Report on the Conditions Database Workshop

(CERN 8-9 Dec. 2003)

Andrea Valassi

(CERN IT-DB)

http://lcgapp.cern.ch/project/CondDB/

Where the Workshop comes from

Feb.2000: "CondDB Interface Specification Proposal" by Pere Mato (LHCb

Feb.2000-Sep.2000: Requirement collection by Stefano Paoli (IT-DB)
- Emphasis on functional requirements and definition of common *C++ API*
- Active participation by many experiments (Harp, Compass, LHCb, Atlas…)
- Earlier experience in BaBar and RD45 taken into account

Oct.2000–Oct.2001: ***Objy implementation*** by Stefano Paoli et al. (IT-DB)
- Used for Harp data-taking in 2001-2002, evaluated for Compass data-taking

Mar.2002-Aug.2002: ***Oracle implementation*** by Emil Pilecki (IT-DB)
- Harp data migrated from Objy to Oracle in Nov. 2003 (keeping the same API)

Jun.2002-Dec.2003: ***MySQL implementation*** by Jorge Lima et al. (Atlas)
- More requirements collected from Atlas users, leading to API extensions
- *Used by Atlas for test beam data-taking since June 2003*

May 2003: "Proposal to bring CondDB into LCG AA" by Pere Mato
- LCG Conditions Database project launched within the Persistency Framework

Dec 2003: LCG Conditions Database Workshop at CERN

Workshop Agenda (1.5 days)

- Introduction and review of CERN 'common API' projects
  - Introduction (Dirk Duellmann) and Common API (A.V.)
  - Oracle implementation and tools (A.V.)
  - MySQL implementation and tools (Luis Pedro)
- Conditions DB projects at past/present experiments
  - Babar (Igor Gaponenko)
  - Harp (Ioannis Papadopoulos)
  - Compass (Damien Neyret)
  - CDF/D0 (Jack Cranshaw)
- Input from the LHC experiments (online and offline)
  - Atlas (Richard Hawkings, Joe Rothberg, Lorne Levinson, David Malon)
  - CMS (Frank Glege, Martin Liendl)
  - LHCb (Pere Mato, Clara Gaspar)
- Summary

http://agenda.cern.ch/fullAgenda.php?ida=a036470

The ConditionsDB in a Nutshell

The common API was designed to handle data "objects" that

- Can be classified into many *independent data items*
- ***VARY IN TIME***
- Can have many different ***versions*** (for a given time and data item)

A "CondDBObject"

A CondDBObject has

- Metadata:
  - Data item identifier
  - Start-of-time-validity
  - End-of-time-validity
  - Version number
- Data:
  - Actual condition data

