The making of BornAgain

Jan Burle, Céline Durniak, Jonathan Fisher,
Marina Ganeva, David Li, Walter Van Herck,
Gennady Pospelov, Joachim Wuttke

Scientific Computing Group at MLZ
Jülich Center for Neutron Science

SINE2020 Coordination Meeting, Villigen PSI, April 5, 2016
Outline

- Introduction
- Software architecture
- Demonstration
- Under the hood
- Closing remarks
Welcome to BornAgain

BornAgain is a software package to simulate and fit small-angle scattering at grazing incidence. It supports analysis of both X-ray (GISAXS) and neutron (GISANS) data. Its name, BornAgain, indicates the central role of the distorted wave Born approximation in the physical description of the scattering process. The software provides a generic framework for modeling multilayer samples with smooth or rough interfaces and with various types of embedded nanoparticles.
Experiment

- the beam is directed on a surface with a very small incident angle
- 2D detector records the intensity of scattered wave giving access to lateral and vertical sample structure information

Simulation

- Intensity is calculated from known sample structure using Distorted Wave Born Approximation

\[ \frac{d\sigma}{d\Omega} = \langle |F_{DWBA}|^2 \rangle S(q) \]
Motivation

Support for specific instruments at MLZ
- Serve our users, support in house research, at Maria and REFSANS instruments

Limited functionality of existing software
- No support for polarized neutrons, limitations in sample geometry
- Usability issues, lack of support

High Data Rate Processing and Analysis initiative (HDRI)
- Call to create simulation software for non-expert users for GISAS field
- Provide functionality/extensibility for broader usage
IsGISAXS as an example

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

**IsGISAXS: a program for grazing-incidence small-angle X-ray scattering analysis of supported islands**

R Lazzari - Journal of Applied Crystallography, 2002 - scripts.iucr.org

This paper describes a Fortran program, IsGISAXS, for the simulation and analysis of grazing-incidence small-angle X-ray scattering (GISAXS) of islands supported on a substrate. As is usual in small-angle scattering of particles, the scattering cross section is...

Cited by 257 Related articles All 7 versions Cite

- Simulation in DWBA
- FORTRAN 90, 13k lines of code
- No longer actively supported

IsGISAXS parameter file
- Introduction
- **Software architecture**
- Demonstration
- Under the hood
- Closing remarks
User needs

BornAgain framework
User needs

Users → Easy-to-use → GUI
Requirements

- Users
  - Easy-to-use
  - GUI
  - Availability
    - Open source
    - No proprietary software
    - Easy-to-install
  - Multi platform
    - Windows
    - Mac
    - Linux
    - Clusters
  - Complex models
  - Performance
  - Extensibility
  - Scripting
    - C++
    - OO, decoupling
    - Python
Software Architecture

- Open-source framework written in C++, interfaced with Python
  - distributed under GPL3 license

- Multi-platform
  - Unix flavors, source code
  - Windows, binary installer package
  - Mac OS, binary installer package

- Object-oriented approach for sample description
Software Architecture

- Open-source framework written in C++, interfaced with Python
  - distributed under GPL3 license
- Multi-platform
  - Unix flavors, source code
  - Windows, binary installer package
  - Mac OS, binary installer package
- Object-oriented approach for sample description

```python
# defining materials
m_air = HomogeneousMaterial("Air", 0.0, 0.0)
m_substrate = HomogeneousMaterial("Substrate", 6e-6, 2e-8)
m_particle = HomogeneousMaterial("Particle", 6e-4, 2e-8)

# collection of particles
cylinder_ff = FormFactorCylinder(5*nanometer, 5*nanometer)
cylinder = Particle(m_particle, cylinder_ff)
prism_ff = FormFactorPrism3(10*nanometer, 5*nanometer)
prism = Particle(m_particle, prism_ff)
particle_layout = ParticleLayout()
particle_layout.addParticle(cylinder, 0.0, 0.5)
particle_layout.addParticle(prism, 0.0, 0.5)

# air layer with particles and substrate form multi layer
air_layer = Layer(m_air)
air_layer.addLayout(particle_layout)
substrate_layer = Layer(m_substrate)
multi_layer = MultiLayer()
multi_layer.addLayer(air_layer)
multi_layer.addLayer(substrate_layer)
```
Package structure

