GDML - recent developments

Witek Pokorski

29.11.2004

Outline

- Some background info
- GDML Schema
- GDML readers/writers
- Some examples
- Conclusion

GDML - historical background...

- GDML stands for Geometry Description Markup Language
- first appearance around year 2000 (?)
  - activity started by Radovan Chytracek
    - until ~May 2004 all work done by Radovan
  - initial frame: XML based geometry description for Geant4
  - motivation:
    - move away from hardcoded geometry
    - enable geometry interchange between different applications
- discussed at 'Geometry Description' *Requirement Technical Assessment Group* (RTAG), became 'LCG-supported' project in October 2003
- presently part of the Simulation Framework subproject (Simulation Project), as the geometry interchange format workpackage

GDML - ideological background…

- purpose of GDML is to describe data
  - to dump geometry data
  - not procedural, but markup language
- format has to be application independent
  - there is nothing more universal than an ASCII file
- 'human-readability' is a big advantage
  - there is nothing more readable than an ASCII file…
- should be easily extensible and modular

GDML designed as an application of XML

GDML - technical background…

- GDML is defined through XML Schema (XSD)
    - XSD = XML based alternative to Document Type Definition (DTD)
    - defines document structure and the list of legal elements
    - XSD are in XML -> they are extensible
  - valid GDML document conforms to GDML schema
- GDML would be useless without I/O…
  - C++ implementation of GDML processor was started (by Radovan) in parallel to GDML Schema
    - allows writing-out GDML data to a stream
    - uses Xerces-C SAX parser to create 'in-memory' representation of the geometry description
    - allows easy bindings to different applications (Geant4, ROOT)
  - alternative Python-based processing architecture under development
    - uses xml.sax Python module
    - allows very light binding to applications

GDML Schema

GDML Schema - structure (1/2)

- located in $GDML_BASE/GDMLSchema/
- top level file: gdml_X.Y.xsd
  - defines the general structure of GDML document
    - specifies the geometry tree implementation
  - includes
    - gdml_simple_core.xsd (core types - vectors, etc)
    - gdml_simple_defines.xsd (constants, positions, rotations, etc)
    - gdml_simple_materials.xsd (elements, isotopes, materials, etc)
    - gdml_simple_solids.xsd (all supported solids)
    - gdml_simple_replicas.xsd (replicas, divisions, etc)
    - gdml_simple_parameterised.xsd (parameterised volumes)

GDML Schema - structure (2/2)

```
<?xml version="1.0" encoding="UTF-8"?>
<gdml xsi:noNamespaceSchemaLocation="GDMLSchema/gdml_2.0.xsd">
    <define>
            …
            <position name="TrackerinWorldpos" unit="mm" x="0" y="0" z="100" />
    </define>
    <materials>
            …
            <material formula=" " name="Air" >
                    <D value="1.290" unit="mg/cm3"/>
                    <fraction n="0.7" ref="Nitrogen" />
                    <fraction n="0.3" ref="Oxygen" />
            </material>
    </materials>
    <solids>
            …
            <box lunit="mm" name="Tracker" x="50" y="50" z="50" />
    </solids>
    <structure>
            …
            <volume name="World" >
                    <materialref ref="Air" />
                    <solidref ref="world" />
                    <physvol>
                            <volumeref ref="Tracker" />
                            <positionref ref="TrackerinWorldpos" />
                            <rotationref ref="TrackerinWorldrot" />
                    </physvol>
            </volume>
    </structure>
    <setup name="Default" version="1.0" >
            <world ref="World" />
    </setup>
</gdml>
```

GDML Schema - status

- about to release GDML_2.0.0
  - core, defines, materials schema - complete
  - solids supported:
    - box, sphere (G4Sphere and G4Orb), tube, cone, polycone, parallepiped, trapezoid (G4Trap and G4Trd), torus, polyhedra, hype (tube with hyperbolic profile), elliptical tube
    - boolean solids:
      - union, subtraction, intersection
  - assembly volumes supported
  - replicas and divisions (on their way)
  - parameterised volumes (position, rotation and size)
    - gradually adding parameterisation capability for all the solids

