BornAgain: a versatile framework for modelling and fitting GISAS

Céline Durniak, Walter Van Herck, Gennady Pospelov, Joachim Wuttke

*Scientific Computing Group at MLZ*
*Jülich Center for Neutron Science*

JCNS Koordinierungstreffen, October 28-29, 2013, Jülich
Scientific Computing group at MLZ

Organization
- A joint group of HGF and TUM

Group members
- Joachim Wuttke, nominated July 2011
- Walter Van Herck, since Januar 2012
- Gennady Pospelov, since Januar 2012
- Céline Durniak, since Januar 2013
- Antti Soininen, since June 2013
- Marina Ganeva, since September 2013

Scope of responsibility
- Support instruments for questions at the interface of physics and IT
- Enhance understanding of experimental results by providing simulations
Scientific Computing group at MLZ

Current projects

- BornAgain: GISAS simulation and analysis
  *Céline Durniak, Walter Van Herck, Gennady Pospelov, Joachim Wuttke*

- Multiple scattering simulation
  *Antti Soininen, Joachim Wuttke*

- Raw data reduction for SPHERES and TOFTOF
  *Céline Durniak, Marina Ganeva, Joachim Wuttke*

- Data catalogue
  *Joachim Wuttke*
BornAgain project

- Motivation
- Architecture
- Functionality
- Graphical User Interface prototype
- Infrastructure for code development
- Summary
**Motivation**

- Architecture
- Functionality
- Graphical User Interface prototype
- Infrastructure for code development
- Summary
Motivation

Heinz Maier-Leibnitz Zentrum (MLZ)  
FRM-II neutron source in Garching bei Munich

HDRI  
High Data Rate Processing and Analysis Initiative

From local (MLZ) and more broader needs (HDRI)

- Start community project for data analysis and simulation
- Grazing incidence small angle scattering as a first task
  - different user experiments (neutron and x-rays)
  - more generally structured than existing software
  - IsGISAXS as reference software
- Apply acquired experience to other experiments
Grazing Incidence Small Angle Scattering

- beam is directed on a surface with a very small incident angle
- deviation from perfectly smooth layered system leads to beam scattered in an out-of-specular direction
- 2D detector records the intensity of scattered wave giving access to lateral and vertical sample structure information
Distorted Wave Born Approximation (DWBA)

- accounts for reflection-refraction effects close to the critical angle
- interference between scattering from direct and reflected beam

\[ \frac{d\sigma}{d\Omega} = \left| F_{\text{DWBA}} \right|^2 S(q_i) \]

Scalar case

\[
\mathcal{F}_{\text{DWBA}}(k_i, k_f, R_z) \equiv T_i T_f F^{ii}(k_i - k_f) e^{i(k_{i_z} - k_{f_z})R_z} + R_i T_f F^{ii}(\tilde{k}_i - k_f) e^{i(-k_{i_z} - k_{f_z})R_z} \\
+ T_i R_f F^{ii}(k_i - \tilde{k}_f) e^{i(k_{i_z} + k_{f_z})R_z} + R_i R_f F^{ii}(\tilde{k}_i - \tilde{k}_f) e^{i(-k_{i_z} + k_{f_z})R_z} ,
\]

\[ f(\theta, \phi) \sim \langle \Psi_{out} | V | \Psi_{in} \rangle \rightarrow 2 \cdot 2 = 4 \text{ terms} \]

Scalar case

\[ f(\theta, \phi) \sim \langle \Psi_{out} | V | \Psi_{in} \rangle \rightarrow 4 \cdot 4 = 16 \text{ terms} \ (2 \times 2) \]

polarization

See talk of Walter Van Herck given at ILL
http://apps.jcns.fz-juelich.de/doku/sc/posters_and_talks
IsGISAXS as a starting point:

- Successful software which is a de facto standard in the user community

- Simulation in DWBA
- FORTRAN 90, 13k lines of code
- Windows (with GUI), Linux (without GUI)
- No longer actively supported
IsGISAXS limitations

- no support for multi layer systems
- no multiple types of particles in single layer
- no polarized neutrons and magnetic materials
- no parallelization
- usability

