Marlin
Modular Analysis and Reconstruction for the Linear Collider

Frank Gaede, DESY
SLAC Simulation Meeting 2005
March 16-17 2005
Outline

- Introduction
- Design
- Implementation
- Status
- Ongoing work/ Users
- Summary & Outlook
Introduction

**Modular Analysis & Reconstruction for the Linear Collider**

- modular C++ application framework for the analysis and reconstruction of LCIO data
- uses LCIO as transient data model
- software modules called Processors
  (for similarity with org.lcsim – U.S Java based reconstruction)
- provides main program!
- provides simple user steering:
  - program flow (active processors)
  - user defined variables per processor and global
  - input/output files
Processors and the LCEvent

MyInput2.slcio
MyInput1.slcio
MyInput0.slcio

LCEvent

collection0

read and add collections

MyInput.slcio

marlin::main

Processor0
Processor1
Processor2
...
ProcessorN
OutputProcessor

MyInput2.slcio
MyInput1.slcio
MyInput0.slcio
Scope for Marlin

- Generator
  - Java, C++, Fortran
  - Geant3, Geant4

- Simulation
  - Java, C++, Fortran

- Reconstruction
  - Java, C++, Fortran

- Analysis
  - Java, C++, Fortran

Persistence Framework

Marlin

e.g. Mokka or real data!

geometry
marlin::Processor

- marlin::Processor: base class for all user processors
- provides hooks (callbacks) for user actions:
  - `init()`
    - called once at program start
    - use to initialize histograms, counters, etc.
  - `processRunHeader(LCRunHeader* run)`
    - called for bookkeeping – new run conditions?
  - `processEvent( LCEvent* evt)`
    - the working horse – this where the analysis takes place
  - `check( LCEvent* evt)`
    - optional method, e.g. for checkplots, consistency checks…
  - `end()
    - called once at end of job
Marlin core processors I

- **AIDAProcessor**: Histogram module
  - based on AIDA (Abstract Interface for Data Analysis)
  - AIDA implementations, e.g. JAIDA/ AIDAJNI, OpenScientist,…
  - easy to create histograms, clouds and n-tuples
  - one folder per processor
  - compressed xml-data files (any AIDA tool)
  - can use JAS3 to view histograms
  - root files (need OpenScientist)
  - does not depend on root but can use root to view histograms
  - NB: users not required to use AIDAProcessor

- **OutputProcessor**
  - simply writes out the current event (no user code needed)
Conditions Processor (new)
- based on LCCD (see next talk)
- provides transparent access to conditions data
  - from LCIO files
  - conditions database (CondDBMySQL)
- conditions data attached to LCEvent as collections of LCGenericObjects
- code independent of actual conditions format

OverlayProcessor
- under development
- will provide event mixing feature (background)
How to write your own processors

- inherit from marlin::Processor
- implement callbacks, e.g. init(), processEvent()
- register processor parameters
  - with name, description and default value
- edit steering file (see next slide)
  - use ‘MyMarlin –l’ for help:
    - prints all available processors with example steering
- rest is done by the framework!
- example template exists in Marlin cvs
  - $MARLIN/examples/mymarlin
    -> see tutorial on DVD
  “LC Simulation Software, Status 2/2005”
Marlin steering files

- global parameters, e.g.
  - LCIOInputFiles – the files to read in
  - ActiveProcessors – define which processors to run
  - SupressCheck – don’t call check()
  - MaxRecordNumber

- processor specific parameters:
  - ProcessorType
  - parameters registered with the processor
    - named int, float and string variables or vectors
    - e.g.: InputCollectionName, CutValue, AlgorithmType

→ A Marlin application is fully configured through the steering file (no code change needed)!
Marlin status

- v00-08 released
- ConditionsProcessor
- available via cvs web @
- improved Makefiles
- improved Processorparameters
- new homepage: http://ilcsoft.desy.de/marlin
  (old download page obsolete!)
- improved documentation
- overview & API doc
Marlin ToDo-List

- error handling
- log files
- error/warning messages
- naming convention for common parameters, e.g. InputCollectionName, OutputCollectionName
- convention for passing user data between Processors, e.g.:
  - as LCCollections of LCOBJECTS
  - as global objects (singletons) ?
- some logic to control execution and I/O of events
  - e.g. a Processor might want to decide that the event is not worth processing
    then the rest of the Processors should not be called …
  - under development (J.Samson)
- additional functionality ?
- user feedback needed !
Marlin users

- CALICE testbeam software
  - Digitization (G. Lima)
  - Ganging and Calibration (R. Poeschl)
- Analysis software
  - LCLeptonFinder (J. Samson)
  - JetFinder (Th. Kuhl)
  - Thrust Finder (Th. Kraemer)
- Reconstruction software
  - Wrapper for Brahms-Tracking code (S. Aplin)
  - clustering algorithms (Ch. Ainsley, G. Mavromanolakis)
  - clustering and pflow – SNARK in C++ (A. Raspereza)
- probably others ...
- aim: have (at least one) complete set for standard reconstruction in C++ soon!
- need common repository or web portal to provide entry point for users to download and configure their marlin application!
Summary & Outlook

- Marlin is a simple C++ application framework for reconstruction and analysis of ILC data
- based on LCIO (transient and persistent)
- provides simple base class for user Processors
- convenient steering for run control and user parameters
- no need to deal with I/O or write main program
- available at homepage: [http://ilcsoft.desy.de/marlin](http://ilcsoft.desy.de/marlin)

Future work

- improve Marlin (To-do list)
- investigate integration with org.lcsim
  - tricky: need generic Java/C++ interfacing on basis of LCIO...
- need web portal for Marlin processors

Marlin is meant as a starting point for a common modular C++ Reconstruction-Framework for the ILC. If you use this for your algorithm it can easily be shared with and compete against others.