Conditions Database

Andrea Valassi

(CERN IT-ADC)

Outline

- Overview
- Work plan, manpower, status
- Software details

Apologies: I will not cover any database internal details

Some other time, after the software is released and stable ☺

What are conditions data?

- **Non-event detector 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 and calibration
  - Event reconstruction and analysis
- **Other presentations for fewer, or more, details**
  - 3D Workshop presentation
  - October AAM presentations
  - CHEP presentations

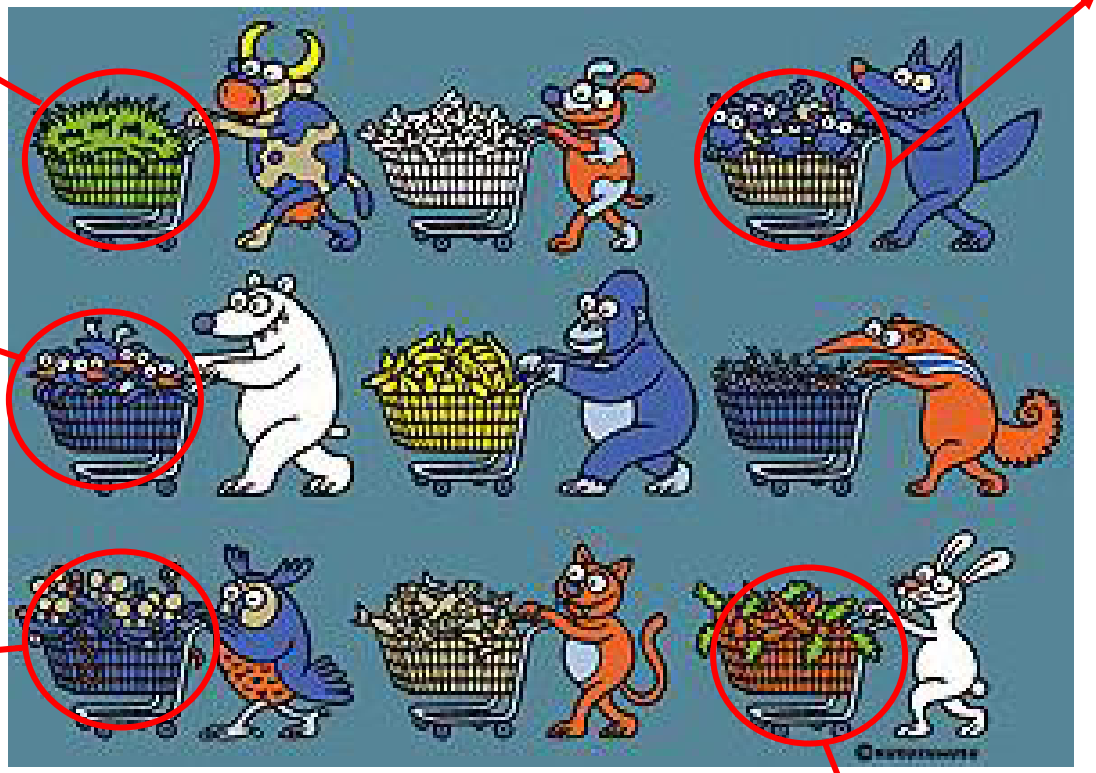
About the "common" project…

- **Somewhat challenging to identify *common* requirements**

Reading the XML BLOB containing the LHCb calibration data valid for the event processed

Retrieving the POOL alignment object for the run processed

Registering that the CMS detector geometry in a set of Oracle tables is valid for 2008 and 2009



© KUKUXUMUSU

Storing Atlas high voltages from PVSS into MySQL whenever the values change (every few seconds)

Reading Alice alignment from the ROOT files for the run processed

Commonalities? Project goals and non-goals

- **Project non-goals (experiment-common, not conditionsDB-specific)**
  - Generic C++ access to relational databases (→ POOL project: RAL)
  - Generic relational database deployment and data distribution (→ 3D project)
    - Integration with data distribution infrastructure, however, is a project goal

