The Bayeux library and the SuperNEMO software

François Mauger, LPC Caen

GDR neutrino Meeting, Orsay, June 18, 2014
Overview

- Introduction
- Bayeux
  a fundation library for data processing, geometry, computing, simulation... and all that sort of thing!
- Falaise
  a few words about the SuperNEMO software framework.
- Perspectives and conclusion
The origins of the SuperNEMO software framework

The SuperNEMO R&D started in 2004 (LPC Caen and LAL):

- 2004: MC study for a Selenium-Xenon TPC\(^a\),
- 2005+: BiPo1 and BiPo2 prototype detectors\(^b\): DAQ, MC, data analysis, visualization software
- 2005+: NAT++ (a C++ based data analysis alternative framework for NEMO3)\(^c\)
- 2006+: SuperNEMO preliminary MC tools

---

\(a\). F. Mauger, Simulation of DBD in the SeXe TPC, TPC 2006, Paris
\(b\). arxiv-1005.0343
\(c\). PhD: Y. Lemière, M. Bongrand, A. Chapon, S. Blondel, C. Hugon, B. Soulé...
Introduction

François Mauger

The Bayeux library and the SuperNEMO software
What about existing third party software... 

- BiPo, NEMO3, SuperNEMO: many needs to address in parallel for MC, data analysis, for various prototype setups with very different geometries and various data models and processing approaches,

- Existing tools (a.k.a. *Third party software*: Root, Geant4...) have nice features but do not share an unified interface with other tools unless we painfully hardcode software bridges,

- Geometry modelling: Geant4 and Root have their own system (two distincts sets of C++ classes), with some recovery but far to be fully compatible, and without interoperability,

- More, they usually force users to *put all one's eggs in one basket*,

- No existing unique SW tool (Root, Geant4...) able to address all these topics in an unified framework unless one hardcodes lots of things, repeat plenty of tedious coding, investigate sparse/obsolete documentation, finally read the source code... then go mad!

François Mauger  The Bayeux library and the SuperNEMO software
Missing features... 

- Some scalable data models that are not fully designed during the R&D stages, still efficient for production, handled through versatile data processing tools,

- An *agnostic* portable I/O system able to serialize plain C++ data structure, STL, smart pointers, crossed referenced objects...

- Unified geometry modelling system with smart associated tools: factories of geometry patterns, volume identification/numbering scheme, locator mechanisms...

- Missing a scalable and unified configuration system with support of system of units, traceability, reuseability, self documentation, integrability in a larger software environment, versionning and configuration variants...

- Good support of (or compatibility with) standard features of the C++ language (templates, STL, Boost),
Problems specific to the SuperNEMO $\beta\beta$ project

Monte-Carlo tools:

- Need for simulation of many different hypothesis (elaborate background models): a lot of radioactive processes ($\beta\beta$ decays, radioactivity generators), vertexes at many different locations/identifiers of the geometry,

- Some similar tools are used for BiPo, SuperNEMO, shielding studies, HPGe: low radioactivity and low energy physics, shared technologies and geometry, signal processing...

\[\sim\text{ avoid to replicate the code.}\]
The basics of the SuperNEMO software (1)

... designed from original works on NAT++ and BiPo1 SW:

- Use plain C++ and standard features of the language (STL, Boost),
- Use and wrap existing features from third party software (TP) whenever it makes sense:
  - do not reinvent the wheel, but hide implementation details from TP and propose an unique user interface,
  - limit to really useful things for the project(s),
  - introduce new features when needed,
  - easier to maintain $O(10 \ y)$ and adapt to future TP interface changes,
- Make things as generic as possible (in the limits of our activities),
- Plugin mechanisms and object factories to support user extensions,
The basics of the SuperNEMO software (2)

- Use the *divide to rule* strategy: implement software modules for well confined independent tasks with a clear interface,
- Minimize coding: scripting facilities through human friendly configuration files, dynamic management of objects through manager classes and object factories,
- Integration: distribute a standard and consistent software framework for all activities in the collaboration,
- Open source (GPL), Subversion, CMake.
- Do not strive for perfection, try to be useful, do our best...
Introduction

Historical architecture: prototype for NEMO3, BiPo1

Experiment(s)
- Package
- Application
- Package

Foundation
- Experiment(s)
- Package
- Application
- Package

Third party
- Geant4
- Root
- CERNLIB
- GSL

Languages
- C/C++
- Fortran 77

OS
- Linux

Languages
- Fortran 77
- C/C++

Third party
- Boost

OS
- Linux
Introduction

Current architecture: production/development for SuperNEMO/BiPo3

Experiment(s)
- App. SN−1 (Topic SN−A)
- App. SN−2 (Topic SN−B)
- App. X−1 (Topic X−A)
- App. X−2 (Topic X−B)

Foundation
- Application 1
  - Topic D
- Application 2
  - Topic C
- Application 2
  - Topic B
- Application 2
  - Topic A

Third party
- CLHEP/G4
- Root
- GSL
- ... (language not specified)

Languages
- C/C++
- Python
- Shell

OS
- Linux
- MacOSX

Topic A
Topic B
Topic C
Topic D
Topic E
Topic F
Topic G
Introduction

Current SW blocks

Falaise (SuperNEMO)  Chevreuse (BiPo)

Bayeux
- Data models, I/O, configuration...
- Computing, PRNG
- Data processing, pipeline
- Geometry modelling
- Vertex, decay generation
- EM field modelling

Geant4 MC driver

Cadfael
- Boost, CAMP, CLHEP, GSL, Xerces, Root, Geant4
- Automated download/build/installation
- CMake-ization

Foundation

Other projects...

System
- Languages
- OS (Linux/BSD)

Third party

Francois Mauger

The Bayeux library and the SuperNEMO software
Overview

- About Bayeux...
- Configuration tools
- Data model(s) and I/O
- Geometry
- Simulation
- Data processing pipeline
- Conclusion
What is Bayeux and what’s in it?

