

Migrating CDB from **Objectivity/DB** to **ROOT & MySQL** (status)

Igor A. Gaponenko
LBL / NERSC / HENPC

Contents

A primary focus is at
“payload” classes and relevant
transient code migration

- **Reasons for the migration**
- **Replacement technologies: ROOT and MySQL**
- **What exactly needs to be migrated**
- **What has already been done and what’s left**
- **A general approach to the “*payload*” migration: through *re-factoring***
- **Tech-neutral proxies, Framework modules and CDB API**
- **Recent changes in BaBar SRT**
- **Straightforward transient-persistent transformation**

Reasons for the migration

- **Save on on a license cost for Objectivity/DB software**
 - Is important, given the expected lifespan of the Experiment: <2.5 of data taking plus 2+ years of a subsequent data analysis
 - CDB will still be needed through all this period
 - Though, very few updates will be need when the data taking & reconstruction will be over
- **Objectivity's development and product support cycle (quite often?) doesn't meet needs of the Experiment**
 - Have/had to use older compilers, operating systems w/ no supported version and bug fixes from Objectivity
 - Potential/known conflicts between Objectivity/DB software and the other software packages we need/want to use
- **A burden of managing database installations**
 - Requires a skilled person per site to take care of Objectivity/DB servers
 - Loosing experts to LHC and other experiments
 - Data distribution is really hard (due to certain limitation of the database software)
- **Potentially, an effect of all previously mentioned problems may get even worse after BaBar will stop taking data**

Where to migrate to (technologies)?

- **Current persistency model is homogeneous:**
 - Use Objectivity/DB everywhere:
 - At least 13 production federations at SLAC alone
 - 7+ installations in Padova: ER1/2/3/...
 - 20+ SP6/8 installations off SLAC
- **Replacing with:**
 - **DDL** persistent classes with **RDL** ones for user defined “*payload*” classes
 - **MySQL** storing CDB metadata (intervals, etc.) and user-defined **RDL** objects as **BLOB-s** (Binary Large Objects) for *read-write* database installations:
 - 5 installations at SLAC: MASTER, IR2, PC1/2/3
 - 7+ installations in Padova: ER1/2/3/...
 - **ROOT** files for *read-only* database snapshots:
 - All analysis
 - All simulation production
 - User-defined contents is shared between two types of databases (copied over from MySQL BLOB-s into ROOT files snapshots)
- **Data conversion services to be established for all three “technologies”**
- **Objectivity/DB will stay for older applications during a transition period**
- **A so called “beta-mini” ROOT files based subset of CDB is available now**

What needs to be migrated

- **Core CDB API, tools, services...**
 - Core CDB API implementations
 - Core Framework modules
 - Data conversion services
 - Data distribution tools, procedures and protocols
 - An environment for testing user development (=test federations)
- **User defined persistent classes (DDL-to-RDL)**
- **Transient user code (proxies, modules, loaders, tests)**
- **Relevant TCL files**
- **Documentation, database use patterns, etc.**

The most difficult problem!

What 's been done and what's left

- **Two new CDB API implementations exist:**
 - Read-write: **MySQL+ROOT**
 - Read-only: **ROOT files**
- **Core Framework modules exist**
 - For managing transactions,
- **Data conversion services exist**
 - Objectivity-to-ROOT
 - Objectivity-to-MySQL(+ROOT)

ATTENTION: due to a varying complexity of users' code it's hard to estimate the amount of efforts needed to migrate that code!

- ~ **40% of user-defined persistent classes (DDL-to-RDL)**
- << **50% of transient user code: proxies, modules**
- << **50% of data distribution tools, procedures and protocols**

Persistent classes/contents to be migrated

- **~215 classes in present Objectivity *schema***
 - See a hierarchy snapshot from some older release:
 - <http://www.slac.stanford.edu/~gapon/CDB/ObjySchema/14.3.0-test-BdbObject/tree.html>
- **There are ~2100 conditions known to MASTER CDB**
 - 1600+ are related to PID Efficiency Tables (have been migrated)
- **There are ~3 millions of persistent payload objects**
 - Most of them are *rolling calibrations*

