DDG4 A Simulation Framework based on the DD4hep Detector Description Toolkit

DDG4
A Simulation Framework based on the DD4hep Detector Description Toolkit

M. Frank[1], F. Gaede[2], N. Nikiforou[1], M. Petric[1], A. Sailer[1]
[1] CERN, 1211 Geneva 23, Switzerland
[2] Desy, 22607 Hamburg, Germany

E-mail: `Markus.Frank@cern.ch`

Abstract.
The detector description is an essential component that has to be used to analyse and simulate data resulting from particle collisions in high energy physics experiments. Based on the DD4hep detector description toolkit a flexible and data driven simulation framework was designed using the Geant4 tool-kit. We present this framework and describe the guiding requirements and the architectural design, which was strongly driven by ease of use. The goal was, given an existing detector description, to simulate the detector response to particle collisions in high energy physics experiments with minimal effort, but not impose restrictions to support enhanced or improved behaviour.
Starting from the ROOT based geometry implementation used by DD4hep an automatic conversion mechanism to Geant4 was developed. The physics response and the mechanism to input particle data from generators was highly formalized and can be instantiated on demand using known factory patterns. A palette of components to model the detector response is provided by default, but improved or more sophisticated components may easily be added using the factory pattern. Only the final configuration of the instantiated components has to be provided by end-users using either C++ or python scripting or an XML based description.

1. Introduction
The development of a coherent set of software tools for the description of high energy physics detectors from a single source of information has been on the agenda of many experiments for decades. The application simulating the detector response from particle collisions are of major importance both for the studies during the planning and construction phase as well as during the running of the detector. The simulation applications are expected to mimic the energy deposits of particles in the detector which then are digitized to emulate the response of the readout electronics. This simulated data is then fed, in the same way as data collected from the electronics in a real experiment, into the reconstruction and further on into the analysis applications. The generic detector description toolkit DD4hep [1] provides an appropriate, consistent and generic detector view to support simulation, reconstruction and analysis applications for high energy physics experiments. The detector simulation application DDG4 which shall be presented in the following is based on the DD4hep detector description and uses the Geant4 toolkit [2] as a simulation engine. The design is strongly driven by ease of use; developers of detector descriptions and applications using them should have to provide only minimal information and minimal specific code to achieve the desired result.

The organization of this paper is the following. We will briefly recapitulate the general structure of the DD4hep geometry description toolkit. Then the the guiding requirements and the architectural design is discussed. From the client's view DDG4 should be a minimalistic toolkit to perform detector simulation seamlessly integrated in Geant4, the detector simulation software used in high energy physics. Finally we will briefly discuss a few examples realized with this design and formulate some conclusions.

2. The DD4hep Detector Description Toolkit

The design of the DD4hep toolkit [1] is shaped by the experience of detector description systems, which were implemented for the LHC experiments, in particular the LHCb experiment [3, 4], as well as the lessons learnt from other implementations of geometry description tools developed for the Linear Collider community [5, 6]. DD4hep aims to widely reuse existing software components, in particular the ROOT geometry package [7], part of the ROOT project [8], a tool for building, browsing, navigating and visualizing detector geometries. The code is designed to optimize particle transport through complex structures and works standalone with respect to any Monte Carlo simulation engine. The second component is the Geant4 simulation toolkit [2], which is used to simulate the detector response from particle collisions in complex designs.

