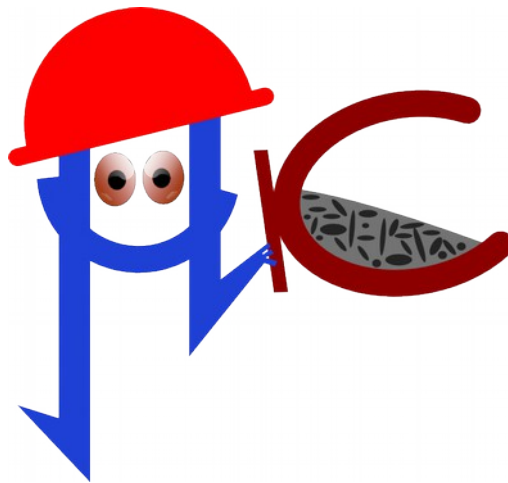


The User Documentation for μ ic,

the plugins included in μ ic, its Graphical user interface
and XML input file



By: Shashank Bishnoi
Version: 24 May 2012

Find the latest version of this document on:
<https://web.iitd.ac.in/~bishnoi/mic.html>

Table of Contents

Table of Contents.....	ii
Conditions of use.....	ix
Chapter 1: Quick reference to the structure of mic.....	1
1.1 Package org.micthemodel.asciiIO.....	1
1.2 Package org.micthemodel.comparators.....	1
1.3 Package org.micthemodel.elements.....	1
1.4 Package org.micthemodel.factory.....	2
1.5 Package org.micthemodel.helpers.....	2
1.6 Package org.micthemodel.occupancy.....	2
1.7 Package org.micthemodel.particles.....	2
1.8 Package org.micthemodel.pixels.....	2
1.9 Package org.micthemodel.plugins.....	2
1.9.1 Package org.micthemodel.plugins.densityVariation.....	2
1.9.2 Package org.micthemodel.plugins.grainProportionInitialiser.....	3
1.9.3 Package org.micthemodel.plugins.kinetics.....	3
1.9.4 Package org.micthemodel.plugins.materialDistributionProfile.....	3
1.9.5 Package org.micthemodel.plugins.nucleiGenerator.....	3
1.9.6 Package org.micthemodel.plugins.packer.....	3
1.9.7 Package org.micthemodel.plugins.particleDistributionProfile.....	3
1.9.8 Package org.micthemodel.plugins.particleIterator.....	3
1.9.9 Package org.micthemodel.plugins.postProcessor.....	4
1.9.10 Package org.micthemodel.plugins.reactionTrigger.....	4
1.9.11 Package org.micthemodel.plugins.recalculator.....	4
1.10 Package org.micthemodel.porosity.....	4

1.11 Package org.micthemodel.shapes3D.....	4
1.12 Package org.micthemodel.start.....	4
1.13 Package org.micthemodel.structures.....	4
1.14 To compile µic.....	5
1.14.1 Using Netbeans and source files.....	5
1.14.2 Using Netbeans and svn repository.....	5
Chapter 2: Defining user customisable elements.....	6
2.1 Defining the reactor (Reactor).....	6
2.1.1 Size (double).....	6
2.1.2 Project folder (String).....	6
2.1.3 Resource folders (String) @deprecated.....	7
2.1.4 Copy from folders (String).....	7
2.1.5 Time step file (String).....	7
2.1.6 Relative paths and root folder.....	7
2.1.7 Images.....	8
2.1.8 Grid overlay.....	8
2.1.9 Simulating flocculation.....	9
2.1.10 Speed Mode.....	10
2.1.11 Intense Mode.....	10
2.1.12 Number of sampling points on surface of particles.....	11
2.1.13 Locating neighbourhood spheres for distribution.....	11
2.1.14 Output files.....	11
2.1.15 Example setup of a reactor in the XML file.....	12
2.2 Defining materials (Material).....	13
2.2.1 Name (String).....	13
2.2.2 Id (int).....	13
2.2.3 Density (double).....	13
2.2.4 Initial fraction (double).....	14
2.2.5 Inner (boolean) and Variant (Material).....	14
2.2.6 Discrete (boolean) and HostMaterial (Material).....	14
2.2.7 Universal (boolean).....	15
2.2.8 User defined parameter (double).....	15
2.2.9 Material class (String).....	15
2.2.10 Replace (Material).....	15
2.2.11 Color (Color/String).....	15
2.2.12 Example setup of a material in the XML file.....	16

2.3 Defining particle models (ModelGrain).....	16
2.3.1 Name (String).....	16
2.3.2 Layers (Material).....	17
2.3.3 Initial particle size distribution (String).....	17
2.3.4 Example setup of a ModelGrain in the XML file.....	18
2.4 Defining Reactions.....	19
2.4.1 Name (String).....	19
2.4.2 Order of reactions.....	19
2.4.3 Reactants and Products.....	19
2.4.4 Discrete (boolean).....	19
2.4.5 Successor (Reaction).....	20
2.4.6 Example setup of a reaction in the XML file.....	20
2.5 Run Options.....	21
2.5.1 Clean start with new packing.....	21
2.5.2 Start from stored packing or earlier simulation.....	21
2.5.3 Recreate images with a new resolution.....	21
2.6 Other run options only in the XML file.....	22
 Chapter 3: Plugins in <code>µic</code>	24
3.1 Plugins.....	24
3.1.1 Type <code>densityVariation</code>	24
3.1.2 Type <code>grainProportionInitialiser</code>	25
3.1.3 Type <code>kinetics</code>	25
3.1.4 Type <code>materialDistributionProfile</code>	26
3.1.5 Type <code>nucleiGenerator</code>	26
3.1.6 Type <code>packer</code>	27
3.1.7 Type <code>particleDistributionProfile</code>	28
3.1.8 Type <code>particleIterator</code>	28
3.1.9 Type <code>postProcessor</code>	29
3.1.10 Type <code>reactionTrigger</code>	29
3.1.11 Type <code>recalculator</code>	30
3.2 To create, compile and use an external plugin.....	30
3.3 Loading a plugin using the XML file.....	31
3.3.1 <code>classPath</code> and <code>className</code>	32
3.3.2 Loading parameters using the XML file.....	32
3.4 Loading parameters using the GUI.....	35
3.4.1 Choosing the class path, name and type.....	35

3.4.2 Declaring parameters using the GUI.....	36
Chapter 4: A summary of the algorithm of μic	39
4.1 Control flow.....	39
4.2 Initialising reactor in μic	40
4.3 Calculating evolution.....	40
4.4 Changing layers and materials in particles.....	41
4.5 Calculating free surface of a particle.....	42
4.6 Function of overlaid occupancy grid.....	42
4.7 Generics in μic	42
Chapter 5: Running μic	44
5.1 Executing user and GUI generated XML input files.....	44
5.1.1 Running the graphical user interface (GUI).....	44
5.1.2 Executing from the GUI	44
5.1.3 Executing using command line.....	45
5.2 Using the user menus in μic	46
5.2.1 Simulation of the reactions.....	46
5.2.2 Pore-sizes and connectivity.....	46
5.2.3 Calculating pore-sizes on refined mesh.....	48
Chapter 6: Output files.....	50
6.1 Details file.....	50
6.2 Grains files.....	50
6.3 Images.....	50
6.4 Pixels.....	51
6.5 Custom output files.....	51
6.6 Note about CSV files.....	51
Chapter 7: Packaged plugins in μic	52
7.1 Plugins.....	52
7.2 Type densityVariation.....	52
7.2.1 DefaultDensityVariation.....	53
7.2.2 FixedVolumeDensityVariation.....	53
7.3 Type grainProportionInitialiser.....	53

7.3.1 DefaultGrainProportionInitialiser.....	54
7.3.2 LimitedRandomGrainProportionInitialiser.....	54
7.3.3 RandomGrainProportionInitialiser.....	54
7.4 Type kinetics.....	54
7.4.1 AvramiKinetics.....	55
7.4.2 AvramiKineticsNoReduction.....	55
7.4.3 DensifyingNGKinetics.....	55
7.4.4 DiffusionKinetics.....	58
7.4.5 DiffusionProportionateKinetics.....	58
7.4.6 ExternallyDefinedKinetics.....	58
7.4.7 FixedRadiusKinetics.....	59
7.4.8 NGKinetics.....	59
7.4.9 NucleiGrowthKinetics.....	60
7.4.10 OuterNucleiGrowthKinetics.....	60
7.4.11 PhaseBoundaryKinetics.....	61
7.4.12 ReactAllKinetics.....	61
7.4.13 RoughDissolvingSurfaceKinetics.....	61
7.4.14 SphericalDiffusionProportionateKinetics.....	62
7.4.15 SurfaceNucleiGrowthKinetics.....	62
7.4.16 VolumeProportionateKinetics.....	63
7.5 Type materialDistributionProfile.....	63
7.5.1 InnerLayerDistributionProfile.....	64
7.5.2 RandomDistributionProfile.....	64
7.5.3 SurfaceDistributionProfile.....	64
7.5.4 VolumeProportionateDistribution.....	65
7.6 Type nucleiGenerator.....	65
7.6.1 DendriticNucleiLocationGenerator.....	65
7.6.2 DependentExponentialNucleiGenerator.....	66
7.6.3 DependentLinearNucleiGenerator.....	66
7.6.4 ExponentialNucleiGenerator.....	66
7.6.5 FixedNuclei.....	66
7.6.6 LinearNucleiGenerator.....	66
7.6.7 SingleDendriticNucleiLocationGenerator.....	66
7.6.8 SurfaceNucleator.....	67
7.6.9 SurfaceNucleiLocationGenerator.....	67
7.7 Type packer.....	67
7.7.1 FlocPacker.....	68
7.7.2 RandomPacker.....	69

7.8 Type particleDistributionProfile.....	69
7.8.1 DefaultParticleDistributionProfile.....	70
7.8.2 NoReductionParticleDistributionProfile.....	70
7.9 Type particleIterator.....	70
7.9.1 SingleParticleIterator.....	71
7.10 Type postProcessor.....	71
7.10.1 ClassWiseMaterialDetails.....	72
7.10.2 LayerThicknessPlotter.....	72
7.11 Type reactionTrigger.....	72
7.11.1 ActiveTrigger.....	73
7.11.2 AmountControlledTrigger.....	73
7.11.3 InactiveTrigger.....	73
7.11.4 Multiple AmountControlledTrigger.....	73
7.11.5 PeakOffTrigger.....	73
7.11.6 SurfaceRebalancingTrigger.....	74
7.11.7 TimeTrigger.....	74
7.12 Type recalculator.....	74
7.12.1 DefaultRecalculator.....	74
7.12.2 SurfaceRebalancer.....	74
 Chapter 8: Known issues and workarounds.....	 76
8.1 Known issues and workarounds with μic	76
8.1.1 Reduction in outer radii of particles.....	76
8.1.2 Compatibility issues.....	76
8.1.3 Boundary conditions.....	77
8.2 Known issues and workarounds with GUI for μic	77
8.2.1 Deletion of plugins of ParticleIterator type.....	77
8.2.2 Editing defined plugins.....	77
 Chapter 9: Example tutorials.....	 78
9.1 Nucleation and growth of single material.....	78
9.1.1 Defining the reactor.....	79
9.1.2 Defining the material.....	80
9.1.3 Defining the particle model.....	80
9.1.4 Defining the reaction.....	81
9.1.5 Defining plugins.....	81
9.1.6 Creating the input file.....	83

9.2 Hydration of alite.....	83
9.2.1 Defining the reactor.....	84
9.2.2 Defining the materials.....	85
9.2.3 Defining the particle models.....	87
9.2.4 Defining the reaction.....	88
9.2.5 Defining plugins.....	88
9.2.6 Creating the input file.....	91
9.3 Adding inert filler particles to alite hydration.....	91
9.3.1 Defining the reactor.....	92
9.3.2 Defining the materials.....	92
9.3.3 Defining the particle models.....	93
9.3.4 Defining reactions.....	93
9.3.5 Defining plugins.....	94
9.3.6 Creating the input file.....	95
9.4 Adding belite to hydration with alite and filler particles.....	95
9.4.1 Defining the reactor.....	96
9.4.2 Defining the materials.....	96
9.4.3 Defining the particles.....	96
9.4.4 Defining the reactions.....	96
9.4.5 Defining plugins.....	97
9.4.6 Creating the input file.....	98
 Chapter 10: MicWizard.....	 99
10.1 Introduction.....	99
10.2 Input files.....	99
10.2.1 Library file.....	99
10.2.2 Powder file.....	100
10.2.3 Mix design file.....	101
10.3 Running a mix design XML file.....	102
10.4 Running the MicWizard graphical user interface.....	102
 References.....	 103

Conditions of use

The use of this documentation and the programmes described in this documentation, including but not limited to `µic` and the graphical user interface (GUI) of `µic` is subject to conditions. This documentation is has been developed by Shashank Bishnoi, henceforth referred to as the author. The author reserves all rights to the name `µic` and logo of the software. In all works including, but not limited to, publications, articles, reports, web-pages, presentations and communications, scientific, corporate, or otherwise, please cite `µic` AND its author.

All use of this documentation, `µic` and the GUI of `µic` should be cited with the article provided in the CITATION.TXT file provided with this distribution. If you have not received this file, all use has to be cited with the following article: Bishnoi S. and Scrivener K.L., *µic: A new platform for modelling the hydration of cements*, Cement and Concrete Research, Vol. 39, 2009, pp.266-274. This notice should be kept intact, without additions, deletions, or insertions, in all modifications, uses or distributions of this file.

The above conditions apply in addition to the terms of the GNU General Public Licence below. See the COPYING.TXT file that comes with this distribution. This documentation to `µic` and `µic` itself is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This documentation, `µic` and the GUI for `µic` are distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with `µic`. If not, see <<http://www.gnu.org/licenses/>>.

Chapter 1: Quick reference to the structure of μ ic

μ ic is a microstructural modelling framework. It has been designed to give maximum flexibility to the user. So setting up problems in μ ic may be tedious at first. This document presents a quick overview of the structure of μ ic, the procedure to compile μ ic itself and external plugins, the user input required and the algorithm of μ ic.

μ ic is written in Java and is purely object-oriented. All packages of μ ic are contained inside the `org.micthemodel` package. μ ic contains 13 top level packages and 9 packages inside the plugin package. A summary of the packages is listed below.

1.1 Package `org.micthemodel.asciIO`

This package contains files to help with reading and writing of ASCII text files, including XML input files.

1.2 Package `org.micthemodel.comparators`

This package contains comparators to sort collections or arrays.

1.3 Package `org.micthemodel.elements`

Contains customisable classes for the main elements of the simulations, e.g. materials, reactions, reactor, particle models, etc.

1.4 Package org.micthemodel.factory

Contains classes with static (global) fields and methods. The Globals classes contains system fields. The Images class contains methods to write image files.

1.5 Package org.micthemodel.helpers

Contains classes that are not main elements but are used as important field members and abstract classes for some important elements.

1.6 Package org.micthemodel.occupancy

Contains classes that contain the implementation for the superimposed grid. The grid improves the speed of the calculation of overlaps.

1.7 Package org.micthemodel.particles

Contains specialised and generic implementation of spherical particles. The programme automatically chooses the correct implementation for all applications.

1.8 Package org.micthemodel.pixels

Contains implementations for classes that divide the computational volume into discrete pixels to save images or other purposes.

1.9 Package org.micthemodel.plugins

Contains the definition of the MicPlugin interface, which has to be implemented to create a valid plugin. The inner packages contain the abstract, in some cases default implementations and/or other specialised implementation of the plugins. The details of plugins are discussed later in this document.

1.9.1 Package org.micthemodel.plugins.densityVariation

Contains the implementation for plugins that control the variation of density of materials.

1.9.2 Package `org.micthemodel.plugins.grainProportionInitialiser`

This package contains the plugins to distribute the space in the particles to the different materials that may be present in the particle.

1.9.3 Package `org.micthemodel.plugins.kinetics`

The package contains plugins that control the rate of reactions.

1.9.4 Package `org.micthemodel.plugins.materialDistributionProfile`

This package contains plugins that control the distribution of newly produced material to different particles.

1.9.5 Package `org.micthemodel.plugins.nucleiGenerator`

This package contains plugins that controls the number and location of new particles.

1.9.6 Package `org.micthemodel.plugins.packer`

This package contains plugins that control the packing of the initial particles in the system.

1.9.7 Package `org.micthemodel.plugins.particleDistributionProfile`

This package contains plugins that distribute the new space available in a grain to different materials.

1.9.8 Package `org.micthemodel.plugins.particleIterator`

Contains the class `ParticleChooser` and the abstract main class `ParticleIterator`. `ParticleIterator` plugins can be used to control the set of particles that reactions process together.

1.9.9 Package org.micthemodel.plugins.postProcessor

Contains the abstract class PostProcessor, which can be extended to make plugins that are executed before the start of the simulations and at the end of each step of the simulation.

1.9.10 Package org.micthemodel.plugins.reactionTrigger

Contains plugins that can be used to activate or deactivate reactions.

1.9.11 Package org.micthemodel.plugins.recalculator

Contains plugins that can be used to recalculate various parameters in the simulation.

1.10 Package org.micthemodel.porosity

This package contains a class that is used to do approximate calculation of the pore-sizes in the computational volume. This calculation is done on output files created by the simulation. These calculations are only possible if the project folder contains the porein folder with a porein* file for the required step.

1.11 Package org.micthemodel.shapes3D

This package contains implementations of three-dimensional objects used to generate the microstructure, including spheres and the periodic computational volume.

1.12 Package org.micthemodel.start

The package start contains the Main class which is the executable starting point for `mic`.

1.13 Package org.micthemodel.structures

This package contains implementations of collections for different classes which may be ordered in specific ways.

1.14 To compile μic

1.14.1 Using Netbeans and source files

To compile all sources of μic , create a new project folder in Netbeans called MicTheModel and copy all source files in the src folder of the project folder. Now go to project properties and add the jar files FileIO.jar, jlfr-1_0.jar and swing-layout-1.0.3.jar to the libraries. You should now be able to compile μic using the compile tool of Netbeans.

1.14.2 Using Netbeans and svn repository

Use this method if you would like to obtain the latest builds of μic . Please be aware that since the svn repository usually contains the latest code, it may contain bugs. Stable version of the code may be found on the webpage. Make sure you have collaboration tools installed in Netbeans. Then use the server <https://micepfl.svn.sourceforge.net/svnroot/micepfl> with anonymous user to check out the latest version of μic .

Chapter 2: Defining user customisable elements

This chapter describes how different elements can be defined using the GUI and in an input XML file. All the parameters that can be defined for each element type have been described here and the last sub-section for each element type shows an example block of XML input with all the parameters.

2.1 Defining the reactor (Reactor)

The computational volume in μm is called the reactor. The reactor is a cubic periodic volume with the following parameters. The XML tag to define a reactor is `<reactor>`.

2.1.1 Size (double)

This parameter defines the side of the cubic periodic volume. The XML tag for this property is `<size>`.

2.1.2 Project folder (String)

This parameter gives the path of the folder in which the output from the programmes are stored. This folder can also define the root folder from where the output of a previous run can be read. The path can be absolute, or relative to the run folder of the programme. The XML tag for this property is `<fileFolder>`.

2.1.3 Resource folders (String) @deprecated

This parameter contains a list of paths to folders where the programme should look for input files. The input files includes time step files for the reactor (see section 2.1.5 on page 7) and the particle size distribution files for particle models (see section 2.3.3 on page 17). The XML tag for this property is `<resourceFolder>`.

