Geometry Description Markup Language (GDML)

Gabriele Cosmo, Andrei Gheata, Witek Pokorski

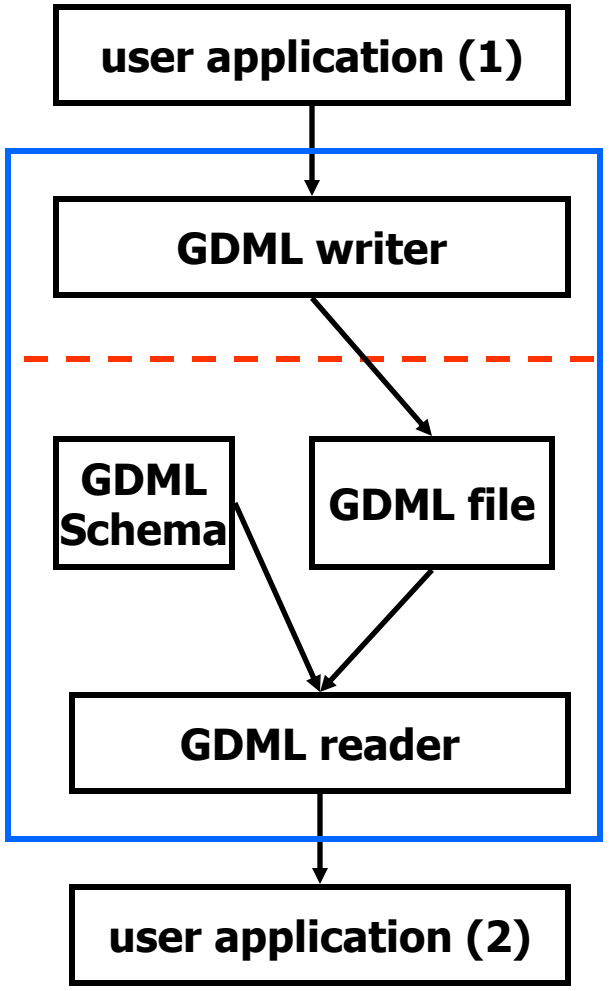
07.06.2021

GDML - Motivation

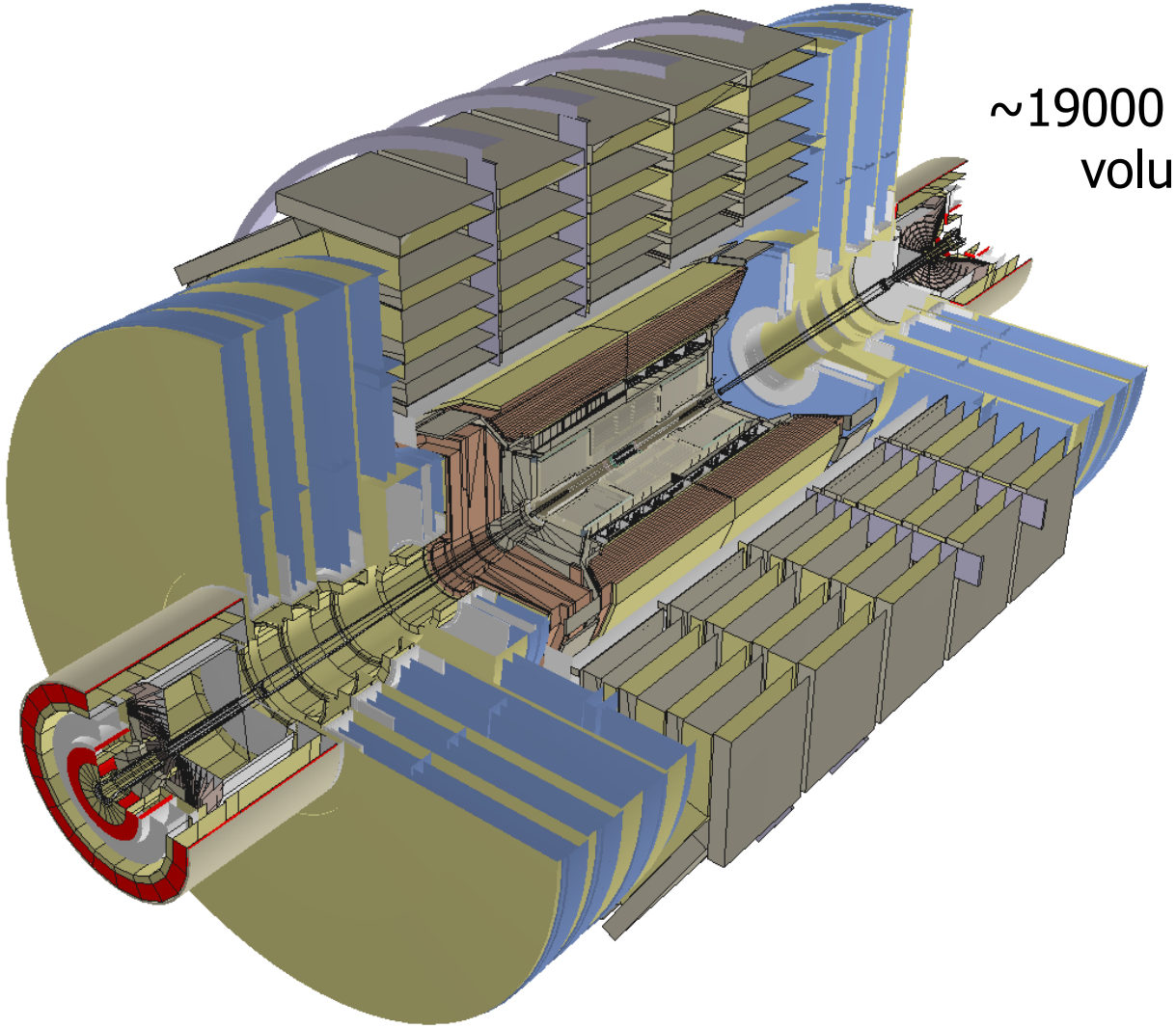
- simulation toolkits come with their native geometry description formats
  - many (most?) of the users do not implement geometry in those formats
- users use their own geometry description formats providing more flexibility, but:
  - they are integral parts of experiment software frameworks
  - cannot be easily exported in application independent way
- GDML has been developed (~15 years ago)
  - to have an application independent and flexible geometry format
  - to be able to interchange geometry between different applications for the purpose of
    - physics validation/comparison, visualization, debugging