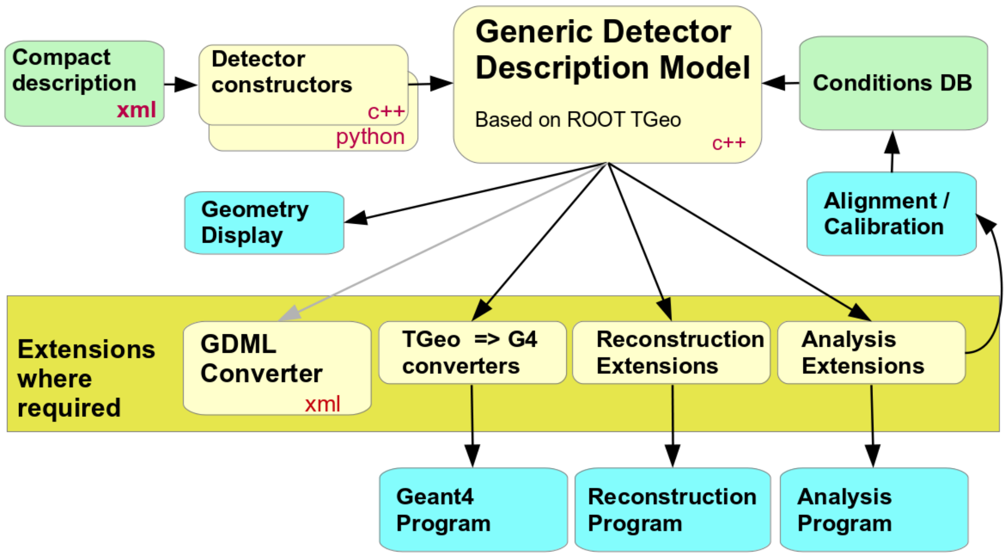

Figure 1. The components of the DD4hep detector geometry toolkit.

DD4hep implements a mechanism to create from a compact detector description in XML format with the help of specialized code fragments, so called *Detector constructors* [1] a model of a detector in memory as shown in Figure 1. The code fragments instantiate a model of the detector defined by a set of C++ classes. This in-memory model is the base to create any other representation such as the Geant4 geometry, a GDML file or others. In the following sections the focus will be put on the automatic conversion leading with supplementary though minimal information to a valid application simulating detector collisions using the Geant4 toolkit [2].

3. Concepts and Toolkit Design

3.1. Toolkit Requirements

The detector simulation application DDG4 should provide the detector response of any experimental setup defined in DD4hep similar to the real experiment. The motivation of the chosen approach is to minimize effort. However, the effort of users to add new or to enhance existing functionality must not be limited:

- **Minimalistic Approach**: The effort of users to retrieve simulated detector response should be minimal. Any existing in-memory geometry model together with a selection of plugin-components to compute the detector response and the definition of the contributing physics processes from an existing palette must be sufficient. If desired, users may define specialized components, which are automatically integrated into the existing toolkit.
- **Ease of Use** influenced the design and the implementation. A minimalistic approach means not burdening users with code development. The usage of provided components must be easy.
- **Flexible Extensions**: The presence of an appropriate mechanism to build complete execution blocks from existing simple components. The configuration of the building blocks must be easy. Component configuration must be possible both during the configuration phase and, once Geant4 is started, from the Geant4 provided interactivity mechanism.

3.2. User Entry Points defined by Geant4
It was found that the requirements enumerated in section 3.1 can easily be implemented by modeling all Geant4-provided user entry points as user defined sequences of more atomic actions each handling one aspect. These sequences have customized interfaces depending on the calling signatures. Geant4 provides the following entry points to allow for user interaction [2]:

- *The Generation Action* is called before the the event processing starts. The Geant4 kernel expects here all primary particles and primary vertices to be declared.
- *The Event Action* is called at the begin or the end of the event processing.
- *The Tracking Action* is called at the begin and the end of the tracking of one track object through the detector.
- *The Stepping Action* is called at the beginning of each step of the simulation process.
- *The Sensitive Response*: Callback to compute the energy deposit of a given step in a sensitive volume of the detector.
- *The Physics List*: This object only needs to be instantiated and defines the processes used for interaction with material during the overall simulation process.

3.3. Design Choices
A design of a simulation application fulfilling the requirements of section 3.1 must respect a number of criteria. These choices, which strongly influence the implementation are as follows:

- The natural sequencing of the simulation application as it is foreseen by Geant4 should remain unchanged. Consequently only Geant4 provided mechanisms for the interaction between Geant4 and user code should be used.
- In DD4hep the in-memory geometry representation is the blue-print for any other representation needed by user applications. The translation of the geometry to the Geant4 representation should be automatic and generic. Users should not deal with it.
- Flexible component instantiation in C++ is typically based on a factory mechanism. After being instantiated the components need to be configured using the same interface: each component offers a set of properties to the user, who then may alter the default values.
- The user entries listed in section 3.2 must offer a flexible interface. Building complex actions by chaining simple atomic actions must be simple. Hence, all user entries are modeled as programmable action sequences with callback signatures suitable for the action in question. Event actions for example receive a reference to the G4Event structure and a tracking actions receives a reference to the G4Track structure. When called, these action sequences internally call a sequence of function callbacks which were registered earlier by the user

and a sequence of objects for the callback type in question, which were instantiated and assigned by the client during the setup phase.

