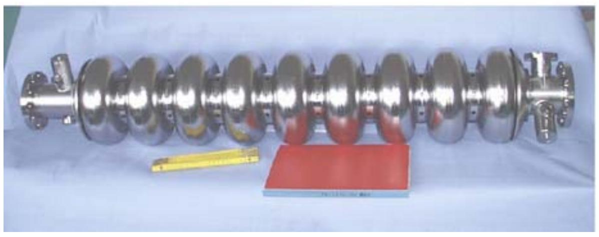
SRF



CARE deliverable report

WP 9 – software management

Calibration Parameters Database

Marcin Wójtowski
Mariusz Grecki
Technical University of Lodz, Department of Microelectronics and Computer Science

08/12/2005

Work supported by the European Community-Research Infrastructure Activity under the FP6 "Structuring the European Research Area" programme (CARE, contract number RII3-CT-2003-506395).

CARE deliverable report

WP 9 – software management

Calibration parameters database

Marcin Wójtowski
Mariusz Grecki

Technical University of Lodz, Department of Microelectronics and Computer Science

08/12/2005

Introduction

The Low Level Radio Frequency (LLRF) subsystem of the VUV-FEL accelerator control system has complex, distributed and networked nature. The accelerating cavities are supplied with RF power from klystrons (Fig.1). The klystrons are controlled by DSP/FPGA based boards that get information about electric field in the accelerating cavities through probes, down converters and Analog to Digital Converters (ADCs). The output signals computed by DSP/FPGA drive klystron through Digital to Analog Converters (DACs) and vector modulator. The DSP/FPGA boards are controlled by VME embedded SUN computers. LLRF subsystem is a part of control system of accelerator and consist of a number of modules. In the Fig.1 only one module of LLRF control system is presented. The control computers communicate with each other and with other systems (e.g. client control panels) through the distributed, networked DOOCS environment. DOOCS is a software C++ library of classes for distributed control system and gives programmers API for writing servers and clients [1]. Clients and servers communicate using properties – collections of values stored in servers. Each property has a defined type. It can be a basic type like string or numeric value and also complex types like structures. Each property has its own network address that allows to read and write to property using C++ API functions. The bottom layer of DOOCS based communication can be Remote Procedure Call (RPC) protocol, share memory, TINE or CA (channel access). DOOCS is in a active development stage however suggestions of deep modifications of DOOCS C++ classes are not welcomed by DOOCS developers.