Transient user code to be migrated

- **Proxies (reading from database only)**

Your expertise is needed

- ~40 : directly based on **CdbBdb/CdbBdbProxyBase**
- ~40 : directly based on **CdbBdb/CdbBdbEnvProxy**
- ?** : indirect (via inheritance) ones

- **Framework modules**

Your expertise is needed

- ~<100 (including those for storing conditions)

- **Framework based applications**

- At least 315 applications in recent nightly builds (who may need CDB)
- More applications in private directories and CVS
- Each has to be analyzed and (if needed) modified
- **Affected ones have to be tested(!)**

Your help is needed

- **Stand-alone applications (loaders, tests, etc.)**

- ~100 of direct those directly using CDB API
- **Have to be migrated and tested**

Your help is needed

The biggest problem: Calibration Toolkit

- **Documentation:**
 - <http://online04.lbl.gov/~brownd/docs/BbrCalib.pdf>
- **Heavily used by DCH, DIRC and EMC calibrations in both ONLINE and OFFLINE**
- **Very rich (in terms of its functionality) software**
- **Tightly coupled with the Objectivity/DB persistency**
 - When it was designed, Objectivity/DB was the only technology
 - No clear persistent-to-transient separation
 - Key classes are intended to be used in both transient and persistent contexts
 - Spreading a calibration into multiple “*parallel*” conditions in the database
 - Hidden mapping to condition names in the database
- **In fact, it’s another API to the Condition/DB**
 - Knows how to store/fetch itself to/from the database
 - straightforward migration principles are very hard to implement
- **A partial (read-only) migration for a subset of classes has been made in a course of the BetaMiniApp migration to ROOT-based CDB (Jane Tinslay, 2005).**
- **Still not clear what to do with it!**

