Grant Agreement No: 101004761

AIDAinnova

Advancement and Innovation for Detectors at Accelerators

Horizon 2020 Research Infrastructures project AIDAINNOVA

DELIVERABLE REPORT

TURNKEY SOFTWARE STACK (KEY4HEP)

DELIVERABLE: D12.1

Document identifier:	AIDAinnova-D12.1
Due date of deliverable:	End of Month 46 (Jan 25)
Report release date:	30/01/2025
Work package:	WP12: Software for Future Detectors
Lead beneficiary:	DESY
Document status:	Final

Abstract:

Studying the physics potential of future colliders requires a well maintained, easy to use software stack. The Key4hep projects aims at providing such a stack by combining the efforts of all the involved communities and by fostering collaboration amongst them. In the context of this AIDAinnova deliverable several core components of this software ecosystem have been brought to production readiness, or close to it. These include the common event data model, EDM4hep; the PODIO toolkit that is used to implement it; as well as some core framework components that provide the necessary interfaces for algorithm developers. Additionally, workflows for building and deploying the whole software stack have been developed and are now used routinely. In this report an overview of the core components is given, along with the developments that were accomplished to make it possible to do comprehensive physics studies with the Key4hep software ecosystem.

The Advancement and Innovation for Detectors at Accelerators (AIDAinnova) project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 101004761. AIDAinnova began in April 2021 and will run for 4 years.

Delivery Slip

	Name	Partner	Date
Authored by	T. Madlener [Task coordinator]	DESY	15/01/2025
Edited by	S. El yacoubi	CERN	20/01/2025
Reviewed by	G.Stewart [WP coordinator] F.Gaede [WP coordinator]	CERN DESY	23/01/2025
Approved by	Paolo Giacomelli [Scientific coordinator] Steering Committee	INFN	30/01/2025

TABLE OF CONTENTS

1. INTRODUCTION 4

2. CORE KEY4HEP COMPONENTS 6

2.1. EDM4HEP – THE COMMON DATAMODEL 7
2.1.1. PODIO – A datamodel toolkit for HEP 7
2.2. K4FWCORE – CORE FRAMEWORK COMPONENTS 8
2.3. DD4HEP – THE COMMON DETECTOR GEOMETRY TOOLKIT 8
2.4. K4MARLINWRAPPER – INTEGRATING EXISTING SOFTWARE 9

3. RESULTS 10

3.1. A FIRST STABLE RELEASE FOR PODIO 10
3.1.1. New multithreading concept for PODIO 10
3.1.2. Schema evolution facilities 11
3.1.3. Extending existing data models 12
3.1.4. Interface data types 12
3.1.5. Developments after v1.0 12
3.2. FINALIZING THE EDM4HEP SCHEMA 13
3.2.1. Usability in multithreaded contexts 13
3.2.2. Other developments towards v1.0 14
3.3. MULTITHREADING WITH K4FWCORE 15
3.3.1. Rework of the PODIO integration 15
3.3.2. Functional algorithms and related framework support 16
3.3.3. Other developments 16
3.4. DDDIGI – THE DIGITIZATION EXTENSION FOR DD4HEP 17
3.5. INTEGRATION OF LC RECONSTRUCTION 17
3.6. BUILDING AND DEPLOYING KEY4HEP 18
3.6.1. Usage of CI/CD 19

4. CONCLUSIONS AND POSSIBLE FUTURE WORK 21

5. REFERENCES 22

ANNEX: GLOSSARY 24

Executive summary

The Key4hep project aims at providing a comprehensive suite of software for studying the physics potential of detectors at future colliders. As such, it needs to focus on ease of use, possibility of rapid development of novel reconstruction techniques, as well as stability for large scale productions. In the context of AIDAinnova major steps have been taken towards production readiness and Key4hep is now used by all communities that are involved in future collider studies, including FCC, ILC, CEPC, EIC and MuonCollider.

The detector description toolkit used by Key4hep, DD4hep, is already widely adopted for geometry description and simulation. In the context of AIDAinnova a new digitization extension has been designed and implemented.

EDM4hep, the common event data model for the Key4hep project, has been a key focus and is now close to a first stable release. Its implementation is generated by the PODIO toolkit, which has been finalized in the context of this deliverable and for which a first stable version has been released. This also forms the basis for developments in other core components that are used to provide access to data in EDM4hep format inside the Gaudi based Key4hep framework.

The integration of existing reconstruction software, developed by the linear collider communities, was achieved as an intermediate milestone, MS47, and some additional work building on top of the existing reconstruction workflows has been done to bring particle flow reconstruction to the liquid argon calorimeter of the ALLEGRO concept.

The build and deployment workflows used by Key4hep have been further refined and are now used to provide regular stable releases as well as nightly builds of the complete software stack on two different OSs. Major effort has also been put into creating documentation to allow faster onboarding of new users.

In addition to being a focus point for necessary technical work, Key4hep has also served as an excellent community building tool, which is also reflected by the fact that it was possible to attract new communities to contribute to the common and shared effort.

1. INTRODUCTION

Various different future collider projects and associated detector concepts are currently under discussion. Detector optimization and physics performance are an integral part of these studies and require a well maintained and easy to use suite of software. Given the limited available person power for these efforts, a common approach is vital. The Key4hep project aims at providing and maintaining a turnkey software stack with all the necessary tools for future collider studies, covering event generation, detector simulation, reconstruction and analysis. A consensus for collaboration was originally reached by the FCC, CLIC, ILC and CEPC communities [1] and the project has also been able to attract contributions from several other communities, most notably the EIC and the Muon Collider [2].

A core goal of the Key4hep project is to extend existing software solutions, providing the necessary glue to ensure compatibility of the different components. This allows all the communities to profit from common developments and simultaneously fosters consolidation of different approaches, as

well as cross-community communication. Only in limited areas, where no suitable solution or tool already exists, does Key4hep embark on new developments.

The major goal of this deliverable was the provision of a fully functional software stack that enables physics and detector development studies. Achieving this goal entails bringing several partially interdependent components to production readiness, or at least to the advanced prototype stage. It also requires the development of reliable and consistent build and deployment workflows to distribute the resulting software ecosystem as well as providing documentation and means to report issues back to the developers.

Some core components of the software stack, e.g. the ones related to data formats and persistency, need to be stabilized with higher priority as changing them later on becomes more and more difficult as further developments are built on top of them. Hence, finalizing the PODIO toolkit which forms the basis for data I/O was one of the key goals for this deliverable. Properly integrating this into the core framework components is a natural extension of this and was also tackled.

DD4hep is a well-established tool for detector geometry description, supporting simulation. Extending its capabilities to also cover the digitization of the simulated hits, such that the resulting events resemble the events that would be obtained by a real detector was also part of this deliverable. The basic framework and functionality have been achieved.

A final goal of this deliverable was an R&D study on frameworks to manage heterogeneous resources. Since the recruitment of dedicated person power to work on this study failed without suitable candidates, this study only saw very limited work, which was mostly carried out by the associated partners at IHEP and Shandong University. However, integrating these back into the Key4hep ecosystem has not yet started and will require significantly more person power.

In this report a brief introduction to the components that received significant developments for achieving the goals of the deliverable is given in Section 6, where also some more detailed information about the specific goals for each components are listed. The results are then presented in Section 3. Section 4 gives a summary as well as the wider context and discussion of possible next steps and future work.