IsGISAXS parameter file

```plaintext
# GISAXS SIMULATIONS : INPUT PARAMETERS

# Base filename
isgi_2-types-of-particles

# Framework Diffuse, Multilayer, Number of index slices, Polarization
DWBA DA 0 25 ss

# Beam Wavelength : Lambda(nm), WL_distribution, Sigma_WL/WL, WL_min(nm), WL_max(nm), nWL, xWL
0.1 none 0.3 0.08 0.12 20 3

# Beam Alpha_i : Alpha_i(deg), Ai_distribution, Sigma_Ai(deg), Ai_min(deg), Ai_max(deg), nAi, xAi
0.2 none 0.1 0.15 0.25 30 2

# Beam 2Theta_i : 2Theta_i(deg), Ti_distribution, Sigma_Ti(deg), Ti_min(deg), Ti_max(deg), nTi, XTi
0. none 0.5 -0.5 0.5 10 2

# Substrate : n-delta_S, n-beta_S, Layer thickness(nm), n-delta_L, n-beta_L, RMS roughness(nm)
6.E-06 2.e-8 0. 1.E-05 5.E-07 0.

# Particle : n-delta_I, n-beta_I, Depth(nm), n-delta_SH, n-beta_SH
6.E-04 2.e-8 0 8.E-04 2.e-8
```
New software platform or extend existing?

Extending IsGISAXS
- code reuse

From scratch C++
- object oriented approach in sample description
- performance
- developers background

BornAgain framework
Strategy

Reproduce functionality of IsGISAXS
- IsGISAXS examples as milestones along the way

Extend it with most demanded features
- multilayers, particle assemblies, polarized neutrons and magnetic domains
- usability

Other requirements
- open source code
- no dependency on proprietary software
- decouple physical modelling, data, GUI
- multi platform (Linux, Mac, Windows)

Build user community and follow their demands
Development started in April 2012

The name of the software, BornAgain, indicates the central role of the Distorted Wave Born Approximation

- generic frame for modeling and fitting multi layer samples at grazing incidence geometry
- Current Release – 0.9.1
  status – beta
  transition to production release with user help
- Motivation
- Architecture
- Functionality
- Graphical User Interface prototype
- Infrastructure for code development
- Summary
Main features:
- Programming language C++
- Core library consist of 30k lines of code
- Python bindings with boost-python
- Linux, Mac, Windows
Basic software architecture

External dependencies:
- Well established libraries: Boost, fftw3, Eigen, GSL
- ROOT from High Energy Physics community (optional)
Basic software architecture

Working with BornAgain.
- Running user C++ program

User C++ program

External graphics

BornAgain

Python bindings

libBornAgainCore
Samples and algorithms

libBornAgainFit
Interface to minimizers

GSL
Boost
Eigen
ROOT

JCNS Koordinierungstreffen, October 28-29, 2013, Jülich
Working with BornAgain.

- Running user Python script
- Motivation
- Architecture
- **Functionality**
  - Graphical User Interface prototype
  - Infrastructure for code development
- Summary
Functionality (scalar)

- Arbitrary number of layers
- Arbitrary number of different included particles (form factors, material)
- Arbitrary number of interference functions between the particles
- Inter-layer roughness (also with correlation)
- Nanoparticle assemblies
Functionality (polarization)

- Arbitrary number of layers
- Arbitrary number of different included particles (form factors, material)
- Arbitrary number of interference functions between the particles
- Inter-layer roughness (also with correlation)
- Nanoparticle assemblies
Usage examples

Example: simulation of cylindrical nano particles with interference function on top of substrate using python script
Usage example: simulation with python script

```python
from libBornAgainCore import *

# defining materials
air = HomogeneousMaterial("Air", 0.0, 0.0)
substr = HomogeneousMaterial("Substr", 6e-6, 2e-8)
gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

# two empty layers
air_layer = Layer(air)
substrate_layer = Layer(substr)

# creating particle
pyramid_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
pyramid = Particle(gold, pyramid_ff)

# particles with interference
interference = 1DParaCrystal(20*nm, 7*nm, 1e7*nm)
decoration = ParticleDecoration()
decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

# decorating air layer with particles
air_layer.setDecoration(decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving data
arr = GetOutputData(simulation)
```
Usage example: simulation with python script

```python
from libBornAgainCore import *

# defining materials
air = HomogeneousMaterial("Air", 0.0, 0.0)
substr = HomogeneousMaterial("Substr", 6e-6, 2e-8)
gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

# two empty layers
air_layer = Layer(air)
substrate_layer = Layer(substr)

# creating particle
pyramid_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
pyramid = Particle(gold, pyramid_ff)

# particles with interference
interference = 1DParaCrystal(20*nm, 7*nm, 1e7*nm)
decoration = ParticleDecoration()
decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

# decorating air layer with particles
air_layer.setDecorations(decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving data
arr = GetOutputData(simulation)
```
from libBornAgainCore import *

