Requirements to Software Framework and Comparison of Possible Candidates

R. Itoh, KEK

Outline

1. Requirements to Software Framework

2. Possible Candidates and Comparison
   - BASF
   - GAUDI
   - Marlin
   - roobasf

3. Future direction and plan

Requirements:

- Katayama-san (and I) are now trying to collect requirements to the framework starting from the use-case study.
- It is on the wiki:
  http://wiki.kek.jp/display/sbelle/Use+cases

Use cases

作成者: KATAYAMA Nobuhiko; 最終更新者: KATAYAMA Nobuhiko; 最終更新日

Here we would like to write down use cases of the frame work.

子ページ (15)

- Japanese version (incomplete and stopped as of Feb. 23)
- Use case 1 - Itoh san
- Use case 2 - MacNaughton san (shifter)
- Use case 3 - Adachi san
- Use case 4 - Mike Jones san
- Use case 5 - Ozaki san
- Use case 6 - Iwasaki san
- Use case 7 - Nishida san
- Use case 8 - Hara san
- Use case 9 - Nakamura san
- Use case 10 - Schumman san
- Use case 11 - Inami san
- Use case 12 - Wicht san
- Use case 13 - Kinoshita san
- Use case 14 - Uehara san

Use case 1 - Itoh san

作成者: KATAYAMA Nobuhiko; 最終更新者: KATAYAMA Nobuhiko; 最終更新日: Feb 24, 2009

Itoh san is responsible for the DAQ system for Belle. We use Basf for online data taking system. Just like tiny streams become big river, data from hundreds of thousands of sensors are gathered event by event and events are build and high level software triggering is performed. Some events are thrown away and others are kept and written to storage system. The daq system is almost all automated.

Functional requrements:

1. parallel processing through out the daq system
2. synchronization of events during the event building processes
3. software trigger to select events should be run
4. results of software trigger should be sent to colleagues(not just downstream)
5. data should be corrected for calibration due to electronics and other things
6. must deal with wave form sampling data somewhere (upstream/hardware?)
7. zero suppression and compression
8. need to monitor status of processes and sensors
9. begin run/ end run processing; system resets, reload constants
10. output file size, change file
11. run changes should be instantaneous
12. the order of events are not too important. There could be more than one output streams
13. multiple streams for different types of events?

1. Requirements to SB Framework

a) Modular structure = Software Bus

* Dynamic-linkable user-supplied analysis modules
* Flexible execution control of modules

b) Object I/O

* Capability to read/write C++ object.
* Independent of specific data models

c) Interface to distributed computing

* Parallel processing
* Data file access over WAN (i.e. GRID)

d) Input/Output file management

* Database interface

e) Integrated histogram/N-tuple management

f) Programmable script parser for the control

g) Dynamic configuration of framework components

h) Target Use: DAQ(readout/HLT), data production, MC production, user analysis

* Unified framework for all cases is desired.

i) Backward compatibility

* Compatibility with Belle software/data

a) Modular structure

- Modules are coded by inheriting a common “module” class
- The class should consist of at least
  * initialization
  * event processing
  * termination

  functions.
  + *More functions might be necessary for the management of histograms, statistics display, begin_run/end_run processing, etc.*
- Coded primarily in C++. We should forget legacy languages.
- Built as a shared object and plugged into framework by dyn. link.
- Flexible execution of module chain driven by a script language
- Object passing among modules:
  * pass pointer to a common object
  * define input and output objects of each module
  * access through proxy

b) Object I/O

"Object serialization (streaming)" is the main concern.

- Event object in a file is stored in "serialized" format.
  <- convert "objects" to some format which can be stored in a file.
- Known object serialization(streaming) method:
  1. ROOT IO (used in GAUDI and roobasf)
  2. SIO (SLAC product for ILC, used in Marlin)
  3. BOOST serialization (Advanced C++ standard)
  4. Panther C++ interface (used in BASF)
- Serializer code:
  * ROOT IO can automatically generate the code by running "rootcint" with class header files.
  * Need to be provided by users for SIO and BOOST
  * Serializer is supported for "LCIO with SIO" and Panther, however, the object component type is restricted.
    - *Marlin's LCIO is basically a collection of serializer codes assuming the LC data model. Addition of new object is possible, but the structure is restricted to array of ....*
    - *Panther C++ interface works similarly by defining objects as Panther tables.*

c) Interface to distributed computing

- For the massive data processing (HLT processing in DAQ, data production, MC production and heavy user analyses), a distributed parallel processing is required.
  * Belle's BASF and dBASF(RFARM) have a parallel procesing capability for a single data stream and it is quite successful in the rapid data processing (both DAQ and offline) with less load for the file management.
  * Method of parallel processing:
    a) multiple I/O streams processed by a collection of single framework nodes
       -> parallel processing scheme outside framework is required.
          ... not suited for HLT
       => GAUDI, Marlin
    b) single I/O stream with the parallel processing integrated in the framework
       => BASF, dBASF, roobasf

- Access to distributed file is also required.
  * GRID based remote file access : SRM, SRB.....
  * xrootd

d) Input/Output file management

- Management of names of the input/output files
  * logging to database whenever read/written.
  * retrieve unique file name in the distributed space by a generic specification of data stream.

