The Software Library of the Coming Belle II Experiment and its Simulation Package

Doris Yangsoo Kim, *Department of Physics, Soongsil University*
On behalf of the Belle II Software Group

***Abstract*–The Belle II experiment at KEK, Japan, is a next generation experiment utilizing the SuperKEKB accelerator, which is currently being upgraded. SuperKEKB is expected to deliver a 50 times larger data sample than its predecessor, KEKB. In this letter, we explain the design of the Belle II software structure in detail, with the emphasis on the simulation package.**

I. Introduction

THE heavy flavor physics is a very active field in experimental high energy physics. The heavy particles, especially B mesons have interesting decay modes. Their identification is relatively easy and the decay processes give deep-level information on the underlying physics. During the last decade, the B factories such as BaBar and Belle produced many excellent physics results: Discoveries of CP violation in various B meson decay channels, unexpected new particles such as X(3872), precision measurements of the Cabibbo–Kobayashi–Maskawa (CKM) matrix, etc.

Based on this success, a next generation B factory, SuperKEKB, is being constructed in Japan [1]. There are excellent physics opportunities with the coming B factory data sets. The upgraded measurement of the CKM matrix elements will push the limit of the Standard Model. There are possibilities of observing new phenomena such as CP violation from new physics, lepton flavor violations in tau lepton decays, charged Higgs bosons, and new hadrons.

The luminosity of the SuperKEKB accelerator is expected to be increased by 40 times compared to the previous accelerator, KEKB. The event rate and the background will be increased by a factor of at least 10. The upgraded detector, Belle II, requires a computing system which should handle a sample 50 times larger than the previous Belle data sample, to be collected between 2016 and 2022 [1]. The DAQ system should handle the data rate of 1,800 MB/sec. The Belle II offline software system is designed to meet these challenges [1]. In summary, it is a framework consisting of built-in functional objects called modules, with which various tasks involving event generation, Monte Carlo (MC) simulation, reconstruction of tracks, and physics analysis are processed. In the following sections, the structure of the Belle II framework will be explained in detail [2].

Manuscript received November 22, 2013. This work was supported in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (2013R1A1A3007772) and by the Supercomputing Center/Korea Institute of Science and Technology Information with supercomputing resources including technical support (KSC-2012-C1-19).

Author is with Department of Physics, Soongsil University, 369 Sangdo-ro Dongjak-gu, Seoul, South Korea, 156-743 (telephone: + 82 2 820 0427, e-mail: dorisykim@ssu.ac.kr).

II. The Belle II Software System

The Belle II software system is called *basf2*, which is an acronym of Belle AnlaysiS Framework 2. The design of the system is based on the ideas from the previous system, BASF, and constructed from scratch. Both the old and the new systems are written in C++. The new system uses object oriented software to store data while the old system used Panther tables, Belle's own persistency tool. In addition, useful concepts are imported from other high energy experiments: ILC, LHCb, CDF, and Alice. Third-party software libraries are incorporated in the system: ROOT, Boost, CLHEP, and many others [3]-[5]. Software developers from all around the world are participating in developing the framework and related libraries.

Basf2 uses dynamic module loading and has capabilities of parallel processing. The management of job execution is done by Python steering scripts [6]. ROOT I/O is used for data storage and reading. Geant4 is used for the full detector simulation [7]. The basf2 library code is written in C++, the standard object oriented language used in high energy physics field, with gcc 4.7 as the compilation tool. All common Linux operating systems are supported: Scientific Linux, Fedora, Ubuntu, etc. Formatting of the code is checked by Artistic Style and building of the code is done by SCons with an automatic Buildbot system [8]-[10]. Fig. 1 shows a screen shot on the daily Buildbot result.

All the software libraries and related tools are managed by Subversion, at a central system located at KEK [11]. To facilitate the code management and communication, doxygen is used to document the code and TWiki is used to further explain the software system in details [12]-[13]. Redmine is used to manage issues and problems and track their history and development [14]. The system to executed basf2 jobs on distributed computing resources is DIRAC [15]. The installation of the software on most of the computing cites is done via CVMFS [16]. Both grid and local resources are utilized for MC event generation and for analysis.