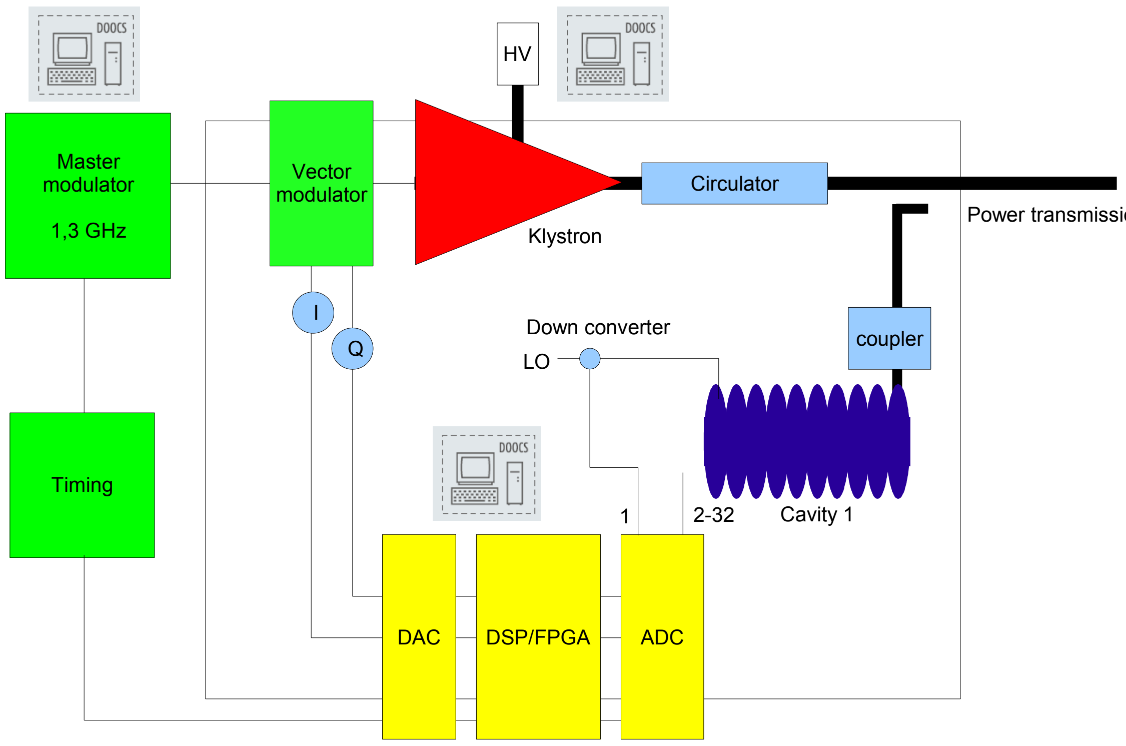


Figure 1 LLRF block diagram

The algorithms realized by DSP/FPGA aim to keep stable (with given accuracy) amplitude and phase of accelerating electric field in the cavity. These computations have to take into account many parameters of the used hardware and their fluctuations in time, temperature etc. Parameters are evaluated by DOOCS servers. Many computations depend on constant values – some of then are calibration factors, other are physical parameters of electric and electronic equipments. Parts like

ADCs, DACs, etc. have to be calibrated and their parameters (gain, offset, nonlinearities etc.) have to be used in computations during system operation. The system calibration parameters are identified in calibration process and stored for future use.
Unfortunately no version of DOOCS (including current) implements a unified method for storage of calibration data. The DOOCS programmers handle this problem using “ad hoc” methods and store calibration data in disk files (without using any standardized format) or as constants in server code. These practice lead to many problems with data consistency (calibration data of the same device stored in different DOOCS servers can be different), data exchange between DOOCS servers (different DOOCS servers can use different format of data storage) and others (even loss of calibration data in the case of server crash). DOOCS also lacks history of the changes of parameters values since DOOCS properties correspond to current value of parameters and cannot store archival data. Data management of many DOOCS properties is complicated – there is no build-in mechanism to do that automatically and all changes have to be inserted and propagated in the system manually. Data sets can be distributed across many DOOCS servers and access can be unacceptable long.

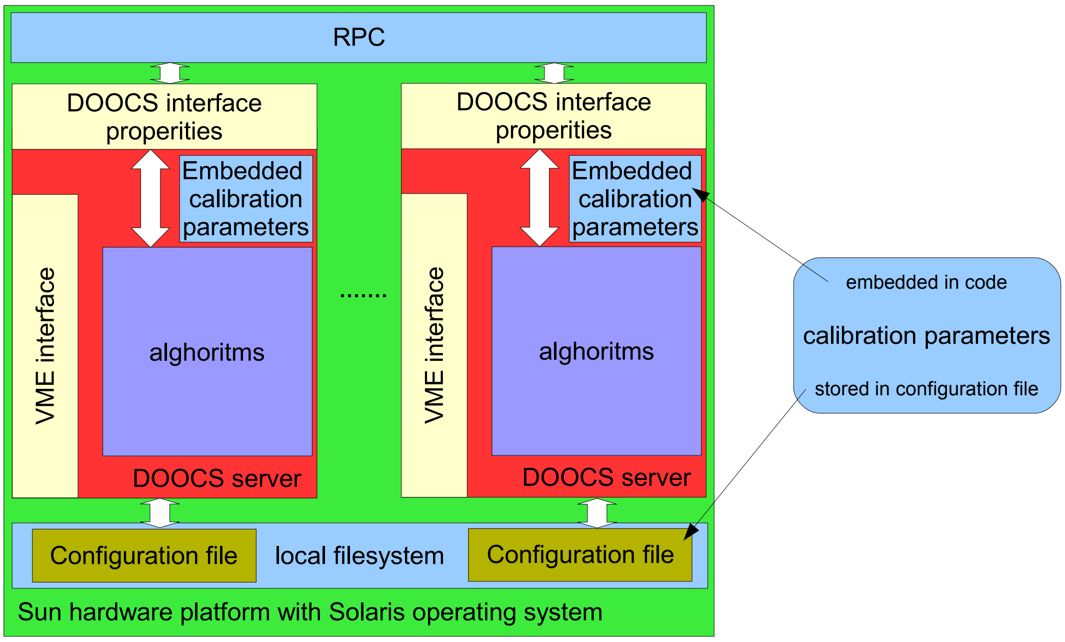

RPC
DOOCS interface properities
Embedded calibration parameters
VME interface
alghoritms
DOOCS server
....
DOOCS interface properities
Embedded calibration parameters
VME interface
alghoritms
DOOCS server
embedded in code
calibration parameters
stored in configuration file
Configuration file
local filesystem
Configuration file
Sun hardware platform with Solaris operating system