Care should be taken that if two or more resource folders contain files with the same name any of those files may be selected during a simulation. For this reason, although this function still works if provided in the XML file, it has been removed from the GUI. The complete path, absolute or relative, to all input files has to be now provided.

2.1.4 Copy from folders (String)

This parameter contains the path of the folder from where the earlier step files have to be copied if the simulation is being restarted from a point in an earlier run (see section 2.5.2 on page 21). If this path is not defined, the programme will assume that the files are present in the current file folder. (This option can be found in the packing and interactions tab of the GUI.) The XML tag for this property is `<copyFromFolder>`.

2.1.5 Time step file (String)

This parameter gives the name of the file which contains a list of the times at which the simulation is done. The time steps in the simulation are the differences between the listed time points. The file should be in ASCII format listing the times as doubles. The XML tag for this property is `<timeStepFile>`.

Time steps can also be added directly using several sets of minimum and maximum times (both inclusive), with steps between them. The XML tags for this feature are `<minTime>`, `<maxTime>` and `<step>`, to be written in consecutive lines.

2.1.6 Relative paths and root folder

In the default mode, the GUI stores the absolute values of all paths in the input XML file produced. If the “use relative paths” option is selected, all paths

will be stored relative to the root folder defined in the text field below. XML input files generated this way will be more portable from one computer to another, if all computers have the same directory structure inside the `µic` root folder. Please note that the paths to external jar files is also made relative. Apart from changing the appearance of the file paths in the XML file, this feature can also be defined in the tag `<useRelativePaths>` of the XML file to ensure that the setup file saved in the data folder also follows the defined rule.

2.1.7 Images

The programme provides supports saving 2D slices of the 3D computational volumes at the middle plane along all three directions and an image with depth perception (deep images). The vector information in the computational volume is discretised using the given voxel (or pixel) size. The XML tag for defining the size of the pixels is `<pixelSize>`. These voxels are then used to generate coloured images. The colours for the materials can be defined (see section 2.2.11 on page 15). The deep images are created using an algorithm suggested by Cyrille Dunant.

The deep images may or may not be saved, as defined by the user. The depth of the deep images may be defined as the number of pixels which should be included along the depth. The normal images are always stored by default. The XML tag for deep images is `<saveDeepImages>` (see section 2.1.15).

The background colour of the images is black by default and can be changed by the user. This colour is used when no material is present at a voxel. The XML tag for this property is `<backgroundColor>`.

The programme contains an option to recreate the images at a new or the same resolution using the information about the grains stored earlier. The XML tag for this property is `<redoPixels>` in the input file and recreate images in the Images tab of the reactor in the GUI. If this parameter is set to true the simulation is not repeated and only the information from an earlier run is read.

2.1.8 Grid overlay

The reactor overlays a regular cubic grid over the entire three-dimensional volume to increase the speed of the calculations of overlaps. By default, the reactor sets the size of each of the grid element as the diameter of the median initial particle in the system. The user can, however, define the maximum and the

minimum number of divisions to make in each dimension. This grid has a significant memory overhead, but around 100 divisions should be possible on most computers. 100 is also the default maximum number of divisions and 1 is the default minimum. The XML tags for these properties are `<maxOccupancyPixels>` and `<minOccupancyPixels>`.

2.1.9 Simulating flocculation

The programme contains an option to simulate flocculation on the initial packing of the particles. Apart from the early versions of `µic`, packing has been implemented as a plugin and a plugin called `FlocPacking` has been created to provide the functionality described here. However, to keep backwards compatibility with input files created earlier, flocculation parameters can still be defined as described below. For newer input files, it is recommended to use the input method used for other plugin types.

In the flocculation algorithm, the closest particle to each particle is located. The user can define the maximum distance to look for this particle to increase the speed of the calculation. Once the closest particle is found, the first particle is moved closer to the closest particle by a factor defined by the user. This process of bringing particles closer is repeated the number of times defined by the user. In the last step the particles are rotated along the surface of their closest particle thousand times or until a collision with a third particle occurs. This process gives a denser packed structure. This option can be used in two ways discussed in the following subsections. This algorithm is only meant to demonstrate that how it is possible to move particles during packing and is not meant to emulate real processes. In the programme, this process can be found in the method `flocculate(GrainList<E> inputSizeList, int steps, double factor, double range)` in the class `shapes3D.PeriodicVolume`.

2.1.9.1 High density packing using progressive flocculation

When this option (called `duringDistribution` in input XML file) is used, the programme tries to obtain a higher density packing of the particles by bringing the ones already placed in the volume closer together. The programme uses a random parking algorithm, which places particles at random locations in reducing order of their size, in the volume. For each particle a new random location in the computational volume is generated. The position is kept if the particle does not

overlap with any other particle, otherwise a new random location is tried. If progressive flocculation is used, the user can define the maximum number of times new random locations will be generated for a particle before the particles already in packed are flocculated.

2.1.9.2 Flocculating final packings

When the option of progressive flocculation is not used, the final packing obtained is flocculated using the process described above. It must be noted that this process does not actually produce flocculation the way it occurs in real systems, as this process does not attempt to reduce the number of flocks in a system to a minimum.

2.1.10 Speed Mode

WARNING: Use with care after you have carefully read and understood this function.

If this option is selected (called `checkNoArea` in the XML input file), the free area of the particles is not calculated, possibly saving significant time in the calculations. This option should not be used in cases where overlaps between particles can affect results and the actual volumes of materials are required. As a consequence, this option is not suitable for most cases. It should be only used if the calculation of free surface area is being explicitly called by the user at some other point in the simulation. Any overlaps between particles in this mode will not be calculated so the images and voxels produced will not be correct. This is because although the total amounts of all materials in the system and in each particle will be correct, the radii of the particles will not be adjusted for the overlaps.

2.1.11 Intense Mode

This option (called `checkIndividualArea` in the XML input file) can be used to enable the calculation of the free surface area of the all the particles each time their size is changed. This option can be used when the calculation of free surface at each step of the simulation is not sufficient and the errors are observed to be large. One of the ways of calculating the error is by comparing the real volume and the pixel volume using a high-resolution mesh. This error should be

reduced when this option is enabled. A small error is always obtained for all conditions.

2.1.12 Number of sampling points on surface of particles

This option (called `sphereSamplingPoints` in the XML input file) can also be used to improve the accuracy of the calculation of free surfaces. The free surface of the particles is calculated by finding the free or covered status of a fixed number of points distributed in a randomly homogeneous way on the surface of all particles. To maintain the vector nature of the programme, the number of points is the same on all particles, independent of their size. By default the programme uses a hard-wired set of 92 points. The selection of these default is based on the method described by Tegmark¹. If any other number of points is chosen, the distribution of the points is random². Although increasing the number of should theoretically increase the accuracy of the calculations, since the distribution of the default 92 points is statistically uniform, this behaviour is not guaranteed. The time required for calculations depends linearly on the number of points used for each sphere. The memory overhead for additional points is small.

2.1.13 Locating neighbourhood spheres for distribution

This option (called `findCoveringSphere` in XML input file) enables the location of an overlapping particle of the same type for all particles. This particle can be used to deposit materials formed from the first particle if the surface of the first particle is completely covered. The covering particle found always has the particle model same as the first particle, and should have free surface, so finding a matching particle is not always guaranteed. In case such a particle is not found, the materials produced are passed to the next particle of that type processed.

2.1.14 Output files

Three main types of files, apart from the image files, can be stored using the programme. Information from all particles can be stored in binary files. This information can be reread for regeneration of images, or restarting using the same particle packing. The files are stored in the “grains” folder inside the project folder.

In another format, just the outer radii and positions of all particles can be stored in ASCII. This format can be useful as an input for the analysis of the

porosity. The approximate porosity calculation modules of this programme also uses these files. The files are stored in the “porein” folder inside the project folder.

The main information about all the particles can also be dumped in, usually large, ASCII files for manual reading later. These files are stored in the “hydout” folder inside the project folder.

2.1.15 Example setup of a reactor in the XML file

The following block of XML shows the complete definition for a reactor. The line numbers are not a part of the XML format.

```
1  <reactor name="myReactor">
2    <size>100.0</size>
3    <pixelSize>0.25</pixelSize>
4    <fileFolder>Data\myReactor\</fileFolder>
5    <resourceFolder>Data\resources\</resourceFolder>
6    <copyFromFolder>Data\myOlderReactor\</copyFromFolder>
7    <timeStepFile>hydsteps40.txt</timeStepFile>
8    <backgroudColor>black</backgroudColor>
9    <maxOccupancyPixels>100</maxOccupancyPixels>
10   <minOccupancyPixels>50</minOccupancyPixels>
11   <flocculate value="true">
12     <duringDistribution>true</duringDistribution>
13     <range>5.0</range>
14     <factor>0.5</factor>
15     <cycles>3</cycles>
16     <maxTrials>10000</maxTrials>
17   </flocculate>
18   <sphereSamplingPoints>92</sphereSamplingPoints>
19   <checkIndividualArea>false</checkIndividualArea>
20   <checkNoArea>false</checkNoArea>
21   <findCoveringSphere>false</findCoveringSphere>
22   <saveHydout>false</saveHydout>
23   <savePorein>false</savePorein>
24   <saveGrains>true</saveGrains>
25   <saveDeepImages depth="128">true</saveDeepImages>
26 </reactor>
```

If the time steps can be divided into blocks of time with regular limits, line 6 can be replaced by the block shown below.

```
1    <minTime>0.0</minTime>
2    <maxTime>24.0</maxTime>
```

```
3    <step>1.0</step>
4    <minTime>26.0</minTime>
5    <maxTime>72.0</maxTime>
6    <step>2.0</step>
```

The above block adds time steps from 0 to 24.0 in steps of 1.0 and then from 26.0 to 72.0 in steps of 2.

2.2 Defining materials (Material)

The user has to define all materials, including the reactants, products and the unreactive materials in the system. Some other materials may have to be defined to store buffers or to serve as intermediate products. Individual parameters and their meaning are listed below. The titles of the following subsections show the names of the parameters and the class or type of the parameter is given in parentheses after the name. The XML tag for defining materials is `<material>`.

2.2.1 Name (String)

This field defines the case-sensitive name of the material. Materials have to be assigned a unique name. In the GUI, creating a new material with the same name will replace the earlier material. This property is defined using the attribute “name” with the `<material>` tag.

2.2.2 Id (int)

This field defines the unique index of the material. The user may define ids of materials to control the order of processing and listing in output files. The id is an optional attribute. The value has to be an integer greater than zero and should be unique. Ids are automatically assigned to the other materials in no particular order. This property is defined using the attribute “id” with the `<material>` tag.

2.2.3 Density (double)

This field defines the “base” density of the material. Although this is the default density of the material, a `DensityVariation` plugin can be defined to vary this density (see section 3.1.1 on page 24). It is assumed that the volume of the materials defined in the reactions is based on this density (see section 2.4.3 on page 19). The XML tag for defining this property is `<density>`.

2.2.4 Initial fraction (double)

This field defines the fraction of the computational volume occupied by this material at the beginning of the simulation. For example, if a material occupies half the volume in the beginning, this value has to be set to 0.5.

2.2.5 Inner (boolean) and Variant (Material)

This field is true if this material can be formed only inside the original bounds of the particles. This implies that if this material is produced in a reaction where another material from the same particle is consumed, the volume of this material produced can not be more than the volume of the reactant material from that particle.

If a material is defined to be inner, a variant material for this material has to be defined. Since the inner material can only occupy the space within the original boundaries of the particle, any amount of this material produced beyond this limit is transferred to the variant material, which can either increase the size of its own particles, or transfer this amount to another form due to a change defined in a reaction. The XML tags for defining these properties are `<inner>`, `<hasVariant>` and `<variantName>`.

2.2.6 Discrete (boolean) and HostMaterial (Material)

Materials can be classified as discrete or diffuse. Diffuse materials are assumed to be uniformly mixed in discrete materials. Only discrete materials can be hosts for diffuse materials and each discrete material can host several diffuse materials. For example the user can use the names of the host materials to define layers in particles (see section 2.3.2 on page 17). The averaging of the amounts of the materials is done at the level of each particle.

If a material is not defined to be discrete, a host for this material has to be defined. Only discrete materials are drawn in the images produced by the programme. The XML tags for defining these properties are `<discrete>` and `<hostName>`.

Note: A material can be present only in one of the layers of only one of the particle models. If a material is required to be present in several particle types, several materials and rule for inter-conversion or distribution have to be defined.

2.2.7 Universal (boolean)

If this parameter is set to be true, the material is defined to be present “universally”, implying that it is not present in a layer in any of the particles. The material can still have an initial and current amount and buffers for reactions, and can be used as reactant or product (see section 2.4.3 on page 19). The XML tag for defining this property is `<universal>`.

Note: All universal materials have to be defined to be discrete.

2.2.8 User defined parameter (double)

This field defines any arbitrary user defined parameter that may be used by any plugin. For example the `DiffusionControlledKinetics` plugin uses this parameter as the diffusivity of the material (see section 3.1 on page 24). The XML tag for defining this property is `<userDefinedParameter>`.

2.2.9 Material class (String)

This property can be used to group materials of different names together, e.g. to calculate the total mass of all materials in a given class. This property is defined in the XML file with the attribute `class` in the material definition.

2.2.10 Replace (Material)

As most reactions entail consumption and production of materials, the phase present at one of the pixels of a discretised computational volume may change with time. Also, since the size of spherical particles only represents a circumsphere, many points inside this circumsphere may not belong to that particle and may come from a layer in another particle. This means that rules to change the phase in the pixels may have to be defined. To do that, a list of other discrete materials that a material is allowed to replace due to reaction in an image can be defined. In all cases, the programme tries to guess the correct settings for this parameter. The XML tag for defining this property is `<replace>`.

2.2.11 Color (Color/String)

This parameter defines the colour of the material in images produced by the programme. If a string is passed for this parameter, the programme tries to match

the string with English names of colors and hexadecimal RGB definitions. If a colour for a discrete material is not defined, a random colour is assigned. The XML tag for defining this property is `<color>`.

2.2.12 Example setup of a material in the XML file

The block of XML below shows an example setup for a material

```
1 <material id="2" name="myMaterial">
2   <density>2.0</density>
3   <diffusivity>1.0</diffusivity>
4   <inner>true</inner>
5   <universal>false</universal>
6   <discrete>false</discrete>
7   <hostName>myOtherMaterial</hostName>
8   <hasVariant>true</hasVariant>
9   <variantName>myThirdMaterial</variantName>
10  <initialAmount>0.30</initialAmount>
11  <color>red</color>
12  <replace>anotherMaterial</replace>
13 </material>
```

If the material is discrete (line 6), line 7 (hostName) can be omitted. If the field hasVariant (line 8) is false, line 9 (variantName) can be omitted. For automatic detection of the materials to replace, line 12 can be omitted.

2.3 Defining particle models (ModelGrain)

Particle models (or ModelGrains) define the concentric arrangement of materials in same and different particles. Any number of particle types can be defined to accommodate all the materials. The XML tag for defining particle models is `<modelGrain>`.

2.3.1 Name (String)

Every particle model has to be given a unique case-sensitive name. In the GUI, creating a new particle model with the same name will replace the earlier particle model. This property can be defined as the attribute “name” with the `<modelGrain>` XML tags.

2.3.2 Layers (Material)

Concentric layers of discrete materials can be defined to be present in a ModelGrain. Since diffuse materials are assumed to be mixed with the discrete materials, the layers can only contain discrete materials. All non-universal discrete materials have to be present in one of the ModelGrains and any material can be present only in one of the layers of one of the ModelGrains. Layers can be added using the XML tag `<layer>`.

2.3.3 Initial particle size distribution (String)

The file-name of the file containing the initial particle size distribution of this particle type is assigned to this parameter. The XML tag for defining this property is `<filePSD>`. This file can have one of the several different formats, as listed in the following subsections.

2.3.3.1 Cumulative particle size distribution

In the first possible format, each line of this file should contain a diameter (double) followed by the fraction of the powder by volume that is smaller than this size (double), separated by space or tab. Although the diameters can be listed in any order, if the diameters are listed in ascending order, the fractions listed should be in ascending order as well. The programme renormalises the fractions assuming that 100% of the powder is smaller than the largest diameter listed. The word “Cumulative” (without quotes) can be optionally provided on the first line of the file to denote this format. The distribution will be assumed to be linear between the provided points.

2.3.3.2 Differential particle size distribution

In the second possible format, similar to the first format, each line of this file should contain a diameter (double) followed by the fraction (double) of the volume of the powder present that is of this size, separated by space or tab. The diameters can be listed in any order. In most practically observed particle size distributions, a peak in the fraction values is observed for one of the diameters between the first and the last listed diameters exclusive. This format can be automatically recognised if such is the case. The word “Differential” (without quotes) can otherwise be put on the first line of the file to denote this format. The distribution will be assumed to be linear between the provided points.

2.3.3.3 Mono-disperse particle size distribution

A powder can be defined to be mono-disperse (having all particles of the same size) using the third file format. In this format the file should contain only one line of data followed by a line-break. This line should contain the diameter of the particles followed by any arbitrary number, preferably 1.0, , separated by space or tab. This format can be automatically recognised if only a single line of text. The word “Single” (without quotes) may otherwise be optionally included on the first line of the file to denote this format. The number of particles is rounded to the nearest integer calculated from the total initial material of the materials present in the particles and the size of these particles.

2.3.3.4 Rosin-Rammler particle size distribution

A powder can be defined to have a particle size distribution as defined by a Rosin-Rammler function using the fourth file format. In this format, the word “Rosin-Rammler” (without the quotes) is given on the first line. The median diameter of the distribution (d_m) is given on the second line. The distribution width is given on the third line of the file (w_d). The fraction (f) of the powder below a certain diameter (d) is given by the function below.

$$f = 1.0 - \exp\left(-\left(\frac{d}{d_m}\right)^{w_d}\right) \quad (2.1)$$

The list of diameters at which the fractions are to be calculated are then listed on different lines on the input. The diameters can be listed in any order. The fraction below the smallest diameter is assumed to be zero and that below the largest diameters is assumed to be 100%. The distribution is assumed to be linear between the listed diameters.

2.3.4 Example setup of a ModelGrain in the XML file

The following XML block shows the setup of a ModelGrain the XML input file.

```
1 <modelGrain name="myGrain">
2   <layer>myInnermostMaterial</layer>
3   <layer>mySecondMaterial</layer>
4   <layer>myThirdMaterial</layer>
5   <filePSD>psdfile.txt</filePSD>
6 </modelGrain>
```

2.4 Defining Reactions

All conversion between different materials have to be defined as reactions. Consequently, all the reactions defined do not necessarily require to be actual chemical processes in the system and can be used for many different purposes. The XML tag for defining a reaction is `<reaction>`.

2.4.1 Name (String)

Reactions have to be assigned a unique case-sensitive name. In the GUI, defining a new reaction with the same name replaces the earlier reaction. This property can be defined using the attribute “name” in the tag `<reaction>`.

2.4.2 Order of reactions

The reactions can be arranged in the order they are required to be implemented. This order is important as it might affect the results. In the GUI, the reactions can be moved up or down to set the order. This property can be defined using the XML tag `<order>`.