Your expertise & help is needed

Migration through Re-factoring

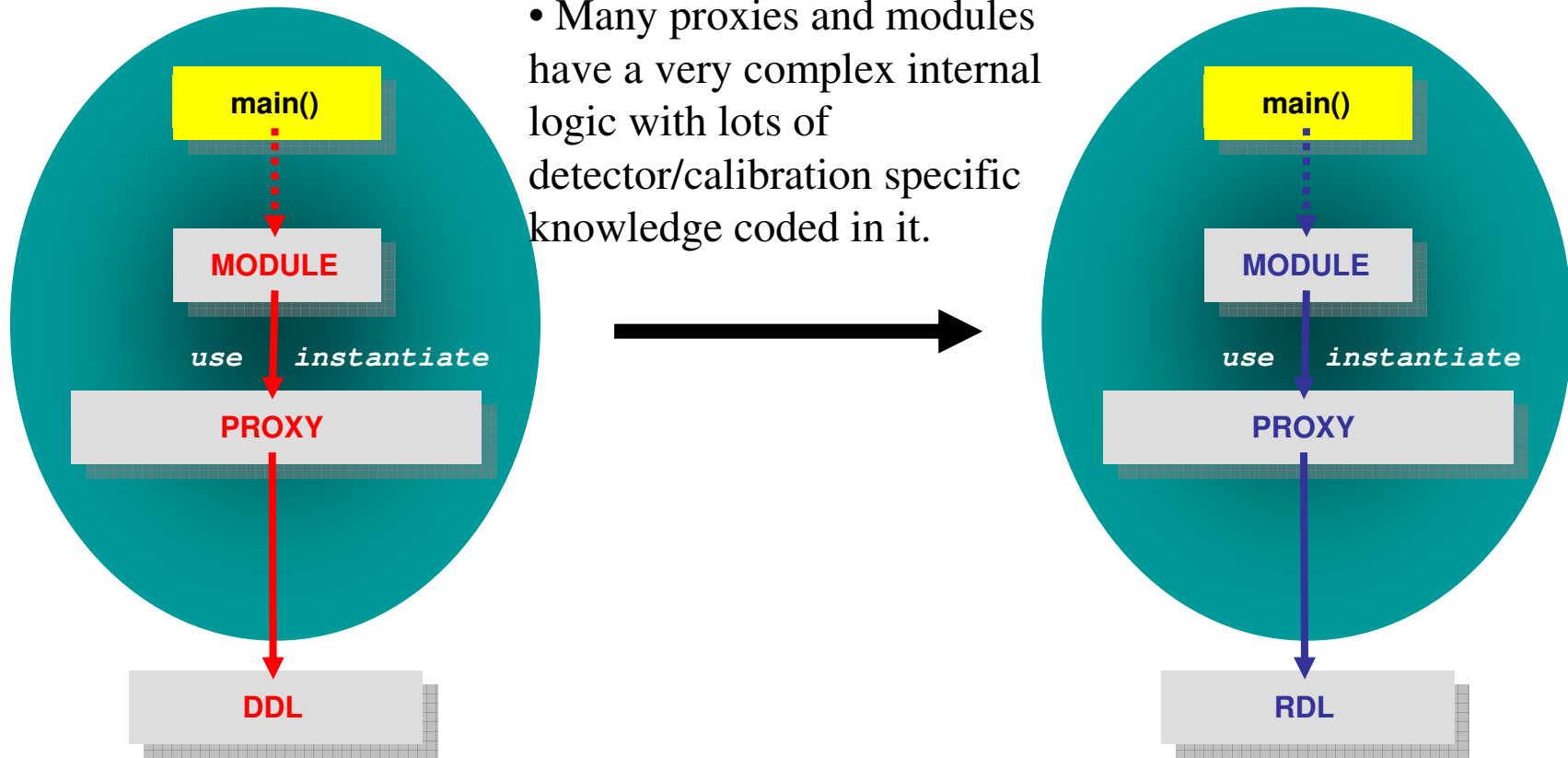
- “Certain restrictions apply”
 - Working on a frozen snapshot of users’ code is **NOT** an option
 - Ideally, the migration must be incremental
 - To facilitate testing
 - Do things in parallel with other developments
 - The BaBar software must be kept functional and free from CDB bugs!
 - We really can **NOT** afford analyzing a dependency diagram for each (hundreds) existing BaBar application:
 - To modifying its code to instantiate tech-specific modules & sequences
 - To maintaining two sets of TCL files (with tech-specific modules & sequences “mixed in”)
 - Both persistent technologies (**DDL & RDL**) should co-exist w/ no code duplication at a level of physics logic

During the transition period

- Hence: do **RE-FACTORING** to make the code ready for the **MIGRATION**
- The general idea is to reorganizing packages to take out technology-specific code and confine it in tech-specific packages:
 - We want to make as little changes to a user-defined logic of applications
 - We want to avoid explicit changes in AppUserBuild-s of applications
 - **We do NOT want functionally rich users’ code to depend on any persistency**
 - Will migrate Framework modules and proxies to the *technology-neutral* interfaces instead
 - Automate an SRT build process for eligible applications to link them against tech-specific packages/libraries

So, What's wrong with this code?

- If we migrate it using an alternative *straightforward* approach then we'll have to *duplicate* and *maintain* two versions of the same code - one per technology