- Calls to user defined member functions are needed by specialized objects, which collect information during various stages of the processing of one event such as e.g. the propagation of Monte Carlo truth information. The difference to the inheritance approach is marginal: whereas in the first event the object to be called must provide a member function with the required signature, in the latter the behavior is enforced by inheritance from a base class common to one of the different action types listed above.

4. Toolkit Implementation

4.1. Geometry Translation to Geant4

Geant4 requires an internal, Geant4 specific, representation of the detector geometry, which is different from the geometry implementation of DD4hep based on the ROOT geometry package. A generic mechanism implementing the Geant4 provided user interface *G4VUserDetectorConstruction* [2] automatically converts all geometry entities like shapes, materials, volumes and volume placements to Geant4. The mechanism is identical for all applications. The implemented mechanism scans the geometry tree starting with the top level element and recursively collects all entities. The translation of objects representing this geometry hierarchy happens in the reverse order i.e. first the volumes and placements representing the most detailed structures are translated before larger substructures are created. The recursion ends with the translation of the top level volume. Special attention was put on entities where the implementation concepts differ between the Geant4 geometry and the ROOT based geometry such as shape assemblies. Figure 2 illustrates the geometry scan of a hypothetical Time Projection Chamber (TPC) detector.

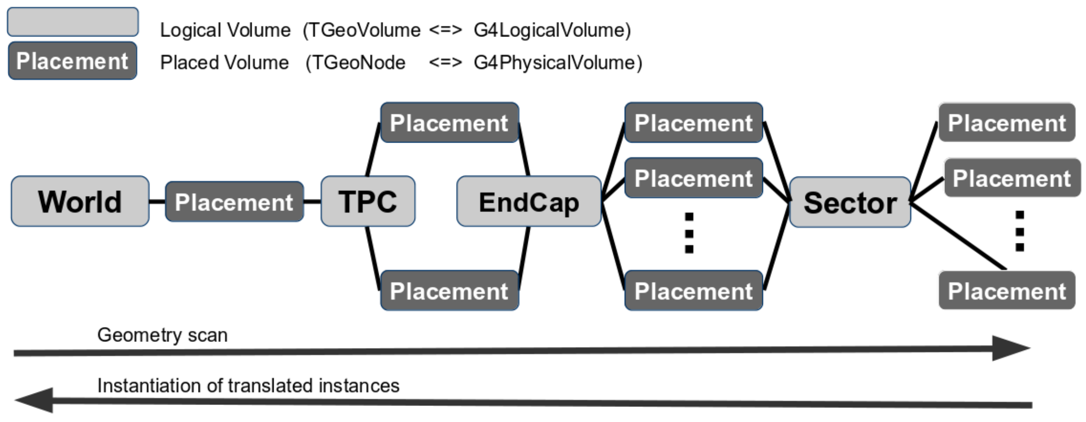
Logical Volume (TGeoVolume <=> G4LogicalVolume)
Placement
Placed Volume (TGeoNode <=> G4PhysicalVolume)
World
Placement
TPC
Placement
Placement
EndCap
Placement
Placement
Placement
Sector
Placement
Placement
Placement
Geometry scan
Instantiation of translated instances

Figure 2. The geometry scan of a hypothetical TPC. The scan starts from the top level volume "World" and recursively descends to all placed children within each volume. The instantiation of the translated objects is reverse: small structures get placed into bigger structures.

4.2. Common Base Class for all Components

The implementation of plugin components and their use, is greatly simplified if all collaborating components use a common base class. The base class should be simplistic, but still provide access to all objects needed to perform its work. It should implement provisions to deal with all common functionalities such as external configuration of the component properties, Geant4 interactivity and access to data provided either by Geant4 or the detector description toolkit. All components inherit from the *Geant4Action*. The diagram in Figure 3 illustrates the connections between the base class and the embedding framework:

- The reference to the *Geant4Context* allows the user to access Geant4 interna such as the run manager, the geometry setup and the DD4hep detector description.
- The *PropertyManager* allows the user to externalize object properties. These may be atomic primitives or complex structures[1].
- The reference to the *Geant4UIMessenger* allows to export component properties and component functions to the Geant4 interactive prompt.
- To facilitate object instantiations, all *Geant4Action* derivatives must conform to common constructor patterns. Sensitive actions, which compute the detector response of active volumes have a constructor of the form:
  ```
  Geant4Action* (*plugin)(Geant4Context* context, const string& name,
                                DetElement sd, LCDD& lcdd);
  ```
  All other actions must provide the simplified constructor of the form:
  ```
  Geant4Action* (*plugin)(Geant4Context* context, const string& name)
  ```
  All actions have access to the Geant4 and geometry internals. Sensitive elements for convenience reasons also have access to the attached detector and its sub-components like the readout structure and the segmentation of the sensitive areas [1].

