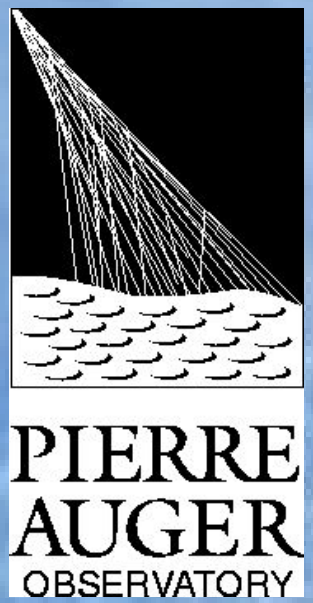
The Offline Framework of the Pierre Auger Observatory



Lukas Nellen (I. de Ciencias Nucleared, UNAM)

Stefano Argiro (University of Torino)

Thomas Paul (Northeastern University)

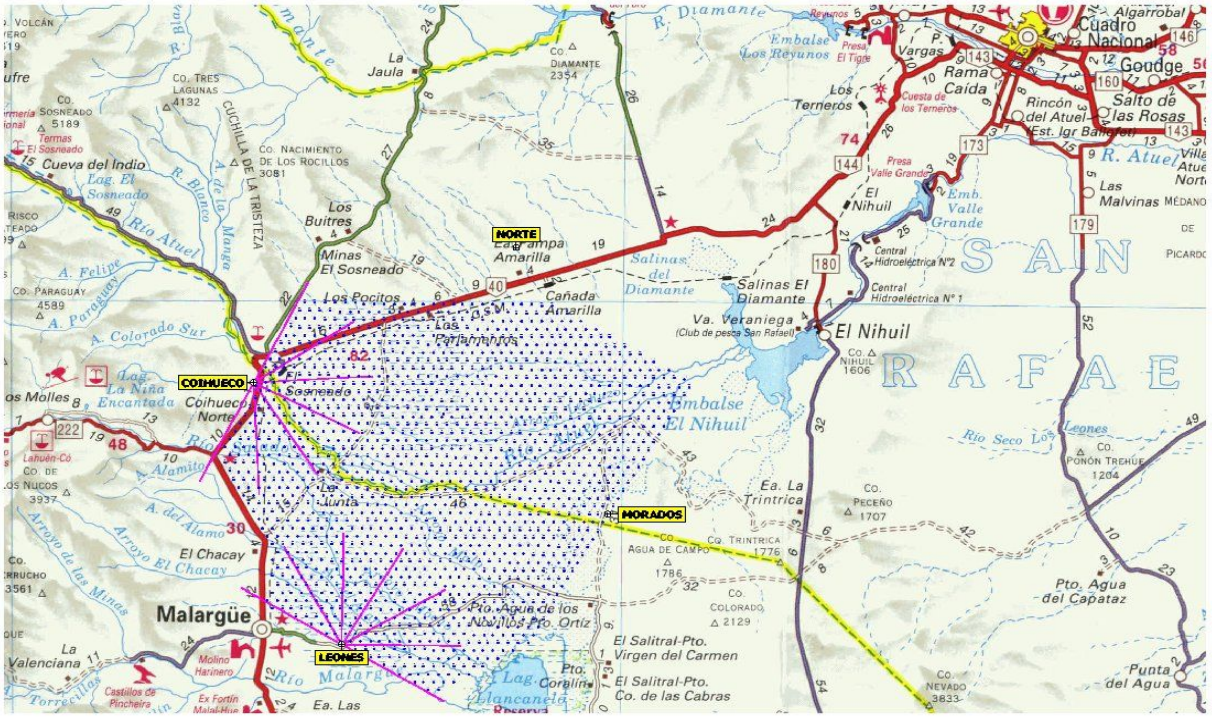
Troy Porter (Louisiana State University)

Luis Prado Jr (State University of Campinas)

27 September 2004, Interlaken, CHEP04

Auger – a reminder

- Hybrid cosmic ray experiment, designed to detect cosmic rays above 19^{10} eV
- Southern site located in Malargue, Mendoza, Argentina, just east of the Andes



Goals

- Provide the collaboration with a unified framework for data analysis, implemented in C++
- Support and encourage independent analysis and comparison of results
- Flexibility: event selection, simulation, analysis, ...

