A software to simulate and fit neutron and x-ray scattering at grazing incidence Some software

1 Introduction

2 Installation

This is how to install

3 Data classes for simulations and fits

This section will give an overview of the classes that are used to describe all the data needed to perform a single simulation. The prime elements of this data are formed by the sample, the experimental conditions (beam and detector parameters) and simulation parameters.

These classes constitute the main interface to the software's users, since they will mostly be interacting with the program by creating samples and running simulations with specific parameters. Since it is not the intent to explain internals of classes in this document, the text and figures will only mention the most important methods and fields of the classes discussed. Furthermore, getters and setters of private member fields will not be indicated, although these do belong to the public interface. For more detailed information about the project's classes, their methods and fields, the reader is referred to the source code documentation. REF?

3.1 The Experiment object

The Experiment class holds all references to data objects that are needed to perform a simulation. These consist of a sample description, possibly implemented by a builder object, detector and beam parameters and finally, a simulation parameter class that defines the different approximations that can be used during a simulation. Besides getters and setters for these fields, the class also contains a runSimulation() method that will generate an ISimulation object that will perform the actual computations. The class diagram for Experiment is shown in figure **??**.

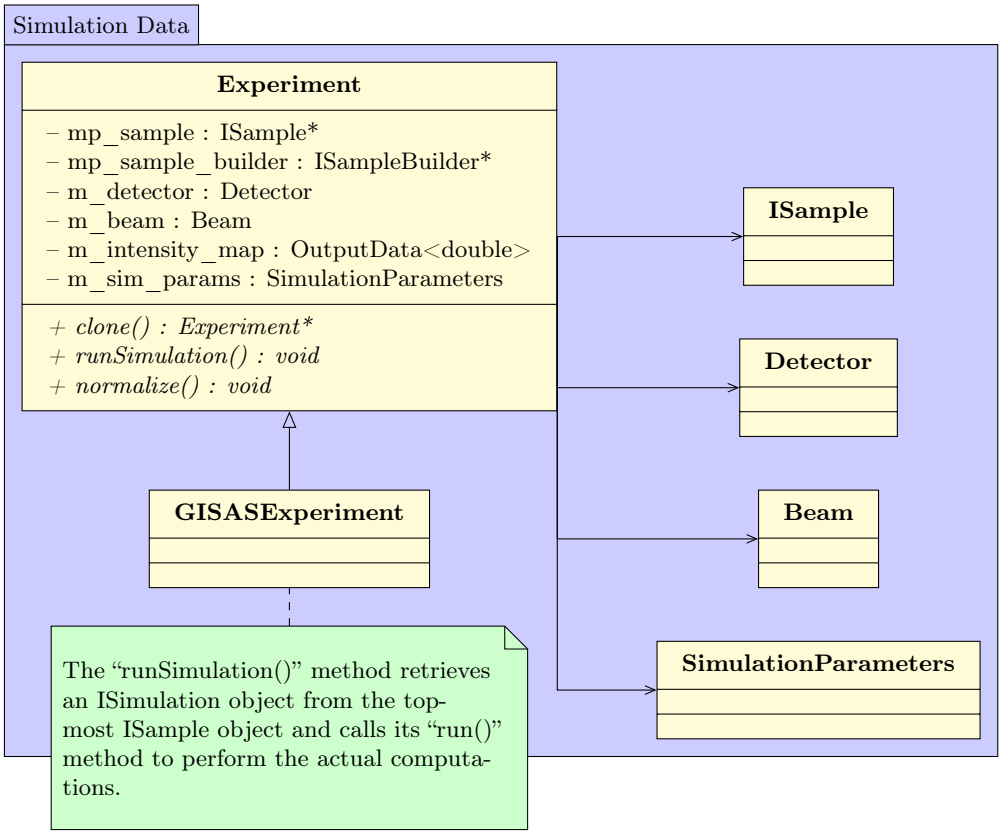

Figure 1: The Experiment class as a container for sample, beam, detector and simulation parameters.

3.2 The ISample class hiërarchy

Samples are described by a hiërarchy tree of objects which all adhere to the ISample interface. The composite pattern is used to achieve a common interface for all objects in the sample tree. The sample description is maximally decoupled from all computational classes, with the exception of the "createDWBASimulation()" method. This method will create a new object of type "DWBASimulation" that is capable of calculating the scattering contributions originating from the sample in question. The coupling is however not very tight, since the ISample subclasses only need to know about which class to instantiate and return.

