

ARIADNE: a Tracking System for Relationships in LHCb Metadata

I Shapoval^{1,2,3,4}, M Clemencic¹, M Cattaneo¹

¹ European Organization for Nuclear Research (CERN), CH-1211 Geneva 23, Switzerland

² Kharkiv Institute of Physics and Technology (KIPT), UA-61108 Akademichna 1, Kharkiv, Ukraine

³ Sezione INFN di Ferrara (INFN-FE), IT-44122 Via Saragat 1, Ferrara, Italy

⁴ Università di Ferrara (UNIFE), IT-44121 Via Savonarola 9, Ferrara, Italy

E-mail: illya.shapoval@cern.ch

Abstract. The data processing model of the LHCb experiment implies handling of an evolving set of heterogeneous metadata entities and relationships between them. The entities range from software and databases states to architecture specifiers and software/data deployment locations. For instance, there is an important relationship between the LHCb Conditions Database (CondDB), which provides versioned, time dependent geometry and conditions data, and the LHCb software, which is the data processing applications (used for simulation, high level triggering, reconstruction and analysis of physics data). The evolution of CondDB and of the LHCb applications is a weakly-homomorphic process. It means that relationships between a CondDB state and LHCb application state may not be preserved across different database and application generations. These issues may lead to various kinds of problems in the LHCb production, varying from unexpected application crashes to incorrect data processing results. In this paper we present Ariadne - a generic metadata relationships tracking system based on the novel NoSQL Neo4j graph database. Its aim is to track and analyze many thousands of evolving relationships for cases such as the one described above, and several others, which would otherwise remain unmanaged and potentially harmful. The highlights of the paper include the system's implementation and management details, infrastructure needed for running it, security issues, first experience of usage in the LHCb production and potential of the system to be applied to a wider set of LHCb tasks.

1. Introduction

The LHCb [1] data processing [2] implies handling of an evolving heterogeneous metadata and relationships in it. The data processing workflow is entirely dependent on the question of what is the valid combination of various metadata entities for a particular step of the workflow. The fact that the relationships between the entities, which define the combination, are not tracked systematically and are managed mostly in a manual and very limited way may lead to a diverse set of harmful consequences in the LHCb production, some of which may be visible while others – hidden. One can experience such after-effects, for instance, when LHCb software evolution passes non-homomorphic development phases, or when CondDB [3] undergoes structural changes. These phenomena have occurred to be very annoying in the LHCb collaboration running into a thousand of members and need to be reduced to minimum. In the previous papers we discussed a possible design [4] and described a prototype of the system [5] which was meant to track and



analyze the relationships in LHCb metadata. This work crowns further evolution of requirements to the system and corresponding evolution of the prototype that turned it into the system recently exposed in the LHCb production.

We start out with an introduction to the LHCb metadata space where the relationships need to be tracked (section 2), then, following a description of one of the relationships problems met by the LHCb experiment (section 3), we will present the design of Ariadne (section 4).

2. Metadata specification

We define an LHCb metadata entity in the following way:

Definition 2.1. *The LHCb metadata entity is a unique descriptive entity which represents a basic unit of any information resource used in the LHCb experiment.*

The LHCb metadata space encompass quite heterogeneous set of metadata entities that play governing roles in the configuration of the LHCb data processing. For instance, such metadata entity can represent a data processing application, a CondDB state, a type of data processing, a data or software deployment location, a computing platform the software is built for, etc. We describe the most important of them in the following subsections.

2.1. Software metadata

In the LHCb data processing, applications are the pillars of the LHCb software that perform specialized processing steps. Some flavors of the LHCb applications are as follows:

- Moore – used for high level triggering;
- Brunel – used for event reconstruction;
- DaVinci – used for event selection and data analysis;
- Gauss – used for event generation and detector simulation;
- Boole – used for simulation of the detector response and of the readout electronics.

In addition to the application flavor another important aspect is coming from the fact that the LHCb data processing model requires reproducibility of analyses which implies the software to be versioned. We associate an application flavor and version with a software metadata entity – a basic unit of the two-dimensional LHCb software metadata space with dimensions being the application flavor and version.

2.2. Conditions database metadata

A general CondDB state consists of three substates each describing a state of one of four CondDB partitions [4]. We allot the role of the CondDB metadata entity to the CondDB substate. Further on, each substate is tagged at specific moments of partition's evolution. So an unambiguous reference to the general CondDB state is a triplet of CondDB metadata entities each defined, in turn, by the tag. Not every triplet of the CondDB metadata entities forms a self-consistent general CondDB state.