- Distributed database combined with Object I/O is required.

- Candidates:
  * POOL (as used with GAUDI)
    = combination of MySQL (or other DB) + ROOT IO
  * xrootd

- More consideration is needed.....

e) Integrated histogram/N-tuple management

- Histograms/N-tuples are defined/accumulated by modules independently and a mechanism to organize them is required in the framework.

- ROOT based histogram management is the default.

- Have to manage the histogram collection from multiple nodes in case of parallel processing.

- Real-time histogram collection mechanism for the online monitoring is necessary when used in HLT farm.

f) Script parser for the control

- Need a good script parser to describe the data processing on the framework.

- It is desired to use widely-used scripting language as the parser:
  * Custom (BASF, GAUDI, roobasf-prototype) – not desired
  * Tcl/Tk (BaBar/CDF framework)
  * Perl
  * Python (roobasf in future)
  * XML (Marlin)

- Could be a "programming language" to describe the data processing sequence, i.e. loop, conditional jump, etc in module execution, providing transparent interface to C++ in which the framework codes is written.

2. Possible candidates for SuperBelle framework

a) BASF/dBASF

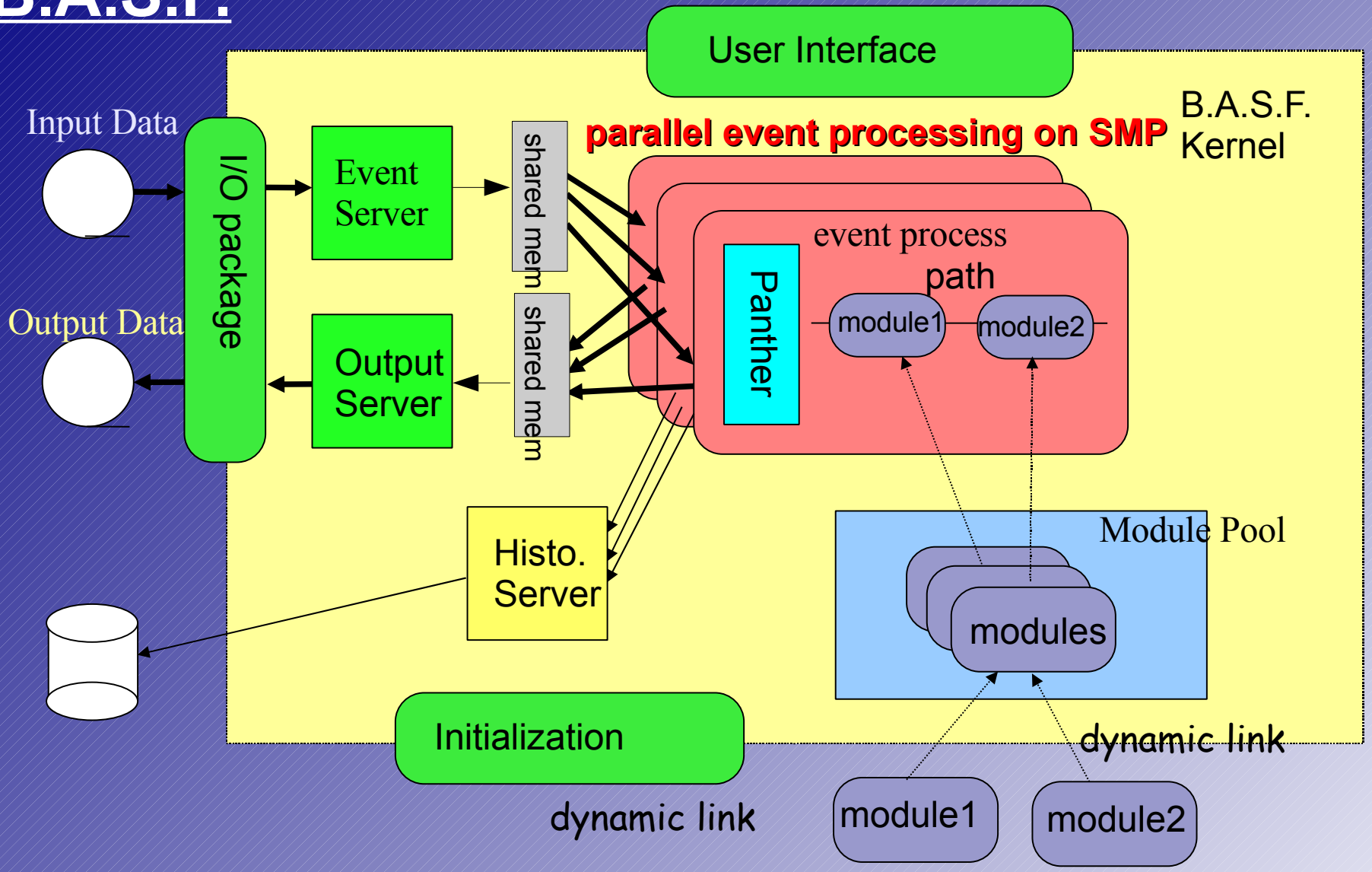
- Currently used in Belle experiment

Software bus	: Yes
Object I/O	: Panther C++ interface
Dist. comp.	: Integrated parallel processing (SMP/network cluster)
File mngmnt	: No (ad-hoc GRID interface exists)
Histogram	: HBOOK4 + Wrapper (BelleHistogram/Tuple) Real time collection mechanism implemented.
Script Parser	: Custom, user interface is replaceable by dyn. link