2.4.3 Reactants and Products

Materials, along with the volumetric ratios of the materials being formed or used, can be defined as reactants and products. The volumetric ratios are read as doubles. The main reactant should be assigned a ratio of 1.0, and the other reactants are assigned ratios that can be calculated from the stoichiometry and the base density of the material (see section 2.2.3 on page 13). These properties are defined using the tags `<reactant>` or `<product>` and `<materialName>` and `<ratio>`.

2.4.4 Discrete (boolean)

If a reaction is set as discrete, the reaction rate is calculated individually for each particle. Also, any change in amount in the materials present in this particle, due to the reaction of this particle, is carried out from the same particle. In order for a reaction to be discrete, it should contain at least one non-universal reactant.

If this parameter is set to false, the rate of calculation is calculated for the entire system and then the reactant and the product materials are distributed

between particles according to the plugins defined. This property can be defined using the XML tag `<discrete>`.

2.4.5 Successor (Reaction)

Other reactions can be added as successors to this reaction. When a reaction is deactivated by a trigger, its successor reaction is activated. The triggers of the successor reactions still have to keep them activated (see section 3.1.10 on page 29)

2.4.6 Example setup of a reaction in the XML file

The following XML block shows the setup of a reaction the XML input file.

```
1 <reaction name="myReaction">
2   <order>0</order>
3   <reactant>
4     <materialName>myMainReactantMaterial</materialName>
5     <ratio>1.0</ratio>
6   </reactant>
7   <reactant>
8     <materialName>mySecondReactantMaterial</materialName>
9     <ratio>0.5677</ratio>
10  </reactant>
11  <product>
12    <materialName>myMainProductMaterial</materialName>
13    <ratio>0.596</ratio>
14  </product>
15  <product>
16    <materialName>mySecondProductMaterial</materialName>
17    <ratio>1.57</ratio>
18  </product>
19  <discrete>true</discrete>
20 </reaction>
```

The name of the reaction can be given with the attribute name. The order defines the position of this reaction in the list of all reactions. All the reactants and products can be listed using the tags `reactant` and `product` respectively. The name of the material for the reactant of the product can be given between the tag `materialName` and the volumetric ratio of the reactant or the product can be given between the tag `ratio`. The value of the ratio is read as a double.

2.5 Run Options

One of several options can be chosen in the GUI and the XML file. In the XML file they can be given using the command tag and in the GUI they can be accessed in the edit reactor dialogue.

2.5.1 Clean start with new packing

This option can be found in the “Packing and Interactions” tab in the edit reactor dialogue, and can be enabled using the following line in the XML file.

```
1 <command>start</command>
```

If this option is selected, a new packing is generated and the simulation is carried out.

2.5.2 Start from stored packing or earlier simulation.

This option can be found in the “Packing and Interactions” tab in the edit reactor dialogue, and can be enabled using the following line in the XML file.

```
1 <command>restart</command>
```

If this option is selected, the initial packing of particles generated in a previous run is reused to carry out a new simulation. In order to use this option, the initial packing should have been saved in the grains folder using the saveGrains option in the reactor (see section 2.1.14 on page 11). The same command can also be used to restart the process from a later step in an earlier simulation. For example, to restart from step 10 of an earlier simulation, the following command can be used:

```
1 <command fromStep="10">restart</command>
```

The information from the earlier run can also be contained in a folder other than the current file folder. This folder can be defined between <copyFrom> tags in the reactor definition (see section 2.1.4 on page 7).

2.5.3 Recreate images with a new resolution

This option can be found in the “Images and Optimisation” tab of the edit reactor dialogue and can be enabled using the following line in the XML file.

```
1 <command>redoPixels</command>
```

If this option is selected, the results from an earlier simulation are used to regenerate images, sections or deep (see section 2.1.7 on page 8). In order to use this option, the initial packing should have been saved in the grains folder using the saveGrains option in the reactor (see section 2.1.14 on page 11).

2.6 Other run options only in the XML file

Some other command options are available only in the XML file. They may be added in the GUI in the future. Some other commands and their functions are listed below. All these commands have to be given between the tags named command.

- savePixels: saves all the voxels in a binary files (see section 6.4 on page 51).
- pixelsNoHeaders: saves all the voxels in a binary files without the ASCII header at the top of the file (see section 6.4 on page 51).
- readStep: can be used to read the information at only one of the steps from one of the previous simulations. The step number has to be added as the value for the attribute step to the command tag. The images and files can be produced for this step.
- saveElements: can be used to save nodes and elements of the voxel mesh that can be loaded in an FEM programme.
- markNoPores: ensures that the pore elements are not stored in the elements files, to reduce their size. If this case is selected, the program also wraps a single voxel layer of another material around the elements. The ID of this material is the larger of 12 or one more than the number of elements.
- doContacts: enables the calculation of contact surface between particles, both by calculating the overlap between the spherical particles and by calculating the contacts between pixels from different particles. This option can be computationally expensive and increases the memory consumption of the pixel mesh by 5 times. The results are stored in the details file (see section 6.1 on page 50).
- doVectorBurning: (opposite of noVectorBurning) enables the calculation of the solid fraction of the volume that is connected from one boundary to another. This calculation also counts dead-end branches that may be connected to both ends. The results for each time step are stored in the file connectionMyProject.csv (where MyProject is the name of your project).

- `doBurning`: can be used to save only the pixels that are from the particles that are connected to the boundary, while the others are marked as pores. This option increases the memory consumption of the pixel mesh by 5 times. The connected spheres are identified using vector burning that can be turned on using the `doVectorBurning` command. This command can be followed by the letters X, Y or Z in capitals to set the direction of burning (e.g. `doBurningX` or `doBurningY`).
- `noBurning`: ensures that all pixels from all sphere, and not just the ones from the particles connected to the boundary are saved.

Chapter 3: Plugins in μic

3.1 Plugins

μic uses plugins to customise the models used in the simulations. The plugins can be used to control many different aspects of the simulations. The package `plugins` contains the definition of the `MicPlugin` interface, which has to be implemented to create a valid plugin. The inner packages of this package contain the abstract, in some cases default implementations and/or other specialised implementation of the plugins. The individual packages and plugin types are described below. Please see chapter 7 for details of plugins included with μic .

3.1.1 Type `densityVariation`

The `densityVariation` type plugins control the variation of density of materials. The density of the materials is averaged at the level of each particle. The abstract class is `DensityVariation` and the default implementation is `DefaultDensityVariation`. The default implementation fixes the density of the material to the base value given during its definition.

This plugin is called by each layer of each particle when the amount of the material in the layer is changed. The change in volume of the layer is then calculated according to the new density returned by the plugin.

Apart from the requirements from all plugins of μic , an instance of this plugin has to implement three methods:

- `double getDensity(ReactingParticle grain, double time)`
This method returns the new density of the material at any given time, if the density varies for each particle individually.
- `double getDensity(Material material, double time)`

This method returns the new density of the material at any given time, if the density varies for the entire system.

- `double getInitialDensity(Material material)`

This method returns the initial density of the material at the beginning of the simulation.

3.1.2 Type `grainProportionInitialiser`

The `grainProportionInitialiser` type plugins distribute the space in the particles to the different materials that may be present in the particle. The main abstract class is `GrainProportionInitialiser` and the default implementation is `DefaultGrainProportionInitialiser`. The default implementation distributes the space in the particle in proportion to the total amount of the materials.

This plugin is called by each `ModelGrain` during initialisation of the reactor.

Apart from the requirements from all plugins of μic , an instance of this plugin has to implement one more method:

- `void initialiseMaterialProportions(double rCubeTot)`

This method distributes the initial amount of material, given by the parameter `rCubeTot` to the particles.

3.1.3 Type `kinetics`

The kinetics type plugins control the rate of reactions. The main abstract class is called `Kinetics`. No default implementation of the type is provided.

All kinetics plugins defined for each reaction is called before each step of implementation of the reaction.

Apart from the requirements from all plugins of μic , an instance of this plugin has to implement two more methods:

- `getAmount(double time, double dt, ReactingParticle particle)`

This method returns the amount of material reacting at the given time in the duration `dt` in the given particle. If the `returnsDr` field of the instance is set to true, as it is by default, the reduction of the radius of the reactant has to be returned. If the field is set to false, the volume of the reactant material is to be returned as `drCube`. The `drCube` is the cube of the radius of a sphere that has the same volume as the volume of the material to react.

- `double getAmount(double time, double dt)`

This method returns the amount of material reacting at the given time in the duration `dt` for all particles. The volume of the reactant material is to be returned as `drCube`. The `drCube` is the cube of the radius of a sphere that has the same volume as the volume of the material to react.

3.1.4 Type `materialDistributionProfile`

The `materialDistributionProfile` type plugins control the distribution of newly produced material to different particles. The main abstract class is called `MaterialDistributionProfile`. The default implementation is called `RandomDistributionProfile`. The default implementation distributes the material randomly to the particles. This plugin can be equally well be used to remove reactant materials from particles in specific fractions.

This plugin is called by the plugin `particleDistributionProfile`, which is in turn called by each particle type for each particle.

Apart from the requirements from all plugins of `µic`, an instance of this plugin has to implement two more methods:

- `double getPortion(ReactingParticle grain, double time)`
This method has to return the proportion of the material that goes to this particle. The proportions only have to be consistent between all particles and can have any value.
- `double getReportion(ReactingParticle grain, double time)`
This method has to return the proportion of the material that goes to this particle to distribute the amount of material that could not be distributed to the particles in the first attempt. The proportions only have to be consistent between all particles and can have any value.

3.1.5 Type `nucleiGenerator`

The `nucleiGenerator` type plugins control the number and location of new particles. The main abstract class for controlling only the number of nuclei is `NucleiGenerator` and the main abstract class for controlling both the number and location of the nuclei is `NucleiLocationGenerator`. A default implementation is not provided.

All instances of this plugin defined for each `ModelGrain` is called the `ModelGrain` before the implementation of reactions at each step of the simulation

The total number of nuclei is the sum of the number of particles returned by all instances of the plugins defined.

Apart from the requirements from all plugins of μic , an instance of `NucleiGenerator` has to implement one more method:

- `int numberOfNuclei(PeriodicVolume volume, double time, double dt)`
This method returns the total number of particles of this particle type at any moment in time. The parameter `dt` gives the length of the simulation step.

This method in an instance of `NucleiLocationGenerator` has to return 0. This method should also generate the location of the new particles and register them by calling the method `registerNewParticle` in the `NucleiLocationGenerator` class.

3.1.6 Type packer

The packer type plugins pack the particles present in the initial volume. The abstract implementation for this type is called `Packer`. The default implementation is `RandomPacker`, which packs the particles at random locations starting from the largest particle, moving in descending order of radii. This plugin is called by the `Reactor` when it is post-conditioned (method `postCondition()`).

Apart from the requirements from all plugins of μic , an instance of this plugin has to implement three more methods.

- `void pack(GrainList<ReactingParticle> grainList)`
This method is called to pack the given list of grains in the volume.
- `boolean placeNewGrain(ReactingParticle grain, int rejects)`
This method can be called to add a new particle to the system even at a later step in the simulation. The integer `rejects` represents the maximum number of trials the programme must make to place the particle. If a suitable position for the particle is not found after these trials, the method would return false, else true would be returned.
- `void placeNewGrain(ReactingParticle grain)`
Like the previous method, this method can be used to find the position for a new grain in the system, even at a later step in the simulation. However, since the maximum number of trials is not specified here, the programme should attempt to place the particles until a suitable position is found.

3.1.7 Type `particleDistributionProfile`

The `particleDistributionProfile` type plugins distribute the new space available in a grain to different materials. The main abstract class is called `ParticleDistributionProfile` and the default implementation is `DefaultParticleDistributionProfile`. The default implementation distributes materials to a particle based on the `MaterialDistributionProfile` of the materials in the grain.

This plugin is called by all `ModelGrains` after each reaction to account for distribute all the material in the buffers.

Apart from the requirements from all plugins of `µic`, an instance of this plugin has to implement one more method:

- `void distribute(double time, Reactor reactor)`
This method distributes the materials in the grains at the given time.

3.1.8 Type `particleIterator`

The `particleIterator` type plugins choose sets of particles that have to be given to a reaction together. The package for this plugin contains the class `ParticleChooser` and the abstract main class `ParticleIterator`. The reaction uses the particles returned by the plugin to calculate the kinetics and if the reaction is discrete, the reactants and products that are present in these particles are given to these particles. The default implementation is called `SingleParticleIterator` and returns particles of the defined `ModelGrain` one at a time in the order returned by the iterator of the `grainList` of the `ModelGrain`.

This plugin is called upon by all discrete reactions when they are implemented, after the amount for reaction has been calculated. If any of the particles contains a material that is changed in that step of the reaction, the material is changed in the particle.

Apart from the requirements from all plugins of `µic`, an instance of this plugin has to implement three more methods:

- `boolean hasNext()`
This method returns true if the iterator has more particles to return
- `ReactingParticle[] next()`
This method returns the next set of particles

- `void remove()`

While this method should, theoretically, remove the last returned set of particles from the underlying structures, it can be left empty.

3.1.9 Type `postProcessor`

This plugin type allows the user to execute arbitrary code before the start of the simulation and at the end of each step of the simulation. The abstract main class for this plugin type is `PostProcessor`. The reactor element calls the `process(double time, int step)` method of this plugin just after initialisation and after each step of the simulation. An example use of this plugin type would be to create and store files with the specific information that the user requires in a user-defined format. By default, the reactor does not have any post-processors defined. A reactor can be defined to contain multiple post-processors and the `process` method of each one of the defined instances will be called in no particular order. This method is called when either of `hydrate`, `restartHydration` or `redoPixels` options is chosen.

Apart from the requirements from all plugins of μic , an instance of this plugin has to implement one more method:

- `process(double time, int step)`

This method can be used to carry out any arbitrary procedure for the given time and step in the simulation.

3.1.10 Type `reactionTrigger`

The `reactionTrigger` type plugins can be used to activate or deactivate reactions. The main abstract class is called `ReactionTrigger` and the default implementation is `ActiveTrigger`. The default implementation keeps the reaction active at all times.

The triggers are checked by each reaction, at each step, just before the implementation of the reaction for that step. The reaction is implemented only if the trigger returns true. When a reaction is deactivated if a trigger returns false, or otherwise, if any reactions defined as successors are activated unconditionally. The triggers of the successor reactions still have to return true to keep them active.

Apart from the requirements from all plugins of μic , an instance of this plugin has to implement four more methods:

- `public boolean hasNext()`
This method returns true if the iterator has another array of `ReactingParticles` to return.
- `public ReactingParticle[] next()`
Returns the next array of `ReactingParticles`.
- `public void remove()`
Although this method should theoretically remove the last array of values returned from the underlying collection, in this implementation would usually do nothing.
- `public ParticleIterator newCopy()`
Returns a new copy of this iterator initialised using the original initialisation parameters.

3.1.11 Type recalcuator

The recalcuator type plugins can be used to recalculate various parameters in the simulation. The main abstract class is called `Recalcuator` and the default implementation is called `DefaultRecalcuator`. The default implementation does not change anything.

The recalcuator is called by each reaction, at each step, just before the implementation of the reaction for that step. The plugin can be used to change system variables, or the reaction itself.

Apart from the requirements from all plugins of `µic`, an instance of this plugin has to implement one more method:

- `void recalculate(double time)`
Any required changes to the parameters are to be made in this method.

3.2 To create, compile and use an external plugin

- Write a new class that is a subclass of one of the plugin classes.
- Do not forget to include a `noarg` constructor and the correct list of constructor parameter types and names
- Create another constructor which initialises all the required parameters
- The array `initialisationValues`, which is included in the super-implementation of all plugin types, has to be initialised with the constructor parameters used to create the plugin, in the order given in the constructor.

- Provide the functionality in the relevant method(s) of the plugin by overloading the abstract methods in the main abstract plugin.
- The method `constructorParameterClasses` should return an array of the classes of the required constructor parameters, as returned by the `.class` field of the class, in the order in which they are defined in the constructor
- The method `constructorParameterNames` should return an array of the names of the constructor arguments, in the order they are defined in the constructor
- The constructor arguments have to be given in the order they are present in the constructor
- Go to the root source folder of the project in which the `.java` file is included
- Compile using: `javac -classpath <classpath to MicTheModel.jar> <path to new plugin>`
- Compile the set of classes into a jar file jar use the command:
`jar cf myExternalPlugins.jar */*.class`

3.3 Loading a plugin using the XML file

Both internal and external plugins can be loaded into a simulation using the XML file. In the file, the name, type, path and parameters for loading the plugin have to be defined, as shown in the following XML block.

```
1 <plugin type="kinetics">
2   <classPath>lib/externalPlugins/</classPath>
3   <className>myKineticsPackage.myKineticsClass</className>
4   <parameter name="constant 1" type="number">
5     <class>float</class>
6     <value>1.14E-4</value>
7   </parameter>
8   <parameter name="constant 2" type="number">
9     <class>int</class>
10    <value>3.0</value>
11  </parameter>
12  <parameter name="constant 3" type="number">
13    <class>double</class>
14    <value>3.232e-10</value>
15  </parameter>
16  <parameter type="reaction">
17    <name>myReaction</name>
18  </parameter>
19  <parameter type="reactant">
```

```
20         <hostReaction>myReaction</hostReaction>
21         <name>myMaterial</name>
22     </parameter>
23 </plugin>
```

3.3.1 classPath and className

The classPath parameter in the XML loader has to show the path to the root folder where the compiled classes are present. For example, if your compiled plugin class file is called myKineticsPackage.myKineticsClass.class in the folder lib/externalPlugins/myKineticsPackage, the classPath has to be set to lib/externalPlugins/, which is the root folder of the classes, as in line 2. It must be noted that classes in Java have to be in a folder that has the same name as the package name, and internal packages which are separated by period marks should be recursively kept inside inner folders. When loading a class which is contained in the compiled µic package, the word precompiled can be set as the classPath.

The className has to be marked as <package>.<pluginName>, myKineticsPackage.myKineticsClass in the example above, as on line 3. The attribute type has to be set to the type of the plugin (for types of plugins see section 3.1 on page 24).

3.3.2 Loading parameters using the XML file

The parameters required for loading the constructor have to be listed in the order they are present in the definition of the constructor. The parameters can be declared using the tag parameter, as shown in lines 4, etc. in the XML example above. The attribute type of the parameter tag defines the class of the parameter. The types that can be loaded are listed in the subsections below. The types are all in lower case. The name of the parameter can be optionally given as the value of the attribute name for easier readability.

3.3.2.1 boolean

Parameters of type boolean can be used to load a true or false value. An example of the definition is shown below.

```
1 <parameter name="isTrue" type="boolean">
2   <value>true</value>
3 </parameter>
```

The value between the value tags has to be one of true or false.

3.3.2.2 material

Parameters of type material can be used to load one of the materials defined. An example of the definition is shown below.

```
1 <parameter name="materialParameter" type="material">
2   <name>myMaterial</name>
3 </parameter>
```

The value between the name tags has to be the name of the material to be loaded.

3.3.2.3 modelGrain

Particle models (modelGrain) can be loaded using parameters of type modelGrain. An example of the definition is shown below.

```
1 <parameter name="modelGrainParameter" type="modelGrain">
2   <name>myModelGrain</name>
3 </parameter>
```

The value between the name tags has to be the name of the particle model to be loaded.

3.3.2.4 number

Parameters of type number can be used to load one of the seven primitive number types, viz. byte, char, short, int, long, float, and double. An example definition is shown below.

```
1 <parameter name="constant 1" type="number">
2   <class>float</class>
3   <value>1.14E-4</value>
4 </parameter>
```

The value between the class tags has to be the name of primitive type. The value between the value tags is the value for the parameter.

