LCG MCDB and HepML,
next step to unified interfaces of Monte-Carlo simulation

ACAT 2008, Erice (Sicily), Italy
04.11.2008

Sergey Belov, JINR, Dubna
belov@jinr.ru

on behalf of LCG MCDB group:

S. Belov, JINR
L. Dudko, SINP MSU
A. Ribon, CERN
A. Sherstnev, Univ. of Oxford

http://mcdb.cern.ch

Outline

- **Few tasks in Monte-Carlo simulation in HEP**
  - data reliability, correctness and availability
  - passing data through the simulation chain (formats, standards)
  - metadata handling and persistence (models, processes, cuts, etc.)
- **LCG Monte-Carlo Data Base (MCDB)**
  - provides storage, bookkeeping and access to MC simulated events
  - numerous ways to access both MC data and metadata
  - way to automatize passing data from ME to SH generators
- **HepML**
  - schemes and libraries; automatic documentation of MC samples
  - usage inside LHEF-formatted event samples
- **LCG MCDB + HepML + LHEF**
- **Summary**

Few tasks in Monte-Carlo simulation

- **Data reliability, correctness and availability**
  - verified reliable Monte-Carlo data
  - comprehensible detailed description (for the future use)
  - public available data (simulated events, models, cuts, etc.)
- **Passing data through the simulation chain (formats, standards)**
  - different data formats (now less problems with LHEF introduction)
  - need to have additional information along with Monte-Carlo events
- **Metadata handling and persistence (models, processes, cuts, etc.)**
  - unified way to describe and store these objects
  - re-usability of such descriptions
  - good place to have it – with Monte-Carlo events

MCDB project overview

- **MCDB is an GENSER subproject within LCG Application Area Simulation project**
  - main activity is to provide *configuration, book-keeping, documentation and storage* for shared Monte-Carlo event files
  - to *keep track of the full generation chain*, exploiting the competences of Monte Carlo experts and Monte-Carlo authors
- **Main content of LCG MCDB**
  - *shared* Monte-Carlo event samples
  - *detailed descriptions* (articles) of such event samples

MCDB prerequisites

- Some useful physical generators provide *time-consuming event generation*
- To be properly used and tuned, some generators *require expert knowledge*
- Often different groups of physicists need *the same Monte-Carlo data*
- Monte-Carlo *simulation chain could be automated* (to work with minimal human involvement)

centralized storage to keep correct and well-documented Monte-Carlo generated files is useful

Overview of MCDB as Knowledge Base

- **The major tasks of LCG MCDB**
  - share sophisticated Monte-Carlo generated samples between different groups of physicists
    - samples prepared by experts in MC generation
    - resource-intensive samples (human and/or CPU resources)
  - provide infrastructure to keep Monte-Carlo samples and sample documentation
  - facilitate communication between MC experts and users in LHC collaborations
- **The main content of LCG MCDB**
  - shared Monte-Carlo event samples
  - detailed descriptions (articles) of such event samples

MCDB description is published in CPC (*hep-ph/0703287*)

The major features of LCG MCDB

- Powerful Web-interface with content management system for the authors of Monte-Carlo event samples and end-users
- Handy site navigation and powerful search engine
- Flexible and reliable authorization system (to manage MCDB content)
- Free access to MCDB content for everybody
- Very structured Monte-Carlo event samples documentation
- Full backup of the samples and their description

The major features of LCG MCDB (cont.)

- CERN Advanced STORage (CASTOR) is the native storage for event samples
- Direct files uploading to MCDB from Grid as well as CERN AFS and CASTOR
- Direct access to MCDB samples from Grid
- Application Programming Interface (API) for the LHC collaborations environment software (C++)
- Optional LHEF/HepML unification of event files format and automation of sample processing (LHEF: ***hep-ph/0609017***)

Current content of LCG MCDB

- **More than 20 registered MC experts**
- **Near 90 articles with event samples e.g.:**
  - Top quark production (19 articles)
  - Higgs production (20)
  - Gauge bosons (42)
- **About 300 event samples generated by**
  - ALPGEN
  - CompHEP
  - GlauberXS
  - MadGraph
  - HIJING
  - ISAJET
  - …

MCDB is a deployed system, as well as software package

HepML

- **Unified XML format of MC event files metadata**
  - to store all possible information from MC generators in XML view
  - to store generator input parameters and setup
  - an effort to state a unified extensible way of MC events description
  - is an allowed part of LHEF standard event file header
- **Main purposes**
  - to unify MC event files description (parton and particle levels of Monte-Carlo simulation)
  - to facilitate passing information from Matrix Element generators to Shower generators
  - usage in MC generators tuning and testing
- **Contributors**
  - CEDAR *http://www.cedar.ac.uk*
  - LCG MCDB *http://mcdb.cern.ch*
- **Homepage** *https://twiki.cern.ch/twiki/bin/view/Main/HepML*