- A collection of C++ classes and functions designed for the simulation, recording and analysis of data for experimental particle and nuclear physics projects.

- Split into several specialized submodules:
  - **datatools**: Serializable data structures based on Boost and core utilities (object factories, services, configuration…).
  - **brio**: A Boost over ROOT I/O serialization system (extends datatools I/O).
  - **cuts**: Generic tools for making arbitrary data selections.
  - **materials**: Description of isotopes, elements and materials plus tools for input to simulation applications (i.e. GDML@Geant4/Root).
  - **mygsl**: C++ wrappers and extensions to the GNU Scientific Library (C).
What is Bayeux and what’s in it?

- A collection of C++ classes and functions designed for the simulation, recording and analysis of data for experimental particle and nuclear physics projects.

- Split into several specialized submodules:
  - **geomtools**: Generic modelling tools for working with experiment geometries and provide input to simulation and visualization tools (i.e. GDML for Geant4/Root).
  - **emfield**: Electromagnetic field modelling and management.
  - **dpp**: A basic data processing pipeline API.
  - **genbb_help**: C++ port and extensions to the Decay0/GENBB program by Vladimir Tretyak, for input to simulation libraries and applications.
  - **genvtx**: Vertex random generator tools for input to simulation libraries and applications.
  - **mctools**: Utilities for particle and nuclear physics simulation with a Geant4 interface.
Those who make Bayeux...

- François Mauger (NEMO3, SuperNEMO, LPCTrap...), LPC Caen,
- Benoît Guillon (SuperNEMO, SoLid), LPC Caen,
- Xavier Garrido (SuperNEMO), LAL,
- Benjamin Morgan (SuperNEMO, Geant4...), Warwick University,
- Other contributors.
Why Bayeux?

Provide tools to describe and use an experimental setup in terms of:

- Geometry (for MC and visualization),
- Data models, I/O, services,
- Data processing for real or MC data: digitization, reconstruction, analysis,
- Monte-Carlo studies and production (using Geant4).
The **datatools::properties** class

- A generic store for configuration parameters or versatile data
- Dictionary of key/value entries
- Types: boolean, integer & real numbers, character strings and arrays of that types
- Description embedded in the parameters’ entries
- Serializable through the **datatools** I/O system
- Loadable from/storable to ASCII files through a dedicated parser/writer
- Support a large set of units (ala CLHEP): time, length, mass...
- Other features...
Example : configure an algorithm object (1)

```cpp
#include <datatools/properties.h>
#include <datatools/clhep_units.h>
class my_analysis {
public:
    void initialize(const datatools::properties & setup_) { /* ... */ }
    void run() { /* ... */ }
    void reset() { /* ... */ }
};
int main(void) {
    datatools::properties setup;
    setup.set_description("Configuration parameters for my data analysis");
    setup.store_flag("debug", "Debug mode activation");
    setup.store("number_of_threads", 4, "For multi-core host");
    setup.store("tolerance", 1.0 * CLHEP::mm, "Length tolerance");
    setup.store("histograms.mode", "1d+2d", "Generation of histograms");
    std::vector<double> fdims;
        fdims.push_back(1.2 * CLHEP::cm);
        fdims.push_back(3.6 * CLHEP::cm);
    setup.store("fiducial.dimensions", fdims);
    std::vector<std::string> ifiles;
        ifiles.push_back("${DATA_DIR}/data/run_1.data.gz");
        ifiles.push_back("${DATA_DIR}/data/run_2.data.gz");
    setup.store("files.input", ifiles, "Input data files");
    setup.tree_dump(std::clog, "The configuration parameters: ");
    my_analysis my_ana;
    my_ana.initialize(setup);
    my_ana.run();
    my_ana.reset();
    return 0;
}
```
Example: configure an algorithm object (2)

The configuration parameters:
|-- Description : 'Configuration parameters for my data analysis'
|-- Name : "debug"
  |-- Description : Debug mode activation
  |-- Type : boolean (scalar)
  `-- Value : 1
|-- Name : "fiducial.dimensions"
  |-- Type : real[2] (vector)
  `-- Value[0] : 12
  `-- Value[1] : 36
|-- Name : "files.input"
  |-- Description : Input data files
  |-- Type : string[2] (vector)
  `-- Value[0] : "${DATA_DIR}/data/run_1.data.gz"
  `-- Value[1] : "${DATA_DIR}/data/run_2.data.gz"
|-- Name : "histograms.mode"
  |-- Description : Generation of histograms
  |-- Type : string (scalar)
  `-- Value : "1d+2d"
|-- Name : "number_of_threads"
  |-- Description : For multi-core host
  |-- Type : integer (scalar)
  `-- Value : 4
`-- Name : "tolerance"
  |-- Description : Length tolerance
  |-- Type : real (scalar)
  `-- Value : 1
Configuration with Bayeux/datatools

Example: configure an algorithm object (3)

```cpp
void my_analysis::initialize(const datatools::properties & setup_) {
    bool debug = false;
    bool h1d = false;
    bool h2d = false;
    double tolerance = 0.5 * CLHEP::mm;

    if (setup_.has_flag("debug")) debug = true;

    if (setup_.has_key("histograms.mode")) {
        const std::string & hmode = setup_.fetch_string("histograms.mode");
        h1d = (hmode.find("1d") != hmode.npos);
        h2d = (hmode.find("2d") != hmode.npos);
    }

    if (setup_.has_key("tolerance")) {
        tolerance = setup_.fetch_real("tolerance");
        if (! setup_.has_explicit_unit("tolerance")) {
            tolerance *= CLHEP::mm;
        }
    }

    /* ... */
}
```
Example: configure an algorithm object (4)

From a ASCII configuration file:

```plaintext
# config Configuration parameters for my data analysis
# description Debug mode activation
debug : boolean = 1
# description For multi-core host
number_of_threads : integer = 4
# description Length tolerance
tolerance : real as length = 1 mm
# description Generation of histograms
histograms.mode : string = "1d+2d"
fiducial.dimensions : real[2] in cm = 1.2 3.6
# description Input data files
files.input : string[2] as path = 
  "$DATA_DIR/data/run_1.data.gz"
  "$DATA_DIR/data/run_1.data.gz"
```