Problem when used as SB framework:
* Limitation in the object data structure for Object I/O.
* ROOT (data stream/histograms/N-tuples) is not supported.

B.A.S.F.

developed in 1996



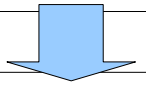
* Green boxes are linked using dynamic link as well as “modules”.
* Data handling is done through “Panther” package.
* Separate package for histogram/N-tuple management : HBOOK4

Panther and Object I/O

TDF file -> “bbstable” command -> Converted to a class header file

TDF file

```
Define_Table Gen_PakVTX 200 "Packed Gen_HEPEVT (vertex part)"
  Real*4  VX      "VX (no change)"
  Real*4  VY      "VY (no change)"
  Real*4  VZ      "VZ (no change)"
  Real*4  T       "T  (no change)"
End_of_Define_Table Gen_PakVTX
```



generated by “bbstable”

Header file

```
class Gen_pakvtx {

  public:
     Gen_pakvtx():m_p(0){}
     Gen_pakvtx(const Gen_pakvtx&e):m_p(e.m_p){}
  private:
  friend class Table_Manager<Gen_pakvtx>;
     Gen_pakvtx(const Panther_ID&, PNT_TableIndex*);
     void newent(void);
  public:
    ~Gen_pakvtx(){}

  public: // Extractors
    float VX(void) const;
    float VY(void) const;
    float VZ(void) const;
    float T(void) const;
    Panther_ID get_ID(void) const { return Panther_ID(m_p?m_p->m_ID:0); }

  public: // Modifiers
    float VX(float);
    float VY(float);
    float VZ(float);
    float T(float);


  ..........
};
```

Panther Object I/O (cont)

- Panther I/O : Event by event I/O of a set of tables
  -> equivalent to serialized object I/O

- Schema evolution is (supposed to be) implemented.
  (in principle, but is not working for now.)

- No dependency on any specific data models

- Entity-relation can be defined between tables=objects
  1-to-n, n-to-1 and n-to-n

- Limitation:
  * Data type in the object is restricted.
    Fortran data type + pointer to other table entity
  * Table must be defined for every objects in event
    in DDL(data definition language).
    => Not true OO data handling.

b) GAUDI

- Currently used in ATLAS and LHCb

Software bus : Yes
Object I/O : ROOT IO
Dist. comp. : Outside framework is required = GRID batch
File mngmnt : POOL (ROOT IO + GRID DB)
Histogram : ROOT
No real time collection.
Script Parser : Custom

Problem when used as SB framework:
* Need to develop outside parallel processing framework
* Too heavy.
* No experts around here.....