- C++ kernel for simulation description and fitting
- Python bindings
- Graphical User Interface
- Well established Open Source libraries as external dependencies
- CMake based
Working with BornAgain

- Using Graphical User Interface
- Running Python script with simulation description
Working with BornAgain

- Using Graphical User Interface
- Running Python script with simulation description

**Diagram:**
- Python script
- User
- matplotlib

**Dependencies:**
- Python bindings
- libBornAgainCore
  - Samples and algorithms
- libBornAgainFit
  - Fitting routines

**Other:**
- Eigen
- GSL
- Boost
- fftw3
- Qt5

BornAgain framework
Functionality

- X-rays, non-polarized and polarized neutrons
- Arbitrary number of layers
- Simple and composite particles
- Correlated positions
- Rough interfaces
- Nanoparticle assemblies
- Off-specular geometry, beam divergence
o Introduction
o Software architecture
o Demonstration
o Under the hood
o Closing remarks
- Introduction
- Software architecture
- Demonstration
- Under the hood
- Closing remarks
Under the hood

Number of lines of code vs time
Development tools

- Version control system (git)
- Issue tracking (redmine)
- Nightly build, CI (teamcity -> docker + vagrant + buildbot)
- Unit tests (googletest, QtTest)
- Functional tests
- Release procedure

- Other
  - Google analytics
  - slack
  - Doxygen
  - Valgrind, Coverity, MacOS/Instruments
  - Blender/Inkscape
o Introduction
o Software architecture
o Demonstration
o Under the hood
o Closing remarks
Python bindings

C++/Python relationship

Extension writing Python access to C++

Embedding C++ access to the Python interpreter
Wrapper function
- Converts function arguments from Python to C, returns results in Python expected form
- Has to be registered for Python interpreter

```c
int fact(int n)
{
    if (n <= 1)
        return 1;
    else
        return n * fact(n - 1);
}
```

```c
#include <Python.h>

PyObject *wrap_fact(PyObject *self, PyObject *args)
{
    int n, result;
    if (!PyArg_ParseTuple(args, "i:fact", &n))
        return NULL;
    result = fact(n);
    return Py_BuildValue("i", result);
}

static PyObject* exampleMethods[] = {{"fact", wrap_fact, 1}, {NULL, NULL}};

void initexample()
{
    PyObject *m;
    m = Py_InitModule("example", exampleMethods);
}
```
Python bindings

Choosing technology to wrap a complex C/C++ application

- External dependencies?
- What is the performance?
- Build system integration?
- Is wrapping code on Python side or on C++ side?
- How much code should be written additionally?
- Should I affect or duplicate existing C++ code?
- How big is the community?
- Is it possible to fully automate wrappers generation?
- Do I need bindings with another languages?

After careful consideration we have chosen

```
boost::python
```
Choosing technology to wrap a complex C/C++ application

- External dependencies?
- What is the performance?
- Build system integration?
- Is wrapping code on Python side or on C++ side?
- How much code should be written additionally?
- Should I affect or duplicate existing C++ code?
- How big is the community?
- Is it possible to fully automate wrappers generation?
- Do I need bindings with another languages?

After careful consideration we have chosen **boost::python**.

3 years later...
After careful consideration we have switched to **SWIG**.
Python bindings

SWIG bindings in BornAgain (starting from next release 1.6)

- Binding generation is governed by a SWIG interface file
  ```cpp
  %{  
    include "ISample.h"  
  }  
  %include "ISample.h"  
  %feature("director") ISample;
  ```

- Interface file can be fine-tuned to ignore certain methods of classes or tweak existing one
  - No change to the original C++ code is required

- Generation of bindings is done via swig executable
  ```bash
  $ swig libBornAgainCore.i
  ```