Another auxiliary CondDB metadata, that we have to consider, is *detector type*. In the LHCb data processing it is mainly used for the purposes of grouping the CondDB substates by the LHCb detector layout which may evolve from one data taking period to another. Thus each CondDB substate is associated with a detector type and each CondDB state has to be formed out of the substates associated to common detector type.

2.3. Data processing metadata

The LHCb physics data, be it real data, collected by the detector, or Monte Carlo data, produced in simulations of the detector, pass through a workflow consisting of several steps of processing by the applications mentioned in subsection 2.1. Configuration of the steps may differ from one workflow to another and is identified by a processing type. Naturally, the type falls into one of the two categories:

- Reconstruction types – used in real data processing;
- Simulation types – used in Monte Carlo data processing.

We associate the metadata entity of data processing with the processing type only, thus making the data processing metadata space one-dimensional.

2.4. Other metadata

There are many other examples of metadata that the LHCb data processing involves. Some are application specific, others are common to all applications:

- specifiers of computing platforms that applications are built for;
- locations of software and data deployment;
- trigger configuration keys used during data collection, etc.

In this paper we will not consider these varieties of metadata entities.

3. Relationships in metadata: problem statement

There are plenty of relationship types in LHCb metadata that are heavily used in the LHCb data processing but there is one that stands out in the midst of others - the *compatibility*. Historically, this is the very type that triggered the development of the relationships tracking system. It will be used to demonstrate, first, the nature of relationship problems that the LHCb experiment met during its operation, and second, how this kind of problems can be solved.

3.1. Relationships of compatibility

Before going any further we have to specify what stands behind the notion of compatibility in the LHCb environment.

Definition 3.1. *A pair of metadata entities, used to configure an LHCb job, is called compatible if the job, that has all the rest of configuration being valid:*

- *behaves exactly in the way a developer anticipated it;*
- *never fails during execution.*

Implication 3.1. *A set of metadata entities is called compatible if and only if each and every pair of the entities out of the set is compatible.*

The LHCb data processing model dictates the following compatibility requirements:

- each data processing step is accomplished using an application compatible with it;
- every CondDB state has to be composed of a set of compatible substates;
- every application has to use a CondDB state compatible with it;
- every CondDB state has to be compatible with detector and data processing types of interest.

Thus, the requirements infer the existence of the following categories of compatibility relationships between the entities, described in section 2:

- Compatibility between CondDB metadata entities;
- Compatibility between triplets of the CondDB metadata entities and the entities of
 - software metadata;
 - metadata of data processing;
 - detector metadata.
- Compatibility between software metadata and
 - metadata of data processing;
 - detector metadata.

Consequently, the compatibility problem can be formulated as the problem of resolving a fully compatible set of metadata entities, which satisfy the constraints on their attributes established by requirements of a concrete data processing step.

4. Ariadne

The problem, formulated in the previous section, yields requirements for a system needed to solve the problem. The system would have to:

- A. Provide an operational space with generic way of
 - expressing constraints on entities and relationships;
 - tracking entities and relationships;
 - extracting relationship solutions;
- B. Feature the ease of management of relationships information;
- C. Support extensibility of application area.

In the following subsection we will see how all mentioned requirements can be satisfied within a generic approach.

4.1. Modeling structured metadata

Handling all types of entities and their relationships in a generic way requires a data model which is abstract enough to encompass the use cases outlined in section 3. We will use the case of compatibility relationships to demonstrate how our considerations could be put on a higher abstraction level.

All metadata that is structured by relationships can be naturally expressed in the frame of the graph theory in terms of an attributed graph with typed relationships. One can define the graph elements in the following way:

- A node: a metadata entity
 - with dimensions of its metadata space mapped to node attributes.
- An edge: a relationship between metadata entities
 - with its type mapped to the edge's attribute.

Thus, as an example, all compatibility categories, described in section 3.1, and metadata entities, described in section 2, can be synthesized into a graph shown in figure 1. It's easy to see, this form of tracking unit satisfies the above mentioned requirement *C* since it is able to reflect any kind of flavors of entities and relationships, as well as any level of complexity of the relationships topology. More over, the graph representation is also very attractive in its convenience to model the sets of relationships patterns using three degrees of freedom:

- constraints on metadata entities specification;
- constraints on topology of relationships;
- constraints on relationship specification;

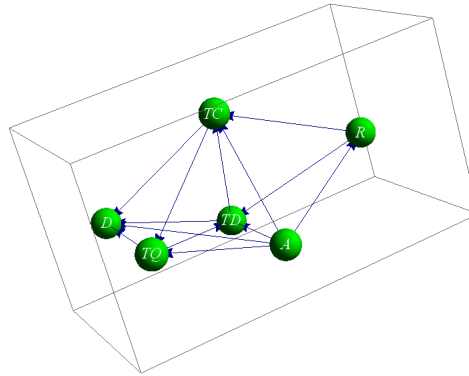


Figure 1. A graph that models the topology of the compatibility problem. The A-node represents an application, the T(D,C,Q)-nodes - three CondDB substates, while the D- and R-nodes model the detector and data processing types. The full list of nodes' attributes of this graph is shown in table 1.

