Simlator Status of JUPITER and Satellites

Kotoyo Hoshina

Tokyo University of Agriculture and Technology
February 13, 2002

1 Introduction

Monte Carlo simulation comprises an essential part of any modern experimental high energy physics. In the designing stage of a large scale project such as the JLC, its primary goal is to identify important physics targets and then set machine parameters such as beam energy, luminosity, beam energy spread, beamstrahlung, beam related background, etc. and detector parameters such as momentum resolution for charged particle tracking, calorimetric energy resolution, impact parameter resolution, minimum veto angle, particle identification, and so on. In the early phase of the project design, we usually use a quick simulator, which roughly models a detector mainly through parameterization of its performance. Largely based upon parameterization, the quick simulator lacks, however, geometrical details of the detector structure and, consequently, cannot be used to design details of the detector configuration. In order to choose base technology for each detector component and draw its concrete design, we thus need a full simulator which tracks particles through the detector, taking into account various physics processes such as energy loss, multiple scattering, etc. with details of the detector configuration implemented. Such a full simulation is also indispensable for evaluating the effects of beam-related background coming from beam lines or to demonstrate existence proof of some concrete detector design that satisfies the performance goal set with the quick simulation. The other role of the full simulator is to calibrate the quick simulator. From the view point of CPU economy, this is very important, since physics simulation studies usually demand high statistics and the use of any full simulator is impracticable.

The JLC group has a full simulator named JIM[1], which is based on a legacy FORTRAN program, Geant3. Considering the life cycle of the JLC project, however, it is desirable to take full advantage of modern software technology such as **O**bject-**O**riented **P**rogramming (OOP). We are thus developing a full simulator using C++ OOP language. In this article, we first describe the basic design of the new simulator and its current state, and then present an example of its application to studies for central tracker design.

2 Basic Design of the New Full Simulator

The full simulation starts with the tracking of particles fed into the simulator through the detector, taking into account various physical processes such as decays into daughter particles, if unstable, or interactions with materials such as energy loss, multiple scattering, electromagnetic showering, nuclear interactions, and so on. Along the track of each particle, when it passes through a sensitive region of the detector, information such as energy deposit, entering and exiting positions, timing, etc. is stored. These pieces of exact hit information comprise so called *Monte Carlo truth* (MC truth). The most complete simulation involves generation of raw signals from the exact hit information followed by event reconstruction using the same event reconstruction program for real experimental data. Depending upon the study purpose, however, we can sometimes skip the signal generation step and create hit points directly from the exact hit information with appropriate smearing due to finite detector resolutions. The simulator should allow this kind of switching of simulation levels. Since the simulation of various physical processes in the detector is the most time-consuming part of the full event simulation, it is thus advantageous to separate the part that generates the MC truth from the rest of the simulator and reuse the same MC truth for different levels of simulations. The former part is, in our case, named **JUPITER** (**J**LC **U**nified **P**article **I**nteraction and **T**racking **E**mulato**R**)[2] and based on GEANT4 [3]. The latter consists of several modules based on

ROOT[4]/JSF[5]. These modules are named after JUPITER's satellites and thus hereafter called the **Satellites**, whose more detailed explanation is given later.

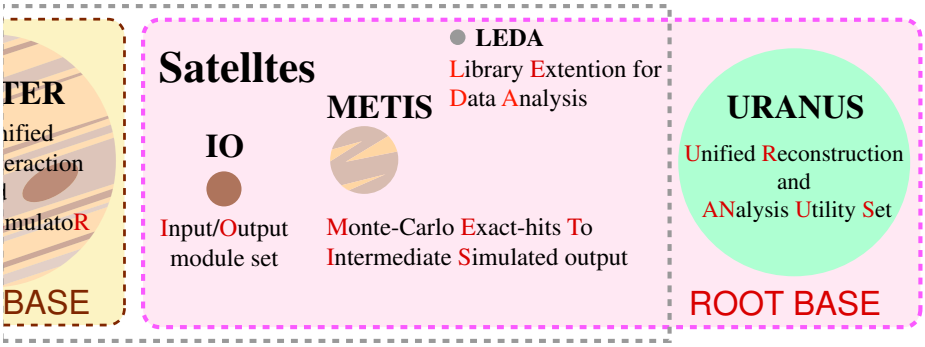

Figure 1:

One of the most important roles of the full simulation is to establish event reconstruction technique for real experimental data. Our event reconstruction program is named **URANUS** (**U**nified **R**econstruction and **AN**alysis **U**tility **S**et). As the name suggests, URANUS does not belong to the Satellites. However, some important fraction of the Satellites inherits from URANUS as explained later so that developing URANUS means developing the Satellites.

2.a JUPITER

One of the design goals of JUPITER is to allow easy modification of geometrical parameters that describe various detector components, since its currently conceived most important usage is to optimize the detector parameters for the JLC. In addition, it should allow simultaneous development of different detector components by different groups of people.