Figure 2 Block diagram of DOOCS control system
(calibration parameters stored in configuration files and directly in server code)

Because of lack of unified method for calibration data storage in DOOCS a new system of data storage was proposed (CDB - calibration parameters database). The features of the CDB should overcome all the drawbacks of existing DOOCS system in the area of data storage. Many solutions have been discussed – from local files storage of the parameters up to application of commercial and “open source” database engine. Local files storage has many disadvantages – optimal design of the internal structure of such files and implementation of fast search algorithms is complicated and time consuming. There are also problems connected with scalability.
Finally the application of general purpose database engine has been chosen. Since the data management is well handled in commercial database products it was decided to use commercial database engine and equip it in specialized DOOCS interface. This solution gives benefits of using standardized communication languages – SQL for relation oriented databases and OQL for object oriented databases. There exist many many products having similar functionality but different in many aspects (performance, concurrency, integrity, scalability, reliability etc.). Many of database

engines do not implement all type of queries, others have limitation on database size. Concerning the data model (relational, hierarchical and network) it was decided to use relational one since relation oriented database technology is well established for many years and the SQL language is widely known. Using SQL language it is easy to get various combination of data. Servers do simple queries – get last or previous value of data, but users applications may want to get more complex sets of data. One of the most advanced commercial (and one of the most expensive in the term of license fee) products fulfilling CDB requirements is ORACLE database engine [2]. Fortunately in DESY ORACLE database is used for other purposes and its application in CDB does not generate additional costs.

The first idea to connect DOOCS servers with CDB was to extend C++ DOOCS class and equip C++ API with methods *cdb_get_property* and *cdb_set_property* that gets and sets respectively data values from/in database. This direct implementation of CDB interface into DOOCS class (Fig. 3) is flexible and very efficient but requires recompilation of all DOOCS servers that should be equipped with CDB interface. It is required to add some code (discussed wider in IMPLEMENTATION section that follows) to the DOOCS class API. There are several drawbacks of server recompilation – the most important is the need to extensively test of all the recompiled servers and to check not only whether they behave correctly storing and restoring data in CDB but also whether the CDB interface does not affect other functions. Moreover, the CDB interface is different from typical DOOCS client API and requires programmers to learn how to operate them. After presentation of the features of this solution to DOOCS developers they strongly persuaded CDB developers to try other methods.

