `KineticLaw`

KineticLaw

KineticLaw	
math	Math { namespace="http://www.w3.org/1998/Math/MathML" }
parameter	Parameter[0..*]
timeUnits	SId { use="optional" }
substanceUnits	SId { use="optional" }

An SBML reaction is a statement describing some transformation, transport or binding process that can change the amount of one or more species. Typically, an SBML reaction represents a chemical reaction which in turn describes how certain reactants are transformed into products. Reactions have associated kinetic rate expressions describing how often they take place.

Note that Finney and Hucka, 2003 states that reaction rate expressions should be written such that their units evaluate to "per second". This is a little awkward for the type of reactions MesoRD deals with, because doing so would require one to include the reaction volume in the mathematical expression. In MesoRD the reaction volume is identical to the volume of a subvolume, and this quantity in turn is not necessarily part of the model. Therefore, MesoRD allows reaction rates to be specified in molar per second. If a rate expression evaluates in concentration per time MesoRD attempts to transform the rate to in molar per second automatically. If the reaction rate has the per second unit, MesoRD will assume you know what you are doing and apply no transformations. Thus MesoRD has *relaxed* the SBML specification a little, not *restricted* it. However, to make thing simple we strongly recommend that your rate expressions evaluates in concentration per time, or you will have problems every time you change the subvolume size.

- Note that only a subset of MathML is supported by SBML for use in kinetic rate laws. To make matters even worse, only a subset of the SBML-supported features of MathML are supported in MesoRD. The SBML-supported MathML that is unsupported in MesoRD is limited to some of the less common trigonometric functions. The reason is that internally, MesoRD for efficiency reasons only works with real numbers, as opposed to complex numbers. On the other hand, the reactions one could define using the unsupported features, would probably be non-physical.
- Internally MesoRD ignores the modifier species since they only defined which species that are allowed to be included into the kineticLaw.
- In contrast to the SBML standard, all species IDs occurring in the kinetic rate expressions are treated as concentrations counted in Molar. The volume used in calculating concentrations from absolute numbers of molecules in a subvolume is `volume_subVolume`.
- MesoRD does currently not support reversible reactions. Since the default is that any given reaction is reversible, reactions must include `reversible="false"`. If one wants reversible reactions, one should define an additional reaction going backwards relative to the former.
- The `fast` argument is completely ignored by MesoRD. `fast` reactions are not meaningful in the stochastic treatment as performed by the program.
- For species, using the `stoichiometryMath` is unsupported. Rather, one must define the `stoichiometry` as a constant, floating point value. It is actually recommended in Finney and

Hucka, 2003 to do it this way (event though the specification also says that software, for inter-operability reasons, should be able to handle all possible cases).

- Because user-defined functions are not supported by MesoRD, we cannot have functions in the kinetic rate expressions.
- Reactions in MesoRD occur in a single compartment only. This is different from SBML, where reactions are multi-compartment by design. There is however no need to specify the compartment in which a reaction occurs. Since the species that take part in a reaction are defined in a unique compartment, MesoRD can figure out the compartment to which a reaction belongs by checking the participating species. This can obviously not be done if the participating species were defined in different compartments. Therefore: all species that take part in a given reaction must be defined in the same compartment and the species must be referred to by the species id, not their names.
- `substanceUnits` and `timeUnits` are ignored. Rather, they are deduced from the expressions. Actually, if a kinetic law is defined so that the unit evaluates to molar per second, MesoRD will internally multiply by the volume of a subvolume and Avogadro's number, so that the reaction unit correctly evaluates to per second.

This perticular version of MesoRD also allows for reaction rates to be calculated using the number of moleulces in nearest neighbouring subvolumes. To specifiy if the reaction include this type of behaviour or not, an annontaiton is added to the reaction node. *Please note that the flags* `MesoRD:inter_sv_assn_reaction` *and* `MesoRD:inter_sv_diss_reaction` *are mandatory fields.* The example below show an association reaction with nearest neighbour interactions turned on.

Example 3-5. Reaction Defintion

```
     <reaction id="Reaction" reversible="false">
<annotation xmlns:MesoRD="http://www.icm.uu.se">
  <MesoRD:inter_sv_assn_reaction MesoRD:state="true"/>
  <MesoRD:inter_sv_diss_reaction MesoRD:state="false"/>
</annotation>
<listOfReactants>
  <speciesReference species="B"/>
  <speciesReference species="A"/>
</listOfReactants>
<listOfProducts>
  <speciesReference species="C"/>
</listOfProducts>
<kineticLaw>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <apply>
      <times/>
      <ci> ka </ci>
      <ci> A </ci>
      <ci> B </ci>
    </apply>
  </math>
</kineticLaw>
     </reaction>
```

The file `steady_state.xml` in the repository of test files that is bundled with MesoRD contains a complete example including nearest neighbour interactions.

The example below show the very hypothetical reaction A+B->C , with a reaction-rate constant k.

Example 3-6. Reaction Defintion.

```
<reaction id="Reaction1" reversible="false">
  <listOfReactants>
    <speciesReference species="A"/>
    <speciesReference species="B"/>
  </listOfReactants>
  <listOfProducts>
    <speciesReference species="C"/>
  </listOfProducts>
  <kineticLaw>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <times/>
        <ci>k</ci>
        <ci>A</ci>
        <ci>B</ci>
      </apply>
    </math>
  </kineticLaw>
</reaction>
```

Event

SBML provides events for modelling instantaneous, discontinuous change in a set of variables of any type. Events are triggered by certain conditions. Events are also not supported by MesoRD.

Function

SBML allows the definition of named mathematical functions that may be used throughout the rest of a model. This is a feature MesoRD does not support.

Rule

Rules in SBML are mathematical expressions used in combination with any equations derived from the reactions. Rules can be used to establish constraints between variables, define variable values in terms of other variables or define the rate of change of a variable. Rules are not supported at all by MesoRD.

Notes

1. The recently added SBML feature SpeciesType is not yet implemented in MesoRD
2. Rather, defining a part of a real cell compartment as a MesoRD compartment has proven to be very useful to create specific initial distributions of molecules or for monitoring the time evolution of a specific part of a cell.
3. The latest SBML standard implements a feature called SpeciesType which may actually solve this problem, MesoRD does, however, not yet support this feature

Chapter 4. Constructive Solid Geometry

The previous chapters introduced several differences between standard SBML and the particular dialect MesoRD understands. In this chapter attention is turned to the last remaining aspect of writing SBML for MesoRD: how to specify the geometry of compartments.

Volumetric system geometry is essential to the way MesoRD works: simulating diffusion in three dimensions would be difficult without some space to diffuse in. As was mentioned in the Section called *Compartment* in Chapter 3, each compartment in MesoRD must have its own geometry definition. The geometry of the entire system is the *union* of all compartment geometries. The volume of the system geometry must be discretized in terms of subvolumes before it is ready for simulation use.

Current versions of SBML have no means to define volumetric geometries [Finney and Hucka, 2003]. However, the issue is being worked on, and in a not too distant future there may be standard ways to deal with geometries. [1]. In the meantime, this entire chapter describes a MesoRD-specific extension to SBML.

Introduction to Constructive Solid Geometry

Since subvolumes are inherently three-dimensional, a geometry definition language for MesoRD is not required to handle anything but volumes [2]. Neither users nor developers of MesoRD are expected to be highly skilled mathematicians. Thus, the geometry language should be easy to speak, yet powerful enough to enable construction of non-trivial geometries. A very intuitive way to describe geometry comes from computer aided design and is also used in many three-dimensional modelling packages. It is called constructive solid geometry[3], or CSG for short.

CSG gives a high-level description of geometry. It is intuitive because its syntax resembles the way humans [4] may describe objects in space. Reverting for a moment to two dimensions, the object in Figure 4-1 could, sacrificing any mathematical accuracy, be described as ”a circle with four rectangles sticking out its sides” or ”two rectangles crossed on top of a circle”. Having a computer come up with such a description is surprisingly hard, mainly because humans, unlike computers, can ”see” the circle in spite of the fact that its characteristic circumference is interrupted by rectangles. As it turns out, having the same computer *generate* a geometry from such a description is relatively simple.

Figure 4-1. A Simple Two-Dimensional Object

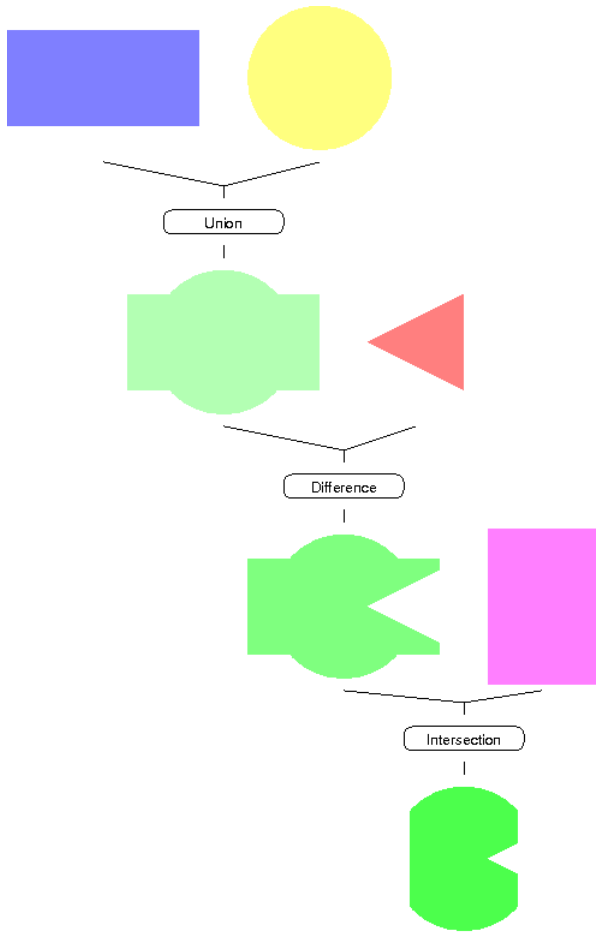
This object could be described as ”a circle with four rectangles sticking out its sides” or ”two rectangles crossed on top of a circle”.