Parameterised volumes in GDML

- how could we persistify parameterisation algorithms?
  - the only (?) way: to dump the parameters as a table
    - we cannot 'guess' the form of the function, we can only dump the values
  - while reading back GDML we instantiate 'tabularised' parameterisation algorithm
    - although the 'algorithm' changes with respect to the original one the resulting parameterised volume should be identical
  - this is in the context of GDML as exchange/persistency format
    - to use GDML for geometry implementation ('by hand') some 'predefined' algorithms will be provided (for ex. linear)

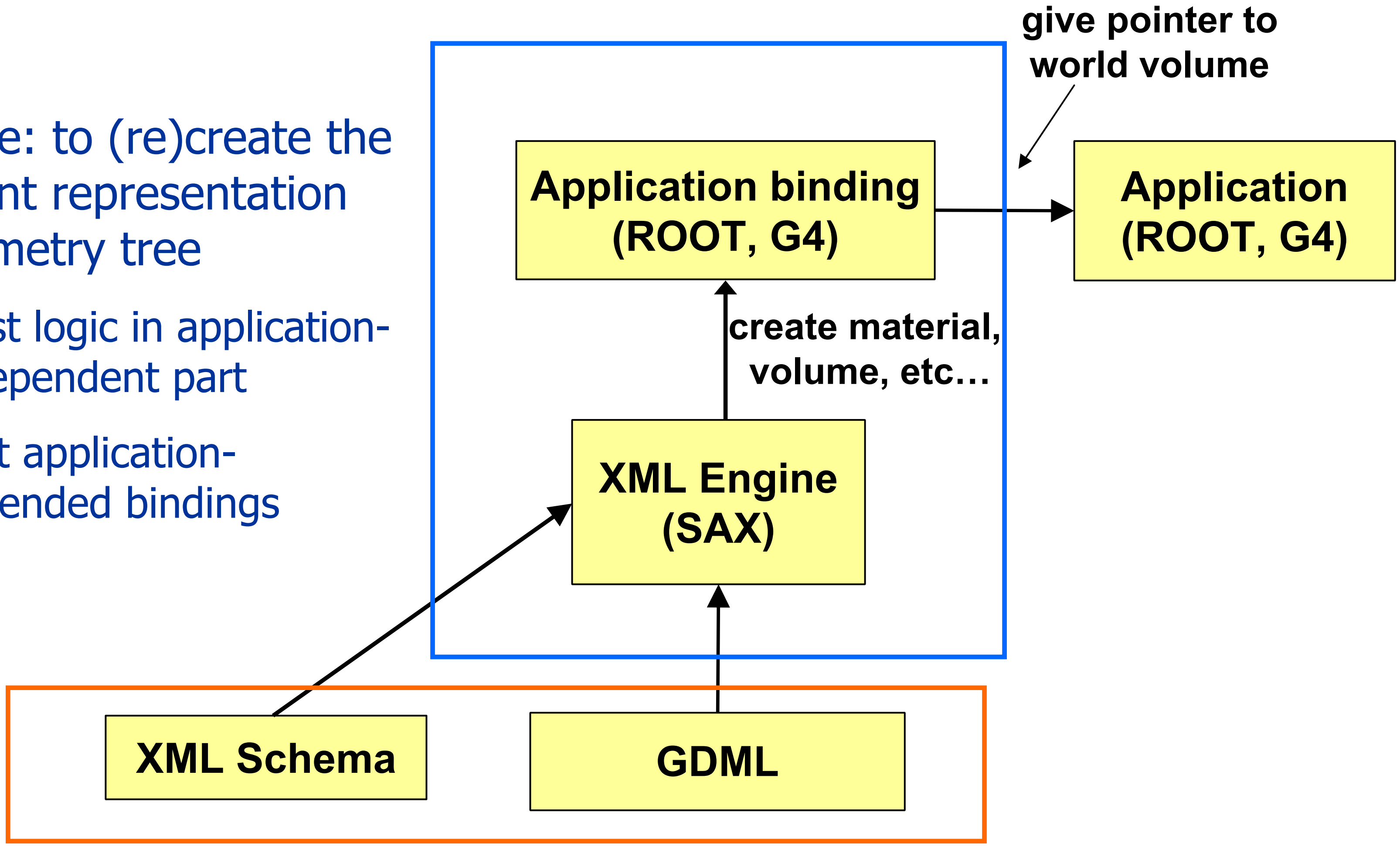
Parameterisation example

```
<volume name="Tracker" >
        <materialref ref="Air" />
        <solidref ref="tracker" />
        <paramvol ncopies="5" >
                <volumeref ref="Chamber" />
                <parameterised_position_size>
                        <parameters number="1" >
                                <position name="copy1pos" x="0" y="0" z="-700" />
                                <box_dimensions x="672" y="672" z="100" />
                        </parameters>
                        <parameters number="2" >
                                <position name="copy2pos" x="0" y="0" z="100" />
                                <box_dimensions x="1104" y="1104" z="100" />
                        </parameters>
                        <parameters number="3" >
                                <position name="copy3pos" x="0" y="0" z="900" />
                                <box_dimensions x="1536" y="1536" z="100" />
                        </parameters>
                        <parameters number="4" >
                                <position name="copy4pos" x="0" y="0" z="1700" />
                                <box_dimensions x="1968" y="1968" z="100" />
                        </parameters>
                        <parameters number="5" >
                                <position name="copy5pos" x="0" y="0" z="2500" />
                                <box_dimensions x="2400" y="2400" z="100" />
                        </parameters>
                </parameterised_position_size>
        </paramvol>
</volume>
```

