Overlay - A Marlin Package for Event Merging

DESY Summer Student Programm 2007

Nicola Chiapolini
Supervisor: Frank Gaede

14.09.2007

Abstract

The task of this work was, to implement a mechanism for overlaying several events of International Linear Collider (ILC) collision data. The resulting Overlay Processor has been developed in the Marlin/LCIO software framework used by the European ILC community In addition a simple analysis program was developed. Although it did not reach a state where the Overlay Processor could be tested several problems with the existing analysis applications were found. Finnally as a by product of the development a tool to display the content of data files in a tree like pattern was implemented.

Contents

1 Motivation

The next big project in High Energy Physics is the planned International Linear Collider (ILC) that will collide electrons and positrons at a 500 GeV centre of mass energies. The European ILC community uses the Marlin C++ application framework for its analysis and reconstruction studies. Marlin is based on the LCIO persistency framework and data model.

Neither Marlin nor LCIO offered a native way to overlay events from different data sources, an ability frequently requested by the users as background and physics events are generated separately and need to be combined in order to study the effects of various backgrounds to physics events.

The Overlay Processor addresses this need and offers a simple way to merge collections based on different parameters.

2 Requirements

To address the needs of the ILC community, the solution had to be implemented as a processor for the Marlin framework. In contrast to other processors, the merge mechanism needs to alter existing data and therefor requires a forth, additional access mode to LCIO's three major ones (writing, reading and extending and read only).

As normally only simulated data will be processed, it initially sufficed to implement merging for few collection types, namely MCParticles, SimTrackerHit and SimCalorimeterHit. Collections of type TrackerHit and CalorimeterHit where added later too but nevertheless the implementation needed to be easily extendable to further collection types.

In addition it must be simple to merge collections or whole events based on different criteria and it should not require much work to implement additional criteria.

3 Implementation

To fullfill all the requirements above, the implementation is divided into two parts. The Merge Class consits of all the methods needed to merge two collections or events respectively and the Overlay Processor is used to configure the merge process based on settings given in the Marlin steering file.

3.1 Overlay Processor

The Overlay Processor is configured with the Marlin steering file. Five parameters can be set (see Table 1). At least the *InputFileNames* must be set, defining the background events that are to be merged with the physics events.

By default the processor merges all collections of the types MCParticles, SimTrackerHit, SimCalorimeterHit, TrackerHit and CalorimeterHit with the same collection names. Optionally, the collections that should be merged can be specified with *CollectionMap*.

This is needed if the Monte Carlo particles of the background processes should be stored separately from the particles of the physics processes or if the generators used different naming conventions.

The three remaining parameters allow to specify the number of background events added to each physics event. Note worthy here is the parameter *runOverlay*. Using this background following an arbitrary distribution can be added to the events. To do this, the background event need to be grouped into runs according to the distribution.

Table 1: The different parameters for the Overlay Processor that can be specified within the Marlin steering file

Parameter	**Type**	**Function**
InputFileNames	StringVec	The names (with absolute or relative pathes) of the files from which the background should be read. Multiple files can be given by a white spaces separated list or by setting this parameter multiple times. If the end of the last file is reached, before all events have been processed, a warning will be printed and reading restarted with the first file.
CollectionMap	StringVec	Pairs of collection names. The input collection (given first) will be merged into the output collection. If the output collection does not exist, it will be created. It is recommended to set this parameter once for each collection pair. If this parameter is not set, all collections with the same name and type will be merged.
NumberOverlayEvents	int	Fixed number of background events that should be added to each physics event.
expBG	double	If this value is set, a random number of background will be added to each physics event. The Random numbers will be thrown according to a Poisson distribution with this expectation value. If set, NumberOverlayEvents will be added to the random number.
runOverlay	bool	If true, NumberOverlayEvents and expBG will be ignored. Instead one run of background events will be added to each physics event.

3.2 Merger Class

The Merger Class provides the low-level functions to merge events and collections. The work horse of this class is the method merge(LCCollection$*$, LCCollection$*$). This is the most basic merge method and contains the actual implementation of the merge. To implement the merge for an additional collection type a new block needs to be added to the if-then sequence in this method. The if-sequence is used, as most of the collections can be handled by the same block and the implementation can thus be almost completely independent of the collection type.

In addition to the basic method explained above, the Merger class contains different wrapper methods with different lists of arguments. The Overly Processor calls either the merge(LCEvent$*$, LCEvent$*$) or merge(LCEvent$*$, LCEvent$*$, map$*$). The first method merges all the collections in two events with matching collection name and type. While the second method merges collections according to the map given as third parameter.