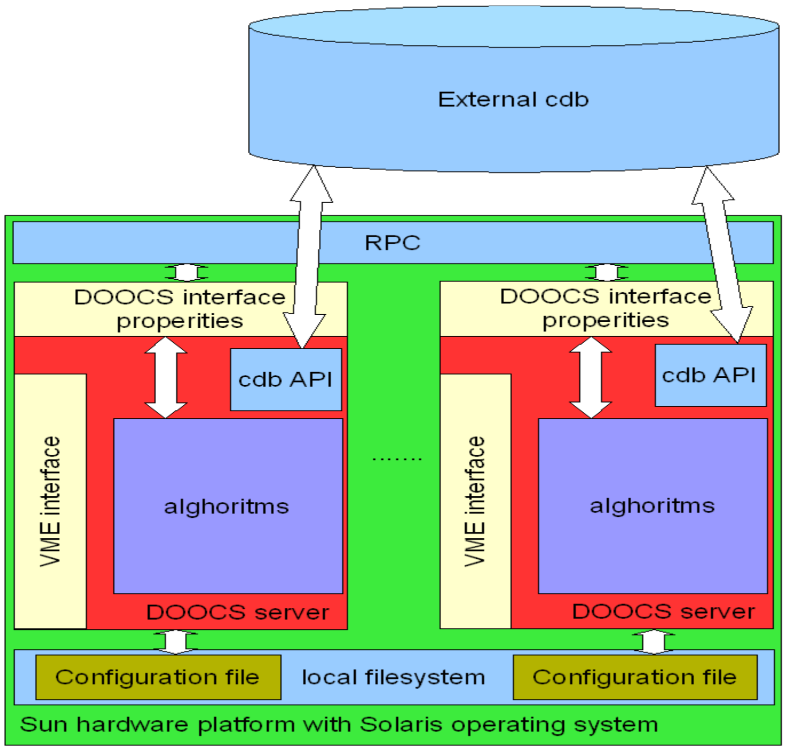


Figure 3 Block diagram of DOOCS control system with direct API connection to CDB

After discussions with DOOCS development team it was decided to implement DOOCS interface to CDB using external application. This method does not require modifications of server code. The external application reads and writes calibration data using DOOCS properties mechanism. This methods has also some drawbacks – the most important is that it is not possible to store in CDB data not exported as DOOCS properties (such data are not visible to external application). As a result, data not related to any servers (ex. cable attenuations) can not be transferred through CDB interface. The big advantage of this method is that it does not require recompilation and testing of existing DOOCS servers. Only the interfacing application has to be extensively tested but this demands much less work than testing many DOOCS servers. This way of CDB interface operation was agreed to implement.

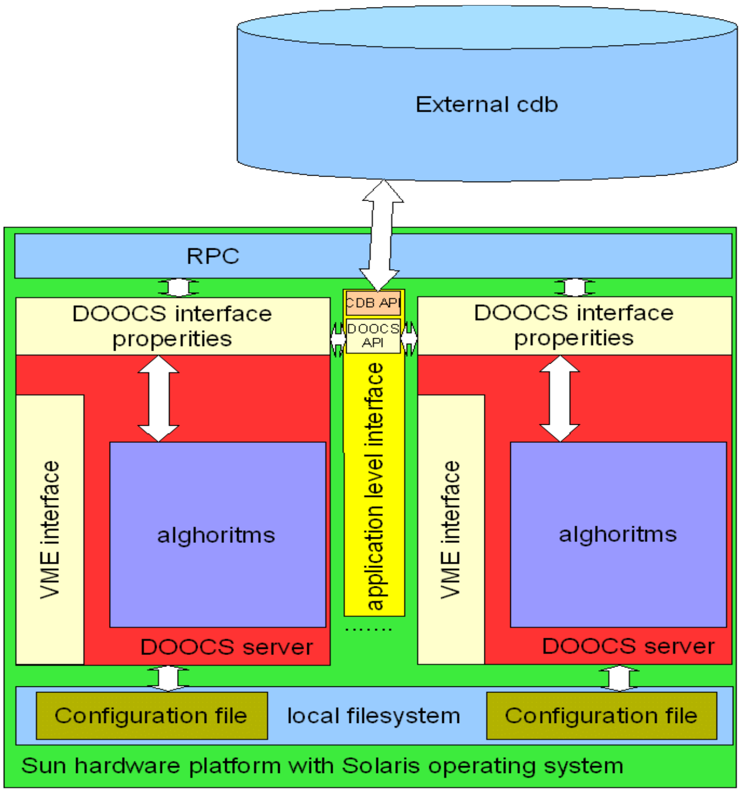


Figure 4 Software part of control system with indirect cdb connection by external application

Main prerequisites and requirements for the CDB and DOOCS-CDB interface are listed below:

- The database to store all calibration parameters from DOOCS system is needed.
- History of modification of calibration data should be available.
- Database interface should be compatible with DOOCS.
- Database should be easy to maintain – new data types may be added as required.
- The data access time should be short.
- Database should be reliable.
- CDB interfaces (API) to programing language (C/C++, Java) should be developed as well as example applications.
- Data addressing scheme should correspond to the one used in DOOCS system.
- Implementation of database should be as simple as possible.

Implementation

Database structure

To match many of CDB requirements a commercial database engine (ORACLE) has been chosen. It implements mechanisms easily allowing to scale size of data, add indexes, have fast access to data, restrict access to data, do backup and more. Fortunately it is also used in DESY for other purposes so it has good support and its application does not require to invest money.

The structure of database has been divided into four tables:
structure_tbl - stores connections between devices (objects) and their locations. Each record stores logical address of device, name (serial number) of device, operation made on location and timestamp of the modification. This table stores current and archival data, however by default only newest record with given location is restored on request.
objects_tbl –stores objects definitions. Records in other tables refer always to records in that one. The *structure_tbl* and *values_tbl* refer to this table using serial number of the device (object) and timestamp respectively. This mechanism allows changing whole element at specific location without rewriting all device properties. It means that operator may prepare new device parameters and then assign the device to the location in one step. This table stores object serial number, type of object, operation made on object and timestamp.
values_tbl – stores values of devices properties. Records are built from names of properties, numerical or text values, timestamp of object which value refers to, factor of numerical value (not used for now), unit (not used for now), type of data stored (int as DOOCS), operation made on value and its own timestamp.
access_history_tbl – stores information about access to the data. Each record consists location of object, name of property, client (program) location, operation which is made on data (read/write) and timestamp.