*Produces additionally 130k lines of C++, 25k lines of Python*
from bornagain import *

def buildSample():
    air = HomogeneousMaterial("Air", 0.0, 0.0)
    gold = HomogeneousMaterial("Gold", 6e-4, 2e-8)

    cylinder_ff = FormFactorCylinder(5.0, 5.0)
cylinder = Particle(gold, cylinder_ff)
particle_layout = ParticleLayout(cylinder)

    air_layer = Layer(m_ambience)
    air_layer.addLayout(particle_layout)

    multi_layer = MultiLayer()
    multi_layer.addLayer(air_layer)

    return multi_layer

#include "MultiLayer.h"

std::unique_ptr<ISample> buildSample()
{
    HomogeneousMaterial air("Air", 0.0, 0.0);
    HomogeneousMaterial gold("Gold", 6e-4, 2e-8);

    FormFactorCylinder ff_cylinder(5.0, 5.0);
    Particle cylinder(gold, ff_cylinder);
    ParticleLayout particle_layout(cylinder);

   .Layer air_layer(air);
    air_layer.addLayout(particle_layout);

    std::unique_ptr<MultiLayer> result
        = std::make_unique<MultiLayer>();
    result->addLayer(air_layer);

    return result;
}
Python bindings

Achieved results

- Supports both Python 2.7 and 3
- Generated code is portable (compiles with gcc, clang and Visual Studio)
- Supports shared ownership, transfer of ownership
- Automatic conversion between many C++ types/containers and those on Python side
  - std::string/Python string, std::vector/Python list, std::map/Python dict
- Allows custom conversions
  - vector<vector<double>> -> Numpy array
- Python docstring is made out of C++ doxygen comments
- Cross-language polymorphism

```cpp
class IFitObserver {
    virtual void update(FitSuite *suite);
};

class FitSuite {
    void attach(IFitObserver *observer);
    void runFit() {
        observer->update(this);
    }
};
```

```python
class DrawObserver(IFitObserver):
    def __init__(self):
        IFitObserver.__init__(self)

    def update(self, fit_suite):
        pyplot.imshow(fit_suite)

observer = DrawObserver()
fitSuite = FitSuite()
fitSuite.attach(observer)
```
- Introduction
- Software architecture
- Demonstration
- Under the hood
- Closing remarks
GUI main features

- 60k lines of code, Qt5 based, C++
- Additional 3rd party code (included in source tree)
  - QCustomePlot (scientific graphics)
  - Qt-manhattan-style (few styles/widgets borrowed from Qt creator code)
  - Qt-propertybrowser-framework (dynamic property editors generation)
GUI main features

The Model/View architecture
- The data (model), user interface (view) and interactions (controller) are separated

Advantages
- Same data can be displayed in many views
- Increased flexibility and reuse
- Possibility to unit-test GUI logic outside of GUI context
GUI main features

Presentation Model

- Holds all the data (sample parameters, presentation attributes, widgets status)
- Every row in the model corresponds to `SessionItem`

```cpp
class SessionModel
{
    SessionItem *rootItem;
};

class SessionItem
{
    QString itemType;
    QString displayName;
    QVariant data;

    vector<SessionItem *> children;
};
```
GUI main features

**Presentation Model**
- Holds all the data (sample parameters, presentation attributes, widgets status)
- Every row in the model corresponds to `SessionItem`

```cpp
class SessionModel {
    SessionItem *rootItem;
};

class SessionItem {
    QString itemType;
    QString displayName;
    QVariant data;
    vector<SessionItem *> children;
};
```
GUI main features

Presentation Model and its Views
- Part of presentation model related to job results
- Job views representing different items of job model
GUI main features

Presentation Model
- Conform to QAbstractItemModel interface

```cpp
signals:
    void dataChanged(const QModelIndex & topLeft, const QModelIndex & bottomRight);
    void rowsInserted(const QModelIndex & parent, int first, int last);
```

- Various proxy models allows to hide certain model parts from the view

- Serialization is done via XML stream
  - Saving the model in file, drag-and-drop, copying/cloning across the model

- Additional machinery allows non-Qt objects to be notified on SessionItem change

```cpp
Widget::Widget(SessionItem * item)
{
    item->mapper()->setOnSiblingsChange([this](){ onSiblingsChange(); });
}

void Widget::onSiblingsChange()
{
    // do something special when any of siblings of given item are changed
}
```
GUI main features