Theoretically, CSG is a functional procedure that defines a combined, solid, volumetric object from a number of primitive solid volumes by the application of bounded, Boolean set operations [Burger and Gillies, 1989] [Foley et al., 1996]. The set of primitive volumes varies with the particular CSG implementation, while the set of operations is quite similar across the board. MesoRD's set of primitives, the parameters needed to define them as well as transformations applicable to them, are inspired by the Virtual Reality Modelling Language, VRML [Hartman and Wernecke, 1996]. Note that standard VRML does *not* actually support CSG -- its geometric primitives cannot be combined using set operations. Also, VRML is not an extended markup language. There are efforts underway to address both these issues.

Tree Representation of CSG

A CSG procedure can be represented by a tree structure. The root of the tree defines the object of interest, and the leaf nodes are geometric primitives. In between the root and the leaves lie operator nodes. The root object is determined by combining the leaves according to the operator nodes. In a CSG procedure, the operators are either transformations or set operations. A simple, two-dimensional, CSG procedure without transformations is illustrated in Figure 4-2.

Figure 4-2. Constructive Solid Geometry Set Operations in Two Dimensions



Set operations and geometric primitives in a simple two-dimensional CSG procedure. This particular specimen is one of the few trees in computer-land that has its root pointing towards the bottom of the page. The blue box is combined with the yellow circle in a union operation, resulting in the upper green object. The red triangle is subtracted from the green object using the difference operator. The last operation, intersection, is illustrated at the bottom part of the figure. For clarity, all transformation operators have been omitted. Note also that this is a binary tree. In general, CSG trees need not be binary: union and intersection nodes can have any number of children.

In general, the Boolean-valued set operations used to combine the primitives are not commutative. Therefore, the edges of the tree need to be ordered. It is not difficult to see that exchanging the operands of a difference operation will yield different results, just as $a - b \neq b - a$. The ordering of operands is relevant from a computational aspect as well: if a point in space is contained by the first operand of a union, the algorithm need not spend time checking for containment in the remaining operands.

Other properties of the tree representation of a CSG procedure are:

- The leaf nodes always define a geometric primitive. The inverse is also true: every geometric primitive is represented by a leaf node. By definition, leaf nodes do not have any children. In MesoRD, the `compartment` primitive, see the Section called *The `compartment` Primitive*, is a strange exception: it is a ”leaf” node that could be said to have children.
- Transformation nodes only have one single child.
- Difference nodes have exactly two children, such that the sought object is defined by the volume of the left node *minus* the volume of the right node. Intersection and union nodes can have any number of children.
- A CSG tree does not provide a unique representation. As was exemplified in the figure caption, there are at least two ways to describe the object in Figure 4-2. Most objects could be described by several different CSG procedures.

In the world of MesoRD, every volume element of the full geometry must belong to exactly one compartment. Because every compartment must define its own CSG tree, it is possible to specify systems with overlapping CSG trees, so that some volume element is ”owned” by several compartments. This is not an error *per se*, but merely a bad thing to do, as it will cause confusion when it comes to assigning the volume element to a compartment. The scenario is easily avoided using the difference set operation as described in the Section called *The `compartment` Primitive*. The responsibility of avoiding these situations lies entirely on the side of the user.

MesoRD’s geometries are defined by a hierarchical, textual, XML representation of the desired CSG procedure. The XML representation is merely a transcription of the CSG procedure in its tree form. Each element in the XML hierarchy is identified as either a set operation node, a transformation node or a primitive geometry node. The parameters needed to completely define each node, such as the side lengths in case of a box, or the displacements in case of a translation transformation, are supplied as attributes. The XML is kept as an annotation of the compartment, the geometry of which the corresponding CSG procedure defines. The namespace of any such annotations must be http://www.icm.uu.se, otherwise they will be ignored by MesoRD.

> **Note:** The entire purpose of the CSG algorithm is to transform a compartmentalised volume description to a compartmentalised subvolume discretization. Exactly how this is done will not be discussed here at all. The algorithm *is* described in the source-level documentation, and also in this report.

CSG Trees in MesoRD

There are many ways to implement CSG. The details will vary with the intended application. The MesoRD CSG implementation draws its inspiration from a number of sources, most notably VRML.

Geometric Primitives

Primitives in MesoRD can be boxes, cones, cylinders, meshes and spheres. Just like objects in VRML, primitives are defined centred at the origin of a *local*, right-handed coordinate system [Hartman and Wernecke, 1996]. In the case of cones and cylinders, the axis of symmetry is defined to lie parallel to the local `y`-axis. The local coordinate systems are related to the *global* coordinate system, the right-handed coordinate system of the full geometry, by a set of affine transformations.

Note: In the examples below it is assumed that the unit `um` is defined as one micrometre, *i.e.*

```
<unitDefinition id="um">
  <listOfUnits>
    <unit kind="metre" scale="-6"/>
  </listOfUnits>
</unitDefinition>
```

is somewhere in the `listOfUnitDefinitions`.

> **Warning**
>
> The format of the unit definitions has changed since earlier versions of MesoRD (version 0.2). Each node now needs to specify a unit by means of the `MesoRD:units` parameter. This is the unit in which all lengths, radii, angles, *etc.* of the primitive are to be interpreted.

The `box` Primitive

The `box` node defines is a block-shaped geometric object centred at the origin of the local coordinate system. Its sides are aligned with the local coordinate axes. The parameters in defining a box are the side lengths of the box along the `x`-, `y`- and `z`-axis. The example below defines a 5 by 30 by 10 `um` box. It extends 5 `um` in the `x`-direction, 30 `um` in the `y`-direction and 10 `um` in the `z`-direction.

Example 4-1. Box Geometry

```
<compartment id="box">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:box MesoRD:x="5"
                MesoRD:y="30"
                MesoRD:z="10"
                MesoRD:units="um"/>
  </annotation>
</compartment>
```

The `cone` Primitive

The `cone` node defines a simple cone oriented along the local `y`-axis. An untransformed cone is centred at the origin, extending half its height along the positive `y`-axis and half its height along the negative `y`-axis. The parameters in defining a cone are the `bottomradius` and the `height`. The example below defines a cone with radius 5 `um` and height 20 `um`.

Example 4-2. Cone Geometry

```
<compartment id="cone">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:cone MesoRD:bottomradius="5"
                 MesoRD:height="20"
                 MesoRD:units="um"/>
  </annotation>
</compartment>
```

The `cylinder` Primitive

The `cylinder` node builds a simple, capped cylinder centred at the origin and oriented along the local `y`-axis. The cylinder extends half its height along the positive `y`-axis and half its height along the negative `y`-axis. The parameters in defining a cylinder are the `radius` and the `height`. The example below defines a cylinder with radius 5 `um` and height 20 `um`.

Example 4-3. Cylinder Geometry

```
<compartment id="cylinder">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:cylinder MesoRD:height="20"
                     MesoRD:radius="5"
                     MesoRD:units="um"/>
  </annotation>
</compartment>
```

Cylinders are really special cases of generalised cones, in the sense that the radius is not dependant on the `y`-coordinate. This means that the radius at the base is always equal to the radius at the top. The fact that the areas of the base and the top are equal is also what makes periodic boundary conditions, see the Section called *Periodic Boundary Conditions*, possible for cylinders.

The Triangular `mesh` Primitive

The `mesh` primitive enables non-trivial geometries to be built without worrying about set operations or transformations. The mesh is probably the most powerful primitive available in MesoRD's CSG implementation, as it allows for the creation of natural shapes. A mesh is in principle a list

of *sided* triangles -- sided meaning that the triangles have a front face and a back face. A triangle is an ordered list of exactly three vertices. A vertex is one of the three corners of a triangle. The triangles are not necessarily centred at the origin.

The sidedness of a triangle is determined by computing the normal of the triangle, $\mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)$, where $\mathbf{p}_0$, $\mathbf{p}_1$ and $\mathbf{p}_2$ are the ordered vertices of the triangle. By convention, the normal is taken to point away from the front face of the triangle. Equivalently, the normal is assumed to point out of the solid enclosed by all triangles. Another way to state the above is this: when looking at the triangle's front face, that is when inspecting the exterior surface of the solid, the vertices of the triangle should be listed in counterclockwise order. The volume behind the list of triangles should be closed. Interesting surprises await those who define unclosed meshes.

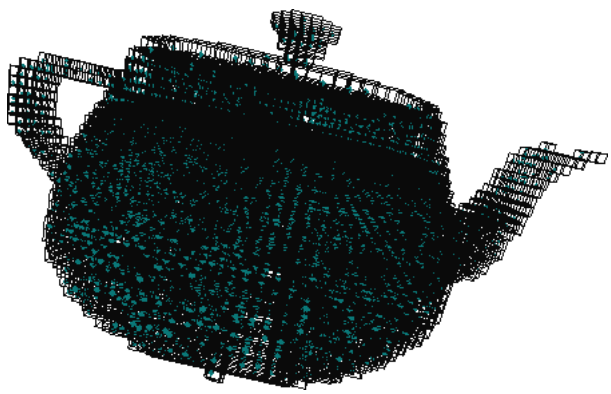
The example below is possibly the simplest instance of a `mesh` object: a tetrahedron with a side length of 8 `um`. A bigger example is shown graphically in Figure 4-3. By counting the number of lines required for a typical `mesh` object, it becomes apparent that the added flexibility comes at the price of increased verbosity. However, it should not be too difficult to write tools to convert models built in external modelling programs to MesoRD `mesh` objects. In fact, SBML is designed to be machine-written, not human-written. Since CSG is a MesoRD-specific extension to SBML, there are no external tools for generating geometries. Yet.