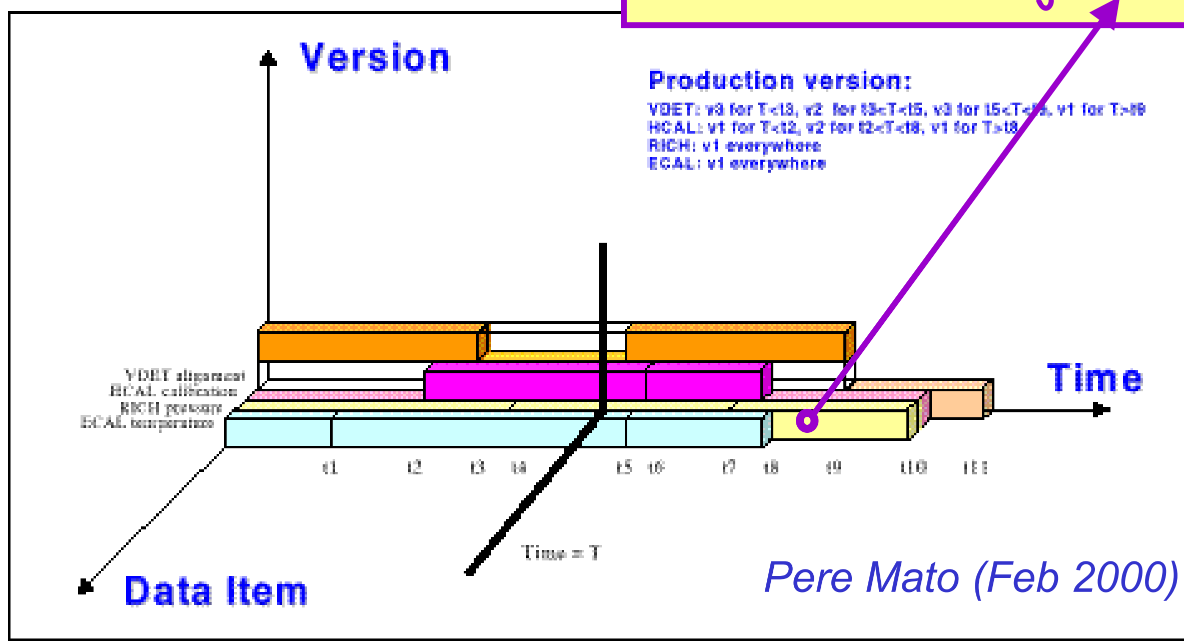

Figure 1 The three axes for identifying uniquely each data item in the condition database

Common API features

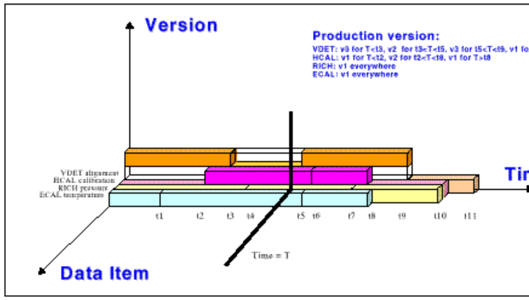

Figure 1 The three axes for identifying uniquely each data item in the condition database

- **Data item classification**
  - Data items organized in "folders" identified by name "/SlowControl/Ecal/Mod1"
- **Versioning and tagging**
  - Different versions of an object with the same validity may exist
  - A consistent set of objects with different versions may be tagged (CVS HEAD)
- **Time axis is 64-bit integer**
  - Intervals have validity range [since, till) and are looked up by validity point
  - Condition data and event data are loosely coupled by design
    - Condition data stored separately and looked up by relevant event time
- **Physical storage and partitioning**
  - Common conventions and directives missing in the API
- **Data content: string/blob**
  - Flexible but cumbersome and often not optimal
- **Design driven by metadata model**
  - C++ API implemented using both ODBMS (Objy) and RDBMS (Oracle, MySQL)

Oracle implementation

- **Developed by Emil Pilecki in IT-DB in spring 2002**
  - Essentially frozen on August 2002 status (Emil left the group in late 2002)
  - Minor ad-hoc changes by A.V. for Harp migration in November 2003
- **Oracle 9i implementation issues**
  - Purely relational data model, no object features
  - Client access through OCCI library (more user friendly than OCI)
    - *Concern for Linux: library only released for gcc2.9x, no gcc3.2 version yet*
  - Use of PL/SQL stored procedures, materialized views, indices
- **Performance still far from optimal**
  - Bulk retrieval of BLOB data not yet implemented
  - Bulk insertion of BLOB data not yet implemented
    - Needs deeper reengineering (now HEAD versioning forces use of autocommit)
- **Strict conformance to original common API**
  - Only minor changes to user code in Harp migration from Objy to Oracle
  - Data migration using export/import to binary files (tools can be readapted)

MySQL implementation

- **Continuous development (Atlas Lisbon) since summer 2002 to-date**
  - 0.2.x (Aug 2002): implementation of original API, very fast and promising
  - 0.3.x (Apr 2003): **API extended**, PVSS *"tiny object"* support (~native int/float)
  - 0.4.ß (Dec 2003): **API extended**, *"CondDBTable"* (complex relational data)
    - Ongoing effort to integrate with POOL (timescale: May 2004)
  - Performance improvements in each release
  - Also include many useful tools (PVSS interface, data browser…)
- **"Far beyond the BLOBs": 7 types of data storage in 0.4.ß**
  - BLOBS (with/without versioning)
  - CondDBTable (with/without versioning)
  - CondDBTable with Id (with/without versioning)
  - POOL and ROOT
- **Development driven by Atlas user requests**
  - 2003 test beams (using PVSS tiny objects)
  - 2004 combined test beam (using CondDBTable)

BaBar's "CDB" (1)

- **Conditions Database was fully redesigned while in data-taking**
  - Design started summer 2001, in production since October 2002
  - "Our dissatisfaction with the older database grew as our experience did"
  - Migration from older database was "half evolution, half revolution"
- **The data model and its implementation**
  - *Design driven by user metadata model:* forget "nice features" of Objy, Oracle…
    - Implementation in production uses **Objectivity as persistent backend**
  - *Metadata model very very similar to that used by CERN common API projects*
    - Condition objects live in 2D space validity-time vs insertion-time (version)
    - "Revision" (insertion-time high watermark) more intuitive concept than "tag"
    - Validity-time axis defined using special BdbTime class
  - Separation of metadata and "payload" (actual data)
    - Metadata has links to existing user *objects*
    - No reverse link: the data itself does not know its validity
  - Data partitioning fully addressed by the logical model of the data
  - C++ API is 95% technology independent