GAUDI

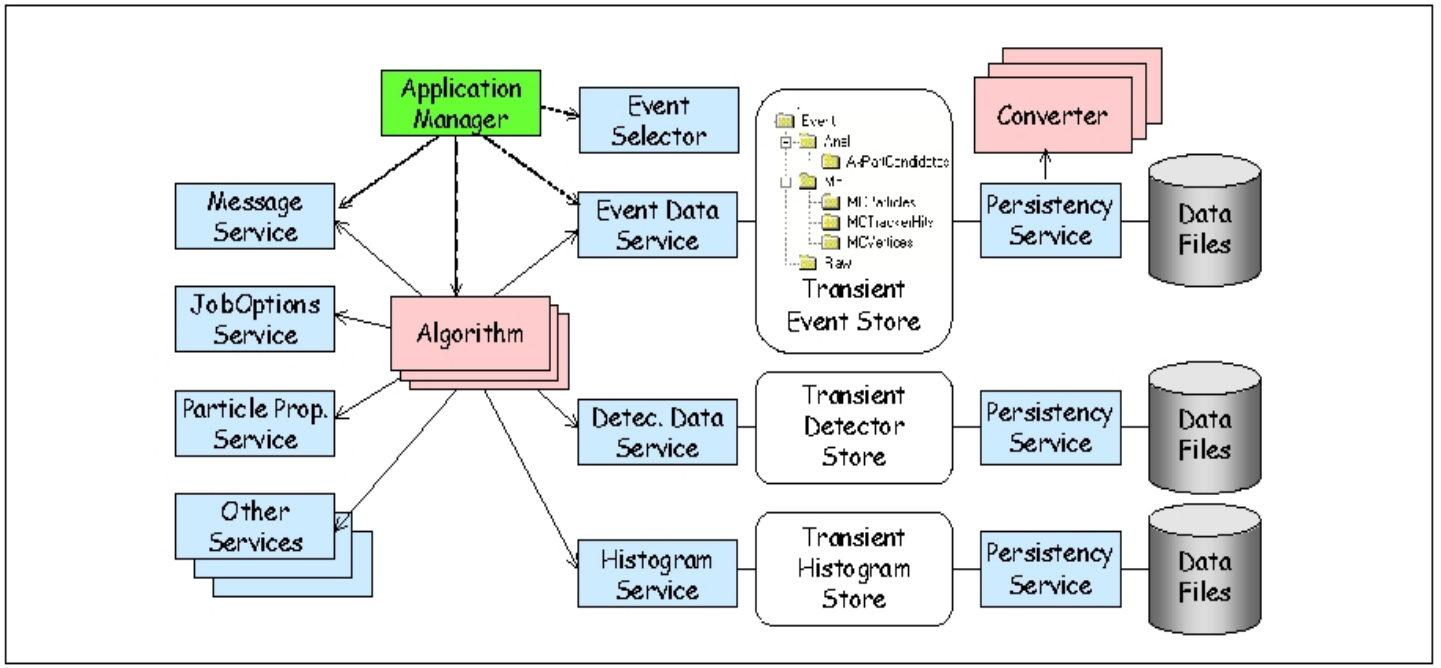


Figure 2.1 Gaudi Architecture Object Diagram

- "module"(BASF) = "algorithm" (GAUDI) , more versatile management
- Algorithm and data are clearly separated
- ROOT IO based object persistency (POOL)
- ROOT based histogram/N-tuple management

  * "Proxy" based access to event and other objects.

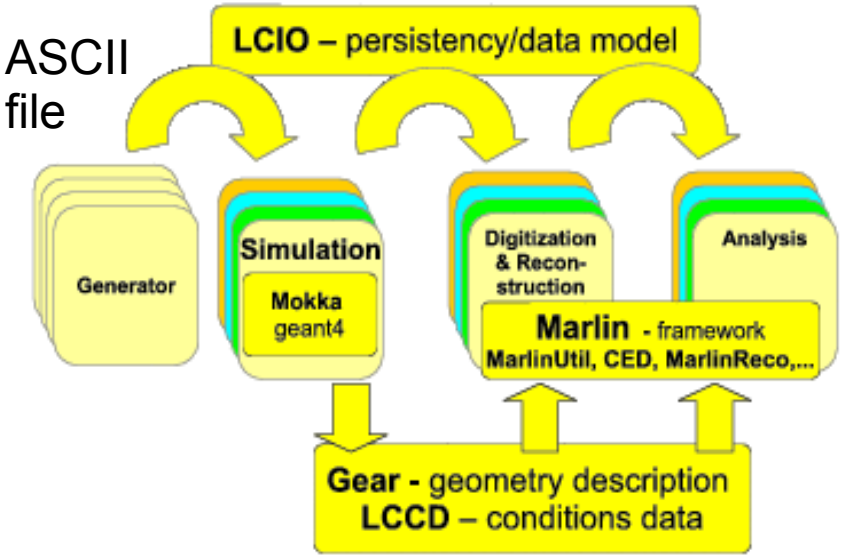
c) Marlin

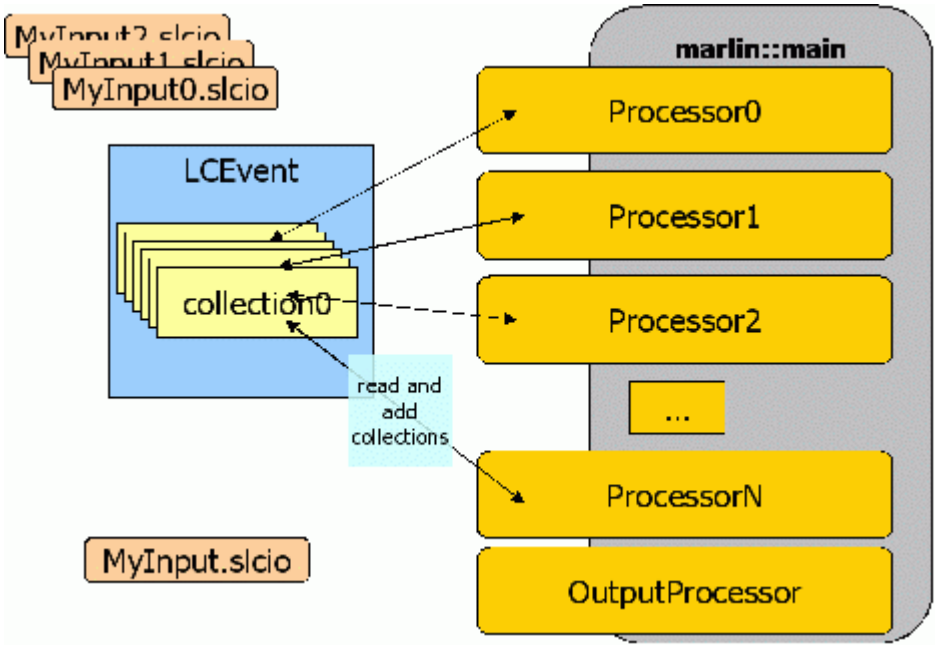
- Currently used in ILD (German group)