2. CORE KEY4HEP COMPONENTS

Figure 1 shows a typical high energy physics (HEP) event processing workflow. Starting from an event generator that produces stable particles, detector simulation propagates these particles through the detector, recording energy deposits in active detector components. Then various data processing steps, also referred to as algorithms or processors, are run in sequence to reconstruct the decay particles from the raw detector signals, before they are used in analysis to infer the original physics reaction. Each step in this chain performs one small piece of reconstruction or analysis. This modular approach allows the parallel development of reconstruction algorithms, but replacement of parts of the reconstruction with alternative approaches. It is possible to run all of these steps in one go, conducted by the event processing framework. However, in Key4hep, event generation and detector simulation are typically run as stand-alone processes and only reconstruction and analysis are run within the event processing framework.

The core components in this workflow are

- an Event Data Model (EDM) that allows for the exchange of information between the different algorithms;
- an event processing framework that conducts the different reconstruction and analysis algorithms;
- and a detector geometry description that is available to all algorithms.

Key4hep has chosen the Gaudi event processing framework [3], which was originally developed by the LHCb experiment, but has since also been adopted by the ATLAS experiment. For detector geometry description the DD4hep toolkit [4] is used. The shared and common EDM has been newly developed for Key4hep in the form of EDM4hep. A brief description of these components, presenting the core ideas and their respective goals during this project follows. The discussion of the developments towards these goals and results will be given in Section **Error! Bookmark not defined.**.

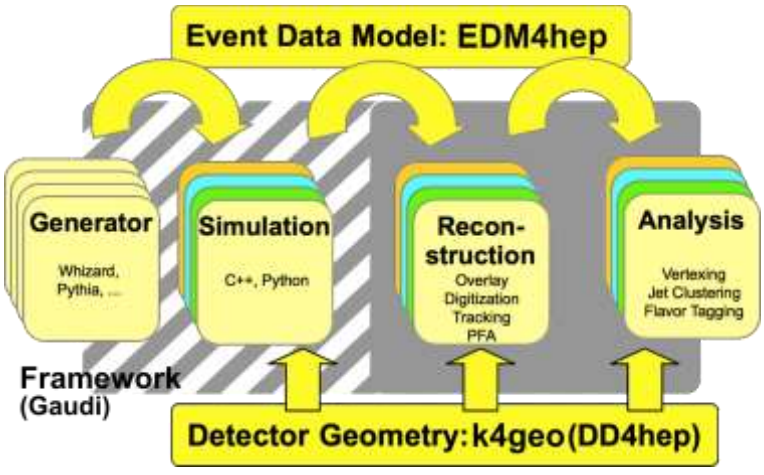

Figure 1 Schematic flow diagram of a typical HEP event processing framework and the core components that are involved.

Another key goal of Key4hep is the smooth transition of the participating communities from their existing software solutions towards the ones provided by Key4hep. Most importantly, the integration

of the almost two decades of work and experience gathered by the linear collider communities in the form of iLCSoft needs to be made available without changing the existing software. This has been achieved via the k4MarlinWrapper, which will also be described briefly below. The finalization and validation of this component has also been the main ingredient towards achieving MS47 [5].

Finally, Key4hep needs a way to build its software packages consistently and reliably in order to allow for frequent deployments and testing by users. This is achieved via the spack package manager [6], which is used to build all Key4hep releases as well as nightly builds, which are necessary to iterate on and test the latest developments.

2.1. EDM4HEP – THE COMMON DATAMODEL

The EDM not only defines the structure in which different software components exchange data. It also acts as the language in which physicists express their ideas for reconstruction and analysis. Additionally, since it also defines the format in which data is stored on disk it also plays a key role in data preservation efforts.

For Key4hep a new EDM has been developed in the form of EDM4hep. It is inspired by the LCIO EDM [7], developed by the linear collider community, and FCC-edm. A focus has been put on usability in reconstruction and analysis. The current EDM4hep schema can be seen in Figure 2.

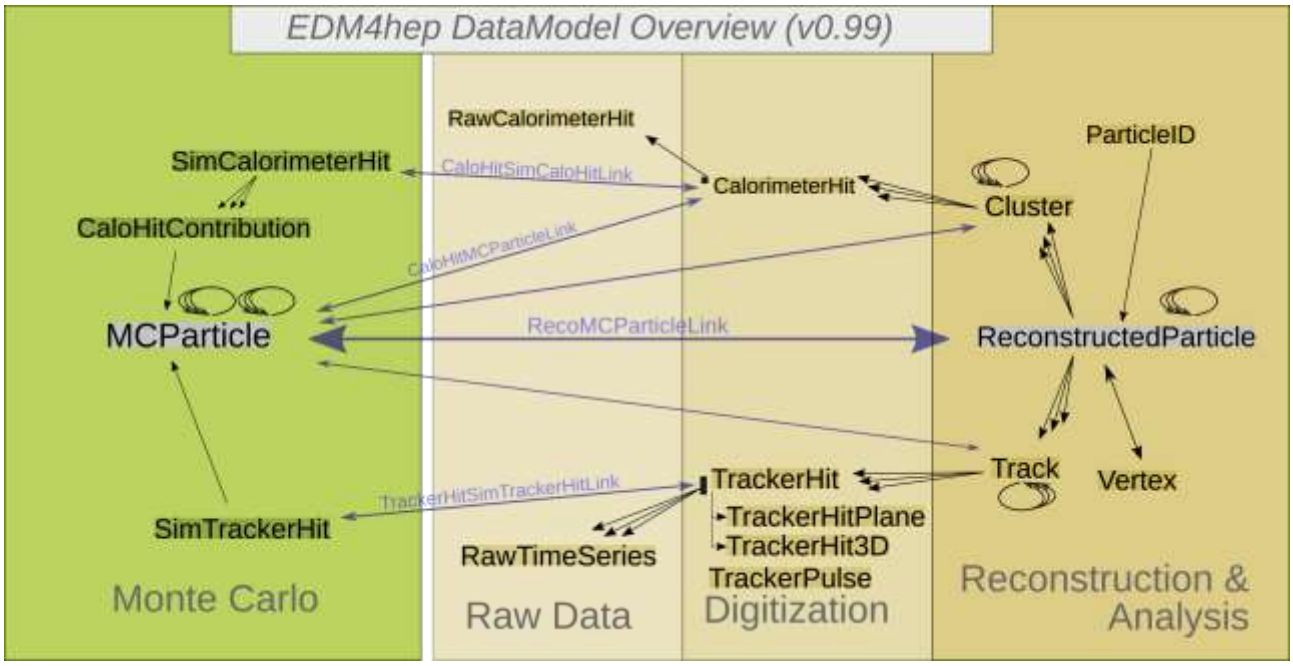

Figure 2 Schema diagram of the current version of EDM4hep showing the available datatypes as well as their relations among each other.

While LCIO has proven to be a good starting point for the schema of EDM4hep, some conceptual changes are necessary to remedy certain design choices in LCIO that were taken when multithreading was not a concern. Additionally, this switch in datamodels is a good time to reflect on the things that worked well for LCIO and those that did not. Hence, the main objective for EDM4hep was to finalize the schema such that potential future changes can be implemented transparently through schema evolution, thus, keeping backwards compatibility for files written with an older version.

2.1.1. PODIO – A datamodel toolkit for HEP

Given the central role an EDM plays in the core of event processing for HEP experiments, a performant implementation is crucial. Nowadays this implies thread-safety, as well as cache friendliness. Both of these goals are achieved by leveraging the PODIO EDM toolkit, which was originally developed in the AIDA-2020 project. They main idea for PODIO is to generate low level implementation code from a high level definition of the EDM, where users can focus on "what they want" rather "how they want it". This approach allows generation of an efficient and thread-safe implementation with an easy to use interface.