# defining materials
air = HomogeneousMaterial("Air", 0.0, 0.0)
substr = HomogeneousMaterial("Substr", 6e-6, 2e-8)
gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

# two empty layers
air_layer = Layer(air)
substrate_layer = Layer(substr)

# creating particle
pyramid_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
pyramid = Particle(gold, pyramid_ff)

# particles with interference
interference = 1DParaCrystal(20*nm, 7*nm, 1e7*nm)
decoration = ParticleDecoration()
decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

# decorating air layer with particles
air_layer.setDecoration(decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setSample(multi_layer)
simulation.runSimulation()

#retrieving data
arr = GetOutputData(simulation)
Nano particle is defined by the material and form factor.

```
from libBornAgainCore import *

# defining materials
air = HomogeneousMaterial("Air", 0.0, 0.0)
substr = HomogeneousMaterial("Substr", 6e-6, 2e-8)
gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

# two empty layers
air_layer = Layer(air)
substrate_layer = Layer(substr)

# creating particle
pyramid_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
pyramid = Particle(gold, pyramid_ff)

# particles with interference
interference = 1DParaCrystal(20*nm, 7*nm, 1e7*nm)
decoration = ParticleDecoration()
decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

# decorating air layer with particles
air_layer.setDecoration(decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setSample(multi_layer)
simulation.runSimulation()

#retrieving data
arr = GetOutputData(simulation)
```

Additional transformation can be defined for orientation and position of the particle.
InterferenceFunction1DParaCrystal defines interference function of regular lattice with long range order gradually destroyed.

```python
from libBornAgainCore import *

# defining materials
air = HomogeneousMaterial("Air", 0.0, 0.0)
substr = HomogeneousMaterial("Substr", 6e-6, 2e-8)
gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

# two empty layers
air_layer = Layer(air)
substrate_layer = Layer(substr)

# creating particle
pyramid_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
pyramid = Particle(gold, pyramid_ff)

# particles with interference
interference = 1DParaCrystal(20*nm, 7*nm, 1e7*nm)
decoration = ParticleDecoration()
decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

# decorating air layer with particles
air_layer.setDecoration(decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving data
arr = GetOutputData(simulation)
```
**InterferenceFunction1DParaCrystal**

defines interference function of regular lattice with long range order gradually destroyed

**ParticleDecoration**

Holds information about particle types, positions and interference

```python
from libBornAgainCore import *

# defining materials
air = HomogeneousMaterial("Air", 0.0, 0.0)
substr = HomogeneousMaterial("Substr", 6e-6, 2e-8)
gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

# two empty layers
air_layer = Layer(air)
substrate_layer = Layer(substr)

# creating particle
pyramid_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
pyramid = Particle(gold, pyramid_ff)

# particles with interference
interference = 1DParaCrystal(20*nm, 7*nm, 1e7*nm)
decoration = ParticleDecoration()
decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

# decorating air layer with particles
air_layer.setDecoration(decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setSample(multi_layer)
simulation.runSimulation()

#retrieving data
arr = GetOutputData(simulation)
```
**Usage example: simulation with python script**

**InterferenceFunction1DParaCrystal**
Defines interference function of regular lattice with long range order gradually destroyed

```
from libBornAgainCore import *
# defining materials
air = HomogeneousMaterial("Air", 0.0, 0.0)
substr = HomogeneousMaterial("Substr", 6e-6, 2e-8)
gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

# two empty layers
air_layer = Layer(air)
substrate_layer = Layer(substr)

# creating particle
pyramid_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
pyramid = Particle(gold, pyramid_ff)

# particles with interference
interference = 1DParaCrystal(20*nm, 7*nm, 1e7*nm)
decoration = ParticleDecoration()
decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

# decorating air layer with particles
air_layer.setDecoration(decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setSample(multi_layer)
simulation.runSimulation()

#retrieving data
arr = GetOutputData(simulation)
```
Usage example: simulation with python script

```python
from libBornAgainCore import *

# defining materials
air = HomogeneousMaterial("Air", 0.0, 0.0)
substr = HomogeneousMaterial("Substr", 6e-6, 2e-8)
gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

# two empty layers
air_layer = Layer(air)
substrate_layer = Layer(substr)

# creating particle
pyramid_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
pyramid = Particle(gold, pyramid_ff)

# particles with interference
interference = 1DParaCrystal(20*nm, 7*nm, 1e7*nm)
decoration = ParticleDecoration()
decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

decoration

# decorating air layer with particles
air_layer.setDecoration(decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setSample(multi_layer)
simulation.runSimulation()

#retrieving data
arr = GetOutputData(simulation)
```