This interface and two of its subclasses are sketched in figure **??**.

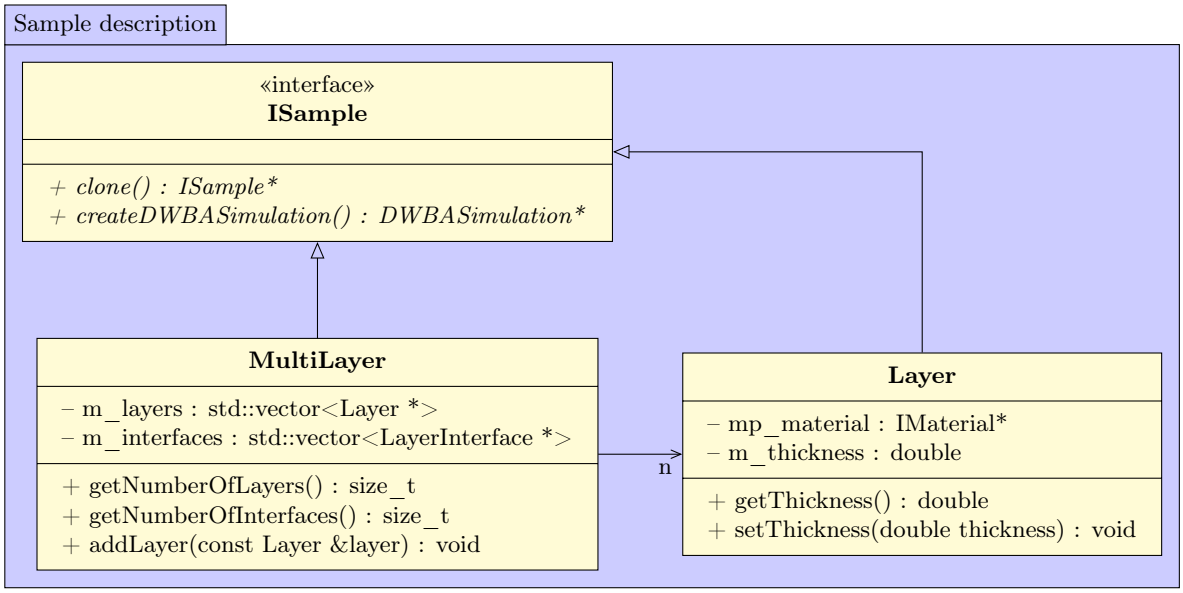

Figure 2: The ISample interface

4 Fitting

5 Examples

5.1 Hello listing

```
mAmbience = MaterialManager.getHomogeneousMaterial("Air", 1.0, 0.0 )
mSubstrate = MaterialManager.getHomogeneousMaterial("Substrate", 1.0-6e-6, 2e-8 )
n_particle = complex(1.0-6e-4, 2e-8)
cylinder_ff = FormFactorCylinder(5*nanometer, 5*nanometer)
cylinder = Particle(n_particle, cylinder_ff)
prism_ff = FormFactorPrism3(5*nanometer, 5*nanometer)
prism = Particle(n_particle, prism_ff)
particle_decoration = ParticleDecoration()
particle_decoration.addParticle(cylinder, 0.0, 0.5)
particle_decoration.addParticle(prism, 0.0, 0.5)
interference = InterferenceFunctionNone()
particle_decoration.addInterferenceFunction(interference)
# air layer with particles and substrate form multi layer
air_layer = Layer(mAmbience)
air_layer_decorator = LayerDecorator(air_layer, particle_decoration)
substrate_layer = Layer(mSubstrate, 0)
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer_decorator)
multi_layer.addLayer(substrate_layer)

# build and run experiment
simulation = Simulation()
simulation.setDetectorParameters(100,-1.0*degree, 1.0*degree, 100, 0.0*degree 2.0*degree, True)
simulation.setBeamParameters(1.0*angstrom, -0.2*degree, 0.0*degree)
simulation.setSample(multi_layer)
simulation.runSimulation()
```