The main goals for PODIO in the AIDAinnova project are closely related to the requirements necessary to achieve the objectives for EDM4hep. Most importantly, a mechanism for schema evolution for generated EDMs has to be designed and developed. Additionally, several other features have become necessary in order to support all the workflows required by Key4hep and EDM4hep. These include:

- the possibility to do multithreaded I/O;
- support for *interface types* that allow to access data from different datatypes via a common interface;
- and the option to extend an existing EDM with custom types for prototyping.

2.2. K4FWCORE – CORE FRAMEWORK COMPONENTS

In order to do actual event processing with the Gaudi framework some core components have to be implemented on top of it. Most importantly, the EDM and related I/O functionality have to be integrated to make event reading and then processing possible. These components are provided by the k4FWCore package [8]. Apart from these I/O related components k4FWCore also hosts some generally useful services and functionality, e.g., services to easily retrieve the detector geometry or reproducible random number generator seeds in multithreaded contexts.

At the start of AIDAinnova the basic functionality for reading and writing EDM4hep was already present. However, it was still based on a legacy flavour of Gaudi that did not yet allow for multithreaded running. Hence, the main objective for the framework core components was to upgrade them to be compatible with the most recent versions of Gaudi, making it possible to run algorithms on multiple threads. Furthermore, many of the changes in PODIO and EDM4hep also require continuous adjustments to the core components.

2.3. DD4HEP – THE COMMON DETECTOR GEOMETRY TOOLKIT

To get reliable results a consistent description of the detector geometry through all stages of the processing workflow is necessary. This includes the correct description of sensitive, but also passive, material during simulation as well as approximate material distributions to facilitate reconstruction algorithms. The DD4hep toolkit has been developed in previous iterations of AIDA projects with the main goal of providing this consistent description from one source of truth. The specific representations are computed on the fly in memory from a canonical detector description that is loaded from a configuration file.

The core functionality of DD4hep is by now rather complete and it is a very well-established toolkit in the HEP community; it is not only used for future collider studies, but also by currently running LHC experiments, e.g., LHCb and CMS. The goal in AIDAinnova was to implement an extension of the toolkit that allows the digitization of simulated energy deposits.

2.4. K4MARLINWRAPPER – INTEGRATING EXISTING SOFTWARE

The event processing workflow used by the linear collider communities in iLCSoft is conceptually identical to the one used in Key4hep depicted in Figure 1. The main difference is the actual software components that are used. Where Key4hep uses Gaudi as its event processing framework, iLCSoft uses Marlin [9] and, as already mentioned, LCIO is used as EDM instead of EDM4hep. The k4MarlinWrapper package provides the necessary Gaudi algorithm that wraps Marlin processors and allows to run them unchanged within the Gaudi framework of Key4hep. It also comes with tools for the automated conversion of the Marlin runtime configuration in XML format into the Python format that is used by Gaudi.

A key ingredient for the smooth transition from iLCSoft components towards Gaudi based algorithms is the possibility to run existing software with new algorithms. This requires a conversion in both directions between the involved EDMs, since Marlin expects its inputs and outputs to be in LCIO, whereas Key4hep Gaudi algorithms use EDM4hep. The *MarlinProcessorWrapper* algorithm provides the necessary hooks to configure this on-the-fly conversion as necessary. This makes it possible to convert the necessary inputs from EDM4hep to LCIO before the wrapped processor is executed and vice-versa after it has finished, as shown in Figure 3.

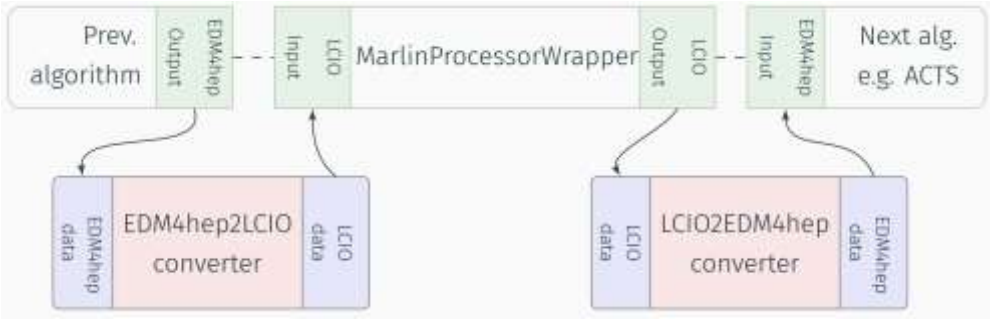

Figure 3 Flow diagram depicting the scheduling of a wrapped Marlin processor and the adjacent EDM converters in conjunction with Gaudi algorithms that run before and after it.

Developing the necessary software components to allow for this kind of workflow was a crucial goal towards a turnkey software stack for doing fully fledged physics studies. The key deliverables for interoperability were

- the Gaudi algorithm that wraps Marlin processors;
- a Gaudi algorithm for ingesting LCIO input files;
- the conversion script for translating the runtime configurations;
- and the EDM conversion library that supports conversions in both directions.

Since Marlin itself does not support multithreaded execution many Marlin processors themselves are not threadsafe. Hence, executing wrapped processors in a multithreaded context was not considered a goal and consistency with Marlin was prioritized. Nevertheless, many of the involved components are in fact threadsafe.

3. RESULTS

All the software that has been developed in the context of the AIDAinnova project is hosted on public GitHub repositories, most of them as part of the Key4hep GitHub organization [10]. All the software is open source licensed (mostly as Apache 2). GitHub is also used to manage issue reports and help the planning of development milestones. For some of the software packages Zenodo references [8, 11-13] have been created to make the software citable and thus to facilitate the attribution of dedicated versions used for physics publications.

3.1. A FIRST STABLE RELEASE FOR PODIO

At the start of the AIDAinnova project the PODIO toolkit was at the mature prototype stage but could not yet be used for production purposes. Given its central role in Key4hep, PODIO needs to be very stable and provide strong backwards compatibility guarantees. In order to achieve that, several developments were necessary and in what follows the most significant ones will be outlined. The first stable version of the PODIO toolkit was released in June 2024 [14], this included a comprehensive set of documentation, including an API reference for the C++ and Python interfaces provided by PODIO [15].

3.1.1. New multithreading concept for PODIO

While the generated EDMs of PODIO were always designed to be thread safe, some accompanying components were not originally developed with that goal in mind. Most importantly, an example implementation of an *EventStore* has been used in several places throughout Key4hep, far outside its original scope. This component provides the essential interface to access collections of objects that are read from files and that should be persistified in the end. Since it was only designed to show some basic principles, thread safety was not a concern and consequently any software that used it was not threadsafe either. This was addressed by designing the I/O components, as well as the interface that users see, from scratch making the usage in multithreaded contexts the key consideration. PODIO I/O components need to remain usable in and outside of an external framework that controls the threading context. Hence, one design choice was made to assume that file readers and writers need to be synchronized externally, i.e. PODIO assumes that at most one thread will access them concurrently. Conversely, the newly introduced *Frame*, a generalized event data container, provides concurrent access from arbitrarily many threads as shown in Figure 1Figure 4(b).

The Frame owns all collections to which it provides access and putting a new collection into an existing Frame also implies relinquishing ownership. These properties are also reflected in the Frame interface, as shown in Figure 4(a). Apart from collection data, it is also possible to store some additional metadata or parameters in each Frame. Conceptually the Frame is deliberately kept very general, such that the meaning of any given Frame is effectively defined by its contents. This allows its usage for typical HEP concepts like events and runs but also makes it possible to use Frames for, e.g., detector readout frames. To further generalize the usability, Frames can be constructed from almost arbitrary data, facilitating the addition of additional I/O backends.

