PROCEEDINGS OF SCIENCE

EDM4hep – a common event data model for HEP experiments

Placido Declara Fernandez,[b] **Frank Gaede,**[a,*] **Gerardo Ganis,**[b] **Benedikt Hegner,**[b] **Clement Helsens,**[c] **Thomas Madlener,**[a] **Andre Sailer,**[b] **Graeme A. Stewart**[b] **and Valentin Völkl**[b]

[a] *Deutsches Elektronen-Synchrotron DESY, Germany*
[b] *CERN, Switzerland*
[c] *KIT, Germany*

E-mail: frank.gaede@desy.de

A shared, common event data model, EDM4hep, is an integral part of the Key4hep project. EDM4hep aims to be usable by all future collider projects, despite their different collision environments and the different detector technologies that are under discussion. This constitutes a major challenge that EDM4hep addresses by using podio, a C++ toolkit for the creation and handling of event data models, developed in the context of the AIDA R&D program. This approach allows for quick prototyping of new data types and provides a streamlined framework for updates. After presenting an overview of the basic features of EDM4hep and podio, we will discuss the current experience with an initial version of EDM4hep in different physics studies. Additionally, we will present the planned developments that are necessary for a first stable version of EDM4hep, addressing in particular backward compatibility aspects and schema evolution. We will conclude with an outlook on the future developments directions beyond this first stable version.

41st International Conference on High Energy physics - ICHEP2022
6-13 July, 2022
Bologna, Italy

*Speaker

PoS(ICHEP2022)1237

1. Introduction

One of the most important pieces of software in a HEP processing framework is the Event Data Model (EDM). It defines the language that is used throughout the full processing chain of HEP experiments, including event generation and simulation, as well as reconstruction and analysis. Given this central role, having a well defined and efficiently implemented EDM is of key importance. EDM4hep is the common, and shared EDM for the Key4hep project [1], which is developing a common software stack for all future colliders. Covering all of the possible physics use cases in the various involved communities constitutes a major challenge for EDM4hep. Various detector technologies that are currently under consideration as well as different collision environments have to be accomodated. The podio EDM toolkit addresses the two afforementioned challenges by offering the possibilty of generating performant C++ code from a high level definition of an EDM. This allows for quick prototyping of new data types without having to implement any C++ code manually. Additionally, this makes it possible to roll out implementation improvements easily by simply updating the code generation.

In these proceedings we start with introducing the basics of the podio toolkit, highlighting some of the recent developments and their importance for EDM4hep. In Sec. 3 we focus on EDM4hep from a physics point of view and discuss some of the design choices before we report on experiences from using EDM4hep in physics studies in the different communities in Sec. 4. We end with a brief summary and an outlook into the future of EDM4hep and podio.

2. The podio EDM toolkit

An EDM is only as useful as the information content that can be expressed within it. This comprises not only the data that can be stored in the members of the data types that are defined within the EDM, but also, and equally important, the relations between them. These relations allow to express hierarchical structure that is inherent in the data of HEP experiments. Implementing these capabilities efficiently is a non-trivial task, but also lends itself to a certain degree of automation. Leveraging these possibilities is one of the key ideas of podio: starting from a high level description of the desired data types and their relations among each other, generate an efficient implementation of the corresponding EDM in C++. This approach completely frees the user from implementation details and instead allows them to focus entirely on the best definition of the EDM.

For an introduction to the general design choices and capabilities of the podio toolkit we refer to previous publications [2–4]. Here, we will instead entirely focus on the most recent developments that we consider to be the most important ones; the *Frame* concept, Schema evolution of generated EDMs and the possibility to extend EDMs. The former two of these features are still under active development. Hence, the following descriptions are still subject to potential changes in the future but we consider the basic concepts and design as stable.

2.1 The podio Frame concept

The original *EventStore* that has been shipped with podio since the begining was intended mainly as a simple example implementation of a transient event store that grants access to a *coherent* set of collections of data types. It has by now far outlived its example status and several

short comings have become apparent in (unintended) production use. To address several of these issues the *Frame* has recently been developed.

The Frame is a "generalized event data container". It aggreggates all relevant data of a given *interval of validity* or category, e.g. a HEP event or run, but also something like a readout time frame for experiments that have no clearly defined notion of an "event". One of the major design goals was thread-safety. The most important design choice towards this goal was to clearly express the ownership of the data contained in a Frame, by e.g. making it mandatory to explicitly *move* collections into a Frame, thus relinquishing ownership and in consequence mutable access. Once a collection has been put into a Frame, only immutable read-only access is possible. This ownership model makes it possible to safely access collections stored in a Frame concurrently from multiple threads, while leaving users in full control of threading while preparing collections.