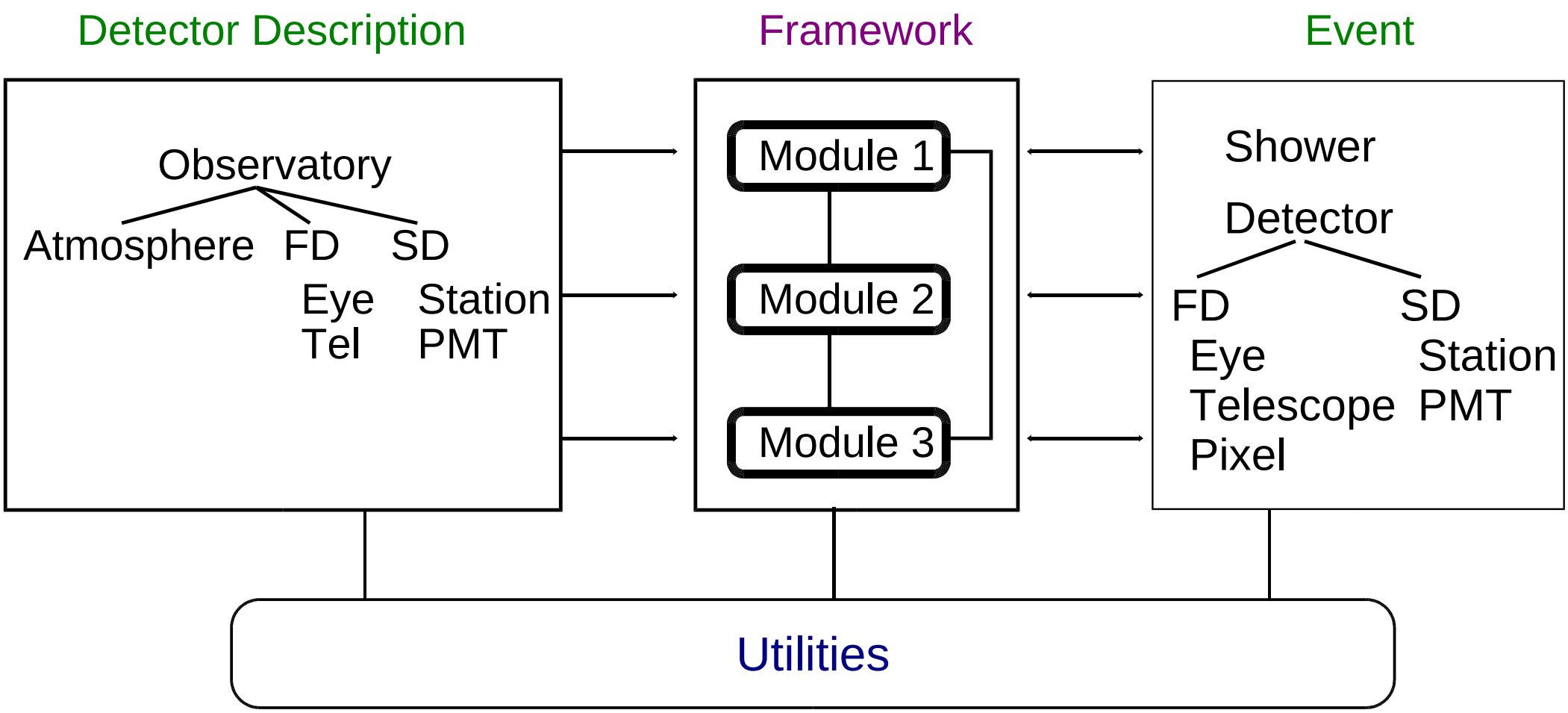
Design decisions

- Rely as much as possible on open, well supported standards:
  - C++, XML
  - Likely long-term support
  - Existence of support libraries and tools
  - Choice of libraries and tools
- Make sure we maintain the possibility to change libraries and tools
  - Development might cease or go into a undesired direction
  - The consequence is often a wrapper for components exposed to the end user

What is provided

- Provides unified, standardised access to Event and Detector information
- Provides a collection of utilities for:
  Error Logging, Geometry (see poster),
  Configuration Management, Math, Physics, ...
- Plug-in framework for modules
  - Physics modules (not part of the framework):
    Simulation, reconstruction, ...
  - Service modules:
    Event I/O, Event selection, Event display, ...
- Simple language for module sequencing