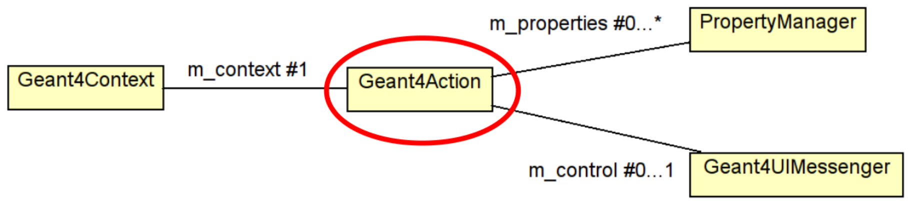

Figure 3. The basic relationships of the base class *Geant4Action* common to all components.

Using the *Geant4Action*, specific sequence actions can be constructed as illustrated in Figure 4 for an example of the event generation action:

- The sequence starts with an initialization step. Event related properties are initialized and the event context available to all actions is prepared.
- Then the first input data source is accessed: Data of event type "Signal" is read from persistent medium (or alternatively generated). The resulting primary vertices of this sub-event and all resulting tracks are then boosted and the primary vertices are smeared in subsequent modules according to certain beam profiles and beam conditions.
- In a third step a second data source is accessed: Data of type "Background" is read. Again the interaction is boosted and the primary vertices smeared. Note that though the modules used to boost and smear the primaries are identical for both input streams, the instances are different, the property values and thus the component behavior may differ.
- So far both interactions are kept separate. They now need to be merged to one single record combining the primary vertices and outgoing primary tracks of both interactions.
- Finally the record containing all primary vertices and all primary tracks must be given to Geant4. Now the detector simulation can start.

4.3. Monte Carlo Truth Handling

A basic functionality every simulation application must solve is the correct assignment of generated energy deposits, so-called "hits" to the track object which created the deposit. In practice there are multiple approaches to achieve this functionality.

[1] Any data structure, which can be described by the mechanisms provided by boost::spirit [9] is understood.

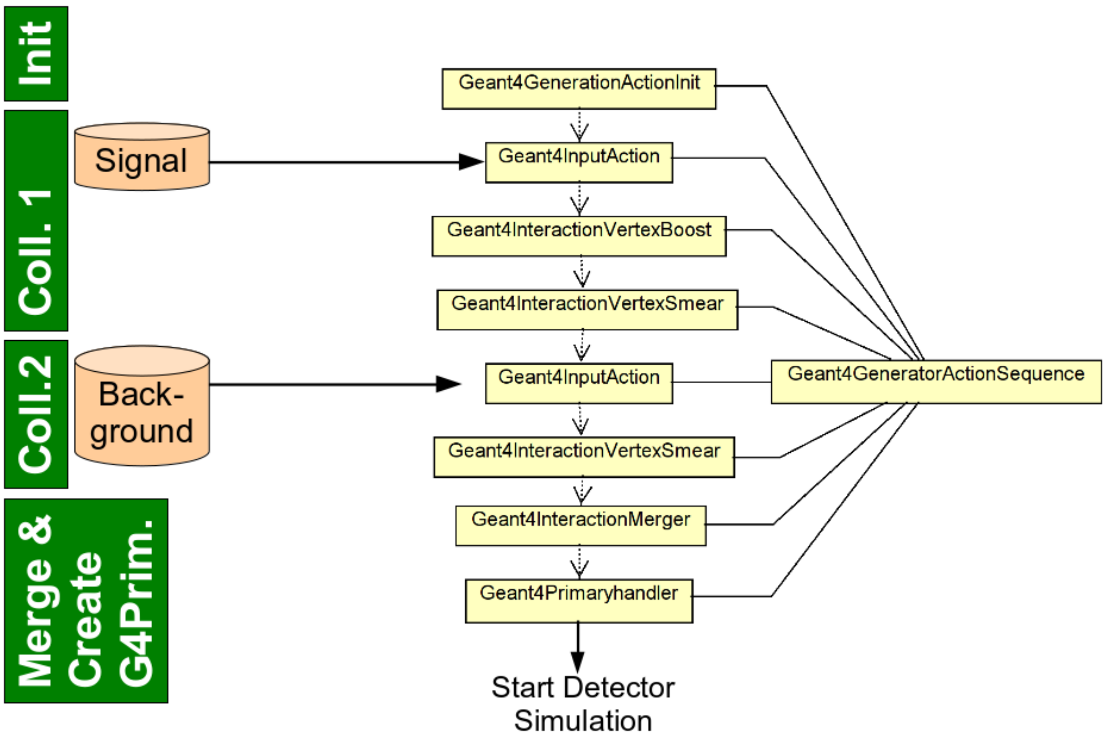
Init
Coll. 1
Coll.2
Merge & Create G4Prim.
Signal
Back-ground
Geant4GenerationActionInit
Geant4InputAction
Geant4InteractionVertexBoost
Geant4InteractionVertexSmear
Geant4InputAction
Geant4InteractionVertexSmear
Geant4InteractionMerger
Geant4Primaryhandler
Geant4GeneratorActionSequence
Start Detector Simulation