Example 4-4. Mesh Geometry

```
<compartment id="mesh">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:mesh>
      <MesoRD:triangle>
        <MesoRD:vertex MesoRD:x="0"
                       MesoRD:y="4"
                       MesoRD:z="0"
                       MesoRD:units="um"/>
        <MesoRD:vertex MesoRD:x="-4"
                       MesoRD:y="0"
                       MesoRD:z="0"
                       MesoRD:units="um"/>
        <MesoRD:vertex MesoRD:x="4"
                       MesoRD:y="0"
                       MesoRD:z="0"
                       MesoRD:units="um"/>
      </MesoRD:triangle>
      <MesoRD:triangle>
        <MesoRD:vertex MesoRD:x="0"
                       MesoRD:y="4"
                       MesoRD:z="0"
                       MesoRD:units="um"/>
        <MesoRD:vertex MesoRD:x="4"
                       MesoRD:y="0"
                       MesoRD:z="0"
                       MesoRD:units="um"/>
        <MesoRD:vertex MesoRD:x="0"
                       MesoRD:y="0"
                       MesoRD:z="-4"
                       MesoRD:units="um"/>
```

```
      </MesoRD:triangle>
      <MesoRD:triangle>
        <MesoRD:vertex MesoRD:x="0"
                       MesoRD:y="4"
                       MesoRD:z="0"
                       MesoRD:units="um"/>
        <MesoRD:vertex MesoRD:x="0"
                       MesoRD:y="0"
                       MesoRD:z="-4"
                       MesoRD:units="um"/>
        <MesoRD:vertex MesoRD:x="-4"
                       MesoRD:y="0"
                       MesoRD:z="0"
                       MesoRD:units="um"/>
      </MesoRD:triangle>
      <MesoRD:triangle>
        <MesoRD:vertex MesoRD:x="-4"
                       MesoRD:y="0"
                       MesoRD:z="0"
                       MesoRD:units="um"/>
        <MesoRD:vertex MesoRD:x="0"
                       MesoRD:y="0"
                       MesoRD:z="-4"
                       MesoRD:units="um"/>
        <MesoRD:vertex MesoRD:x="4"
                       MesoRD:y="0"
                       MesoRD:z="0"
                       MesoRD:units="um"/>
      </MesoRD:triangle>
    </MesoRD:mesh>
  </annotation>
</compartment>
```

Figure 4-3. The Utah Teapot in MesoRD CSG



The Utah teapot as a MesoRD `mesh` primitive with 3751 triangles. The 18761 lines long SBML code for this geometry was generated from an `s3d`-file using a small shell script.

There is no CSG "standard". Meshes are "non-standard" only in the sense that many CSG implementations do not support them.

> **Warning**
>
> The `mesh` primitive is a *highly experimental* feature! In fact, the are only a few cases in which the `mesh` primitive has been tested and is *known* to work. Those who try the `mesh` primitive will discover that geometry discretization takes significantly longer than it does for the simpler primitives.

The `sphere` Primitive

The `sphere` primitive defines a sphere centred at the origin. The single parameter defining a sphere is the `radius`. The example below defines a sphere with radius 5 `um`.

Example 4-5. Sphere Geometry

```
<compartment id="sphere">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:sphere MesoRD:radius="5"
                   MesoRD:units="um"/>
  </annotation>
</compartment>
```

The `compartment` Primitive

MesoRD supports one more primitive: the special `compartment` pseudo-primitive that allows the geometry of another compartment to be treated just like any other geometric primitive. The `compartment` primitive is not a *true* primitive, because it is not a proper terminal node.

While this sort of "copy operation" allows for new creative ways in the definition of geometries, the main reason for its inception is to allow one compartment to surround another compartment. As was mentioned in the Section called *Tree Representation of CSG*, it is not a good idea to define geometries so that a point in space could belong to several compartments. Using the `compartment` primitive together with the difference set operation, the *containing* compartment may be defined as the left operand of a difference operation. The right operand is set to the geometry of the *contained* compartment. This will create a hole in the containing compartment, just big enough to fit the contained compartment.

A referenced compartment is identified by its unique `id` field. The example below makes use of the geometry of the `original` compartment, defined elsewhere, in the definition of the `copy` compartment by means of a difference operation.

Example 4-6. The `compartment` Primitive

```
<compartment id="copy">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:difference>
      <MesoRD:box MesoRD:x="5"
                  MesoRD:y="30"
                  MesoRD:z="10"
                  MesoRD:units="um"/>
      <MesoRD:compartment MesoRD:id="original"/>
    </MesoRD:difference>
  </annotation>
</compartment>
```

> **Note:** Previous version of MesoRD required a compartment to be defined *before* any reference through the `compartment` primitive. This requirement is now lifted. However, creating cycles is not permitted. An easy way to create a cycle is to create a compartment `foo` that references `bar` that in turn references `foo`.

Set Operations

The set of transformed primitives, or objects, are combined using difference, intersection and union set operations. These operations take two or more operand objects and combine them into a new object. Objects resulting from a set operation can act as operand objects in transformations or other set operations.

> **Note:** The unary complement operation, which in combination with the union operation would obsolete the difference operation, is not supported, since MesoRD does not define a "universe".

The `difference` Operation

The `difference` set operation computes the difference between exactly two operands. The second child of a `difference` node is subtracted from the first. The `difference` operation has no param-

eters. The example below defines the geometry that results when a sphere with radius 2 `um` and centred at the origin, is subtracted from a cubic box with sides 8 `um`, also centred at the origin.

Example 4-7. Difference Operation

```
<compartment id="difference">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:difference>
      <MesoRD:box MesoRD:x="8"
                  MesoRD:y="8"
                  MesoRD:z="8"
                  MesoRD:units="um"/>
      <MesoRD:sphere MesoRD:radius="2"
                     MesoRD:units="um"/>
    </MesoRD:difference>
  </annotation>
</compartment>
```

The `intersection` Operation

The `intersection` set operation computes the intersection of an arbitrary number of operand geometries. The `intersection` operation has no parameters. The example below defines the geometry that results when a cubic box with sides 8 `um` and centred at the origin is intersected with a sphere with radius 5 `um`, also centred at the origin.

Example 4-8. Intersection Operation

```
<compartment id="intersection">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:intersection>
      <MesoRD:box MesoRD:x="8"
                  MesoRD:y="8"
                  MesoRD:z="8"
                  MesoRD:units="um"/>
      <MesoRD:sphere MesoRD:radius="5"
                     MesoRD:units="um"/>
    </MesoRD:intersection>
  </annotation>
</compartment>
```

The `union` Operation

The `union` set operation computes the union of an arbitrary number of operand geometries. The `union` operation has no parameters. The example below defines the geometry that results when a cubic box with sides 9 `um` and centred at the origin is united with a sphere with radius 5 `um`, also centred at the origin.

Example 4-9. Union Operation

```
<compartment id="union">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:union>
      <MesoRD:box MesoRD:x="9"
                  MesoRD:y="9"
                  MesoRD:z="9"
                  MesoRD:units="um"/>
      <MesoRD:sphere MesoRD:radius="5"
                     MesoRD:units="um"/>
    </MesoRD:union>
  </annotation>
</compartment>
```

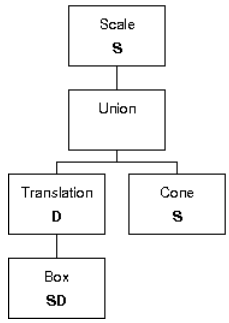
Transformations

Clearly the shape of a compound object depends on the exact location and orientation of the individual primitives. Therefore, CSG trees in MesoRD contain transformation nodes in addition to the set operation nodes. Transformations allow subtrees to be rotated, scaled, sheared and translated. These are all affine transformations, preserving the parallelism of lines, but not lengths or angles. Every primitive has its own transformation matrix which relates its local coordinate system to the global coordinate system. For example, a cylinder parallel to the `x`-axis is defined by rotating a cylinder node by 90 degrees around the `z`-axis. In case no transformation is specified, the unit transformation is assumed. Transformations have no direct effect on set operations, but are propagated to its children.

A transformation node will affect all leaf nodes below it in the tree. The transformation that will take a primitive leaf node from its local coordinate system to the global coordinate system is defined by the composition of all transformations on the path between the root and the respective leaf. The compound transformation is calculated as the matrix product of the corresponding transformation matrices.

An example which describes the union of a cone and a box is illustrated in Figure 4-4. The root of the tree is a scale node, which defines a scale matrix $\mathbf{S}$. The path from the root to the cone primitive only passes through one transformation node, namely the root. Thus, the transformation matrix relating the local coordinate system, in which the cone is defined, to the global coordinate system of the entire procedure, is $\mathbf{T}_c = \mathbf{S}$.

Figure 4-4. A CSG Tree with Transformations



The tree describes the union of a cone and a box. The box is translated with respect to the cone. The union of both primitives is scaled.

The box node is separated from the union node by a translation node which defines a displacement matrix $\mathbf{D}$. The path from the box primitive to the root passes through two transformation nodes. The transformation matrix that relates the local coordinate system of the box to the global coordinate system is $\mathbf{T}_b = \mathbf{SD}$.

The `rotation` Transformation

The `rotation` transformation in MesoRD is defined by an axis of rotation and a rotation angle. The rotation axis is given in the local coordinate system, its length is irrelevant. Thus, the parameters of a `rotation` node are the Cartesian components of the direction of the rotation axis and the counterclockwise rotation angle. The example below defines the geometry that results when a cube with side length 5 `um` is rotated 20 degrees around the local `z-axis`. One could just as well have used radians for the rotation angle, since SBML defines a `radian` unit.

Example 4-10. Rotation Transformation

```
<compartment id="rotation">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:rotation MesoRD:x="0"
                     MesoRD:y="0"
                     MesoRD:z="1"
                     MesoRD:angle="20"
                     MesoRD:units="degree">
      <MesoRD:box MesoRD:x="5"
                  MesoRD:y="5"
                  MesoRD:z="5"
                  MesoRD:units="um"/>
    </MesoRD:rotation>
  </annotation>
</compartment>
```

The `scale` Transformation

The `scale` transformation in MesoRD is defined by a scaling vector. The scaling vector consists of the diagonal elements of the diagonal, affine scale transformation matrix. The elements of the scaling vector give the scale factor in each of the three Cartesian dimensions in the local coordinate system. Thus, the parameters of a `scale` node are the components of the scale vector. The example below defines the geometry that results when a sphere with radius 5 is scaled by a factor of two along the local `z-axis`.

Example 4-11. Scale Transformation

```
<compartment id="scale">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:scale MesoRD:x="1"
                  MesoRD:y="1"
                  MesoRD:z="2">
      <MesoRD:sphere MesoRD:radius="5"
                     MesoRD:units="um"/>
    </MesoRD:scale>
  </annotation>
</compartment>
```

The `shear` Transformation

The `shear` transformation in MesoRD is defined by a shear axis and a shear angle. The shear axis is given in the local coordinate system, its length is irrelevant. Thus, the parameters of a `shear` node are the Cartesian components of the direction of the shear axis and the counterclockwise shear angle. The example below defines the geometry that results when a cube with side length 5 is sheared 2 radians around the local `z-axis`.