BaBar's "CDB" (2)

- **Distributed database**
  - Distributed model defines masters and slaves
  - Export and import is possible on individual partitions
- **Usage patterns and statistics**
  - Total number of condition items: ~500
    - Slowly updated (*loaded by hand*): alignment, materials…
    - Frequently updated (*loaded automatically ~once per run*): calibrations…
  - Total number of condition objects (user payload): ~1M
    - Using ~400 persistent user classes
  - Total size of the conditions database: ~43 GB
- **My comments**
  - A lot of useful material (presented over the phone at midnight SLAC time!)
  - Not surprisingly, many similarities to the CERN common projects
  - I believe that ***we can still learn a lot from the BaBar experience!***

Harp

- **Data-taking using Objy implementation of common API**
  - Pro: flexible C++ API easily integrated into software frameworks
  - Pro: Objectivity used for event data persistency, no need for special service
  - Pro: "excellent interaction with development team" in CERN-IT/DB
  - Con: performance concern for slow-control data
  - Con: Objectivity-related problems (physical storage, schema changes)
- **Data sources**
  - Online: beam/detector controls (from PVSS, LabView…)
    - Total size ~ 15 GB (rate ~ 2kB/min, compare to 250MB/min event data)
    - *Asynchronous writing process through intermediate ASCII files*
    - All data classes streamed to string
  - Offline: channel mapping, calibration, alignment
    - Condition objects: ASCII files (names stored in conditions database)
- **Data migrated to Oracle implementation of the same API**
  - Harp was the only production user for Objy and is so far for Oracle too

Compass

- **Objy implementation of common API initially considered then abandoned**
  - Lot of work but still not usable when data taking started in 2002
  - Lack of manpower
  - *Lack of user-friendly tools to insert data and browse contents*
- **File-based "FileDB" used for 2002 data-taking**
  - Calibration data in ASCII files
  - Metadata (validity interval) hardcoded in the file names
    - File names: *"DetectorName~~StartOfValidity~~EndOfValidity"*
- **Metadata duplicated in "MysqlDB" in 2003**
  - Much easier bookkeeping of calibration files using an RDBMS
  - Additional metadata not available in file names
  - Other functionalities and tools introduced
- **All 3 implementations hidden behind very simple Compass-specific API**

CDF and D0

- **Both CDF and D0 chose Oracle**
  - Objectivity initially considered then abandoned
  - Different software/schema designs using shared Oracle support at Fermilab
  - *Design driven by choice of persistency solution*
    - C++ API determined by the data structures accessed in the specific schemas
    - Individual schema for each D0 detector vs. unified approach for all CDF detectors
    - Emphasis on calibration data; slow control data also present but separate
  - Separate online and offline servers with different optimizations
    - Data replication via Oracle tools, data distribution via MySQL or text files
- **Size of the project**
  - For each of CDF/D0: ~100 tables, ~100 GB total size (2 years)
  - Estimated ~16 FTE needed (DBA, programming, detector…)

Atlas (overview)

- **Conditions database is one of many databases**
  - Upload online data from DCS, configuration DB, online bookkeeper
  - Conditions database is the main source of info for offline software
- **Prototyping work and timescales**
  - Use Lisbon **MySQL implementation of extended API**
  - Online: successful 2003 test-beam, looking towards 2004 combined test-beam
    - PVSS data stored through direct interface, other data as strings or blobs
  - Offline: focus on 2004 combined test-beam and data challenges
    - Prototype variety of storage formats (native, string, blobs, complex POOL objects…)

Atlas (online)

- **Test-beam data types**
  - Raw data from PVSS (temperatures, voltages…)
    - Time stamp; Stored on change; Relational tables of numbers
  - Processed data (alignment, calibration…), stored periodically
    - Interval-of-Validity; Stored periodically; XML blobs
- **Test-beam requirements**
  - Data browsing tools with plotting capabilities (via ROOT)
  - Enhanced tagging and higher level interfaces
  - Performance tests for data insertion and retrieval
- **User (detector expert)'s point of view: need full-function RDBMS**
  - Internals of the Conditions DB should be accessible via queries
    - *"The APIs should be optional toolkits, which do not exclude direct SQL queries"*
  - Interval-of-Validity aware storage for opaque blobs is not enough
  - *Should be used for detector problem diagnosis, not only offline reconstruction*
  - Separation of Conditions DB and Configuration DB should be removed