3.3.2.5 product

Products in reactions can be loaded using parameters of type product. An example of the definition is shown below.

```
1 <parameter name="myProduct" type="product">
2   <hostReaction>myReaction</hostReaction>
```

```
3     <name>myMaterial</name>
4   </parameter>
```

The value between the hostReaction tags has to be the name of the reaction to which the product belongs. The value between the name tags has to be the name of the product material to be loaded.

3.3.2.6 reactant

Reactants in reactions can be loaded using parameters of type reactant. An example of the definition is shown below.

```
1   <parameter name="myReactant" type="reactant">
2     <hostReaction>myReaction</hostReaction>
3     <name>myMaterial</name>
4   </parameter>
```

The value between the hostReaction tags has to be the name of the reaction to which the reactant belongs. The value between the name tags has to be the name of the reactant material to be loaded.

3.3.2.7 reaction

Parameters of type reaction can be used to load one of the reactions defined. An example of the definition is shown below.

```
1   <parameter name="reactionParameter" type="reaction">
2     <name>myReaction</name>
3   </parameter>
```

The value between the name tags has to be the name of the reaction to be loaded.

3.3.2.8 reactor

Parameters of type reactor can be used to give the main reactor as a constructor parameter to any plugin type. An example definition is shown below.

```
4   <parameter name="reactorParameter" type="reactor">
5     <name>myReactor</name>
6   </parameter>
```

The value between the name tags has to be the name of the reactor as defined during its definition. In the GUI, the user is given a drop down list containing only one element which can be chosen. This element shows the name of the reactor as defined.

3.3.2.9 array

Arrays of any objects of any of the types above can also be loaded as a parameter. The elements in the array can be of the same or of different types. Elements have to be added using the tag element, with the type and the value of the element being defined in the same way as for the parameters. An example is shown below.

```
1 <parameter name="randomArray" type="array">
2   <element type="reaction">
3     <name>myReaction</name>
4   </element>
5   <element type="modelGrain">
6     <name>myModelGrain</name>
7   </element>
8 </parameter>
```

The file reader currently only supports one-dimensional arrays.

3.4 Loading parameters using the GUI

Both internal and external plugins can be loaded using the GUI as long as the requirements for the plugins have been satisfied (see section 3.2 on page 30 and section 3.1 on page 24).

3.4.1 Choosing the class path, name and type

In the GUI the plugins can be added using the add plugin button, or using the “Add new plugin” item in the Edit menu of the menu bar. The first window that appears asks for the class path, the type of plugin and the name of the class.

This dialogue box can load plugins packaged with μic , a local compiled plugins packaged in a jar file, or a local class directory structure. By default the dialogue box displays the list of available plugins packaged with μic for the chosen type. The list updates itself when another jar file or a class directory structure is chosen.

When choosing a directory structure of class files, the user must select the root folder of the build. This means that the folder containing the first set of packages as subdirectories should be selected.

When the user changes the selection of the plugin type, the list of names of classes available is loaded. Please note that the dialogue does not load internal member classes even if they provide a complete plugin implementation.

If the plugin class is successfully located, a form is displayed where the user can set the values for the parameters when ok is clicked. An error dialogue box is shown if the programme is not able to locate or load the plugin class and the user can correct the fields and retry.

3.4.2 Declaring parameters using the GUI

The parameter form contains the Name and type of the plugin at the top, followed by a list of the names of the constructor parameters required, as defined in the plugin loaded. The type of the parameter is also shown next to the parameter name. The user has to define the values in the third column. The parameters are listed in the order in which they are required by the constructor of the plugin.

The type of input required depends on the class of the required parameter, as defined by the plugin. The components displayed to set the values for the parameters of different input types are listed in the subsections below.

3.4.2.1 boolean

A check-box is provided to choose true or false. If the check-box is selected true is passed and false is passed if it is not selected.

3.4.2.2 material

A drop-down list of the available materials is provided. The user can choose the material to be passed as a parameter.

3.4.2.3 modelGrain

A drop-down list of the available particle types is provided. The user can choose the particle type to be passed as a parameter.

3.4.2.4 number

An empty text box is provided, in which the user can enter the number to be passed as a parameter. The value is first parsed as double and then cast to the required type.

3.4.2.5 product

A drop-down list of the available reactions is provided, followed by a second empty list. The user first has to choose the reaction to which the product belongs. The second list is then populated with the products of the reaction chosen. The product is listed as the name of the reaction, followed by the name of the material and then the volumetric ratio defined in the reaction. The user can choose from the products listed in the second box to pass as a parameter.

3.4.2.6 reactant

A drop-down list of the available reactions is provided, followed by a second empty list. The user first has to choose the reaction to which the reactant belongs. The second list is then populated with the reactants of the reaction chosen. The reactant is listed as the name of the reaction, followed by the name of the material and then the volumetric ratio defined in the reaction. The user can choose from the reactants listed in the second box to pass as a parameter.

3.4.2.7 reaction

A drop-down list of the available reactions is provided. The user can choose the reaction to be passed as a parameter.

3.4.2.8 reactor

The user is given a drop down list containing only one element with the name of the reactor which is already chosen.

3.4.2.9 array

A label saying “Number of elements” followed by an empty text box and a “Create” button is provided. The user first has to fill the empty text field with the number of elements in the array to be passed. When the create button is clicked, a new form is created based on the number of elements provided by the user.

If the plugin reports the requirement of an array of one of the recognised types discussed above, the new form gives rows of elements that depend on the type of the element. The input components displayed are the same as those listed above, depending on the type of the element. For example, a drop-down list of reactions is displayed if the required type is a reaction, or an empty text box is displayed if a number is required.

If the plugin does not define the type of the array, a form is displayed where the types of the elements in the array can be individually chosen using drop-down lists. Based on the chosen types, a new form is generated where the values of the elements can be set. The input component displayed is the same as those listed above, depending on the type of the element.

It must be noted that the GUI currently only supports one-dimensional arrays. The behaviour for multi-dimensional arrays is unknown. The GUI also does not support editing plugins. A copy of a pre-defined plugin can, however, be loaded and redefined, after which the earlier plugin may be deleted. It must however be noted that some plugins may automatically replace other plugins and the earlier instance will be automatically deleted. So, it is advised to refresh the view and view the plugin before a plugin is deleted. Deletion is not supported for ParticleIterator type plugins due to the semantics of `µic`.

Chapter 4: A summary of the algorithm of μic

4.1 Control flow

- The programme generally starts at the method `main(String[] args)` in the class `start.Main`
- If an XML file input is chosen the control goes to the method `doXMLStart(String[] args)` in the class `start.Main` and from there to the method `readFile(String fileName)` in the class `asciiIO.XMLInputReader`
- The control is passed around in the class `asciiIO.XMLInputReader` by the method `createReactor(Document document)` to read the input file and progressively build the problem in the memory
- The `readFile(String fileName)` method in the `XMLInputReader` class returns an instance of the `elements.Reactor` class to `Main`, based on the input file
- `Main` calls the method `implement()` in the `elements.Reactor` class to start the simulations
- In the class `reactor`, the control-flow is decided based on what the programme has been instructed to do, e.g. to run a new simulation (`hydrate`), to restart the simulation from a starting point generated in an earlier simulation (`restartHydration`) or to just regenerate the pixels from an earlier simulation (`redoPixels`)
- For a new simulation the system is initialised and the method `implementReactions()` is called; which in turn calls the method `implementReactions(double time, double dt, double correctionFactor, int step)` for each step in the simulation

4.2 Initialising reactor in μic

The following steps are followed to build the initial reactor.

- Create materials
- Create reactor with given size
- Generate particle types and choose the optimal implementation for individual types
- Calculate initial amount of materials
- Generate particles from particle size distributions
- Initialise plugins
- Distribute initial materials in the particles based on particle initialisation plugins
- Place the particles in the reactor based on the defined Packer plugin.
- Initialise reactions
- Call the `process(0.0f, 0)` method of all post-processors.

4.3 Calculating evolution

- For each time step:
 - Calculate free area of particles if requested at each step
 - Go to each particle-type
 - Nucleate new grains according to the nucleation controlling plugins
 - Go to each reaction
 - Rebalance based on any recalculation plugins defined
 - Check activity of reactions based on defined trigger plugins
 - If reaction is set to be calculated for individual particles (discrete as opposed to universal), go to each set of particles chosen by the particle chooser plugin
 - Go to each particle in the chosen set
 - Calculate reaction amounts for the given time step based on the given kinetics plugins
 - Check for availability of materials and if needed correct reacting amount
 - Remove reactant materials from the chosen particles based on material distribution plugins defined
 - Distribute product materials to the chosen particles based on material distribution plugins defined

- Keep the amounts of reactants and products of materials that do not go to the chosen particles, for use later
- If reaction is set to be calculated globally:
 - Calculate reaction amounts based on the given kinetics plugins
 - Check for availability of all materials and if needed correct reacting amount
 - Keep the amounts of reactant and product products for use later
- Distribute the reactant and product materials accumulated to different particles based on material distribution plugins for materials in particles or simply change amounts for universal materials
- Distribute any remaining material to particles according to defined material distribution plugins
- Save information at the end of the step (material amounts, reaction information, particle sizes, generate voxelised images, etc.)
- Call the process(double time, int step) method of all post-processors.

4.4 Changing layers and materials in particles

- Calculate the free surface area if the user has requested to do so before each change in the particle
- Particle checks for the required change in volume
- If space is available to place or remove all the materials:
 - Go to each layer starting from the inner-most layer going outwards
 - In each layer correct the new amounts and densities of all materials
 - Inform the materials of the changes in amounts
 - Change the radius of the layer to accommodate the changes
- If no free surface is available and an increase in volume is required
 - Go to each layer starting from the inner-most layer going outwards
 - In each layer account for the densities and any partial amount of material that can be accounted for
 - Keep remaining material for use later
 - Inform the materials of the changes in amounts
 - Change the radius of the later to accommodate the changes
 - Attempt to distribute remaining amounts to another particle in the neighbourhood, or return the remaining amounts to material buffers

- Correct the outer radius of the particle

4.5 Calculating free surface of a particle

- Sample the surface of the particle using homogeneously or randomly distributed points on the surface (see section 2.1.12 page 11)
- For each point on the surface
 - Use the overlaid occupancy grid to find other particles in the neighbourhood
 - Check if the point lies inside the circumsphere of any of these particles
 - If true the point is covered
- The free surface of the particle is calculated as the fraction of points that are not covered
- The particle is informed of the calculated free surface

4.6 Function of overlaid occupancy grid

- Divide the volume into a regular grid of smaller cubes, with each cube containing a non-duplicate list of particles that, even partially, occupy the space in the cube
 - Care has to be taken to cover all the cubes that lie within the reach of all particles or edge-effects may occur
- Access methods allow quick assignment, removal and access of “occupants” in each cube
- The size of the grid is provided externally, typically the diameter of the median particle, in the list of all particles ordered by size, can be used as the side of each cube. For each case, a speed-wise optimal size of the grid elements exists
- When looking for the neighbourhood of a point, all occupants of the box in which the point lies can be returned
- When looking for overlaps with another sphere, occupants from all boxes this sphere even partially occupies have to be compiled

4.7 Generics in μic

Several implementations, including the periodic volume and the occupancy grid, have been templatised to avoid run-time type-checks, which can be expensive. This is mostly for particles, which inherit from a deep hierarchical structure

starting from a three-dimensional vector. Different levels of the hierarchy provide for different functionality and compatibility with other possible types. At the last level, five specialised implementations of reactive particles have been provided, which are automatically chosen based on the number of layers and the reactivity of the materials in each layer. For most classes, the abstraction is at the level of `IndexedSphere`, which is simply a sphere with fields to store the radius, the x, y and z of the centre, a unique index for the particle and a boolean to store if the particle currently has free surface area.

Chapter 5: Running μ ic

5.1 Executing user and GUI generated XML input files

5.1.1 Running the graphical user interface (GUI)

In versions starting September 2009, the graphical user interface (GUI) for μ ic has been included in the main code. A separate executable for the graphical user interface is no longer available. If the name of the executable jar file is `MicTheModel.jar`, the graphical user interface can be started using the following command:

```
1 java -jar MicTheModel.jar
```

5.1.2 Executing from the GUI

Versions of the GUI starting April 2009 support running an XML file directly from the interface. If the user clicks on the “Run” button in the menu or the toolbar (shortcut key F6), a dialogue box will appear. The box contains some information followed by a text field containing the command which will be executed.

WARNING: The GUI can only execute the instructions in a pre-saved XML file. Any changes made after a file was saved will not be included in the run. In order to include the latest changes in your definitions, please save the system as an XML input file. Please make sure that the problem is fully defined in the memory and the latest XML file has been saved before attempting to open the run dialog box.

Please make sure that the command in the text field correctly points towards the μ ic executable jar file, the input XML file and the output file. The screen

output for the new process created will be saved in the output file indicated in the run command (the file name following the '-o=' in the last argument of the command) and will not be displayed on the GUI console. The programme automatically creates a new randomly named output file. The user may change this name in the command to any name he prefers. Please note that if the user changes this name to that of a file that already exists, the program will overwrite the file without warning the user.

Please refer to section 5.1.3 for more information about the command used to execute μic .

The command is executed when the user clicks on 'Yes' and a process monitor is displayed. This monitor shows the name of the input file being executed, the time at which the process was started, the state of the process and the time at which the process completed or exited. The monitor has a button to kill the process that it displays. There is another button “Show/Update output”. This button will display the latest version of the output from the programme in the window. The output can be refreshed by pressing the “Show/Update output” button again.

The process monitor cannot be brought back up if it is closed. To stop a process after closing the process monitor, please use the process/task manager of your operating system.

5.1.3 Executing using command line

The XML input files generated by the GUI or by the user can be executed using the executable jar file generated by compiling μic (see section 1.14 on page 5). Assuming the compiled name of the executable programme is MicTheModel.jar, the following command can be used to execute the XML file.

```
1  java -Xmx2000m -jar MicTheModel.jar xmlfile1.xml xmlfile2.xml  
   xmlfile3.xml -o=outputfile.txt
```

The phrase -Xmx2000m reserves a maximum of 2000 megabytes of memory in the system for the execution and is a native java option. All the XML input files listed in the command will be executed sequentially.

The last argument (-o=outputfile.txt) routes all the screen output information into the file outputfile.txt. If this argument is not used, all screen output will be displayed on the console. If the user is not executing the code

through a console and not using the '-o=' argument, the screen output information may be lost.

Note: To execute the GUI execute the following command:

```
1 java -jar MicTheModel.jar
```

The XML file is obtained only when the information in the GUI is saved using the save dialogue.

5.2 Using the user menus in μic

μic also contains a simple user menu to allow users to carry out additional operations on the results produced from the earlier executions of the programme. The command to get this user menu is shown below.

```
1 java -Xmx2000m -jar MicTheModel.jar -menu
```

When this command is executed, the first question displayed allows user to choose between three options by entering a number after the following message: “Hydration(1) or Porosity using pixels(2) Refined Porosity (other than 1 or 2)?”

5.2.1 Simulation of the reactions

If the number entered is 1, the user is then directed to enter the name of an XML file to be executed. This file is then executed and the programme is terminated at the end of the simulation.

5.2.2 Pore-sizes and connectivity

If the number entered is 2, the user can analyse the porosity in the volume in an approximate simplified way. The pore-size distributions of the microstructures are calculated using a three-dimensional pixel-erosion method. In the method used here, the vector microstructure was overlaid with a cubic mesh and the voxels were marked to be solid or pore based on their occupancy. The pore voxels sharing a face with a solid voxels were marked with the number 1, and in the next step the pore voxels sharing a face with the voxels marked 1, were marked with 2, and so on until all voxels were marked with the number of steps required to reach them from solid boundaries. This method allows identification of voxels farthest away from the solid walls in each pore, which are defined as the centres of the pores. The schematics of this algorithm are shown in figure 5.1.

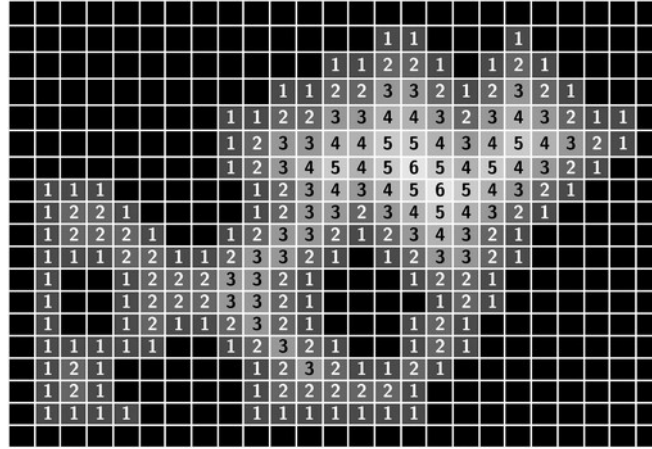


Figure 5.1: Erosion to identify pore-skeleton using pixels. The solids are shown in black and pores in shades of grey. The numbers in the pores show the number of steps needed to reach the closest solid pixel.

The extent of each pore can be identified by a walk in the reverse direction from the pore-centres. While this is only one of the many possible definitions of pores and pore-sizes, and ignores information regarding pore-topology, it can be used to obtain quick comparative results.

This calculation is possible only if the user chose to store the porein files during the simulation of the reactions. When this option is selected, the user is asked to input the simulation step for which the calculation has to be carried out. This is the number of the step in the original simulation where the porein files were generated.

The user has to then input the size of the pixels (voxels) to be used for the analysis. The user is then asked if the simulation of mercury-intrusion porosimetry (MIP) is to be carried out. This simulation of MIP is also only approximate and only gives a qualitative idea of the ink-bottle effect.

In the calculation of the MIP, once the size of all the pores is marked, a liquid is assumed to flow into the largest pores on the surface and the pores of the same size directly connected to them. The size of the pore on the surface in which the fluid is then reduced one voxel at a time. All the pores that are larger than or equal to this size and are connected to a pore already filled with the fluid are then recursively filled. A relation between the size into which the fluid is allowed to flow and the total number of the pore-pixels filled by the fluid is obtained during this process.

If the user chooses to perform MIP, he is asked for a multiplication factor. This factor is the number of times the pixelised computational volume has to be reproduced along each axis. Since the volume has periodic boundary conditions, tiling produces continuous volume. Therefore, if a factor of 2 is input, the total volume increases 8 times. This can be used to study the effect of size on MIP.

The results are stored in the folder pixelPorosity, which is created inside the project folder. The file name for step 10 for a simulation with 400 pixels on each side will be step0010number400.csv. The file contains a list of sizes, followed by the number of pixels and the fraction of pixels larger than the size. The same information is repeated for each set of MIP performed.

A simulation using 1 billion voxels usually consumes around 1.5 to 2 GB of RAM, depending on the number of particles and the settings used. The number of voxels is multiplied if the MIP is carried out on a multiplied volume.

5.2.3 Calculating pore-sizes on refined mesh

An even more approximate analysis of pore-sizes on the refined mesh at a user-chosen position is carried out if a number other than 1 or 2 is chosen. If this option is chosen, the user is asked for the step in the simulation for which the analysis is to be carried out. This is the number of the step in the original simulation where the porein files were generated.

The user is then asked for the pixel size for total porosity. Here a size that gives approximately 10 pixels on each side can be provided. For example, if the size of the volume is 100, the user can give 10 here.

