The POOL Relational Abstraction Layer

Database Workshop
CERN, January 2005

Radovan Chytracek
CERN/IT/ADC - LCG

Outline

- Introduction
- POOL architecture & RAL
- Features
- Example
- Common status
- Per-plug-in status
- Issues
- New developments
- Conclusions
- Hands on session info

Introduction

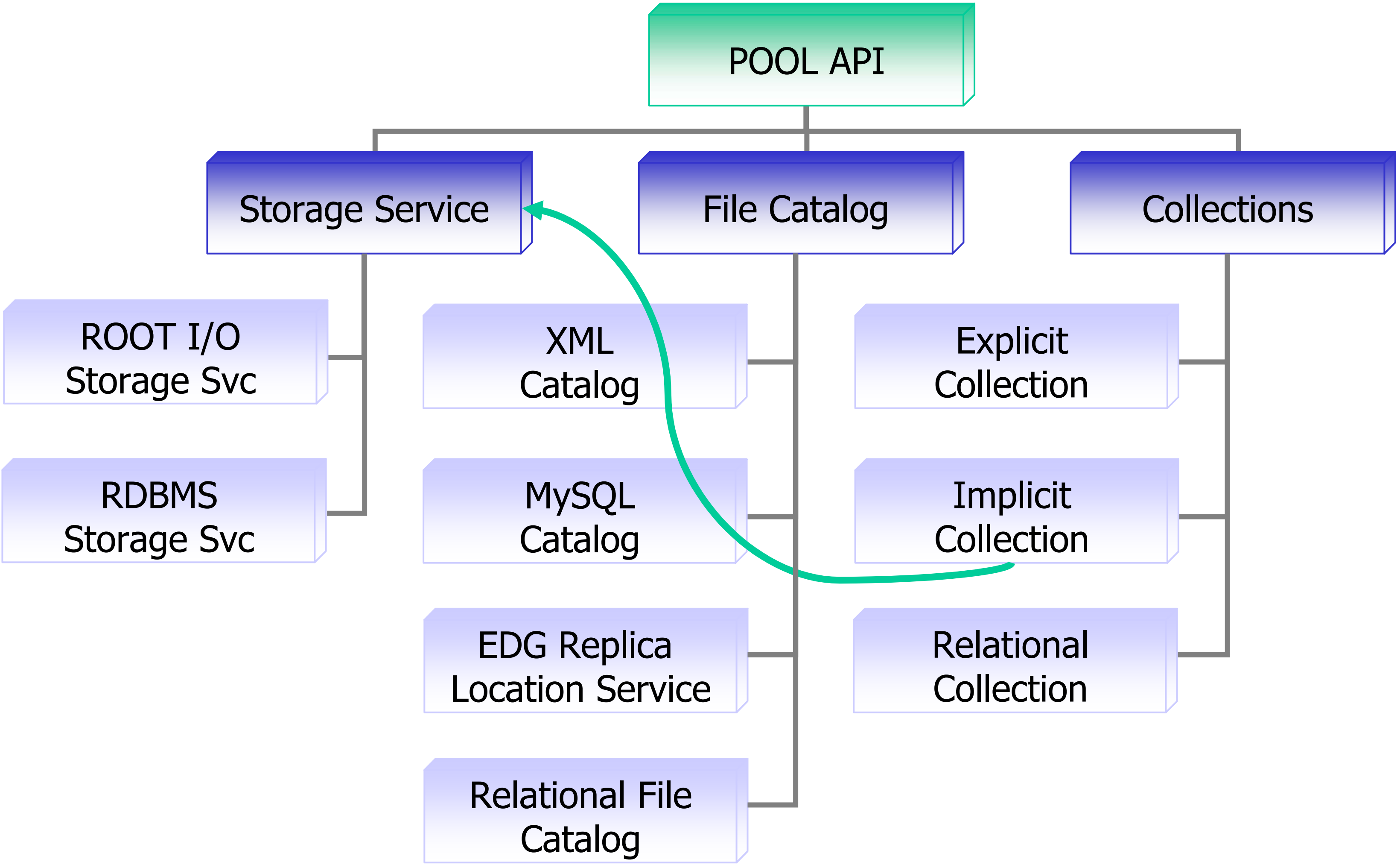
- RAL is addressing the needs of the existing POOL relational components (FileCatalog, Collection), the POOL object storage mechanism (StorageSvc) and eventually also the ConditionsDB (if requested by the experiments).

