Journal of Physics: Conference Series **331** (2011) 032024 doi:10.1088/1742-6596/331/3/032024

The Software Framework of the Belle II Experiment

Andreas Moll

Max-Planck-Institut für Physik, Föhringer Ring 6, 80805 München, Germany
Exzellenzcluster Universe, Boltzmannstr. 2, 85748 Garching, Germany

E-mail: molland@mpp.mpg.de

Abstract. The future of CP-Violation experiments is to begin in 2014 with the launch of the SuperKEKB collider in Tsukuba, Japan. As a part of this process the BELLE experiment will undergo an upgrade, giving rise to the BELLE II experiment. The BELLE II detector will include improvements and redesigns of various subdetectors, as well as the addition of an entire new subdetector for precise vertexing. In order to reflect these changes in the existing BELLE software framework, major modifications of nearly all parts of the software would have been necessary. As a result the decision was made to completely rewrite the software framework. In this article the main concepts of the new framework and the applied technologies are presented.

1. Introduction

The BELLE II experiment is the upgrade of the highly successful BELLE experiment located at the KEK research center in Tsukuba, Japan. The BELLE experiment, running until June 2010, was located at the interaction point of the KEKB collider [1], a ring accelerator for electrons and positrons at the asymmetric energies of 8 GeV (e^-) and 3.5 Gev (e^+). The main physics goal of BELLE, and in turn BELLE II, is the measurement of CP-Violation in the B-meson system. To measure CP-Violation to a high precision, a large number of B-meson pairs is necessary. This requires a major increase of the instantaneous luminosity, the driving force for the design and operation of the new SuperKEKB [2] collider and the BELLE II experiment. The final peak luminosity will be $8 \cdot 10^{35} cm^{-2} s^{-1}$. The upgrade will not only increase the rate of the data that has to be written out, but will also increase the expected background for all subdetectors of BELLE II. The current BELLE detector is not able to handle the new conditions that will be found at SuperKEKB, hence an upgrade of the BELLE detector is necessary. The upgraded detector, BELLE II, will see improvements and redesigns of various subdetectors, as well as the addition of an entire new subdetector. This new subdetector, the Pixel Vertex Detector (PXD), is developed by the DEPFET [3] collaboration to precisely measure the decay vertices of B-mesons, allowing the measurement of time-dependent CP-Violation.

2. The impact on the software framework

The software framework used successfully for over 10 years at BELLE is called **BASF** (**B**elle **A**nalysi**S** **F**ramework). One of its strengths lies in its ability to be used not only for offline purposes (such as reconstruction and analysis), but also for online tasks (such as data aquisition, data quality monitoring and the high level trigger). In order to reflect the changes made in the BELLE II detector upgrade in the existing software framework, major modifications of nearly all parts of the software would have been necessary. The vast amount of required modifications and

the lack of object oriented persistency (BASF uses Fortran based Panther tables to store data) led to the decision to completely rewrite the software framework. The new software framework, called **basf2**, incorporates concepts of other HEP experiments, such as ILC, LHCb, CDF and ALICE, into its design, but primarily follows concepts proven in BASF. In addition, the reuse of well-written algorithms of BASF is encouraged in order to maintain the excellent work done in the past 10 years at BELLE. To facilitate and accelerate the development, established third-party libraries such as ROOT, boost, CLHEP and libxml are used throughout the entire software.

3. The basic architecture of basf2

Like any other HEP software framework, basf2 was developed to process large data sets in a flexible and configurable way. This flexibility is achieved by making the methods applied to the data user-adjustable. Each data processing method is encapsulated within a **module**, with multiple modules arranged in a strict linear order inside a container called a **path**. The user can create an arbitrary number of modules and paths with one or more paths linked together to form a data processing chain.

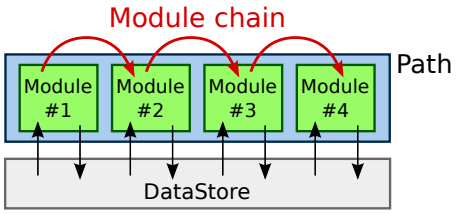

Figure 1. A simple data processing chain. One path containing 4 modules that exchange data with each other using the common DataStore.

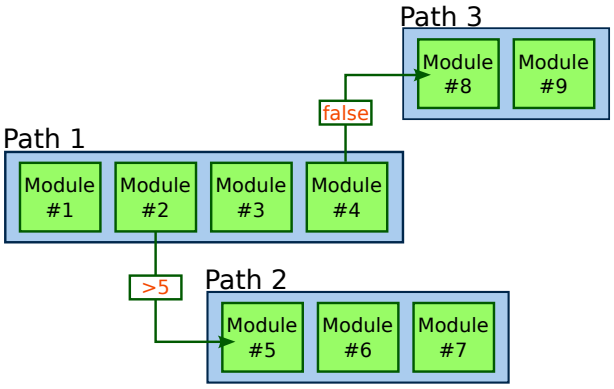

Figure 2. Multi path scheme in basf2: Three paths connected by conditions. Both possible condition types are shown: An integer value and a simple boolean value.