The user is then asked for the x, y and the z coordinate of the starting point for the volume to be refined one by one. The user has to then define the size of the pixel for the refined analysis and then the size of the smaller volume where the refined analysis has to be carried out.

The results are stored in the folder limitedPixelPorosity, which is created inside the project folder. The file name for step 10 for a simulation with 400 pixels on each side will be step0010number400.csv. The file contains a list of sizes, followed by the number of pixels and the fraction of pixels larger than the size.

The analysis carried out on the refined voxels in this case uses the same algorithm as for the previous option. The voxels on the boundary of the volume to be refined as set to be solid before this analysis. This analysis is expected to give

hugely variable and incorrect results in most cases, but may still be useful for some users. The user should still consider all the implications of the method used. A simulation using 1 billion voxels usually consumes around 1.5 to 2 GB of RAM, depending on the number of particles and the settings used.

Chapter 6: Output files

6.1 Details file

A file called details followed by the name of the project and then .csv is created in the project folder. This file contains the overview of all the materials and reactions at every step in the simulation.

- The first column lists the time at each simulation step.
- The second column lists the degree of reaction, which shows the fraction of the initial materials remaining in the system.
- The following columns list the volume of the main reactant or the product of the reactions used or produced respectively at each step in the simulation.
- The columns after that list the number of pixels of each of the materials in the voxel discretised form of the computational volume. It should be noted that in these images all the diffuse materials are replaced by their discrete host materials.

6.2 Grains files

This folder contains all the grains for all particle models at each step in the simulation in a binary format. This file is written in the method `saveGrains(int step)` in the class `elements.ModelGrain`.

6.3 Images

Images at the planes at the centre of the computational volume along each axes, at each step in the simulation is stored in folders which are called `imagesX`, `imagesY` or `imagesZ`, followed by the number of pixels in each direction. If the user

has chosen to save images with depth perception, a folder called `deepimages`, followed by the number of pixels in each direction.

6.4 Pixels

If the user has chosen to store the voxels in a file, they are stored in a folder called `pixels`, followed by the number of pixels in each direction. This file contains a 5 line ASCII header, followed by the phase number at each voxel in binary. The first line says `VOXELS`, the second says `"1"` (without the quotes) and the following three lines the number of pixels along the three directions. The files are named `pixels` followed by the simulation step.

6.5 Custom output files

New plugins of the type postprocessor can be created to generate user defined output files. Since the postprocessor plugins can get access to all information available in the Reactor, almost any information can be output after any step of the reaction. More details on this type of plugin can be found in section 3.1.9 on page 29.

6.6 Note about CSV files

Most of the ASCII files produced by `µic` are in the CSV format. For best performance, CSV files should be opened in a spreadsheet application. They can also be viewed in any text viewer or editor. This file is in ASCII format and contains values separated by commas. If the default separator for a system is not a comma, for example in French systems, the file can be opened in any text editor and all the commas can be replaced by semicolons, or any other separator the system might require. If the system requires a comma to in place of the period character to separate the fractional part of a number, after the commas have been replaced by semicolons or another separator, all the period characters can be replaced by commas.

The language for non-unicode programmes in the system can also be changed to International English to overcome this problem.

Chapter 7: Packaged plugins in μ ic

7.1 Plugins

μ ic uses plugins to customise the models used in the simulations. The plugins can be used to control many different aspects of the simulations. The package `plugins` contains the definition of the `MicPlugin` interface, which has to be implemented to create a valid plugin. The inner packages of this package contain the abstract, in some cases default implementations and/or other specialised implementation of the plugins. The individual packages and plugin types with the plugin implementations provided in μ ic are described in the following subsections. (Also see chapter on plugins in the documentation for μ ic.)

7.2 Type `densityVariation`

The `densityVariation` type plugins control the variation of density of materials. The density of the materials is averaged at the level of each particle. The abstract class is `DensityVariation` and the default implementation is `DefaultDensityVariation`. The default implementation fixes the density of the material to the base value given during its definition.

This plugin is called by each layer of each particle when the amount of the material in the layer is changed. The change in volume of the layer is then calculated according to the new density returned by the plugin.

Apart from the requirements from all plugins of μ ic, an instance of this plugin has to implement three methods:

- `double getDensity(ReactingParticle grain, float time)`
This method returns the new density of the material at any given time, if the density varies for each particle individually.

- `double getDensity(Material material, float time)`
This method returns the new density of the material at any given time, if the density varies for the entire system.
- `double getInitialDensity(Material material)`
This method returns the initial density of the material at the beginning of the simulation.

Following are the implementations pre-packaged in μic .

7.2.1 DefaultDensityVariation

This implementation keeps the density of the material in all particles fixed at the base amount defined with the material.

7.2.2 FixedVolumeDensityVariation

This implementation changes the density of a material in the particles such that the size of the layer stays the same. If the mass of the material is zero, the density is set to zero.

7.3 Type `grainProportionInitialiser`

The `grainProportionInitialiser` type plugins distribute the space in the particles to the different materials that may be present in the particle. The main abstract class is `GrainProportionInitialiser` and the default implementation is `DefaultGrainProportionInitialiser`. The default implementation distributes the space in the particle in proportion to the total amount of the materials.

This plugin is called by each `ModelGrain` during initialisation of the reactor.

Apart from the requirements from all plugins of μic , an instance of this plugin has to implement one more method:

- `void initialiseMaterialProportions(double rCubeTot)`
This method distributes the initial amount of material, given by the parameter `rCubeTot` to the particles.

The implementations pre-packaged for this plugin are described below.

7.3.1 DefaultGrainProportionInitialiser

This implementation assigns the same fraction of space to the materials in each particles. This means that the fraction of space occupied by a given material in a particles to the total volume of the particle equals the total volume of that material in the system to the total volume of all materials in the system.

7.3.2 LimitedRandomGrainProportionInitialiser

This implementation of the plugin assigns the space available in a particle randomly to the available materials. The user can however can limit some of the materials only to particles that lie between given minimum and maximum size.

7.3.3 RandomGrainProportionInitialiser

This implementation of the plugin assigns the space available in a particle randomly to the available materials.

7.4 Type kinetics

The kinetics type plugins control the rate of reactions. The main abstract class is called Kinetics. No default implementation of the type is provided.

All kinetics plugins defined for each reaction is called before each step of implementation of the reaction.

Apart from the requirements from all plugins of `µic`, an instance of this plugin has to implement two more methods:

- `getAmount(float time, float dt, ReactingParticle particle)`

This method returns the amount of material reacting at the given time in the duration `dt` in the given particle. If the `returnsDr` field of the instance is set to `true`, as it is by default, the reduction of the radius of the reactant has to be returned. If the field is set to `false`, the volume of the reactant material is to be returned as `drCube`. The `drCube` is the cube of the radius of a sphere that has the same volume as the volume of the material to react.

- `double getAmount(float time, float dt)`

This method returns the amount of material reacting at the given time in the duration `dt` for all particles. The volume of the reactant material is to be

returned as drCube. The drCube is the cube of the radius of a sphere that has the same volume as the volume of the material to react.

7.4.1 AvramiKinetics

This plugin gives rates of reaction as given by the Avrami equation³.

The rate of reduction of the radius of a particle can be calculated from the equation:

$$\frac{dr}{dt} = -k \cdot n \cdot t^{n-1} \cdot \exp(-k \cdot t^n) \quad (7.1)$$

In the above equation, r is the radius of the reacting layer of the particle, k and n are Avrami parameters and t is the time.

The global rate of reaction is given by the equation:

$$\frac{dV}{dt} = \bar{k} \cdot n \cdot t^{n-1} \cdot \exp(-\bar{k} \cdot t^n) \cdot V_m \quad (7.2)$$

In the above equation, V is the volume of the material reacting, k and n are Avrami parameters, t is the time.

7.4.2 AvramiKineticsNoReduction

For individual particles, this plugin returns the rate of reaction as given by the Avrami equation (eqs. 7.2 and 7.1 above) until the peak is reached. A constant rate, given by the rate at the peak, is returned after the peak.

7.4.3 DensifyingNGKinetics

This plugin simulates the growth of a product using nucleation and growth kinetics. The product is assumed to grow outwards from a spherical surface. The initial density of the product is assumed to be lower than the highest possible density. The rate of growth of the product depends on the free surface area of the product. The density of the product is assumed to increase linearly with time at a given rate. For more information about this information see Bishnoi 2008^{4,5}.

This plugin does not support the calculation of the universal rate of reaction and only the rate for the individual particles is calculated.

At any moment in time, the fraction of covered surface on the particle is assumed to be given by the Avrami equation in two equations, as follows:

$$f_{covered} = 1 - \exp(-k_{hor} \cdot t^n) \quad (7.3)$$

In this equation k_{hor} (parameter kHor) depends on the rate at which the product covers the surface, n (parameter n) is a coefficient that should ideally be 2 if newer nuclei of the product do not form on the surface after the first nucleation event and 3 if the nucleation continues at a constant rate. Fractional values for n may also be used, but their physical meaning is not clear. Time is given by t and $f_{covered}$ is the fraction of the surface covered by the product. Instead of defining k_{hor} , which can be difficult to calculate, the user can define the time (t_{frac} , parameter tFrac) at which a given fraction of the surface ($f_{initial}$, parameter initialFraction) is covered.

It is assumed that the product grows at a constant rate outwards from its current surface at the rate given by parameter kVer. Since it is assumed that the density of the product is lower in the beginning and grows with time, the initial density (ρ_{min} , parameter minDensity), the maximum possible density (ρ_{max} , parameter maxDensity) and the rate of increase of density (k_{den} , parameter kDen) also have to be defined. If the age of the product at any point is given by τ , the density of the product at that point is given by:

$$\rho_{\tau} = \rho_{max} - (\rho_{max} - \rho_{min}) \cdot \exp\left(\frac{-k_{den} \tau}{\rho_{max} - \rho_{min}}\right) \quad (7.4)$$

As the product is assumed to grow at a constant rate outwards from the surface, the age of the product at any point at a distance r from the surface can be given by:

$$\tau = t - \frac{r}{k_{ver}} \quad (7.5)$$

The fraction of the volume occupied by the product at a distance r from the surface can be calculated as:

$$V_{filled} = f_{filled-\tau} = 1 - \exp(-k_{hor} \cdot \tau^n) \quad (7.6)$$

In the above equation, it is assumed that the internal surface available for densification reduces linearly with density. The density of the material at any point around the particle includes the effect of both, the lower packing and the

fraction of the volume filled by the lower-density product at that time. This means, that the density of the product at any point is given by:

$$\rho_{app} = V_{filled}(t, r) \cdot \rho_{\tau}(t, r) \quad (7.7)$$

Based on the above equations, the total amount of the product to be present and hence the amount to be added in the particle at any given time is calculated. This is done by dividing the layer of products into discrete spherical shells of equal thickness. The number of shells is based on the parameter `noOfSamplingPoints`, which defines the number of shells to be cut at the time when the surface of the particle is, arbitrarily, 99.999% covered. The density and the fraction filled is calculated at the centre of each shell and the total mass of material to be present is calculated using the following equation:

$$m_{total}(t) = \sum_{i=1}^n \rho_{app} \cdot (r_i^3 - r_{i-1}^3) \quad (7.8)$$

The average apparent density can be calculated as below:

$$\bar{\rho}_{app} = \frac{\sum_{i=1}^n \rho_{app} \cdot (r_i^3 - r_{i-1}^3)}{\sum_{i=0}^n (r_i^3 - r_{i-1}^3)} \quad (7.9)$$

The total amount of material to be produced in a step is calculated for all particles assuming that they can derive reactants from other particles as well. The total amount of reactant required is distributed to individual particles in proportion to the amount of the reactant material available in each particle.

This plugin uses some more plugins defined as inner classes to enforce some of the rules of this plugin, as described in the following subsections.

7.4.3.1 `DensifyingNGKinetics.ProductDistributionProfile`

As the kinetics only control the amount of material reacting, this sub-plugin ensures that the correct amount of product is distributed to the particles, as shown above.

7.4.3.2 `DensifyingNGKinetics.ReactantDistributionProfile`

This sub-plugin ensures correct collection of reactants from the particles.

7.4.3.3 DensifyingNGKinetics.SpaceDensityVariation

This sub-plugin controls the average density of the material in the particle. This is the average density calculated as shown in equation 7.8 above.

7.4.4 DiffusionKinetics

Reaction kinetics governed by the layers of materials deposited over particles with different diffusion coefficient and some free surface available for ion interchange. For this plugin, only the rate of reaction of individual particles can be calculated.

It is assumed that the resistance offered to reaction is the sum of the resistance offered by individual layers, which in turn is proportional to the thickness of the layer, as in the equation below.

$$\Omega_{total} = \sum_{layers} \frac{thickness}{diffusivity} \quad (7.10)$$

The rate of change of radius can then be calculated as:

$$\frac{dr}{dt} = - \frac{free\ area\ fraction}{\Omega_{total}} \quad (7.11)$$

The user defined parameter defined with each material (see 2.2.8 on page 15) is used as the value of diffusivity of each material. The free area denotes the fraction of the free surface of the particle.

7.4.5 DiffusionProportionateKinetics

This plugin controls the rate of reaction in a way similar to that for DiffusionKinetics, except that the user can scale the diffusivity of all materials using the parameter called factor. So equation 7.9 can be rewritten as:

$$\Omega_{total} = \sum_{layers} \frac{thickness}{factor \cdot diffusivity} \quad (7.12)$$

7.4.6 ExternallyDefinedKinetics

This plugin forces the degree of hydration of a given material to that defined in an ASCII text file. The text file can provide information about the rate of reaction of the material by prescribing a value of the degree of hydration, amount

remaining, fraction remaining or amount produced at given points in time. The type of the value is defined by the text in the first line of the text file, which can be degree, amount, fraction or produced respectively. The user only needs to provide different times followed by the value defined, in separate lines for each time. Values will be linearly interpolated between the prescribed points.

This implementation only works for universal (as opposed to discrete) reactions and the users should consider defining customised `MaterialDistributionProfiles` for their reactant and product materials.

7.4.7 FixedRadiusKinetics

This plugin changes the size of all particles of the given material to the given value. When a new particle is created, this plugin attempts to set its size to the given radius. Once the size is changed to the given radius, or the size of the particle is not zero, the size is not changed. This plugin assumes that the particle contains a single layer with a single material.

7.4.8 NGKinetics

This plugin simulates the growth of a product on the surface of particles, modelling nucleation and growth mechanism. It is assumed that the product forms on the surface of the particle and grows outwards at a constant rate. The functionality of this plugin is similar to `DensifyingNGKinetics`, except:

- The product is assumed to be uniform in density, though the density has to be varied to control the fraction of volume filled by the product at any given distance from the surface of the particle.
- The reactant is only derived from the particle on which the product grows. This assumption artificially leaves spaces in the volume, which lie within the reaches of the particles that have exhausted their reactants before the density of the product could reach a maximum and/or the entire volume within their reach could be filled by lateral growth of the product.

This plugin should be used keeping the points above in mind and may lead to errors. For cases where the product can derive its reactants from other particles, `DensifyingNGKinetics` can be used by defining a zero k_{den} .

This plugin only provides the implementation for individual particles and cannot calculate the universal reaction rate.

This plugin uses a sub-plugin to enforce the lower average density of the product in the particle resulting from only a fraction of the volume being filled.

7.4.8.1 NGKinetics.NGDensityVariation

This sub-plugin enforces the lower average density of the product in the particle resulting from only a fraction of the volume being filled.

7.4.9 NucleiGrowthKinetics

A constant rate of growth of radius is simulated in this plugin. This also means a rate of growth proportional to the free surface area of a particle.

7.4.10 OuterNucleiGrowthKinetics

This plugin creates an assembly of spheres starting from the surface of the particles to generate conical growths on the surface of the particles. This plugin simulates the nucleation and growth of a material at different rates perpendicular and parallel to the surface of a particle.

This plugin uses the following parameters:

- kHor: The rate of growth of the material parallel to the surface of the particle.
- Kvert: The rate of growth of the material perpendicular to the surface of the particle
- minNuclDensity: Minimum density of nuclei on the surface (number per unit surface)
- minNuclNum: Minimum number of nuclei for each particle
- kNucl: Rate of formation of nuclei per unit surface per unit time
- t0: Initial time
- alpha0: Starting degree of reaction (not used, kept for compatibility with older input files)
- minRad: Not used, kept for compatibility with older input files
- order: Order of this kinetics in the reactant
- reaction: The reaction for this kinetics
- reactant: The reactant for which the reaction amount is to be calculated
- nucleatingMaterial: The material growing on the surface

This plugin only provides functionality for individual particles and does not support universal calculations.

This plugin uses other plugins defined as sub-classes to enforce some rules, as described in the following sub-sections.

7.4.10.1 OuterNucleiGrowthKinetics.SurfaceNucleator

This sub-plugin enforces the location of the creation of new nuclei to form the conical shape required by the plugin.

7.4.10.2 OuterNucleiGrowthKinetics.SurfaceNucleiProfile

This plugin distributes product material to the nuclei formed.

7.4.11 PhaseBoundaryKinetics

This plugin defines a constant rate of reduction of radius of a product layer. This means that the change of radius of the reactant radius can be given by the equation:

$$dr = -k \cdot dt \quad (7.13)$$

7.4.12 ReactAllKinetics

Reaction kinetics that react all available material in a step. For individual particles this plugin returns the amount of material in the particle and for universal reactions this plugin returns the total amount of material in the system.

7.4.13 RoughDissolvingSurfaceKinetics

Assumes the formation of pits on the surface of the particles that dissolve at a rate proportional to the surface of the pit. In this plugin, the fraction of the surface of a particle reacting is assumed to be given by the equation:

$$f_{reacting} = 1 - \exp(-k \cdot t^n) \quad (7.14)$$

This equation is similar to equation 7.3 above. It is assumed that the reaction proceeds inwards at a constant rate k_{ver} until the entire particle is consumed. The functionality of this class is similar, though inverse as it is for a reactant rather than a product, to NGKinetics.

The fractional consumption in the outer parts of the reactant layer are simulated using a plugin defined as a sub-class.

7.4.13.1 RoughDissolvingSurfaceKinetics.RoughDissolvingDensityVariation

This sub-plugin enforces the average change in density due to partial consumption of the reactant from the outer part of the reactant layer.

7.4.14 SphericalDiffusionProportionateKinetics

Reaction kinetics governed by the layers of materials deposited over particles with different diffusion coefficient and some free surface available for ion interchange. For this plugin, only the rate of reaction of individual particles can be calculated.

It is assumed that the resistance offered to reaction is the sum of the resistance offered by individual layers. As it is in spherical coordinates, this equation assumes perfectly radial flows and isotropic diffusion coefficients. The consumption of reactant is also assumed to be uniform over the surface of the particle. The rate is corrected for the reduction of the surface of hydrates in direct contact with water, due to impingement from neighbouring particles.

$$\Omega_{total} = \sum_{layers} \left(\frac{1}{radius_{inner}} - \frac{1}{radius_{outer}} \right) \cdot \frac{1}{factor \cdot diffusivity} \quad (7.15)$$

The rate of change of radius can then be calculated as:

$$\frac{dr}{dt} = - \frac{free\ area\ fraction}{\Omega_{total}} \quad (7.16)$$

The user defined parameter defined with each material (see 2.2.8 on page 15) is used as the value of diffusivity of each material. The free area denotes the fraction of the free surface of the particle.