The introduction of the Frame and the newly designed multithreading concept for PODIO and EDMs that are generated by it also required some minor modifications to the generated implementations for some classes. However, these changes were completely transparent to users.

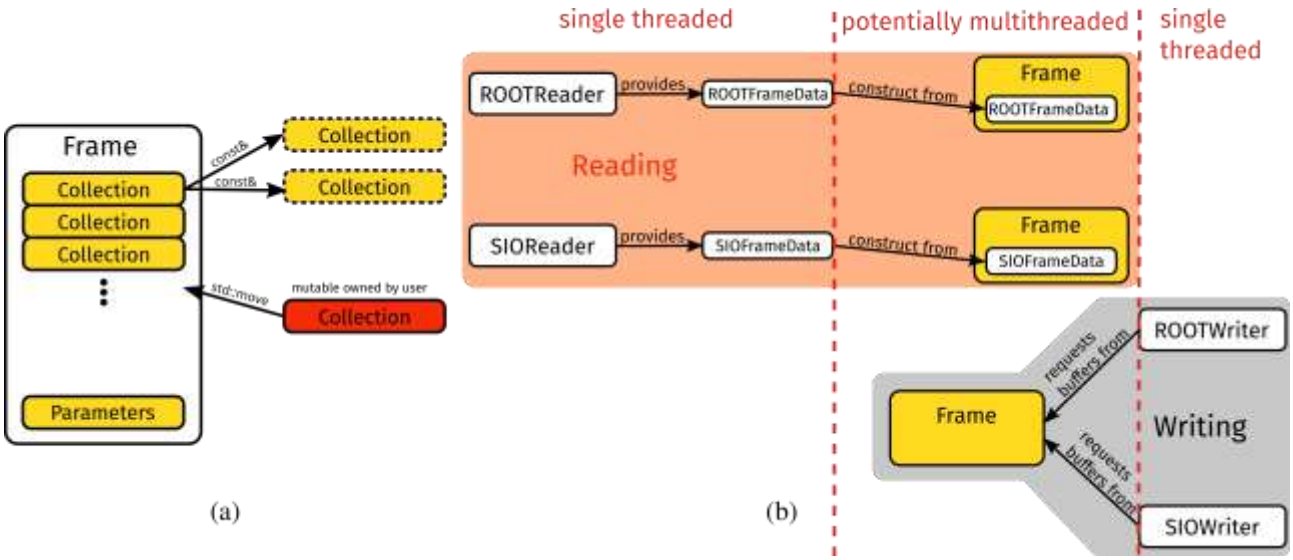

Figure 4 (a) schematic view of how collections are stored in a Frame and how access is granted, and (b) flow diagram of how data is read from file and Frame construction and how it relates to threaded operation, resp. which components have internal synchronization and which require external synchronization.

3.1.2. Schema evolution facilities

The possibility to evolve the schema of a data model, while keeping backwards compatibility, is crucial to ensure that data written with a previous version remains readable and usable. This usually requires the capability to read the data in the old version, then evolving it to the new version on-the-fly in memory. However, some I/O backends, e.g. ROOT, can also do some parts of this evolution themselves. Hence, PODIO's schema evolution mechanism conceptually has to be able to support full and partial schema evolution. A general design decision was made to implement all the necessary building blocks to do arbitrary schema evolution, but to only implement the actual schema evolution functionality for specific schema changes on demand. Otherwise, a large amount of person power would be expended with no clear benefit. These generic hooks also allow users to provide their own customised schema evolution functionality that will be executed at the correct point. The original set of supported schema changes has been based on the schema evolution capabilities of ROOT, with the additional possibility of renaming data members.

The first step to schema evolution is the detection of schema changes. This is achieved via a dedicated Python script that compares two schema versions of a data model definition in YAML format. The script effectively compares the data members and relations of the datatypes, ignoring all parts of the data model definition that are not reflected in the on-disk data. As a core feature it has to reliably detect schema changes that are not yet supported, to avoid accidentally breaking backwards compatibility to data that has already been written.

The second step of schema evolution is the generation of the code that actually does the schema evolution and that is hooked into PODIO's facilities. Here no code generation for generic schema evolution is done yet. The only mechanism that is not yet transparently supported by ROOT is the renaming of member variables. However, ROOT's schema evolution mechanism also allows modelling these changes, by hooking into its schema evolution facilities. Hence, for these cases, PODIO generates the additional code that allows to do these changes within the ROOT schema evolution facilities.

The schema evolution facilities have been designed to hook directly into the newly designed and implemented I/O system of PODIO described in Section 3.1.1.

3.1.3. Extending existing data models

Developing new detector concepts might require developing new data types to store the data that can be used for either simulation or reconstruction. Existing data models usually have policies focusing on stability which run counter to the dynamic nature of the prototyping phase. Nevertheless, prototyping can rely on already existing and well defined datatypes. To enable this form of interaction a new mechanism that allows the extension of an upstream data model has been added to PODIO. This involved adapting the YAML schema validation to account for a potential upstream data model additions, to verify that all datatypes that appear in relations actually exist and can be used.

Currently extending data models is limited to only one level of extension. This deliberate choice encourages users to try and stabilize their data types rather quickly, so that they can then be included into the upstream data model, e.g. EDM4hep. The mechanism is used by several communities at the moment. Most prominently by EIC, who implement their own EDM4eic [16] data model on top of EDM4hep. Additionally, it is also exploited by dual-readout calorimetry groups to study and establish their simulation and reconstruction workflows.

3.1.4. Interface data types

In some cases, it can be useful to refer to several similar data types via one conceptual interface. The prototypical example is tracker hits, where different tracking detectors might produce different tracker hit data types. Nevertheless, a track should be able to refer to all of the hits that were used to construct it. Ideally, these relations can also keep an ordering for the hits.

To collectively allow for these features *interface types* were developed and introduced as a new category in the YAML grammar for data model definitions. Interface types are defined via the functionality they should provide, as well as a set of data types that support this interface. These supported types can then be used wherever such an interface type appears in a relation in the data model. Contrary to regular datatypes, interface types do not have a corresponding collection where the data is stored, rather they only exist in memory and wrap the underlying data objects. To keep the thread safety guarantees of PODIO generated EDMs, interface types only allow immutable access to the data. Interface types can allow the full access to the underlying data objects via some type checks and type casting.

3.1.5. Developments after v1.0

Several additional developments have happened in parallel to the previously mentioned larger features and are briefly summarized here, since they also impact other components.

For the HL-LHC the ROOT team has developed a new I/O format, RNTuple [17], with improved read speeds and better event data compression. To test PODIO's ability to accommodate different I/O backends a new backend to support the RNTuple format was developed.

The readers and writers that PODIO provides for the different supported I/O backends are rather low level. Hence, some of the details of the I/O backend are still visible in the interfaces, which makes it hard to switch between different backends at runtime. To solve this, generic *Reader* and *Writer* components have been developed that hide all these details, while supporting all available backends.

The format in which PODIO generated EDMs are stored in ROOT RNTuple files are already well suited for columnar analysis, e.g., with RDataFrame [18]. However, resolving the relations between different objects is cumbersome and error prone when not going through the PODIO generated interfaces. A new *data source* has been developed that allows the use of data inside RDataFrame through the PODIO generated interfaces. This first implementation enables all features, however, still requires performance improvements.