Figure 1 shows a simple example of a path consisting of 4 modules. The tasks of the modules vary from simple ones, such as reading data from a file, to complex tasks, such as full detector simulation. Unlike most other HEP software frameworks, basf2 employs the module concept for all tasks and data processing methods within the framework. Even the reading of data from and writing to disk is done through modules.

During the processing of data the framework executes the modules of a path, starting with the first one and proceeding to the next. The modules are executed one at a time, exactly in the order in which they were placed into the path. The data to be processed by the modules is stored in a common storage, the **DataStore** (see Figure 1). As previously mentioned more than one path can be created in basf2. Multiple paths can then be connected to each other using **conditions**. Each module can return an integer or a boolean value. Depending on these return values and the user defined conditions, the process flow can then switch from one path to another (see Figure 2).

3.1. Libraries and modules

The main idea behind the module concept is to have the whole framework functionality distributed over small, reusable and independent modules. The latter feature leads to the

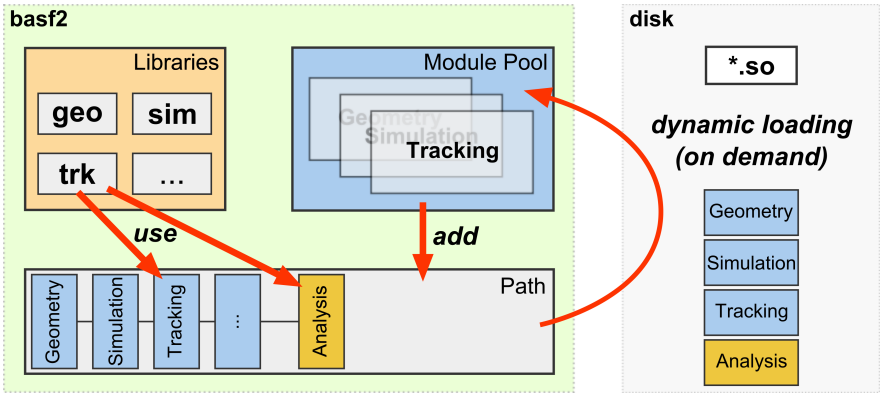

Figure 3. Modules and libraries are separated in basf2. Modules are loaded on-demand by a user request.

requirement that modules are not allowed to directly include any other modules. While often developers tend to implement methods and algorithms within one single module, hiding those algorithms from other developers, basf2 overcomes this drawback by separating modules and their functionality. The functionality is encapsulated in shared, and therefore reusable, libraries. A module then uses one or more of these libraries to carry out a specific task. This has the advantage that one algorithm can be used by more than one module. For example, the tracking algorithms can be used by the tracking module, the alignment module or a user analysis module. The final architecture of this concept is shown in figure 3.

Modules are located inside shared object files and only loaded **on-demand**. If a module is requested by the user, the associated shared object file is dynamically loaded, and the contained modules are added to a common **Module Pool**. This allows the reuse of loaded modules the next time the user requests an already existing module. To build up a data processing chain, the requested module is then added to a path.

3.2. The DataStore

Modules read data from and, after processing, write it back to the DataStore (see Figure 1). This means that a module always has access to the data which was processed by its preceding modules. The DataStore is divided into three categories: one for data which should be stored only for one single event; one for data which is stored over a complete run; and a persistent category for data which should be collected over the entire data processing. The basf2 framework deletes the data in the category of the DataStore after the event, run or data processing, respectively. The DataStore itself is based on ROOT persistency and can therefore handle any class with a ROOT dictionary attached. Saving and loading the data to or from disk is also performed using ROOT. This leads to files which are compatible with any ROOT based tool.

3.3. Parallel processing

For the real-time systems of the detector (such as the high level trigger) the data processing speed of the framework is crucial. A proven concept, providing the required computing speed, is to run the data processing in parallel. But not only the real-time systems profit from processing the data in parallel. In recent years multi-core processors have become common for desktop computers and are the standard for modern computers nowadays. In order to make use of multi-core systems for both real-time systems and desktop computers, the basf2 framework has an advanced and user-friendly parallel processing mechanism built-in. This means, to make