François Mauger
The Bayeux library and the SuperNEMO software
Example: configure an algorithm object (5)

Use the configuration file from C++:

```cpp
#include <datatools/properties.h>
#include <datatools/clhep_units.h>
class my_analysis {
public:
  void initialize(const datatools::properties & setup_) { /* ... */ }
  void run() { /* ... */ }
  void reset() { /* ... */ }
};

int main(void) {
  datatools::properties setup;
  datatools::properties::read_config("my_ana.conf", setup);
  if (setup.has_key("histograms.mode")) {
    std::clog << "About to run the histogram mode: '"
    << setup.fetch_string("histograms.mode") << "'") << std::endl;
  }
  if (setup.has_key("tolerance")) {
    std::clog << "... with tolerance: "
    << setup.fetch_real("tolerance") / CLHEP::um << std::endl;
  }
  my_analysis my_ana;
  my_ana.initialize(setup);
  my_ana.run();
  my_ana.reset();
  return 0;
}
```
The `datatools::multi_properties` class (1)

- A store of stores for configuration parameters (ala `.ini` format)
- A dictionary of `datatools::properties` objects (*configuration sections*)

```plaintext
#description Parameters for a processing chain
#key_label "name"
#meta_label "type"
[name="calib.calo" type="processing::calibration"]
#config Configuration parameters for the calorimeter calibration
#description Activation of the debug mode
deploy : boolean = 0
#description Calibration mode
mode : string = "calorimeter"

[name="calib.tracker" type="processing::calibration"]
#config Configuration parameters for the tracker calibration
#description Calibration mode
mode : string = "tracker"
#description Tracker calibration algorithm
tracker.algo : string = "cellular_automaton"
#description Tolerance for the tracker calibration algorithm
tracker.tolerance : real as length = 1.3 mm

[name="rec" type="processing::reconstruction"]
#config Configuration parameters for the reconstruction
mode : string = "vertex+pretracks"
```

François Mauger

The Bayeux library and the SuperNEMO software
The `datatools::multi_properties` class

- Used with object factories within a dedicated `manager` object,
- Allow dynamic instantiation of a collection of objects that are configured on-the-fly and then used by a client application,
- Such a mechanism is used in many places in Bayeux:
  - **Bayeux/datatools**: a manager of service objects
  - **Bayeux/cuts**: a manager of data selector objects
  - **Bayeux/geomtools**: a manager of geometry models objects
    (factory of logical/physical volumes)
  - **Bayeux/genvtx**: a manager of vertex generator objects
  - **Bayeux/genbb_help**: a manager of decay generator objects
  - **Bayeux/emfield**: a manager of electromagnetic field objects
  - **Bayeux/dpp**: a manager of data processing module objects
Configuration with Bayeux/datatools

The `datatools::multi_properties` class (3)

- All stages of the simulation, data processing, analysis can be addressed/configured using this mechanism,
- We can publish official configuration files as well as use ones provided by users (and mix them),
- Flexible and scalable configuration system,
- Human friendly: any configurable object/class is associated to its setup file in a comprehensive format, possibly with support for explicit physical units,
- The setup objects (`datatools::properties` and `datatools::multi_properties`) are transient and serializable,
- They can be stored in some run header, database...
- Traceability, versioning, reusability, expandable.
- The `datatools Object Configuration Description (OCD)` mechanism provides documentation for such configurable objects (ReST, HTML...).
The **Boost/Serialization** library

- A tool to store and load arbitrary objects and data structures,
- Separate the format in flavored **archives** (Input/Output, ASCII/XML/binary) and the medium (files, memory buffers...),
- Based on template methods/functions and a few utility macros,
- **Powerful features** : support natively STL containers, memory tracking, smart pointers, versioning, portability,
- Used as the base of the Bayeux I/O system.

```cpp
struct hit {
    int32_t id;
    uint16_t tdc;
    uint16_t qdc;
    template<class Archive>
    void serialization(Archive ar, unsigned int version) {
        ar & boost::serialization::make_nvp("id", id);
        ar & boost::serialization::make_nvp("tdc", tdc);
        ar & boost::serialization::make_nvp("qdc", qdc);
    }
};
```
Serialization with Bayeux/datatools

The **datatools** I/O system

- An interface for all serializable objects (`datatools::i_serializable`),
- Generic I/O reader/writer classes: `datatools::data_reader`, `datatools::data_writer`,
- Support crossed references objects through the `datatools::handle` template class (wrapper for `boost::shared_ptr`),
- Sequential storage of objects in archive, or archives in files,
- Mechanism to *register* classes for the I/O system (macros),
- Storage strategies:
  - Store several objects (possibly of different types) in one archive (needed for cross-references between objects),
  - Store one object per archive (memory tracking confinement),
- Note: the *brio* module extends the **datatools** I/O tools with random access to data files (using low level I/O system from Root).
Serialization with Bayeux/datatools

Serialize a `datatools::properties` object

```cpp
#include <datatools/properties.h>
#include <datatools/io_factory.h>
int main(void) {
    {
        datatools::properties setup;
        setup.store_flag("debug", "Debug mode activation");
        setup.store("number_of_threads", 4, "For multi-core host");
        setup.store("tolerance", 1.0, "Length tolerance");
        setup.store("histograms.mode", "1d+2d", "Generation of histograms");

        datatools::data_writer out("setup.xml"); // XML
        out.store(setup);
        datatools::data_writer out2("setup.txt"); // bzip2-compressed portable ASCII
        out2.store(setup);
        datatools::data_writer out3("setup.data.gz"); // gzip-compressed portable binary
        out3.store(setup);
    }
    {
        datatools::properties setup;

        datatools::data_reader in("setup.xml");
        in.load(setup);

        setup.tree_dump(std::cout, "My setup parameters:");
    }
    return 0;
}
```
Serialization with Bayeux/datatools