**MultiLayer**

owns decorated air layer and substrate layer
Usage example: simulation with python script

```python
# defining materials
air = HomogeneousMaterial("Air", 0.0, 0.0)
substr = HomogeneousMaterial("Substr", 6e-6, 2e-8)
gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

# two empty layers
air_layer = Layer(air)
substrate_layer = Layer(substr)

# creating particle
pyramid_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
pyramid = Particle(gold, pyramid_ff)

# particles with interference
interference = 1DParaCrystal(20*nm, 7*nm, 1e7*nm)
decoration = ParticleDecoration()
decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

# decorating air layer with particles
air_layer.setDecoration(decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving data
arr = GetOutputData(simulation)
```

**MultiLayer**
owns decorated air layer and substrate layer
Usage example: simulation with python script

MultiLayer
owns decorated air layer and substrate layer

Simulation
Holds beam and detector parameters, accept multi layer as an input and perform calculations

```
from libBornAgainCore import *

# defining materials
air = HomogeneousMaterial("Air", 0.0, 0.0)
substr = HomogeneousMaterial("Substr", 6e-6, 2e-8)
gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

# two empty layers
air_layer = Layer(air)
substrate_layer = Layer(substr)

# creating particle
pyramid_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
pyramid = Particle(gold, pyramid_ff)

# particles with interference
interference = 1DParaCrystal(20*nm, 7*nm, 1e7*nm)
decoration = ParticleDecoration()
decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

# decorating air layer with particles
air_layer.setDecoration(decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setSample(multi_layer)
simulation.runSimulation()

# retrieving data
arr = GetOutputData(simulation)
```
OutputData
Simulated intensity as a function of outgoing $\alpha_f$, $\phi_f$ angles can be retrieved as NumPy array

```python
from libBornAgainCore import *

# defining materials
air = HomogeneousMaterial("Air", 0.0, 0.0)
substr = HomogeneousMaterial("Substr", 6e-6, 2e-8)
gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

# two empty layers
air_layer = Layer(air)
substrate_layer = Layer(substr)

# creating particle
pyramid_ff = FormFactorPyramid(5*nm, 5*nm, 54.73*deg)
pyramid = Particle(gold, pyramid_ff)

# particles with interference
interference = 1DParaCrystal(20*nm, 7*nm, 1e7*nm)
decoration = ParticleDecoration()
decoration.addParticle(pyramid, RotateZ3D(45.*deg))
decoration.addInterferenceFunction(interference)

# decorating air layer with particles
air_layer.setDecoration(decoration)

# creating multilayer
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)

# simulating
simulation = Simulation()
simulation.setSample(multi_layer)
simulation.runSimulation()

#retrieving data
arr = GetOutputData(simulation)
```
Example: fitting
Usage examples: fitting the data

BornAgain framework

Koordinierungstreffen, October 28-29, 2013, Jülich

this is your minimizer
Usage examples: fitting the data

Fitting in BornAgain

- Variety of multidimensional minimization algorithms and strategies
- The choice over possible fitting parameters, their properties and correlations
- Full user control on chi2 calculations
- Simultaneous fit of arbitrary number of datasets
Usage example: fitting

- Every number (or group of numbers) used in sample construction can be used as fit parameter

**Minimization algorithms provided by ROOT**

<table>
<thead>
<tr>
<th>Library</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minuit2</td>
<td>Migrad, Simplex, Fumili, Scan</td>
</tr>
<tr>
<td>GSL</td>
<td>Fletcher-Reeves Conjugate Gradient Algorithm</td>
</tr>
<tr>
<td></td>
<td>BFGS Conjugate Gradient Algorithm</td>
</tr>
<tr>
<td></td>
<td>Levenberg-Marquardt Algorithm</td>
</tr>
<tr>
<td></td>
<td>Simulated Annealing Algorithm</td>
</tr>
<tr>
<td>TMVA</td>
<td>Genetic Algorithm</td>
</tr>
</tbody>
</table>

```
from libBornAgainCore import *
from libBornAgainFit import *

# creating minimizer
minimizer = MinimizerFactory.createMinimizer("Minuit2","Migrad")

# creating fitting engine
fitSuite = FitSuite()
fitSuite.setMinimizer(minimizer)

# creating fit parameters
fitSuite.addFitParameter("*/FormFactorCylinder/height", 4.*nm, AttLimits.lowerLimited(0.01))
fitSuite.addFitParameter("*/FormFactorCylinder/radius", 4.*nm, AttLimits.lowerLimited(0.01))

# setting simulation (from previous example) and real ASCII data
fitSuite.addSimulationAndRealData(simulation, data)

# running fit
fitSuite.runFit()
```
Usage example: demo
Motivation