- **Project goals (experiment-common, conditionsDB-specific)**
  - Common software and tools for non-event time-varying versioned data
    - You will need to work a lot to customize the common solution to your needs!
    - Central coordination of activities inside each experiment is a necessity

- **Project non-goals (experiment-specific)**
  - Specific data models for calibration/geometry/… (→ experiments)
  - Specific payload format encoding (→ experiments)
    - That is to say: how you use relational databases, RAL or POOL is up to you!
  - Specific time encoding and other conventions (→ experiments)

Overview - first phase of the project (Dec 2003-Oct 2004)

- **LCG integration of existing Oracle and MySQL packages**
  - Latest release CONDDB_0_2_0 in July
  - Atlas test beam data taking using the Lisbon MySQL API and implementation
  - Main problems: lack of development manpower and divergence of two packages

- **In parallel: coordinate discussion about new API, software and tools**
  - Review limitations of current API and software
  - Collect new user requirements

- **Decision to start a new implementation taken at the AAM mid-October**
  - Work plan circulated beginning of November
  - First two milestones due in December and the next one last week

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)

A CondDBObject has

- Metadata:
  - Data item identifier
  - Start-of-time-validity
  - End-of-time-validity
  - Version number
- Data "payload":
  - Encoded as BLOB/string

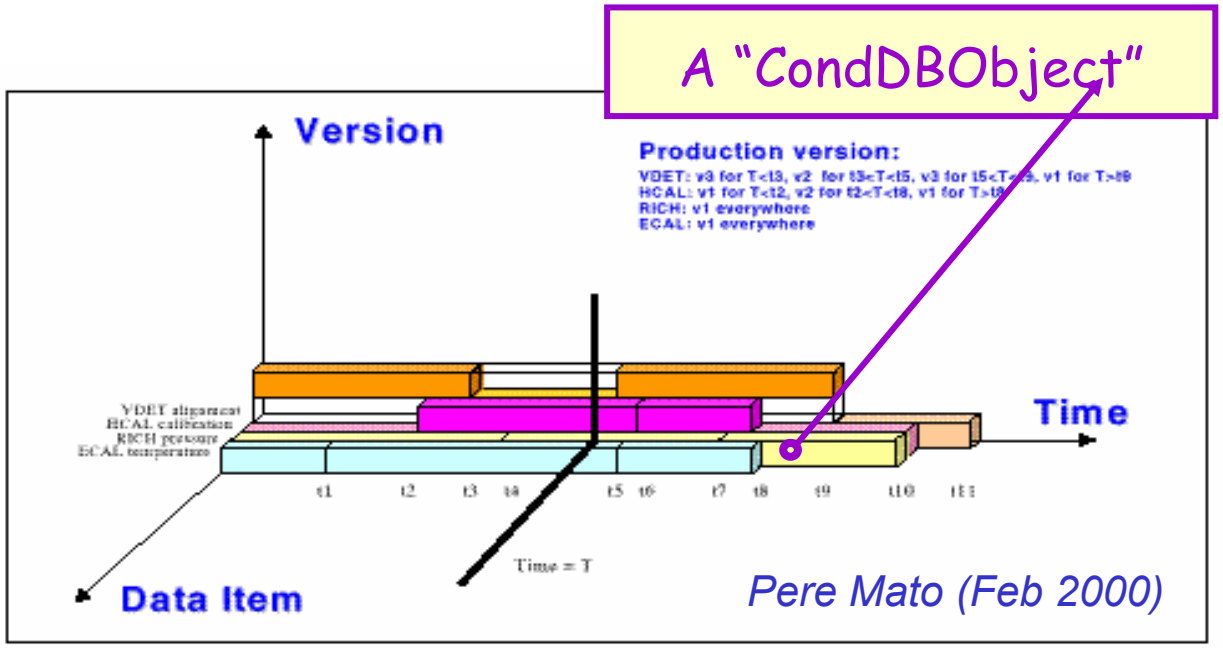


Figure 1 The three axes for identifying uniquely each data item in the condition database

Atlas-Lisbon API extensions

- **Support for data payload other than BLOBs**
  - "ICondDBTable" interface for user-defined data fields
  - ICondDBTable "with ID" for improved data item addressing
- **Support for "online" data not requiring versioning**
  - Measured (temperature) rather than computed (alignment)
- **MySQL implementation of extended API**
  - Useful tools built above extensions (PVSS, data browser)
  - Used for Atlas test beams and integrated with Athena

Why a new software implementation?