7.4.15 SurfaceNucleiGrowthKinetics

Creates nuclei on the surface of particles and makes them grow proportional to their surface area. The parameters for this plugin as as follows:

- kNucl: The rate for formation of nuclei on the surface per unit time per unit surface area

- distanceFactor: The rate at which the new nuclei move away from the particles
- minNuclDensity: The minimum number of nuclei per unit surface of the particles
- minNuclNum: The minimum number of nuclei on each particle
- kGrowth: The rate of growth of radius of the spherical nuclei
- t0: The starting time
- alpha0: The starting degree of reaction (Not used, kept for compatibility with older input files)
- minRad: Not used, kept for compatibility with older input files
- order: The order of this instance of this plugin
- reaction: The reaction for which this plugin is defined
- reactant: The main reactant for the reaction
- nucleatingMaterial: The material that forms the nuclei on the particles

This plugin uses plugins defined as sub-classes to enforce certain other rules.

7.4.15.1 SurfaceNucleiGrowthKinetics.SurfaceNucleator

This sub-plugin creates the locations of the new nuclei.

7.4.15.2 SurfaceNucleiGrowthKinetics.SurfaceNucleiProfile

This plugin distributes the material to the particles.

7.4.16 VolumeProportionateKinetics

This plugin reacts a given fraction of a material present in a particle or universally.

7.5 Type materialDistributionProfile

The materialDistributionProfile type plugins control the distribution of newly produced material to different particles. The main abstract class is called MaterialDistributionProfile. The default implementation is called RandomDistributionProfile. The default implementation distributes the material randomly to the particles. This plugin can be equally well be used to remove reactant materials from particles in specific fractions.

This plugin is called by the plugin `particleDistributionProfile`, which is in turn called by each particle type for each particle.

Apart from the requirements from all plugins of `µic`, an instance of this plugin has to implement two more methods:

- `double getPortion(ReactingParticle grain, float time)`
This method has to return the proportion of the material that goes to this particle. The proportions only have to be consistent between all particles and can have any value.
- `double getReportion(ReactingParticle grain, float time)`
This method has to return the proportion of the material that goes to this particle to distribute the amount of material that could not be distributed to the particles in the first attempt. The proportions only have to be consistent between all particles and can have any value.

7.5.1 InnerLayerDistributionProfile

This profile distributes the material proportional to the square of radius of of the layer of the given material for each grain. The minimum radius defined gives a bias to particles with smaller layers. If the radius is smaller than the given minimum radius the portion for the particle is proportional to the square of the given minimum radius.

7.5.2 RandomDistributionProfile

This plugin randomly distributes material produced to available grains.

7.5.3 SurfaceDistributionProfile

This profile distributes the available material proportional to the available surface area of each grain, i.e. the surface area of the circumsphere of the particle times the fraction of the surface that does not overlap with other particles. The minimum radius defined gives a bias to smaller particles. If the radius is smaller than the given minimum radius the portion for the particle is proportional to the surface of a particle with the minimum radius.

7.5.4 VolumeProportionateDistribution

Distributes the material in proportion to the amount of a given material in the particle.

7.6 Type nucleiGenerator

The nucleiGenerator type plugins control the number and location of new particles. The main abstract class for controlling only the number of nuclei is NucleiGenerator and the main abstract class for controlling both the number and location of the nuclei is NucleiLocationGenerator. A default implementation is not provided.

All instances of this plugin defined for each ModelGrain is called the ModelGrain before the implementation of reactions at each step of the simulation. The total number of nuclei is the sum of the number of particles returned by all instances of the plugins defined.

Apart from the requirements from all plugins of μic , an instance of NucleiGenerator has to implement one more method:

- `int numberOfNuclei(PeriodicVolume volume, float time, float dt)`

This method returns the total number of particles of this particle type at any moment in time. The parameter `dt` gives the length of the simulation step.

This method in an instance of NucleiLocationGenerator has to return 0. This method should also generate the location of the new particles and register them by calling the method `registerNewParticle` in the NucleiLocationGenerator class.

7.6.1 DendriticNucleiLocationGenerator

This plugin places end to end to form chains by overlapping the new particles by a given fraction of the radius. This plugin allows more than one particle to grow from the older particles, in effect creating branches. The structure formed is not technically dendritic as the locations of the spheres are random.

In this plugin, for each sphere in a chain (which has only one element in the beginning), a random point on the surface is generated. This point is projected outwards to the defined distance to the centre of the new sphere. If it is possible to place a new sphere at this location, without the new sphere overlapping with other

spheres (except the sphere with which it is supposed to overlap), the location is accepted. Otherwise the location is rejected. Only one location is attempted for each particle and the calculation moves on to the next sphere present in the chain. (see section 7.6.7 on page 66 below)

7.6.2 DependentExponentialNucleiGenerator

This nucleiGenerator generates exponentially decaying number of nuclei in each step. The final number of nuclei can be defined to be a given fraction or multiple of the number of grains of another particle model.

7.6.3 DependentLinearNucleiGenerator

This nuclei generator generates nuclei linearly with time based on initial number and initial and final times given by the user. The final number of nuclei can be defined to be a given fraction or multiple of the number of grains of another particle model.

7.6.4 ExponentialNucleiGenerator

This nucleiGenerator generates exponentially decaying number of nuclei in each step. The initial and final number of particles is fixed.

7.6.5 FixedNuclei

This is a non-generator as the number of nuclei remains fixed.

7.6.6 LinearNucleiGenerator

This nuclei generator generates nuclei linearly with time based on initial and final numbers and initial and final times given by the user.

7.6.7 SingleDendriticNucleiLocationGenerator

This plugin places end to end to form chains by overlapping the new particles by a given fraction of the radius. This plugin does not allow more than one particle to grow from the older particles, in effect creating single chains growing in random direction. The structure formed is not technically dendritic.

In this plugin, for the end spheres in a chain (which has only one element in the beginning), a random point on the surface is generated. This point is projected outwards to the defined distance to the centre of the new sphere. If it is possible to place a new sphere at this location, without the new sphere overlapping with other spheres (except the sphere with which it is supposed to overlap), the location is accepted. Otherwise the location is rejected. Only one location is attempted for each particle and the calculation moves on to the next chain. (see section 7.6.1 on page 65 above).

7.6.8 SurfaceNucleator

This plugin has a functionality similar to the plugin SurfaceNucleiLocationGenerator (see section 7.6.9 on page 67), but has been implemented without use of the NucleiLocationGenerator class, inheriting only from the NucleiGenerator class. In general the SurfaceNucleiLocationGenerator plugin should be used. This class is only an example of how the same functionality can be achieved without using NucleiLocationGenerator.

7.6.9 SurfaceNucleiLocationGenerator

This plugin creates new particles on the surface of particles of a given type. The parameters minNuclDensity and minNuclNumber can be used to control the number and density of new particles on the surface of the particles of the type surfaceGrain. The new particles are of the type modelGrain. kNucl controls the rate of increase of the density of nuclei on the surface. distanceFactor can be used to put the nuclei at an increasing distance from the particle surface, so at time t , the new nuclei can be placed at any random location between the surface and the distance $\text{distanceFactor} * \text{time}$ from the surface.

7.7 Type packer

The packer type plugins pack the particles present in the initial volume. The abstract implementation for this type is called Packer. The default implementation is RandomPacker, which packs the particles at random locations starting from the largest particle, moving in descending order of radii. This plugin is called by the Reactor when it is post-conditioned (method postCondition()).

Apart from the requirements from all plugins of `µic`, an instance of this plugin has to implement three more methods.

- `void pack(GrainList<ReactingParticle> grainList)`

This method is called to pack the given list of grains in the volume.

- `boolean placeNewGrain(ReactingParticle grain, int rejects)`

This method can be called to add a new particle to the system even at a later step in the simulation. The integer `rejects` represents the maximum number of trials the programme must make to place the particle. If a suitable position for the particle is not found after these trials, the method would return `false`, else `true` would be returned.

- `void placeNewGrain(ReactingParticle grain)`

Like the previous method, this method can be used to find the position for a new grain in the system, even at a later step in the simulation. However, since the maximum number of trials is not specified here, the programme should attempt to place the particles until a suitable position is found.

7.7.1 FlocPacker

This implementation can be used to, rather naively, simulate flocculation or to obtain higher packing density of spherical particles. In this plugin, the particles are parked at random locations, starting from the largest particle, moving in the descending order of radii, similar to that in `RandomPacker` (see section 7.7.2 on page 69), with the only difference being that if the `duringPacking` option is set to `true`, random locations are tried only a given number of times (given by the `trials` parameter). If after the given number of attempts, a location to place the particle is not found, a flocculation of the particles already packed is attempted to create more empty space in the system.

The flocculation is carried out using the following steps:

- For each particle already packed:
 - The closest particle within the range given by the parameter `flocculationRange` is located
 - The particle is moved so that the distance between these two particles is reduced by a factor given by `flocculationFactor`
- The above is done a number of times defined by `flocculationCycles`
- After the cycles above, for each particle already packed:

- The closest particle is within the range given by the parameter `flocculationRange` is located
- The particle is moved around the centre of the closest particle in small random steps, until overlap with another particle occurs or 1000 times, whichever occurs first
- The above process is done once for each particle

The `placeNewGrain` methods in this class behave similar to the behaviour in the plugin `RandomPacker` (see section 7.7.2 on page 69).

7.7.2 RandomPacker

This class simulates a random packing of the particles, by placing the particles in the reducing order of their radii at random locations in the volume. A random location for the centre of each particle is generated and is tested for overlaps with particles already present. The location is accepted if no overlap is found and another random location is tried if an overlap is detected.

The `placeNewGrain(ReactingParticle grain, int rejects)` method attempts to place a new particle in the volume without any overlaps with the other particles. A fixed number of new locations, given by the parameter `reject`, are tried and `false` is returned if a new location is not found within the given number of attempts. `True` is returned if a location is successfully assigned.

The method `placeNewGrain(ReactingParticle grain)`, like the previous method, attempts to place a new particle in the volume without any overlaps with the other particles. This method, however, does not stop after a given number of attempts and continues until a location is found, which may be infinite.

7.8 Type `particleDistributionProfile`

The `particleDistributionProfile` type plugins distribute the new space available in a grain to different materials. The main abstract class is called `ParticleDistributionProfile` and the default implementation is `NoReductionParticleDistributionProfile`. The default implementation distributes materials to a particle based on the `MaterialDistributionProfile` of the materials in the grain ensuring that the volume of the particles is not reduced.

This plugin is called by all ModelGrains after each reaction to account for distribute all the material in the buffers. In most cases, the MaterialDistributionProfile plugin should be used instead of this type.

Apart from the requirements from all plugins of `µic`, an instance of this plugin has to implement one more method:

- `void distribute(float time, Reactor reactor)`

This method distributes the materials in the grains at the given time.

7.8.1 DefaultParticleDistributionProfile

This distribution profile distributes the materials in the particle in the way they are given by the MaterialDistributionProfile (see section 7.5 on page 63) of the materials in all the layers. This distribution profile was the default implementation of this plugin type in the earlier versions of `µic`. However, this implementation can lead to a reduction in the volume of some particles in certain cases. A newer implementation called NoReductionParticleDistributionProfile (see section 7.8.2) has been provided and is now the default implementation for this plugin type.

7.8.2 NoReductionParticleDistributionProfile

This distribution profile distributes the materials in the particle in the way they are given by the MaterialDistributionProfile (see section 7.5 on page 63) of the materials in all the layers. In most cases, the MaterialDistributionProfile plugin should be used instead of this type. The difference from DefaultParticleDistributionProfile is that this distribution profile ensures that if possible, the size of the particles is not reduced. This means that when a material distribution profile extracts more volume of reactant than the volume of product added, the profile attempts to allocate more of the products to the particle or to draw less reactant from the particle.

7.9 Type particleIterator

The particleIterator type plugins choose sets of particles that have to be given to a reaction together. The package for this plugin contains the class ParticleChooser and the abstract main class ParticleIterator. The reaction uses the particles returned by the plugin to calculate the kinetics and if the reaction is

discrete, the reactants and products that are present in these particles are given to these particles. The default implementation is called `SingleParticleIterator` and returns particles of the defined `ModelGrain` one at a time in the order returned by the iterator of the `grainList` of the `ModelGrain`.

This plugin is called upon by all discrete reactions when they are implemented, after the amount for reaction has been calculated. If any of the particles contains a material that is changed in that step of the reaction, the material is changed in the particle.

Apart from the requirements from all plugins of μic , an instance of this plugin has to implement three more methods:

- `boolean hasNext()`
This method returns true if the iterator has more particles to return
- `ReactingParticle[] next()`
This method returns the next set of particles
- `void remove()`
While this method should, theoretically, remove the last returned set of particles from the underlying structures, it can be left empty.

7.9.1 SingleParticleIterator

The implementation `SingleParticleIterator` returns one particle at a time of the given `modelGrain`.

7.10 Type `postProcessor`

This plugin type allows the user to execute arbitrary code before the start of the simulation and at the end of each step of the simulation. The abstract main class for this plugin type is `PostProcessor`. The reactor element calls the `process(float time, int step)` method of this plugin just after initialisation and after each step of the simulation. An example use of this plugin type would be to create and store files with the specific information that the user requires in a user-defined format. By default, the reactor does not have any post-processors defined. A reactor can be defined to contain multiple post-processors and the `process` method of each one of the defined instances will be called in no particular order. This method is called when either of `hydrate`, `restartHydration` or `redoPixels` options is chosen.

Apart from the requirements from all plugins of `µic`, an instance of this plugin has to implement one more method:

- `process(float time, int step)`

This method can be used to carry out any arbitrary procedure for the given time and step in the simulation.

7.10.1 ClassWiseMaterialDetails

This plugin can be used to produce an output file called `classWiseDetails[Project Name].csv`, which lists the time, degree of reaction, the amounts of each reaction carried out and the masses of all materials present grouped by the defined class of the material (see section 2.2.9 on page 15).

7.10.2 LayerThicknessPlotter

This plugin can be used to store the minimum, maximum and average thickness of a given layer number in all grains of a given `modelGrain` type.

7.11 Type reactionTrigger

The `reactionTrigger` type plugins can be used to activate or deactivate reactions. The main abstract class is called `ReactionTrigger` and the default implementation is `ActiveTrigger`. The default implementation keeps the reaction active at all times.

The triggers are checked by each reaction, at each step, just before the implementation of the reaction for that step. The reaction is implemented only if the trigger returns true. When a reaction is deactivated if a trigger returns false, or otherwise, if any reactions defined as successors are activated unconditionally. The triggers of the successor reactions still have to return true to keep them active.

Apart from the requirements from all plugins of `µic`, an instance of this plugin has to implement four more methods:

- `public boolean hasNext()`

This method returns true if the iterator has another array of `ReactingParticles` to return.

- `public ReactingParticle[] next()`

Returns the next array of `ReactingParticles`.

- `public void remove()`
Although this method should theoretically remove the last array of values returned from the underlying collection, in this implementation would usually do nothing.
- `public ParticleIterator newCopy()`
Returns a new copy of this iterator initialised using the original initialisation parameters.

7.11.1 **ActiveTrigger**

This is the default implementation of this plugin, which always keeps the reaction active, unless it is externally disactivated, or the trigger to the reaction is externally changed.

7.11.2 **AmountControlledTrigger**

This trigger sets the reaction as active only when the amount of a given material is between the given minimum amount and maximum amount.

7.11.3 **InactiveTrigger**

This implementation of the plugin keeps the reaction always inactive, unless the trigger to the reaction is externally changed. When a successor of a reaction is assigned, it is assigned the `InactiveTrigger`, which is changed to `ActiveTrigger` when the successor reactions are activated.

7.11.4 **Multiple AmountControlledTrigger**

This trigger sets the reaction as active only when the amount of all given materials is between the given minimum amounts and maximum amounts for each material.

7.11.5 **PeakOffTrigger**

This implementation of the plugin can be used to activate a reaction once another reaction has reached its peak. While in most cases this behaviour may be obtained by using specific kinetics, the use of this plugin may be more convenient.

7.11.6 SurfaceRebalancingTrigger

This implementation of the plugin behaves in the same way as the plugin `recalculator.SurfaceRebalancer` (see section 7.12.2 on page 74) and should not be used.

7.11.7 TimeTrigger

This implementation of this plugin limits the activity of the given reaction between the given `startingTime` and `endTime`.

7.12 Type recalculator

The recalculator type plugins can be used to recalculate various parameters in the simulation. The main abstract class is called `Recalculator` and the default implementation is called `DefaultRecalculator`. The default implementation does not change anything.

The recalculator is called by each reaction, at each step, just before the implementation of the reaction for that step. The plugin can be used to change system variables, or the reaction itself.

Apart from the requirements from all plugins of `µic`, an instance of this plugin has to implement one more method:

- `void recalculate(float time)`
Any required changes to the parameters are to be made in this method.

7.12.1 DefaultRecalculator

This is the default implementation of this plugin type and does not do anything.

7.12.2 SurfaceRebalancer

This implementation of the plugin changes the ratios of the products, based on the available total free surface area of all the particles of each product, assuming that all products defined are variants of the same material. The rebalancing is done according to the defined base densities of the product materials. For example, if a reaction produces two variants of a material that have

to be distributed to two different types of particles, the user can arbitrarily divide the mass of the materials between the two variants. For each step in the simulation, this plugin will change the amounts of the two products forming such that:

- The total mass of the two materials is preserved and
- The individual amounts of the materials produced is proportional to the free surface area of their individual particle types.

Chapter 8: Known issues and workarounds

8.1 Known issues and workarounds with μic

Following are the known issues with μic .

8.1.1 Reduction in outer radii of particles

The programme will not be able to handle the reduction in the outer radius of particles. Any possible reduction due to a reaction has to be compensated by the addition of a product with at least the same amount. This limitation can be overcome by creating a material which represents empty space. This material must fill the space created due to the reduction and can either be diffused in the same layer as the reacting material or be added as an outer layer in the particle. The latter option is preferable.

The class PhasePixels, which is used to create pictures is also not able to adapt to reduction in the outer radii of particles and does not change the colour of a voxel once allocated back to the background colour. If the user adds the replacement material as another layer, the colour of the material can be set to the background colour in the images for an appearance of reduction in the particle size.

8.1.2 Compatibility issues

Starting from the version released on 21.01.2009, many floats used in the earlier version of the programme have been replaced by doubles. This might prevent the new programme from being able to read the grain files generated by

the older version of `µic`. Please use the older version of the programme to read the older format files, or regenerate the grain files using the new version of the program.

8.1.3 Boundary conditions

`µic` only supports periodic boundary conditions. This means that the part of any element crossing any boundary of the box is assumed to re-enter from the opposite boundary. For example, if the size of the box is 100, the coordinate 102, 202, 302, etc. are all the same as the coordinate 2. There is no fool-proof workaround for this issue, though various application dependent tricks, such as setting up a wall of particles at a boundary, can work.

8.2 Known issues and workarounds with GUI for `µic`

Following are the known issues with the GUI for `µic`.

8.2.1 Deletion of plugins of ParticleIterator type

The plugins of type `ParticleIterator` cannot be deleted directly from the GUI. A new definition of the plugin should, however, replace the earlier definition. The definition can also be deleted in the input XML file generated.