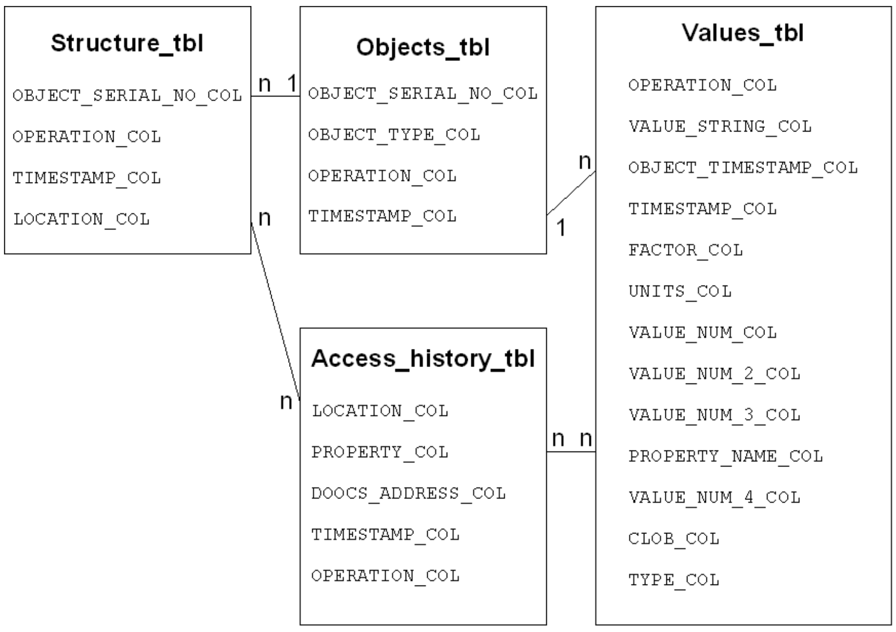


Figure 5: Database structure with relations

Detailed tables structure is included in APPENDIX C.

All tables and columns have `_tbl` and `_col` suffixes respectively. In each table exists field *operation_col* which informs whether record is added or removed (it can take values 'add' and 'del'). This allows to keep history of data insertion and removal.
There are two reference fields (the `structure_tbl.object_serial_no` is a reference to `objects_tbl.object_serial_no`, and `values_tbl.object_timestamp_col` is a reference to `objects_tbl.timestamp_col`) that match records from one table to another. First reference matches record in *structure_tbl* having particular *location_col* value with corresponding object in *object_tbl*. It is possible to have many objects with the same serial number in *objects_tbl* for archival purposes. Reference connects only most recent record from *objects_tbl* having the same *object_serial_no* as selected record in *structure_tbl*. The reference between *values_tbl* and *objects_tbl* cannot work in the same way because values from *values_tbl* are assigned to concrete instance of object. In order to establish this reference the unique field (timestamp) is used. All timestamps field are unique in whole database.

Usage:
In order to put new property for device which does not exits in database first create new object in *objects_tbl* and assign unique serial number of the device. The same should be performed for non-existing location (create new object in *structure_tbl* with the same serial number). To move device from one location to another it is needed to add two records for the same object serial number to *structure_tbl* – one with the old location name and operation 'del', second with the new location and operation 'add'. To override location with the new object add new record with the same location name but different object serial number.

Example.
To change value of output offset parameter of ADC located at address
`'TTF2.RF/ADC/GUN1.SCOPE1/CH0.OFFSET'` it is needed to:
1. Select latest record in *structure_tbl* with *location_col* equal to `'TTF2.RF/ADC/GUN1.SCOPE1'`
2. Read value of *object_serial_no* from selected record
3. Select latest record in *objects_tbl* with corresponding *object_serial_no* read from *structure_tbl*
4. Read *timestamp_col* from this record
5. Add new record to *values_tbl* with *property_name_col* set to 'CH0.OFFSET', *value_num_col* set to new value, *type_col* set to 2 (means DATA_FLOAT as defined in *pvak_types.h* header file in DOOCS include path), operation set to 'add' and *object_timestamp_col* set to value got in step 4.