- **CondDBOracle: poor performance and missing functionalities**
  - Bulk insertion and retrieval missing in key points of the code: very slow!
  - No user defined data payload (only BLOBs)
  - No online mode (only versioned data)

- **CondDBMySQL: improvements necessary in the API and schema**
  - Cleaner API: whole table vs. individual row (object); schema vs. data
  - Unified approach to schema and code for online and versioned conditions data

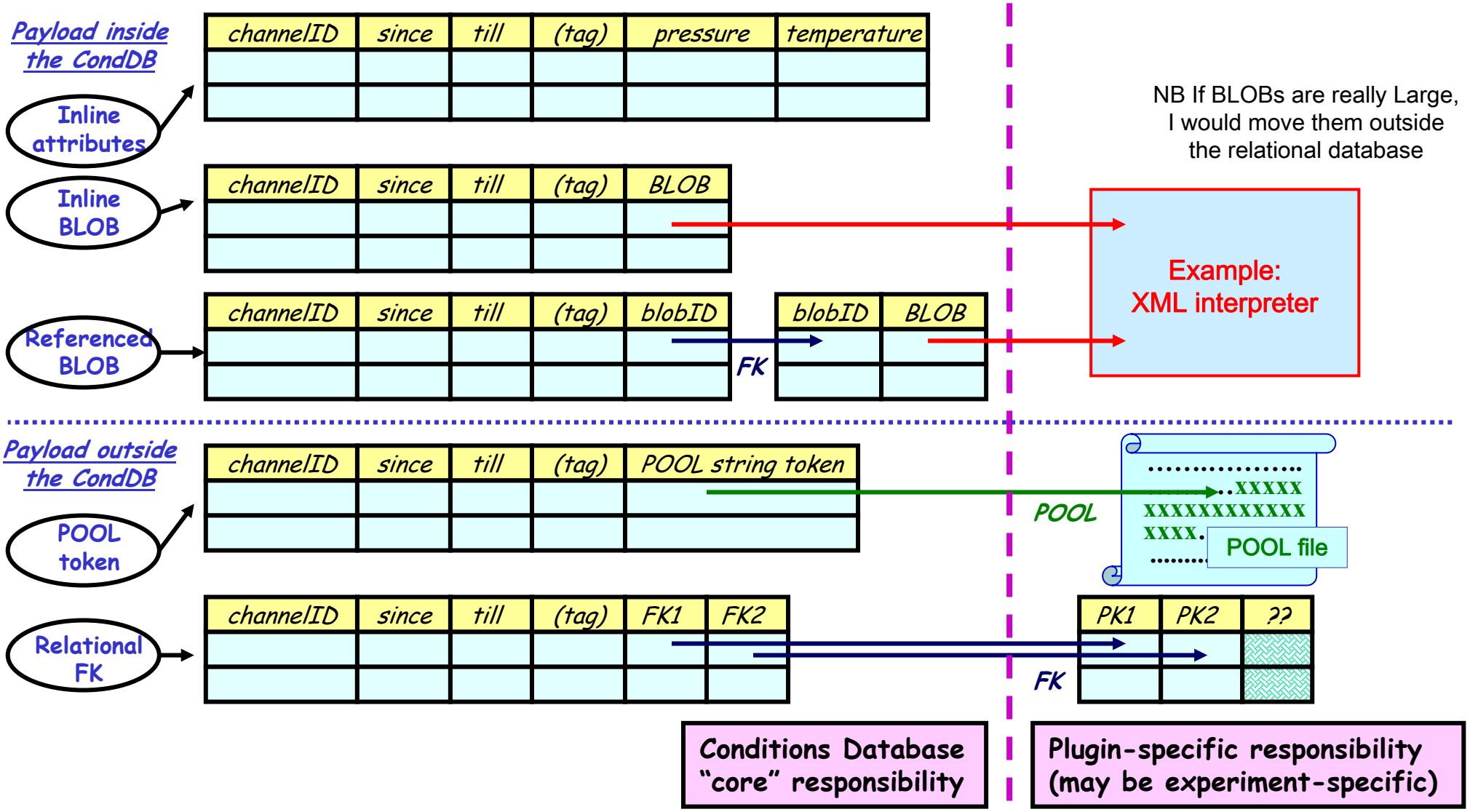
- **Minimize differences between Oracle and MySQL implementations**
  - Use the same data model and schema to simplify data copy across backends
  - Address new user requirements (eg user tags, HVS) using a unified approach
  - Minimize duplication of effort, factor out common code whenever possible

