Conditions DB & Update Manager
status and plans

Marco Clemencic
marco.clemencic@cern.ch
Overview

- Main new features
  - DetCond package
    - interface to COOL
  - Update Manager Service
  - DetCondExample package

- Conclusions
  - Summary
  - Plans / missing features
DetCond
**DetCond: principles**

- **Requirements:**
  - Gaudi-integrated way of using COOL
  - Simplify user–DB interaction
  - Use more than one DB instance (local copy for testing + master DB)
### DetCond: layout

#### DB access

- **ICondDBAccessSvc**
  - `+createFolder()`
  - `+storeXMLString()`
  - `+tag()`
  - `+setTag()`

- **CondDBAccessSvc**
  - `-m_db: cool::IDatabasePtr`

#### Converter

- **IConversionSvc**
- **ConditionsDBCnvSvc**
  - `-m_dbAccSvcs: vector<ICondDBAccessSvc*>`

- **CondDBGenericCnv**
  - `#m_condDBCnvSvc: ConditionsDBCnvSvc*`
  - `#m_detDataSvc: IDetDataSvc*`
  - `#getObject()`
  - `#eventTime()`

- **RelyConverter**
  - `+createObj()`
  - `+updateObj()`

#### Utility methods

- `#m_dbAccSvcs: vector<ICondDBAccessSvc*>`

#### CnvSvc with fall-back converter

- **DetCond**

### Notes

- Persistent to transient cnv.
- Uses detector persistency svc.
- Utility methods with fall-back converter.

LHCb s/w week 25/5/2005 – p. 5/26
DetCond: main features

- ConditionsDBCnvSvc
  - conversion service with fall-back:
    - *if no specific converter* → *use RelyConverter*
  - hold pointers to many ICondDBAccessSvces
DetCond: main features

- ConditionsDBConvSvc
  - conversion service with fall-back: 
    *if no specific converter → use RelyConverter*
  - hold pointers to many ICondDBAccessSvcs

- CondDBGenericCnv
  - encapsulate object retrieval
    *tries the acc. services known by ConditionsDBConvSvc*
**DetCond: main features**

- **ConditionsDBCnvSvc**
  - conversion service with fall-back: *if no specific converter → use RelyConverter*
  - hold pointers to many ICondDBAccessSvces

- **CondDBGenericCnv**
  - encapsulate object retrieval
    *tries the acc. services known by ConditionsDBCnvSvc*

- **RelyConverter**
  - ask DetectorPersistencySvc to convert the string got from CondDB
    *currently only XML strings supported*
DetCond: main features (2)

- (l)CondDBAccessSvc
  - connection to COOL API
  - database handling
    - folder creation
    - object storage
    - tagging
DetCond: how does it work

Example options:

```c
ApplicationMgr.ExtSvc += {"ConditionsDBConvSvc",
    "CondDBAccessSvc"};
DetectorPersistencySvc.CnvServices +=
    {"ConditionsDBConvSvc"};
CondDBAccessSvc.HostName = "conddbhost.cern.ch";
CondDBAccessSvc.User = "conddb";
CondDBAccessSvc.Password = "*****";
CondDBAccessSvc.Database = "LHCB";
CondDBAccessSvc.Schema = "COOL";
CondDBAccessSvc.TAG = "production_v1";
CondDBAccessSvc.BackEnd = "mysql";
```
**DetCond: how does it work**

**Example options:**

<table>
<thead>
<tr>
<th>Load needed services</th>
</tr>
</thead>
</table>

```
ApplicationMgr.ExtSvc += {"ConditionsDBCnvSvc",
                          "CondDBAccessSvc"};

DetectorPersistencySvc.CnvServices +=
                          {"ConditionsDBCnvSvc"};

CondDBAccessSvc.HostName = "conddbhost.cern.ch";
CondDBAccessSvc.User = "conddb";
CondDBAccessSvc.Password = "******";
CondDBAccessSvc.Database = "LHCB";
CondDBAccessSvc.Schema = "COOL";
CondDBAccessSvc.TAG = "production_v1";
CondDBAccessSvc.BackEnd = "mysql";
```
DetCond: how does it work

Example options:

```cpp
ApplicationMgr.ExtSvc += {"ConditionsDBCnvSvc",
  "CondDBAccessSvc"};
DetectorPersistencySvc.CnvServices +=
  {"ConditionsDBCnvSvc"};
CondDBAccessSvc.HostName = "condbhost.cern.ch";
CondDBAccessSvc.User = "conddb";
CondDBAccessSvc.Password = "*****";
CondDBAccessSvc.Database = "LHCB";
CondDBAccessSvc.Schema = "COOL";
CondDBAccessSvc.TAG = "production_v1";
CondDBAccessSvc.BackEnd = "mysql";
```