Table 1. Node's attributes of the graph shown in figure 1. The main attributes define a metadata entity uniquely, while the extra ones play a supplementary role allowing more fine-grained resolution of entities.

Type of graph node	Main attributes	Extra attributes
Application	name, version	release date
CondDB substate	tag name, partition	release date, payload hash sum
Detector type	type name	-
Data processing type	type name	-

which makes the first item of requirement A met. In this way, going back to the case of compatibility problem, if all the attributes' values of the graph (figure 1) are resolved by the system, satisfying a set of constraints, then the graph becomes a concrete *compatibility solution* holding a fully compatible set of metadata entities.

In the frame of such approach the full metadata space can be represented as a big graph with all metadata being interconnected according to application domain and its data model. We call this graph the *LHCb metadata knowledge graph*.

Having part of the requirements met already by the proposed data model, we will show how the rest of them are satisfied by the developed system itself.

4.2. System architecture

To fulfill the goals, stated in the beginning of the section, the Ariadne¹ system was developed being heavily grounded on the graph-based data model. Ariadne is able to accumulate heterogeneous metadata primitives in the common knowledge graph. More over, Ariadne provides means for analyzing the knowledge graph in order to infer solutions to relationships problems of any level of intrication. This procedure is narrowed down to discovery of requested *subgraph patterns*, satisfying requested constraints, in the knowledge graph. The architecture of

¹ The system was called after ancient Greek character of Ariadne, who, according to the legend, was in charge of the Cnossian Labyrinth and assisted Theseus, with a clue of thread, to find a way back from the labyrinth after killing the Minotaur.

Ariadne is shown in figure 2 by means of a data flow diagram. The data flow diagram of the

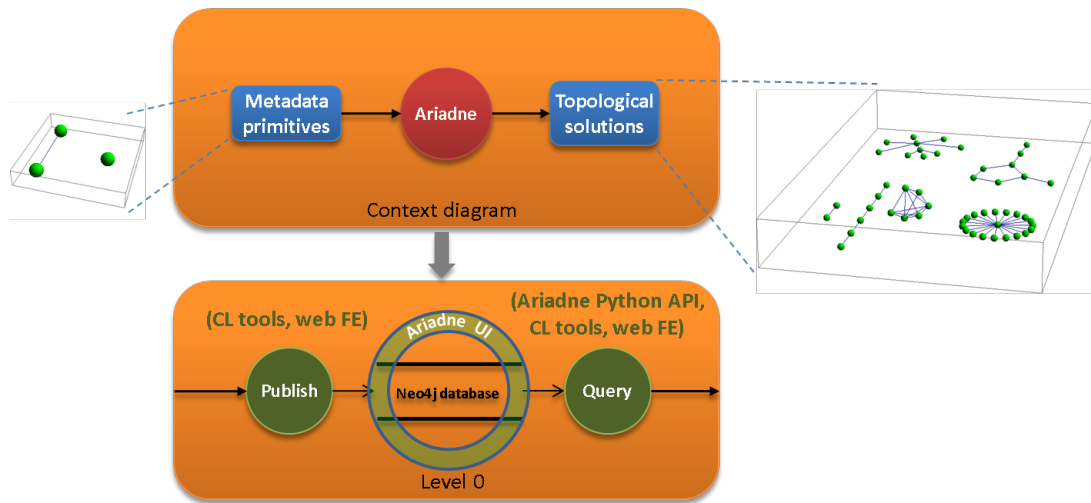


Figure 2. Levelled data flow diagram of Ariadne (shown in the Yourdon-DeMarco DFD notation [6]). The *Level 0* layer depicts the Ariadne intrinsics. See subsection 4.2.1 for description of the database, used in Ariadne.

system demonstrates that the system expects to receive the input being the metadata primitives, which can be:

- new metadata entities (or editions to existent ones);
- new relationships in metadata (or editions to existent ones).

The output is a solution obtained using the pattern and constraints, described in a query. The solution is typically a graph pattern with its nodes' attributes being entirely resolved according to the constraints provided in a query.

At this point it may become apparent to the reader that the chosen data model, together with the system that naturally boosts its benefits, make the graph-based tracking unit act as a common language through all the stages of the problem: its determination, processing and solution declaration.