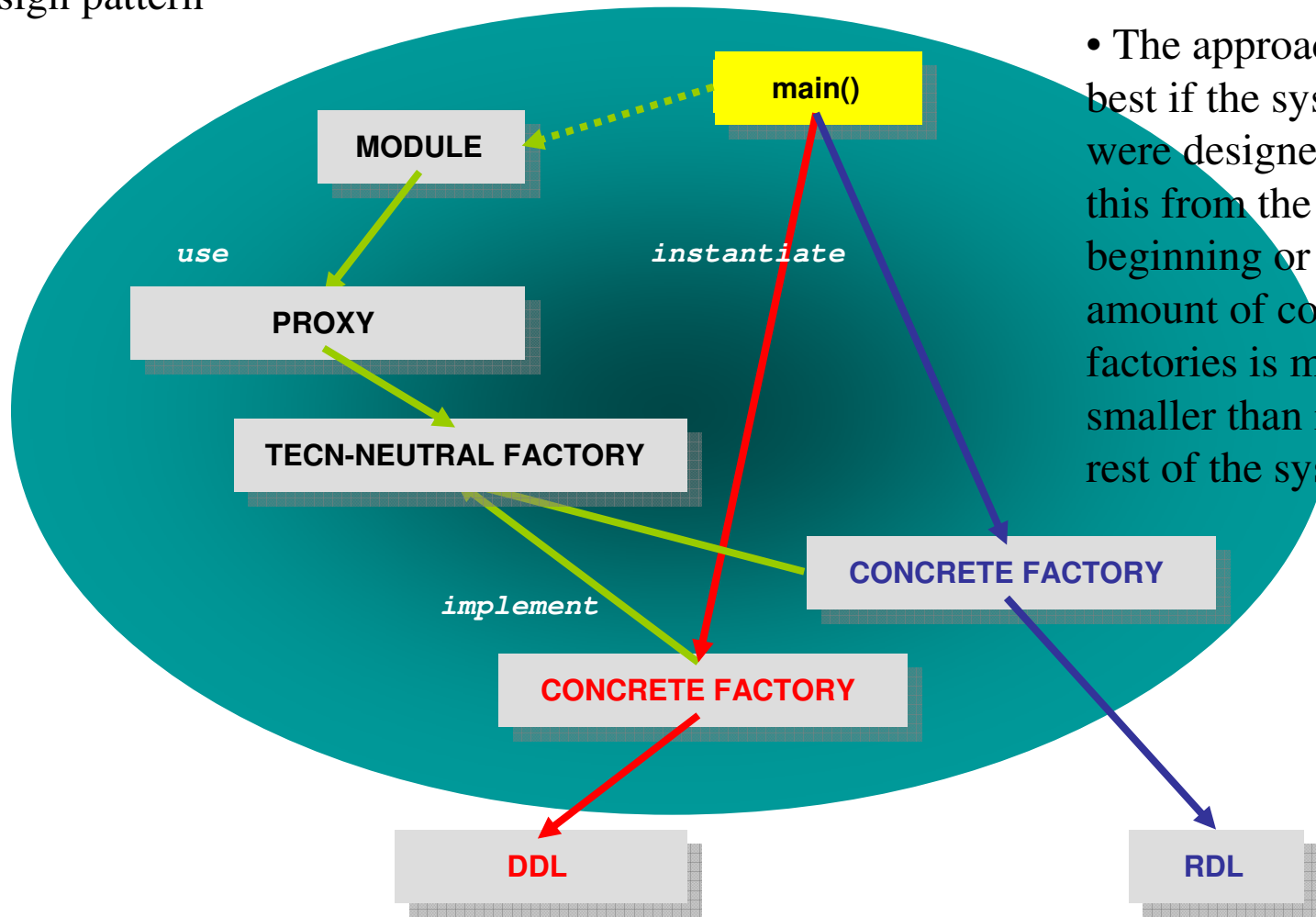


(Problematic) Technology-specific proxies

- **Problems with existing proxies is that:**
 - they *mix* (quite often, very heavily) calibration logic with persistency specific operations
 - So, Framework modules, instantiating the proxies will also depend directly or indirectly onto a persistent technology's API
 - Migrating these proxies means the *evil code duplication*. With all known terrible consequences
 - Unfortunately, we're already facing this problem with the BetaMiniUser application initially migrated to ROOT
- **Here is what each CDB proxy does:**
 - *Manages a transaction (tech-specific operation)*
 - Finds a 'payload' object in CDB (via tech-neutral core CDB API)
 - *Turns this object into a persistent object of the expected (by a particular proxy's context) type and converts the object into a transient one*
 - Caches the transient object along its validity interval in the proxy (to optimize further requests for events falling into the same interval)
 - Delivers the object to a caller
- **In addition, many proxies may have quite complex logic:**
 - To figure out parameters of CDB API requests (from which conditions to fetch objects)
 - Get persistent data from multiple sources and merge results
 - Perform extra calculations/modifications on the loaded data
 - Etc.