GDML components

- 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
- GDML can be written by hand or generated automatically
  - 'GDML writer' allows writing-out GDML file
- GDML needs a 'reader'
  - 'GDML reader' creates 'in-memory' representation of the geometry description



CMS detector: G4->GDML->ROOT



~19000 physical volumes

GDML document – core parts

positions, rotations

materials

solids

geometry tree

'world' volume

```
<?xml version="1.0" encoding="UTF-8"?>
<gdml xsi:noNamespaceSchemaLocation="GDMLSchema/gdml.xsd">
    <define>
        …
        <position name="TrackerinWorldpos" unit="mm" x="0" y="0" z="100"/>
    </define>
    <materials>
        …
        <element name="Nitrogen" formula="N"  Z="7.">
        <atom value="14.01"/>  </element>
        <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>
```

Auxiliary information

- allows to embed arbitrary user-defined information
  - (lists of) structs containing (type, value, [unit])
    - can contain several levels
      - an auxiliary information can have its sub-information
  - can be per volume
    - sensitive detector, visualisation color
  - or global
    - regions, cuts
- up to the user's code to use the auxiliary information in the program

```
<volume name="Boxvol" >
       <materialref ref="Air" />
       <solidref ref="Box" />
       <auxiliary auxtype="SensDet" auxvalue="veloSD"/>

       <auxiliary auxtype="sometype" auxvalue="somevalue">
        <auxiliary auxtype="somesubtype" auxvalue="somesubvalue"/>
       </auxiliary>
</volume>


<userinfo>
     <auxiliary auxtype="Region" auxvalue="myregion2">
        <auxiliary auxtype="RootLogicalVolume" auxvalue="myvol"/>
        <auxiliary auxtype="pcut" auxvalue="2.2" auxunit="mm"/>
        <auxiliary auxtype="ecut" auxvalue="1.5" auxunit="mm"/>
     </auxiliary>

     <auxiliary auxtype="ulimits" auxvalue="electron">
        <auxiliary auxtype="ustepMax" auxvalue="5" auxunit="mm"/>
        <auxiliary auxtype="utrakMax" auxvalue="5" auxunit="cm"/>
        <auxiliary auxtype="uekinMin" auxvalue="900" auxunit="keV"/>
     </auxiliary>
</userinfo>
```

Using auxiliary information

- Accessing auxiliary information per volume

```
G4GDMLAuxListType auxInfo =
parser.GetVolumeAuxiliaryInformation(*lvolume);
```

- Accessing global auxiliary information

```
G4GDMLAuxListType auxInfoGlobal =
parser.GetAuxList();
```

```
struct G4GDMLAuxStructType
{
 G4String type;
 G4String value;
 G4String unit;
 std::vector<G4GDMLAuxStructType>* auxList;
};

using G4GDMLAuxListType =
std::vector<G4GDMLAuxStructType>;
```

Loops, matrices

defining matrix

declaring variables and constants

using loop to create several boxes

using loop to create volumes out of the boxes above

using loop to place those volumes

using elements from the matrix for coordinates

```
<define>
        <matrix name="m" coldim="5" values="0 4.25 8.0 11.25 14
                                            5 4    3.2  2.56 2.048" />
        <variable name="i" value="0" />
        <variable name="num" value="5" />
</define>

<solids>
...
        <loop for="i" from="1" to="num" step="1">
            <box name="box[i+1]" x="10-i" y="5-i/2" z="m[2,i]" />
        </loop>
</solids>

<structure>
        <loop for="i" from="1" to="num" step="1">
            <volume name="volbox[i+1]">
                <materialref ref="iron" />
                <solidref ref="box[i+1]" />
            </volume>
        </loop>

        <volume name="world">
        ...
            <loop for="i" from="1" to="num" step="1">
                <physvol>
                    <volumeref ref="volbox[i+1]" />
                    <position name="pos" x="5" y="m[1,i]" z="0"/>
                </physvol>
            </loop>
        </volume>
</structure>
```