XML format: very useful for debug

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive" version="10">
<record>datatools::properties</record>
<record class_id="0" tracking_level="1" version="0" object_id="_0">
  <description></description>
  <properties class_id="1" tracking_level="0" version="0">
    <count>4</count>
    <item_version>0</item_version>
    <item class_id="2" tracking_level="0" version="0">
      <first>debug</first>
      <second class_id="3" tracking_level="0" version="0">
        <description>Debug mode activation</description>
        <flags>1</flags>
        <boolean_values>
          <count>1</count>
          <item>1</item>
        </boolean_values>
      </second>
    </item>
  </properties>
  <item>
    <first>histograms.mode</first>
    <second>
      <description>Generation of histograms</description>
    </second>
  </item>
</record>
...
Serialization with Bayeux/datatools

**Portable ASCII format:** more compact

<table>
<thead>
<tr>
<th>Portable ASCII archive format</th>
</tr>
</thead>
<tbody>
<tr>
<td>22 serialization::archive 10 21 datatools::properties 1 0</td>
</tr>
<tr>
<td>0 0 0 4 0 0 0 5 debug 0 0 21 Debug mode activation 1 1 1</td>
</tr>
<tr>
<td>15 histograms.mode 24 Generation of histograms 4 0 0 1 0 5 1d+2d</td>
</tr>
<tr>
<td>17 number_of_threads 19 For multi-core host 2 1 0 4</td>
</tr>
<tr>
<td>9 tolerance 16 Length tolerance 3 1 0 1</td>
</tr>
</tbody>
</table>

**Portable binary format (compressed) or brio**

Hum! Not human friendly display. However the best for production runs (storage/CPU).

**Feedback from a real use case**

Have been used for BiPo data storage (real, MC, reconstructed, analysis) and processing for \( \approx 8 \) years \( (\mathcal{O}(10 \text{ TB})) \).
A geometry manager class which provides

- Description of a **hierarchical setup of geometry volumes** with standard shapes, materials and arbitrary list of additional properties,
- Support for arbitrary materials and compounds (through Bayeux/materials),
- ASCII files: text based description of the geometry (no coding) thanks to registered factories of **geometry models**, 
- Primitives for basic shapes: box, cylinder... tesselated..., 
- Models with automated placement algorithms: linear and circular replicated, gridded, stacked, surrounded volumes,
- The **GID manager**: apply rules for the absolute identification of (all) geometry volumes using unique **geometry identifiers** (GID) (geomtools::geom_id class), taking into account their hierarchy (mother/daughter relationship),
- **geometry mapping**: automated construction of lookup tables of GIDs,
A geometry manager class which provides

- A *plugin system*: adding EM field manager (through Bayeux/emfields), specialized mapping and/or *locators* (GID↔placement),
- Embeddable in a *geometry service* reusable anywhere along the data processing chain: MC, reconstruction, analysis, visualization,
- Export to GDML for G4 and/or Root if needed,
- Standalone debugging and visualization tool (Gnuplot-based *geometry inspector*).
Main configuration (**datatools::properties** format)

```properties
# @description The label identifying the virtual geometry setup
setup_label : string = "MyNeutrinoExperiment"
# @description The version string of the virtual geometry setup
setup_version : string = "0.1"
# @description Files that contain geometry models’ definitions
factory.geom_files : string[4] as path =
   "$SETUP_CONFIG_DIR/geometry/models/sensors.geom"
   "$SETUP_CONFIG_DIR/geometry/models/detector.geom"
   "$SETUP_CONFIG_DIR/geometry/models/experimental_area.geom"
   "$SETUP_CONFIG_DIR/geometry/models/world.geom"
# @description The file defining geometry categories
id_mgr.categories_list : string as path =
   "$SETUP_CONFIG_DIR/geometry/categories.lis"
# @description Generation of a map of geometry IDs
build_mapping : boolean = 1
# @description Files containing geometry plugins’ definitions
plugins.configuration_files : string[1] as path =
   "$SETUP_CONFIG_DIR/geometry/plugins/materials.conf"
```

François Mauger  The Bayeux library and the SuperNEMO software
Geometry ID mapping

The *GID manager* knows the hierarchical rules for *categories* of geometry volumes:

```yaml
# @description A list of geometry ID categories/types
# @key_label  "category"
# @meta_label "type"

[category="detector.gc"  type="400"]
addresses : string[1] = "unit"  # GID=[400:0] ... [400:7]

[category="electrode.gc"  type="410"]
inherits : string = "detector.gc"  # GID=[410:0] ... [410:7]

[category="sensor.gc"  type="420"]
extends : string = "detector.gc"
by : string[1] = "position"  # GID=[420:0.0], [420:0.1] ...
```