What we really want is this

- In OOP is known as “dependency inversion”, and also as the “abstract factory” design pattern



- The approach works best if the system were designed like this from the very beginning or when an amount of code in factories is much smaller than in the rest of the system.

Technology-neutral proxies (solution)

- The ***new model*** of proxies (see a slide with References for a documentation):
 - Instead of:
 - *Managing a transaction using a tech-specific API*
 - It will be:
 - *Doing the same using tech-neutral CDB API*
 - And, instead of:
 - *Turning a found object into a persistent object of the expected (by a particular proxy's context) type and converting the object into a transient one*
 - It would defer this operation to so called:
 - Persistent-to-transient translation service
- The ***translation service***:
 - Has a transient CDB API to a special dictionary of “translators”
 - Uses transient type ID and a persistent type name as keys to find a translator
- A ***translator*** is a specially designed in most cases **_very_ simple transient class provided by a developer of the corresponding persistent class**
-
- A dictionary of translators gets ***automatically*** populated with a set of translators needed for a particular application and a persistent technology
 - This automation is implemented by BaBar SRT and a minor extensions to a design of provider (tech-neutral proxies) and consumer (where persistent classes are) packages
 - See a dedicated slide for details

Benefits of the new proxies

- **Practically no changes to a physics logic of the most proxies**
 - Calibration Toolkit is still a pain
- **Proxies and Framework modules all of a sudden become *tech-neutral***
- **No changes to TCL files!**
- **Can build the same (Framework-based, reading from CDB) application for any technology**
 - Simply by turning to the “left” or to the “right” an SRT switch (**Makefile** variable)
 - Would automatically work for all applications

A price to pay for it

- **Re-factor proxies:**
 - To migrate them to tech-neutral transaction management
 - This happens automatically by replacing a proxy base classes from **CdbBdb/CdbBdbProxyBase** to **CdbBase/CdbProxyBase**, and **CdbBdb/CdbBdbEnvProxy** to **CdbBase/CdbEnvProxy** plus some modifications in user-defined function signatures
 - see documentation on the new proxies for detail
 - In **CdbProxyBase::redefinedFaultHandler()** one would need to defer the persistent-to-transient transformation to a translation service
 - And that's for the most proxies is just a simplification of the proxy's code
- Put one line into the proxy package's **link_<package>.mk** file with a list of transient classes the package needs translators for. For example:

override CDBIMPORT += MyTransientClass

- Implement translators in the corresponding persistent package (details in the documentation) and declare the package as a provider of these translators for the relevant transient classes in a new file (one per package) called **<package>_Bdb.mk** for DDL (replace the suffix with **_Roo** for RDL):

override CDBEXPORT_MyTransientClass = MyPersPackage

Automated Dependency Generation (SRT)

- **Then the application's logic can be written purely and safely in terms of transient classes**
 - No persistent technology-specific Framework modules and/or sequences!
 - No explicit dependencies (through link_*.mk or bin_*.mk) onto
- **The rest will be done by SRT, which will:**
 - Figure out which transient classes are needed by an application
 - Find out persistent provider (of translators) packages for these classes and a specified (or the default) persistent technology
 - Generate a little fragment of extra C++ code (stored into tmp/\$BFARCH/) which would instantiate translators and register them with the translation service
 - Compile & link that extra code to an application. CDB API would automatically invoke that code at its (API's) very first invocation.
- **Potentially, when the dependencies list or the code will get frozen one may consider setting up an explicit dependencies)**
 - **DON'T DO IT NOW! THIS IS JUST A NICE OPTION.**

The Best Design of Persistent Classes

- The best design for persistent classes. Is used in most of existing ‘payload’ classes in BaBar.

```
// transient class
//
class T;

// Persistent class (for Objectivity/DB)
//
class P : public BdbObject {
public:
    P( const T& );
    T* transient() const;
};
```

- Keep the a functionality of a persistent class at a bare minimum.
- The corresponding transient class can be of any complexity.
- This scheme will allow correct dependencies between the corresponding persistent and transient packages.
- It would allow a trivial (using code templates) one-to-one transient-persistent translation for tech-neutral proxies: TRANSLATE<P,T>

What will happen after the re-factoring?