Main in ME event sample description

- **General information**
  - title
  - abstract
  - authors
  - experiment and/or groups
- **Physics process**
  - initial state
  - final state
  - QCD scale
  - process PDF
- **Event files**
  - physics process/subprocesses
  - events number
  - cross section and uncertainty
  - file name, location(s)
- **Used generator**
  - name and version
  - description
  - home page
- **Theoretical model**
  - name
  - description
  - set of parameters and their values with author's description
- **Applied cuts**

HepML inside LHEF

- J. Alwall et al., A standard format for Les Houches Event Files (2006) (*hep-ph/0609017*)
- structure:

```
<LesHouchesEvents version="1.0">
  <header>
    <hepml>
        <!--
          HepML sample description here
        -->
    </hepml>
  </header>
  <init> ... </init>
  <event> ... </event>
  <event> ... </event>
  ........
</LesHouchesEvents>
```

Some C++ classes of *libhepml / libmcdb*

```
namespace mcdb
{
    class Article;
    class File;
    class Author;
    class Cut;
    class Generator;
    class Model;
    class Process;
    class Subprocess;
}
```

```
class Generator{
public:
   Generator();
   ~Generator();
   string& name();
   string& name(const string&);
   string& version();
   string& version(const string&);
   string& homepage();
   string& homepage(const string&);
private:
   ....
};

class Process{
public:
   Process();
   ~Process();
   string& initialState();
   string& initialState(const string&);
   string& finalState();
   string& finalState(const string&);
   string& factScale();
   string& factScale(const string&);
   string& renormScale();
   string& renormScale(const string&);
   string& pdf();
.....
```

```
class Model{
public:
   Model();
   ~Model();
   class ModelParameter;
   string& name();
   string& name(const string&);
   string& description();
   string& description(const string&);
   vector<ModelParameter>& parameters();
   vector<ModelParameter>&
   parameters(const vector<ModelParameter>&);

    class ModelParameter
    {
     public:
        ModelParameter();
      ~ModelParameter();
       string& name();
       string& name(const string&);
       string& value();
      string& value(const string&);
      private:
      string name_;
      string value_;
    };
     private:
```