- **Maximize integration with and reuse of LCG software (SEAL/POOL)**
  - Utilities (Time, AttributeList), Plugin management, RAL, POOL integration…

New software - overview

- **Single implementation for Oracle/MySQL using RAL**
  - Strictly the same relational schema for both back-ends
  - Direct implementations (MySQL, OCCI, …) may always come later
- **User-defined data payload modeled by POOL AttributeList**
  - Support for simple types (int, float, bool, string) of various precisions
  - Support for BLOB data type as just one of many possible data types
- **Support both versioned data and online non-versioned data**
  - Start with online data (simpler), then focus on offline (main use case)

Data payload: typical use cases



Work plan and manpower

- **2005 work plan will depend on user priorities and manpower**
  - Present workplan (Nov04 – Feb05) focuses on Atlas March timescales
- **Active manpower**
  - Andrea Valassi (IT) – coordination
  - Sven A. Schmidt (ATLAS) – core development
  - Antoine Perus (ATLAS) – performance testing
- **Additional manpower?**
  - ATLAS: Nuno Fiuza da Barros (Atlas integration), Vakho Tsulaia (HVS?), Andrea Formica (Atlas integration?), …
  - LHCb: Marco Clemencic and Nicolas Gilardi (LHCb integration?)
  - CMS: Ricky Egeland (PVSS?), Michael Case (CMS integration?), Lee Lueking (FroNTier?)
  - *Users and experiment integration developers are important*

Work plan (November 2004 - February 2005)

- **Milestone 1 (December 3) – Andrea V. and Sven A. Schmidt**
  - Online data: single object insertion/retrieval with AttributeList data payload
- **Milestone 2 (December 17) – AV and SAS**
  - Online data: bulk insertion/retrieval
- **Milestone 3 (January 21) – Antoine Perus**
  - Online data: performance comparison with Lisbon CondDBMySQL
- **Milestone 4 (February 12) – AV, SAS, AP**
  - Offline data: versioning and HEAD tagging
- **Next steps (still missing features)**
  - Core software: partitioning, tag extensions, HVS, concurrent tests, other platforms, data cloning, out-of-line data, non-int64 keys, channel attributes…
  - Tools and components: PVSS manager, POOL handler, data browser, data copy…

Status

- **Milestone 1 (December 3) – OK! (AV and SAS)**
  - Online data: single object insertion/retrieval with AttributeList data payload
  - Additional problems/limitations:
    - 64-bit integers (validity keys) not yet in AttributeList/RAL: use 32-bit for now
    - BLOBs not yet in AttributeList/RAL: use strings (<4000 or 255 char) for now
- **Milestone 2 (December 17) – OK! (AV and SAS)**
  - Online data: bulk insertion/retrieval
  - Additional problems/limitations: as above, plus need some cleanup
- **Milestone 3 (January 21) – work in progress (AP, AV and SAS)**
  - Promising results from first comparison with CondDBMySQL (up to 200k rows)
    - Writing faster in COOL Oracle/MySQL than CondDBMySQL (but not strictly linear)
    - Reading faster in CondDBMySQL by ~ one order of magnitude
  - Further understanding and optimizations are necessary (SQL? ODBC? ...?)
- **Milestone 4 (February 12) – will be late (versioning/tagging software)**
  - SAS and AV also got involved with performance studies (higher priority)

New software – first prototype
COOL (COnditions Objects for Lcg)