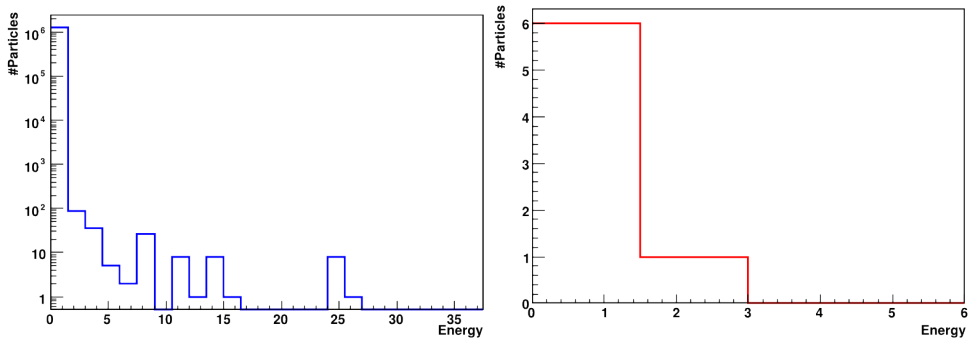
The remaining methods are provided for users who want to implement their own merging tool. In addition these users can easily add other wrappers methods to the Merger class. The existing methods and the algorithm of the Merger class are documented in the Appendix A.

4 Test Analysis

After the first version of the Overlay Processor was completed, a test analysis was planned up. This analysis had to overcome different obstacles. First of all the used analysis processor PandoraPFA seemed to not work correctly as there were no charged particles reconstructed by the analysis. As the charged particles were missing in the example analysis form the PandoraPFA package too, the bug seems to be on the side of the analysis package.

Instead of PandoraPFA a TrackBased reconstruction algorithm was therefore used. The first tries with 500 GeV centre of mass energy showed, that the tool is at the moment too slow for use at such high energies. As a result data files at Z-Pole energy were used instead. Analysing the reason for the time needed to analyse the data, hints for several memory leaks were found. This problem will need further investigation beyond the scope of this work.

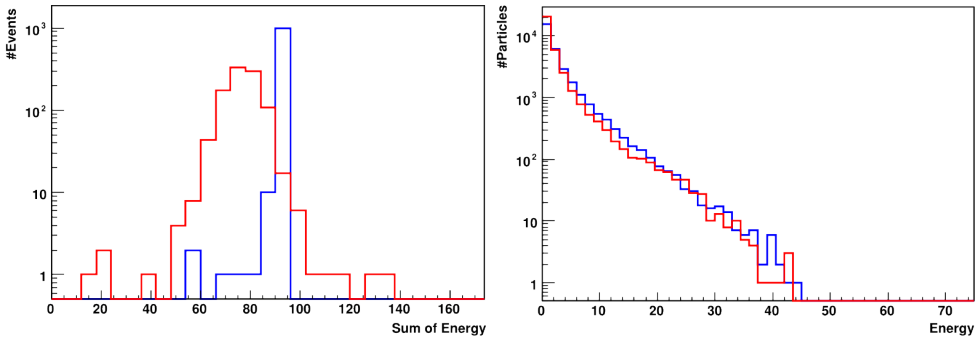
At first, $e^+ - e^-$-pair background at 500 GeV was used. Running the analysis on the original background files containing 100 background events with almost 3 million particles in total , only 7 particles were reconstructed. This background seems negligible and thus could not be used to test the Overlay Processor. Further studies with other background sources will have to be carried out.



(a) Simulated ($1.2 \cdot 10^6$ particles) (b) Reconstructed (7 particles)

Figure 1: Energy Spectrum of Background

The analysis of the ZPole events produced results in acceptable agreement with the expectations. Nevertheless further studies are required to understand the results and their reasons better but this was not possible anymore within the time frame of the summer students program.



(a) Total Energy in Events (b) Energy Spectrum of Particles

Figure 2: ZPole Analysis

5 LCIO Shell

While working on the Overlay Processor the content of the events and collections used for testing needed to be displayed and analysed repeatedly. As there existed no tool to print selected data from slcio files, this could only be done by adjusting a small program over and over again.

This prompted for a tool to display selected content of slcio files. This tool was then implemented as the LCIO Shell. The user can now browse and display the content of slcio files opened with the shell similar to the way a directory tree works (see Figure 3).

Figure 3: A Session in the LCIO Shell

```
Large file: not preparing map of events in runs!
24 Runs - ?? Events
OutputBG.slcio$ cd 1/5/7
OutputBG.slcio/run1/evt5/TrueTracks$ ls
name:     TrueTracks
type:     Track
elements:   0
OutputBG.slcio/run_1/evt_5/TrueTracks$ cd ..
OutputBG.slcio/run_1/evt_5$ ls
[ 0] ClustersFromTrackBasedPFlow  Cluster                0
[ 1] ECAL                         CalorimeterHit         0
[ 2] EMShowerCandidates           Cluster                0
[ 3] HCAL                         CalorimeterHit         0
[ 4] MCParticle                   MCParticle             1
[ 5] RecoParticlesTrackBasedPFlow ReconstructedParticle  0
[ 6] RelationCaloHit              LCRelation             0
[ 7] TrueTracks                   Track                  0
[ 8] TrueTracksToMCP              LCRelation             0
OutputBG.slcio/run_1/evt_5$
```

What started as a quick and dirty hack, had by the end of the project developed into a utility with many functions. Figure Figure 4 shows the output of the internal help, explaining the different commands. Special abilities include the dumping of events and collections using the functions provided by `LCTOOLS`, interrupting large outputs using [CTRL]+[C] and redirecting output to a file and subsequently opening the file with a pager.