III. The Basic Structure of Basf2

The basic processing unit of basf2 is a *module*. A module can be a simple task handler such as reading data from a file or can be a complex system such as simulation or tracking packages. Depending on the demands by a user, modules are selected and plugged into a *path* as a linear chain, and each event is processed following this path. All the processing works are done in modules. When processing of an event is finished, the output record is usually written in a file. The common data storage, *DataStore,* is used to transfer data. Multiple processing paths are allowed when processing an event. The paths can be created and merged by conditions set by a user. This concept is useful for skimming data.

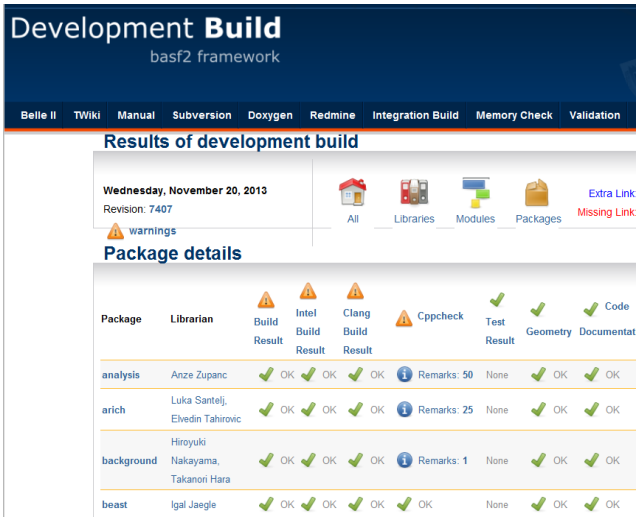

Fig. 1. The partial screen shot of development build result by Buildbot is shown in the figure.

Event-by-event parallel processing is incorporated as an option into basf2 [17]. Parallelism is achieved by forking additional Linux processes after initialization of the run. Switching between the normal mode and the parallel processing mode is transparent to users. Not all the processes could be handled in the parallel processing mode. Input and output processes on data are run in a single path. Only the modules with parallel processing property such as MC simulation can be run in parallel paths.

Code libraries are built separately from modules, in order to increase reusability. Methods and algorithms are encapsulated in libraries. So a library (i.e., algorithm) can be utilized and shared by several dynamically loaded modules.

The following script, Fig. 2, is an example of a Python steering file used to create a MC simulation data set. Event generation, simulation, and reconstruction activities are executed by the corresponding modules as shown in the script.

IV. Handling of Geometry

All the geometry parameter values are stored in the central repository using the XML format. The actual geometry for the Geant4 simulation is created from the repository parameters based on C++ algorithms. Since the geometry parameter values are directly available to users as an XML document, it is easy to maintain the parameter set and quick updates are possible. Employing the C++ algorithm increases efficiency of the overall geometry creation for simulation and reconstruction. It is straightforward to implement the detector geometry for other environments, for example, the test beam setup and BEAST II (the commissioning detector). For reconstruction, this Geant4 geometry is converted to a ROOT TGeo object by the VGM software [18]. For digitization, the relevant parameters can be imported directly from the repository.

```
# ----------     Main Steering Script Example    ----------
from basf2 import *
from simulation import add_simulation_demo
main = create_path ()

# event meta information ( One run, 100 events)
event_info_demo = register_module ('EventInfo')
event_info_demo.param ('evtNum', [100])
event_info_demo.param ('run', [1])
main.add_module (event_info_demo)

# generator selection (EvtGen, particle gun, etc.)
generator_demo = register_module ('EvtGen')
main.add_module (generator_demo)

# simulation (imported from the simulation script)
add_simulation_demo (main)

# reconstruction
...
# output
...
process (main)
```

Fig. 2. A Python script used to generate a MC data set is shown. At the beginning of the script, functions are imported from the basf2 library. A path is created as *main*. User selected modules are added into the path. After this, processing of main is invoked.