François Mauger  The Bayeux library and the SuperNEMO software
Materials plugin

Provides informations about the materials used within the geometry setup:

```python
#@description Material geometry plugin
#@key_label "name"
#@meta_label "type"

[name="materials_driver" type="geomtools::materials_plugin"]
materials.alias_allow_overload : boolean = 1
materials.configuration_files : string[5] as path =
   "@materials:data/std_isotopes.def"  # Standard definitions
   "@materials:data/std_elements.def"  # Standard definitions
   "@materials:data/std_materials.def"  # Standard definitions
   "$SETUP_CONFIG_DIR/geometry/my_materials.def"  # User
   "$SETUP_CONFIG_DIR/geometry/material_aliases.def"  # User
```
Definitions of isotopes, elements, materials and material aliases

```plaintext
#@description Syntax for isotopes, elements...
#@key_label  "name"
#@meta_label "type"
[name="H" type="isotope"]
z : integer = 1
a : integer = 1
[name="Hydrogen" type="element"]
z : integer = 1
isotope.names : string[2] = "H" "D"
isotope.weights : real[2] = 99.9885 0.0115
[name="std::air" type="material"]
density : real = 1.2931 mg/cm3
state : string = "gas"
temperature : real = 300. kelvin
pressure : real = 1.0 bar
composition.mode : string = "fraction_mass"
composition.fraction_mass : real[2] = 0.8 0.2
```
Definitions of the geometry models: world

```ruby
#@description The geometry model of the world volume
#@key_label "name"
#@meta_label "type"
[name="world" type="geomtools::simple_shaped_model"]
shape_type : string = "box"
x : real as length = 8.0 m
y : real as length = 8.0 m
z : real as length = 20.0 m
material.ref : string = "rock" # That's underground physics!
internal_item.labels : string[1] = "area"
internal_item.model.area : string = "exp_area.model"
internal_item.placement.area : string = "0 0 0 (cm)"
```
Add daughter volumes

```grey
#@key_label "name"
#@meta_label "type"
[name="exp_area.model" type="geomtools::simple_shaped_model"]
shape_type : string = "box"
x : real as length = 5.0 m
y : real as length = 5.0 m
z : real as length = 9.0 m
material.ref : string = "low_pressure_air"
internal_item.labels : string[3] = "det0" "det1" "Gladys"
internal_item.model.det0 : string = "detector.model"
internal_item.placement.det0 : string =
  "2 0 -100 (m) / y 90 (degree)"
internal_item.model.det1 : string = "detector.model"
internal_item.placement.det1 : string =
  "-2 0 -100 (m) / y 90 (degree)"
internal_item.model.Gladys : string = "miss_jones.model"
internal_item.placement.Gladys : string =
  "0 100 -200 (cm) / z +45 (degree)"
```

François Mauger
The Bayeux library and the SuperNEMO software
Add *mapping* rules

We set some specific rules that enable the GID manager to automatically build the list of unique GIDs associated to some volumes:

```c
# Automatically generate the GID [400:0] for this volume
mapping.daughter_id.det0 : string = "[detector.gc:unit=0]"

# Automatically generate the GID [400:1] for this volume
mapping.daughter_id.det1 : string = "[detector.gc:unit=1]"
```
Add auxiliary parameters for client application

By convention, properties starting with the `visibility` prefix are related to visualization tools:

```csharp
#@description The recommended color for rendering the volume
visibility.color : string = "cyan"

#@description The visibility hidden flag for rendering
visibility.hidden : boolean = 0

#@description Another rendering hint
visibility.texture : string = "foggy_atmosphere"
```

By default, all `properties` starting with `visibility.`, `mapping.`, `material.` and `sensitive.` are preserved and stored in the `logical geometry volume` it is associated to.

Can be extended on user request, allow `plugins` and `client applications` to fetch any parameters they may need from any geometry hierarchy node.
Geometry with Bayeux/geomtools

Set of simple configuration files:

|-- manager.conf  # The main configuration file
|-- categories.conf # The definition of geometry categories
|-- models # Here we put geometry models definition files
  |-- misc.geom # Gladys’ geometry model is here!
  |-- sensor.geom
  |-- detector.geom
  |-- experimental_area.geom
  `-- world.geom
`-- plugins # Here we put plugins definition files
  |-- materials.conf # The plugin for materials
  `-- material_aliases.def # Definition of materials’ aliases
The geomtools’ inspector: a dedicated shell

shell$ bxgeomtools_inspector -c manager.conf

    GEOMTOOLS INSPECTOR
Version 4.0.0

Copyright (C) 2009-2013
Francois Mauger, Xavier Garrido, Benoit Guillon
and Ben Morgan

immediate help: type "help"
quit: type "quit"
support: Gnuplot display
support: Root display from GDML

geomtools> help
...

François Mauger  The Bayeux library and the SuperNEMO software
Geometry with Bayeux/geomtools

The geomtools’ inspector : setup display

geomtools> display -3d
Press [Enter] to continue...

geomtools> display -3d detector.model
Press [Enter] to continue...
The geomtools’ inspector: browse the hierarchy

geomtools> list_of_gids --with-type 420
List of available GIDs:
[420:0.0] as 'sensor.gc' [420:0.1] as 'sensor.gc'
[420:1.0] as 'sensor.gc' [420:1.1] as 'sensor.gc'
geomtools> display -3d [420:0.1]
Press [Enter] to continue...
The geomtools’ inspector: GDML export

```
geomtools> export_gdml
GDML file 'MyNeutrinoExperiment-0.1.gdml' has been generated!
geomtools> quit

shell$ cat MyNeutrinoExperiment-0.1.gdml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<gdml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="gdml.xsd">
    <define>
        <position name="miss_jones.model.log.stacked__0__.phys.pos"
            x="0" y="0" z="-512.5" unit="mm" />
    ...
    <volume name="world.log">
        <materialref ref="rock" />
        <solidref ref="world.log.solid" />
        <physvol>
            <volumeref ref="exp_area.model.log" />
            <positionref ref="world.log.area.phys.pos" />
        </physvol>
    </volume>
    </structure>
    <setup name="Setup" version="1.0">
        <world ref="world.log" />
    </setup>
</gdml>
```
Do more with `geomtools`:

- SuperNEMO demonstrator’s virtual geometry:
  - \( O(10^5) \) volumes, each with its unique GID (mapping, on user request),
  - use mainly generic primitives and only a few hardcoded geometry models for complex shapes (example: OM’s extracted light guides),
  - used as Geant4 geometry input through the GDML automated interface,
  - used by the SuperNEMO event visualization program (see Falaise),
Do more with `geomtools`:

- **SuperNEMO demonstrator’s virtual geometry:**

 ![Diagram](image)

- used by the vertex generation manager: because in DBD physics, all elements in the geometry are susceptible to host radioactivity ($\beta\beta$ source foil strips, tracker field wires, scintillator block wrapper films, screws, bolts...) and thus generate background,

- used by event reconstruction and analysis processing modules through a dedicated geometry service.
Simulation with Bayeux

What modules are involved?