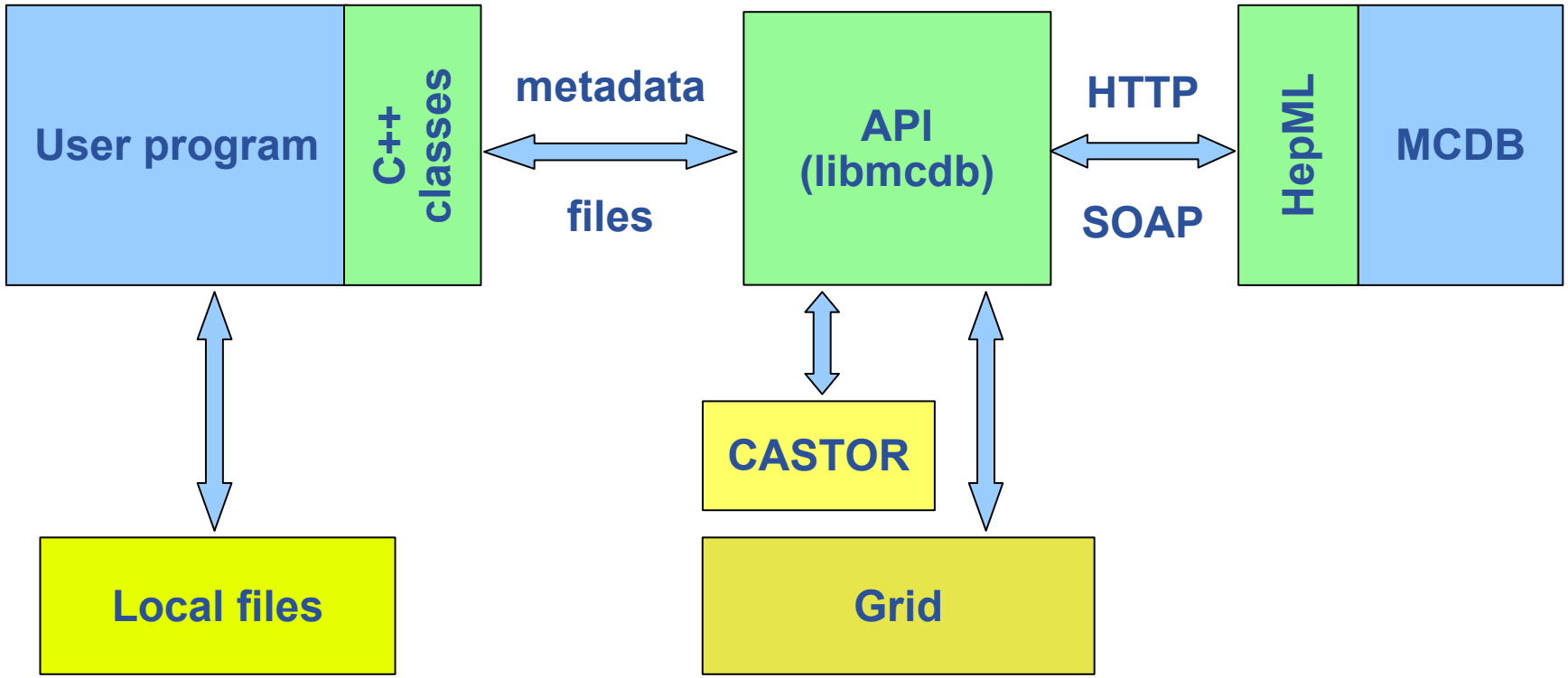
Some C functions of *libhepml*

```
void * init_lhaef_document (void);
int set_general_description(void * doc, lhaef_general_description * sample);
int set_file_description(void * doc, lhaef_file * file);
int set_generator(void * doc, lhaef_generator * gen);
int set_model(void * doc, lhaef_model * model);
int add_model_parameter(void * doc, lhaef_parameter * par);
int set_cutset(void * doc, int cutsetnumber);
int add_cut(void * doc, lhaef_cut * cut);
int set_author_record(void * doc);
int add_author(void * doc, lhaef_author * author);
int create_process(void * doc, lhaef_process * proc);
int add_process_beam(void * doc, int i, lhaef_beam * beam);
int add_process_beam_pdf(void * doc, int i, lhaef_pdf * pdf);
int add_process_beam_pdf_alphas(void * doc, lhaef_alphas * alphas);
int add_process_alphas(void * doc, lhaef_alphas * alphas);
int add_subprocess(void * doc, lhaef_subprocess * subproc, int cutsetnumber);
char * form_hepml_document(void * doc);
```