V. SIMULATION

Simulation of Monte Carlo event samples is done in three steps: Event generation, detector simulation, and digitization. Fig. 3 is a sample Python script which controls the simulation functionality of the basf2 library. This script is called by the main script, which is represented by the *add_simulation_demo* function under the # *simulation* comment line shown in Fig. 2.

```
# --------     Simulation Steering Script Example    ---------
from basf2 import *

def add_simulation_demo (path,   components=None) :
"""
This function adds the standard simulation modules to a path
"""
    # geometry parameter database
    gearbox = register_module ('Gearbox')
    path.add_module (gearbox)

    # detector geometry
    geometry = register_module ('Geometry')
    if components:
        geometry.param ('Components', components)
    path.add_module (geometry)

    # detector simulation
    Geant4_simulation = register_module ('Simulation')
    path.add_module (Geant4_simulation)

    # background mixing
    background_mixer = register_module ('MixBackground')
    path.add_module (background_mixer)

    # PXD simulation (digitization, clustering )
    If components == None or 'PXD' in components :
        pxd_digitizer = register_module ('PXDDigitizer')
        path.add_module (pxd_digitizer)
        pxd_clusterizer = register_module ('PXDClusterizer')
        path.add_module (pxd_clusterizer)

    # SVD simulation (digitization, clustering)
    ...
    # Other sub-detectors here.
    ...
```

Fig. 3. A sample Python script which uploads modules used for simulation activities. The modules are plugged to the path by the order shown in the script. This simulation script is called by the main Python script shown in Fig. 2.

A. A Simulation Example: The Secondary Particles Created by Geant4

Geant4 simulation creates a huge amount of secondary particles during event processing. Most of these particles are optical photons created by Cherenkov or scintillator detectors. For example, simulation of an Y(4s) decay into a pair of B and anti-B mesons creates 34,000 particles even when the calorimeter detector and K Long-and-muon detector of Belle II are turned off. Table 1 shows the composition of the simulated particles according to their particle identification. Fig. 4 shows the production vertex position of the particles. Since we cannot write information from all these particles in the output data file, we had to remove secondary particles from the output to reduce the size of output data files.

First, we decided not to write the information from the optical photons in the output file. Also the information from the secondary particles with kinematic energy less than 1 MeV was not written. This reduced the size of the output file tremendously.

However, when we turned on the entire Belle II detector for simulation, we still found the size of the output file was too large. Information from 2,000 particles per event was found in the output file. Therefore, we decided to keep the information from only the necessary secondary particles in the output file. The selected ones are essentially decays-in-flight and secondary particles leaving hits in the sensitive detector areas. As a result, the size of the output file size was reduced more than 90 %, which is acceptable for our resource management.

TABLE I. IDENTIFICATION OF SIMULATED PARTICLES BY GEANT4

Percent	Identification of Particles
82 %	Optical photons from Cherenkov or scintillator detectors
18 %	Electrons or positrons
0.32 %	Photons
0.20 %	Others including primary particles

B. Digitization

The simulation hits created by Geant4 are converted into measured detector signals via a digitization process. The output result of the digitization process corresponds to the real raw data. To simulate the process, detailed knowledge and information on the detector material and electronics should be exploited. Experts for each detector component handled the digitization libraries in C++. Since different versions of digitization libraries could be utilized as an option, the codes for the process were written as separate modules from Geant4, avoiding complete integration into Geant4.