Example 4-12. Shear Transformation

```
<compartment id="shear">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:rotation MesoRD:x="0"
                     MesoRD:y="0"
                     MesoRD:z="1"
                     MesoRD:angle="2"
                     MesoRD:units="radian">
      <MesoRD:box MesoRD:x="5"
                  MesoRD:y="5"
                  MesoRD:z="5"
                  MesoRD:units="um"/>
    </MesoRD:rotation>
  </annotation>
</compartment>
```

The `translation` Transformation

The `translation` transformation in MesoRD is defined by a displacement vector. The components of the displacement vector give the displacements in each of the three Cartesian dimensions in the local coordinate system. Thus, the parameters of a `translation` node are the components of the displacement vector. The example below defines the geometry that results when a sphere with radius 5 is moved two `um` two along the positive local `z-axis`.

Example 4-13. Translation Transformation

```
<compartment id="translation">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:translation MesoRD:x="0"
                        MesoRD:y="0"
                        MesoRD:z="2"
                        MesoRD:units="um"/>
      <MesoRD:sphere MesoRD:radius="5"
                     MesoRD:units="um"/>
    </MesoRD:translation>
  </annotation>
</compartment>
```

Periodic Boundary Conditions

Unlike many other CSG implementations, MesoRD supports periodic boundary conditions, or PBC:s, for boxes and cylinders. For a box, PBC:s mean that a pair of opposing sides are topologically equivalent: if a molecule diffuses out one side, it will reappear at the other. For a cylinder[5], periodic boundary conditions, or toroidal boundaries, mean that molecules leaving through one circular cap will reappear at the other. Periodic boundary conditions have an important history in the sort of simulations MesoRD deals with, since they allow the creation of ”infinite” geometries.

> **Note:** A subvolume on the boundary of a primitive need not be a boundary subvolume of the *total* geometry: if the geometry in Figure 4-1 is defined by the union of two crossed boxes and a sphere, the boxes have "virtual boundaries" inside the sphere. These boundaries are not *true* boundaries of the total geometry, since they are contained within the sphere. Periodic boundary conditions on virtual boundaries are pathological cases: it makes little sense to define an object with PBC, only to subsequently hide the periodic boundaries within a bigger volume. Such situations should generally be avoided; periodic boundaries should also be true boundaries.

In the case of boxes there can be three separate boundary conditions along the `x`-, `y`- and `z`-axes. The example below shows how to define boundary conditions for the box along the `z`-axis. Molecules diffusing out the negative `z`-end of the box will reappear at the positive `z`-end and *vice versa*.

Example 4-14. Box Periodic Boundary Condition

```
<compartment id="box_pbc">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:box MesoRD:x="5"
                MesoRD:y="5"
                MesoRD:z="5"
                MesoRD:units="um">
      <MesoRD:pbc MesoRD:x="false"
                  MesoRD:y="false"
                  MesoRD:z="true"/>
    </MesoRD:box>
  </annotation>
</compartment>
```

Cylinders can have periodic boundary conditions only along the `y`-axis. A cylinder with boundary conditions is topologically equivalent to a torus. For instance:

Example 4-15. Cylinder Periodic Boundary Condition, or toroidal boundary condition

```
<compartment id="cylinder_pbc" units="um">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:cylinder MesoRD:height="5"
                     MesoRD:radius="2"
                     MesoRD:units="um">
      <MesoRD:pbc MesoRD:y="true"/>
    </MesoRD:cylinder>
  </annotation>
</compartment>
```

If the `MesoRD:pbc` tags are omitted, there will not be any boundary conditions active.

Issues with MesoRD CSG

The CSG extension in MesoRD's SBML dialect is by far the biggest departure from the standard. In addition to all the warnings sprinkled throughout this chapter, there are some additional "features" that deserve special attention.

- Error checking is minimal. For instance, it makes no sense to specify a cylinder with radius -42. MesoRD will happily let you do that. Sometimes it handles bad input sensibly, other times it just breaks.
- As was noted in the Section called *Compartment* in Chapter 3, one may inadvertently specify geometries that have dimensionality less than three by discretising the geometry with to large subvolumes.

Notes

1. Different proposals for geometry support in upcoming versions of SBML are discussed on the SBML Discussion Lists (http://www.sbml.org/forums).
2. In MesoRD it is impossible to simulate systems with spatial dimensionality different from three. However, surfaces may be modelled by ensuring the thickness of the geometry is much smaller than its area. Similarly, curves will need to be very thin in directions orthogonal to the tangent. Because any physical object in The Real World™ extends in all three dimensions, MesoRD's idea of a curve or a surface may arguably be more realistic than a *true* one- or two-dimensional object.
3. Also called "constructed solid geometry" by some authors [Angel, 1997].
4. At least, it resembles the way those humans who wrote this text would describe objects in space.
5. Periodic boundary conditions are the reason why MesoRD uses separate primitives for cones and cylinders. Cones could more generally be described by their height, the radius of the bottom cap, and the radius of the top cap. A "proper" cone would have its top cap radius equal to zero, and a cylinder would be defined by the special case when both radii are equal. Since PBC:s require topologically equivalent sides to be *exactly* the same size, they would not be applicable to generalised cones.

Chapter 5. The User Interface

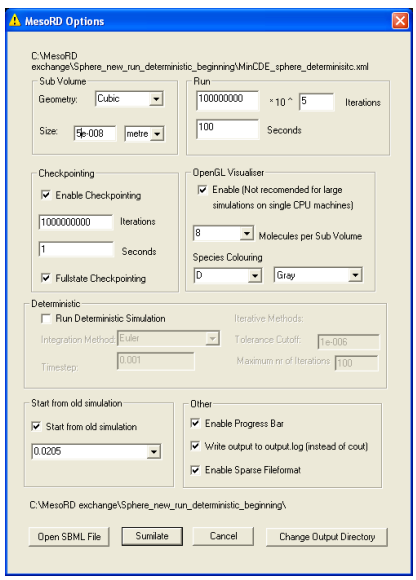
Available Options

MesoRD comes with two different user interfaces, that both accomplish approximately the same thing. The graphical user interface as shown in Figure 5-1 has not been implemented on all supported platforms yet. The command-line interface is more efficient to use, but may be intimidating to users not accustomed to it. Since version 0.3, MesoRD also has a command line interface for Windows. The synopsis for the **mesord** executable is as follows:

mesord [-C *its*] [-c *secs*] [-l | -L] [-d *molecs*] [-g | -G] [-h] [-i *its*] [-I *scale*] [-p | -P] [-q *size*] [-s | -S] [-t *secs*] [-V] [-D] [-T *secs*] [-r *tol*] [-x *its*] [-m *method*] [-o *line#*] [-e | -E] {*model*}

The last argument is the name of the SBML file that contains the model definition. Most other individual options are discussed further below. The options that will only be mentioned here are `h`, which prints a short help message to standard out, and `V` which prints version information to standard out before exiting normally. Note that MesoRD supports long option names on platform which have the `getopt_long` function. Check the output of **mesord -h** if in doubt.

Figure 5-1. The GUI on Windows XP



Sub Volume Geometry

MesoRD will build the geometry of the system using fixed size sub volumes. Theoretically, there are several differently shaped sub volumes one could use for this construction. Currently, MesoRD

only knows about one kind of sub volume: the cube.

To give physically reasonable results the size of the cubic subvolume must meet some criteria that are described in Chapter 6. However practically the most important criteria is the result of the simulations should not change if the size of the artificial subvolume changes a little. Thus we suggest that you run your simulations with doubled and halved subvolume size to make sure the result does not depend on the artificial discretisation.

`-q` *size*

Tells MesoRD to build the geometry using cubic sub volume of with the given size. Note that the size has to be given with a valid unit. All units specified in the model definition that evaluate to multiples of metres are valid units in this context. If the unit `um` from the Section called *Units* in Chapter 7 is in the model definition, the following will build the geometry using cubic sub volumes with side 0.1 micrometres:

```
... -q 0.1 um ...
```

Default subvolume size is 10^{-7} metres.

Simulation Duration

Any simulation is typically run until a certain termination criterion is met. One could imagine implementing all kinds of complicated convergence criteria. We have been very unimaginative with our termination criteria: a simulation is run either a certain number of iterations or a certain number of simulation seconds. Simulation is terminated when either of these conditions are met. There is one trick with setting the maximal number of simulation iterations: it may very well occur that the number of simulations one would want to run will not fit into a C long integer type. This may turn into a problem, especially on 32-bit architectures. Thus, the maximal number of iterations must be specified using a multiplier and a scale. The maximal number of iterations is then interpreted as the multiplier times ten to the power of the scale.

`-i` *its*

This option will specify the multiplier for the number of iterations to run. (Default: 10^8).

`-I` *scale*

This option will specify the scale for the number of iterations to run. (Default: 5).

`-t` *secs*

This option will specify the duration of the simulation using the clock of the simulated system. Note that this time is different from the wall clock time. (Default: 100 seconds).

`-s`

This option will suppress starting of simulation all together. If you specify this option, all that will be done is model parsing.

`-S`

This option will make sure that the system is actually simulated (default).

Checkpointing

A checkpoint is a dump of the current state of the system. There are two types of checkpointing in MesoRD: Full state checkpointing and compartment state checkpointing. In full state checkpointing the whole state of the system, i.e. the number of molecules in the different subvolumes is append to several files, as further discussed in the Section called *Making Sense of MesoRD Output*. If the whole state is saved, it is possible to restart the a simulation from this point in time. In compartment state checkpointing the number of molecules of different species are saved for each compartment.

`-C` *`its`*

This option will control how many iterations will pass between checkpoints. The system is checkpointed whenever `its` iterations have passed since the last checkpoint. (Default: 10^9).

`-c` *`secs`*

This option will control how many seconds will pass between checkpoints. The system is checkpointed whenever `secs` seconds have passed since the last checkpoint. Note that this time is measured with respect to the simulation system clock, not the wall clock. (Default 1 second).

`-l`

This option will turn the full state checkpointing off.

`-L`

This option will turn the full state checkpointing on (default).

Note that if both `its` and `secs` are greater than zero, MesoRD will checkpoint whenever `its` iterations *or* `secs` seconds have passed since the last checkpoint. Since checkpoints can consume much disk space and are quite expensive in terms of running time, this may not be desirable.

Sparse Output

In stochastic simulations there will be a high number of empty subvolumes, especially if the number of molecules is low, therefor MesoRD has the possibility to write the full state checkpoints in a sparse format. The format of the files is described in the Section called *Making Sense of MesoRD Output*. *Note: The deterministic simulation mode has no support for the sparse format, weird this might happen if you use them toghether.*

`-e`

This will turn the sparse output format off (default for deterministic simulations).