To these ends, JUPITER provides (1) individual work space directory for each detector or accelerator subgroup to assure inter-subgroup independency and (2) a set of base classes to mimic detector building process: assembling of each device and its subsequent installation, thereby facilitating easy replacement, un-installation and re-installation, of any detector component.

2.a.1 Work Space

Fig. 2 shows the directory tree of JUPITER. Every detector or accelerator subgroup is assigned its own directory, in which all the program code related to his/her subgroup is stored. Consequently, developers of a subdetector can update the code for their detector or accelerator component with small labor by just replacing their assigned directory.

The **kern** directory does not belong to any subgroup but is managed by an administrator, and provides a minimum set of JUPITER consisting only of one empty world volume. As its name shows **kern** stands for "**kern**el of JUPITER" and contains common classes for all subgroups such as several base classes that characterize JUPITER, as well as those required by Geant4 including `J4RunAction`, `J4EventAction`, `J4PrimaryGeneratorAction`, and `J4PhysicsList`.

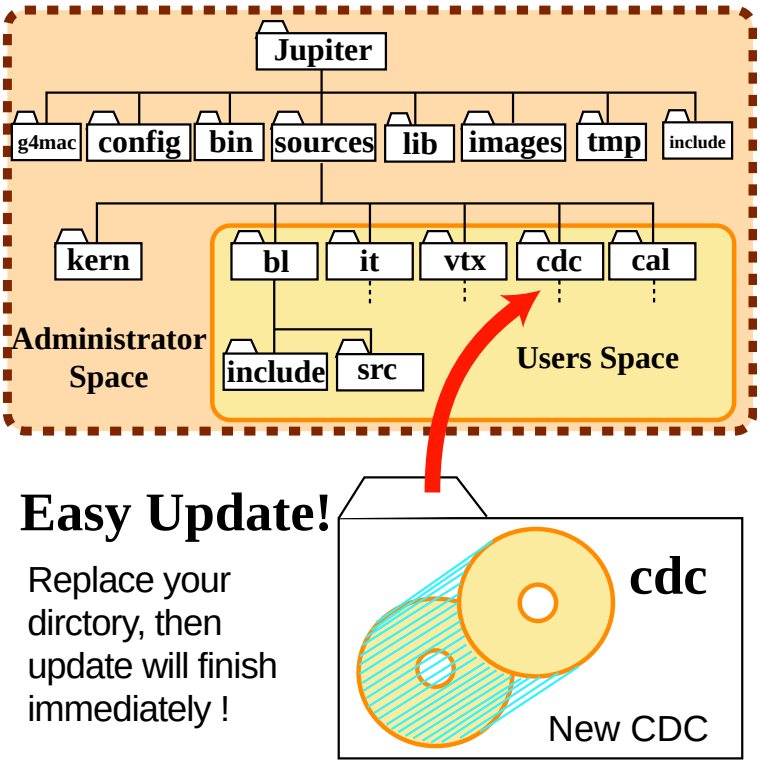

Figure 2:

2.a.2 Base Class of JUPITER

In our design, every geometrical component, even the World Volume, must be a derived class from a base class called `J4VComponent`. This pure virtual base class specifies basic interfaces that any detector component must be equipped with: an `Assemble()` method to define a shape and material of the object as a `G4Solid` and a `G4LogicalVolume`, an `InstallIn()` method to install it into its mother volume, thereby appending positional information to the object as a `G4VPhysicalVolume`. When the detector component in question has a substructure consisting of subdetector components, the invocation of the parent's `Assemble()` method induces recursive calls of the subdetector-components' `Assemble()` methods, in order to assemble all the substructures. The base class also provides an automatic naming system for the hierarchy of the detector components as well as some handy interfaces to frequently used shapes including tubes and boxes.

In addition to the `Assemble()` method, `J4VComponent` has other recursively-called virtual methods `OutputAll()` and `SwitchOn()` & `SwitchOff()`: the former manages output of hit data and the latter two activate/inactivate the sensitive detectors attached to the detector components. The `OutputAll()` method invokes the `OutputAll()` method of its attached sensitive detector, if any and that activated, and then recursively calls the `OutputAll()` method of its subcomponents. The `OutputAll()` method of the attached sensitive detector invokes the virtual `Output()` method of a base class named `J4VHit` derived from `G4VHit`, which induces a call to the `Output()` method of the actual hit class inheriting from `J4VHit` of the detector component by polymorphism. Consequently, all you have to do for saving hit data is to create your own Sensitive Detector and Hit classes by inheriting from `J4VSD` and `J4VHit`, respectively, and then implement the `Output()` method in your Hit class. As already noted, those virtual methods related to the hit data generation are in effect for a given instance of `J4VComponent`

only when it has a pointer to an instance of `J4VSensitiveDetector`, namely, the component is registered as a Sensitive Detector.