- Motivation: independence from DB vendors
  - Various issues with available C++ DB APIs
    - Non-standard C++, poor abstraction
  - Each vendor has its own native DB API
    - Usually C based & very verbose
  - Minimal POOL code maintenance costs & flexibility

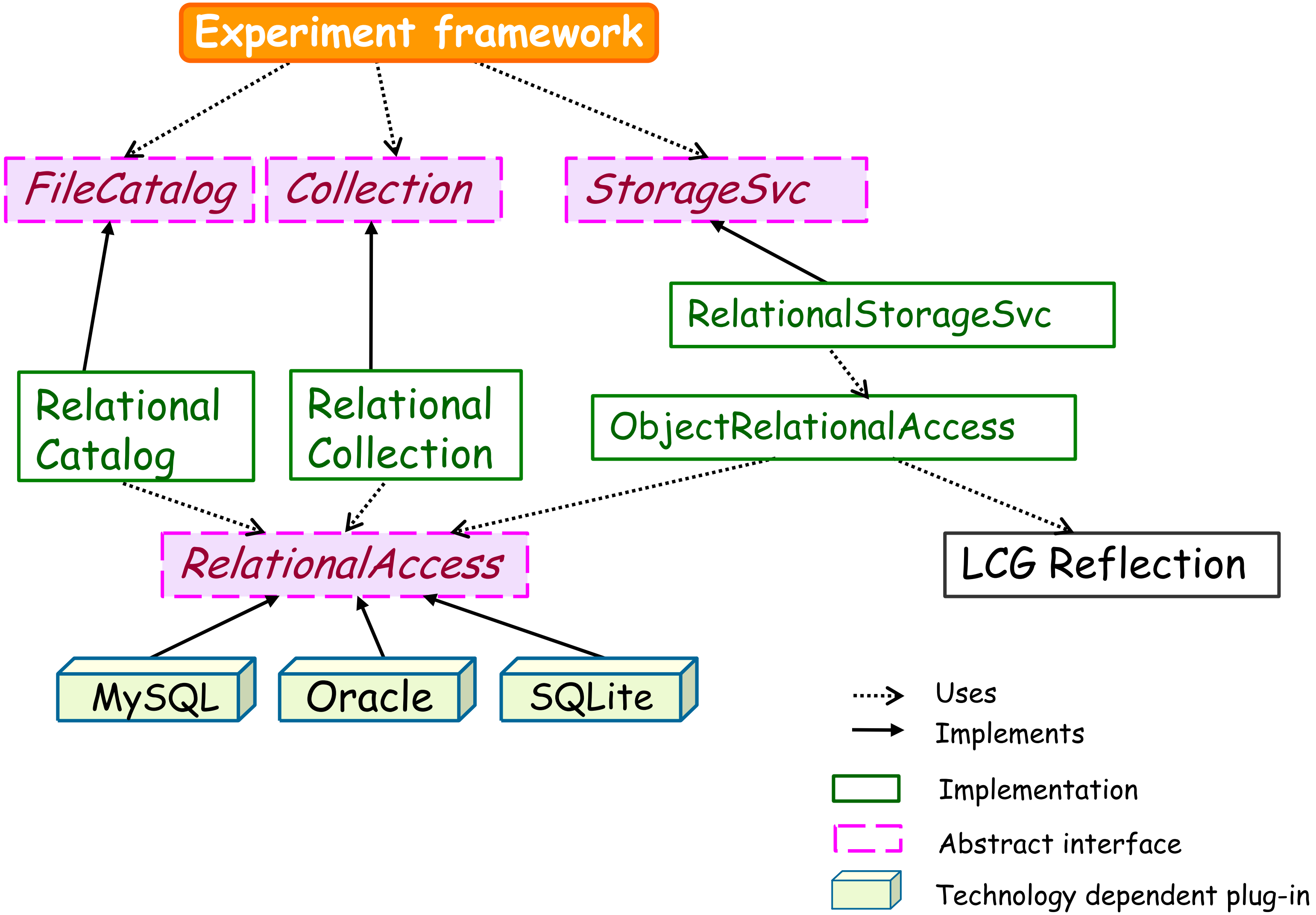
- Activity started for most parts only in March 2004.
  - Requirements collection & domain decomposition
  - Draft project plan