4.2.1. Ariadne database. The Ariadne database is the core of the system, which provides the persistency layer for all known relationships in LHCb metadata. We considered the most relevant database management systems and concluded [5] that the Neo4j [7] graph database satisfies most of the requirements implied by our approach. In particular, it features the graph-oriented persistency layer and an extremely powerful graph analyses framework embedded.

A subspace of the LHCb metadata knowledge graph, stored in the database, is visualized in figure 3. Currently the entire knowledge graph contains around 600 nodes connected by around 50000 relationships.

4.3. Ariadne: management

The Ariadne management narrows down to management of its database content. Two sets of management tools are available:

- Ariadne Python command line administration scripts (developed specifically for the LHCb environment), which are:

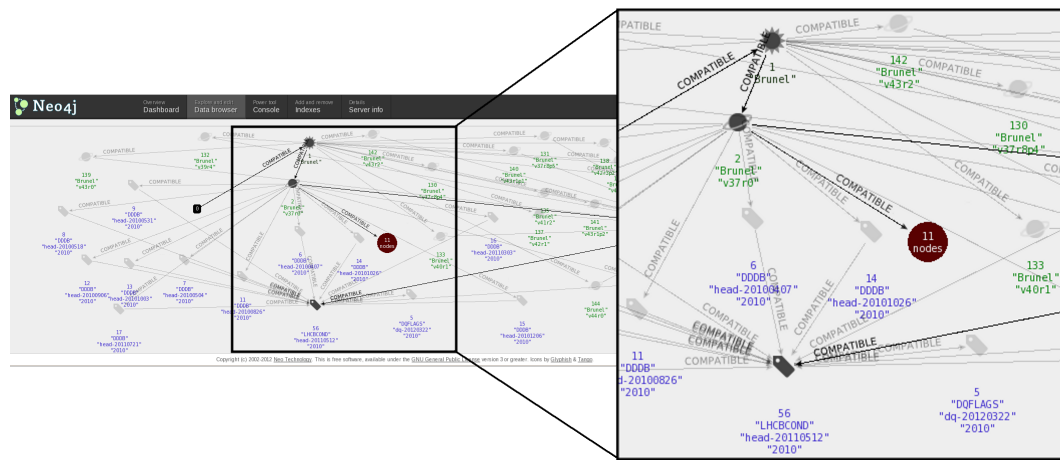


Figure 3. The web administration interface, shipped with Neo4j, provides an interactive environment to browse, add or modify nodes, relationships and their attributes.

- built on developed Ariadne API, based, in turn, on *Py2neo* [8] library which interacts in a RESTful style with the Ariadne XMLRPC Server;
- aimed for management of LHCb metadata primitives in the Ariadne's database;
- aimed for high-level management of the LHCb metadata knowledge graph.
- Neo4j Server Web Administration interface (a native Neo4j administration tool, see figure 3):
 - monitor the Neo4j Server;
 - manipulate and explore the graph data;
 - interact via the *Cypher* [9] and *Gremlin* [10] consoles.

5. Conclusion

In this paper we described the essence of the problem of tracking the metadata relationships faced by the LHCb data processing. We described a generic approach to solve the problem. This approach is grounded on the graph-based data model, which, boosted by the graph-friendly tracking system, provides a common operational space to elegantly express, track and extract any relationship patterns in the heterogeneous metadata space of the LHCb experiment.

References

- [1] LHCb Collaboration 2008 *The LHCb detector at the LHC* JINST **3** S08005 (Geneva, CERN)
- [2] LHCb Collaboration 2005 *LHCb computing: technical design report* CERN-LHCC-2005-019 LHCb-TDR-11 (Geneva, CERN)
- [3] Clemencic M, Gilardi N and Palacios J 2006 *LHCb conditions database* CERN-LHCb-2006-017 CHEP'06 (Mumbai) pp 347-350
- [4] Clemencic M, Shapoval I, Cattaneo M *et al* 2012 *LHCb conditions database operation assistance systems* J. Phys.: Conf. Series **396** 052022
- [5] Cattaneo M, Clemencic M and Shapoval I 2012 *LHCb software and conditions database cross-compatibility tracking system: a graph-theory approach* IEEE NSS/MIC '12 (Anaheim, California) CERN-LHCb-PROC-2012-058
- [6] Yourdon E and Constantine L L 1979 *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design* (Yourdon Press)
- [7] *Neo4j NOSQL graph database project* Available: <http://neo4j.org>
- [8] *Py2neo Python library* Available: <http://py2neo.org>
- [9] *Cypher - a declarative graph query language* Available: <http://docs.neo4j.org/chunked/stable/cypher-query-lang.html>
- [10] *Gremlin - a Turing-complete graph-based programming language* Available: <http://gremlin.tinkerpop.com>