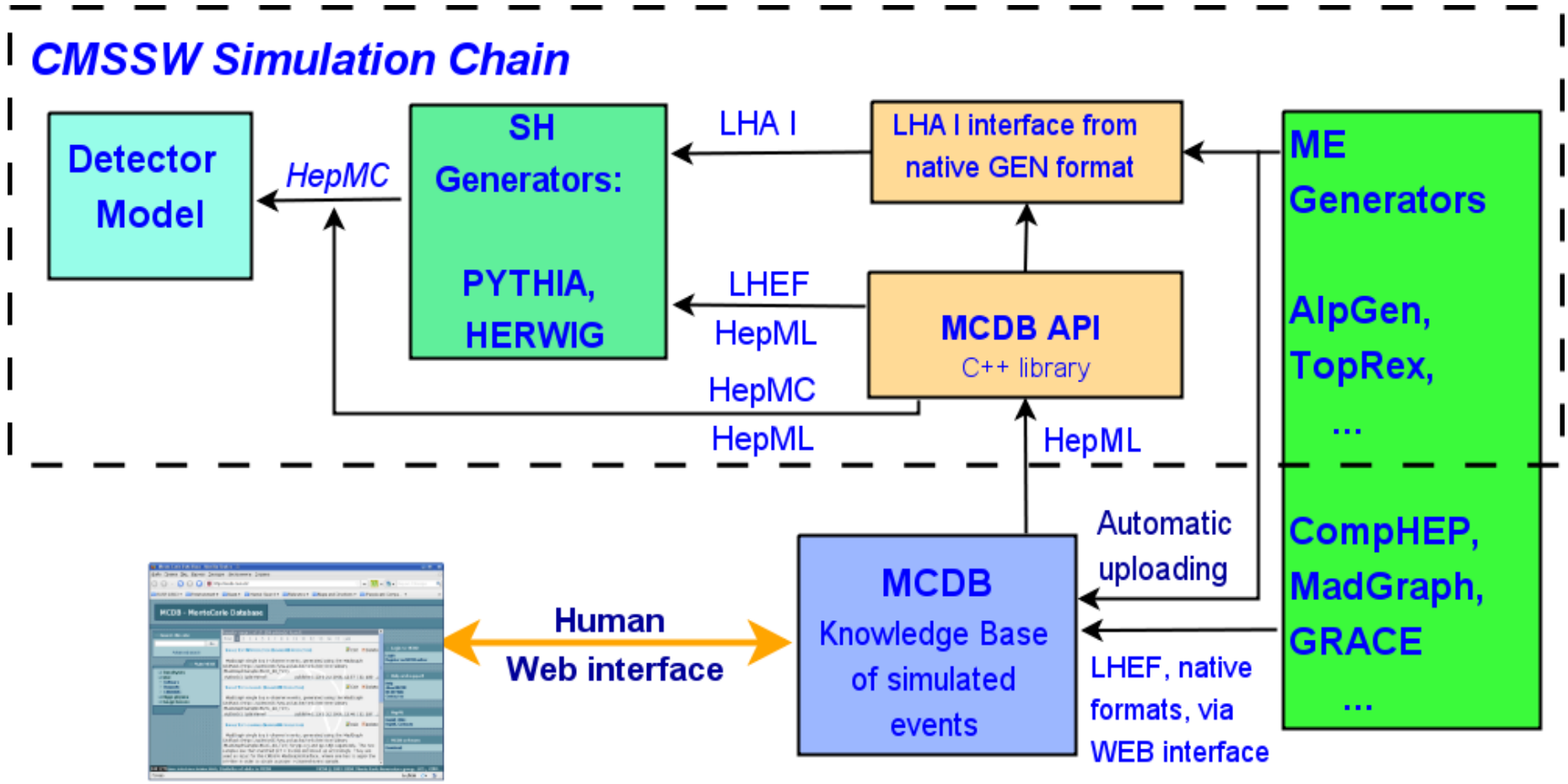
HepML current usage

- **in CompHEP**
  - generation metadata in HepML format as part of LHEF header (*self-documented event sample*)
  - event samples mixing using information from HepML block
- **in MCDB**
  - API to automated access of MCDB (upload, download event samples and its' descriptions in HepML form)
  - storage of MC event samples and HepML-compatible descriptions
- **in CMSSW**
  - access to MCDB content
- **libhepml – standalone library**
  - C/C++ interfaces (available)
  - writing HepML documents
  - parsing HepML

Software access to MCDB



Simulation chain: CMSSW example



Ways to access LGC MCDB

- **Upload data to the system**
  - via web-interface
  - from user/experiment software
    - using provided uploading script
    - upload files from command line
    - MadGraph header is supported, HepML one is upcoming
- **Get Monte-Carlo event samples from MCDB**
  - using web-site
  - query from program with API libraries
  - download samples from Grid (or from CERN CASTOR)
    - Monte-Carlo event samples could be easily populated and replicated in Grid
    - suitable for large-scale analysis

Possible way to unify a part of MC simulation

- **Matrix Element generator level**
  - produced events are in LHEF format
  - detailed description in HepML format inside LHEF header (*self-documented event sample*: process, model, cuts, etc.)
    - **libhepml** (C/C++) is available for doing that
  - event samples could be uploaded to MCDB
- **Shower generator level**
  - obtain metadata from MCDB, download event samples
    - **libmcdb** (C++) library could be used for this
  - input events in LHEF, description in HepML format
  - output events in HepMC
- **Detector simulation level**
  - input data – HepMC

LCG MCDB in collaborations

- Used in CMS collaboration as an official storage of a particular type for Monte-Carlo event samples
- Access to MCDB is supported in CMSSW
- In CompHEP to create self-documented Monte-Carlo event samples

- ✔ API to automated access of MCDB is already deployed (C++ libraries)
- ✔ Grid-ready
- ✔ All demands and suggestions are welcome

Summary

- LCG MCDB Knowledge Base is already in use
- APIs for MCDB and HepML are available
- Detailed documentation allows users and experts easily begin to work with LCG MCDB
- Using MCBD + HepML allows to make Monte-Carlo simulation more automatic and reliable

- Monte-Carlo experts are kindly invited to share their knowledge using of LCG MCDB
- Any feedback from Collaborations, Users and Experts will be appreciated

Thank you for your attention!

LCG MCDB project site:

http://mcdb.cern.ch

HepML project site:

https://twiki.cern.ch/twiki/bin/view/Main/HepML

libhepml, libmcdb, LCG MCDB package:

http://mcdb.cern.ch/distribution/