In line with the overall design choices of podio, the Frame offers value semantics. This is achieved using the *type erasure* implementation technique. As a side benefit of this technique and the overall design choices the Frame also very nicely decouples I/O concerns from how users access data, by being constructible from almost arbitrary *Frame data*. These can be defined by every I/O backend separately and only have to offer a very minimal interface to be usable by a Frame.

The first version of the Frame and I/O functionality the ROOT [5] and *Simple Input/Output* SIO [6] backends has recently been released. We are currently working on rolling this out to the various packages in the Key4hep software stack that use podio. Additionally, we are also working on additional features, e.g. adding policies that alter the Frame behavior but not its interface. For example, having the possibility to control when and how a Frame reconstructs the collections from the data read from file.

2.2 Schema evolution of generated EDMs

Schema evolution, i.e. having the possibility to adapt an EDM to the ever evolving use cases without losing access to data that has been written with a previous version, is a crucial feature. Implementing a schema evolution mechanism in all generality is a highly non-trivial problem to solve. Given the sheer size of the problem space to cover here, podio does not even attempt to do so. Rather, we focus on providing the very basic features now and design the schema evolution mechanism such that it allows to tackle more complex changes to an EDM definition once the need for it actually arises.

The first step that needs to be tackled in schema evolution is to actually find out which changes occur between two version of a given EDM definition. To tackle this, we have implemented an automated tool that reads the high level definition files of a given EDM in two different versions and spits out the differences between the two. From these differences we then intend to generate the entire migration code automatically or inform the user if that is not possible. A key advantage of this approach is that we can tell users very early whether their intended change is currently supported; before they even get a chance to write data that would result in breaking backwards compatibility.

In our approach, users will only ever have access to the latest version of the EDM in memory. All the necessary schema evolution happens before the collections are constructed internally. This allows us to take advantage of existing schema evolution mechanisms for I/O backends that offer them, e.g. ROOT. The necessary hooks for schema evolution to happen for I/O backends that do

not have mechanisms themselves have been added to the Frame, where all of the unpacking of the data read from file is happening.

To allow for detecting the EDM version when reading a file, additional meta data has to be present, most importantly the schema version of the EDM at the time of writing. This schema version is taken directly from the EDM definition and has to be provided by the user. We also plan to actually store the complete schema definition into the metadata of files that are written with podio for later reproducibility. Work is currently ongoing to finalize a first version of schema evolution.

2.3 Extending existing EDMs

Although an EDM should in general be designed to be stable and complete, new developments in detector technology may require to develop new reconstruction techniques which might in turn require new data types to represent the measurements. In order to allow for an easy way of prototyping in such an environment we have recently developed a mechanism that allows to extend an existing EDM with additional data types. The new data types can simply be defined in a separate definition file and passed to the code generator together with the existing EDM definition. Since this extension mechanism is rather powerful and has the potential to proliferate a multitude of several almost, but not entirely compatible EDMs, we advocate to use this feature with restraint and great care.

3. EDM4hep

EDM4hep is largely based on LCIO [7], the EDM successfully shared and used by the linear collider community for the last almost two decades. As can be seen in Fig. 1 it features the typical data types that are necessary to express the use cases that are encountered in HEP experiment workflows. These include data types to represent data generated during simulation, as well as the ones encountered during reconstruction starting from raw data from the detector, up to the data types representing higher levels of reconstruction. The general structure of EDM4hep and the relation between LCIO and EDM4hep is discussed in slightly more detail in [4].

As already mentioned in Sec. 2 the extension mechanism has potential for misuse. Hence, in the EDM4hep context it is only allowed for prototyping in new detector technology contexts, and the goal of such ventures should always be the eventual integration into EDM4hep.

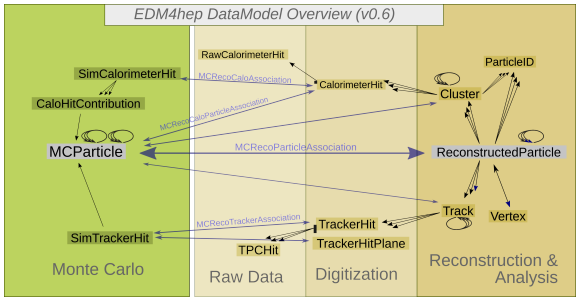

Figure 1: EDM4hep schema organized by data types necessary for simulation (green background) and the ones required in reconstruction going. The black arrows indicate relations defined within the data types, while the blue arrows indicate *associations* that are defined externally.

4. Applications of EDM4hep

All communities that participate in and contribute to the Key4hep project are by now using EDM4hep, or are at the very least in the process of migrating to it. The migration status and process exhibits quite some differences between the different communities, mostly reflecting their corresponding starting points.