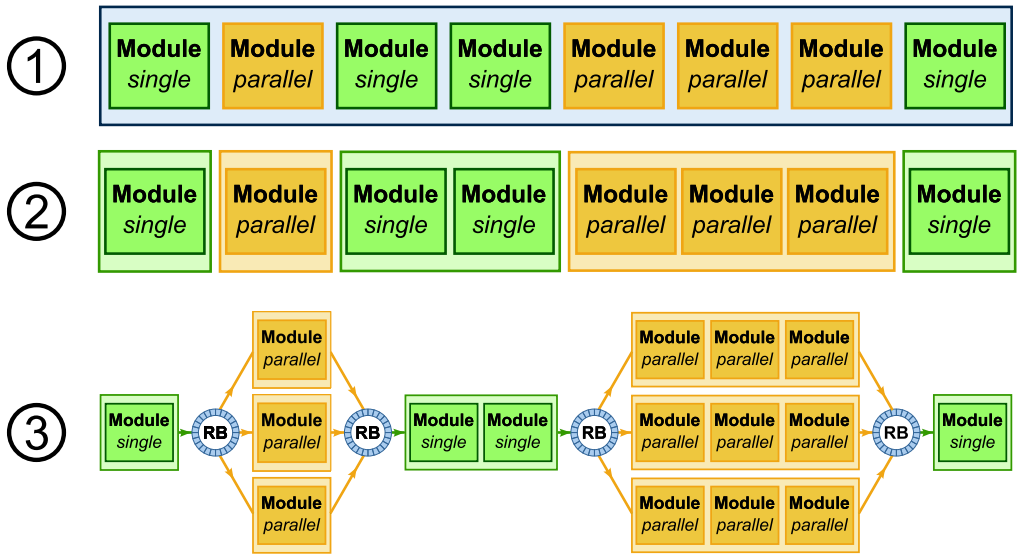

Figure 4. The parallel processing mechanism.

full use of the parallel processing capabilities of basf2, the user simply activates the parallel processing feature while still being able to keep the steering files (see section 4) unchanged. In order to provide such a user-friendly experience, basf2 uses the concept of so called **module properties**. Each module in basf2 can carry properties that give the framework additional information about the capabilities of a module. For the parallel processing mechanism a special property called **ParallelProcessingCertified** was introduced. This property flags modules that can be run in parallel processing mode safely. Only modules which comply with certain standards are allowed to carry this property. In particular, all computation-intensive modules, such as detector simulation and reconstruction modules, carry this property. If the parallel processing feature is turned on, the framework scans the module paths for continuous blocks of modules carrying this parallel processing property. Having identified the blocks of parallel processing aware modules, the framework splits the user-defined paths into a tree consisting of single processing and parallel processing paths. Figure 4 illustrates this process.

The parallel processing paths are forked and the single processing paths are connected to these by **ring buffers**. A ring buffer is responsible for either distributing the data coming from a single processing path to multiple parallel processing paths or for simply collecting the data from parallel processing paths. This concept avoids the drawbacks often found in parallel processing HEP frameworks. For example, no special parallel file reading or writing code is necessary and user-written analysis modules are executed automatically in the safer single processing mode.

4. The steering file

The basf2 framework is written in **C++** and provides a **Python** interface. This interface allows the user to communicate with the framework in a simple and intuitive way. In particular, the user writes a Python script to configure the framework and process the data. Such a configuration script is called a **steering file** and is a plain Python script. Therefore, the full set of Python functionality can be used in addition to the basf2 specific one. A typical steering file is shown in listing 1. It starts by importing the basf2 Python module in order to make the basf2 Python classes available. Then the two modules *Demo* and *Test* are registered followed by the creation of two paths (*main*, *path2*). The modules are added to the paths and both paths are connected by a condition. The final statement of the steering file begins the data processing.

Listing 1. A simple basf2 steering file

```
from basf2 import *

#Register modules
demo = register_module("Demo")
test = register_module("Test")

#Create paths
main = create_path()
path2 = create_path()

#Add modules to path
main.add_module(demo)
path2.add_module(test)

#Set condition
demo.condition(">5",path2)

#Process events
process(main)
```

5. Geometry

Nearly all tools of the offline software chain need a geometrical description of the BELLE II detector, starting from the Monte Carlo based simulation of the passage of particles through the detector, to the simulation of the response of the subdetector hardware, to the final reconstruction (e.g. tracking) algorithms. To make sure that all tools of the framework have access to the same version of the detector description, it must be stored centrally. The central repository stores all parameter values needed to fully describe the BELLE II detector. The concrete geometry used for the simulation and reconstruction algorithms is created using C++ source code from the stored parameters. Storing parameter values instead of concrete geometry in the central repository allows for a simple and generic way to handle time varying geometry. For example, the position of the sensors of the PXD have to be known to a high precision. Various effects, such as temperature variations, may change the position of these parts over time. By measuring the position of the sensors at a given time, the deformation of the geometry can later be taken into account for the reconstruction of the particle tracks.