GDML Readers/Writers

GDML reader - structure

- purpose: to (re)create the transient representation of geometry tree
  - most logic in application-independent part
  - light application-depended bindings


give pointer to world volume
Application binding (ROOT, G4)
Application (ROOT, G4)
create material, volume, etc…
XML Engine (SAX)
XML Schema
GDML

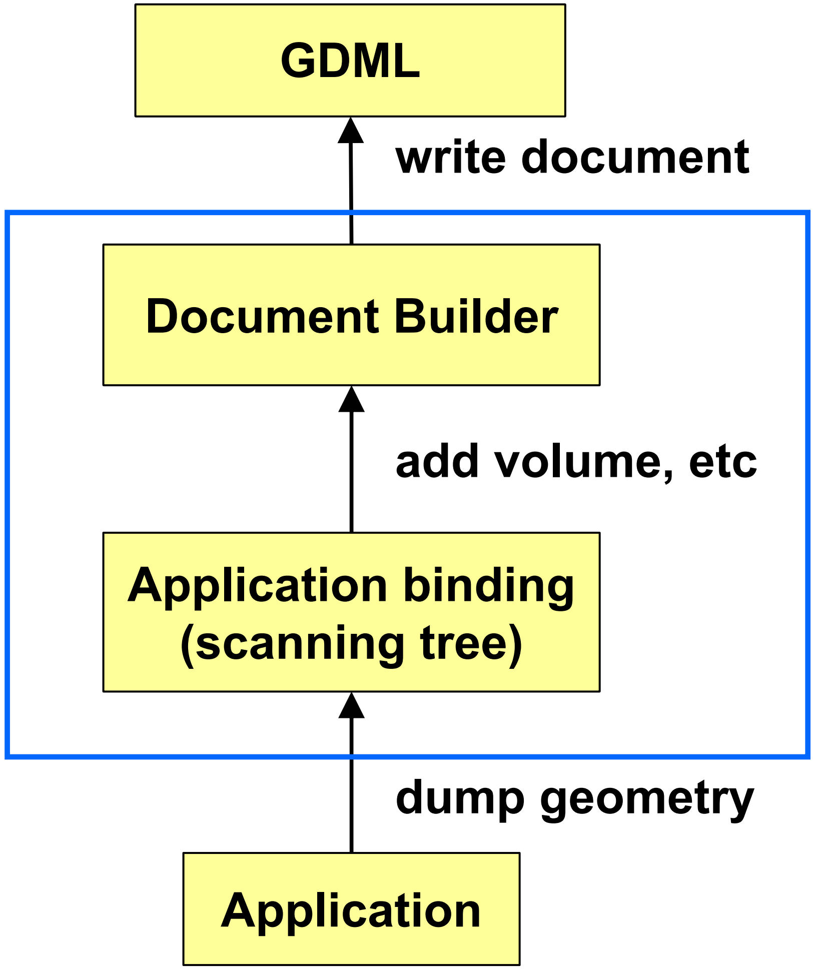
GDML reader - status

- C++ implementation:
  - application-independent part complete for the present schema
  - application-dependent part:
    - complete for Geant4
    - on its way for ROOT…
  - runs on rh73_gcc323, Cygwin, Mac OS X