`-E`

This will turn the sparse output format on (default for stochastic simulations).

Visual Appearance

MesoRD contains a few visual elements. The most prominent of these is the visualiser, which enables real-time viewing of the system. Using the visualiser it is possible to see whether the simulation does what one would expect and it thus enables debugging of model definitions. The visualiser is *not* intended for any accurate analysis of simulation results. The visualiser is also not necessary for the simulation per se, and since it consumes a fair share of available resources, its use is optional. If you have a poor graphics card, you are therefore recommended to use the visualiser only for debugging the SBML file. To make optimal use of the visualiser you should have a fast dual processor machine with a high end graphics card.

The least prominent visual element is the progress bar. The progress bar displays the current iteration number and the the current simulation time. The progress bar also enables clean, premature termination of a simulation.

`-g`

This option will suppress launching of the OpenGL visualiser.

`-G`

This option will launch the OpenGL visualiser once the simulation is started (default).

`-d` *`molecs`*

This option will control how many molecules are rendered in one sub volume. For cubic sub volumes, `molecs` is the number of molecules that will fit along one each dimension of the sub volume. The total number of molecules rendered in a cubic sub volume is thus `molecs` to the power of three (default 2).

`-p`

This option will suppress launching of the progressbar.

`-P`

This option will launch the progress bar once the simulation is started (default).

Deterministic mode

MesoRD will run a mean-field simulation of the model when the deterministic mode is selected. There are a few numerical integration methods to choose among. All of them will require a time step for each iteration. The implicit methods will in addition require some parameters for the equation solver.

`-D`

This will turn on the deterministic simulations in MesoRD. When this flag is set the sparse output format should not be used at all. It should also be noted that the visualiser will give very misleading information [1] since the visualiser is plotting integer number of molecules and the deterministic simulations are using a continuous description of the molecule numbers (default: the simulation mode is set to stochastic).

`-T` *secs*

This option will specify how large the time step will be for each iteration. The time step should always be given in seconds. Small time steps give more stable and accurate simulations but may on the other hand take very long time. The implicit methods (see deterministic models) can usually handle larger time steps but are also more time consuming in each step. (Default 0.001).

`-r` *tol*

When using implicit methods (see deterministic methods) a system of equations is solved in each time step. This option specify a cut-off limit for the iterative method used to solve this system of equations. The iterative method exits if the norm of the residual, r = Ax - b, from the equation system Ax = b is smaller then `tol`. (Default: 10^{-7}).

`-x its` *its*

The variable `its` specify the maximum number of iteration used when solving the system of equations. If the cutoff tolerance is not reached within `its` MesoRD will print an error message. (Default: 100).

`-m` *method*

This will specify which method to use in deterministic simulation. The available methods found in Figure 5-2 (default BDF2).

```
... -m BDF2 ...
```

Figure 5-2. The available deterministic methods

Methods	
Euler	First order accuracy explicit method.
EulerBackward	An unconditionally stable implicit method of first-order accuracy
Trapezoid	An unconditionally stable implicit method of second-order accuracy
RungeKutta4	The fourth-order Runge Kutta scheme. An explicit method.
BDF2	Implicit linear multi-step method, implemented in two steps.

Starting from an old checkpoint

A simulation can be resumed from an old checkpoint assuming that the SBML .xml file is read from a directory where a previous simulation has been run with the same geometry and discretisation and that all old checkpointing files are still present. (default this feature is turn off).

The output files in the directory will overwrite the old simulation data that is used for to define to start state, i.e. **make a copy of your old simulation data before starting from an old checkpoint.**

`-o` *`line#`*

Settings the `line#` will start the simulation in the state that can be found at checkpoint number `line#`.

```
... -o 234 ...
```

Note: You cannot start a deterministic simulation from a previous stochastic simulation that has been stored in the sparse format. You cannot start a stochastic simulation from a previous deterministic simulation, since there is not an integer number of molecules in the different subvolumes. If you try this there will be no warnings, just strange output from your new simulation.

Making Sense of MesoRD Output

A typical simulation will create several files. Note that the format of these output files is far from optimal and subject to change in future versions.

`geometry.txt`

This file contains whitespace-separated Cartesian `x-`, `y-` `z`-coordinates and the compartment name, in that order, of every sub volume in the geometry. The coordinates of each sub volume are printed on a separate line. The order in which the sub volumes appear is the same order in which MesoRD has stored them. Note that the file contains no information about how a subvolume is connected to other sub volumes. The compartment belonging is printed along with the coordinates. The unit for the coordinates is number of subvolumes.

`species_X.txt`

This file contains whitespace-separated number of molecules of species `X` for every sub volume and for each checkpoint. The sub volumes are listed in the order in which MesoRD has stored them. The contents at each checkpoint are printed on a separate line. If the sparse format has been chosen the files instead gives data only for non-zero subvolumes. In this case the number of subvolume is followed by the number of molecules in that subvolume, which is followed by the number of the next non-zero subvolume etc. The numbers of the subvolumes corresponds to the line numbers in the geometry file (where the first line is subvolume 0).

`time.txt`

This file contains the simulation time at which each checkpoint was written. Every checkpoint is printed on a separate line.

`output.log`

This file is only created at Windows systems (optionally). On other systems this information is sent to standard output. It contains parsing, geometry building, reaction folding information etc. and also error messages.

Note: If you want to trust the compartment state checkpointing will need to define speciesID:s for all species in every compartment.

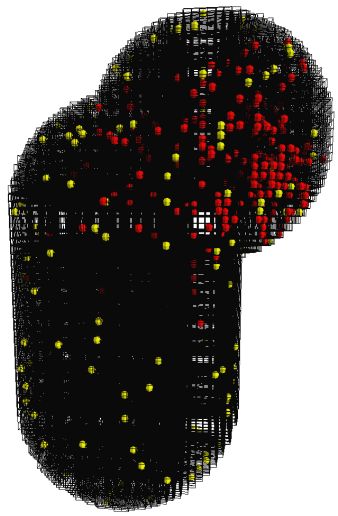
Note: You can not start a stochastic simulation from a previous deterministic simulation, since there is not an integer number of molecules in the different subvolumes. There will be no warnings, just strange output from your new simulation.

The OpenGL Visualiser

If the visualiser is enabled, it will render an approximation of the system. A screen shot is provided in Figure 5-3. There are a few commands to control the visualiser.

- Moving the mouse while pressing the left mouse button will rotate the system. Rotation is a little counterintuitive compared to how rotation works in other visualisers.
- Moving the mouse while pressing the left mouse button *and* holding down the **Ctrl** key will move the slabbing plane backwards and forwards along the z-axis. This adjusts how much of the z-axis extent of the geometry will be rendered.
- Moving the mouse while pressing the left mouse button *and* holding down the **shift** key will move the camera backwards and forwards along the z-axis. This is equivalent to zooming.
- Pressing the **G** key toggle display of the grid. The grid is what marks the individual sub volume boundaries.
- Pressing the **C** key toggle display of the connectors. The connectors are arrows between sub volumes that are topologically connected. This feature is intended for debugging, mostly.

Figure 5-3. The OpenGL Visualiser



The Visualiser versus The Absolute Truth

The visualiser cannot be trusted to give a *completely* accurate picture of the system. One multiprocessor systems, the visualiser will typically run on its own CPU, while the simulation itself runs elsewhere. Thus the system may change while the visualiser is rendering it, and the instantaneous picture as seen on the screen may actually be a blend of one ore more simulation steps. Furthermore, the number of molecules in a sub volume is typically much larger than what can be displayed. When rendering a subvolume, the visualiser will cycle through the different species in the sub volume, rendering one molecule of each species in every cycle. If there are no molecules of a certain species, the visualiser proceeds to the next. Thus, if there were one thousand `A` molecules and only one `B` molecule in the sub volume, but the visualiser could fit eight molecules into a sub volume, the sub volume will be rendered containing seven `A` molecules and one `B` molecule.

Notes

1. More then usual, that is.

Chapter 6. Theory: How MesoRD does Stochastic and Deterministic Simulation of Reaction-Diffusion Kinetics

Introduction

Chemical reactions are stochastic event, meaning that is not possible to know when and where the next reaction will occur. The probabilities for the reaction events can however be modelled and the time evolution of the system can therefore be described probabilistically. Stochastic reaction-diffusion kinetics is commonly modelled by the reaction-diffusion master equation (RDME) Kuramoto, 1974 Gardiner et al., 1976 Baras and Mansour, 1997 Stochastic reactions-diffusion kinetics can be modelled in many other ways and at different levels of detail. The stochastic simulations performed by MesoRD do however represent the RDME level.

In the RDME framework the total system volume is divided into a large number of subvolumes and the state is described by how many molecules there are of the different species in the different subvolumes. The subvolumes must be small enough to be homogenised by diffusion on the timescale of the chemical reactions. The same time they must be significantly larger than the molecules themselves Elf and Ehrenberg, 2004 so that different molecules can be fully dissociated within a subvolume. This means that we may need hundreds of thousands of subvolumes to correctly discretise a 3D reaction volume.

State changes

The number of molecules in the different subvolumes, i.e. the state of the system, changes when the molecules in any subvolume react or when one molecule diffuse between subvolumes.

Rates and probabilities

In the stochastic framework the reaction and diffusion events are treated probabilistically and the state changes in discrete steps when an event occurs. The probability that a certain reaction, j, occurs in a subvolume with volume Ω during the next time dt is modelled as $dt \cdot \Omega \cdot r_j(x_i)$, where it is indicated that the rate r_j depends on the concentrations of reactants x_i in subvolume i. The $r_j(x_i)$ expressions are given in the SBML model description and they should evaluate in *concentration per time*. For instance the rate law of the reaction $A + B \underset{q}{\overset{k_a}{\rightleftharpoons}} C$ can be defined as $k_a \cdot a_i \cdot b_i$ in the SBML file. Here a_i and b_i are given in *molar* and the second order rate constant of association k_a is given in *per molar per second* $(M^{-1}s^{-1})$. MesoRD interprets the rate as a *probability per time unit* by evaluating the expression $\Omega \cdot r = \Omega \cdot ka_a \cdot (n_{A,i} \cdot \Omega^{-1}) \cdot (n_{B,i} \cdot \Omega^{-1})$, where $n_{A,i}$ and $n_{B,i}$ are number of A and B molecules in the subvolume, i. If this reaction occurs the number of A and B molecules in the subvolume are reduced by one and the number of C molecules is increased by one. Generally the number of molecules will change by the stoichiometry defined for the reaction in the SBML file.