- The use-cases and requirements are defined and updated in close cooperation with experiments

POOL components



in POOL



Experiment framework

FileCatalog

Collection

StorageSvc

RelationalStorageSvc

Relational Catalog

Relational Collection

ObjectRelationalAccess

RelationalAccess

LCG Reflection

MySQL

Oracle

SQLite

⇢ Uses

→ Implements

□ Implementation

⬚ Abstract interface

▭ Technology dependent plug-in

Features

- Abstract, SQL-free API
  - With exceptions of WHERE & SET clauses
- Connection strings storable in a file catalog
  - Example: mysql://raltest/RAL
  - Design decision: no connection credentials in the connection string
- Schema, table, constraints & index handling
  - DDL and meta-data functionality
- Variable binding
  - Named variables syntax supported, e.g. :VARNAME
  - ODBCAccess plug-in accepts positional ?-syntax as well
- Queries against single or multiple tables
  - Left joins possible
  - Sub-queries (back-end dependent)
- Cursors
  - Scrollable (forward-only in some cases)
- Bulk inserts
  - Emulated if not supported by the back-end client API or server

Domain decomposition

- Database access
  - IRelationalService, IRelationalDomain, IRelationalSession, IAutheticationService
- Schema handling
  - IRelationalSchema, IRelationalTable, IRelationalTableDescription, IRelationalTableSchemaEditor, IRelationalTableIndexEditor, IRelationalIndex, IRelationalPrimaryKey, IRelationalForeignKey, IRelationalTablePrivilegeManager, IRelationalTypeConverter
  - AttributeListSpecification, AttributeList
- Queries
  - IRelationalQuery, IRelationalSubQuery, IRelationalQueryWithMultipleTable, IRelationalCursor, IRelationalTableDataEditor, IRelationalBulkInserter
- Transactions
  - IRelationalTransaction

Example - Connection

```
POOLContext::loadComponent("POOL/Services/XMLAuthenticationService" );
POOLContext::loadComponent("POOL/Services/RelationalService" );

seal::IHandle<IRelationalService>
  serviceHandle = POOLContext::context()->
     query<IRelationalService>("POOL/Services/RelationalService");

IRelationalDomain& domain = serviceHandle->
    domainForConnection("mysql://raltest/RALTEST");

std::auto_ptr<IRelationalSession>
  session(domain.newSession("mysql://raltest/RALTEST"));

session->connect();

session->transaction().start();
session->userSchema().dropTable( "DataTable" );
session->transaction().commit();
```

Example - Create Table

```
session->transaction().start();

std::auto_ptr<IRelationalEditableTableDescription>
  desc( new RelationalEditableTableDescription( log, domain.flavorName()));

desc->insertColumn("id", AttributeStaticTypeInfo<int>::type_name());
desc->insertColumn("x", AttributeStaticTypeInfo<float>::type_name());
desc->insertColumn("y", AttributeStaticTypeInfo<double>::type_name());
desc->insertColumn("c", AttributeStaticTypeInfo<std::string>::type_name());

IRelationalTable&
  table = session->userSchema().createTable( "DataTable", *descr );

session->transaction().commit();
```

Example - Insert Data

```
session->transaction().start();

IRelationalTable& table = session->userSchema().tableHandle("DataTable");

AttributeList data( table.description().columnNamesAndTypes() );

IRelationalTableDataEditor& dataEditor = table.dataEditor();

for ( int i = 0; i < 5; ++i ) {
  data["id"].setValue<int>( i + 1 );
  data["x"].setValue<float>( ( i + 1 ) * 1.1 );
  data["y"].setValue<double>( ( i + 1 ) * 1.11 );

  std::ostringstream os; os << "Row " << i + 1;
  data["c"].setValue<std::string>( os.str() );

  dataEditor.insertNewRow( data );
}

session->transaction().commit();
```