Software bus	: Yes
Object I/O	: LCIO implemented using SIO
Dist. comp.	: Outside framework is required
File mngmnt	: No (thru. SIO)
Histogram	: No particular package/management assumed. No real time collection.
Script Parser	: XML

Problem when used as SB framework:
* Too much dependence on LC data model
* Restriction to object handling by LCIO (just like BASF/Panther)
* Need to develop outside parallel processing framework

Mokka and Marlin





- Event generator and simulator(Mokka/G4) are separated from framework(Marlin) by some historical reason and difficult to be implemented as Marlin processors (by Frank).
- Data modeling and object I/O is strongly coupled in LCIO. since LCIO is a collection of serializer of LC objects for “SIO”.

- Actual I/O is based on “SIO” whose support was terminated....
- Another framework for ILD called JSF/Jupiter exists which is based on ROOT and the merging with it is under discussion.

LCIO data model :

- Cross reference of objects can be handled:
  1-to-n, n-to-1, n-to-n
- Supported data types in base objects are restricted to
  `(arrays of) char, int, float and double`
  <- limitation from Fortran I/F? Quite similar to Panther!

* Adding new objects to LCIO data model
  - Basically very difficult
  - It is possible to add user object using “LCGenericObject”, however,the data types are also restricted and the access has to be made using special functions.

```
class LCGenericObject:
create                  -> pgob    = lcgobcreate()
create (dimensions)     -> pgob    = lcgobcreatefixed( nint, nfloat,
ndouble )
delete                  -> status  = lcgobdelete( pgob )
setIntVal               -> status  = lcgobsetintval( pgob, i, ival )
setFloatVal             -> status  = lcgobsetfloatval( pgob, i, fval )
setDoubleVal            -> status  = lcgobsetdoubleval( pgob, i,
dval )
id                      -> id      = lcgobid( pgob )
getNInt                 -> nint    = lcgobgetnint( pgob )
getNFloat               -> nfloat  = lcgobgetnfloat( pgob )
getNDouble              -> ndouble = lcgobgetdoubleval( pgob )
getIntVal               -> ival    = lcgobgetintval( pgob , i )
(i=1,...,nint)
getFloatVal             -> fval    = lcgobgetfloatval( pgob , i )
(i=1,...,nfloat)
getDoubleVal            -> dval    = lcgobsetdoubleval( pgob , i )
(i=1,...,ndouble)
```

d) roobasf

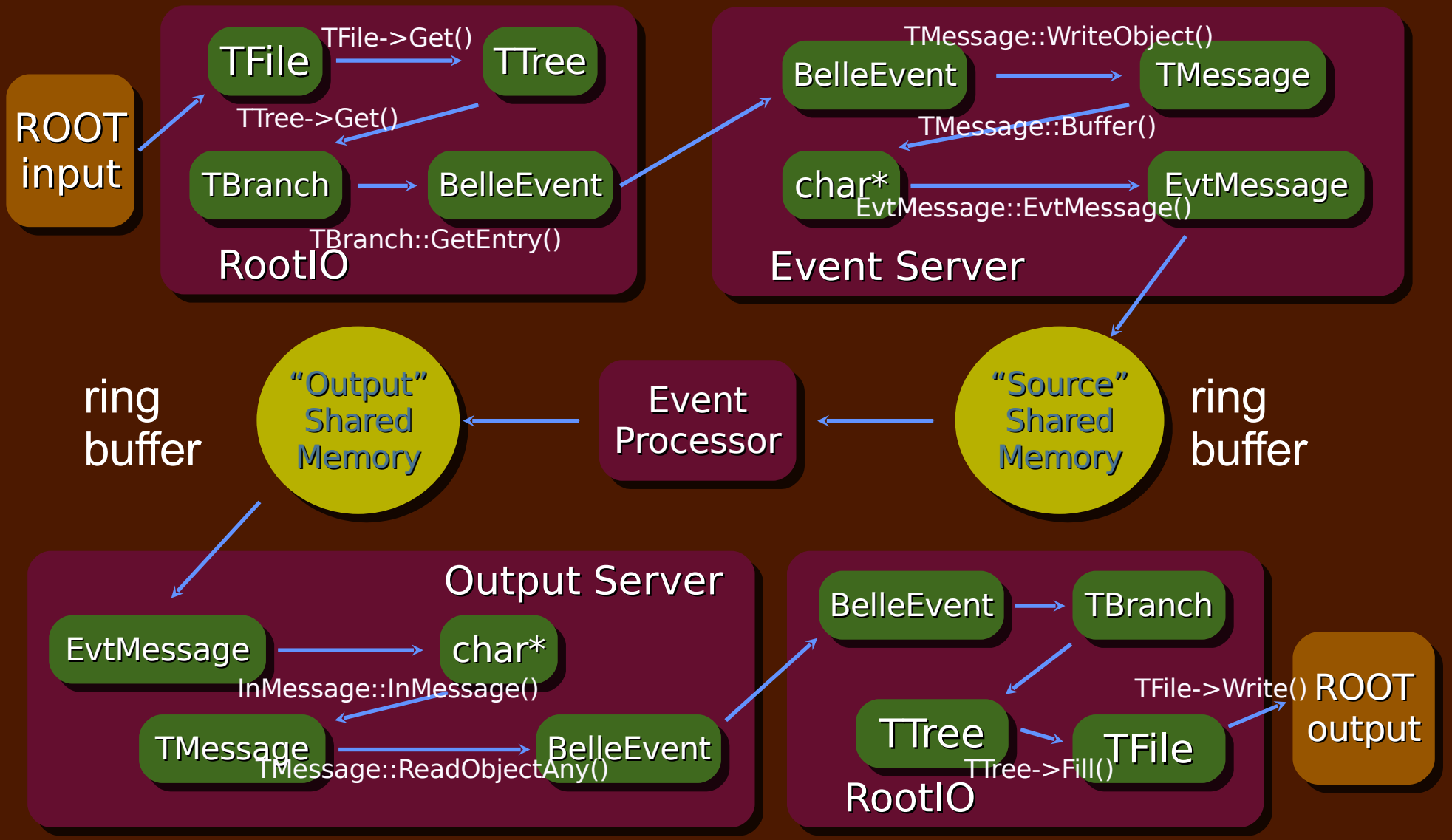
- Under development for SuperKEKB and HSC based on BASF