8.2.2 Editing defined plugins

The plugins defined in the GUI cannot be edited. They can, however, be copied. If the copy option is selected, a dialogue box with the defined values will appear, which can then be edited to create a new copy of the plugin. While in some cases the new plugin will automatically replace the earlier instance, in other cases the earlier instance will have to be deleted. To check if the earlier instance has been deleted, update the list of plugins and view the plugin to be deleted.

Chapter 9: Example tutorials

This chapter lists the step-wise procedure for setting up some simple problems using the graphical user interface (GUI). The steps to be followed for the definition of any problem are:

1. Open the GUI
2. Define the properties of the reactor (see section 2.1)
3. Create the materials and define their properties (see section 2.2)
4. Create the particle models and define their properties (see section 2.3)
5. Create the reactions and define their properties (see section 2.4)
6. Create all required plugins (see section 3.4)
7. Save the settings in an XML input file

In the following section, the above steps are presented in more details for specific example problems.

9.1 Nucleation and growth of single material

In this tutorial, we will set up the nucleation and growth of spherical grains of a single material. The main features of this problem are listed below.

- The size of the volume is 100 length units
- The simulation is to be carried out for 100 time units with steps of 1 time unit.
- There is a single material called myMaterial.
- The initial number of nuclei of this material is 0.
- 50 nuclei of this material are formed per unit time at random locations in the volume.
- The rate of change of the radius of any nucleus 0.5 length units per unit time.

9.1.1 Defining the reactor

The following steps have to be followed to define the reactor.

1. Click the “Reactor” button and select the “Main Properties” tab if it isn't already selected.
2. Set the size of the reactor to 100.
3. Set the project folder to the folder where you want to store the output files. You will notice that the name of the reactor is changed automatically.
4. If needed, check “Use relative paths” and set the root folder to the root μ ic folder (see section 2.1.6).
5. We will leave the resource folder and the time step file empty for this example.
6. For the time steps, enter 1 in the Min box, 100 in the Max box and 1 in the Step box. Then press add. If a pop-up message asks you to create a new reactor with the current values, select yes. You should see a list of numbers from 1 to 100 in the box next to time.
7. Select the “Images and Optimisation” tab.
8. If your computer has less memory available, change the field “Pixel size” to 1.0. The default value is usually set to 0.25 and will give images 400x400 pixels in size as the size of the volume is 100 length units on the side.
9. If you want to change the background colour of the images produced, to say white, click on the “Background color” button and select the colour of your choice. Black is used as the default background colour. The other options on this tab do not require modifications for this example.
10. Select the “Packing and Interactions” tab.
11. If you are low on hard-disk space, uncheck “Save particle information in binary” check-box as this will prevent saving of all particle information. The other options of this tab need not be changed for this example.
12. Press the save button.
13. Click on “Main Properties” to go back to the main properties tab.
14. The Reactor panel can be closed if needed. The defined values will not be lost.

9.1.2 Defining the material

This problem requires the definition of only one material. The following steps have to be followed to define the material.

1. Click on “Materials” on the toolbar to open the materials panel.
2. Define the name of the material myMaterial.
3. Leave the Id field blank for this example or change it to 1.
4. Define an arbitrary density for this material, say 1.0.
5. Define an arbitrary value for Diffusivity, say 0.0.
6. Set the initial fraction to 0 as this is the initial amount of the phase of interest of the material present in the system.
7. Leave “Inner” and “Universal” unchecked and check “Discrete”.
8. Skipping the other fields, move directly to the “Color” button. Click on the button and select a colour to use for the material in the images to be produced, say red. To see anything in the images, this colour should be different from the background colour defined in the reactor earlier.
9. Click on the “Add/Save” button to save this material. You will see that “myMaterial” appears in the “Materials Defined” list.
10. The “Materials” panel can be closed if needed. The defined values will not be lost.

9.1.3 Defining the particle model

In this example we have one particle model. This is a single layered particle with myMaterial in the layer. Follow the following steps to create this particle model.

1. Click on “Particle Models” in the toolbar to open the particle models panel.
2. Enter the name of the particle model, say myParticle in this case.
3. You should see myMaterial listed in the “Discrete Materials” list. If not, ensure you followed all the steps section 9.1.2. Select myMaterial in the “Discrete Materials” list and click on the “Add” button below the list.
4. Leave the “PSD File” field empty and click on the “Add/Save” button. You will notice that myParticle is added to the “Particle Models” list.
5. The “Particle Models” panel can be closed if needed. The defined values will not be lost.

9.1.4 Defining the reaction

In this example we have only one reaction, which is the production of myMaterial. Here we are not keeping track of the reactant and assuming it is available in sufficient amounts. So the reaction has no reactant and one product. Follow the following steps to define this reaction.

1. Click on the “Reactions” button to open the “Reactions” panel.
2. Enter the name of the reaction, say myReaction.
3. You should see myMaterial in the “Materials” list. If not make sure you followed all the steps in section 9.1.2. When you select myMaterial in the “Materials” list, the ratio box, the “Add Reactant” and “Add Product” buttons will be activated.
4. Enter 1.0 in the ratio field and click on “Add Product”. You will notice that “myMaterial, 1.0” appears in the “Products” list.
5. Leave the “Discrete” box checked, or check it if it is unchecked.
6. Click on Add/Save. You will notice that myReaction is added in the “Reactions Defined” list. If not, make sure you followed all the steps in sections 9.1.1, 9.1.2 and 9.1.3 and restart from step 1 to define the reaction.
7. The “Reactions” panel can be closed if needed. The defined values will not be lost.

9.1.5 Defining plugins

This problem requires the definition of two plugins. The first plugin to define the rate of nucleation of myParticle and the second the define the rate of growth of the particles.

Follow the steps below to define the rate of nucleation.

1. Click on the “Add Plugin” button in the toolbar to open the “Define plugin” panel.
2. Do not click on the “Browse” button as here we will use a plugin already defined in μic .
3. In the drop-down menu next to “Class type” you should see a list of the various types of plugins available.

4. In this menu select NucleiGenerator. You will notice that the list nest to “Class name” will change, listing the available plugins of “NucleiGenerator” type.
5. From the list select LinearNucleiGenerator and click on Ok to open the “Define nucleiGenerator plugin” panel (see section 7.6).
6. In this panel, make sure that the Name and Type of the plugin is correct.
7. Enter 0 in the “Initial number of nuclei” box.
8. Enter any number larger than 5000 in the “Maximum number of Nuclei” box. This number should be higher than the maximum number of nuclei possible in the time defined for the simulation.
9. Enter 50 in the “Coefficient” box.
10. Enter 0 in the starting time.
11. Enter a number either larger than the end time of the simulation (here 100) or a negative number here.
12. In the drop down menu next to “Model Grain to control” select myParticle. If you do not see this option, make sure that you have followed the steps in section 9.1.3.
13. Click on “Save” to create this plugin.
14. To ensure that this plugin has been properly created, click on the “View Plugins” button in the toolbar to open the “View plugins” panel. You should see the plugin in the “List of currently defined plugins”. You should be able to see the defined values if you select this plugin and click “Show”. The “View plugins” panel can now be closed.

Follow the steps below to define the rate of growth.

1. Click on the “Add Plugin” button in the toolbar to open the “Define plugin” panel.
2. Do not click on the “Browse” button as here we will use a plugin already defined in μic .
3. In the drop-down menu next to “Class type” you should see a list of the various types of plugins available.
4. In this menu select Kinetics. You will notice that the list nest to “Class name” will change, listing the available plugins of “Kinetics” type.
5. In “Class name” select “NucleiGrowthKinetics” (see section 7.4). Click on Ok to open the “Define kinetics plugin” panel.
6. Ensure that the Name and type of the plugin are correct.
7. In “Constant k” enter 0.5, which is the rate of growth of the particles.

8. In “Starting time”, “Initial degree of hydration” and “Minimum radius” enter 0.
9. In the “order of implementation of kinetics” box enter 1.
10. In Reaction, select myReaction. If this option is not available, make sure you have followed the steps in section 9.1.4 correctly.
11. In reactant, first select myReaction in the list on the left. You will notice that “myReaction myMaterial 1.0” will appear in the box on right. Select “myReaction myMaterial 1.0”.
12. In Material select myMaterial.
13. Click on Save to create the plugin.
14. To ensure that this plugin has been properly created, click on the “View Plugins” button in the toolbar to open the “View plugins” panel and click on the “Update” button. You should see the plugin in the “List of currently defined plugins”. You should be able to see the defined values if you select this plugin and click “Show”. The “View plugins” panel can now be closed.

Definition of the problem is now complete.

9.1.6 Creating the input file

The problem has to be saved before it can be run. Click on File -> Save or the save button in the toolbar to open the “Choose the XML file” dialog box”. Go to the folder where you want to save your XML file, enter the File name, say Nucleation.xml and click on save. The input file should now have been created. See section 5.1.2 to run this file.

9.2 Hydration of alite

In this example, we simulate the hydration of alite to produce C-S-H and CH. The main points of this problem are listed below.

1. Alite is the only material present in the initial system and it occupies 44.25% of the initial volume (equivalent to 0.4 w/c for an alite with 3.15 density).
2. The particle size distribution of the alite particles is defined in an external text file.
3. The remaining initial volume is filled with water.

4. The only two reaction products are C-S-H and CH.
5. C-S-H has a density of 2.0 and CH of 2.24
6. For every unit volume of alite reacting, 1.57 unit volume of C-S-H is formed and 0.593 unit volumes of CH are formed and 1.32 unit volumes of water are consumed.
7. The C-S-H formed grows on the surface of the same particle where the alite reacted.
8. The CH forms new nuclei in the pore that increase linearly from 0 to 10,000 in 10 hours.
9. The CH formed is randomly distributed to all the CH particles.
10. The water is assumed to be uniformly distributed throughout the empty space and is assumed to not control the hydration kinetics.
11. The kinetics of the acceleration part is assumed to be governed by the Avrami equation, with $k = 0.0001$ and $n = 3$ (see section 7.4.2) and the deceleration by the resistance to flow through the hydrates, with the diffusivity of C-S-H being 0.01 and other materials being zero (see section 7.4.5). Although these kinetics models have been shown to not to be necessarily correct, we will use these relations just as an example.
12. The time steps at which the simulation is to be carried out is contained in a text file.

9.2.1 Defining the reactor

The following steps have to be followed to define the reactor.

1. Click the “Reactor” button and select the “Main Properties” tab if it isn't already selected.
2. Set the size of the reactor to 100.
3. Set the project folder to the folder where you want to store the output files. You will notice that the name of the reactor is changed automatically.
4. If needed, check “Use relative paths” and set the root folder to the root μic folder (see section 2.1.6).
5. We will leave the resource folder empty for this example.
6. Click on the “Time step file” button and select the file containing the list of times at which the calculation is to be carried out (see section 2.1.5). You will notice that the folder in which the file is contained is

automatically added to the “Resource Folders” list and the name of the file appears in the “Time step file” box.

7. You can also enter the time steps manually by either entering individual times in the “Min” box and clicking “Add” or by entering one or multiple sets of the minimum time, maximum time and step and then clicking “Add”.
8. Select the “Images and Optimisation” tab.
9. If your computer has less memory available, change the field “Pixel size” to 1.0. The default value is usually set to 0.25 and will give images 400x400 pixels in size as the size of the volume is 100 length units on the side.
10. If you want to change the background colour of the images produced, to say white, click on the “Background color” button and select the colour of your choice. Black is used as the default background colour. The other options on this tab do not require modifications for this example.
11. Select the “Packing and Interactions” tab.
12. If you are low on hard-disk space, uncheck “Save particle information in binary” check-box as this will prevent saving of all particle information. The other options of this tab need not be changed for this example.
13. Press the “Save” button.
14. Click on “Main Properties” to go back to the main properties tab.
15. The Reactor panel can be closed if needed. The defined values will not be lost.

9.2.2 Defining the materials

This problem requires the definition of four materials: Alite, Water, C-S-H and CH. The following steps have to be followed to define Alite.

1. Click on “Materials” on the toolbar to open the materials panel.
2. Define the name of the material Alite.
3. Change the Id field to 1. (If the order in which the materials are listed is not important, this field can be left blank for all materials).
4. Define the density for this material, say 3.15.
5. Define the value for Diffusivity, say 0.0.
6. Set the initial fraction to 0.4425 as this is the initial amount of alite present in the system.

7. Leave “Inner” and “Universal” unchecked and check “Discrete”.
8. Skipping the other fields, move directly to the “Color” button. Click on the button and select a colour to use for the material in the images to be produced, say red. To see anything in the images, this colour should be different from the background colour defined in the reactor earlier.
9. Click on the “Add/Save” button to save this material. You will see that “Alite” appears in the “Materials Defined” list.

The steps to define “Water” are only slightly different from that of Alite:

1. Define the name of the material Water.
2. Set the Id to 2.
3. Define the density for this material, say 1.00.
4. Set the value for Diffusivity to any arbitrary value, say 0.0.
5. Set the initial fraction to 0.5575 as this is the remaining amount of space. If this is the last material being defined, which has a non-zero initial amount, the word remaining can be entered in the Initial fraction and the programme will calculate the space remaining for this material. If another material with a non-zero initial amount has to be defined, the initial amount of this material will have to be redefined.
6. Leave “Inner” unchecked and check “Discrete” and “Universal”. The water is defined to be universal as it will not be allocated space in any of the particles.
7. We do not need to define a colour for water, as it will not be represented in particles.
8. Click on the “Add/Save” button to save this material. You will see that “Water” appears in the “Materials Defined” list.

The steps to define C-S-H and CH are also similar to that of the other materials. Follow the following steps first for C-S-H and then for CH:

1. Define the name of the material C-S-H or CH.
2. Change the Id field to 3 for C-S-H and 4 for CH.
3. Define the density for this material, say 2.0 for C-S-H and 2.24 for CH.
4. Define the value for Diffusivity, say 0.01 for C-S-H and 0.0 for CH.
5. Set the initial fraction to 0. as as no C-S-H or CH is present in the beginning.
6. Leave “Inner” and “Universal” unchecked and check “Discrete”.

7. Skipping the other fields, move directly to the “Color” button. Click on the button and select a colour to use for the material in the images to be produced, say blue for C-S-H and green for CH. To see anything in the images, this colour should be different from the background colour defined in the reactor earlier.
8. Click on the “Add/Save” button to save this material. You will see that the material name appears in the “Materials Defined” list.

The “Materials” panel can now be closed if needed without the loss of any data.

9.2.3 Defining the particle models

In this example we have two particle models. The first is the alite particle with an inner layer of Alite followed by an outer layer of C-S-H. The second is the CH particle with a layer of CH. Follow the following steps to create the two particle models.

1. Click on “Particle Models” in the toolbar to open the particle models panel.
2. Enter the name of the alite particle model, say AliteParticle in this case.
3. You should see Alite, C-S-H and CH listed in the “Discrete Materials” list. If not, ensure you followed all the steps section 9.1.2. Select Alite in the “Discrete Materials” list and click on the “Add” button below the list, then similarly add C-S-H. You should see Alite followed by C-S-H appear in the “Layers” list. If they are not in the correct order, the arrow buttons next to the list can be used to move or delete the materials.
4. Click on the button next to “PSD File” and select the file that contains the particle size distribution for alite particles (see section 2.3.3). You will notice that the name of the file appears in the text box and if you now open the Reactor panel, you will notice that the folder in which the file is contained is added to the “Resource Folder” list. If not, add the folder manually.
5. Click on the “Add/Save” button. You will notice that AliteParticle is added to the “Particle Models” list.
6. Now enter the name of the CH particle model, say CHParticle.
7. Delete the materials that may be present in the Layers list.
8. Select CH in the “Discrete Materials” list and click on Add.

9. Clear any text in the PSD File box, since we do not need to define the initial particle size distribution of the CH particles.
10. Click on the “Add/Save” button. You will notice that CHParticle is added to the “Particle Models” list.
11. The “Particle Models” panel can be closed if needed. The defined values will not be lost.

9.2.4 Defining the reaction

In this example we have only one reaction, which is the reaction of Alite with Water to produce C-S-H and CH. Follow the following steps to define this reaction.

1. Click on the “Reactions” button to open the “Reactions” panel.
2. Enter the name of the reaction, say AliteHydration.
3. You should see Alite, Water, C-S-H and CH in the “Materials” list. If not make sure you followed all the steps in section 9.1.2. When you select Alite in the “Materials” list, the ratio box, the “Add Reactant” and “Add Product” buttons will be activated.
4. Enter 1.0 in the ratio field and click on “Add Reactant”. You will notice that “Alite, 1.0” appears in the “Reactants” list.
5. Similarly, select Water, enter 1.32 in Ratio and press “Add Reactant”.
6. Now select C-S-H, enter 1.57 in Ratio and press “Add Product”. You will notice that C-S-H, 1.57 is added to the Products list.
7. Similarly select CH, enter 0.593 in ratio and press “Add Product”.
8. Leave the “Discrete” box checked, or check it if it is unchecked.
9. Click on Add/Save. You will notice that AliteHydration is added in the “Reactions Defined” list. If not, make sure you followed all the steps in sections 9.1.1, 9.1.2 and 9.1.3 and restart from step 1 to define the reaction.
10. The “Reactions” panel can be closed if needed. The defined values will not be lost.

9.2.5 Defining plugins

This problem requires the definition of three plugins. The first plugin to define the rate of nucleation of CHParticle and the remaining two plugins define the kinetics of the reaction.

Follow the steps below to define the rate of nucleation.

1. Click on the “Add Plugin” button in the toolbar to open the “Define plugin” panel.
2. Do not click on the “Browse” button as here we will use a plugin already defined in `µic`.
3. In the drop-down menu next to “Class type” you should see a list of the various types of plugins available.
4. In this menu select `NucleiGenerator`. You will notice that the list next to “Class name” will change, listing the available plugins of “`NucleiGenerator`” type.
5. From the list select `LinearNucleiGenerator` and click on Ok to open the “Define nucleiGenerator plugin” panel (see section 7.6).
6. In this panel, make sure that the Name and Type of the plugin is correct.
7. Enter 0 in the “Initial number of nuclei” box.
8. Enter 10000 in the “Maximum number of Nuclei” box.
9. Enter 1000 in the “Coefficient” box.
10. Enter 0 in the starting time.
11. Enter a number either larger than the end time of the nucleation (10 here) or a negative number here.
12. In the drop down menu next to “Model Grain to control” select `CHParticle`. If you do not see this option, make sure that you have followed the steps in section 9.1.3.
13. Click on “Save” to create this plugin.
14. To ensure that this plugin has been properly created, click on the “View Plugins” button in the toolbar to open the “View plugins” panel. You should see the plugin in the “List of currently defined plugins”. You should be able to see the defined values if you select this plugin and click “Show”. The “View plugins” panel can now be closed.

To define the hydration kinetics, we will use two plugins: `AvramiKineticsNoReduction` and `DiffusionProportionateKinetics` (see sections 7.4.1 and 7.4.5 respectively). At all steps in the simulation, the reactions will use the lower of the two value to calculate the rate of reaction for each particle.

Follow the steps below to define the Avrami kinetics plugin.