Modules

- any GDML file can be used within another GDML file
  - physvol tag can use a logical volume from another file
- one can place the selected volume from the ‘child’ geometry tree in any volume of the ‘mother’ geometry tree using the standard position and rotation

- allows to split complex geometry trees into modules as in real life (tracker.gdml, calorimeter.gdml, beampipe.gdml) and combined them in detector.gdml

```
// child.gdml
<volume name="childvol">
    <materialref ref="Alluminium"/>
    <solidref ref="ChildBox"/>
</volume>

// mother.gdml
<volume name="mother">
    <materialref ref="Iron"/>
    <solidref ref="MotherBox"/>
    <physvol>
      <file name="child.gdml” volname=”childvol”/>
      <positionref ref="center"/>
      <rotationref ref="identity"/>
    </physvol>
</volume>
```

GDML readers and writers

- GDML reader and writer for Geant4 is part of the Geant4 release
- GDML reader and writer for ROOT is part of the ROOT release
- GDML reader for VecGeom part of VecGeom release
- some support for CAD to GDML (using STEP files and tesselated solids)
- DD4Hep can use GDML as exchange format
  - used by experiments for migration to DD4Hep

- and PyG4ometry can read and write GDML !

GDML in Geant4

- reader and writer fully supporting the GDML schema with all the components
- based on XercesC XML parser
- writer can be invoked from C++ or by a command in macro file
- several examples in Geant4 in GEANT4/geant4-dev/examples/extended/persistency/gdml

GDML in Geant4

- support for all the Geant4 solids, replicas, divisions, parameterized volumes, optical surfaces, etc
- support for NIST materials
- import/export of geometrical regions associated to volumes for importing and storing production cuts and user-limits as global auxiliary_info entity
  - enabled/dissabled using a flag in the parser
- import/export of arbitrary user information (auxiliary information)
  - sensitive detectors, visualization colours, etc
    - done via parser.AddVolumeAuxiliary(...) and parser.AddAuxiliary(...)

GDML in ROOT

- ROOT geometry modeller (TGeo classes) provide built-in support for GDML persistency
  - Implemented based on ROOT ***TXMLEngine***
  - access through GeoManager
    - `TGeoManager::Import("geometry.gdml")`
    - `gGeoManager->Export("geometry.gdml")`

- Development driven by the need to import Geant4 geometry setups in ROOT
  - ROOT GDML parser used extensively in DD4HEP

- some parts of the GDML schema for read/write not supported
  - no *paramvol* - missing support for parameterized placements
  - No support for generic *userinfo auxiliary* tags - only *Region* info read so far (and connected to TGeoVolume)

GDML in ROOT

- Support for Geant4 units was added in both TGeoManager and its GDML import/export
  - TGeo was intended unit-less (unit defined by user), however:
    - Implicit connection to units via material properties
    - Default units for GDML import/export are now the Geant4 ones

- Most recent additions
  - Optical properties - *opticalsurface, skinsurface, bordersurface*
  - Tessellated solids - *tessellated*
    - Tessellation definitions in separate files not supported yet

- No developments for additional support for GDML features planned
  - The current model is to add support on demand

GDML parsing in VecGeom

- Functionality added recently
  - Using ***xerces-c*** for the implementation
  - Separate library (*libvgdml*)
    - User interface implemented as: vgdml::Frontend::Load(...)
    - VecGeom data structures created by vgdml::Middleware class
    - xerces-c interface separated in vgdml::Backend class
  - VecGeom can also import GDML geometry via ROOT + transient conversion

- Just the “basic” GDML functionality available
  - Only the GDML reading part, no use case so far for writing GDML from VecGeom
    - Constants, positions, rotations, solids, structure
  - Several consistency fixes done recently

Unhandled GDML tags in VecGeom

- VecGeom is agnostic to physics
  - Strategy: Expose unhandled data in a raw form to the user
  - Special interface vgdml::Frontend::Parser::Load() returning a pointer to *vgdml::Middleware* having unhandled info attached
- Material and auxiliary info
  - Elements, isotopes, materials read into a ***MaterialInfo*** structure (maps of key/value strings)
  - *userinfo auxiliary* tags read into vgdml::Auxiliary class
- Support for more GDML tags will be needed
  - To read more complex geometry files
  - To allow full VecGeom persistency decoupled from ROOT

Summary – what GDML can do

- GDML is an application independent geometry description language
  - basically any detector geometry can be described using it
    - GDML provides means (auxiliary information) of storying any application-specific data
- GDML is extensive used for the physics testing and validation in simulation
- past and current simulation R&D projects (GeantV, AdePT, Celeritas) rely on GDML for the geometry import
- DD4Hep can use GDML as interchange format
- GDML manual at: https://gdml.web.cern.ch/GDML/doc/GDMLmanual.pdf