Notice:
Currently CDB can handle DATA_INT (value_num_col), DATA_FLOAT (value_num_col), DATA_BOOL (value_num_col), DATA_STRING (value_string_col), DATA_STRING16 (value_string_col), DATA_XML (value_string_col), DATA_IFFF (value_num_col, value_num_2_col, value_num_3_col, value_num_4_col), DATA_IIII (value_num_col, value_num_2_col, value_num_3_col, value_num_4_col) , DATA_TTII (value_num_col, value_num_2_col, value_num_3_col, value_num_4_col) types.
New datatype for storing matrices of floats has been proposed to DOOCS developers.

Model of data flow

To access the CDB data application has to use one of available interfaces – programming language API or execute other application which implements such API. Dependencies between CDB, programing languages API and applications are shown on Fig 6. Currently the interfaces for C++ and Java are worked out and implemented. The C++ interface is used in *t_db_doocs* application (used to store/restore DOOCS properties in CDB). Java interface can be used by Matlab applications.

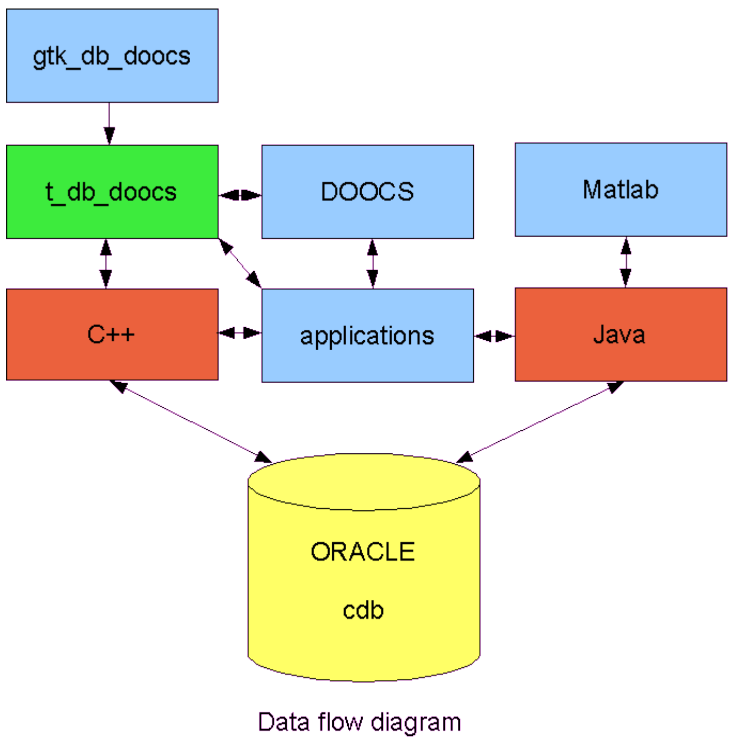


Figure 6: Relations between CDB and other system components

Orange boxes represent interfaces to programming language, green high level batch application and blue represents other applications (e.g. GUI for *t_db_doocs*).

Programming language interfaces

C++ interface

Notice:
All source codes can be obtained from DESY's CVS.

C++ interface has only one user class – `cdb`. To store data use method `set` of the object *cdb*. To get data from database use `get` method of the same object. Their declarations are presented in figure 7.

```
int get(EqAdr *, const EqData *const src, EqData *dst, unsigned history=0);

int set(EqAdr *, EqData *const src, const EqData *const dst);
```

Figure 7: The cdb class public methods for store/restore data

These methods are defined in *cdb.h* header file (appendix D). Their prototypes and method of use are almost the same as in *EqCall* DOOCS object. Functions return 1 if function passes or 0 if fails. Last parameter of `get` method determines which stored archival data should be retrieved (0 means current, 1 - second newest, 2 – third newest etc.). The CDB directory includes object file *cdb.o* which should be linked with compiled program. Constructor of the `cdb` class requires name of the client program. So, to add database storage functionality to application (e.g. DOOCS server/client) it is required to perform following steps:

1. Add

```
#include "cdb.h"
```

   at the begin of program.

2. Put

```
Cdb _cdb("my_program");
```

   to initiate using of cdb object

3. Define other required objects

```
EqData src, dst;
EqAdr ea;
```

4. Put

```
double val;
ea.adr("ttf2.diag/dose/und1.1l/dose.mean");
_cdb.get (&ea,&src,&dst);
```

   to get value of *dose.mean* property from location '`ttf2.diag/dose/und1.1l`'.