For parallel and distributed development of various detector components, JUPITER also provides a base class called `J4VMaterialStore` for material definitions. By instantiating a class object derived from this base class, a developer of each detector component can define materials without conflicting with developers of the other parts.

Fig.3 shows UML picture of those base classes.

2.a.3 Design Features of JUPITER

The main design features of the JUPITER can be summarized as follows:

(1) a base class to provide common, intuitive, thus user-friendly interface to any detector component,

(2) an automatic naming system to avoid name collisions among various geometrical objects representing individual detector components, and

(3) a material management system to allow developers of each subdetector to consistently introduce user-defined materials, while minimizing duplication of pre-registered materials.

Feature (1) sets a unified framework for the software organization, while features (2) and (3) greatly facilitate parallel and distributed development of various detector components by minimizing their interference.

2.a.4 Current Status of JUPITER

Using these classes, we started with development of the CDC part of JUPITER. Soon after this, implementation of the VTX part began. Fig.4-a) shows a top-right quadrant of the tracker part, including VTX, an Inner Tracker (IT), and the CDC. Notice that *very fine structures such as individual sense wires are implemented in the CDC part.* Fig.4-b) is, on the other hand, a blowup of the VTX part, demonstrating the layout of ladders of CCD pixel detectors. Although the calorimeter part of JUPITER is yet to be developed, JUPITER can now handle multi-particle final states as from the process $e^+e^- \to ZH$. A sample event display is shown in Fig.5 for this process generated at $\sqrt{s} = 300$ GeV.

We can clearly see two jets from the $H \to b\bar{b}$ decay, recoiling against a Z boson that decayed invisibly into $\nu\bar{\nu}$. Notice that only charged particles are shown in the figure.

2.b Satellites and URANUS

2.b.1 Structure of Satellites

While JUPITER generates Monte-Carlo truth, its satellites smear it and simulate procedure of event reconstruction. Currently the satellites include the following:

(1) **IO** (**I**nput/**O**utput module set), designed to absorb any change in JUPITER's output format, converts JUPITER's output (plain ASCII text at present) to a ROOT file.

(2) **METIS** (**M**onte Carlo **E**xact hits **T**o **I**ntermediate **S**imulated output) then reads it in and carries out event reconstruction and analysis. METIS inherits from URANUS, which reconstructs and analyzes event data, and facilitates simulation of event reconstruction. In addition, there is