Atlas (offline)

- **Conditions DB scope is that of an Interval-of-Validity database**
  - *Emphasis should be on temporal (IoV) metadata rather than on actual data*
  - Actual condition data may reside outside the IoV DB and be referenced by it
    - "LHC experiments already know how to store complex objects": via POOL
- **Any data object may be assigned an IoV (i.e. registered in IoV DB)**
  - Assigning the IoV may come much later than storing the data object itself
  - The role of the IoV database is to mediate access to the correct data object
    - The object can be accessed also in "unmediated" way (exists independently of IoV)
- **Miscellaneous requirements from common project**
  - Enhanced tagging mechanism (very clearly defined)
  - Support tighter integration with POOL (storing POOL references in IoV DB)
  - Time validity issues: time stamp vs IoV; (run,event) instead of time
- **A relational backend for POOL would fit in the picture**
  - Conditions object definition via LCG SEAL dictionary
  - Conditions data storage via LCG POOL relational backend

CMS (overview)

- **Conditions database is one of four types of databases**
  - Together with construction DB, equipment mgmt DB, configuration DB
- **Two scopes for the Conditions DB**
  - Offline reconstruction
  - Online error tracking of detector
  - *Keeping in mind that the online data volume is much larger*
- **CMS wish list from the Conditions DB project**
  - The implementation shall be relational (and RDBMS tools/features fully used)
  - Data management and data handling tools are needed
- **CMS has no experience with the current API implementations**
  - Are there any alternatives to a classical API?
- **Manpower situation in CMS databases is very difficult**
  - But a Conditions DB must be available for 2005 test-beams and possibly sooner

CMS (core sw issues)

- **Different types of conditions data**
  - Simple (raw data from measurements): no versioning
  - Processed (computed): needs versioning (as well as algorithm metadata)
- **Framework integration issues**
  - Read conditions data relevant to event analyzed (synchronization)
    - Synchronization means retrieval of pointer to relevant condition data objects
    - "Dereferencing" the pointers depends on the choice of storage technology
  - Store condition data computed by algorithm executed
    - Only needed for conditions data computed from event data
  - Data distribution from Tier0 to TierN and viceversa
  - Offline is more object-oriented, online more data-oriented (RDBMS)
- **Offline (object-oriented) vs online (data-oriented, RDBMS)**
  - The way to solve the issue is a **relational backend for POOL**
    - POOL shall store objects in a relational backend
    - POOL shall retrieve data stored using *any* relational design (non-intrusive POOL)
  - *CMS (online) requires freedom to design relational model for condition data*

LHCb (overview)

- **Conditions DB scope is only offline reconstruction**
  - Not intended to troubleshoot the detector or the DAQ
  - Emphasis should be on (distributed) data analysis
- **Prototype work in LHCb (not used in production yet)**
  - Started when common project launched; no work for >1 year now (no manpower)
  - Emphasis on framework integration (fully transparent for the end user)
    - Actual data retrieval from references stored in the Conditions DB as strings
    - Data synchronization
- **Requirements from the common project**
  - *More than one implementations*
  - *Tools for data management, replication, browsing/editing*
  - *DB service deployment*
- **Part of the job is LHCb-specific**
  - Data contents, data sources, integration with Gaudi framework
  - Develop coherent calibration/alignment procedures amongst subdetectors

LHCb (online)

- **Two completely independent users**
  - Experiment Control System: writes raw data into Conditions DB
    - Only output, via PVSS (raw data with no versioning)
  - Event Filter Farm: reads condition data to process events online
    - Input and output (computed conditions data with versioning)
- **A special in-memory implementation of the API is needed for EFF**
  - The Gaudi framework runs on the EFF using the same services and API
  - But faster access to condition data than via a database is needed

General impressions

- **A very useful workshop:** ***thanks again to all speakers/participants!***
  - Nice constructive atmosphere and a lot of interest (
  - Occasion for different experiments/groups to compare their ideas directly
  - Collection of useful reference material about other existing projects
- **Requirements and points of view sometimes very different**
  - Not necessarily "exp. A vs exp. B" differences, also "online vs offline"
- **Some problems are experiment or detector specific**
  - They require specific solutions and are not the common project's job
  - e.g.: experiments' recommended conventions
  - e.g.: design of specific detector data schema or application
- ***The "common" project should concentrate on "common" solutions***
  - Develop generic components/tools that can be used in more than one case
  - "Factor out" from all requirements those that admit common solutions