To build a simulation framework:

- **g)** `geomtools`: geometry modelling,
- **v)** `genvtx`: generation of primary vertex,
- **k)** `genbb_help`: generation of the primary decay/kinematics,
- **s)** `mctools` and its `Geant4` plugin: a simulation driver that integrates the functionalities found in **g)**, **v)** and **k)**.

- Design, implementation, tests and validation of **g)**, **v)** and **k)** are done independently of `Geant4`,
- Flexible (friendly ASCII config files using the `datatools` configuration interface), expandable (support for user plugin at every levels),
- In principle reusable in other context (another MC framework?).
Simulation with Bayeux

### Vertex generation with `genvtx`

- **Primitives for vertex generation** from usual shapes of interest: boxes, cylinders, tubes, polycones... in bulk volume, on surfaces, with arbitrary shift and skin thickness,

- A **vertex generation manager** interfaced with the **geometry manager** for automated instantiation of vertex generators addressed by the **geometry mapping** mechanism,

- Configured with simple ASCII files (`datatools : :multi_properties`): no C++ coding is needed for common shapes (primitives already available),

- Specific needs can be addressed with **vertex generator plugins** using the `genvtx` API (class registration, factories). This plugin may be hardcoded or, better, use special primitives for vertex generation and geometry modelling,

- Possibilities to **combine weighted vertex generators** with their own activities (absolute, surface, volume) expressed in common units (Bq/m², Bq/m³, mBq/kg...).
Simulation with Bayeux

François Mauger

The Bayeux library and the SuperNEMO software
Typical configuration file for automated vertex generators

```plaintext
@description Definitions of some vertex generators
@key_label "name"
@meta_label "type"
[name="from_the_outside.vg" type="genvtx::box_model_vg"]
origin : string = " category='world' " # This rule matches the world volume
mode : string = "surface"
mode.surface.back : boolean = 1
mode.surface.front : boolean = 1
mode.surface.bottom : boolean = 1
mode.surface.top : boolean = 1
mode.surface.left : boolean = 1
mode.surface.right : boolean = 1
[name="electrode_det0_bulk.vg" type="genvtx::cylinder_model_vg"]
# The following rule matches the electrode of detector number 0
origin : string = " category='electrode.gc' unit=0"
mode : string = "bulk"
[name="pixels_det0_row1_surface.vg" type="genvtx::box_model_vg"]
# The following rule matches the pixels in rows 1 of
# detector number 0 for sensors on both sides
origin : string = " category='pixel.gc' unit=1 position=* column=* row=1 "
mode : string = "surface"
mode.surface.top : boolean = 1
[name="pixels_det1_pos0_col3_bulk.vg" type="genvtx::box_model_vg"]
# The following rule matches the pixels in column 3 of
# detector number 1 for sensors on side 0
origin : string = " category='pixel.gc' unit=1 position=0 column=3 row=* "
mode : string = "bulk"
```

François Mauger

The Bayeux library and the SuperNEMO software
Simulation with Bayeux

**Generation of particles/decays with genbb_help**

- A C++ port of the well known Decay0/Genbb Fortran program by Vladimir Tretiak,

- Implement 18 $\beta\beta$ processes for 38 isotopes of interest from $^{48}\text{Ca}$ to $^{198}\text{Pt}$,

- Implement 60 unstable radionucleides (with delayed decay cascades) from $^{14}\text{C}$ to $^{241}\text{Am}$ (background, calibration sources),

- Generic data model: serializable `genbb::primaryParticle` and `genbb::primaryEvent` classes,

- An API for extensions with additional generators (plugins, factories).

- Possibilities to *combine weighted event generators* (example: elaborate a *background model* for a given material),

- A *manager* class with ASCII files: again, no C++ coding in most use cases.
Simulation with Bayeux

The **bxgenbb_inspector** tool

- An application to generate and browse the output from event generators,
- Generate multiplicity and energy histograms for generated particles,
- **Example:**

```bash
shell$ bxgenbb_inspector \
   --configuration "@genbb_help:manager/config/pro-1.0/manager.conf" \
   --action "shoot" \
   --generator "Bi214_Po214" \
   --prng-seed 314159 \
   --number-of-events 100000 \n   --modulo 1000 \n   --histo-def "@genbb_help:inspector/config/le_nuphy-1.0/inspector_histos_prompt.conf" \n   --histo-def "@genbb_help:inspector/config/le_nuphy-1.0/inspector_histos_delayed.conf" \n   --prompt-time-limit 1 \n   --prompt \n   --delayed \n   --title-prefix "Bi214-Po214" \n   --output-file "histos_Bi214_Po214.root"
```

[notice:void genbb::inspector::_run_batch():1620] Generated event #1
[notice:void genbb::inspector::_run_batch():1620] Generated event #1000
[notice:void genbb::inspector::_run_batch():1620] Generated event #2000
...
[notice:void genbb::inspector::_run_batch():1620] Generated event #100000

François Mauger

The Bayeux library and the SuperNEMO software
Simulation with Bayeux

The Bayeux library and the SuperNEMO software
Simulation with Bayeux

*Musical interlude*: Xavier Garrido singing his favorite song...
Now we are ready for the simulation!

- A MC driver wrapping a full Geant4 session manager, with many supported G4 features (GDML, visualization, physics lists, regions, user limits, secondary particle handling)
- Running mode: batch, interactive, pipeline (parallel MC thread)...
- Automated instantiation of sensitive volumes on user request (factories),
- Provides an API for truth hits post-processing: pre-digitization and data storage reduction,
- Provides a few useful truth hits post-processors algorithms: calorimetry, garbage, visualization,
Simulation with Bayeux

Now we are ready for the simulation!