Software bus	: Yes
Object I/O	: ROOT IO
Dist. comp.	: Built-in parallel processing for SMP and network clst.
File mngmnt	: planned (POOL / xrootd)
Histogram	: ROOT+Wrapper class Real time collection will be implemented
Script Parser	: custom -> Python

- Still under development
  refer to status reports by Katayama-san and S.H.Lee at DUMs
- Prototype with parallel processing is being tested /used in HSC now.
- Histogram collection mechanism and Panther interface
  will be implemented in one or two months for the field test in Belle.
- Started the study of Python scripting

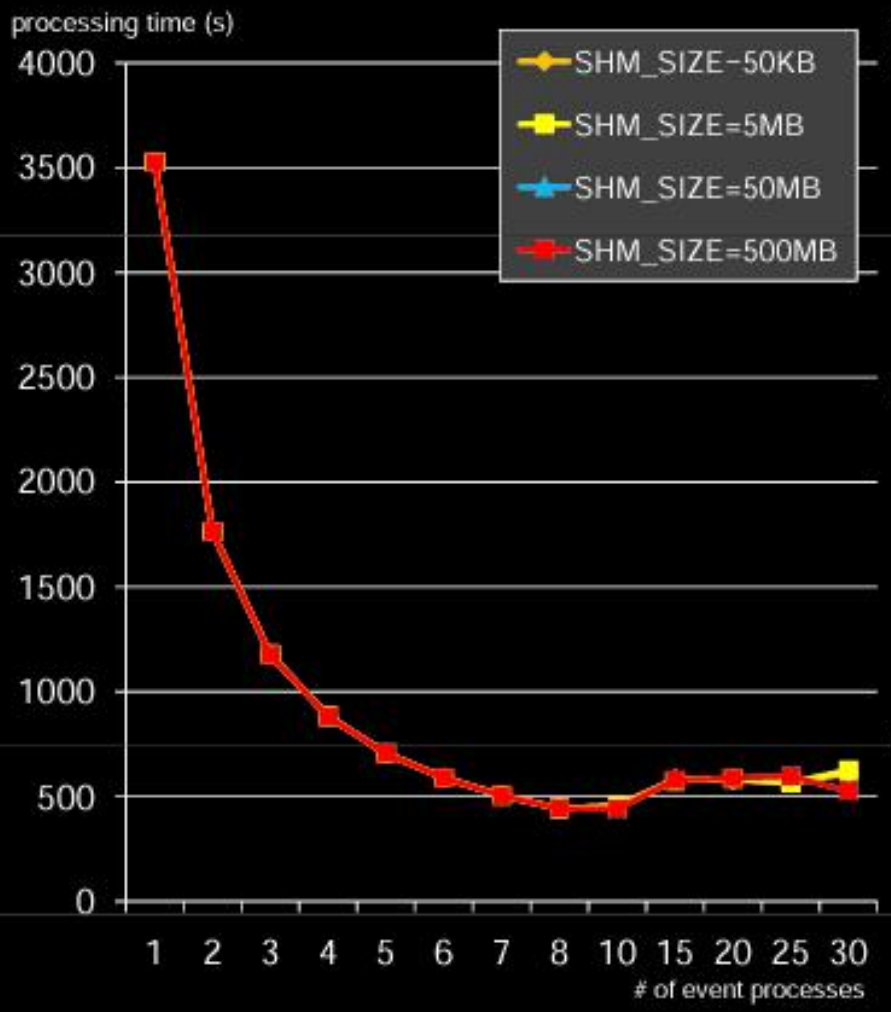
Data Flow Scheme

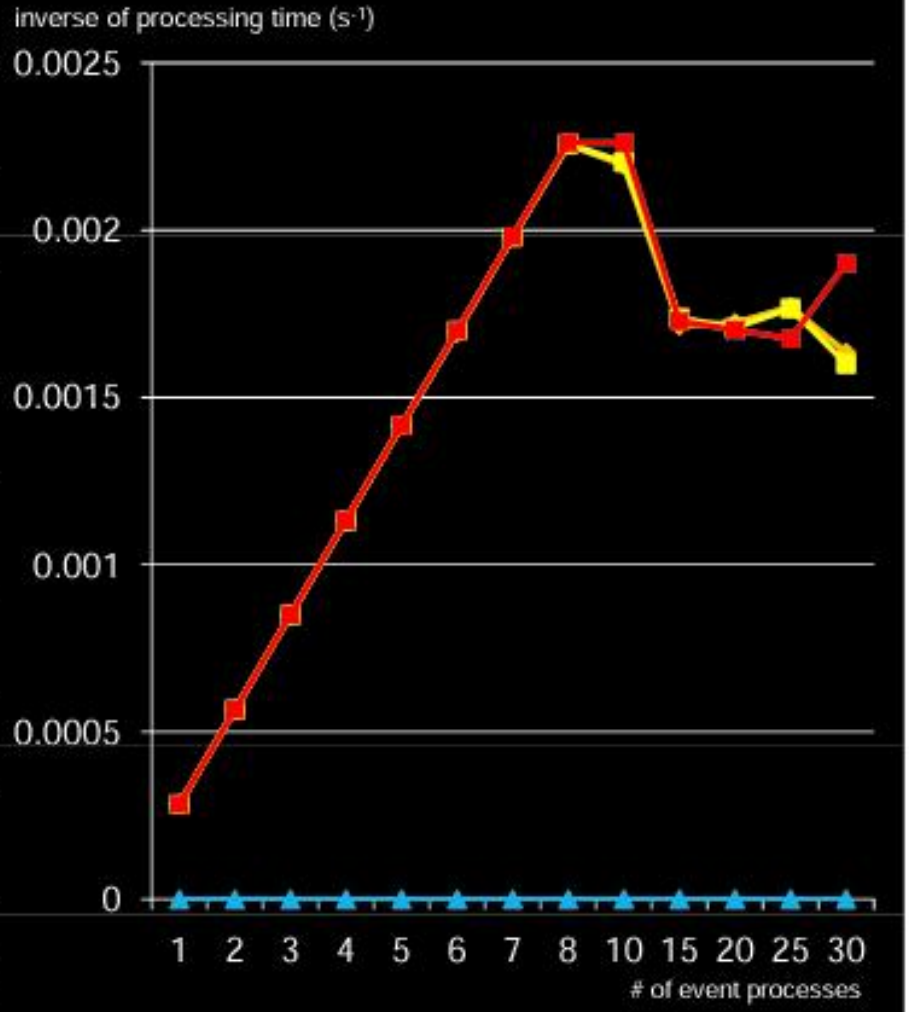
S.H.Lee



Performance Test – vs. size of the shared memory

- 8-core system (Xeon E5405 (2Ghz, Quad-core) * 2, 8GB RAM, 5GB swap space)
- # of events to be processed: 100,000 events (~300KB per events)





R. Itoh (KEK), N. Katayama (KEK)
S. Lee (Korea University), S. Mineo (University of Tokyo)

3. Future direction and plan

- Framework provides the basis for various software development at SuperKEKB
  -> Need to decide as early as possible

- Practically speaking, realistic candidates are two:
  1. “Marlin” offered by German group
  2. “roobasf” being jointly developed by Belle and HSC
     * GAUDI – no experts around here
     * BASF – taken over by roobasf

- Decision should be made by considering the actual requirements to the framework.

- We will focus on the discussion on the requirements and make the decision by autumn at the latest.

Personal pros and cons to candidates

a) Marlin

pros:
- It is actually being used for the SuperKEKB detector simulation.
- Basic required functions as a framework are already built-in.
- Nice scripting system based on XML.

cons:
- Too heavy dependence on LC data model. If we decide to use Marlin, that means SuperKEKB data model has to be constructed based on LC data model.
  -> big issue which affects on the whole SuperKEKB software design.
- Marlin is basically the “old generation” framework to which BASF belongs as well.
  * Object I/O (LCIO) is old style (just like Panther/BASF).
- Worries on future support.
  * Support for SIO (SLAC product) is already terminated.
  * Development team is outside SuperKEKB.
  * LC data model is not combat proven. It is basically developed for MC study.
  * Not established even in ILD community. Merging with JSF is discussed.
- No parallel processing support
  -> External parallel processing framework is necessary to be developed for massive data production.