Components


Detector Description
Framework
Event
Observatory
Atmosphere
FD
SD
Eye
Tel
Station
PMT
Module 1
Module 2
Module 3
Shower
Detector
FD
Eye
Telescope
Pixel
SD
Station
PMT
Utilities

Configuration

- Uses hierarchy of XML files with Schema validation
- Links to sequence-,module- and detector configuration
- Configuration can be logged in XML
  - The log file contains all configuration information
  - Log file can be used as a configuration file for reproducing the conditions of a run
  - The log file can be displayed in any XML capable web browser

Configuration example

- Bootstrap information passed to executable:

```
userAugerOffline -b bootstrap.xml
```

```
\begin{verbatim}
<bootstrap>
  <centralConfig>
    <configLink
        id          = "ModuleSequence"
        type        = "XML"
        xlink:href = "./ModuleSequenceExample.xml" />

    <configLink
        id          = "MyModuleConfig"
        type        = "XML"
        xlink:href = "./MyModuleConfig.xml" />
  </centralConfig>
</bootstrap>
```

Schema verification

Fragment of a configuration file:

```
<WaterRIndex>                  1.33   </WaterRIndex>
<dEdXMuon unit="GeV/m">        0.2    </dEdXMuon>
<PhotonIntLength unit="m"> 0.06 0.07 0.08 0.10 0.12
</PhotonIntLength>
```

Gets checked against:

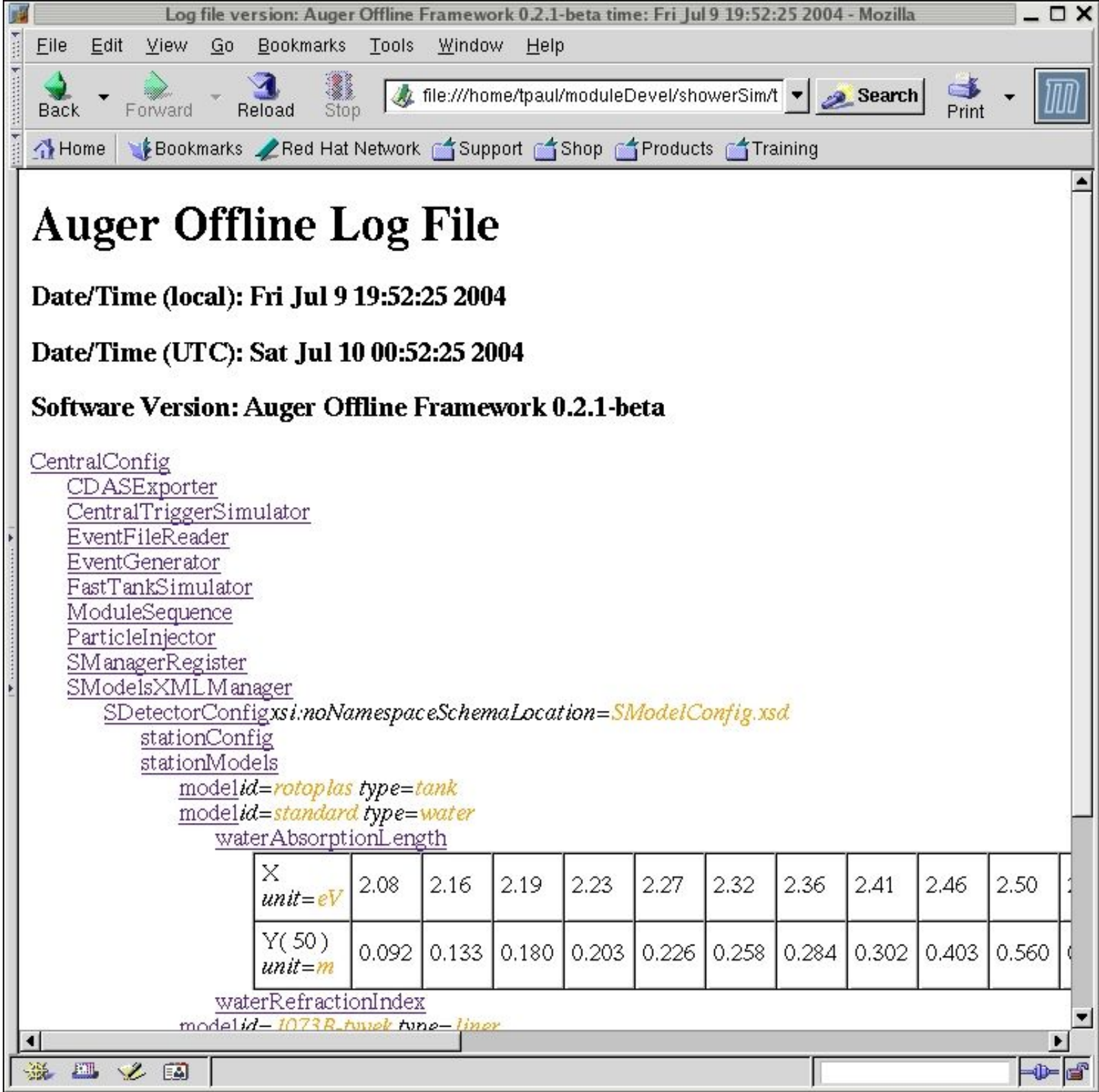
```
<xs:all>
   <xs:element name="WaterRIndex" type="xs:double" minOccurs="1"
               maxOccurs="1"/>
   <xs:element name="dEdXMuon" type="doubleWithUnit"/>
   <xe:element name="PhotonIntLength"
               type="listOfDoublesWithUnits"/>
</xs:all>
```