The possibility to link arbitrary objects is a feature that is frequently used in LCIO. Hence, also EDM4hep needs to support that feature. Initially, this was done by handcrafting datatypes that implement this feature via one-to-one relations. However, this approach does not scale beyond a few involved datatypes. To alleviate this, a generic *Link* has been implemented in PODIO that can be used to link objects of arbitrary datatypes as long as they have been generated by PODIO. To facilitate the usage of this new feature, the *links* category has been introduced into the YAML grammar, keeping the important property of the full EDM being defined in one place.

Finally, improvements were made to the Python interfaces of PODIO itself and the data models that are generated by PODIO. Most importantly, several *pythonizations* have been developed that allow an opt-in for certain behaviours of the Python interface of the generated data models. One of these pythonizations makes it possible to ensure that attributes that have not been generated by PODIO can be accessed, while another one ensures that out of bounds accesses raise an exception in Python.

3.2. FINALIZING THE EDM4HEP SCHEMA

Given that the EDM4hep schema is largely based on the one from LCIO, which has already been used in production for more than a decade for several LC concept studies, the majority of the design choices have already been proven for reconstruction and analysis. Nevertheless, some conceptual issues have also been inherited, mostly related to use in multithreaded contexts, which was not yet a concern when designing LCIO. Some other issues were related to the different implementation approaches followed by LCIO and EDM4hep with PODIO respectively, as well as a general effort to remove some of the unused features of LCIO.

Although the changes that are discussed below only affect a rather small portion of EDM4hep, it became clear that introducing them in a fully backwards compatible fashion would require an unreasonable amount of additional effort. Hence, the decision was made to make a clear cut after version 0.10.5 of EDM4hep to introduce several breaking changes at once and to strive for limited backwards compatibility from the 0.99 pre-release version onwards: the limitations concern newly introduced datatypes for storing generator information, which have not seen any use beyond prototyping.

3.2.1. Usability in multithreaded contexts

A somewhat common approach in the LC reconstruction and analysis workflows is an iterative refinement of reconstructed objects, e.g., the estimation of the energy loss, dE/dx, for a track will be done after the track fit. In LCIO and Marlin it was possible to simply store this information into an already existing track in a separate processor. However, due to the much stricter requirements with regards to mutability this approach is impossible in Key4hep and EDM4hep, where objects are only mutable upon creation, i.e. during the track fit in this case. Another example is attaching ParticleID

objects to a reconstructed particle from different particle identification (PID) algorithms. While easily possible in LCIO, this is again prohibited by the immutability constraints of EDM4hep.

These are conceptual issues that cannot be easily addressed by choosing a different implementation technique. Rather, they require some structural changes to the definition of the EDM. For example, the dE/dx information has been moved into a dedicated datatype, *RecDqdx*, with a one-to-one relation to a track for EDM4hep. Now it is possible to use the tracks for the estimation of the energy loss without having to mutate them, since the results of the estimation go into newly created objects that allow retrieval of the track from which they have been computed.

In the case of the relationship between reconstructed particles and ParticleID objects, the solution was to simply reverse the direction of the relation: from a one-to-many relation from the reconstructed particle to the ParticleID object, to a one-to-one relation in the other direction, as shown in Figure 5. Similar to the case of tracking, this reversal allows the use of the reconstructed particle for the determination of new information, without having to mutate it to attach this information. This change also requires a slightly different approach in analysis, which will now start from the ParticleID object rather than from the reconstructed particle, especially if only one PID algorithm is involved. Additionally, new utilities have been introduced to EDM4hep to enable the same workflows that have been available within LCIO. A small side effect this change removed an unused relation between clusters and ParticleID objects from EDM4hep.

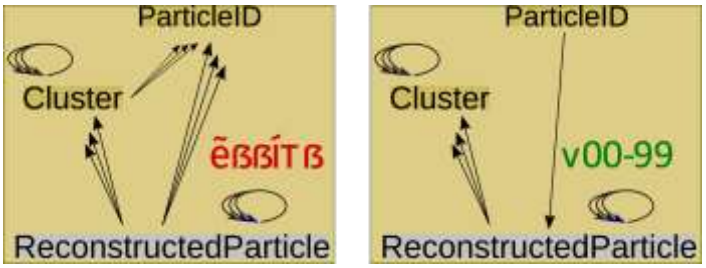

Figure 5 Conceptual changes to the relations between ParticleID objects and ReconstructedParticle objects for EDM4hep for a consistent multithreading concept.

3.2.2. Other developments towards v1.0

As already mentioned in Section 7 several of the requirements for PODIO are directly related to requirements for EDM4hep and many of the results for PODIO, presented in Section 3.1, are directly applicable to other developments for EDM4hep as described below.

A dedicated *TrackerHit* interface was introduced and is now used in track objects to relate tracker hits to tracks. It covers the *TrackerHitPlane* and the *TrackerHit3D* datatypes and an extension to cover a dedicated tracker hit type for drift chamber studies is currently under discussion.

The previously existing *Association*s that bridge the simulation world to the reconstruction world to allow for efficiency and fake rate estimations in simulation have been moved to use the newly developed *Links* from PODIO. This was done in a two-step process to make the transition as smooth as possible. The first step was a renaming of the data types, keeping the existing names with a deprecation warning. This made it possible to easily see all places that needed changing, without breaking anything. Once all the changes were in place, the original data types were replaced with links in the YAML file. This change was made transparent for the user by manually injecting the

necessary schema evolution code into the PODIO facilities, also making this the first real world usage of that system.

In a dedicated meeting with authors of HepMC3 [19] the definition of MC particle contents was reviewed. Apart from some floating-point issues, which were lost in the translation from LCIO to EDM4hep, this also uncovered a few properties that were added to LCIO that should not be in a generic EDM. This was because these properties are not usable correctly without additional generator information or because they are generator specific, potentially even depending on the implementation details of a specific generator. Thus, the *colorFlow* property has been removed and the *spin* property will be replaced by the better defined *helicity* property, in accordance with similar developments in HepMC3.

Apart from the data content of EDM4hep, attention has also been paid to formalizing the storage of related metadata. This mostly relates to introducing dedicated variables that allow for a central and consistent definition of names for keys that are used to store metadata, such as encoding strings or the names of special collections and configuration parameters.

3.3. MULTITHREADING WITH K4FWCORE

The PODIO and EDM4hep integration in the core components for framework usage, developed inside the k4FWCore package, at the beginning of AIDAinnova was targeting a release of Gaudi that was still designed for single threaded usage. Additionally, the implementation for reading EDM4hep files was a re-implementation of the reader available from PODIO still based on the original example EventStore, where not all developments from upstream PODIO have made its way back to the implementation in k4FWCore and vice versa. Hence, some small incompatibilities in the resulting files manifested themselves in spurious crashes when trying to read them. This was remedied by completely reworking the PODIO integration as reported in Section 3.3.1.

This refactoring also was an ideal opportunity to simultaneously switch to a newer release of Gaudi that was designed for multithreading, so called *Gaudi Functional* [20]. With this development algorithms become pure functions, without mutable state, making them trivially thread safe. The framework interface is designed such that it becomes possible to easily collect information about the input and output collections, allowing for intra-event scheduling of the algorithm execution. The work that was necessary to enable this for some of the special use cases of Key4hep are described in Section 3.3.2.