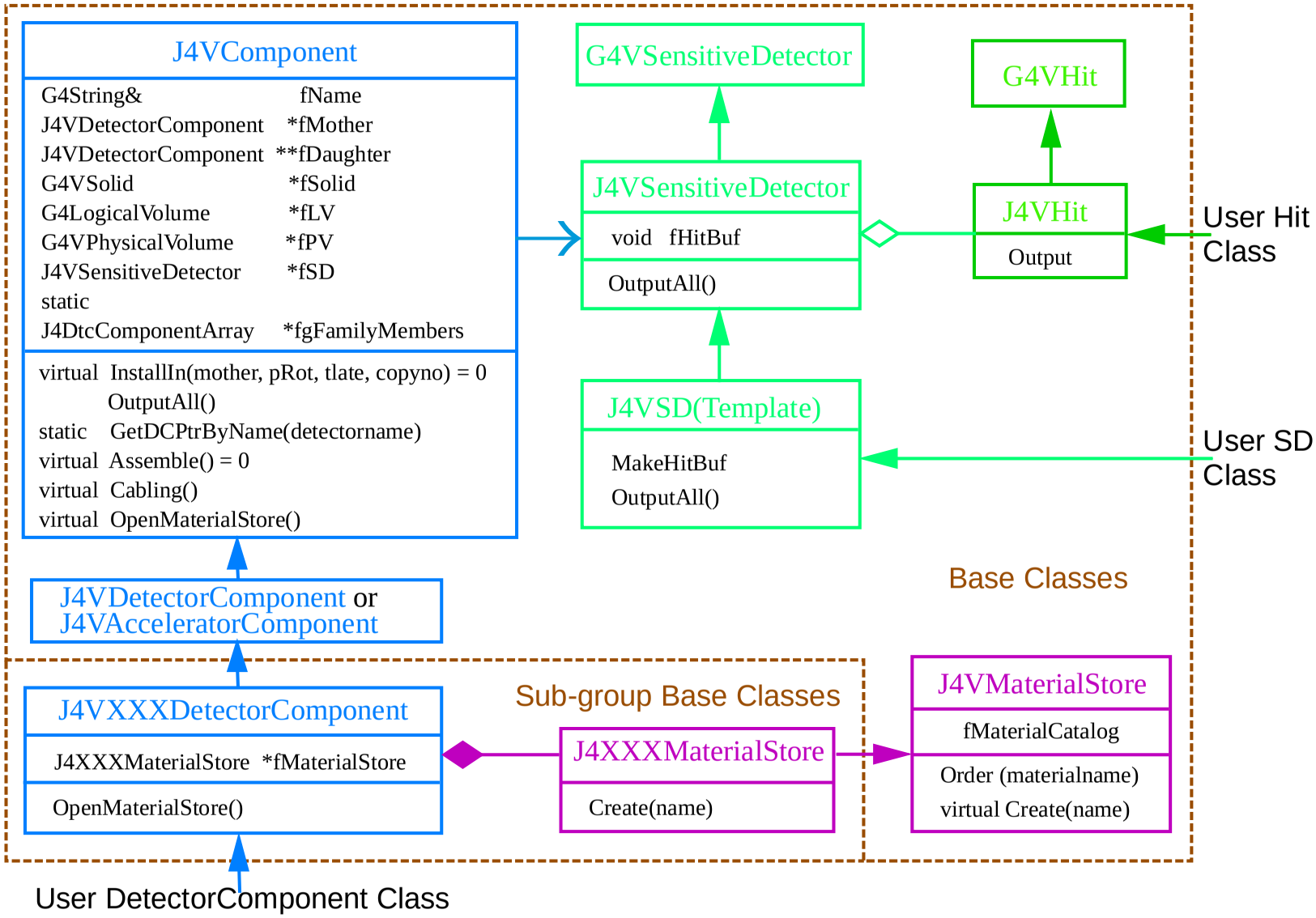
J4VComponent
G4String& fName
J4VDetectorComponent *fMother
J4VDetectorComponent **fDaughter
G4VSolid *fSolid
G4LogicalVolume *fLV
G4VPhysicalVolume *fPV
J4VSensitiveDetector *fSD
static
J4DtcComponentArray *fgFamilyMembers
virtual InstallIn(mother, pRot, tlate, copyno) = 0
OutputAll()
static GetDCPtrByName(detectorname)
virtual Assemble() = 0
virtual Cabling()
virtual OpenMaterialStore()
G4VSensitiveDetector
J4VSensitiveDetector
void fHitBuf
OutputAll()
G4VHit
J4VHit
Output
User Hit Class
J4VSD(Template)
MakeHitBuf
OutputAll()
User SD Class
Base Classes
J4VDetectorComponent or J4VAcceleratorComponent
J4VXXXDetectorComponent
J4XXXMaterialStore *fMaterialStore
OpenMaterialStore()
Sub-group Base Classes
J4XXXMaterialStore
Create(name)
J4VMaterialStore
fMaterialCatalog
Order (materialname)
virtual Create(name)
User DetectorComponent Class

Figure 3:

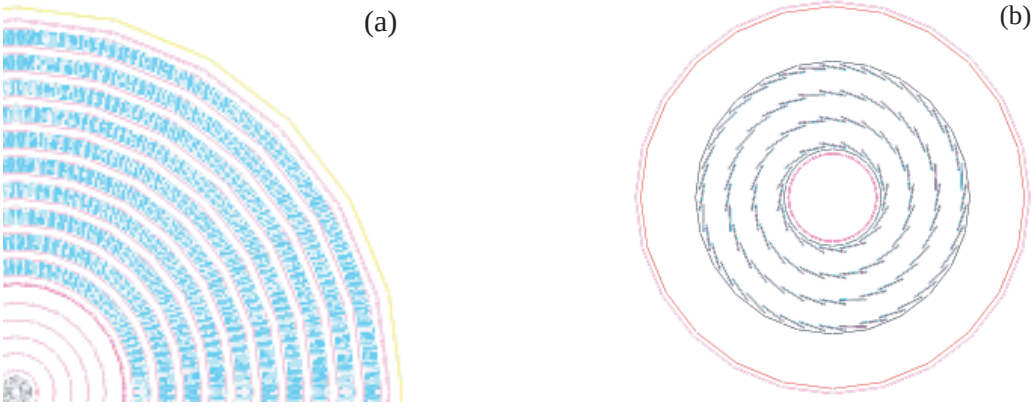

Figure 4:

(3) **LEDA** (**L**ibrary **E**xtension for **D**ata **A**nalysis), which is a general purpose analysis library.

Notice that URANUS is an independent planet, since it is primarily for real data and does not belong to JUPITER.

The concept of our simulation and analysis framework is depicted in Fig.6.

2.b.2 Design Features of Satellites

The framework of Satellites and URANUS is not yet fully fixed, although we can enumerate present features and future plans:

(1) IO will manage all the Input/Output procedure in any stage of the simulation, for example, output of Monte Carlo truth from JUPITER, or intermediate simulated output from METIS.

(2) Because METIS inherits from URANUS, all the event-reconstruction methods in URANUS will be automatically adopted to METIS, thanks to the OOP technology.