Error checking is comprehensive. Eg., following are errors:

```
<WaterRIndex>                1.33xs  <WaterRIndex>
<WaterRIndex unit="EeV">     1.33    </WaterRIndex>
<dEdXMuon unit="GeV/m">      0.2 0.3 </dEdXMuon>
<dEdXMuon>                   0.2     </dEdXMuon>
<PhotonIntLength unit="m">           </PhotonIntLength>
```

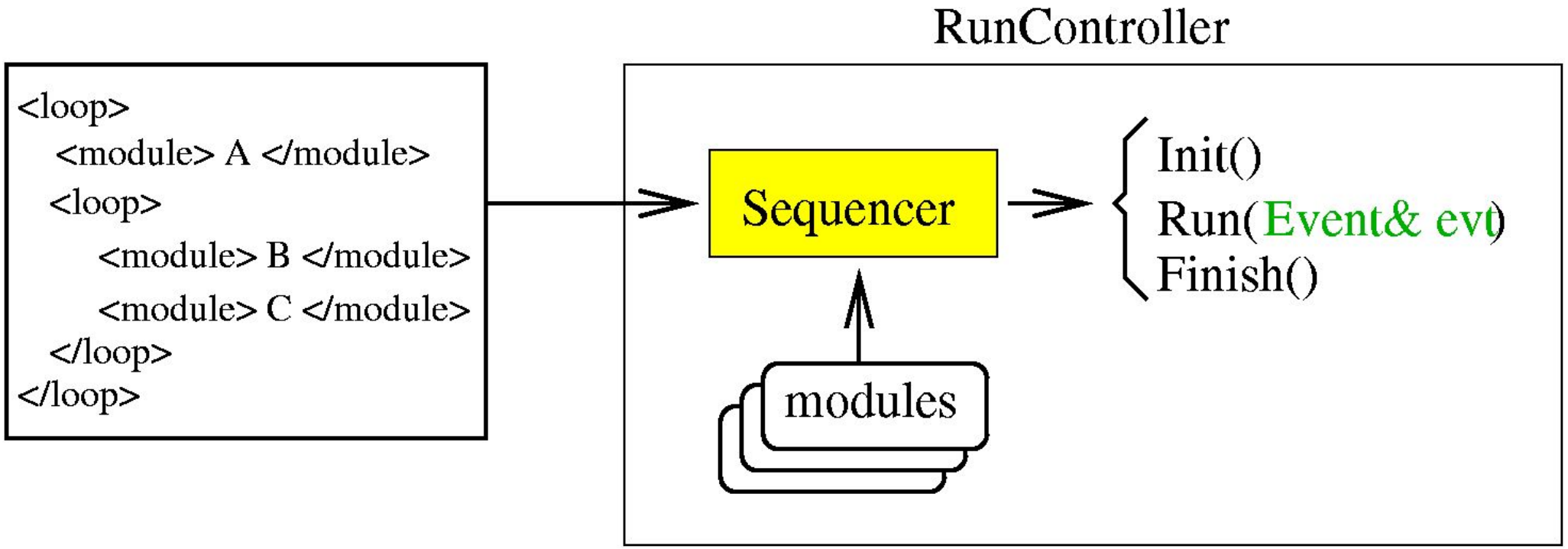
Off line.
Framework

Config log browsing



Module sequencing

- Run controller for sequencing
- Controls module initialisation and execution
- Simple, XML based sequencing language