Figure 4. An example of a more complex generation action sequence used to merge signal and background events for a simultaneous simulation. For simplicity the base classes are not shown.

- *Full history:* The full history track is kept containing all energy deposits and the full record of creating tracks. Except for detailed studies this mode typically is impractical, because the huge number of secondary tracks causes the output record to become too large.
- *Reduced history:* Only the most relevant secondary tracks are kept. Energy deposits created by low energetic tracks are reassigned to the track parents. Typically energy deposits in calorimeters are handled differently from deposits in tracking detectors.

DDG4 offers a generic component to propagate the Monte Carlo truth information between hits and tracks for both modes. The truth handler uses an internal track object model, which at the output stage may be easily converted to a user defined track model. The results during the simulation of the complete event may later be used when the actual energy deposits and tracks are written to the output record.

4.4. The DDG4 Component Palette

In the early stages of an experiment, simulation activities are mostly used to de- and refine the setup of the experiment. For these users a palette of ready-to-use components is provided. These include:

- Standard input modules capable of reading primary interactions created by generators in the format lcio[10], StdHEP[10] or HepMC[11].
- Standard modules to manipulate primary interactions like smearing or boosting the primary vertex (see Figure 4), merging several interactions etc.
- Output modules to write simulated data in the lcio data format or to ROOT using the internal data structures.
- Generic components collecting the detector response of tracking detectors or calorimeters.

This palette allows the user to perform all necessary simulation activities of detector designs such as ILD [5] and SiD [6]. It is understood that the palette is not complete and all users are

welcome to contribute to the palette for example if a new detector response module is being developed or a new input/output data format should be supported. The default palette can only contain modules which are sufficiently generic to be applied to a variety of designs. Specific and detailed implementations are still supported by the DDG4 plugin mechanism. The code however should then be part of the experiment's framework. The same approach was earlier adopted for the DD4hep detector palette.
Though such generic modules are very convenient, a price is to be paid: The Geant4 track objects are stack based, the objects have a limited lifetime. Track objects are only guaranteed to exist between the beginning and the end of the tracking action of one track. To provide track data to populate the output record an internal track model was introduced to save the basic track quantities. Because no structure can a posteriori satisfy all users, client defined extension data may be added to each track structure.

4.5. The Handling of the Physics List
Similar to the instantiation of actions, the Geant4 physics setup is encapsulated in a set of factories. Hence, instantiations to particle constructors, physics processes, physics constructors or predefined physics lists as provided by Geant4 are implemented using the factory bindings. The factory bindings are provided by DDG4. In section 4.6 a simple example of the configuration of a physics list is shown.

4.6. Configuration of the Simulation Application
Special emphasis was put on the configuration of the application. Technically all methods must initiate the following functionality:

- convert the in-memory geometry to Geant4,
- configure action sequences,
- start the simulation activity from the Geant4 run manager.