(3) Users can change simulation level by choosing modules from Satellites and URANUS. In the case of the central tracker (CDC), we can choose, for instance, next module sets:

1. IO → J4CDCHitMaker → J4CDCTrackFinder → U4CDCTrackFitter
2. IO → J4CDCHitMaker → U4CDCTrackFinder → U4CDCTrackFitter
3. ⋮

Procedure of event reconstruction is, however, tightly connected with characteristics of individual detector components. It is thus conceivable that there will be only a little to add to the JSF/ROOT framework to complete a general framework of URANUS and Satellites, except for the IO interface.

3 Example of CDC Analysis

Using JUPITER and its Satellites, we studied surviving hit fraction and momentum resolution for the CDC. As Fig.5 shows, charged particles are often produced in narrowly collimated jets

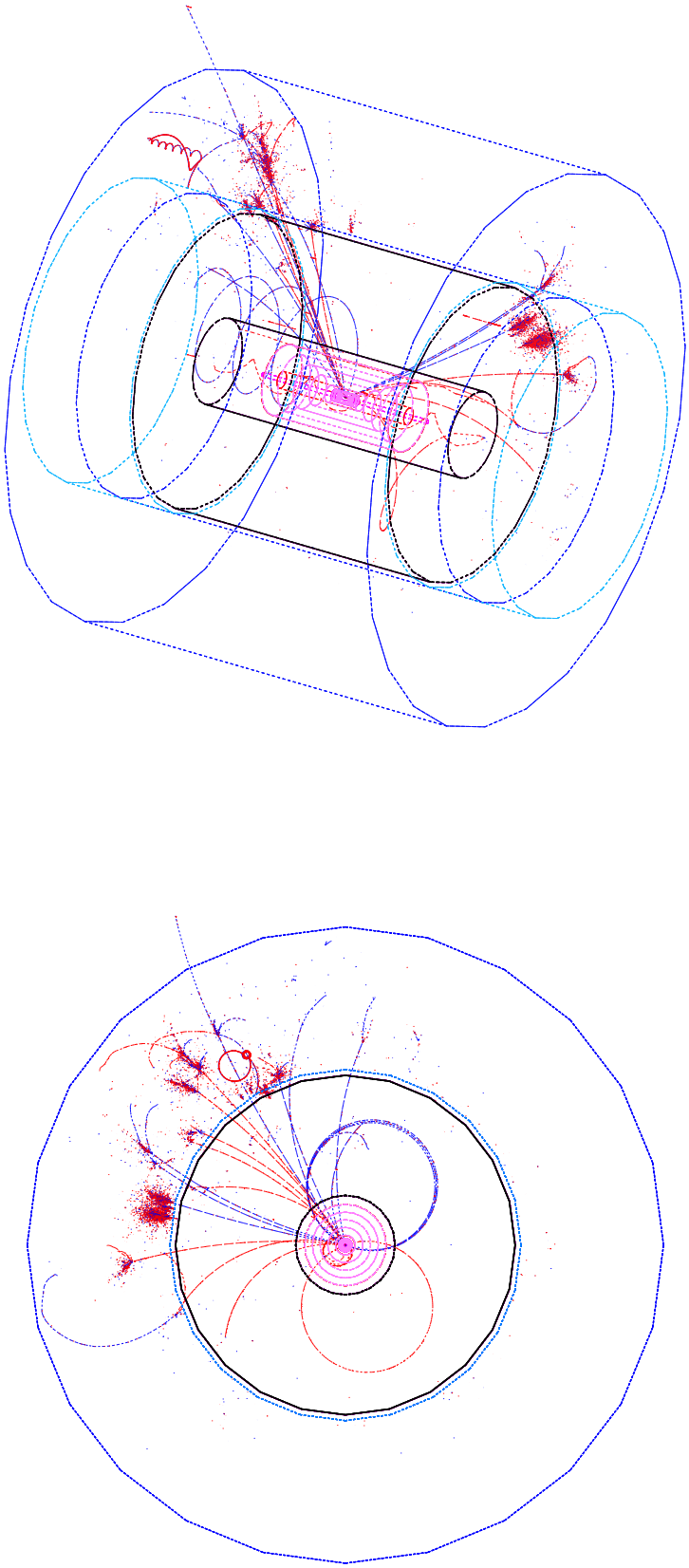

Figure 5: A typical event of $e^+e^- \to ZH$, $Z \to \nu\bar{\nu}$, $H \to b\bar{b}$.