Sequence examples

SD simulation example

```
<loop numTimes="unbounded">

  <module> EventFileReader </module>
  <loop numTimes="10" save="yes">
  <!-- Generate new event from Aires/Corsika
       data stored in Event -->
  <module> EventGenerator     </module>
  <module> Simulator          </module>
  <module> EventFileExporter  </module>
  </loop>
</loop>
```

FD reconstruction example

```
<loop numTimes="unbounded">

  <module> EventFileReader       </module>
  <module> GeometryFinder        </module>
  <module> ProfileFinder         </module>
  <module> EnergyFinder          </module>
  <loop numTimes="10">
  <!-- Iterative Cherenkov subtraction -->
    <module> CherenkovFinder     </module>
    <module> CherenkovSubtracter </module>
    <module> ProfileFinder       </module>
    <module> EnergyFinder        </module>
  </loop>
  <module> Analysis              </module>
</loop>
```

- **<module>** tag for selecting modules
- **<loop>** tag for simple looping control
- Return status of modules can affect sequence

Module Interface

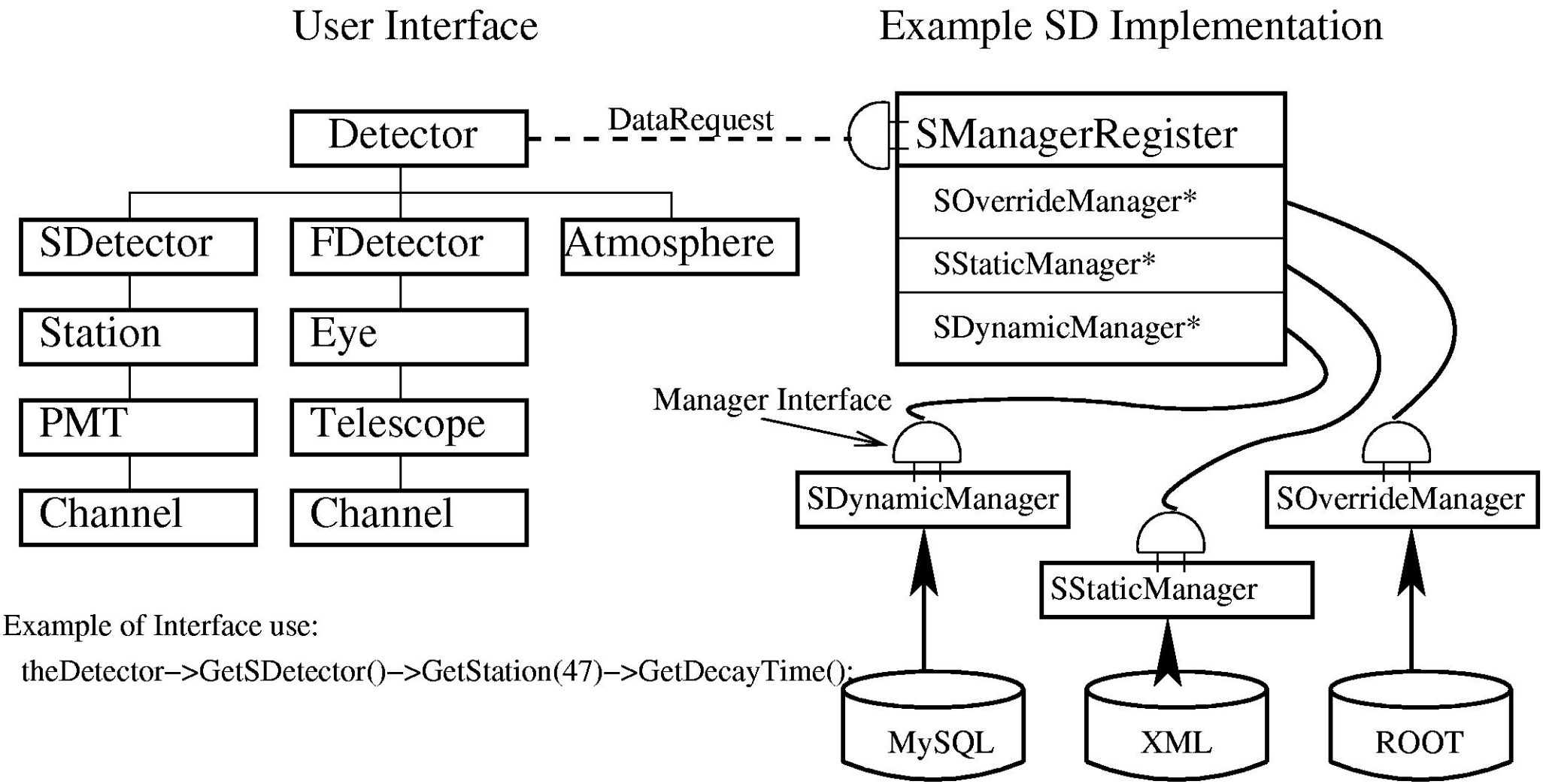
- Modules register with the Run Controller
- The Run Controller initialises, runs, and finalises
- Result affects sequence: ok, fail, break loop, continue

```
class MyModule : public fwk::VModule{

 public:

   MyModule();
   virtual ~MyModule();

   fwk::VModule::ResultFlag Init();
   fwk::VModule::ResultFlag Run(evt::Event& event);
   fwk::VModule::ResultFlag Finish();

 private :

   REGISTER_MODULE("MyModule",MyModule) ;
};
```