Support for relational databases

- **Large interest for relational data**
  - As opposed to opaque blobs, or complementary to them
- **Many (online?) people *also* want the freedom of a full-function RDBMS**
  - *The freedom to design and implement their own data model and schemas*
  - At the same time the common project can only address common solutions (components that can be factored out: metadata model, API, common schema…)
    - The rest should be done by individual detector experts (with help from DB experts)
- **Some degree of agreement to move relational data support to POOL**
  - From both online and offline
    - Even the Lisbon group showed interest in moving some functionality to POOL
  - This may solve various issues and concerns
    - Provide an easy way to store condition data into a (predefined) relational backend
    - Allow condition data stored in ~arbitrary schemas to be seen through a common API
  - *But, to be implemented, this needs formal agreement from the experiments*

Brief overview of other issues

- **Data partitioning and replication**
  - API extensions as well as replication tools are needed
- **Interactive data browsing**
  - Tools needed with query and plotting capabilities
- **Improved data item addressing**
  - Relational rather than by simple folder names
- **Versioning and tagging enhancements**
  - Versioning by insertion time more intuitive
  - User tagging at insertion time
- **Validity time issues**
  - Timestamps vs IoV, (run,event)…
  - Synchronization layer, in-memory implementation…
- **Store data other than BLOBs**
  - Simple data storage may be useful even if relational POOL goes on

Manpower

- Presently: 80% of an FTE (A.V)
- Lisbon developers interested in contributing

Anyone else?

(No formal signup of manpower from the experiments yet)

Workplan (Phase 0)

- **Release both Oracle and MySQL implementations in LCG CVS**
  - In the state they are now, with own APIs and tests/examples
  - With their own build system
  - With basic documentation

Timescale: by next week

- Oracle implementation pre-released today (package CondDBOracle)
  - API, implementation and examples unchanged w.r.t. Emil's 0.4.1.6 version
  - Using SCRAM
  - *Only available for rh73_gcc2952*
  - *http://lcgapp.cern.ch/project/CondDB/ (tag CONDDB_0_0_0-pre1)*
- MySQL implementation will follow soon (Package CondDBMySQL)
  - Move of master CVS repository from Lisbon to CERN
  - Using autoconf/make
  - For the LCG supported platforms

Workplan (Phase 1)

- **Factor out API (as common dependency for both implementations)**
  - As close to original API as possible (a few technical issues to solve)
    - Lisbon API extensions maintained in MySQLCondDB package
    - Minimize hassle to existing users (mainly Atlas, also Harp and LHCb… *anyone else?*)
  - Common tests/examples with consistent basic documentation
  - Same build system (SCRAM)
  - *Start integration of existing tools using common API*
  - *Start development of examples storing POOL references in the Conditions DB*
  - *Start circulating main directions for API extensions to implement in phase 2*
- **Expect external input from three fronts before starting phase 2:**
  - Availability of Oracle OCCI libraries for rh73_gcc32
    - Else: plan port of Oracle implementation to another client library (which one?)
  - Formal agreement on support for relational backend to POOL
    - Else: plan and rediscuss more relational support inside ConditionsDB
  - Manpower allocation

Timescale: by mid-March

Workplan (Phase 2)

- **Design, circulate and agree a new common API**
  - Taking into account ideas expressed at the workshop and later
  - Taking into account POOL software responsibilities too
  - Production-version implementations for both Oracle and MySQL
- **Integration with POOL**
  - No direct dependency, provide component that sits on both POOL and CondDB
- **Tools**
  - Import/export across implementations
  - Data browsing
- **Package-specific issues**
  - Oracle: improve performance by partial reengineering
  - MySQL: continue to support existing users, keep in mind schema evolution

Timescale: a few months

Also depends on POOL workplan, news from Oracle and manpower

Conclusion

- **A very useful workshop:** ***thanks again to all speakers/participants!***

- **Expect input on a few unresolved issues**
  - Oracle OCCI libraries
  - Relational backend support in POOL workplan
  - Manpower

- **Work has started according to the project workplan**

- **Suggestions, ideas, requirements are always welcome!**
  - Sign-up to contribute to the project
  - Express your interest in using existing code and influence the next version
  - E-mail Andrea.Valassi@cern.ch or project-lcg-peb-conditionsdb@cern.ch