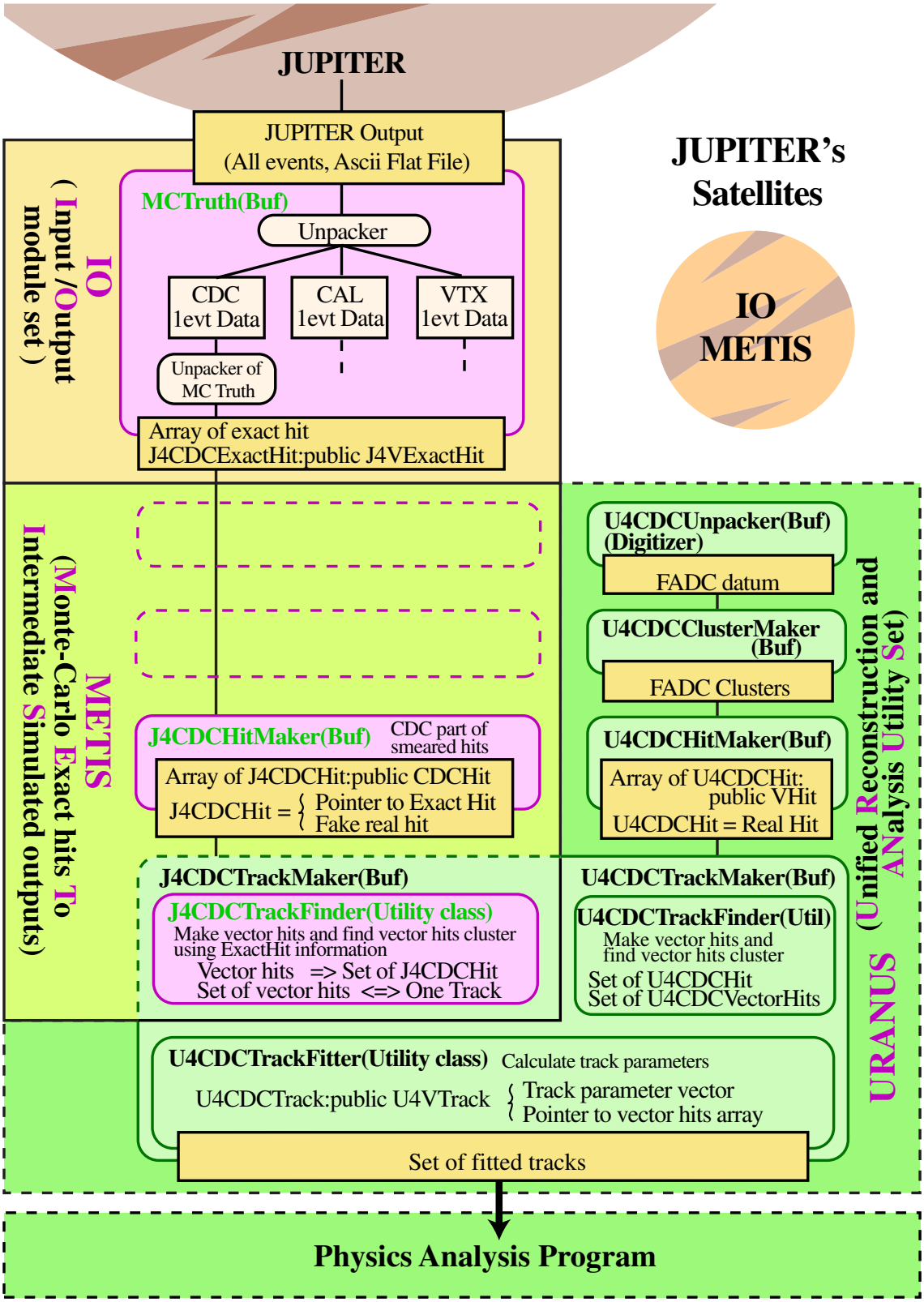
JUPITER
JUPITER Output
(All events, Ascii Flat File)
JUPITER's Satellites
IO METIS
IO (Input /Output module set)
MCTruth(Buf)
Unpacker
CDC 1evt Data
CAL 1evt Data
VTX 1evt Data
Unpacker of MC Truth
Array of exact hit
J4CDCExactHit:public J4VExactHit
METIS (Monte-Carlo Exact hits To Intermediate Simulated outputs)
J4CDCHitMaker(Buf)
CDC part of smeared hits
Array of J4CDCHit:public CDCHit
J4CDCHit = Pointer to Exact Hit / Fake real hit
J4CDCTrackMaker(Buf)
J4CDCTrackFinder(Utility class)
Make vector hits and find vector hits cluster using ExactHit information
Vector hits => Set of J4CDCHit
Set of vector hits <=> One Track
URANUS (Unified Reconstruction and ANalysis Utility Set)
U4CDCUnpacker(Buf) (Digitizer)
FADC datum
U4CDCClusterMaker (Buf)
FADC Clusters
U4CDCHitMaker(Buf)
Array of U4CDCHit: public VHit
U4CDCHit = Real Hit
U4CDCTrackMaker(Buf)
U4CDCTrackFinder(Util)
Make vector hits and find vector hits cluster
Set of U4CDCHit
Set of U4CDCVectorHits
U4CDCTrackFitter(Utility class) Calculate track parameters
U4CDCTrack:public U4VTrack
Track parameter vector
Pointer to vector hits array
Set of fitted tracks
Physics Analysis Program