3.3.1. Rework of the PODIO integration

The situation with similar but slightly differing implementations for framework usage and standalone PODIO applications was highly undesirable. Hence, the redesign of the I/O system in PODIO, as described in Section 3.1.1, was done with this use case in mind. This made it possible to use the readers and writers provided by PODIO to provide the necessary I/O functionality for core framework components. A new data service, based on the newly introduced Frames, was developed, keeping the existing data service, based on the EventStore, alive for a transition period to ease migration.

The newly developed data service based on Frames also made it possible to introduce a better defined concept for metadata. In the original implementation mutable access to the metadata was possible at all stages of the event processing, i.e., during the initialization and finalization of algorithms, as well as during their execution for every event. This is quite obviously not threadsafe and also ran counter

to the definition of metadata, which should not be event level data. Hence, in the new Frame based implementation, a dedicated metadata Frame was introduced: adding metadata is only allowed during algorithm initialization or finalization, while reading metadata is possible at all stages of event processing. A dedicated exception is granted to the EDM converters, which are provided by the k4MarlinWrapper components, as this hard split between metadata and event level parameters does not exist in LCIO and Marlin.

Overall, the interfaces and names of the k4FWCore components that were changed during this refactoring were kept the same as before, renaming the now legacy components instead. This made it possible for the majority of use cases to be switched over transparently, without the need for any user level code changes. Only algorithms that put metadata into the data service needed adaptions, which were minor.

3.3.2. Functional algorithms and related framework support

The rework of the PODIO integration described in Section 3.3.1 does not support multithreaded execution of algorithms on its own. Due to a lack of documentation for Gaudi, a non-negligible amount of effort was spent in understanding how this could be achieved before being able to implement the necessary services and interfaces. In the end a new *IOSvc* was implemented that supports running on multiple threads and integrates closely with the readers and writers provided by PODIO. Currently, this is a partially parallel implementation of the same functionality that has been described in Section 3.3.1 already, with the midterm goal of fully switching to the IOSvc only. To achieve this some components of the k4MarlinWrapper still need some minor adjustments.

With the IOSvc it is possible to define algorithms in Gaudi Functional style and to run workflows on multiple threads concurrently. k4FWCore also provides some thin wrappers for the configuration of the IOSvc and the correct sequencing of the readers and writers, such that they are correctly configured by default for most use cases. On top of the IOSvc, k4FWCore also has some dedicated re-implementations of some of the Functional base algorithms that are shipped with Gaudi. These can be combined freely with each other, but the former provide some additional functionality that is not available from the regular ones: most importantly they have built-in support for accessing EDM4hep, or more general any PODIO generated EDM, collections and allow a variable number of input and output collections. The latter is something that is commonly found in existing Marlin processors.

Most algorithms should be implementable as functional algorithms as only very few should actually require mutable internal state and synchronization across threads. However, the latter use cases also need to be supported; and they in fact are already by Gaudi. Here k4FWCore does not need to do any additional work, as these algorithms and the functional algorithms are all built on top of the same underlying interface, where it is already possible to mix functional and non-functional algorithms. The main thing that has to be kept in mind for these use cases is that algorithms that are not re-entrant become sequence points for the scheduler, which will ultimately be the performance bottlenecks of the workflow execution.

3.3.3. Other developments

Other smaller developments targeted some usability concerns and user feedback. These include the creation of a geometry service that allows loading a DD4hep geometry and making it available to algorithms in just two lines of Python configuration.

Another new feature was the introduction of a *UniqueIDGenSvc*, which allows the generation of reproducible random seeds for algorithms, based on their names and the event and run number. These are necessary to reproduce the same sequence random numbers for a given algorithm and a given event, regardless of how many events have been processed before or how many threads are running concurrently.

Finally, the k4run script that is used to launch the execution of a Gaudi based workflow has been refactored and cleaned up. It now features better error messages that point users to the issues in the configuration files they provide, better command line argument parsing, and functionality to support the dynamic loading of several potentially nested configuration files. The latter allows breaking apart a very large configuration for a full reconstruction and analysis chain into several smaller files that can be assigned to specific part of the workflow, e.g., tracking or high-level reconstruction.

3.4. DDDIGI – THE DIGITIZATION EXTENSION FOR DD4HEP

The core features of a framework that allows for digitization within the DD4hep toolkit, *DDDigi*, have been designed and implemented. The enumeration of all sensitive detector components was identified as the most challenging technical task. The basic features that are currently implemented include energy deposit smearing for tracking and calorimetry devices, handling of multiple interactions, e.g. for event spillover simulation, and input and output via EDM4hep.

All the necessary components for basic functionality have been implemented and tested. However, no realistic use-cases could be identified yet, to properly assess the usability and applicability of the implementation.

3.5. INTEGRATION OF LC RECONSTRUCTION

All of the necessary components linear collider reconstruction were developed on the timescale of MS47, where more details and information can be found [5]. The key result was the validation of all used components via comparisons of the full reconstruction chains of the ILD and CLIC detector concepts as running those requires all of them to be used in a non-trivial fashion. Excellent agreement has been found between running regularly via Marlin, using LCIO as the EDM, as well as using Gaudi, creating EDM4hep outputs.

Since a detailed report is already available that describes all the involved components [5], only a very brief overview of the main components will be presented here. The *MarlinProcessorWrapper* and the *LcioEvent* Gaudi algorithms provide the functionality to wrap and run existing Marlin processors within the Gaudi framework, as well as ingesting data in LCIO format. The conversion from LCIO to EDM4hep and vice-versa is implemented as a standalone library with minimal dependencies. This allows to be used in several places, e.g., via a standalone executable. Large fractions of the work have gone into generalizing the interface of this conversion library such that it can be performantly used in all cases.

After the successful validation in MS47, the focus switched towards starting the migration of existing components towards native Gaudi algorithms and the integration of newly developed algorithms into the existing reconstruction chains.

A task that combines both of these aspects was carried out for particle flow reconstruction for liquid Argon (LAr) electromagnetic calorimeters, specifically for the ALLEGRO concept [21] using PandoraPFA. The migration efforts focused mainly on porting the existing integration of Pandora from a Marlin processor to a Gaudi algorithm. Since Pandora effectively acts as its own micro-framework, this mainly involved small scale rewrites and translations of the glue code that bridges between the transient event store of Gaudi and the data structures inside Pandora.

The new developments that were necessary are all related to the differences in the calorimeter geometry and providing Pandora with accurate estimates of the detector material at a given shower depth. The existing calorimeter concepts from the LC studies were all based on highly granular sandwich calorimeters, making this estimation straight forward. However, the LAr detector consists of steel/Pb absorber plates and readouts immersed in liquid Argon, inclined at an angle of 50° relative to the axis normal to the beam axis. Hence, estimating the material depth can vary from readout cell to readout cell and a new method of estimating material properties was developed. It is based on the *MaterialManager* of DD4hep and allows for a dynamic, model-independent determination of material properties which can then in turn be used for particle flow reconstruction for the ALLEGRO concepts [22].