- **When the re-factoring will be done and applications will eventually become “technology-free” then:**
 - **RDL** classes similar to **DDL** ones will have to be developed
 - Ideally, this step has to be going in parallel with the re-factoring
 - Though, we may not afford it in that way to focus on making the bug-free code reformation, and also because no full database services to compare the payload in two different technologies may exist for a period of time
 - **DDL-to-RDL** converters will have to be written as well
 - Instructions will be provided (no formal document yet)
- **Data conversion services will be able to produce CDB snapshots populated with RDL**
 - This would allow to turn the SRT switch to ROOT persistency and run full tests of the migrated applications
 - **Should we consider a mized-persistency applications as well? This may help with the incremental/selective testing.**
- **The loader applications will have to be migrated as well**
 - There is an intention to extend the translation service in an opposite direction (transient-to-persistent) and make loader application technology-neutral as well. This is currently under a study.

Your Help is needed!!!

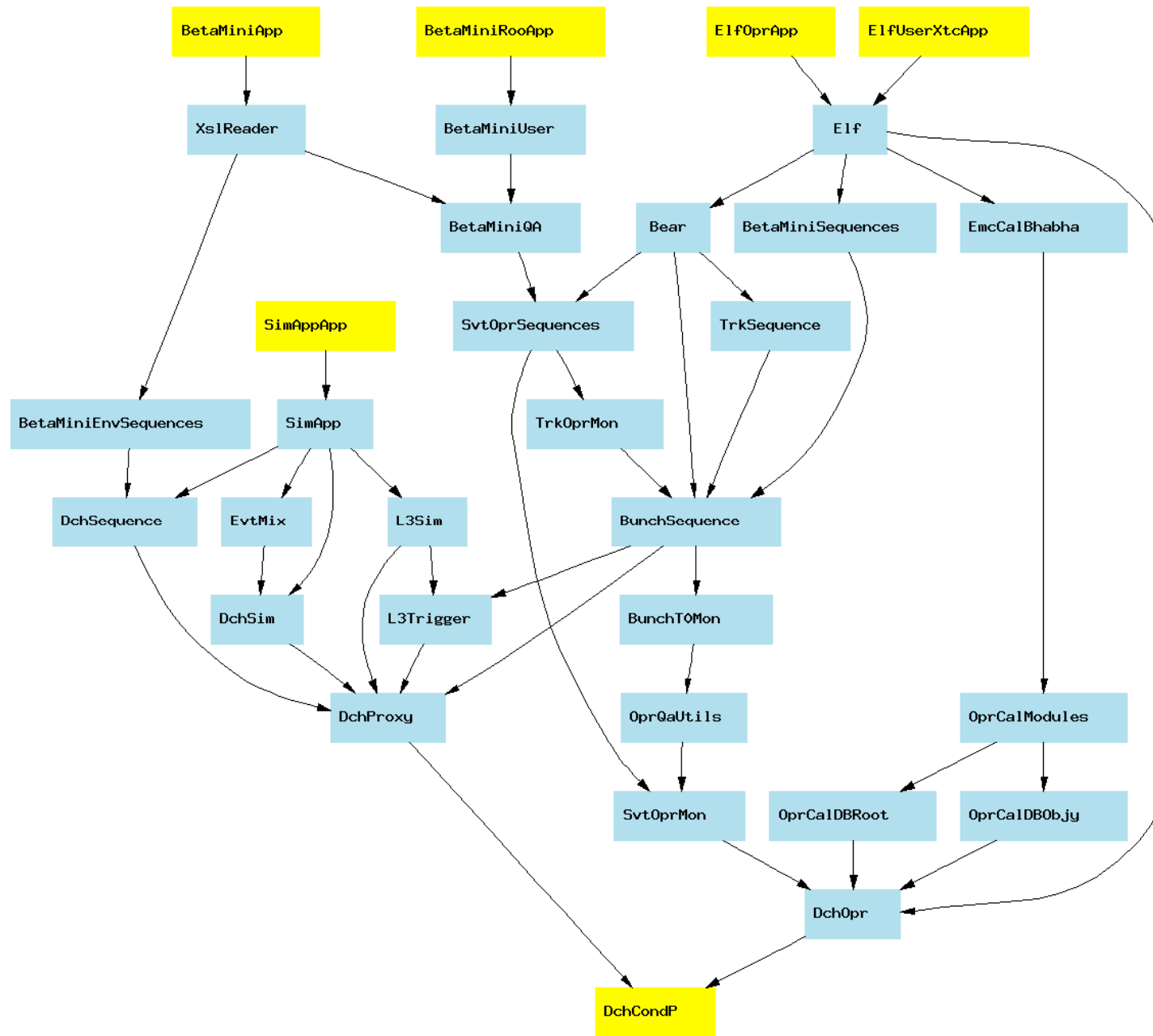
- **It's welcome at various levels, including:**
 - Be cooperative and forgive our mistakes ☺
 - Keep an eye on your packages to see if their functionality isn't broken because of the re-factoring
 - Participate in migrating proxies to tech-neutral model, and in “neutralizing” Framework modules and software packages (which are supposed to be refactored)
 - Participate in migrating persistent classes from DDL to RDL
- **Especially, we need experts in:**
 - **CalDatabase, DCH, DIRC** and **EMC** calibrations!!!
- **We're willing to provide:**
 - A support for your efforts
 - Instructions how to perform the migration
 - Do an initial analysis of your code to suggest how to proceed with the migration in a better way