5. Put

```
double val;
ea.adr("ttf2.diag/dose/und1.1l/dose.mean");
src.set_type(DATA_INT);
src.set(1);
_cdb_set(&ea,%src,&dst);
```

   to put value of *dose.mean* property from location '`ttf2.diag/dose/und1.1l`' to database.

6. Add '–I' option with location of *cdb* header file as one of compiler's parameters. Example Makefile is included in appendix A.

7. Add '–L' option with location of *cdb* object as one of linker's parameters as well as *cdb.o* file to link program with. Example Makefile is included in appendix B. It may be necessary to add additional options like '`-L/opt2/oracle/db_client9i/lib -locci9 -lclntsh`' to use Oracle library (for DESY Solaris system).

8. Set ORACLE_HOME environmental variable to '`/opt2/oracle/db_client9i/`' (for DESY Solaris system)

Application layer interface

The base functions of data management (comparison of data stored in database against values of DOOCS properties, data synchronization between database and DOOCS) is realized by the *t_db_doocs* application. This program can be run directly or through *t_db_doocs.sh* wrapper that additionally sets few environmental variables before executing *t_db_doocs* (these variables may be required to load libraries etc.). The *t_db_doocs* is batch application and can be used as part of other programs - it reads data from command line and input file and write results to standard output. Before execution this application the file containing locations (network addresses) of managed DOOCS properties should be prepared. It may be done manually or by other program like *gtk_db_doocs* (GUI front-end to *t_db_doocs*). The *t_db_doocs a*pplication requires parameters determining the mode of operation (*c* -compare, *f* -data transfer CDB->DOOCS, *t* -data transfer DOOCS->CDB) and the the name of file containing locations. The –h parameter displays help on *t_db_doocs* (the same result generates running program without any parameter). Standard input may be used instead of file if '–' is given as filename. This feature allows to connect input/output streams of parent/child processes and to use *t_db_doocs* in other programs.

Example.
```
t_db_doocs.sh -c ./locations.txt
```
This example compares data from DOOCS properties and corresponding CDB values. The list of DOOCS properties to compare is contained in *./locations.txt* file. Results of comparisons are written to standard output or to to file using UNIX stream redirection mechanism. The source file and compiled code of *t_db_doocs* application is located in *cdb* directory.

GUI application

The *t_db_doocs* can be used also through front-end GUI *gtk_db_doocs* application. It allows to select locations just by clicking them as well as choose the operation mode from X Windows panel. The *gtk_db_doocs* application generates the locations file automatically and calls *t_db_doocs* with appropriate command line. *Gtk_db_doocs* requires few libs which sometimes cannot be found directly. In this case instead of executing *gtk_db_doocs* the wrapper *gtk_db_doocs.sh* (which sets needed environmental variables) can be called. The schematic diagram of the relationship and communication paths between *gtk_db_doocs, t_db_doocs,* CDB and DOOCS properties is presented in Fig. 8. The source file and compiled code of *gtk_db_doocs* application is located in *cdb* directory.