Architecture

Functionality

Graphical User Interface prototype

Infrastructure for code development

Summary
Objectives

- Simulation and fit from graphical user interface
- Short learning curve for users
- Developing through prototyping and frequent usability testing
Bad graphical user interface
**Planning GUI**

**Current GUI prototype:**
- 20k lines of code
- 3 man-months

**First public beta:** Simulation from GUI
- 30k lines of code

**Fitting from GUI**
- 40k lines of code

**Around 60-100k lines of code for the GUI at the end**

In the future

+3 man-months

+3 man-months
Planning GUI

Current GUI prototype:
- 20k lines of code
- 3 man-months

First public beta: Simulation from GUI
- 30k lines of code

"Agile methods breaks tasks into small increments with minimal planning and do not directly involve long term planning"

Around 60-100k lines of code for the GUI at the end

In the future

BornAgain framework

JCNS Koordinierungstreffen, October 28-29, 2013, Jülich
- Motivation
- Architecture
- Functionality
- Graphical User Interface prototype
- Infrastructure for code development
- Summary
Development process

General practices

- More agile development
- Quality control
- 😊 Good documentation

Software tools

- Bug/issue tracking: Redmine
- Version control: Git
- Nightly build server: TeamCity
- Code documentation: Doxygen
- Code browser: OpenGrok
- Unit tests: Googletest

BornAgain git repository
number of lines of code vs time
Development process: agile development

- workflow consists of sprint cycles every 3-4 weeks during which the team creates finished portions of product.

Users → Backlog:
- Item 124
- Item 125
- Item 127
- ...

Developers

Sprint:
- Item 98
- Item 101
- Item 102
- ...

Release

One day I want to fly away...

BornAgain framework

JCNS Koordinierungstreffen, October 28-29, 2013, Jülich
Development process: agile development

Workflow is managed via Redmine management tool
- user request handling, bug tracking tool, issue tracking
Development process: quality assurance

Unit test control
- small test cases are executed after each compilation to ensure code stability

Nightly build system
- based on TeamCity
- automatic nightly compilation and testing of the whole project
Development process: quality assurance

**Functional tests**
- higher level tests produces scattering plots which are automatically analyzed
- 14 scattering geometries tested through Python and C++ API’s
- special tests compare BornAgain results with IsGISAXS results
o Motivation
o Architecture
o Functionality
o Graphical User Interface prototype
o Infrastructure for code development
o Summary
What has been done

April, 2012 – April, 2013
Released first public version of the framework (home workshop at Garching, April, 4)

- Mac/Linux support
- IsGISAXS functionality is mostly in place
- Fitting, Python bindings
- Project infrastructure is settled
- One real life example: mesocrystal of self assembled nanoparticles
**What has been done**

**May – October, 2013**

- Implemented GUI prototype  
  3 man-months

- Replaced build system with user friendly CMake  
  1 man-month

- Wrote alfa-version of user manual 30 pages  
  1 man-month

- DWBA formalism for polarized neutrons studies  
  1 man-month ;)

- Refactored the framework for polarized DWBA  
  1 man-month

- Implemented Windows build and installer  
  2 man-months

- Different IT studies CMake, PyQt4, SIP, Qt5  
  1 man-month

- Made 6 talks at different occasions  
  including Abingdon, UK, “Emerging GISAXS 2013“ workshop  
  1.5 man-months

- Made one poster  
  *for Hamburg, DESY GISAXS2013 workshop*
What has to be done

**Started to receive requests**
- Jean Francois Moulin (3 layered system with hexagonal lattice on top)
- Oleg Petracic, Alice Klapper (BPM media)
- Stefan Mattauch (lateral structured test sample)
- Sabrina Disch (magnetic mesocrystal)
- Elisabeth Josten (mesocrystal)
- A number of requests from ILL

**Plans till June, 2014**
- Public beta of graphical user interface (simulation)
- Test of polarized DWBA on magnetic sample

---

**Possible time budget allocation for the next 6 months**
- Graphical user interface: 30%
- User requests: 30%
- Refactoring, missed functionality: 20%
- User manual: 10%
- Build and release support: 10%