To configure the application DDG4 offers three choices:

- *XML:* The simulation application may be bootstrapped from data in an XML file.
- *C++ script:* Since the toolkit is written in this language, this is the natural way to configure the application, but any change to the configuration requires to recompile and to relink.
- *Python script:* Python and the ROOT supported interface PyROOT is the most flexible and still robust way to bootstrap a DDG4 simulation application. The python interface is for free provided the ROOT dictionaries are generated. The dictionary mechanism exports the same classes to python which are present in C++ leading to very similar configuration code. This approach does not require recompilation after a change.

As an illustration it is best to show a python code snippet which loads the geometry, sets up the Geant4 user interface and then adds a particle gun to the generation action sequence:

```
def run():
  ######## Bootstrap and load the geometry
  kernel = DDG4.Kernel()
  kernel.loadGeometry("file:../examples/CLICSiD/compact/compact.xml")
  ######## Setup the Geant4 user interface
  ui = DDG4.Action(kernel,"Geant4UIManager/UI")
  ui.HaveVIS = True
  ui.HaveUI  = True
  ui.SessionType = ’csh’
  kernel.registerGlobalAction(ui)
  ######## Create particle gun object and set properties:
```

```
gun = GeneratorAction(self.kernel,"Geant4ParticleGun/Gun")
gun.energy       = 25*GeV
gun.particle     = "e-"
gun.position     = ( 0*mm, 0*mm, 1*mm )
self.kernel.generatorAction().add(gun)
######## Instantiate physics list
phys = kernel.physicsList()
phys.extends = "QGSP_BERT"
phys.addParticleConstructor('G4BosonConstructor')
phys.addParticleProcess('e[+-]','G4eMultipleScattering',-1,1,1)
phys.addPhysicsConstructor('G4OpticalPhysics')
.......
```

5. Conclusions

Based on the detector description toolkit DD4hep a component oriented simulation application was designed which allows users to simulate the physics response of detector setups with minimal effort. Though the toolkit is equipped with numerous predefined components suited for many activities, users may at any time add more sophisticated implementations. The early adoption of parts of the linear collider community [5, 12, 13] and the Future Circular Colider Study group (FCC) – though currently only for evaluation purposes – has been an invaluable asset to verify the usefulness of the implemented design. These early end-users help to continuously verify that the implemented functionality fulfills the needs of experiments. The hope is that these early adopters also contribute to complete the existing component palette for the benefit of the high energy physics community.

References

[1] M.Frank et al., DD4hep: A Detector Description Toolkit for high energy physics Experiments, International Conference on Computing in High Energy and Nuclear Physics (CHEP 2013), Amsterdam, NL, 2013, proceedings.
[2] S. Agostinelli et al., "Geant4 - A Simulation Toolkit", Nuclear Instruments and Methods **A** 506 (2003) 250-303.
[3] LHCb Collaboration, LHCb, the Large Hadron Collider beauty experiment, reoptimised detector design and performance, CERN/LHCC 2003-030
[4] S. Ponce et al., Detector Description Framework in LHCb, International Conference on Computing in High Energy and Nuclear Physics (CHEP 2003), La Jolla, CA, 2003, proceedings.
[5] The ILD Concept Group, The International Large Detector: Letter of Intent, ISBN 978-3-935702-42-3, 2009.
[6] H. Aihara, P. Burrows, M. Oreglia (Editors), SiD Letter of Intent, arXiv:0911.0006, 2009.
[7] R.Brun, A.Gheata, M.Gheata, The ROOT geometry package, Nuclear Instruments and Methods **A** 502 (2003) 676-680.
[8] R.Brun et al., Root - An object oriented data analysis framework, Nuclear Instruments and Methods **A** 389 (1997) 81-86.
[9] Boost-Spirit, C++ libraries for parsing and output generation. http://www.boost-spirit.com
[10] F.Gaede et al., LCIO - A persistency framework for linear collider simulation studies, International Conference on Computing in High Energy and Nuclear Physics (CHEP 2003), La Jolla, CA, 2003, proceedings.
[11] M.Dobbs et al., HepMC2 a C++ Event Record for Monte Carlo Generators, http://lcgapp.cern.ch/project/simu/HepMC
[12] CLICdp: CLIC detector and physics study, http://clicdp.web.cern.ch.
[13] A.Sailer at al., Integration of DD4hep in the Linear Collider Software Framework, Poster 290, International Conference on Computing in High Energy and Nuclear Physics (CHEP 2015), Okinawa, Japan, 2015, proceedings.