The linear collider community with many existing data sets in LCIO [7] format opts for a gradual migration towards Key4hep and EDM4hep with the main goal of preserving the existing tools and workflows. On the framework and algorithmic side this is handled by the *MarlinWrapper* [8, 9] that allows to use existing Marlin [10] processors in the Gaudi [11] based framework of Key4hep. This wrapper also comes with the necessary tools to convert between LCIO and EDM4hep on the fly, thus enabling running a complete reconstruction chain and converting the results to EDM4hep at the end. This workflow has been successfully validated on a technical level and physics validation is ongoing.

The FCC community has fully migrated from *fcc-edm* [12] to EDM4hep. For data analysis *FCCAnalyses* [13], based on *RDataFrame* [14] has been developed. It offers a python interface to read and analyse EDM4hep data files in a multi-threaded fashion. As it directly reads the data stored in the root files it does not use the podio generated interface of EDM4hep. All physics analyses of the FCC community are by now using the EDM4hep format, and it is routinely used in large level production campaigns.

Recently, the EIC community has also adapted EDM4hep as base for their EDM, after originally starting with their own EDM also based on podio. Given that not all of the data types that were present in their original EDM are present in EDM4hep, the EIC community is using the newly developed extension mechanism of podio to use EDM4hep data types in conjunction with the ones that are not yet present. The plan here is to over time migrate the majority of these data types to EDM4hep.

5. Summary & Outlook

EDM4hep is a common Event Data Model for HEP that is mainly developed in the context of the Key4hep software ecosystem. At this point in time EDM4hep has been adopted by several communities, including ILC, CLIC, FCC, CEPC but also EIC. For some of these communities EDM4hep is already used routinely in production and for physicis analyses. It is based on the podio EDM toolkit, which has recently undergone some quite significant new developments. The most important of these developments and their implications have been discussed here: The Frame concept, schema evolution of generated EDMs as well as the possibility to extend existing EDMs for prototyping purposes. Overall we find the community spanning effort of defining EDM4hep to be a great success, resulting in a more generally usable EDM by leveraging the experiences from the multiple different use cases that arise in these communities.

The next steps for podio, and in extension also for EDM4hep, is to finalize the outstanding work on schema evolution. Once that work is finished we plan to release a first stable version of both. From this point on podio and EDM4hep will have guaranteed backwards compatibility.

Acknowledgments

This work benefited from support by the CERN Strategic R&D Programme on Technologies for Future Experiments (https://cds.cern.ch/record/2649646/, CERN-OPEN-2018-006) and has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No 101004761.

References

[1] P. Fernandez Declara et al., *Key4hep: Status and Plans*, *EPJ Web Conf.* **251** (2021) 03025.

[2] F. Gaede, B. Hegner and P. Mato, *PODIO: An Event-Data-Model Toolkit for High Energy Physics Experiments*, *J. Phys. Conf. Ser.* **898** (2017) 072039.

[3] F. Gaede, B. Hegner and G.A. Stewart, *PODIO: recent developments in the Plain Old Data EDM toolkit*, *EPJ Web Conf.* **245** (2020) 05024.

[4] F. Gaede, G. Ganis, B. Hegner, C. Helsens, T. Madlener, A. Sailer et al., *EDM4hep and podio - The event data model of the Key4hep project and its implementation*, *EPJ Web Conf.* **251** (2021) 03026.

[5] R. Brun and F. Rademakers, *ROOT: An object oriented data analysis framework*, *Nucl. Instrum. Meth. A* **389** (1997) 81.

[6] "Sio github repository." https://github.com/iLCSoft/SIO, 2021.

[7] F. Gaede, T. Behnke, N. Graf and T. Johnson, *LCIO: A Persistency framework for linear collider simulation studies*, *eConf* **C0303241** (2003) TUKT001 [physics/0306114].

[8] P. Fernandez Declara et al., *The Key4hep turnkey software stack for future colliders*, *PoS* **EPS-HEP2021** (2022) 844.

[9] P. Fernandez Declara, A. Sailer, M. Petric and V. Volkl, *key4hep/k4marlinwrapper: v00-03-01*, Apr., 2021. 10.5281/zenodo.4719245.

[10] F. Gaede, *Marlin and LCCD: Software tools for the ILC*, *Nucl. Instrum. Meth. A* **559** (2006) 177.

[11] G. Barrand et al., *GAUDI - A software architecture and framework for building HEP data processing applications*, *Comput. Phys. Commun.* **140** (2001) 45.

[12] "Fcc-edm github repository." https://github.com/HEP-FCC/fcc-edm.

[13] "Fccanalyses github repository." https://github.com/HEP-FCC/FCCAnalyses.

[14] D. Piparo, P. Canal, E. Guiraud, X. Valls Pla, G. Ganis, G. Amadio et al., *RDataFrame: Easy Parallel ROOT Analysis at 100 Threads*, *EPJ Web Conf.* **214** (2019) 06029.