BornAgain framework
Where to get information

Google
“Scientific Computing at MLZ”

You will arrive
http://apps.jcns.fz-juelich.de

Posters And Talks
Have copy of given talk, see other talks

BornAgain
http://apps.jcns.fz-juelich.de/BornAgain

Source code
Quick start
User manual
Redmine issue tracker
BornAgain
making DWBA manageable

Thank you
Grazing incidence small-angle x-ray scattering from a mesocrystalline system

- mesocrystals have cylindrical shape and size of \(~1000\) nm
- mesocrystal consist of an FCC lattice composed of 5 nm spherical particles
Usage example: mesocrystals

Experimental data compared to a BornAgain simulation

- 11 parameters fit
- 12 hours on six core (12 threads), 2000 iterations, 20 sec/iteration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beam_intensity</td>
<td>5.01e+12</td>
</tr>
<tr>
<td>lattice_length_a</td>
<td>6.21</td>
</tr>
<tr>
<td>lattice_length_c</td>
<td>6.57</td>
</tr>
<tr>
<td>nanoparticle_radius</td>
<td>4.70</td>
</tr>
<tr>
<td>sigma_nano易于_-radius</td>
<td>0.37</td>
</tr>
<tr>
<td>meso_height</td>
<td>112.22</td>
</tr>
<tr>
<td>meso_radius</td>
<td>945.63</td>
</tr>
<tr>
<td>sigma_meso_height</td>
<td>1.00</td>
</tr>
<tr>
<td>sigma_meso_radius</td>
<td>1.61</td>
</tr>
<tr>
<td>sigma_lattice_length_a</td>
<td>1.16</td>
</tr>
<tr>
<td>surface_filling_ratio</td>
<td>0.17</td>
</tr>
<tr>
<td>roughness</td>
<td>28.72</td>
</tr>
</tbody>
</table>

**Experimental data courtesy Elisabeth Josten et al**

**BornAgain simulation in collaboration with Artur Glavic and Elisabeth Josten**
C++ and Python API’s

```c++
const IMaterial *air_material =
    MaterialManager::getHomogeneousMaterial("Air", 0.0, 0.0);
const IMaterial *particle_material =
    MaterialManager::getHomogeneousMaterial("Particle", 6e-4, 2e-8);
FormFactorCylinder *cylinder_ff = new FormFactorCylinder(5*nanometer, 5*nanometer);
Particle *particle = new Particle(particle_material, cylinder_ff);

ParticleDecoration particle_decoration;
particle_decoration.addParticle(particle, 0.5);

InterferenceFunction1DParaCrystal *interference =
    InterferenceFunction1DParaCrystal(20*nanometer, 7*nanometer, 1e3*nanometer);
particle_decoration.addInterferenceFunction(interference);

Layer air_layer(air_material);
air_layer.setDecoration(particle_decoration);
```

```python
from BornAgainCore import *

air_material = MaterialManager.getHomogeneousMaterial("Air", 0.0, 0.0)
particle_material = MaterialManager.getHomogeneousMaterial("Particle", 6e-4, 2e-8)

cylinder_ff = FormFactorCylinder(5*nanometer, 5*nanometer)
cylinder = Particle(particle_material, cylinder_ff)

particle_decoration = ParticleDecoration()
particle_decoration.addParticle(cylinder, 0.5)

interference = InterferenceFunction1DParaCrystal(20*nanometer, 7*nanometer, 1e3*nanometer)
particle_decoration.addInterferenceFunction(interference)

air_layer = Layer(air_material)
air_layer.setDecoration(particle_decoration)
```
Usage example: development of graphical user interface

Status – prototyping

- rely on Qt5 (Qt4) and ROOT libraries
- BornAgainCore library used as a plugin

Objectives

- drag and drop sample editor
- interactive plots
- access to large collection of minimizers
- Python and C++ scripting
Development process: quality assurance

Nightly build system
- based on TeamCity, one build configuration

Code profiling
- regular tracking of memory and CPU consumption
Development process: good documentation

- Doxygen generated, starting to write user manual

Theory and conventions
- described and stored in the repository parallel to the code development
Quick start

http://apps.jcns.fz-juelich.de/BornAgain

on Unix platforms

1. Install third party software
   - Boost, GSL, fftw, numpy-devel
2. Download source code
3. Compile:
   - cmake; make; make install
4. Run example
   - python CylindersAndPrisms.py

on Windows platforms

1. Install necessary Python modules
2. Run BornAgain installer
3. Run example
   - double-click CylindersAndPrisms.py