1. Click on the “Add Plugin” button in the toolbar to open the “Define plugin” panel.

2. Do not click on the “Browse” button as here we will use a plugin already defined in μic .
3. Select Kinetics in the drop-down menu next to “Class type”. You will notice that the list next to “Class name” will change, listing the available plugins of “Kinetics” type.
4. In “Class name” select “AvramiKineticsNoReduction”. Click on Ok to open the “Define kinetics plugin” panel.
5. Ensure that the Name and type of the plugin are correct.
6. In “Avrami constant k” enter 0.0001 and in “Avrami constant n” enter 3.
7. In “Starting time” and “Initial degree of hydration” enter 0.
8. In the “order of implementation of kinetics” box enter 0.
9. In Reaction, select AliteHydration. If this option is not available, make sure you have followed the steps in section 9.1.4 correctly.
10. In reactant, first select AliteHydration in the list on the left. Select “AliteHydration Alite 1.0” in the box on right.
11. Click on Save to create the plugin.
12. To ensure that this plugin has been properly created, click on the “View Plugins” button in the toolbar to open the “View plugins” panel and click on the “Update” button. You should see the plugin in the “List of currently defined plugins”. You should be able to see the defined values if you select this plugin and click “Show”. The “View plugins” panel can now be closed.

Follow the steps below to define the diffusion kinetics.

1. Click on the “Add Plugin” button in the toolbar to open the “Define plugin” panel.
2. Do not click on the “Browse” button as here we will use a plugin already defined in μic .
3. Select Kinetics in the drop-down menu next to “Class type”. You will notice that the list next to “Class name” will change, listing the available plugins of “Kinetics” type.
4. In “Class name” select “DiffusionProportionateKinetics”. Click on Ok to open the “Define kinetics plugin” panel.
5. Ensure that the Name and type of the plugin are correct.
6. In the “order of implementation of kinetics” box enter 1.
7. In Factor enter 1.0. This is the factor with which to multiply the diffusivities of all materials.

8. In Reaction, select AliteHydration.
9. In reactant, first select AliteHydration in the list on the left. Select “AliteHydration Alite 1.0” in the box on right.
10. Click on Save to create the plugin.
11. To ensure that this plugin has been properly created, click on the “View Plugins” button in the toolbar to open the “View plugins” panel and click on the “Update” button. You should see the plugin in the “List of currently defined plugins”. You should be able to see the defined values if you select this plugin and click “Show”. The “View plugins” panel can now be closed.

Definition of the problem is now complete.

9.2.6 Creating the input file

The problem has to be saved before it can be run. Click on File -> Save or the save button in the toolbar to open the “Choose the XML file” dialog box”. Go to the folder where you want to save your XML file, enter the File name, say Alite.xml and click on save. The input file should now have been created. See section 5.1.2 to run this file.

9.3 Adding inert filler particles to alite hydration

In this problem, in addition to the problem discussed in section 9.2, we will add particles of an inert material, which will allow the growth of C-S-H on its surface. The conditions for this simulation, in addition to those in section 9.2 are listed below.

1. The fillers replace 10% of the volume of alite and have the same density as alite.
2. The particle size distribution of the fillers is defined in an external text file.
3. The C-S-H produced in the hydration is divided into three parts. The first part called the InnerC-S-H fills the space created by the consumption of alite. The second and the third part are deposited on the alite and filler particles, in proportion of the free surface area of the individual particles.

The following subsections list the steps to be followed to define the above problem.

9.3.1 Defining the reactor

To define this problem, we will first open the file saved at the end of section 9.2. Loading this file should automatically load all the required conditions for the reactor. Change the name of the Project folder and click on Save.

9.3.2 Defining the materials

Follow the steps below to update the definitions of the materials.

1. Open the “Materials” panel.
2. Select the material C-S-H and press the “Load” button. The properties of the C-S-H material should be loaded to the panel. We will change the name of this material to InnerC-S-H to represent the material that will replace the alite dissolved.
3. Change the name to InnerC-S-H.
4. Check “Inner”.
5. Press “Add/Save”. Press yes when you are prompted to confirm the change.
6. Without changing any other property in the panel, change the name of the material to AliteOuterC-S-H. This is the material that will deposit on the surface of the alite particles.
7. Change the Id to 5.
8. Change the colour to another value, say cyan and press “Add/Save”.
9. Now without changing any other property, change the name of the material to FillerOuterC-S-H. This is the material that will deposit on the surface of the filler particles.
10. Change the Id to 6.
11. Change the colour to any other colour, say violet and press “Add/Save”.
12. Now change the name to OuterC-S-H. This is the material that will be temporarily used to store all the C-S-H that is not placed as InnerC-S-H. This material will only be produced temporarily until it has been calculated how to distribute the outer C-S-H to the filler and the alite particles.
13. Change the Id to 7.
14. Now check Universal, as this material will not be present in particles and press “Add/Save”.
15. Now, select Alite and press “Load”.

16. Change the initial amount to 0.39825 and press Save and confirm changing the material defined earlier. This is 10% less than the earlier defined amount, which is the volume that will be replaced by the filler.
17. Change the name of the material to Filler and the Id to 8.
18. Change the initial amount to 0.04425.
19. Change the Color to say yellow and press “Add/Save”.
20. The materials panel can now be closed without losing any information.

9.3.3 Defining the particle models

Follow the steps below to complete the definition of the particle models.

1. Click on “Particle Models” to open the “Particle Models” panel.
2. Enter FillerParticle in the Name.
3. Select Filler in the Discrete Materials list and click Add.
4. Select FillerOuterC-S-H and click Add.
5. Now click on the button next to the PSD file and select the file with the particle size distribution of the filler. You will notice that the name of the file is added to the PSD file box and the folder is added to the resource folder list in the Reactor panel.
6. Click “Add/Save” to save this model.
7. Select AliteParticle in the Particle Models list and press load.
8. Select AliteOuterC-S-H in the Discrete Materials list and click on Add.
9. Click “Add/Save” and confirm to replace the older definition of the particle model.
10. The Particle Models panel can now be closed.

9.3.4 Defining reactions

Follow the steps below to complete the definition of the reactions.

1. Select AliteHydration in the Reactions Defined list and click Load.
2. Select InnerC-S-H in the Materials list, set the ratio to 1.0 and click Add Product. You will notice that InnerC-S-H, 1.57 is replaced by InnerC-S-H, 1.0 in the Products list.
3. The remaining 0.57 volume of the product will be allocated to OuterC-S-H. Select OuterC-S-H in the Materials list, enter 0.57 in ratio and press “Add Product”.

4. Press “Add/Save” and confirm replacing the earlier definition.
5. Now Change the name of the reaction to OuterC-S-H distribution.
6. Delete all the Reactants and Products.
7. Select OuterC-S-H, enter 1.0 in ratio and press “Add Reactant”.
8. Select AliteOuterC-S-H, enter 0.5 in ratio and press “Add Product”.
9. Select FillerOuterC-S-H, enter 0.5 in ratio and press “Add Product”.
10. Uncheck “Discrete” and click “Add/Save”.
11. The “Reactions” panel can now be closed.

9.3.5 Defining plugins

We will now define plugins to do the following: First to rebalance the stoichiometry of the reaction OuterC-S-H distribution according to the surface area available on the alite particles and filler particles, we will use the SurfaceRebalancer plugin (see section 7.12.2). Secondly to distribute the FillerOuterC-S-H and AliteOuterC-S-H materials to the particles based on their free surface area we will use the plugin SurfaceDistributionProfile (see section 7.8.1). A third plugin ReactAllKinetics (see section 7.4.12) is required to convert all OuterC-S-H to AliteOuterC-S-H and FillerOuterC-S-H.

Follow the steps below to create the SurfaceRebalancer plugin:

1. Click on “Add Plugin” to open the “Define Plugin” panel.
2. Select Recalculator in Class type and SurfaceRebalancer in Class name and click Ok.
3. Select OuterC-S-HDistribution in Reaction and 0.0 in minimum radius.
4. Next to Array of Products, enter 2 in number of elements and click Create to open the new panel.
5. Select OuterC-S-HDistribution in the right drop down list in both the Product boxes. Then select the first product in the right drop down menu of the first Product box and the second product in the second Product box.
6. Click on Save to close the Array panel.
7. Click on Save again to save and close the “Define recalculator plugin” panel.

Now to define the SurfaceDistributionProfile plugin do the following.

1. Click on “Add Plugin” again to open the “Define plugin” panel.

2. Select `MaterialDistributionProfile` in the Class type and `SurfaceDistributionProfile` in the Class name and press Ok to open the “Define materialDistributionProfile plugin” panel.
3. Select `FillerOuterC-S-H` in Material and fill 0.0 in Minimum radius.
4. Click on “Save” to save the plugin and close the panel.

Now repeat the above steps for `AliteOuterC-S-H` material.

Follow the steps below to create the `ReactAllKinetics` plugin:

1. Click on “Add Plugin” to open the “Define Plugin” panel.
2. Select Kinetics in Class type and `ReactAllKinetics` in Class name and click Ok.
3. Enter 1 in order of implementation of reaction.
4. Select `OuterC-S-HDistribution` in Reaction.
5. To select the reactant, select `OuterC-S-HDistribution` in the reaction box next to Reactant and then select `OuterC-S-H` in reactant.
6. Click on Save to save and close the “Define kinetics plugin” panel.

9.3.6 Creating the input file

The problem has to be saved before it can be run. Click on File -> Save or the save button in the toolbar to open the “Choose the XML file” dialog box”. Go to the folder where you want to save your XML file, enter the File name, say `Filler.xml` and click on save. The input file should now have been created. See section 5.1.2 to run this file.

9.4 Adding belite to hydration with alite and filler particles

In this setup, we will add the material Belite to the problem defined in section 9.3. The main features of the problem are listed below.

1. Belite is mixed with Alite, but only in particles larger than 20 unit lengths in size.
2. The initial amount of Belite is 9.825% of the volume and that of Alite is 30% of the volume.
3. One unit volume of Belite hydrates with 1.48 unit volumes of water to produce one unit volume of `InnerC-S-H`, 1.17 unit volumes of `OuterC-S-H` and 0.189 unit volumes of CH.
4. The hydration of Belite starts 24 time units after the start.

5. The kinetics of Belite hydration also follow the Avrami law (see section 7.4.2).

9.4.1 Defining the reactor

Open the file saved in the example with the filler (section 9.3). If you haven't created this file yet, please follow the procedure in section 9.3 to create this file. Once this file has been opened, change the name of the Project folder as needed and click on Save.

9.4.2 Defining the materials

To add the Belite material to the system, follow the steps below.

1. Change the name to Belite and Id to 9.
2. Change the density to 3.28.
3. Change the diffusivity to 0.0.
4. Change the initial fraction to 0.09825.
5. Leave Inner and Universal unchecked.
6. Also uncheck Discrete; you will notice that the Host Material drop down list is activated. Select Alite in the Host Material list and click Add/Save.
7. Select Alite in the Materials Defined list and click Load.
8. Change the initial fraction to 0.30 and click Add/Save and confirm the change.
9. The materials panel can now be closed.

9.4.3 Defining the particles

No changes to the particles has to be made as Belite is already defined to have Alite as the “Host Material”.

9.4.4 Defining the reactions

To add the reaction for the hydration of Belite, follow the steps below.

1. Click on the Reactions button to open the Reactions panel.
2. Change the name of the reaction to BeliteHydration.
3. Select Belite in the Materials list, enter 1.0 in ratio and press “Add Reactant”.

4. Select Water in the Materials list, enter 1.47 in ratio and press “Add Reactant”.
5. Select InnerC-S-H, enter 1.0 in ratio and press “Add Product”.
6. Select OuterC-S-H, enter 1.17 in ratio and press “Add Product”.
7. Select CH, enter 0.189 in ratio and press “Add Product”.
8. Press Add/Save to save the reaction.
9. Now select BeliteHydration in the “Reactions Defined” list and press the up arrow on its right to move the reaction BeliteHydration above OuterC-S-HDistribution. This has to be done so that BeliteHydration is executed before OuterC-S-HDistribution and the OuterC-S-H produced from BeliteHydration is also distributed in the OuterC-S-HDistribution reaction.

9.4.5 Defining plugins

For this problem we will define two more plugins. The first to limit the distribution of Belite only to larger particles and the second to define the kinetics of BeliteHydration.

To define the first plugin follow the steps below.

1. Click on “Add Plugin” and select GrainProportionInitialiser in Class type and LimitedRandomGrainProportionInitialiser in Class name. Click Ok to open the Define grainProportionInitialiser plugin panel.
2. Set ModelGrain to AliteParticle.
3. Enter 1 in all three fields with Number of elements.
4. Press the first create next to Material to limit, select Belite in the panel that appears and click Save.
5. Click Create in the button next to “Lower size limit”, enter 20 in the panel that appears and click Save.
6. Click Create in the button next to “Upper size limit”, enter any number larger than the largest particle size, or a negative number and click Save.
7. Click Save on the “Define grainProportionInitialiser plugin” panel to save the plugin.

To define the kinetics plugin, follow the steps below.

1. Alick “Add Plugin” and select kinetics in Class type and AvramiKineticsNoReduction in Class name and click Ok.

2. In Avrami constant k enter 1.0e-5, and 3 in Avrami constant n.
3. Enter 24 in starting time.
4. Enter 0 in Initial Degree of Hydration and 1 in order of kinetics.
5. Select BeliteHydration in reaction.
6. For the reactant, first select BeliteHydration in the list on the left, then select BeliteHydration Belite 1.0 in the list on the right.
7. Click on Save to create the plugin.

The plugins created can be seen in the list clicking “View Plugins”.

9.4.6 Creating the input file

The problem has to be saved before it can be run. Click on File -> Save or the save button in the toolbar to open the “Choose the XML file” dialog box”. Go to the folder where you want to save your XML file, enter the File name, say Belite.xml and click on save. The input file should now have been created. See section 5.1.2 to run this file.

Chapter 10: MicWizard

10.1 Introduction

A new tool called MicWizard has been developed to simplify the task of modelling microstructural development using μic . Instead of using a single input file for the simulations, this tool allows the user to split the definitions into several individual files. The first type of file is called the library file and is similar to a normal μic input file. It defines all possible materials, particle models, reactions and plugins for the system. The second type of files, called the powder files, define the composition of the powders to be used in the simulations. The third type of file, called the mix-design file, defines all the parameters to be used in the simulation and the powders present with their amounts. More details about each file type are listed below.

10.2 Input files

10.2.1 Library file

This file contains definitions of all possible materials, particles, reactions and plugins needed by a user. A default version of this library file is distributed with MicWizard for the simulation of Portland cements. This file follows the format of a μic XML input file and defines all materials, particle models, the reactor, the reactions, the plugins and the commands. It can be created using the traditional interface of μic . All materials that can be present in the initial system have their initial fraction defined to be 1.0. Any number more than 0 will work, however using 1.0 will ensure that this file will not be mistakenly used directly for a simulation.

The material class for all solution materials, such as water in the case of cement, must be defined as 'solution' (case independent).

The interface for MicWizard loads the distributed library file by default. However, the user can choose another library file by clicking on the “Choose library file” button in the toolbar. Choosing a new library file deletes all active data from the interface.

10.2.2 Powder file

This file lists the name, composition and particle size distribution of the powder. This file can be created by clicking on the “Create powder” button of the interface. The interface then looks for all particle types in the library file which contain the reacting initial materials and lists them for the user to choose the type of the powder to be defined.

Once the powder type is selected, a new window is opened to define the powder composition. The user must define the name of the powder, the mass fraction and the density of each phase listed in this window. If a phase is not present in the powder to be defined, its mass fraction can be defined to be zero. The default density of each phase is listed from the library file and the figures may be changed if needed. Along with the materials present in the selected particle type, the list may also contain the universal materials defined in the library file. The amounts of the universal materials are added together if they are defined in multiple powders.

The user can then select the particle size distribution file for the powder by clicking on the browse button. See section 2.3.3 on page 17 for more details about this file. By clicking on 'save in file' button, the user can save the definition of the powder to an existing or a new powder file. If the file already exists, the current definition will be appended to the end of the XML structure. The programme will not check for any other powder definitions in the same file with the same name. The user must ensure that no other powder with the same name is already present in the file as at the time of the simulation, the programme will use the first definition found in the file.

10.2.3 Mix design file

This file lists the system parameters, the powders to be used in the simulation and their respective quantities. To create this file, the user should click on the 'Create new mix' button in the toolbar. A window with two tabs will appear. The first tab can be used to define the system parameters while the second tab can be used to define the powders to be used in the simulation.

In the first tab the user must choose the Project folder, where all output files from the simulations will be stored, the size of the reactor, the file containing the time steps of the simulation (see section 2.1.5 on page 7) and the pixel size. The user must also choose if relative paths are to be used in the input files and if so the root folder for the paths. The user can also choose to store the binary information for all particles required to resume simulations, or for further analysis of the microstructures, and the ASCII files required for pore-analysis provided in `mic` (See section 5.2.2 on page 46).

In the first tab the user can also choose the material to be used in the solution and its density. A list of materials with their class defined as 'solution' are listed in this panel with its default density defined in the library file in the box next to the list.

In the second tab, the user can select a powder to be associated with each of the available particle types. When a powder file is selected, the list of powders available in the file is updated. The default particle size distribution file is also selected and shown in the 'psd file' box. The user can change the particle size distribution for the simulation. The user must then select the 'Powder type' from 'main', 'replacement' and 'addition'. The main powder is used to calculate the mass of the powders and the solution in the system. The given solution-powder ratio defines the ratio of the initial masses of the solution and the powder, before any part of the powder is replaced by a replacement or any addition is added. This value is used to calculate the volume fraction of all phases in the main powder and the volume fraction of the solution material. For all simulations at least one main powder has to be defined.

Any subsequent powder defined as a replacement will replace the given fraction of the mass of the main powder. It must be noted that if the density of the main and the replacement powders is not the same, the volume fraction of the solution and that of all the powders put together will change.

The mass of a powder defined is calculated the same way as a replacement powder, however, it replaces the solution material instead of the main powder.

The interface is currently limited to using only one powder for each particle type. While the user can choose to add multiple powders for each particle type manually in the XML file, it is recommended that the produced results be checked thoroughly to ensure all reactions have been represented in the created system. It is possible that in certain cases some possible reactions are ignored by the wizard or that some extra reactions have been created due to the availability of the same material in several powders.

Once all powders have been chosen, the user can click on the 'Save input file' button to save the defined system in an XML file. Only the powders for which the defined fraction is more than 0 will be saved.

10.3 Running a mix design XML file

The created mix design file can be run in a way similar to the traditional mic input file using the MicWizard.jar executable file. The command for running input files is:

```
1  java -Xmx2000m -jar MicWizard.jar xmlfile1.xml xmlfile2.xml  
    xmlfile3.xml
```

The phrase -Xmx2000m reserves a maximum of 2000 megabytes of memory in the system for the execution and is a native java option. All the XML input files listed in the command will be executed sequentially.

10.4 Running the MicWizard graphical user interface

The user interface for MicWizard can be run using the following command:

```
1  java -jar MicWizard.jar
```

References

1. Tegmark M., *An icosahedron-based method for pixelizing the celestial sphere*, The Astrophysical Journal, Vol. 470, 1996, pp.L81-L84
2. Weisstein E.W., *Sphere point picking*, from MathWorld--A Wolfram web resource <http://mathworld.wolfram.com/SpherePointPicking.html>
3. Avrami M., *Kinetics of phase change. I*, Journal of Chemical Physics, Vol. 7, 1939, pp.1103-1112
4. Bishnoi S., *Vector modelling of hydrating cement microstructure and kinetics*, Doctoral Thesis, EPFL, Switzerland, 2008
5. Bishnoi S. and Scrivener K.L., *μic : A new platform for modelling the hydration of cements*, Cement and Concrete Research, Vol. 39, 2009, pp.266-274