Predefined LC data model

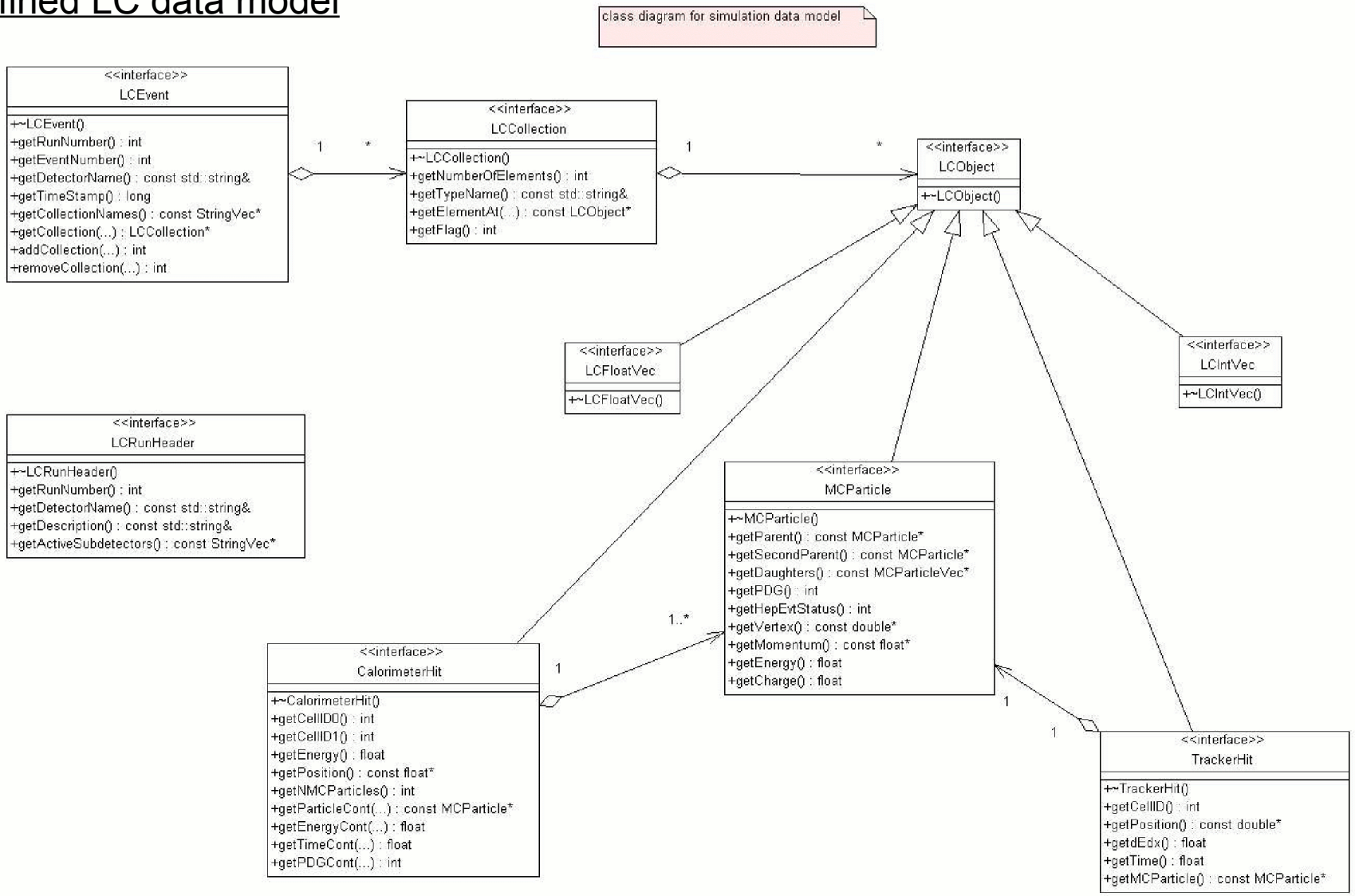


Figure 3: UML class diagram showing the complete data model for simulation output. Event data is stored in LCEvent objects that in turn hold untyped collections (LCCollection). The tagging interface LCObject is needed for C++ which doesn't have a common base class. Existing simulation applications can use LCIO as an output format by implementing the shown interface within their existing classes.

b) roobasf

pros:
- All basic functions are built in the prototype already
- Smooth extrapolation from existing BASF/Belle software
- Independent of the choice of data model
- New generation framework with true object I/O support by ROOT IO, to which GAUDI belongs as well.
- Built-in parallel processing support
  -> unified framework for DAQ, production and user analyses (as well as BASF).
- Being developed/supported inside SuperKEKB collab.

cons:
- Not yet used in SuperKEKB study although prototype is ready and being used for HSC.
- Scripting system of the prototype is poor. It will be replaced with Python-based system.