5.1. Implementation of parameter storage and access

The central repository is realised using **XML** documents for the basic detector parameters. XML documents have the advantage of being human readable and highly extendable. Furthermore, they are widely used in both particle physics and industry, leading to the availability of high quality libraries, tools and software (both Open Source and commercial) to write, read and manage XML documents. The access to the parameters describing the BELLE II detector is handled by a user-friendly library called **Gearbox**. Using the **XPath** query language, the user sends a request for a parameter via the Gearbox library and gets the value of the parameter back. Special care was taken to handle the unit of parameters (e.g. length) consistently. The user is able to attach the unit in which a parameter is specified in the XML file. Upon request of a parameter the library automatically performs the correct unit conversion from its specified unit to the standard unit of the basf2 software framework.

5.2. Creating geometry objects from parameters

The geometry handling system explained so far only stores parameter values and, optionally, their development over time. The missing link between parameter values and geometry objects is filled by C++ code. Figure 5 shows the basic architecture of the geometry object building system.

Each subdetector has associated C++ code that is aware of the available parameters for that specific subdetector. The C++ code requests the parameter values that are needed in order

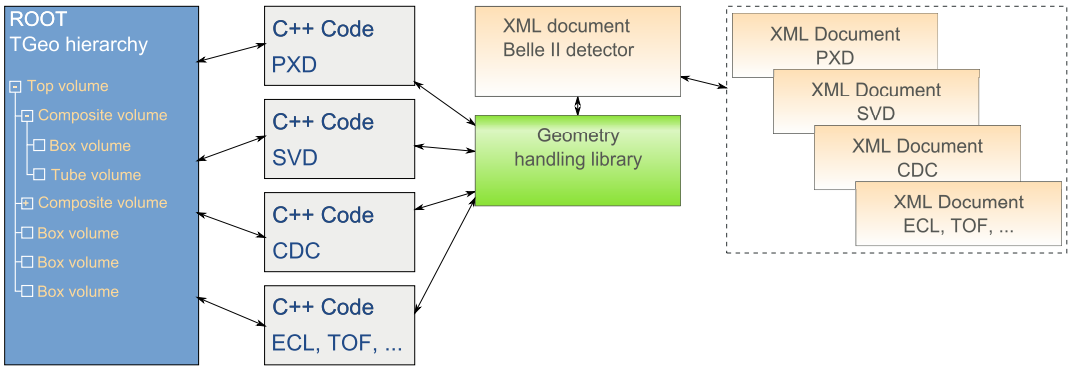

Figure 5. The basic architecture of the geometry object building system.

to build the actual geometry of the subdetector from the Gearbox library. Then, using the parameter values, each subdetector's C++ code creates the actual geometry objects and writes the created geometry into an object hierarchy. The geometry objects and their relationships within the hierarchy are implemented using ROOT **TGeo** objects, a common standard for geometry objects developed at CERN. The hierarchy is shared between all sub-detectors, leading to a single geometry object hierarchy for the whole BELLE II detector. It can then either be kept in memory and directly be used by the simulation/reconstruction software or be saved to disk for later use.

6. Simulation

The full detector simulation in basf2 is based on **Geant4** [4], the standard toolkit for the simulation of particles traversing matter in the HEP community. The TGeo geometry objects have to be made available to Geant4. The basf2 framework uses the **G4Root** library for that purpose as well as performing additional steps. Among those is the special treatment of sensitive detector parts, which have to be connected to a class that handles the feedback from the Geant4 simulation. For example, in the PXD such a class takes the Geant4 output and converts it to detector hits. These hits specify the path along which the particle traversed the detector. The results of the simulation are stored in the DataStore for further processing.

7. Summary

The BELLE II experiment is the upgrade of the highly successful BELLE experiment, located in Japan. The main physics goal of both BELLE and BELLE II, is the measurement of CP-Violation in the B-meson system and the search for rare decays. The vast amount of changes created by the upgrade from the BELLE to the BELLE II detector make it necessary to rewrite the existing software framework. The new framework is called basf2, while incorporating concepts of other HEP experiments, primarily follows concepts proven in BASF. The data processing chain consists of modules grouped into paths and offers a built-in parallel processing mechanism. The geometry is stored as a set of parameters in XML documents used by C++ code to create the actual geometry objects. For the detailed detector simulation Geant4 is used.

References

[1] A. Abashian *et al.*, Nucl. Instrum. Meth. A **479** (2002) 117.
[2] S. Hashimoto *et al.*, LoI for KEK Super B Factory, Part I: Machine, KEK-Report **2004-4** (2004).
[3] J. Kemmer, G.Lutz *et al.*, Nucl. Instrum. Meth. A **253** (1987) 356.
[4] JS. Agostinelli *et al.*, Nucl. Instrum. Meth. A **506** (2003) 250.