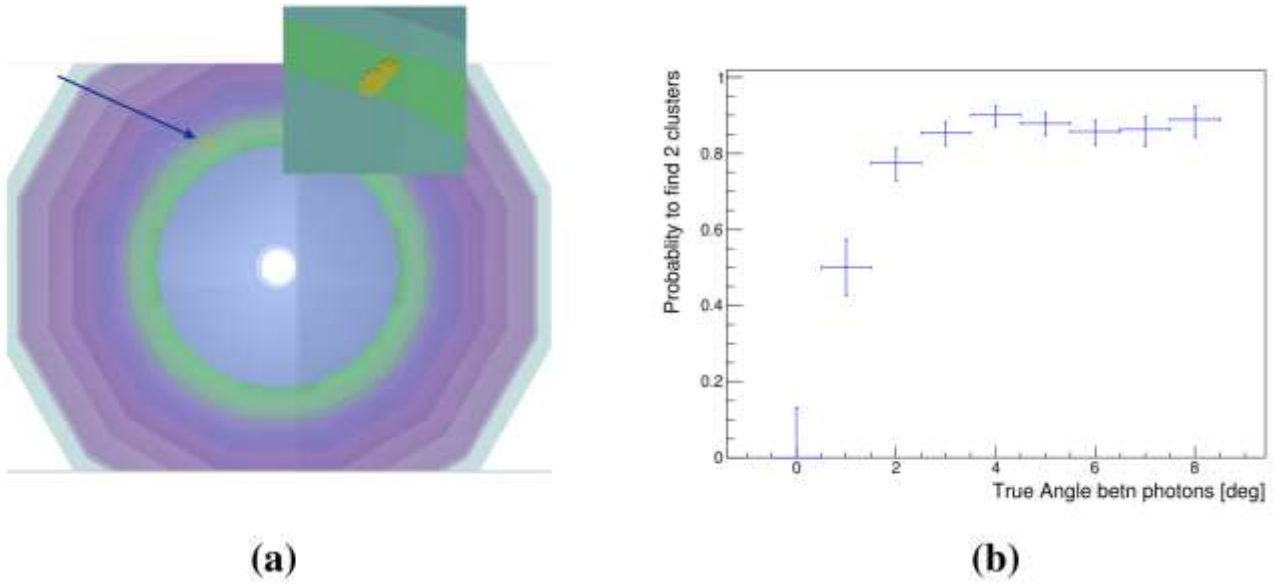

Figure 6 (a) A photon shower inside the ALLEGRO detector concepts reconstructed via PandoraPFA using the newly developed material estimation technique. (b) The diphoton separation efficiency as a function of the true angle between the two photons that can be obtained via PandoraPFA reconstruction in the ALLEGRO detector concept. From [22].

A photon shower that has been correctly clustered, as well as the efficiency of separating two photons depending on their angle is shown in Figure 6. For a fully accurate and unbiased energy reconstruction some dedicated calibration work remains to be done.

3.6. BUILDING AND DEPLOYING KEY4HEP

Early feedback on new features is an important mechanism to identify issues and bugs before the production releases, and also to try and assign effort according to the priorities of the project. However, a certain amount of stability in the software ecosystem is crucial to do any larger scale productions with runtimes in the order of months. Both use cases are covered by Key4hep, which provides nightly builds for early testing and development as well as dedicated numbered releases that

provide the necessary long-term stability. Nightly builds are done by collecting the current latest commit of all packages that are part of Key4hep each night and are usually retained for only a few weeks. The numbered releases, on the other hand, are built from the latest tagged versions of all packages and are retained indefinitely.

Releases and nightly builds are both done using the Spack package manager, which builds all packages from scratch. Key4hep specific package recipes are provided via an additional package repository [23] that is maintained solely by Key4hep developers. Whenever a package becomes useful for general usage outside of Key4hep, it is moved to the central Spack repository, e.g., PODIO, EDM4hep and DD4hep. Spack itself is still under active development and the package repository, together with the core package manager functionality, are maintained in one repository. Hence, potentially breaking changes in the core functionality cannot be easily isolated from simple updates of software packages. Key4hep addresses this by choosing a specific Spack version or commit and then applying selected patches to keep the required package recipes up to date. Every few weeks the nightly builds move to a newer version of Spack to identify breaking changes early and to reduce the number of patches that need to be applied. The commit hash of the Spack version that was used is also persistified into the deployed Key4hep stack, to make reproducible builds possible.

After being built, the software stacks are deployed to CVMFS which allows them to be used on all machines that run a suitable OS and on which CVMFS is available. The Key4hep stacks are currently built for AlmaLinux9 and Ubuntu22.04. The former is running on machines provided by CERN, DESY and various other research institutes, while the Ubuntu builds allow usage and development work on personal laptops. The switch from the previously used CentOS7 to AlmaLinux9 was straight forward due to the usage of Spack and did not require any dedicated migration effort beyond adjusting configuration files. It did, however, entail fixing a few smaller issues in various packages due to updated compiler versions.

3.6.1. Usage of CI/CD

Continuous Integration (CI) and deployment (CD) are software engineering practices that focus on keeping the latest version of a software package building and running, even for intermediate developments. Key4hep also follows this model and does not solely rely on nightly builds to detect issues. Instead, each software package has a dedicated test-suite that is run for every proposed change to a software repository to ensure that only changes that do not introduce bugs are actually accepted.

To facilitate the adoption of these practices, not only in the core packages that are maintained by the core developers but also by repositories from users, dedicated CI workflows that are run on publicly available resources accessible via GitHub have been developed. These workflows are automatically configured to run on every commit or pull request and, at a minimal level, ensure that the software package builds and does not break the nightly builds. These CI workflows use the nightly builds available from CVMFS, such that the latest developments are always available for development and testing.

Similar workflows are also in place to automatically build and publish documentation for several packages whenever a change occurs. In case of the central documentation, covering the basics of Key4hep and also providing tutorials and how-to guides, these workflows are also run on a daily schedule, since they gather information from several different places.

Central configuration for code formatting and static code quality checks are also run via automated CI workflows. This allows for quicker turnaround times on code review as reviewers can focus on the actually important changes, rather than having to deal with inconsequential formatting issues. This also frees precious resources and makes it possible to maintain and develop the Key4hep software stack with the few people that are working on core developments.

4. CONCLUSIONS AND POSSIBLE FUTURE WORK

With the exception of the heterogeneous resources R&D study the goals for this deliverable have been achieved. The PODIO toolkit has been finalized and a stable production ready version has been released in June 2024. Additional developments have gone into subsequent minor releases. These developments form the basis for the backwards compatibility requirement that Key4hep needs in order to serve as a long-term platform for studying future colliders and their detector concepts. The build and deployment pipeline handles nightly builds and has also been used to deliver several release builds to users. All these developments are already in active use for physics studies. The basic functionality for the digitization extension for DD4hep, DDDigi, has also been implemented and tested and is ready for usage in physics studies.

Apart from all the technical aspects that were tackled during the AIDAinnova period, the Key4hep project has also served as an extraordinary community building tool. The willingness to accept contributions from all communities enabled excellent discussions, which then ensured that having experiment specific assumptions in the stack was held to a minimum. Although reaching a consensus among all involved parties usually takes longer than isolated developments, the benefits gained from the shared approach usually far outweigh the investments. Consequently, Key4hep has developed from an idea, in 2020, into an actual product and recognized name in the community and some of the software developed in its context, e.g. EDM4hep, themselves starting to become new standards. Within WP12, Key4hep has served as an integration and test bed for developments from other tasks, e.g., for running generative machine learning models inside DD4hep in combination with full simulation.

Despite all these successes, work is far from done for Key4hep. First and foremost, a first stable version for EDM4hep needs to be released. This is currently in its final stages and should be achieved by the end of the project. Afterwards, substantial efforts need to go into the development of comprehensive tracking functionality, based on ACTS for the Gaudi based Key4hep framework. Here one goal is to profit from similar developments that have already been started in the EIC and Muon Collider communities. In a similar vein, re-establishing expertise for tuning and newly developing particle flow reconstruction within the PandoraPFA framework needs to be prioritized as several detector concepts are starting to be mature enough to be limited by a lack of proper particle flow reconstruction.