- **Infrastructure: SCRAM, single platform so far (rh73_gcc323)**
- **Tight integration with SEAL**
  - COOL database service is a SEAL service (dynamically loadable plugin)
  - Use SEAL logging, exceptions, Time, int64…
- **Two packages**
  - CoolKernel (public API)
    - Compile time dependency on SEAL SealBase and POOL AttributeList
    - API design tries to improve on both original and Lisbon API
  - RelationalCool (implementation using POOL RAL)
    - Compile time dependency on SEAL SealServices and POOL RelationalAccess
    - Runtime loading of POOL OracleAccess/ODBCAccess/SQLiteAccess (on demand)
    - Separate classes for generic 'Relational' implementation and 'Ral'-specific DB access (foresee the possibility to reuse the same schema for direct MySQL/OCCI access)
    - HVS entities factored out in class and schema (only structure: no versioning yet)
  - Database implementation issues
    - The Oracle execution plan for each operation is tested to ensure indexes are used
    - Bind variables and bulk operations are used whenever possible
    - Each user operation offered by the API is encapsulated within one transaction

Example – bootstrap

- **Foreword for all examples**
  - The API and implementation code is all enclosed in "namespace cool { }"
  - The API includes "typedef boost::shared_ptr<xxx> xxxPtr" for all xxx
- **Bootstrap: dynamically load the RalDatabaseSvc**
  - Option 1: from an existing SEAL Context (seal::Context* context)

```
context->component<seal::ComponentLoader>()->load
  ("COOL/KernelServices/RalDatabaseSvc");
seal::IHandle<cool::IDatabaseSvc> dbSvcHandle =
  ctx->query<cool::IDatabaseSvc>("COOL/KernelServices/RalDatabaseSvc");
cool::IDatabaseSvc& dbSvc = *(dbSvcHandle->get());
```

  - Option 2: standalone (if SEAL/POOL is not used anywhere else)

```
cool::IDatabaseSvc& dbSvc =
  cool::RalDatabaseSvcFactory::getDatabaseService();
```

- **The IDatabaseSvc can then be used to create, open, drop databases**

Example – database access

- **Database identification via a single URL-like string** (std::string dbId)
  - One "COOL conditions database" corresponds to
    - Logically, a single hierarchy of folder sets and folders (a single root folder set "/")
    - Physically, a set of tables with the same prefix within a single schema (Oracle user schema or MySQL database) accessed via a (possibly different) authenticated user
      - Only the convention to retrieve a single "bootstrap" table is hardcoded in C++: all other table names are stored and retrieved from the database itself
  - Oracle:

```
dbId="oracle://devdb9;schema=US;user=US;password=PW;dbname=COOLTEST";
```

  - MySQL:

```
dbId="mysql://atlobk01;schema=US;user=US;password=PW;dbname=COOLTEST";
```

- **Use the IDatabaseSvc to create, open, drop databases**

```
dbSvc.dropDatabase( dbId );
cool::IDatabasePtr db = dbSvc.createDatabase( dbId, …optional attributes… );
cool::IDatabasePtr db = dbSvc.openDatabase( dbId );
```

Example – folder access

- **Create an "online" folder with user defined payload specification**

```
pool::AttributeListSpecification payloadSpec;
payloadSpec.push_back("I","int");
payloadSpec.push_back("S","string");
payloadSpec.push_back("X","float");
cool::IFolderPtr folder = db->createFolder
  ( "/a/b/c/myfolder", payloadSpec, cool::FolderVersioning::ONLINE, true );
```

- **Retrieve an existing folder**

```
cool::IFolderPtr folder = db->getFolder( "/a/b/c/myfolder" );
```

- **Drop an existing folder**

```
db->dropFolder( "/a/b/c/myfolder" );
```

- **New API: IFolderPtr is used to access the individual conditions objects**
  - Each folder handle is a manager of the data in the folder (see next slide)