Figure 6:

at JLC experiments. In order to reconstruct such a jetty event in terms of heavy partons such as W/Z bosons, the Higgs boson, or the top quark, we need an energy flow resolution which is good enough to separate a W from Z with the jet invariant mass method. In order to achieve such a high resolution, we need to use tracking information whenever possible and limit the use of calorimetry to neutral particles. Two-hit separation will thus be very important at the JLC to assure good momentum resolution for tracks in jets. The question is thus how many hits will be lost when 2-hit separation is, say, 2mm and how it affects the momentum resolution. In the following subsections, we discuss these items.

3.a Surviving Hit Fraction

In a real experiment, when a pair of hits are created by two closely separated tracks in the same drift cell, the first hit that is closer to the sense wire will leave a local space charge due to avalanche and might affect the second hit that is further away from the sense wire. In the worst case, the second hit will be lost and the momentum resolution of the track to which the lost hit belongs to will be deteriorated.

In order to investigate the effect of such losses of hits, we killed hits artificially when the second hit was created within 2mm from the first one (see Fig. 7). Fig. 8 shows the results of our analysis of (a) how many hits belonging to a track (pt $>$ 1GeV), (b) the number of surviving hits per the number of original hits (surviving hit fraction), (c) Layer dependency of the surviving hit fraction, and (4) the distribution of minimum 2-hit distance, under this most pessimistic condition.

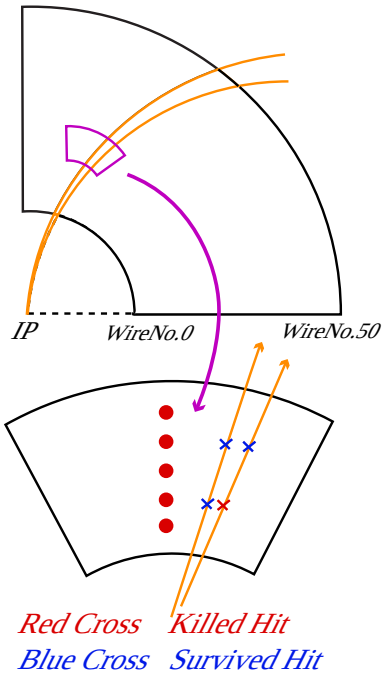

Figure 7:

Fig. 7-(b) tells us that we lose only 10% of hits on average if two-hit separation is 2mm. From the point of view of momentum resolution, however, the loss of the lever arm length is more serious than that of the number of hits. Fig. 7-(c) indicates the surviving hit fraction decreases as as we move from the outermost layers to the innermost layers: 20% of tracks will lose their lever arm lengths.

It is, however, possible to compensate the loss of the lever arm length by using other trackers such as VTX or an Intermediate Tracker. Empirically, it is known that using inner trackers has the same effect as imposing an Interaction Point (IP) constraint. In the next subsection, we discuss the momentum resolution with or without the IP constraint.

3.b Momentum Resolution of the CDC

Fig. 9 plots momentum resolutions in the low momentum region. When 2-hit separation capability is 2mm, the momentum resolution of the CDC alone deteriorates by almost 10% compared to the ideal case of 2-hit separation being 0mm. When the IP constraint is imposed, the momentum resolution recovers almost completely. We can thus conclude that the requirement of the 2-hit separation of 2mm or better can be regarded reasonable.

4 Summary and Future Direction

A basic framework for detector simulation and analysis (**JUPITER**, its **Satellites**, and **URANUS**) based on GEANT4 and ROOT/JSF has been developed and is now being used to develop a full JLC detector simulator. JUPITER, equipped with basic tracking detectors including VTX and the CDC, can generate hits for multi-hadronic final states such as those from $e^+e^- \to ZH$ and can be used, together with its Satellites, to optimize detector parameters, even though some detector components such as calorimeters or muon detectors are still missing. Recently, beamline implementation has also begun to simulate beam-related background.

Using this simulator, we checked surviving hit fraction for the CDC. Under the condition of 2-hit separation of 2mm, we lose 10% of hit points on average and 20% at the innermost layer, implying loss of lever arm length and consequently deterioration of momentum resolution. However, the momentum resolution can be recovered by applying the IP constraint, which has, empirically, the same effect as including information from inner trackers.

Acknowledgments

The authors would like to thank all the members of the ACFA physics working group and JLC software group. This work was supported in part by the JSPS Japanese-German Cooperation Program.