References

- **HN forums for:**
 - Database migration discussions
 - http://babar-hn.slac.stanford.edu:5090/HyperNews/get/db_alternatives.html
 - Production CDB management
 - <http://babar-hn.slac.stanford.edu:5090/HyperNews/get/condmgmt.html>
 - Objectivity/DB technology and database setups issues
 - <http://babar-hn.slac.stanford.edu:5090/HyperNews/get/BdbSOS.html>
- **Documents:**
 - "Tech-neutral proxies and persistent-to-transient translators"
 - <http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/experts/docTechNeutralProxiesAndPayloadTranslators.html>
 - “Re-factoring/migrating core CDB Framework modules”
 - <http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/experts/CdbTechFreeCDBFwkMods.txt>
 - “Proposed changes for SRT to allow dynamic dependencies”
 - <http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/experts/CdbDynDepsProposal.txt>
 - “CDB API overview”
 - <http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/talks/Igor/CdbApiOverview.pdf>
- **Documents related to a recent attempt (2005) to migrate BetaMiniUser to ROOT**
 - <http://www.slac.stanford.edu/~gapon/CDB/Objy2Root/BetaMini/>
 - <http://www.slac.stanford.edu/~gapon/CDB/Objy2Root/Doc/CdbRTestUtils.README>

Tools: Dependency Analysis

- **GUI-based dependency analysis tools developed for package analysis:**
 - Show a full dependency graph for a given package
 - Ditto for an application
 - Show which applications are relying on this package
 - Show NxM connection network between N applications and M packages
- **Similar tools developed for TCL files and modules analysis**
 - Show a full dependency graph (who includes and uses) a particular module (or modules)
- **Developed together with Akbar Mokhtarani (LBL)**
- **Using GRAPHVIZ (<http://www.graphviz.org/>) to visualize graphs**



More Examples from BaBar

- **A bit more complex package dependency diagram:**
 - http://www.slac.stanford.edu/~gapon/TALKS/2006_Jun_Collaboration_Meeting/MooseApp_BdbCond.png
- **A dependency diagram for configuration files of BaBar Framework based applications:**
 - http://www.slac.stanford.edu/~gapon/TALKS/2006_Jun_Collaboration/mod2tcl_CdbBdbInit.png
 - Comments:
 - BaBar Framework uses the TCL scripting language for that
 - The above shown diagram presents how TCL files depend on some Framework *module*. Modules are specially designed configuration objects Framework application are linked with. TCL provides an interactive interface to these objects. A module shown on the diagram is used to control general parameters of CDB.