- **Generic output data models:** `mctools::base_step_hit` (truth hits from G4) and `mctools::simulated_data` serializable classes,

- Generate collections of truth hits for each sensitive part of the detector:
  - *Official* MC hits from real sensitive detectors (calorimeter, tracking fiducial volume)
  - Debug/visualization MC hits from non-sensitive detectors (e.g., mechanics structures that are not sensors)

- Choose the level of details of the truth data output: realistic (production), truth hits (≡G4Step) from everywhere (debug) at the price of more CPU and storage.
Simulation with Bayeux

- Use the Bayeux’s text-based UI interface: no C++ coding unless we must add specialized user plugins using the API and its factory registration mechanism.

- The simulated output, whatever the simulated experiment is, always uses the same data model and `datatools` I/O system.

- Easy to read the output file and process the objects in it (digitization, calibration...), easy to translate in some other formats.

- Provide the `bxg4_production` program to run any MC sessions, whatever the experiment is (SuperNEMO demonstrator, BiPo3, HPGe...).

- Provide a `simulation module` compatible with the processing pipeline implemented in the `dpp` library.
The output we want:

- **Simulated data model**
  - **Vertex** \((x, y, z)\)
  - **Primary event**
    - **Primary particle** (name/type, time, \(\vec{p}\))
    - **Primary particle** (name/type, time, \(\vec{p}\))
    - **Primary particle** (name/type, time, \(\vec{p}\))
  - **Properties**
  - **Collections of MC hits**

- **Official production**
  - "calo"
  - "gg"

- **For debugging purpose**
  - "__visu.tracks"

Scintillation clusters

Geiger avalanches
How we build the output:

- **Geometry hierarchy**
  - top volume
  - sensitive volume 1
  - sensitive volume 2
  - some volume
    - sensitive volume 3
    - sensitive volume 4
- **Collections of sensitive step hits**
  - sensitive category "A"
  - sensitive category "B"
  - sensitive category "C"
  - sensitive category "D"
- **Step hit post-processing**
  - post-processor $P_1$ (kill all)
  - post-processor $P_2$
  - post-processor $P_3$ (push all)
  - post-processor $P_4$
  - post-processor $P_{5\ldots}$
- **Collections of MC hits**
  - hit category $H_1$
  - hit category $H_2$
  - hit category $H_3$
  - Storage
Simulation with Bayeux

A full configuration setup for a simulation:

```
MyNeutrinoExperiment/  # A typical configuration tree
|-- geometry
   |-- 0.1
      |-- manager.conf  # Setup the geometry
      |-- categories.lis # Define the geometry categories of volumes
      |-- models
         |-- source.geom  # The description of the source
         |-- detector.geom # The description of the detector
         |-- world.geom   # The top volume
      `-- plugins
         |-- mappings.conf  # Mapping plugin
         |-- materials.conf # Materials plugin
         `-- material_aliases.def # Definition of materials’ aliases
`-- simulation
   |-- geant4_control
      |-- 0.1
         |-- manager.conf  # Setup the G4 driver
         |-- processes
         |-- em.conf       # Electromagnetic processes
         `-- particles.conf # List of particles
            `-- step_hit_processors.conf # Post-processing of true hits
   |-- primary_events
      |-- 0.1
         |-- manager.conf  # Setup the vertex generation
         |-- dbd.def       # Definitions of DBD decay generators
         `-- background.def # Definitions of background decay generators
`-- vertexes
   `-- 0.1
      |-- manager.conf  # Setup the vertex generation
      `-- generators.def # Definitions of vertex generators
```
mctools' example ex00

A simple geometry: an optical module ala BiPo/SuperNEMO
mctools’ example ex00

Running Geant4 in interactive mode:

```
$ bxg4_production \
   --logging-priority "warning" \
   --number-of-events-modulo 1 \
   --interactive \
   --g4-visu \
   --config "manager.conf" \
   --vertex-generator-name "source_bulk.vg" \
   --vertex-generator-seed 0 \
   --event-generator-name "electron_1MeV_cone" \
   --event-generator-seed 0 \
   --shpf-seed 0 \
   --g4-manager-seed 0 \
   --output-prng-seeds-file "prng_seeds.save" \
   --output-prng-states-file "prng_states.save" \
   --output-profiles "all_details" \
   --output-data-file "mctools_ex00_electron_1MeV_source_bulk.xml" \
   --g4-macro "geant4_visualization.mac"
```

`...`

```
$Idle> /run/beamOn 1  # Fire the G4 gun
```

...
mcTools' example ex00

Geant4 display: a low-energy electron from the source film, backscattered on the attenuator plate in front of the scintillator block, and finally leaving the world volume.
mctools' example ex00

Geant4 display: many electrons with some magnetic field activated
Simulation with Bayeux

mctools' example ex00

Off-line display: sensitive hits in the scintillator block

Energy deposit in the scintillator block
mctools' example ex00

Post-processing of *truth sensitive hits* to build an *output hit* of the "*scin*" category: reduce to useful information for further digitization processing

A dedicated step hit processor clusterizes all the truth step hits from G4 and build a single output calorimeter hit.
Simulation with Bayeux

mctools' example ex00

Output data model: use plain C++ classes with serialization support

```
Simulated data:
|-- Properties: <empty>
|   `-- <no property>
|-- Collection type: 1
|-- Collections of step hit handles:
|   |-- Category '_visu.tracks' has 75 hit(s) [capacity=75]
|   |-- Category 'scin' has 1 hit(s) [capacity=1]
|-- Primary event:
|   |-- Valid: 1
|   |-- Label: ''
|   |-- Time: 0 s
|   |-- Particles: [1]
|     `-- Particle #0:
|       |-- Type: 3 (e-)
|       |-- Particle label: 'e-'
|       |-- Time: 0 ns
|       |-- Kinetic energy: 1 MeV
|       |-- Momentum: (-1.41112, 0.0107192, 0.174975) MeV
|       |-- Vertex: <no vertex>
|       |-- GENBB weight: 1
|       |-- Classification: '1e0p0g0a0X'
|       `-- Vertex: (199.999, -4.92332, -6.91926) mm
```
**Simulation with Bayeux**