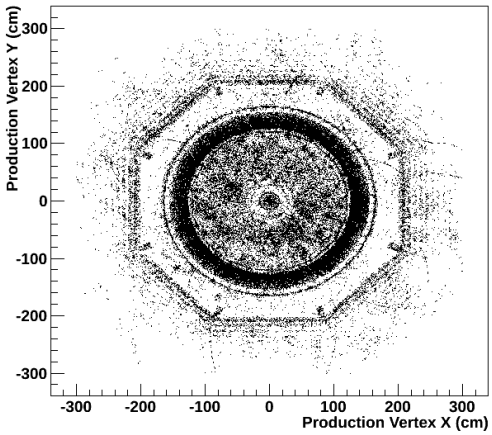

Fig. 4. Location of production vertex positions of secondary particles for 100 simulated events of Y(4s) decays into a pair of B and anti-B mesons in the xy projection. The information from optical photons and secondary particles with kinematic energy less than 1 MeV is not included in this figure, since they are too numerous. The shape of the Belle II detector is nicely reproduced by the vertex positions.

C. *Background Overlay*

The background rate is increased by 10 times for the Belle II detector with respect to the old Belle detector. A special treatment of the background simulation is devised for basf2 to handle the situation. A class of background hits is simulated separately by Geant4, than added as *SimHits* (Geant4 steps) to the already existing SimHits from the physics signal events. Beam interactions (Touscheck events) could be this kind of background. Then both the signal and background contributions are digitized at the same time by basf2. This overlay strategy is similar to what happens inside the Belle II detector during actual accelerator runs. One disadvantage is that the method does not work for measured background data.

VI. Other Tasks

In addition to the tasks explained in the previous sections, various tasks are handled by basf2. The task list includes event reconstruction, track finding and fitting, alignment, vertexing, event display, and many others.

VII. Summary

More than 500 scientists are actively involved in the Belle II project. The Belle II software system is developing successfully. A recent MC simulation campaign could simulate and reconstruct more than half a billion events without a single crash of the basf2 system. Innovative ideas are being incorporated in the system, which should handle an order of magnitude more complex run environments than the previous B factories. We are eagerly expecting the start of the physics run in 2016.

Acknowledgment

Author thanks the Belle II software group for the ideas and suggestions for this presentation.

References

[1] Edited by Z. Dolezal and S. Uno, "Belle II technical design report," *KEK Report* 2010-1, arXiv:1011.0352, Oct. 2010.
[2] A. Moll, "The software framework of the Belle II experiment," *International Conference on Computing in High Energy and Nuclear Physics 2011 (CHEP2011)*, *Journal of Physics: Conference Series,* vol. 331, 032024, 2011.
[3] ROOT, http://root.cern.ch.
[4] Boost, http:/www.boost.org.
[5] CHLEP, http://proj-clhep.web.cern.ch.
[6] Python, http://www.python.org.
[7] S. Agostinelli et al., "Geant4 - a simulation toolkit," *Nuclear Instruments and Methods in Physics Research Section A,* vol. 506, pp. 250-303, 2003; J. Allison et al., "Geant4 developments and applications," *IEEE Transactions on Nuclear Science,* vol, 53, no. 1, pp. 270-278, 2006.
[8] Artistic Style, http://astyle.sourceforge.net.
[9] SCons, http://www.scons.org.
[10] Buildbot, http://www.buildbot.net
[11] Apache Subversion, http://subversion.apache.org.
[12] Doxygen, http://www.doxygen.org.
[13] TWiki, http://twiki.org.
[14] Redmine, http://www.redmine.org.
[15] DIRAC INTERWARE, http://diracgrid.org.
[16] J. Blomer, P. Buncic, I. Charalampidis, A. Harutyunyan, D. Larsen, and R. Meusel, "Status and future perspectives of CERNVM-FS," *Journal of Physics: Conference Series,* vol. 396, 052013, 2012.
[17] R. Itoh et al., "Implementation of parallel processing in the basf2 framework for Belle II," *International Conference on Computing in High Energy and Nuclear Physics 2012 (CHEP2012)*, *Journal of Physics: Conference Series,* vol. 396, 022026, 2012.
[18] Virtual Geometry Model, http://ivana.home.cern.ch/ivana/GVM.html.