- Python implementation:
  - see next slides…

GDML writer - structure

- purpose: to persistify the geometry description in the form of GDML file
  - application independent part generating XML
    - 'cursors' for materials, solids, structure, etc
  - 'light' application dependent bindings
    - scanning the geometry tree and adding elements to the 'cursors'



GDML writer - status

- C++ implementation:
  - application-independent part complete for the present schema
  - application-dependent part:
    - complete for Geant4
    - on its way for ROOT…
  - runs on rh73_gcc323, Cygwin, Mac OS X

- Python implementation:
  - next item on my 'to do' list…

GDML - example use (G4)

to write:

```
#include "WriterG4/G4GDMLWriter.h"

G4GDMLWriter g4writer("GDMLSchema/gdml_2.0.xsd", "g4test.gdml");

try
{
        g4writer.DumpGeometryInfo(g4worldvolume);
}
catch(std::logic_error &lerr)
{
        std::cout << "Caught an exception: " << lerr.what () << std::endl;
}
```

to read:

```
SAXProcessor sxp;
sxp.Initialize();
ProcessingConfigurator config;
config.SetURI( "g4test.gdml" );
config.SetSetupName( "Default" );
sxp.Configure( &config );

sxp.Run()
fWorld =  (G4VPhysicalVolume *)GDMLProcessor::GetInstance()->GetWorldVolume();
```

GDML processing - performance

- GDML G4reader/G4writer (C++) tested on
  - complete LHCb geometry
  - parts of ATLAS geometry
    - problem with full ATLAS geometry - use of custom solids
- for LHCb geometry (~5000 single placements, ~20 million 'real' vols.)
  - writing out ~10 seconds (on P4 2.4GHz)
  - reading in ~ 5 seconds
  - file size ~2.7 Mb (~40k lines)
- also successfully tested G4->GDML->G4->ROOT
  - for G4->ROOT, converter by Ivana Hrivnacova used

GDML reader/writer revisited

- Python - an interesting alternative to C++ for implementing the GDML processing code
  - dealing with XML in Python much easier (less code needed)
  - Python very good for 'glueing' different applications together
  - very easy interaction with C++ objects through:
    - PyROOT for ROOT classes
    - LCGDict/PyLCGDict for Geant4 classes (or any other C++ classes)

PyGDML - status

- first implementation of GDML->ROOT reader ready
  - uses xml.sax Python module
  - uses PyROOT for accessing ROOT classes
  - application-specific part very small (~150 lines of Python)
    - Geant4 binding ready in a day or two…
- works for full LHCb geometry
  - GDML file -> 'in-memory' ROOT TGeo
- next task: ROOT->GDML writer

PyGDML - example (ROOT)

```
import xml.sax
import ROOT
import GDMLContentHandler

ROOT.gSystem.Load("libGeom")
geomgr = ROOT.TGeoManager("World","GDMLGeo")

gdmlhandler = GDMLContentHandler.GDMLContentHandler()
xml.sax.parse('test.gdml',gdmlhandler)

geomgr.SetTopVolume(gdmlhandler.WorldVolume())
geomgr.CloseGeometry()

gdmlhandler.WorldVolume().Draw()
```

Python SAX parser

PyROOT

GDML-specific parser extension

Standard TGeo

GDML parsing

get world volume from GDMLContentHandler

Example use case

- someone provides:
  1. testbeam geometry exported to GDML
  2. dictionary for sensitive detector implementation created with LCGDict (so one can instantiate it in Python and associate it to the specific volume)
- someone else uses it to:
  1. load geometry in Python
  2. run Geant4 in Python
     - this comes almost for free using PyLCGDict/LCGDict
  3. run other simulation using VMC (?)
  4. plot different distributions with ROOT (using PyROOT)
  5. and/or check for overlaps in geometry with ROOT

Conclusions

- there is no doubt about the need to have a geometry exchange format
- GDML - good candidate
  - universal format (ASCII…)
  - human-readable
  - extensible
- interest in GDML from many places
  - motivating and proving usefulness
- Python interfacing provides flexibility
- high priority given to GDML in LCG Simulation Framework subproject
  - development of Geant4 and ROOT bindings will continue with regular releases