All activities are done through the model

- Drag and Drop action adds an item to the model
  - Graphics scene gets notified and draws new item

- Connection of items through node editor leads to request to change the parent in the model
  - Graphics scene gets notified and draws connection
GUI main features

GUI / Core relationship

 Converts domain objects (standard samples, library materials etc) into their GUI counterparts

 Generates core domain simulation object, runs it in non-GUI thread

 Knows how to retrieve simulation results

Core is Qt-independent and fully unaware of GUI existence
- Introduction
- Software architecture
- Demonstration
- Under the hood
- Closing remarks
Testing

Unit tests
- Core library (google-test, 330 tests), GUI models (QtTest, 60 tests)

Functional tests
- Runs simulation for certain geometry, produces intensity plot
- Compares the plot with the reference
  - simulation from previous day
  - simulation through different chain (Core/GUI/Python)
  - simulation of identical samples obtained in different way

- Create particle composition from two hemi spheres
- Assign same material to them
- Compare with normal full sphere, same material, same radius
- Scattering intensities should be identical
Functional tests for Core/GUI/Python domains

- When new functionality is implemented the corresponding standard simulation is added to the factory
- Corresponding intensity data is generated and saved for future reference.

```
make check launches test simulations for all 3 domains

<table>
<thead>
<tr>
<th>Test #</th>
<th>Test Name</th>
<th>Status</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>139/146</td>
<td>GUISuite/BoxCompositionRotateZandY</td>
<td>Passed</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>Start 140: GUISuite/BoxStackComposition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>140/146</td>
<td>GUISuite/BoxStackComposition</td>
<td>Passed</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>Start 141: GUISuite/SimulationWithMasks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>141/146</td>
<td>GUISuite/SimulationWithMasks</td>
<td>Passed</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>Start 142: GUISuite/RectDetectorGeneric</td>
<td></td>
<td></td>
</tr>
<tr>
<td>142/146</td>
<td>GUISuite/RectDetectorGeneric</td>
<td>Passed</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>Start 143: GUISuite/RectDetectorPerpToSample</td>
<td></td>
<td></td>
</tr>
<tr>
<td>143/146</td>
<td>GUISuite/RectDetectorPerpToSample</td>
<td>Passed</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>Start 144: GUISuite/RectDetectorPerpToDirectBeam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>144/146</td>
<td>GUISuite/RectDetectorPerpToDirectBeam</td>
<td>Passed</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>Start 145: GUISuite/RectDetectorPerpToReflectedBeam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145/146</td>
<td>GUISuite/RectDetectorPerpToReflectedBeam</td>
<td>Passed</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>Start 146: GUISuite/RectDetectorPerpToReflectedBeamDpos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>146/146</td>
<td>GUISuite/RectDetectorPerpToReflectedBeamDpos</td>
<td>Passed</td>
<td>0.06</td>
</tr>
</tbody>
</table>
```

100% tests passed, 0 tests failed out of 146

Total Test time (real) = 58.81 sec
[100%] Built target check
Functional tests for Core/GUI/Python domains
Functional tests for Core/GUI/Python domains

1. Generate text of Python script
   - Generate text of Python script

2. Core simulation
   - Core simulation

3. Intensity Data
   - Intensity Data

4. Numeric Difference?
   - Numeric Difference?

5. Reference Data
   - Reference Data

Flow:
- Python script → Embedded Python
- Embedded Python → Intensity Data
- Intensity Data → Numeric Difference?
- Numeric Difference? → Reference Data
- Reference Data → BornAgain framework
Functional tests for Core/GUI/Python domains
Validation

Validation against existing software

BornAgain

IsGISAXS

difference

Validation against experimental data

BornAgain framework
Closing remarks

Horizon 2020 Initiative
o BornAgain as a community project for GISAS and Reflectometry
o Fitting of GISAS, Off-Specular and Specular data in a single framework

Further software development tasks
o Fitting in GUI (prototype in next release)
o Real sample representation using Qt3D
o Switch to Qt installer framework to create MacOS and Windows installers

Further kernel development
o Implement specular intensity
o Magnetic roughness and magnetic domains
bornagainproject.org
making DWBA manageable

Thank you!