The event that a molecule diffuses to a neighbouring subvolume is treated as a first order reaction with a rate constant of $k_{diff} = D/l^2$, where D is the diffusion constant for the diffusing species and l is the side length of the subvolume. The probability that an A molecule diffuses from one subvolume to one of its neighbours in the next short time period dt is thus $dt \cdot \Omega \cdot k_{diff} \cdot a_i = dt \cdot k_{diff} \cdot n_{A,i}$, where a_i is the concentration of A molecules in the subvolume from which the molecules diffuse. If a diffusion event occurs the number of molecules in decreased by one in the subvolume that lost a molecule and the number is increased by one in its neighbour.

The rates, i.e. the probabilities per time unit, of all different diffusion and reaction events define a stochastic process. All events are assumed to be elementary, why the time to the next event of each kind is exponentially distributed in time with a mean corresponding to the inverse rate. When one event occurs, some of the rates for the next event will change. In the stochastic mode MesoRD samples the stochastic process one event at the time and updates the state and reaction rates accordingly. Since there can be millions of different events to chose from, it is critical to do this *very* efficiently. The algorithm used for sampling the process is the *next subvolume method* (NSM). The algorithm is published as supplementary material to Elf and Ehrenberg, 2004 but can also be downloaded from the MesoRD homepage.

Mean-field time evolution

In the deterministic mode MesoRD will do what we call a mean-field approximation. The approximation is that the state, i.e. number of molecules, change continuously and with the average rate given by the stochastic description at each point in time. To see what this means, consider the events that changes the number of A-molecules in subvolume i during the next time period dt , assuming that the systems has only one spatial dimension. The different reactions occurs with probability $dt \cdot \Omega \cdot r_j(x_i)$ and when reaction j occurs the number of A-molecules changes by $S_{A,i}$. The A-molecules can also diffuse away with probability $dt \cdot \Omega \cdot k_{diff} \cdot a_i = dt \cdot k_{diff} \cdot n_{A,i}$ to each of the two neighbouring subvolumes in one dimension. The number of A-molecules in subvolume i can also increase if A-molecules diffuse from the neighbouring subvolumes. They do this with probability $dt \cdot k_{diff} \cdot n_{A,i-1}$ and $dt \cdot k_{diff} \cdot n_{A,i-1}$, respectively. The average change in the number of A-molecules in subvolume i during dt is therefore

Equation 6-1. Average change

$$dn_{A,i} = \Omega \cdot dt \sum_j S_{A,j} \cdot r_i + 2 \cdot (-1) \cdot dt \cdot k_{diff} \cdot n_{A,i} + (1) \cdot k_{diff} \cdot n_{A,i-1} + (1) \cdot k_{diff} \cdot n_{A,i+1}$$

where the values in parenthesis are the stoichiometries for changes of the number of A-molecules of the different events. Similar expressions simultaneously govern the A-molecules in other subvolumes and other species.

If we divide Equation 6-1 by dt we get a system of ordinary differential equations, for instance for $n_{A,i}$

Equation 6-2. System of ordinary differential equation

$$\frac{dn_{A,i}}{dt} = \Omega \sum_j S_{A,j} \cdot r_i + D \left(\frac{n_{A,i-1} - 2n_{A,i} + n_{A,i+1}}{l^2} \right)$$

where $k_{diff} = D/l^2$. In the deterministic mode MesoRD solve this kind of ODE systems numerically.

If we divide Equation 6-2 by Ω and take the limit $l \to 0$ we get a more familiar expression for the reactions diffusion equation

Equation 6-3. Reaction diffusion equation

$$\frac{\partial a(x,t)}{\partial t} = \sum_j S_{A,j} \cdot r_i + D\frac{\partial^2 a(x,t)}{\partial^2 x}$$

with x as the continuous spatial coordinate.

Chapter 7. Tutorial

This chapter describes how one would go about to put the model from the Section called *An introductory Example* in Chapter 1 into a valid SBML definition that can be used with MesoRD. Note that we will construct a complete model definition as we go along. The SBML file is also included in the distribution as `tutorial.xml`.

Units

Before we begin the actual definition, we must set the stage with a few lines as given in the Section called *General SBML Structure* in Chapter 3. The first line tells the parser that we are dealing with an UTF-8 encoded XML file. The line that begins with `<sbml` will qualify all un-prefixed names using the SBML URI given in the `xmlns` attribute. The additional attributes state that we are dealing with level 2 ,version 1 of SBML. The `model` element type, or tag, marks the beginning of the model.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
<model name="MesoRD Tutorial System">
```

Now we can begin the model definition. To prepare for what will come, we will define a few units which will be useful in later definitions.

```
<listOfUnitDefinitions>

  <unitDefinition id="um">
    <listOfUnits>
      <unit kind="metre" scale="-6"/>
    </listOfUnits>
  </unitDefinition>

  <unitDefinition id="pMps">
    <listOfUnits>
      <unit kind="mole" exponent="-1"/>
      <unit kind="litre" exponent="+1"/>
      <unit kind="second" exponent="-1"/>
    </listOfUnits>
  </unitDefinition>

  <unitDefinition id="cm2ps">
    <listOfUnits >
      <unit kind="metre" exponent="2" scale="-4"/>
      <unit kind="second" exponent="-1"/>
    </listOfUnits>
  </unitDefinition>

  <unitDefinition id="ps">
    <listOfUnits>
      <unit kind="second" exponent="-1"/>
```

```
    </listOfUnits>
  </unitDefinition>
</listOfUnitDefinitions>
```

The above lines define four units.

- The unit `pMps` defines the unit "per molar per second". This unit will be used in the definition of the association rate constant.
- The unit `cm2ps` defines the unit "centimetre squared per second". This unit will be used in the definition of the diffusion constants.
- The unit `ps` defines the unit "per second". We will use this unit to define the rate constants for the reactions in the system.
- Finally, the `um` unit is defined equal to one micrometre. This will be useful in the specification of the geometry of the system.

Geometry

Let us now define the geometry of the reaction volume in the Section called *An introductory Example* in Chapter 1. Before we define the compartments, we note that the geometry as defined in the example has three functionally distinct regions "1", "2" and "3". These regions will map exactly to the three compartments we are about to define.

We will use the `sphere` primitive to model the ellipsoid and slightly rotated compartment two. To achieve this we make a sphere with a radius of 2 micrometres. We make it ellipsoid by scaling the sphere two times in the y direction. Next we rotate the ellipsoid slightly around the z-axis. To described this for MesoRD, the `listOfCompartments` may begin like shown below. Note that we make use of the previously defined `um` unit.

```
<listOfCompartments>
  <compartment id="CompartmentTwo"
                units="litre">
    <annotation xmlns:MesoRD="http://www.icm.uu.se">
      <MesoRD:rotation MesoRD:x="0"
                       MesoRD:y="0"
                       MesoRD:z="1"
                       MesoRD:angle="20"
                       MesoRD:units="um">
        <MesoRD:scale MesoRD:x="1"
                      MesoRD:y="2"
                      MesoRD:z="1">
          <MesoRD:sphere MesoRD:radius="2"
                         MesoRD:units="um"/>
        </MesoRD:scale>
      </MesoRD:rotation >
```

```
  </annotation>
</compartment>
```

Next, we will define the cytosol-like `CompartmentOne`. It is built as union of a cylinder and three spheres. Two spheres forms the caps of the cylinder and an additional sphere forms a budding structure. This compartment must exclude `CompartmentTwo`, defined above. To do this we subtract the `CompartmentTwo` from the union of the cylinder and the three spheres.

```
<compartment id="CompartmentOne"
               units="litre">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:difference>
      <MesoRD:union>
        <MesoRD:cylinder MesoRD:radius="4.5"
                         MesoRD:height="8"
                         MesoRD:units="um"/>
        <MesoRD:translation MesoRD:x="0.00"
                            MesoRD:y="-4"
                            MesoRD:z="0"
                            MesoRD:units="um">
          <MesoRD:sphere MesoRD:radius="4.5"
                         MesoRD:units="um"/>
        </MesoRD:translation>
        <MesoRD:translation MesoRD:x="0.00"
                            MesoRD:y="4"
                            MesoRD:z="0"
                            MesoRD:units="um">
          <MesoRD:sphere MesoRD:radius="4.5"
                         MesoRD:units="um"/>
        </MesoRD:translation>
        <MesoRD:translation MesoRD:x="0"
                            MesoRD:y="7"
                            MesoRD:z="3.5"
                            MesoRD:units="um">
          <MesoRD:sphere MesoRD:radius="4.5"
                        MesoRD:units="um"/>
        </MesoRD:translation>
      </MesoRD:union>
      <MesoRD:compartment MesoRD:id="CompartmentTwo" />
    </MesoRD:difference>
  </annotation>
</compartment>
```

To form the membrane like compartment we define a geometry that is slightly larger than `CompartmentOne` and subtract both compartments defined above.

```
<compartment id="CompartmentThree"
               units="litre">
```

```
    <annotation xmlns:MesoRD="http://www.icm.uu.se">
      <MesoRD:difference>
        <MesoRD:union>
          <MesoRD:translation MesoRD:x="-0.00"
                              MesoRD:y="-0.00"
                              MesoRD:z="0"
                              MesoRD:units="um">
            <MesoRD:cylinder MesoRD:radius="5.0"
                             MesoRD:height="8.0"
                             MesoRD:units="um"/>
          </MesoRD:translation>
          <MesoRD:translation MesoRD:x="0.0"
                              MesoRD:y="-4"
                              MesoRD:z="0"
                              MesoRD:units="um">
            <MesoRD:sphere MesoRD:radius="5"
                           MesoRD:units="um"/>
          </MesoRD:translation>
          <MesoRD:translation MesoRD:x="0.0"
                              MesoRD:y="4"
                              MesoRD:z="0.0"
                              MesoRD:units="um">
            <MesoRD:sphere MesoRD:radius="5"
                           MesoRD:units="um"/>
          </MesoRD:translation>
          <MesoRD:translation MesoRD:x="0"
                              MesoRD:y="7"
                              MesoRD:z="3.5"
                              MesoRD:units="um">
            <MesoRD:sphere MesoRD:radius="5"
                         MesoRD:units="um"/>
          </MesoRD:translation>
        </MesoRD:union>
        <MesoRD:union>
          <MesoRD:compartment MesoRD:id="CompartmentOne" />
          <MesoRD:compartment MesoRD:id="CompartmentTwo" />
        </MesoRD:union>
      </MesoRD:difference>
    </annotation>
  </compartment>
</listOfCompartments>
```

The last line in the code above closes the `listOfCompartments` we started earlier.

Species

Each species must be defined in every compartment it may be found. However, we want to make sure we use the same name in all definitions, because it is really the same species. Note, however, that SBML requires the species id to be unique.