Key structures: Detector and Event

- Any type of simulation or reconstruction requires
  - Detector information
    - Changes slowly
    - Detector geometry and material data
    - Long-term calibration and monitoring information
  - Event data
    - Raw and calibrated data
    - Instantaneous calibration and monitoring information (when appropriate)
- Have a parallel structure, following the hardware hierarchy of the observatory

The Detector



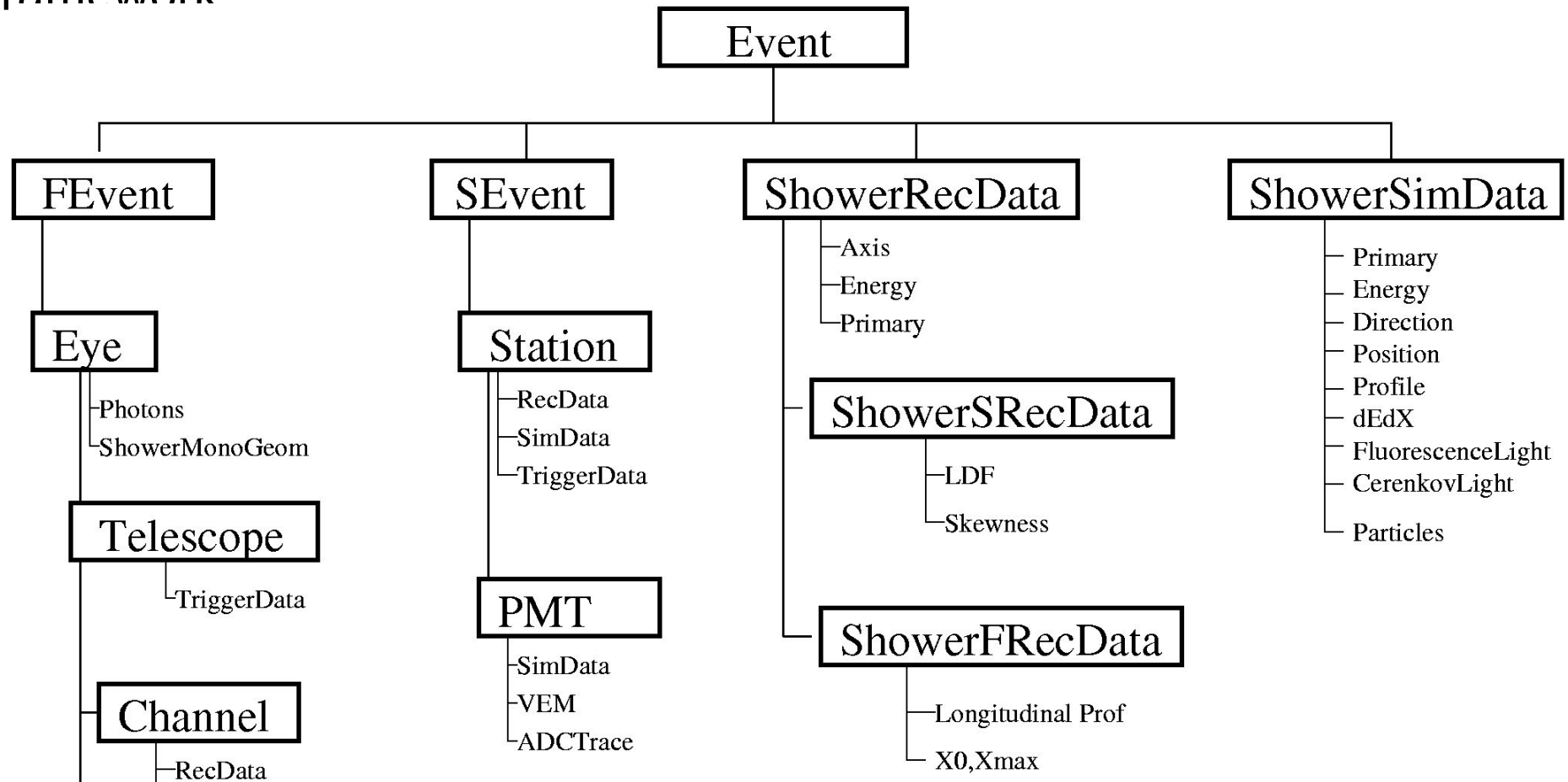
Example of Interface use:

```
theDetector->GetSDetector()->GetStation(47)->GetDecayTime();
```