add the new cnv. service
DetCond: how does it work

Example options:

```cpp
ApplicationMgr.ExtSvc += {"ConditionsDBCnvSvc",
                         "CondDBAccessSvc"};
DetectorPersistencySvc.CnvServices +=
                              {"ConditionsDBCnvSvc"};
CondDBAccessSvc.HostName = "conddbhost.cern.ch";
CondDBAccessSvc.User = "conddb";
CondDBAccessSvc.Password = "*****";
CondDBAccessSvc.Database = "LHCB";
CondDBAccessSvc.Schema = "COOL";
CondDBAccessSvc.TAG = "production_v1";
CondDBAccessSvc.BackEnd = "mysql";
```

set connection parameters
Appropriate IOpaqueAddress generated by XMLCnvSvc for tags like:

\[
<\text{whateverref href="conddb://path/to/object#name"}>
\]
DetCond: how to populate a DB

- you need an access svc

```cpp
ICondDBAccessSvc *accSvc;
StatusCode sc =
    serviceLocator()->service("CondDBAccessSvc", accSvc);
```

- create your folders

```cpp
accSvc->createFolder("/path/to/the/cond","comment",
    ICondDBAccessSvc::XML);
```

- create conditions

```cpp
Condition myCond;
myCond.addParam<double>("temperature", 25.3);
```

- store it

```cpp
accSvc->storeXMLString("/path/to/cond", myCond.toXml(),
    TimePoint(0), TimePoint(10));
```
**DetCond: how to use 2 DBs**

- **Options:**

```cpp
ApplicationMgr.ExtSvc += { "ConditionsDBCnvSvc" };  
ApplicationMgr.ExtSvc += { "CondDBAccessSvc/DB1" };  
ApplicationMgr.ExtSvc += { "CondDBAccessSvc/DB2" };  
ConditionsDBCnvSvc.CondDBAccessServices =  
  { "DB1", "DB2" };  
DB1.HostName = "dbhost1";  
// ...  
DB2.HostName = "dbhost2";  
// ...
```
DetCond: how to use 2 DBs

- Options: 2 instances of the access svc

```c
ApplicationMgr.ExtSvc += { "ConditionsDBCnvSvc" };  
ApplicationMgr.ExtSvc += { "CondDBAccessSvc/DB1" };  
ApplicationMgr.ExtSvc += { "CondDBAccessSvc/DB2" };  

ConditionsDBCnvSvc.CondDBAccessServices =  
  { "DB1", "DB2" };  

DB1.HostName = "dbhost1";  
// ...

DB2.HostName = "dbhost2";  
// ...
```
DetCond: how to use 2 DBs

Options:

2 instances of the access svc registered to the cnv svc

ApplicationMgr.ExtSvc += { "ConditionsDBCnvSvc" };  
ApplicationMgr.ExtSvc += { "CondDBAccessSvc/DB1" };  
ApplicationMgr.ExtSvc += { "CondDBAccessSvc/DB2" };  

ConditionsDBCnvSvc.CondDBAccessServices = { "DB1", "DB2" };  

DB1.HostName = "dbhost1";  
// ...
DB2.HostName = "dbhost2";  
// ...
UpdateManagerSvc
Objects must be updated!
Objects must be updated!

- We cannot go to the DB every event 😞
- We need something
  - automatic: make life simple for users
  - efficient: minimum # of useless operations
  - flexible: handle many use cases
**UMSvc: the problems**

Objects must be updated!

- We cannot go to the DB every event 😞
- We need something
  - *automatic*: make life simple for users
  - *efficient*: minimum # of useless operations
  - *flexible*: handle many use cases

We need an **UpdateManagerSvc**!
An UpdateManagerSvc has to:

- handle generic objects (algorithms, DEs, ...)
  - as user of "condition" objects
  - as "condition" objects
UMSvc: requirements

An UpdateManagerSvc has to:

► handle generic objects (algorithms, DEs, . . .)
  ► as user of “condition” objects
  ► as “condition” objects

► an object can register different methods for different conditions
**UMSvc: requirements**

An UpdateManagerSvc has to:

- handle generic objects (algorithms, DEs, …)
  - as user of “condition” objects
  - as “condition” objects
- an object can register different methods for different conditions
- handle dependencies between registered objects
UMSvc: requirements

An UpdateManagerSvc has to:

- handle generic objects (algorithms, DEs, ...)
  - as user of “condition” objects
  - as “condition” objects
- an object can register different methods for different conditions
- handle dependencies between registered objects
- be dynamically populated (and always consistent)
**UMSvc: requirements**

An UpdateManagerSvc has to:

- handle generic objects (algorithms, DEs, ...)
  - as user of “condition” objects
  - as “condition” objects
- an object can register different methods for different conditions
- handle dependencies between registered objects
- be dynamically populated (and always consistent)
- allow users to trigger an update of all the items that depend on a condition
**UMSvc: requirements**

An UpdateManagerSvc has to:

- handle generic objects (algorithms, DEs, ...)
  - as user of “condition” objects
  - as “condition” objects
- an object can register different methods for different conditions
- handle dependencies between registered objects
- be dynamically populated (and always consistent)
- allow users to trigger an update of all the items that depend on a condition
- be efficient (of course 😊)
UMSvc: an example

Algo1
UMSvc: an example

- Algo1
- mf0
- Cond1
UMSvc: an example
UMSvc: an example

Algo1

mf0

mf1

mf2

Cond1

Cond2
UMSvc: an example
UMSvc: an example

Diagram:

- Algo1
  - mf0
  - mf1
  - mf2
    - Cond1
    - Obj1
      - mf0
      - Cond2
    - D.E.
UMSvc: an example

Diagram:
- Algo1
  - mf0
  - mf1
  - mf2
  - mf3
  - Cond1
  - Obj1
  - D.E.
  - Cond2

LHCb s/w week25/5/2005 – p. 15/26
UMSvc: an example
UMSvc: an example
How to allow users to do different things when different conditions change?

- let them use their own methods (needs templates $\rightarrow$ no pure abstract interface)
- force them to fill a predefined routine + a switch (pure abstract interface, common interface)
UMSvc: interface

- How to allow users to do different things when different conditions change?
  - let them use their own methods (needs templates → no pure abstract interface)
  - force them to fill a predefined routine + a switch (pure abstract interface, common interface)

I prefer the first approach! 😊

- IUpdateManagerSvc:
  - public template methods that call protected abstract methods
Simple case: object depending on conditions

object IOV is the intersection of cond. IOVs
UMSvc: validity

- **Simple case**: object depending on conditions
  - object IOV is the intersection of cond. IOVs

- **More complex**: condition depending on conditions
  - the parent cond. has its own *intrinsic* IOV
  - parent IOV is the intersection of intrinsic IOV and child ones
**UMSvc: validity**

- **Simple case:** object depending on conditions
  - object IOV is the intersection of cond. IOVs

- **More complex:** condition depending on conditions
  - the parent cond. has its own *intrinsic* IOV
  - parent IOV is the intersection of intrinsic IOV and child ones

- **Most complex:** parent needs many methods (>1)
  - each method depend on many conditions
    - intersection of IOV at method level
  - many methods per object
    - intersection of IOV at object level
Given a dependency network:
Given a dependency network:

- start from the **head** (objects without parents)
UMSvc: navigation

Given a dependency network:

- start from the head (objects without parents)
- intersection of validities
Given a dependency network:

- start from the **head** (objects without parents)
- intersection of validities
  - **object level** (methods)
Given a dependency network:

- start from the **head** (objects without parents)
- intersection of validities
  - **object level** (methods)
  - **method level** (child objects)
**UMSvc: navigation**

Given a dependency network:

- start from the **head** (objects without parents)
- intersection of validities
  - **object level** (methods)
  - **method level** (child objects)
- invalidate object
Given a dependency network:

- start from the **head** (objects without parents)
- intersection of validities
  - **object level** (methods)
  - **method level** (child objects)
- **invalidate object**
- **propagate to parents**
objects that need update notifications should register during initialize

if the call of memb. func. is not needed, you can use a NULL pointer

IUpdateManagerSvc::newEvent() must be called before each event (as soon as the event time is known)

UpdateManagerSvc does the rest!
UMSvc: user guide

- options
  - you need DetDesc
- user c++ file

```cpp
#include "DetDesc/IUpdateManagerSvc.h"

StatusCode MyAlg::initialize()
{
    // ...
    IUpdateManagerSvc *ums;
    StatusCode sc = serviceLocator() -> service("UpdateManagerSvc",
        ums, true);
    ums -> registerCondition(this, "/dd/cond/path",
        &MyAlg::i_method1);
    ums -> update(this);
    ums -> release();
    // ...
}
```

- get the service pointer
- register for a condition
- trigger an update for objects created while processing an event
- release the service
DetCondExample
Example package on DetCond and UpdateManagerSvc

- CondDB test:
  - fillCondDB.opts
  - testCondDB.opts
  - testMultiDB.opts

- UpdateManagerSvc test:
  - testUpdateMgr.opts

+ few extra checks

useful as reference code
Conclusions
Summary

- **DetCond operational!**
  - simple and flexible
  - depend on DetDataSvc for the event time
  - allow other possibilities?

- **UpdateManagerSvc ready!**
  - flexible and easy to use
  - independent of the CondDB
  - may need code clean up

- for testing
  - How-To on the LHCb WiKi

- More informations on the web
  - LHCb Home → Computing → Detector Conditions
Plans for next future

- maintenance
  - keep in touch with COOL new features
  - large scale testing

- further development
  - off-line
    - coexistence of 2 set-ups
      - simultaneous analysis & trigger simulation
  - on-line
    - integrate DetCond and “on-line”
    - implement condition pushing