```
<listOfSpecies>

  <species id="A1"
           name="A"
           boundaryCondition="false"
           compartment="CompartmentOne"
           hasOnlySubstanceUnits="true"
           initialAmount="0"
           substanceUnits="item">
    <annotation xmlns:MesoRD="http://www.icm.uu.se">
      <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                        MesoRD:rate="0"
                        MesoRD:units="cm2ps"/>
      <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                        MesoRD:rate="0"
                        MesoRD:units="cm2ps"/>
      <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                        MesoRD:rate="0"
                        MesoRD:units="cm2ps"/>
    </annotation>
  </species>

  <species id="A2"
           name="A"
           boundaryCondition="false"
           compartment="CompartmentTwo"
           hasOnlySubstanceUnits="true"
           initialAmount="0"
           substanceUnits="item">
    <annotation xmlns:MesoRD="http://www.icm.uu.se">
      <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                        MesoRD:rate="0"
                        MesoRD:units="cm2ps"/>
      <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                        MesoRD:rate="0"
                        MesoRD:units="cm2ps"/>
      <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                        MesoRD:rate="0"
                        MesoRD:units="cm2ps"/>
    </annotation>
  </species>

  <species id="A3"
           name="A"
           boundaryCondition="false"
           compartment="CompartmentThree"
           hasOnlySubstanceUnits="true"
           initialAmount="1"
           substanceUnits="item">
    <annotation xmlns:MesoRD="http://www.icm.uu.se">
      <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                        MesoRD:rate="0.01e-8"
                        MesoRD:units="cm2ps"/>
```

```
    <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
  </annotation>
</species>

<species id="B1"
         name="B"
         boundaryCondition="false"
         compartment="CompartmentOne"
         hasOnlySubstanceUnits="true"
         initialAmount="0"
         substanceUnits="item">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                      MesoRD:rate="2.0e-8"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                      MesoRD:rate="2.0e-8"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                      MesoRD:rate="2.0e-8"
                      MesoRD:units="cm2ps"/>
  </annotation>
</species>

<species id="B2"
         name="B"
         boundaryCondition="false"
         compartment="CompartmentTwo"
         hasOnlySubstanceUnits="true"
         initialAmount="0"
         substanceUnits="item">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                      MesoRD:rate="2.0e-8"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                      MesoRD:rate="2.0e-8"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                      MesoRD:rate="2.0e-8"
                      MesoRD:units="cm2ps"/>
  </annotation>
</species>

<species id="B3"
         name="B"
         boundaryCondition="false"
         compartment="CompartmentThree"
```

```
        hasOnlySubstanceUnits="true"
        initialAmount="0"
        substanceUnits="item">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                      MesoRD:rate="2.0e-8"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                      MesoRD:rate="2.0e-8"
                      MesoRD:units="cm2ps"/>
  </annotation>
</species>

<species id="C1"
        name="C"
        boundaryCondition="false"
        compartment="CompartmentOne"
        hasOnlySubstanceUnits="false"
        initialConcentration="0"
        substanceUnits="mole">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
  </annotation>
</species>

<species id="C2"
        name="C"
        boundaryCondition="false"
        compartment="CompartmentTwo"
        hasOnlySubstanceUnits="false"
        initialConcentration="0"
        substanceUnits="mole">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
```

```
  </annotation>
</species>

<species id="C3"
         name="C"
         boundaryCondition="false"
         compartment="CompartmentThree"
         hasOnlySubstanceUnits="false"
         initialConcentration="1e-9"
         substanceUnits="mole">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
  </annotation>
</species>

<species id="D1"
         name="D"
         boundaryCondition="false"
         compartment="CompartmentOne"
         hasOnlySubstanceUnits="true"
         initialAmount="0"
         substanceUnits="item">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
    <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                      MesoRD:rate="0"
                      MesoRD:units="cm2ps"/>
  </annotation>
</species>

<species id="D2"
         name="D"
         boundaryCondition="false"
         compartment="CompartmentTwo"
         hasOnlySubstanceUnits="true"
         initialAmount="0"
         substanceUnits="item">
  <annotation xmlns:MesoRD="http://www.icm.uu.se">
    <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                      MesoRD:rate="0"
```

```
                        MesoRD:units="cm2ps"/>
      <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                        MesoRD:rate="2.0e-8"
                        MesoRD:units="cm2ps"/>
      <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                        MesoRD:rate="0"
                        MesoRD:units="cm2ps"/>
    </annotation>
  </species>

  <species id="D3"
           name="D"
           boundaryCondition="false"
           compartment="CompartmentThree"
           hasOnlySubstanceUnits="true"
           initialAmount="0"
           substanceUnits="item">
    <annotation xmlns:MesoRD="http://www.icm.uu.se">
      <MesoRD:diffusion MesoRD:compartment="CompartmentThree"
                        MesoRD:rate="0"
                        MesoRD:units="cm2ps"/>
      <MesoRD:diffusion MesoRD:compartment="CompartmentTwo"
                        MesoRD:rate="0"
                        MesoRD:units="cm2ps"/>
      <MesoRD:diffusion MesoRD:compartment="CompartmentOne"
                        MesoRD:rate="0"
                        MesoRD:units="cm2ps"/>
    </annotation>
  </species>

</listOfSpecies>
```

Here A and C molecules have non-zero initial amounts. A is initialised with one molecule in `CompartmentThree`. C is initialised with as a concentration in `CompartmentThree`. Neither A or C and diffuse away from `CompartmentThree`, but A makes B molecules that can diffuse into `CompartmentOne` and `CompartmenTwo`.

Parameters

Using the previously defined units we can define the parameter that we need for the reactions as follow.

```
<listOfParameters>
  <parameter id="k" units="ps" value="100.0"/>
  <parameter id="ka" units="pMps" value="1e8"/>
  <parameter id="kd" units="ps" value="0.01"/>
  <parameter id="mu" units="ps" value="0.5"/>
  <parameter id="mu2" units="ps" value="0.005"/>
</listOfParameters>
```

The unit of `k` must be ”per second”, because the annihilation reaction will be defined as a first-order reaction in the concentration of species `A`.

Reactions

There there are five different reactions. In `Reaction1` an A molecule makes a B molecule in `CompartmentThree`. In `Reaction2` D molecules are degraded. In `Reaction3` B molecules are degraded in `CompartmentOne`. In `Reaction4` B molecule dimerize to D molecules `CompartmentTwo`. In `Reaction5` an D molecule fall apart into B molecules.

```
<listOfReactions>

  <reaction id="Reaction1" reversible="false">
    <listOfReactants>
      <speciesReference species="A3"/>
      </listOfReactants>
    <listOfProducts>
      <speciesReference species="A3"/>
      <speciesReference species="B3"/>
    </listOfProducts>
    <kineticLaw>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times/>
          <ci>k</ci>
          <ci>A3</ci>
        </apply>
      </math>
    </kineticLaw>
  </reaction>

  <reaction id="Reaction2" reversible="false">
    <listOfReactants>
      <speciesReference species="D2"/>
    </listOfReactants>
    <kineticLaw>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times/>
          <ci>mu2</ci>
          <ci>D2</ci>
        </apply>
      </math>
    </kineticLaw>
  </reaction>

  <reaction id="Reaction3" reversible="false">
    <listOfReactants>
      <speciesReference species="B1"/>
    </listOfReactants>
    <kineticLaw>
```

```
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times/>
          <ci>mu</ci>
          <ci>B1</ci>
        </apply>
      </math>
    </kineticLaw>
  </reaction>

  <reaction id="Reaction4" reversible="false">
    <listOfReactants>
      <speciesReference species="B2" stoichiometry="2" />
    </listOfReactants>
    <listOfProducts>
      <speciesReference species="D2"/>
    </listOfProducts>
    <kineticLaw>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times/>
          <ci>ka</ci>
          <ci>B2</ci>
          <apply>
            <minus/>
            <ci>B2</ci>
            <ci>one_molecule_molar_spec</ci>
          </apply>
        </apply>
      </math>
    </kineticLaw>
  </reaction>

  <reaction id="Reaction5" reversible="false">
    <listOfReactants>
      <speciesReference species="D2"/>
    </listOfReactants>
    <listOfProducts>
      <speciesReference species="B2" stoichiometry="2" />
    </listOfProducts>
    <kineticLaw>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times/>
          <ci>kd</ci>
          <ci>D2</ci>
        </apply>
      </math>
    </kineticLaw>
  </reaction>
</listOfReactions>
```

Note that we use the id of the species in the definition of the `kineticLaw`. This will define the compartment of the reaction. Note the parameter `one_molecule_molar_spc` in `Reaction 4`. This parameter takes it value at runtime and it is very special. It takes different values in the deterministic and the stochastic modes! In the stochastic mode it takes the value of the concentration of one molecule in a subvolume. In the deterministic mode it is zero, since this is the minimal change in concentration. The parameter was implemented to model the reaction X+X-> with a rate proportional to #X*(#X-1) in the stochastic mode and a rate proportional to #X*#X in the deterministic, using the same SBML code.

Finally, we need to close the `model` and `sbml` definitions we started in the Section called *Units*.