On a longer-term basis, R&D on heterogeneous resource use should also see increased investment and effort. Two key areas have been identified where Key4hep could profit directly: resource aware scheduling of algorithms to optimally exploit the available resources, and a data representation and layer that allows for an efficient switch between them. The latter aspect most likely will also need new developments on the PODIO toolkit, e.g., the possibility to change the in-memory representation of the data objects of generated EDMs, or the possibility to easily move data from main memory to GPU memory.

5. REFERENCES

[1] Sailer, A., Ganis, G., Mato, P., Petric, M., Stewart, G.A. (2020) Towards a Turnkey Software Stack for HEP Experiments. In: *EPJ Web of Conferences Proceedings of the 24th International Conference on Computing in High Energy and Nuclear Physics (CHEP),* 4-8 Nov 2019, Adelaide, AU. EPJ Web Conf. 245 (2020) 10002, *https://doi.org/10.1051/epjconf/202024510002*

[2] Sasikumar, S., et al. (2025) Five years of Key4hep – Towards production readiness and beyond. In: *Proceedings of Science (PoS) 42nd International Conference on High Energy Physics (ICHEP)*, 18-24 July 2024, Prague, CZ. PoS ICHEP2024 (2025) 1029, *https://doi.org/10.22323/1.476.1029*

[3] Barrand, G., et al. (2001) GAUDI – A software architecture and framework for building HEP data proecssing. In: *Comput. Phys. Commun.* 140 (2001) 45-55, *https://doi.org/10.1016/S0010-4655(01)00254-5*

[4] Frank, M., Gaede, F., Grefe, C., Mato, P. (2014) DD4hep: A Detector Description Toolkit for High Energy Physics Experiments. In: *Journal of Physics Conference Series: 20th International Conference on Computing in High-Energy and Nuclear Physics (CHEP),* 14-18 October 2013, Amsterdam, NL. J.Phys.Conf.Ser. 513 (2014) 022010, *https://doi.org/10.1088/1742-6596/513/2/022010*

[5] T. Madlener (2023) *LC reconstruction prototype in Key4HEP*. Zenodo. https://doi.org/10.5281/zenodo.7549856

[6] Gamblin, T., LeGendre, M., Collette, M. R., Lee, G. L., Moody, A., de Supinski, B. R., and Futral, S. (2015). The Spack Package Manager: Bringing Order to HPC Software Chaos. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15).* Association for Computing Machinery, New York, NY, USA, Article 40, 1–12. *https://doi.org/10.1145/2807591.2807623*

[7] Gaede, F., Behnke, T., Graf, N., Johnson, T. (2003) LCIO: A Persistency framework for linear collider simulation studies. In: *Proceedings of the 13th International Conference on Computing in High-Energy and Nuclear Physics (CHEP),* 24-28 March 2003, La Jolla, US. eConf C0303241 (2003) TUKT001. *https://doi.org/10.48550/arXiv.physics/0306114*

[8] Volkl, V., et. al. (2022) *key4hep/k4FWCore*. Zenodo. https://doi.org/10.5281/zenodo.4564604

[9] Gaede, F. (2006) Marlin and LCCD: Software for the ILC. In: *Proceedings of the X International Workshop on Advanced Computing and Analysis Techniques in Physics (ACAT),* 22-27 May 2005, Zeuthen, DE. Nucl.Instrum.Meth.A 559 (2006) 177-180, *https://doi.org/10.1016/j.nima.2005.11.138*

[10] *Key4hep github organization* [online]. Available from *https://github.com/key4hep* [Accessed 15 January 2025]

[11] Madlener, T., et al. (2024) *key4hep/EDM4hep*. Zenodo. *https://doi.org/10.5281/zenodo.4785062*

[12] Madlener, T., et. al. (2024) *key4hep/k4EDM4hep2LcioConv*. Zenodo. https://doi.org/10.5281/zenodo.13837370

[13] Fernandez Declara, P., et. al. (2024) *key4hep/k4MarlinWrapper*. Zenodo. https://doi.org/10.5281/zenodo.4719244

[14] Carceller, J.M., et al. (2024) Towards podio v1.0 – A first stable release of the EDM toolkit. In: *EPJ Web of Conferences Proceedings of the 26th International Conference on Computing in High Energy and Nuclear Physics (CHEP),* 8-12 May 2023, Norfolk, US. EPJ Web Conf. 295 (2024) 06018, *https://doi.org/10.1051/epjconf/202429506018*

[15] *PODIO documentation* [online]. Available from: *https://key4hep.web.cern.ch/podio/* [Accessed 14 January 2025].

[16] *EDM4eic github repository* [online]. Available from *https://github.com/eic/EDM4eic* [Accessed 15 January 2025]

[17] Blomer, J., Canal, P., Naumann, A., Piparo, D. (2020) Evolution of the ROOT Tree I/O. In: *EPJ Web of Conferences Proceedings of the 24th International Conference on Computing in High Energy and Nuclear Physics (CHEP),* 4-8 Nov 2019, Adelaide, AU. EPJ Web Conf. 245 (2020) 02030, *https://doi.org/10.1051/epjconf/202024502030*

[18] Piparo, D., et al. (2019) RDataFrame: Easy Parallel ROOT Analysis at 100 Threads. In: *EPJ Web of Conferences Proceedings of the 26th International Conference on Computing in High Energy and Nuclear Physics (CHEP),* 9-13 July 2023, Sofia, BG. EPJ Web Conf. 214 (2019) 06029. *https://doi.org/10.1051/epjconf/201921406029*

[19] Buckley, A., et. al (2021) The HepMC3 event record library for Monte Carlo event generators. In: *Computer Physics Communications* 260 (2021) 107310. *https://doi.org/10.1016/j.cpc.2020.107310*

[20] Clemencic, M., Hegner, B., Legget, C. (2017) Gaudi evolution for future challenges. In: *Journal of Physics Conference Series: 22nd International Conference on Computing in High-Energy and Nuclear Physics (CHEP)*. 14-16 October 2016, San Francisco, US. J.Phys.Conf.Ser. 898 (2017) 4, 042044, *https://doi.org/10.1088/1742-6596/898/4/042044*

[21] Aleksa, M., Bedeschi, F., Sefkow, F. (2021) Calorimetry at FCC-ee. In: *Eur.Phys.J.Plus* 136 (2021) 10, 1066, https://doi.org/10.1140/epjp/s13360-021-02034-2

[22] Sasikumar, S., Sailer A., Francois, B. (2024) First look at particle flow in LAr calorimeter using Pandora in Key4hep framework. In: *Proceedings of Science (PoS) 42nd International Conference on High Energy Physics (ICHEP),* 18-24 July 2024, Prague, CZ. PoS ICHEP2024 (2025) 1142, *https://doi.org/10.22323/1.476.1142*

[23] *key4hep-spack repository* [online]. Available from *https://github.com/key4hep/key4hep-spack* [Accessed 15 January 2025].

ANNEX: GLOSSARY

Acronym	Definition
EDM	Event Data Model
FCC	Future Circular Collider
ILC	International Linear Collider
CLIC	Compact LInear Collider
EIC	Electron-Ion Collider
CEPC	Circular Electron-Positron Collider
HEP	High Energy Physics
LCIO	Linear Collider Input/Output
LHC	Large Hadron Collider
CMS	Compact Muon Solenoid
LC	Linear Collider
HL-LHC	High-Luminosity LHC
LHC	Large Hadron Collider
CI	Continuous Integration
CD	Continuous Deployment