The Detector (cont.)

- Provides a unified interface to multiple data sources implemented using managers for reading
  - XML
  - SQL
  - Others
- Reflects the hardware hierarchy
- Multiple sources for data: possibility to override data as required
- Data sources selected in the central configuration
- It is transparent to the code which sources are selected

The Event

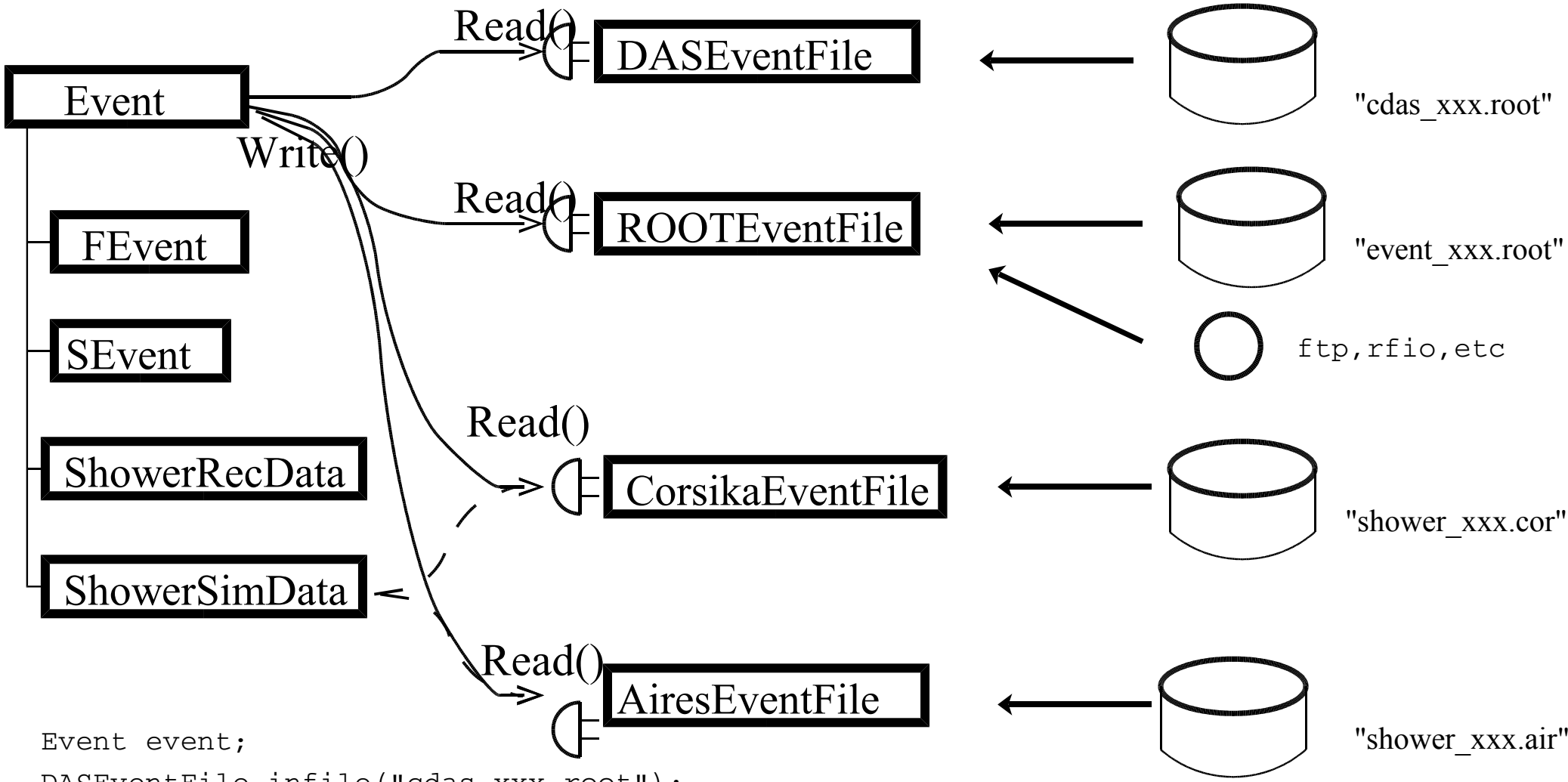



```
Telescope * tel = ...
Telescope::PixelIterator pix;
for (pix = tel->PixelsBegin(); pix!= tel->PixelsEnd(); pix++){
  // retrieve reconstructed photons
  TraceD * recphotons = (*pix)->GetRecData()->GetPhotonTrace();
}
```

The Event (cont.)

- Provides data access:
  - Raw data
  - Extra Monte Carlo data
  - Reconstructed data
- Hierarchy follows hardware
- Catches attempts to retrieve absent information
  ⇒ detect (some) errors in sequencing

Event IO




```
Event event;
DASEventFile infile("cdas_xxx.root");
infile.Read(1,&event);
ROOTEventFile outfile ("ftp://ccali.in2p3.fr/event_xxx.root", "w");
event.Write(outfile);
```

Event IO

- The Offline Framework provides interfaces to read data in different formats into the event
- A smaller set of format is writeable
- The native format stores the full information available in an event
- Streaming in native format is possible at any point during analysis and reconstruction. This way we cam implement partial processing, event selection, and the creation of summary tapes.

Dependences

- Collaboration internal:
  IO libraries
  - CDAS
    Surface detector
  - FDEventLib
    Fluorescence detector
  - AugerIo
    Complete raw event
- External
  - ROOT
  - Boost
  - Xerces
  - CLHEP
  - Aires
  - Geant 4 (opt)

Build system for the core

- Use GNU tools
  - autoconf
  - automake
  - libtool
- Standard open-software tools
- Resulting installation procedure is well-known to system administrators and many users
- Very flexible
- The burden of using the tools is placed on (few) developers, easing the life of the users

Build system for the user

- Provide some GNU makefiles in the examples
- Information on the local installation provided by
  `auger-offline-config`
  script

- Easy to adapt by users

Documentation

- The documentation of the software is generated using Doxygen
- A separate user manual provides additional explanations and examples
- Simple examples for a wide range of applications are provided
- We operate a Wiki web for sharing experiences and comments.

Development considerations

- The expected lifetime of the Pierre Auger Observatory is 20 years
- The developers are geographically dispersed on 3 continents – we have no central lab
- The offline core is divided in several sub-packages to ease loosely coupled development
- Use of support tools essential
  - CVS
  - Bug tracking
  - Wiki space
  - Communication: e-mail, video conference, meetings
- Coding standards and conventions matter

Quality control

- Unit test are developed and maintained together with the core software
- The execution of the tests is incorporated into the release process

- The collaboration is working on the physics verification procedure
- Parallel analysis teams are part of the physics verification

The Future

- The Pierre Auger Collaboration is moving its analysis efforts to the offline framework
- The framework is in a late beta testing phase
- There exists already a number of successful applications in the collaboration
- Partial validation is occurring continuously, procedures for full validation are being developed
- Graphical tools will be added once we have a full release

Event Browsing