Figure 4: Internal Help of the LCIO Shell

```
COMMANDS:
   cd | cl <number of OBJECT|..>    change into OBJECT | leave object
                                    multiple levels can be given at once
                                      e.g.:  file.slcio/run1$ cd ../3/1
   ls                               list elements on next level | list
                                    content of active collection
   dump [-d]                        dump data of active level; -d triggers
                                    detailed dump of event
   print | cat [-s] <collection nr> print content of collection given;
                                    -s triggers short print-out

   open <filename>                  open new file
   exit | quit                      exit program
   help                             print this help text
   pager <command>                  use pager <command> when paging output

 REDIRECTION:
   If '|' or '>' is appended to the a command, the output will be
   redirected to a file and then displayed with the pager (see above).
   If a string is given after the redirect token, it is used as filename
   and the file is stored. (existing files will get overwritten without
   prompt). If no filename is given, a temporary file will be used for
   paging.
      e.g.:   file.slcio/run1/evt1$ dump -d > event1.txt
```

5.1 Redirection

The redirection of the output into a file was a technically interesting task. C++ uses the streambuffer `cout` for its output and a stream can be redirected using its `rdbuf()` method. However this failed due to the use of `printf()` within the dump functions. `printf()` is not sent through the streambuffer but to the system level output directly.

It was therefor necessary to redirect the output on system level. This is done by exchanging the file descriptor of `stdout (1)` by the one of the tempfile (`fd_temp`).

```
dup2(1, fd_old);    // backing up old fd pointed to by stdout (1 = stdout)
dup2(fd_temp, 1);   // changing the fd pointed to by stdout
```

After the redirected command is finished the file descriptor will then be changed back so that the output is printed to the shell again.

A Methods of the Merger Class

A.1 Core Method

void Merger::merge (LCCollection * *src*, LCCollection * *dest*) `[static]`

merge function, takes two collections and adds the elements from src to dest. Both collections need to have same type!
For `MCPARTICLE, SIMTRACKERHIT, TRACKERHIT`: All Hits from the source collection are copied into the destination collection.
For `SIMCALORIMETERHIT, CALORIMETERHIT`: If the destination collection contains a hit with the same cellID, the energy of the source hit will be added to it. Otherwiese the hit will be copied into the destination collection. (In case of simulated data the MCParticle contributions will be preserved.)

It is the callers responsability to make sure the mcParticles pointed to by the hits do exist!

Parameters:

src Collection containing the entries that should be added to another collection.

src Collection to which the new entries should be added.

A.2 Methods used by Overlay

void Merger::merge (LCEvent * *srcEvent*, LCEvent * *destEvent*) `[static]`

Tries to merge collections with a name present in both events.

Parameters:

srcEvent source event.

destEvent destination event

calls **merge(LCCollection*, LCCollection*)** (p. 7) internally

void Merger::merge (LCEvent * *srcEvent*, LCEvent * *destEvent*, map< string, string > * *mergeMap*) `[static]`

Merges the collections of the two events according to a given map

Parameters:

srcEvent source event.

destEvent destination event

**mergeMap* Map containing the src->dest association for the collection names
Map structure: (srcColName, destColName)
If srcCol does not exist, the pair will be ignored, if destCol does not exist, a new collection with the same type as srcCol will be created.

calls **merge(LCCollection*, LCCollection*)** (p. 7) internally

A.3 Additional Wrapper Methods

void Merger::mergeMC (LCEvent * *srcEvent*, LCEvent * *destEvent*, string *mcDestString*) `[static]`

Tries to merge collections with a name present in both events (like **merge(LCEvent*, LCEvent*)** (p. 7) but the MC particle collection in srcEvent is merged with the collection named mcDestString.

Parameters:

srcEvent source event.

destEvent destination event

mcDestString name of the collection that the MCPARTICLE collection should be merged with. If more then one collection of type MCPARTICLE exists in srcEvent the function exits without any action.

calls **mergeMC(LCEvent*, string, LCEvent*, string)** (p. 8) internally

void Merger::mergeMC (LCEvent * *srcEvent*, string *mcSrcString*, LCEvent * *destEvent*, string *mcDestString*) `[static]`

merge function, takes two events and tries to merge collections with a name present in both events.

Parameters:

srcEvent source event.

destEvent destination event

mcSrcString The MCParticle collection in srcEvent

mcDestString The MC particle collection the source particles should be added to. If this collection does not exist, a new collection is created.

calls **merge(LCEvent*, LCEvent*)** (p. 7) internally (after merging the MC collections and removing mcSrcString from the srcEvent)

void Merger::merge (LCEvent * *srcEvent*, string *srcString*, LCEvent * *destEvent*, string *destString*) `[static]`

Merges the two named collections in the given events

Parameters:

srcEvent source event.

srcString name of the source collection

destEvent destination event

destString name of the destination collection

calls **merge(LCCollection*, LCCollection*)** (p. 7) internally