Example – single object insertion

- **Store a single object**
  - In "online" mode, check that *since(IOV # N) > till (IOV # N-1)*
  - Special case: if *till (IOV # N-1) == +infinity*, set it equal to *since(IOV # N)*

```
pool::AttributeList payload( payloadSpec );
payload["I"].setValue<int>( 1 );
payload["S"].setValue<std::string>( "Object 1" );
payload["X"].setValue<float>( 0.001 );
folder->storeObject( 0, 10, payload, 1 ); // since = 0, till = 10, channel# = 1
folder->storeObject( 10, cool::IValidityKeyMax , payload, 1 ); // till = +infinity
folder->storeObject( 20, 30, payload, 1 ); // set previous till to 20
folder->storeObject( 5, 25, payload, 1 ); // EXCEPTION: 5 < 30 (last IOV)
```

  - Type (since, till) : IValidityKey (typedef'ed to seal::LongLong for now…)
  - Type (channel) : IChannelId (typedef'ed to unsigned long for now…)

Example – single object retrieval

- **Retrieve a single object (NB: no concept of tag yet)**
  - Retrieve an object from a folder at a given validity point in a given channel

    ```
    cool::IObjectPtr object = folder->findObject( 5, 1 ); // time = 5, channel# = 1
    ```

  - Retrieve the full object payload as an AttributeList

    ```
    pool::AttributeList payload = object->payload();
    ```

  - Retrieve individual payload items as true types (wrapper for user convenience)

    ```
    int i = object->payload<int>("I");
    string x = object->payload<string>("X"); // EXCEPTION: "X" is a float
    string s = object->payload<string>("S");
    ```

  - Retrieve individual payload items as strings (wrapper for user convenience)

    ```
    string i = object->payload("I");
    string x = object->payload("X");
    string s = object->payload("S");
    ```

Example – bulk object insertion

- **Store many objects in bulk**

```
pool::AttributeList payload( payloadSpec );
payload["I"].setValue<int>( 1 );
payload["S"].setValue<std::string>( "Object 1" );
payload["X"].setValue<float>( 0.001 );
folder->setupStorageBuffer(); // Enable bulk insertion
folder->storeObject( 0, 10, payload, 1 ); // Cache in C++ class memory
folder->storeObject( 10, cool::IValidityKeyMax, payload, 1 );
folder->storeObject( 20, 30, payload, 1 );
folder->flushStorageBuffer(); // The SQL is issued here in one transaction
```

Example – bulk object retrieval

- **Retrieve a horizontal iterator over the objects (bulk retrieval)**
  - In online mode (just like in a tag), only one IOV is valid at any given time
    - Eventually this method will be used to browse horizontally within a tag

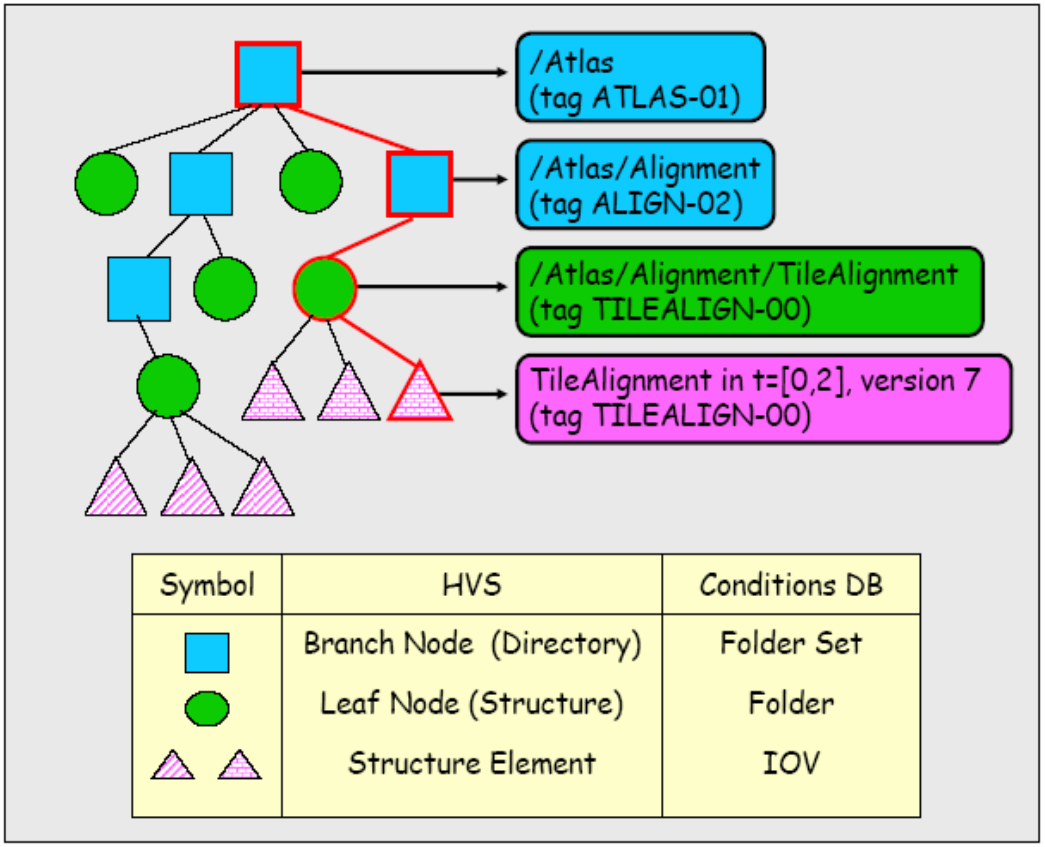
```
cool::IObjectIteratorPtr objectIterator =
  folder->browseObjectsInTag // SQL bulk retrieval in one transaction
  ( "", 1, 5, 25 ); // Tag = "" (only online option), channel# = 1, within [5,25]
objectIterator->goToStart();
while( objectIterator.hasNext() ) {
  cool::IObject object = objectIterator->next(); // In-memory loop
  ...
}
```