### mctools' example ex00

**Raw truth hits of the "__visu.track" category**

<table>
<thead>
<tr>
<th>Truth '__visu tracks' hit:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Official calorimeter truth hits of the "scin" category**

<table>
<thead>
<tr>
<th>Truth 'scin' hit:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Simulation with Bayeux

mctools’ example ex00

- Dedicated configuration parameters for each sensitive detector,
- Ability to choose the volumes from where we’d like to record truth hits and G4 tracking informations (by names, by materials, by geometry categories),
- Ability to choose the level of details (G4 tracking informations) to be saved in a truth hits from within a given sensitive category,
- The truth hit data model uses a versatile/generic container embedded in the object: datatools::properties
Conclusion

- Not a full featured Geant4 driver, however it has fulfilled all our needs so far,
- Simplify/fasten the making of an simulation environment,
- We can add new functionalities when needed,
- Fully integrated in the Bayeux framework : UI, configuration, data model and I/O, so our users are familiar with the approach,
- These tools has been used for years for SuperNEMO activities.
A generic data processing pipeline

- Based on data processing modules,
- Can chain them and organize them along a data pipeline,
- A manager class, module class registration and factories,
- Configuration as usual (see Bayeurx/datatools::multi_properties),
- Provides primitive modules: chain, conditional module, I/O modules...
- Interfaced with the datatools support for services.
- Ability to distribute an official chain of data processing steps through a set of blessed configuration files and modules,
- User can develop and insert their own algorithms along the pipeline,
- An unique program to address any processing chain: bxdpp_processing

Data processing with Bayeurx/dpp
Data processing with Bayeux/dpp

François Mauger
The Bayeux library and the SuperNEMO software
The `datatools::things` class

- A container able to store arbitrary *serializable objects of any type*: data banks,
- Used template, RTTI, registered factories,
- Used as a dictionary with unique string keys to access the data banks,
- Data banks can be added, removed, transformed,
- Can store several versions of a type of objects: "CD1": calibrated data method 1, "CD2": calibrated data method 2, both of type `myexperiment::calibrated_data`
- Support cross-referenced data structures in different data banks (example: the *digitized tracker hits* in the *calibrated data bank* point to some of the *raw MC truth hits* in some *simulated data bank*)
- Ideal candidate class to be used as the data record processed by the data processing pipeline,
- I/O (load/store from Boost archives) is supported by Bayeux serialization system (standard reader/writer classes).
Data processing with Bayeux/dpp

How is organized the data processing

1. Decide for a set of registered data processing modules (eventually load some dedicated plugins for that),
2. Provide the configuration parameters for each of them,
3. Ask the \texttt{bxdpp\_processing} program (or some \texttt{dpp::manager object}) to run and loop on some collection of data records (\texttt{datatools::things}),
4. Enjoy the output!
**Conclusion**

- Bayeux provides an unique framework for various activities within the collaboration,
- Bayeux is built on top of standard features of the C++ language and uses some third party software (automatically managed by the Cadfael aggregator package),
- Has solved many SW technical issues and provides generic tools,
- Bayeux (prototype&beta) has been successfully used during the last 5 years,
- It is now ready for a production release : version 1.0, next week,
- A living project : additional features are planed for future releases,
Conclusion

- Other projects are now starting to use it:
  - Studies at LPC: nuclear safety and radiation protection (under industrial contract),
  - LPCTrap experiment and collaborators: search for weak exotic coupling in $\beta$ decay,
  - Effects of radiation in semiconductor chip structures (SINUS group, with IM2NP Marseille),
- Probably of educational interest: for nuclear/particle physics lab teaching,
- Feel free to make a try!
Dedicated to SuperNEMO

- Built on top of Bayeux (also use SVN, CMake),
- Implement a dedicated C++ library for simulation, data models, data processing,
- Published versioned resource files: configuration for geometry, MC... keep track of former MC/processing setup through ASCII files,
- flsimulate application for MC production:
  - SuperNEMO demonstrator,
  - Tracker commissioning,
  - BiPo3.
- f1reconstruct application for pipelined data reconstruction:
  - blessed modules and algorithms,
  - plugin system for user contribution.
- SuperNEMO and BiPo event browsers.
What we have now

- Simulation module,
- Mock digitization/calibration module,
- Tracker clustering module,
- Track fitting module,
- Track reconstruction module,
- Geometry and event visualization,
- Ready for analysis and studies...
MC background from the source foils ($^{214}$Bi, $^{40}$K):

- $E = 0.88 \pm 0.03$ MeV
  - $t = 290828.69 \pm 0.22$ ns

- $E = 0.47 \pm 0.02$ MeV
  - $t = 4.00 \pm 0.30$ ns

- $E = 0.25 \pm 0.02$ MeV
  - $t = 2.01 \pm 0.41$ ns

- $E = 0.10 \pm 0.01$ MeV
  - $t = 1.87 \pm 0.63$ ns
Conclusion

- Falaise relies on generic functionalities provided by the Bayeux library,
- Concentrate on specific problems for the SuperNEMO experiment,
- Falaise 1.0 will be released end of June,
- Will be improved in the future:
  - Final geometry design: final layout of the calorimeter, shielding,
  - Realistic modelling of the magnetic field,
  - Final raw data model(s) and realistic digitization (waiting for the finalization of the front end electronics design, trigger strategies and DAQ system...)
  - More data reconstruction and analysis tools,
  - Interface to a database system and the control and monitoring system.
General conclusion

- A quick tour of the SW tools used by the SuperNEMO collaboration,
- Based on Cadfael/Bayeux/Falaise packages,
- We can now investigate a lot of physics cases with these tools,
- Bayeux could be useful for many other projects,
- Contact us to investigate/use these tools.
General conclusion

Thanks to

- Core SW: X. Garrido, B. Morgan, B. Guillon, M. Bongrand...
- SN plugins: X. Garrido, F. Nova, S. Calvez...
- Visualization: X. Garrido.
- Songs (particularly confined in a bus): X. Garrido.