References

[1] http://www-jlc.kek.jp/subg/offl/jim/index-e.html

[2] http://www-jlc.kek.jp/ hoshina/

[3] http://wwwinfo.cern.ch/asd/geant4/geant4.html

[4] http://root.cern.ch/

[5] http://www-jlc.kek.jp/subg/offl/jsf/jsf.html

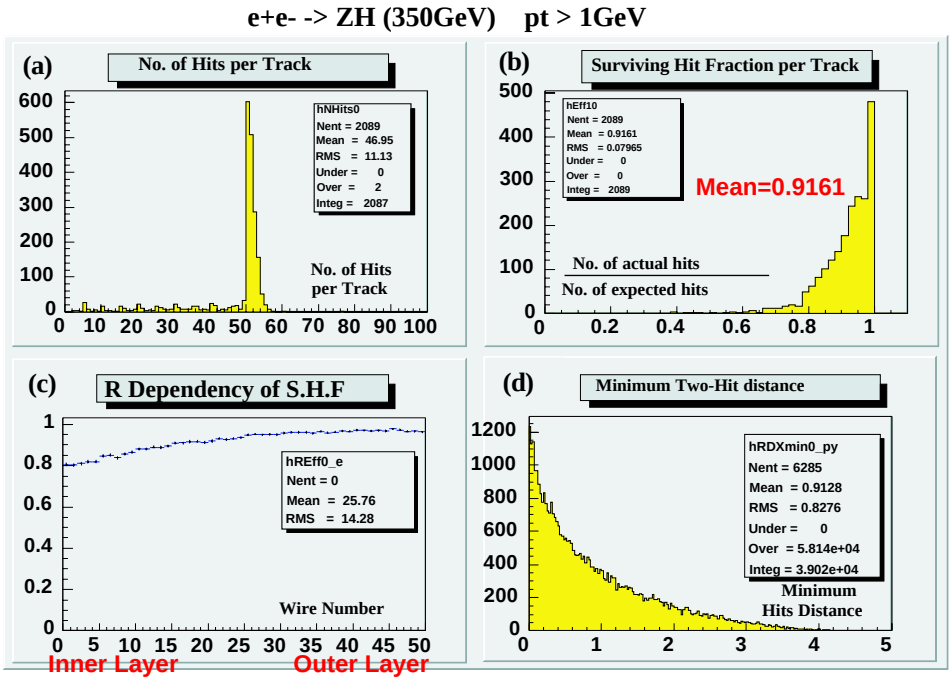

Figure 8:

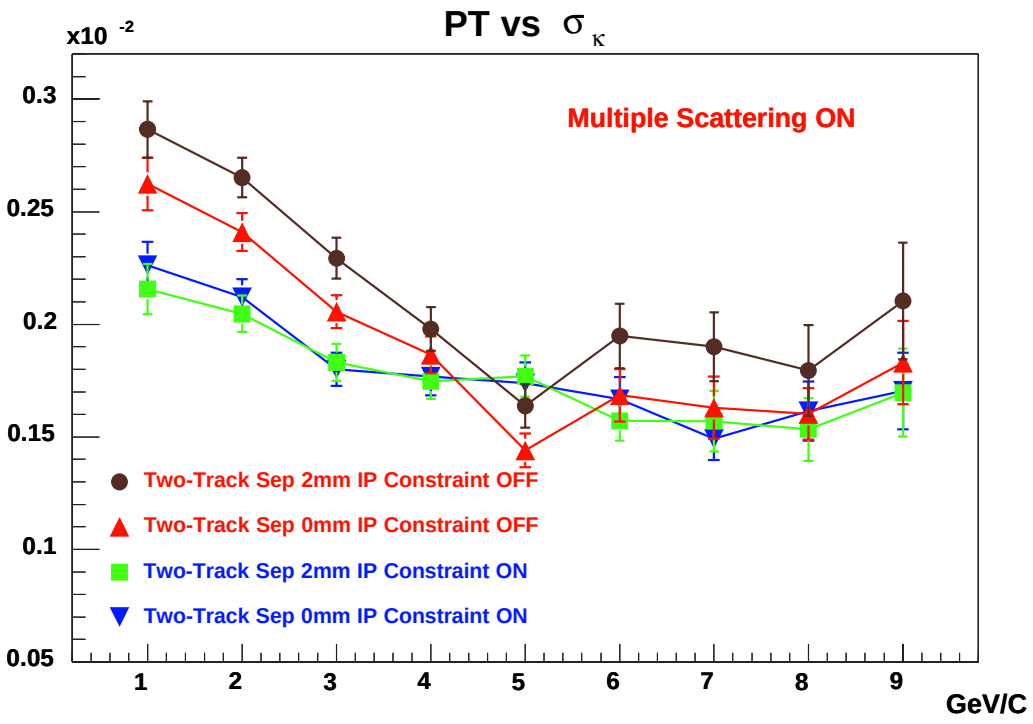

Figure 9: