Conditions Database Project
COOL Status and Plans

Andrea Valassi

(CERN IT-ADC)

Outline

- **Project Overview**
- **Current priorities**
  - Performance tests (this and the following two talks)
  - Data extraction and replication

Project scope

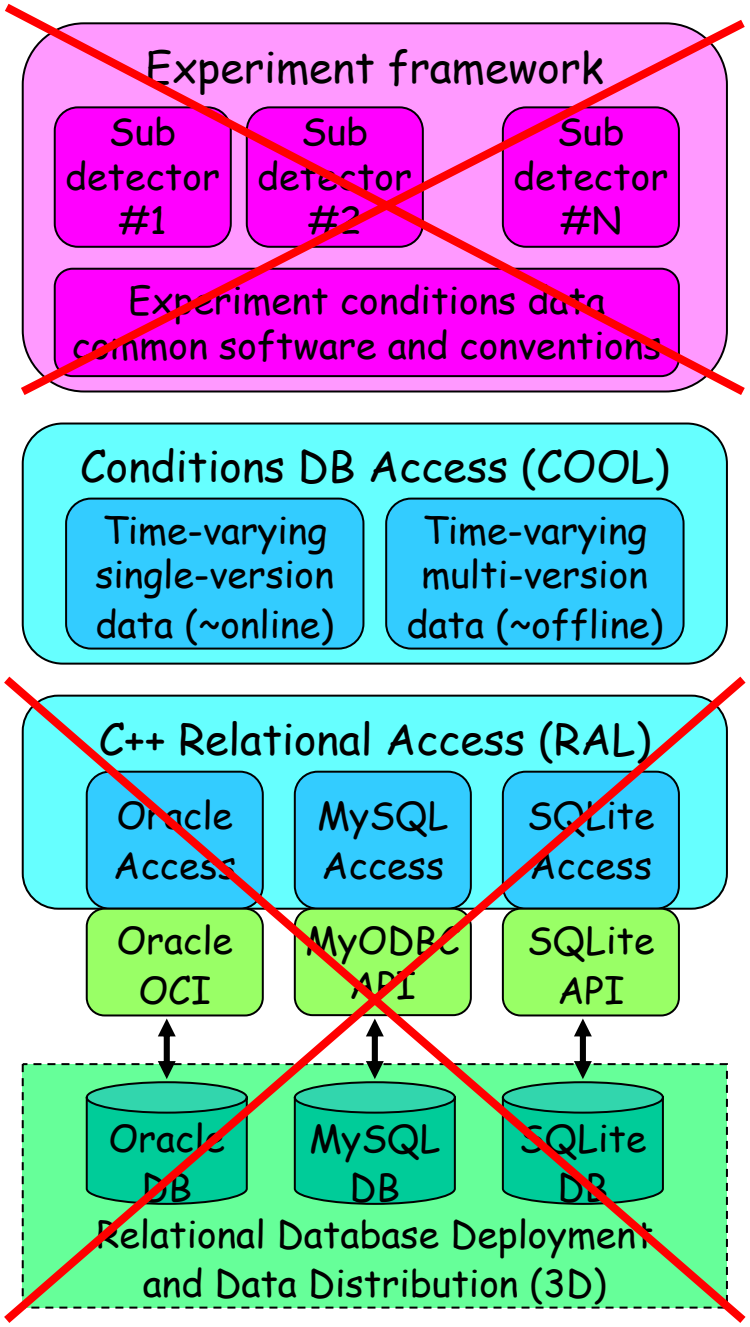
- **Conditions data: non-event data that vary with time**
  - And may also exist in different versions
- **Data producers from both online and offline**
  - *Geometry, readout, slow control, alignment, calibration…*
- **Data used for event processing and more**
  - Detector experts, alignment/calibration, event reconstruction/analysis…
- **Scope of the common project?**
  - Common software and tools for time-varying and versioned data
  - Not the generic access to distributed relational data (RAL, 3D…)
  - Not the problems specific to one experiment or one data type
  - Scope has grown larger in time (from offline to online use case)

COOL – documentation and organization

- **Project web page – http://lcgapp.cern.ch/project/CondDB**
- **Mailing lists and archives**
  - Developers (high traffic) - http://simba2.cern.ch/archive/project-lcg-peb-conditionsdb-developers
  - General (low traffic) - http://simba2.cern.ch/archive/project-lcg-peb-conditionsdb
- **Savannah page – https://savannah.cern.ch/projects/cool**
  - User support (feature requests)
  - User bug reports
  - Internal task list – heavily used
- **Weekly phone meetings (every Monday at 4.30 pm)**
  - 5 to 10 people on average
  - Minutes - http://lcgapp.cern.ch/project/CondDB/snapshot/minutes_list.html
- **DOxygen documentation – http://lcgapp.cern.ch/doxygen**
  - Feedback? Only two (positive!) comments so far
  - First draft of user guide is based on the same DOxygen sources

COOL - people

- **Core development and testing team (~2-3 FTEs for common development)**
  - A. Valassi (IT/ADC) - core development, SCRAM config, release management
  - S. A. Schmidt (Atlas) - core development, PyCool
  - M. Clemencic (LHCb) – SQLite, CLOB, PyCool, testing, CMT config, LHCb integration
  - D. Front (IT/LCG) - performance validation and stress tests
  - U. Moosbrugger (Atlas) – core development (from September 1st)
- **Contrib packages, experiment integration and other related work**
  - N. Barros, J. Cook, R. Hawkings, A. Perus, S. Roe, RD Schaffer, T. Wenaus, F. Zema…
    - Naming only the people who show up most often at the Monday phone meetings
  - S. Stonjek (Atlas) – distribution tests (presently working on Atlas/3D)
  - Y. Benhammou et al. (Atlas) – focus on TGC group needs
- **Collaboration with other LCG projects and teams**
  - POOL (relational abstraction layer (RAL), infrastructure (SCRAM)
  - 3D and IT-ADC-DP: database deployment, distribution, Oracle consultancy
  - SPI: infrastructure (CVS, tools, AFS…)
  - SEAL: plugins, int64, Time, dictionary (PyCool)…
- **There is still room for new people and contributions!...**
  - Contrib packages may later be integrated in the COOL release (e.g. PyCool)



Scope of the "common" project

NOT the problems specific to one experiment or one data type *(too specific: handled by each experiment)*

Common software tools for time-varying (and optionally versioned) data: a component with a well-defined task

(Note: COOL presently has only a relational implementation because this is what was asked for by the experiments, but non-relational implementations of the same API are possible)

NOT the generic C++ access, authentication and monitoring for relational data *(too generic: handled by RAL)*

NOT the generic deployment and distribution of relational data *(too generic: handled by 3D and IT-ADC)*

Original "common API"

- **Designed to handle data "objects" that**
  - Can be classified into ***independent data items***
  - ***VARY WITH TIME***
  - May have different ***versions*** (for given time and data item)

This 3-D metadata model is still valid!

A CondDBObject has
- Metadata:
  - Data item identifier
  - Interval-of-validity [since, until]
  - Version information
- Data "payload":
  - Actual data variables (temperatures, calibration parameters…)
  - Separate fields or encoded as a BLOB

A "CondDBObject"

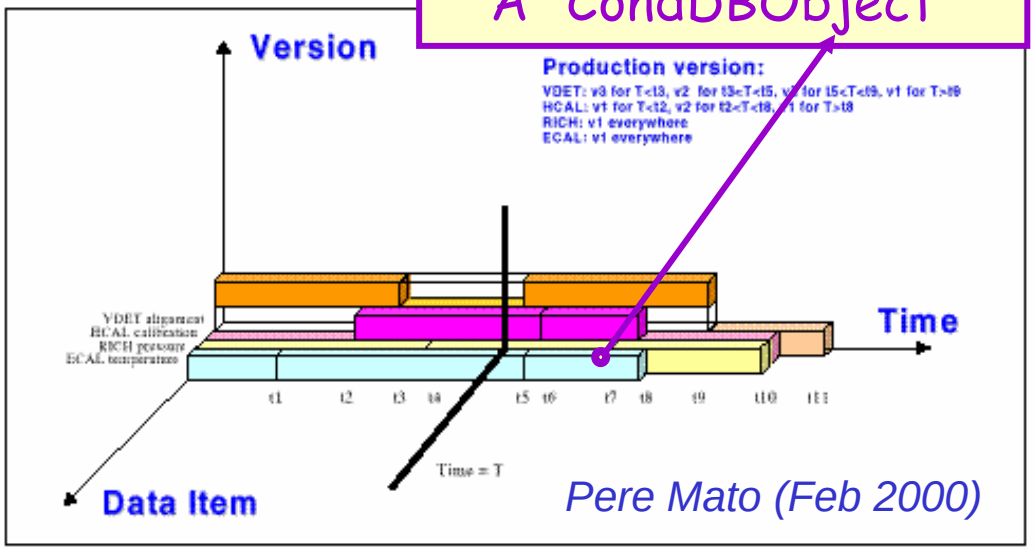


Figure 1 The three axes for identifying uniquely each data item in the condition database

- **Main use case: retrieve data payload valid at given time in given tag**
  - Inverse lookup (from temperature to time or version) not considered

COOL software overview

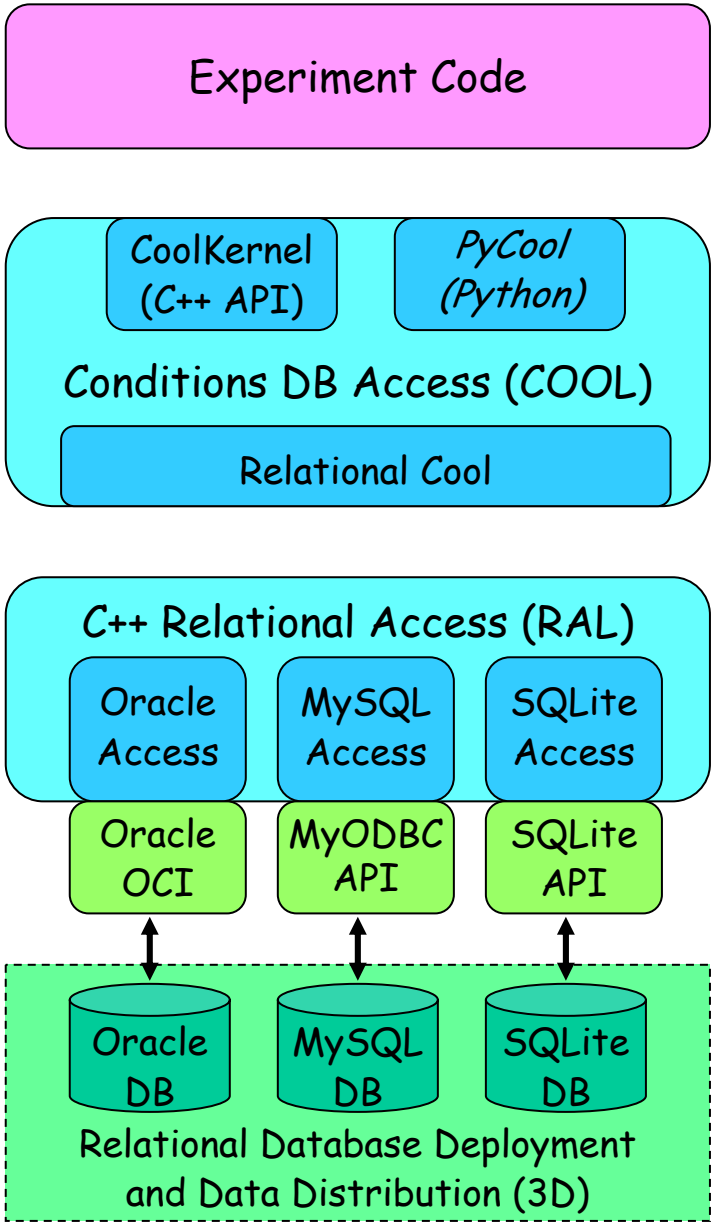
- **Maximize integration with and reuse of LCG software (SEAL/POOL)**
  - And merge the best ideas from the previous Oracle/MySQL implementations
- **Single implementation (same schema) for multiple back-ends using RAL**
  - User access encapsulated behind a C++ API (no direct SQL access)
  - Support for Oracle (main focus) as well as MySQL and SQLite
    - *Bulk operations and bind variables used whenever possible/appropriate*
  - Direct implementations or backend specific features may always come later
- **Modeling of Conditions database "objects" (e.g. a calibration set)**
  - System-managed common **"metadata"**
    - Data item id (e.g. calibration of module 1): "folder" (table) name + "channel" number
    - Interval of validity - IOV: since, until
    - Versioning information with handling of interval overlaps (at any given event time, there is never more than one object valid in a given "tagged version" of the data)
      - Two folder "types" for single-version (online) and multi-version (offline) objects
  - User-defined **"data payload"**
    - Support for simple C++ types modeled as pool::AttributeList
      - With a few additions to determine the storage precision (e.g. string: VARCHAR2 or CLOB?)
    - Different tables ("folders") for data items requiring different schemas

COOL – glossary

(does not cover all properties of each hierarchy entity)

- **Database Service (id: plugin label)**
  - Entry point into the COOL software
  - Software component to manage databases
- **Database (global id: URL)**
  - Entry point into one collection of COOL data
    - of a specific relational technology: Oracle or MySQL
  - 1 database has 1 root folder set (i.e. 1 hierarchy of folder sets and folders)
- **Folder set (id in database: UNIX-like path name)**
  - Contains folders or other folder sets
- **Folder (id in database: UNIX-like path name)**
  - Contains channels
  - Objects in different folders may have different „schemas" (payload specifications)
- **Channel (id in folder: channel number)**
  - Contains objects
  - Objects in different channels in the same folder have the same payload specification
- ***Object (lookup within a channel: by validity time AND version number or tag name)***
  - *Data payload with a specific interval of validity (IOV)*
- **Tag (id in database: tag name)**
  - Object collection that contains only 1 (or 0) object in any channel at any validity time

COOL Dependencies



POOL packages (later: CORAL)

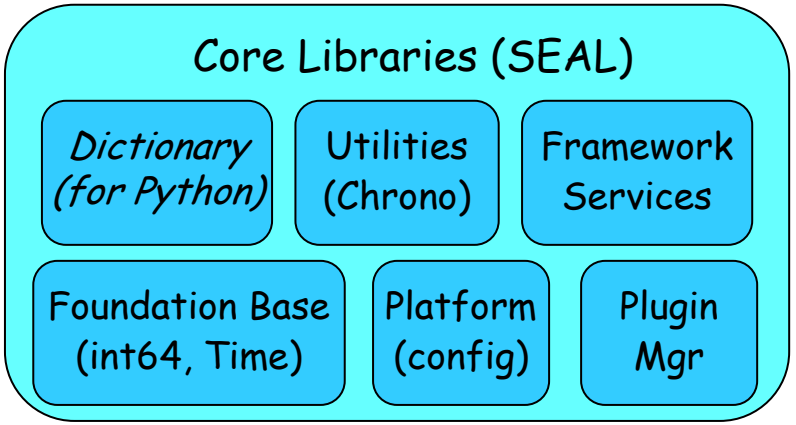
- AttributeList, POOLCore, RelationalAccess; Oracle/ODBC/SQLiteAccess. AuthenticationService

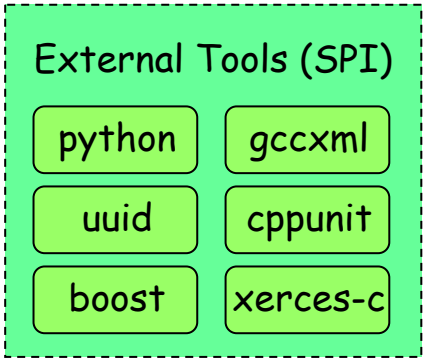
SEAL packages

- Foundation (SealPlatform, SealBase, SealUtil, PluginMgr, PluginChecker), Framework (SealKernel, SealServices), Dictionary (Reflection, *ReflectionBuilder, DictionaryGenerator*), *Scripting (PyLCGDict2)*

External tools

- sockets, pcre, uuid, boost, cppunit, xerces-c, *python, gccxml* (+ platform-dependent compilers)





COOL - evolution

- **Initial developments**
  - Oct-04: decision to develop new software from 2 existing packages
  - *Nov-04: start of COOL development*

- **Software releases**
  - COOL 1.0.0 (Apr-05): first public release
  - …
  - COOL 1.2.4 (Sep-05): current release

COOL – current priorities

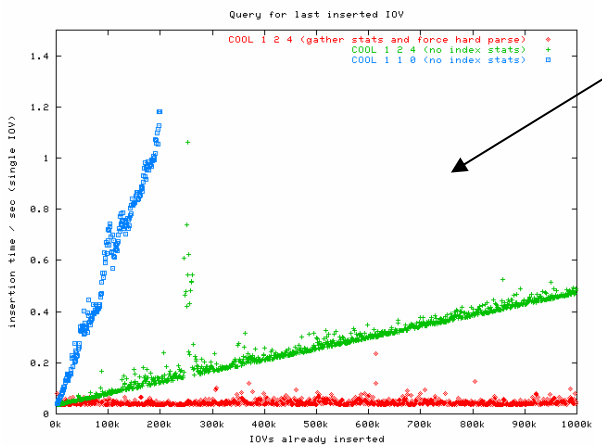
- **Performance validation for realistic experiment workload**
  - Service issues: assess whether database server is properly sized
  - Software issues: identify software bottlenecks (see details in next slide)
    - Also involves addition of a CHANNELS table for each folder
  - *Definition of "realistic" experiment workload still not entirely complete*

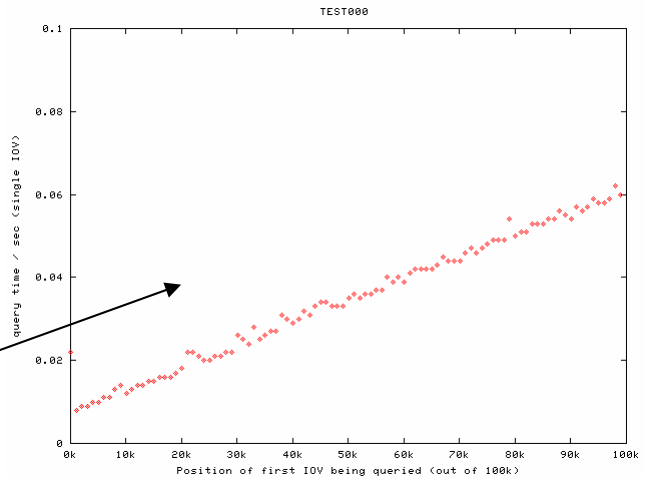
- **Data extraction and replication tools and tests**
  - Basic tools exist (see Sven's slides at the end)
  - At the COOL API level: need API to selectively clone data slices
    - Also start the design of a more distributed data model for a COOL "database"
  - At the back-end level: tests in the context of the 3D project
    - Replicate full databases (Dirk, Eva, Stefan…)

Have a look at the Savannah tasks page for more details

COOL performance – work in progress

- **Recently fixed: insertion time increase with #IOVs**
  - Just need to compute index statistics and force a hard parse





- **Open issue: retrieval time increase with #IOVs**
  - Design for reimplementation exists – next main priority in this area

- **Open issue: multi-channel bulk insertion**
  - Need RAL bulk update/delete to prototype proposed design

COOL performance – recommendations to the users

- **Do NOT artificially inflate data granularity!**
  - All data that have the same IOV should be grouped together and stored in a single "channel" as a single "object"
    - Also because of performance: the data insertion/retrieval rate to/from the database in #rows/second is as relevant as the rate in kilobytes/second…
  - Example: a detector has 100k tubes. The calibration of all tubes is done every day at midnight. One calibration parameter per tube is computed, but all tubes are always calibrated at the same time.
    - A single IOV should be used to describe all 100k tubes

- **Use bulk insertion and bulk retrieval whenever possible!**
  - Bulk retrieval is easily ~30 time faster than single row retrieval
  - Multi-channel bulk retrieval available since COOL_1_2_3
  - Multi-channel bulk insertion will be available in a future release

PyCoolCopy: Copying COOL Content

- PyCoolCopy is a python module that implements 'basic' replication
- Basic replication for COOL databases means:
  - Read only copy
  - Consistent 'logical' copies (as opposed to bitwise copies)
  - e.g. a MultiVersion folder will not have older versions of objects on the target unless they have been explicitly copied through a tag. (There's currently no COOL API to retrieve 'old' versions.)
- Thanks to RAL, this allows copying data between Oracle, MySQL, and Sqlite freely.
- PyCoolCopy does not have a real script interface yet (but is simple to add)
- However, it can be used for copying already:

```
>>> import PyCoolCopy
>>> source = 'sqlite://;schema=COOLTEST.db;dbname=COOLTEST'
>>> target = 'sqlite://;schema=REPLICA.db;dbname=REPLICA'
>>> PyCoolCopy.copy( source, target, '/f1' )
INFO:PyCoolCopy: Copying selection '{ nodeName : '/f1', channels : (0,4294967295), iov : [0,+inf],
tags :  }'
INFO:PyCoolCopy: source: sqlite://;schema=COOLTEST.db;dbname=COOLTEST
INFO:PyCoolCopy: target: sqlite://;schema=REPLICA.db;dbname=REPLICA
```

PyCoolCopy

- See
  https://uimon.cern.ch/twiki/bin/view/LCG/PyCoolUtilities
  for up to date information about PyCoolCopy (which is currently part of PyCoolUtilities)
- Replication beyond the basic replication of CoolCopy is planned but will require COOL API extensions
  - It would be *very* useful to learn about specific replication use cases
  - Where would basic replication not be enough?
- PyCoolCopy in python?
  - Python allows quick prototyping
  - First tests show that the performance overhead is not worryingly large (~30%–50%)
  - A C++ port is always an option and will profit from the python prototype

Some common feature requests

- Improved data item addressing – channels table (medium pr.)
- Tags across different folders – HVS (medium pr.)
- Payload queries – simplest functionality (low pr.), performance warning
- COOL as a configuration database – to be defined/understood
- Additional relational features – to be defined/understood

Summary

- **Latest production release COOL_1_2_4**
- **Deployment in Atlas has started (accounts on Atlas RAC)**
  - Waiting for feedback (on the software and on the deployment)
- **Current priorities**
  - Performance optimization
  - Replication and slicing API and tools