  - Implementation detail: presently the iterator is a wrapper to a vector
    - ALL data from the query are bulk-retrieved immediately in one transaction
    - Eventually, need to split this up into many network round trips in many transactions
    - Query on object insertion time and folder creation time may be used to select consistent state across different transactions (objects are never deleted)

Feedback on RAL and AttributeList

- **Excellent support from the RAL team (thanks!)**
  - Sufficiently easy and intuitive to use if you have experience with SQL C++ APIs
- **RAL performance within COOL is being investigated**
  - Performance for writing data better than CondDBMySQL, no real issue so far
  - Read performance: COOL optimizations needed, some issues with AttributeList
    - No difference observed between MyISAM and InnoDB; any penalty from ODBC?
- **Issues with RAL (and AttributeList)**
  - *Need RAL/AttributeList support for long long (high priority)*
  - *Need RAL/AttributeList support for BLOBS (void* buffer, length)*
  - ORA-01466 observed in READ ONLY transaction: better use SERIALIZABLE?
  - Wish RAL extensions for CLOBS (length of a string to store)
  - Wish RAL/AttributeList support for DATE and RAL interface to SYSDATE()
- **Issues with AttributeList**
  - *A few issues (bugs) in copy constructors and assignment operators*
    - Due to coexistence of AttributeListSpecification 'reference' and 'boost shared pointer' "flavours": suggestion is to get rid of reference and keep only boost pointers

"Hierarchical versioning"



Two ways to store the association of "ALIGN-02" and the "TileAlignment in [0,2], version 7" IOV:

1. Store directly the association between the IOV and the "ALIGN-02" tag; ***although "ALIGN-02" is assigned to all IOVs tagged as "TILEALIGN-00", the association is lost***
2. Store the association between the IOV and the local "TILEALIGN-00" tag; then ***store the association between the "ALIGN-02" and "TILEALIGN-00" tags***

In the Conditions Database context

1. Previous Conditions Database tagging (analogous to CVS): "global tags"
2. Hierarchical versioning: "local tags"

- **Originally designed and currently used for the Atlas Detector Description (Vakho)**
- **Scope of application to the Conditions Database: folder set tag management**
  - The association of IOVs to tags within their folder is unchanged

Conclusions

- **Development of new COOL software is proceeding well**
  - A few weeks behind schedule
  - Pending issue: int64 and BLOBs need to be supported in AttributeList and RAL
  - The development plan aims to make this usable by Atlas in March
- **Work plan for 2005 depends on user priorities and available manpower**
  - Feedback, suggestions, experiment requirements are welcome…