A scalable new mechanism to store and serve the ATLAS detector description through a REST web API

**ATLAS GeoModel**

The ATLAS experiment\textsuperscript{1} uses GeoModel\textsuperscript{2} to describe the **geometry tree**: a set of nodes connected through relationships of different types. The GeoModel tree is **computed on-the-fly** when requested through the experiment framework, and stored in-memory only.

**Current limitations**

- An online Geometry DB stores primary numbers used to build objects. But no structure and **no relationships** are **saved** in the DB; and **no data** are accessible offline;
- **Very difficult to debug** the detector description: have to go through the code;
- No way to open, explore and use the Detector Description without the **whole experiment’s framework**;
- **Strong platform dependency**: SLC6 Linux the only platform supported by ATLAS;
- Not possible to interactively query the GeoModel and retrieve matching volumes only.

**FRESH, NEW IDEAS** for a way to easily store, restore, access and serve the experiment’s geometry

**1\textsuperscript{st} STEP - Decoupling** and GeoModel **persistification**, so that applications can use the Detector Description without the need of run the full ATLAS framework. 

*Presented at CHEP 2016*\textsuperscript{2}.

**2\textsuperscript{nd} STEP – NEW!** Easy, interactive access to the geometry through a REST API, and retrieval of geometry subsets

**TWO GOALS, TWO TECHNOLOGIES** to implement a REST service to serve the geometry, different approaches

**Graph DB (Neo4j\textsuperscript{3})**

*Works on GeoModel nodes*

To store, query and visualize the inner structure of the ATLAS geometry tree, and all the relationships between its nodes.

Neo4j is a **No-SQL DB** based on nodes and relationships, like GeoModel. Cypher is its **query language**, focused on relationships, labels, properties. Users are able to fast query all the instances of GeoModel objects used in the actual ATLAS geometry, and all connections and paths between them. The order of all nodes is stored, because it is used by GeoModel while traversing the tree to construct the detector. Also, users can interactively visualize the nodes and their connections in a graph, which is extremely useful to debug the Detector Description.

**Example queries** (on GeoModel nodes – easy to construct queries to query the tree structure)

Retrieve a specific GeoModel node (GeoLogVol), based on a node property:

\[
\text{match } n:\text{GeoLogVol} \text{ where } n.name = "TileEndcapNeg" \text{ return } n; \]

As above, but retrieve the GeoPhysLogVol node which uses it:

\[
\text{match } n:\text{GeoPhysLogVol} \text{ where } n.name = "TileEndcapNeg" \text{ return } n; \]

Get all “Box” Shape nodes used together with “Air” as Material:

\[
\text{match } n:\text{Shape} \text{ where } n.Name = "Box" \text{ and } \text{exists}(m:\text{Material} \text{ where } m.Name = "Air") \text{ return } n; \]

Get all the children nodes of the RootVolume, and sort them by the ‘position’ property of the CHILD relationship:

\[
\text{match } (\text{r:RootVolume}\rightarrow\text{c:}\text{GeoModel}) \text{ with } \text{r.order} \text{ by } \text{r.position} \text{ return } \text{r.position}; \]

Both Neo4j and ES/Kibana have **REST access** as well as **web interfaces** for interactive queries

**Comparison**

<table>
<thead>
<tr>
<th>Current implementation: Geometry DB</th>
<th>Storage, Query and Visualization of all GeoModel nodes and relationships</th>
<th>Partial retrieval of geometry data (subtrees)</th>
<th>Fast retrieval of final physical volumes with global positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline access to geometry data</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>NEW</strong> Persistent SQLite DB</td>
<td>V (with SQL gimmick)</td>
<td>(with SQL gimmick)</td>
<td>X</td>
</tr>
<tr>
<td><strong>NEWEST</strong> Neo4j Graph DB</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td><strong>NEWEST</strong> ElasticSearch/Kibana</td>
<td>V</td>
<td>X</td>
<td>V</td>
</tr>
</tbody>
</table>

**Search Engine (ElasticSearch / Kibana\textsuperscript{3})**

*Works on end physical volumes*

For a fast retrieval of the **final objects**: all transformations and attributes are **computed** and accumulated at indexing time.

ElasticSearch uses Lucene as query language. Relationships between the nodes and nodes’ order are not stored, all object are “flattened”: all the attributes are collected, and all the space transformations are **computed while inserting the data**. Users then can quickly retrieve the **final volumes**: physical volumes together with their **absolute positions** and all attributes related to it. The goal is to let applications quickly retrieve geometry objects without having to traverse the geometry tree.

**Example queries** (on final physical volumes - easy to construct queries to fast retrieve volumes)

Retrieve a specific volume:

\[
\text{match } n:\text{TileEndcapNeg} \text{ return } n; \]

Retrieve all Pixel volumes in the EndCap side C:

\[
\text{tags:Pixel AND tags:EndcapC} \text{ return } \text{all}; \]

Retrieve all box-shaped volumes whose material is “Air” shape box AND material air:

\[
\text{match } n: \text{Box}\rightarrow \text{r:Material} \text{ where } \text{r.Name} = "Air" \text{ return } n; \]

Riccardo Maria BIANCHI (Pittsburgh), Ilija VUKOTIC (Chicago)

Contact: rbianchi@cern.ch

on behalf of the ATLAS Collaboration