Example - Query

```
// Querying : SELECT * FROM DataTable WHERE id > 2
std::auto_ptr<IRelationalQuery> query( table.createQuery() );
query->setRowCacheSize( 5 );

AttributeList emptyVarList;
query->setCondition( “id > 2”, emptyVarList );

IRelationalCursor& cursor = query->process();

if(cursor.start()) {
 while(cursor.next()) {
  const AttributeList& row = cursor.currentRow();
  for( AttributeList::const_iterator iCol = row.begin();iCol != row.end(); ++iCol ) {
   std::cout << iCol->spec().name() << " : " << iCol->getValueAsString() << "\t";
  }
  std::cout << std::endl;
 }
}

std::cout << “Selected row(s):" << cursor.numberOfRows() << std::endl;

session->transaction().commit());
session->disconnect();
```

Common Status

- The latest is POOL release POOL_2_0_0-iota
  - First RAL components available since POOL 1.7.0
- Base interfaces defined
  - Strictly following requirements
- Oracle, ODBC/MySQL and SQLite plug-ins
  - Native MySQL 4.1.xx development in progress
  - Unit-tested & stressed by ObjectRelational StorageService
- AuthenticationService implementations available:
  - XML and shell environment based
- Proof of concept RelationalFileCatalog implemented
  - tested with Oracle, SQLite and MySQL servers
- First implementation of RelationalCollection

Oracle plug-in

- Oracle plug-in
  - Uses Oracle OCI C API
  - Based on Oracle 10g
    - Supports connection to 9i and 10g servers
    - Makes use of the "binary_float" and "binary_double" SQL types
  - Can be used with the Oracle 10g instant client
- Status
  - Fixed all known bugs and introduced CLOB support

SQLite plug-in

- Flat file database engine
  - Tiny memory footprint
  - Understands most of SQL-92
  - Easy to use API
- First implementation based on SQLite version 2
  - File size and variable binding issues
- Now based on SQLite version 3
  - File size went down by factor of 2
  - Real variable binding implementation in progress

MySQL Status

- MySQL access is via ODBC
  - ODBC-based implementation
  - Native implementation now would run into maintenance problems as MySQL API is changing through versions 4.0 to 4.1 to 5.1
  - Until 5.1 is out POOL access to MySQL via the more generic ODBC plug-in will be maintained
- Uses UnixODBC + MyODBC 3.51
  - Native ODBC manager on Windows
- Tested against MySQL 4.0.18+
- MySQL server requirements
  - InnoDB and ANSI SQL mode are required to keep the RAL semantics

Issues

- Nested queries problems with ObjectRelational StorageService
  - SQLite & MySQL/ODBC (under investigation)
- CLOB trap when using bulk inserts
  - '\0' bytes nor white spaces truncated by MySQL for TEXT columns
  - to be fixed in MySQL & checked for Oracle plug-in
- MySQL 4.0.x InnoDB does not scale well over 10^6 entries
  - Perhaps due to single shared table space file
  - We'll see in 4.1.7 where table space-per-table is possible
  - TEXT column type to be used with care
    - Storage overhead + slow query speed

New Developments

- Will review soon the existing interfaces
  - Extension of the table description interface (column size)
  - Support of BLOB types and “long long”
- After input from LCG 3D project we plan to
  - Add client monitoring support
  - Add Connection pooling
  - Add Database service registry
  - Improve authentication mechanism
- MySQL 4.1.7 native plug-in trial (work in progress)
  - Still no cursors in 4.1 (workaround needed)
  - binary protocol & variable binding (big plus)
  - Easy migration with MyODBC 3.53 for MySQL 4.1.7
    - Available by end of January 2005
- RelationalCollections
  - First prototype is available
  - Testing and integration with real collection data (ATLAS)
- ODBCAccess plug-in re-factoring
  - Allow support for more RDBMs: Oracle, PostgreSQL
  - Most of the points of variability already analyzed
  - Low priority

Conclusions

- We did it, facing the aggressive schedule ☺
  - Coding started in March – full implementations by now
- Oracle plug-in works in all cases
- SQLite & MySQL plug-ins in 99%
- All back-ends heavy stressed by POOL ObjectRelational StorageService
  - see the next talk by Ioannis Papadopoulos
- RAL successfully used in implementation across all POOL application domains
  - File catalog, Collections, StorageService
- Our Thanks to CMS developers and ATLAS geometry database team for close collaboration and useful feedback

RAL Hands on session

- 5 exercises
  - Simple session demonstration
  - Schema listing example
  - Create table
  - Fill table
  - Query table data
- Type “make” to get info how to build & run
- Have a look at the README file
  - To be uplodaed to the workshop page soon ☺

- Have fun!