```
</model>
</sbml>
```

Result

Figure 7-1. Snapshot of simulation at 1 second simulation time.

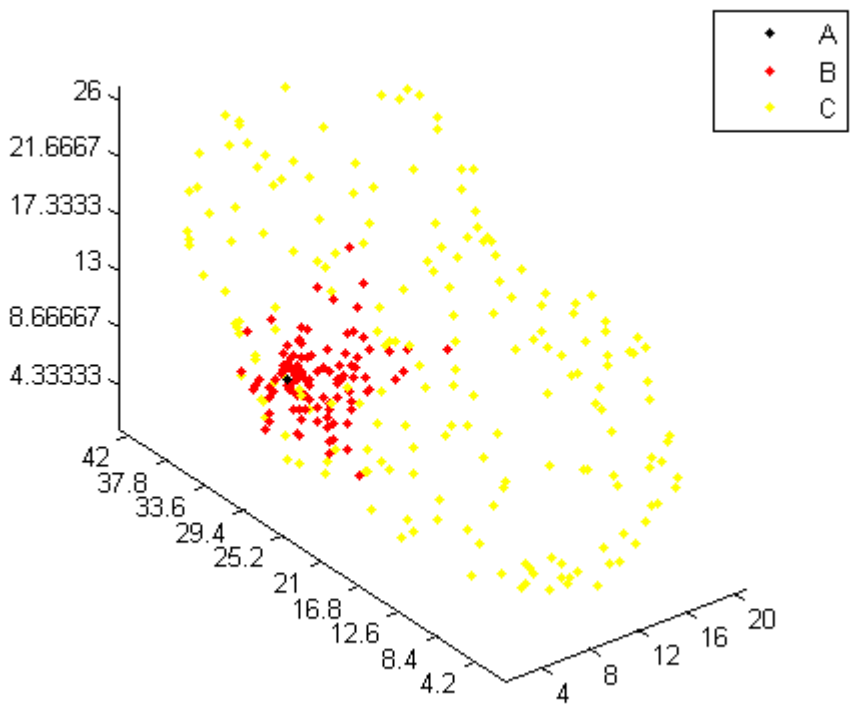
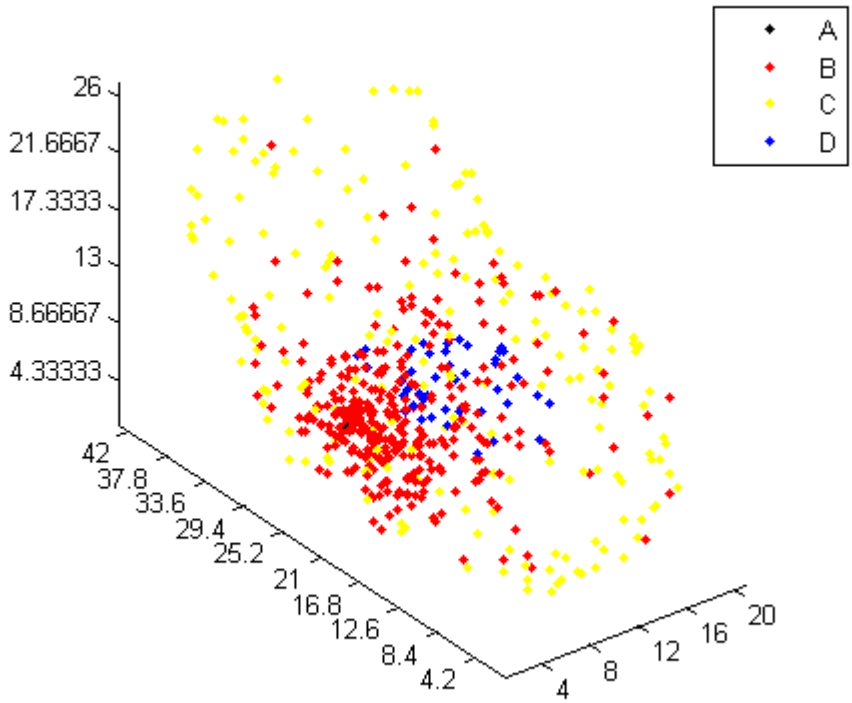


Figure 7-2. Snapshot of simulation at 80 seconds simulation time.



Here it can be seen that the B molecules are created from the immobilised A molecule in CompartmentThree. The B molecules will thereafter diffuse throughout the cell. In the small ComartmentTwo two B molecules can react and create one D molecule as seen in Figure 7-2.

Chapter 8. Hacking

If you intend to work on the MesoRD sources yourself, we would like you to read this section first.

Coding Standards

The developers try to follow the GNU Coding Standards GNU Coding Standards, with as few modifications as possible. The main point, is not to follow the GNU standards in *every conceivable detail*, but to be self-consistent. In particular, attention is paid to the points listed below.

- When dealing with configuration options that are known at compile time, `if (...)` constructs are preferred over conditional compilation. This enables the compiler to do more extensive checking of the possible code paths. For example:

```
if (HAS_FOO)
  ...
else
  ...
```

  is preferred over conditional compilation:

```
#ifdef HAS_FOO
...
#else
...
#endif
```

  A way to achieve the former, when only presented with `define`s is shown below.

```
#ifdef HAS_FOO
const bool hasFoo = true;
#else
const bool hasFoo = false;
#endif
```

  One should be able to trust the compiler enough to optimise away the redundant variable `hasFoo`. Exceptions to the above rule almost exclusively deal with platform specific details. In such cases, a code path will *only* compile on the architecture for which the `define` checks. Attempting to check a code path that can never hope to compile in the first place, is futile.

- Use standard prototype form. In definitions, left-align the name of the function and the open-brace that starts the body. According to the GNU Coding Standards, there are several tools that look for these names and their associated open-braces.

```
int
foo(int x, int y, short z,
    std::string& bar)
{
  ...
}
```

- Because Microsoft Visual C++ lacks the `std::max()` and `std::min()` templates, MesoRD defines its own. These templates are called `MesoRD::max()` and `MesoRD::min()`, respectively. For portability, one should use these MesoRD templates instead of whatever the current tool chain supplies.
- When splitting expressions into multiple lines, split them *before* operators, not after.
- C++ takes type-safety seriously: `foo_t` and `foo_t*` are viewed as distinct types, rather than the second just being a pointer to an object of the first kind. Spacing should be used to emphasise this view, by writing `foo_t* bar`, instead of `foo_t *bar`. In order to avoid confusion, pointers should be declared in separate statements. References, such as `foo_t&` are handled similarly.
- Please put two spaces after the end of a sentence in comments, so that Emacs sentence commands work. Also, please write complete sentences and capitalise the first word.
- When you have an `if` statement nested in another `if` statement, always put braces around the outer `if`.
- Since `NULL` and `0` are really the same thing, one might as well save a macro expansion by using the simpler `0`. In the documentation, the value is sometimes referred to as `null`, because it looks a bit clearer.
- Make use of the `MesoRD` namespace.
- Indentation level is two spaces. If you are using Microsoft Visual C++ select Options... from the Tools menu, pick Tabs and set Tab size and Indent size to 2. If you use Emacs, we can give you our hooks for C++ files.
- We use doxygen to handle source-level documentation. Implementation specific comments go in the `.cpp` file, while the general descriptions end up in the `.hpp` file. Only the `.hpp` files are considered when building the documentation.

Note that in we (the developers) do not always live as we preach. But when we break the rules, it's a bug.

Bibliography

Technical Documentation

[Finney and Hucka, 2003] Andrew Finney and Michael Hucka, Systems Biology Workbench Development Group, June, 2003, *Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions* .

[GNU Coding Standards] Richard Stallman, 1992-2004, *GNU Coding Standards* .

Books

[Angel, 1997] Edward Angel, First edition, 0-201-85571-2, 1997, Addison--Wesley, *Interactive Computer Graphics : A Top-Down Approach with OpenGL™* .

[Burger and Gillies, 1989] Peter Burger and Duncan Gillies, First edition, 0-201-17439-1, 1989, Addison--Wesley Publishing Company, *Interactive Computer Graphics : Functional, Procedural and Device-Level Methods* .

[Foley et al., 1996] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, Second edition, 81-7808-038-9, 1996, Addison--Wesley Longman, *Computer Graphics : Principles and Practice* .

[Gardiner, 1989] Crispin W. Gardiner, 1989, Second edition, 0-540-61634-9, Springer, *Handbook of Stochastic Methods : For Physics, Chemistry and the Natural Sciences* .

[Hartman and Wernecke, 1996] Jed Hartman and Josie Wernecke, First edition, 0-201-47944-3, 1996, Addison--Wesley Publishing Company, *The VRML 2.0 Handbook : Building Moving Worlds on the Web* .

[Physics Handbook] Carl Nordling and Jonny Österman, 1996, 91-44-16575-7, Studentlitteratur, *Physics Handbook for Science and Engineering* .

[Stroustrup, 1997] Bjarne Stroustrup, 1997, Second edition, 0-201-88954-4, Addison Wesley, *The C++ Programming Language : A Top-Down Approach with OpenGL™* .

Papers

[Baras and Malek Mansour, 1997] F. Baras and M. Malek Mansour, *Microscopic simulations of chemical instabilities* , " Advances in Chemical Physics ", 393-474, 1997, Edited by I. Prigogine, Edited by Stuart A. Rice, 0-471-17458-0, 1997, 100.

[Elf et al., 2003] Johan Elf, Andreas Dončić, and Måns Ehrenberg, *Mesoscopic reaction-diffusion in intracellular signaling* , " Proceedings of SPIE ", 114-124, 2003, 5110.

[Elf and Ehrenberg, 2004] Johan Elf and Måns Ehrenberg, *Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases* , " IEE Systems Biology ", 230-236, 2004, 2.

[Hattne et al., 2005] Johan Hattne, David Fange, and Johan Elf, *Stochastic reaction-diffusion simulation with MesoRD* , " Bioinfomatics ", 2923-2924, 2005, 21.

[Baras and Mansour, 1997] F Baras and M.M. Mansour, *Microscopic simulation of chemical instabilities* , " Advances in Chemical Physics ", 393-475, 1997, 100.

[Gardiner et al., 1976] CWF Gardiner, KJ Mc Neil, DF Walls, and IS Matheson, *Correlations in stochastic theories of Chemical Reactions* , " J Stat Phys ", 307-331, 1976, 14.

[Kuramoto, 1974] Y Kuramoto, *Effects of Diffusion on the Fluctuations in Open Systems* , " Prog. Theor. Phys. ", 711-713, 1974, 52.

Appendix A. GNU Free Documentation License

Version 1.2, November 2002

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection

with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The ”Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The ”Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A ”Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not ”Transparent” is called ”Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The ”Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ”Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section ”Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as ”Acknowledgements”, ”Dedications”, ”Endorsements”, or ”History”.) To ”Preserve the Title” of such a section when you modify the Document means that it remains a section ”Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommer-

cially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

GNU FDL Modification Conditions

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant

Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Samle Invariant Sections list

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

Sample Invariant Sections list

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.