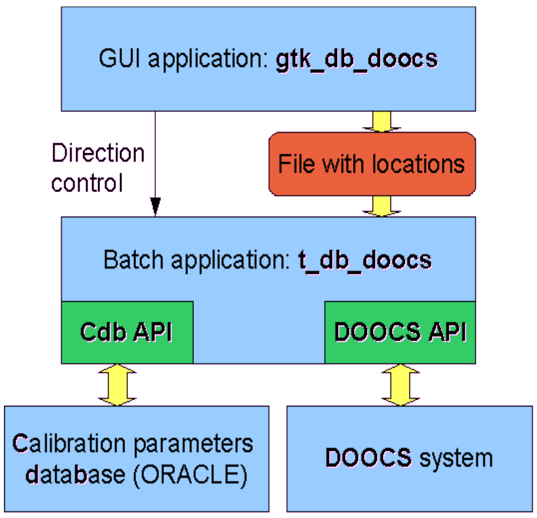
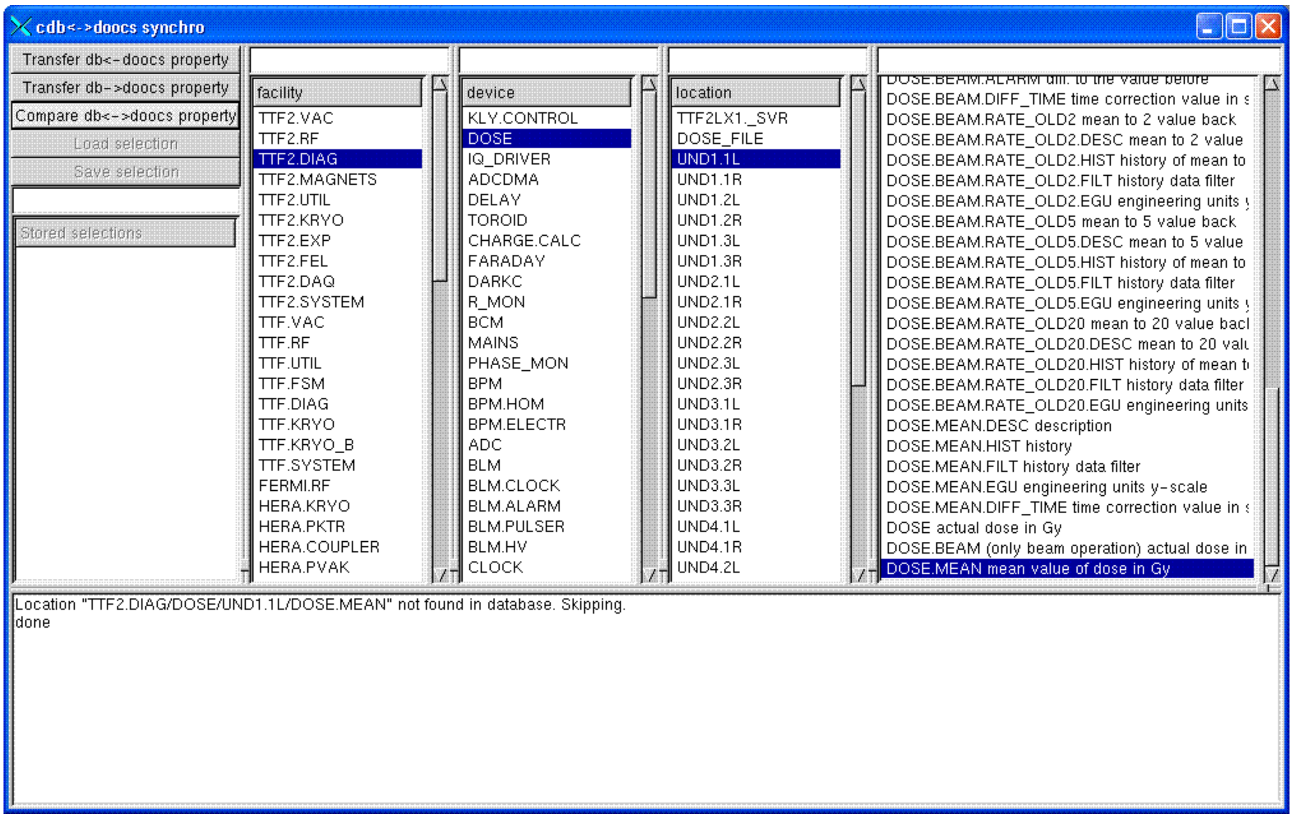

GUI application: gtk_db_doocs
Direction control
File with locations
Batch application: t_db_doocs
Cdb API
DOOCS API
Calibration parameters database (ORACLE)
DOOCS system

Figure 8: Relationship and communication paths between gtk_db_doocs, t_db_doocs, CDB and DOOCS properies

Figure 9 presents snapshot of *gtk_db_doocs* application. On the left side of the window there are buttons corresponding the various actions. Other columns allows to select the DOOCS properties. Bottom part of the panel is dedicated to output results from *t_db_doocs* application performing selected operation.


cdb<->doocs synchro
Transfer db<-doocs property
Transfer db->doocs property
Compare db<->doocs property
Load selection
Save selection
Stored selections

```
facility          device         location
TTF2.VAC          KLY.CONTROL    TTF2LX1._SVR
TTF2.RF           DOSE           DOSE_FILE
TTF2.DIAG         IQ_DRIVER      UND1.1L
TTF2.MAGNETS      ADCDMA         UND1.1R
TTF2.UTIL         DELAY          UND1.2L
TTF2.KRYO         TOROID         UND1.2R
TTF2.EXP          CHARGE.CALC    UND1.3L
TTF2.FEL          FARADAY        UND1.3R
TTF2.DAQ          DARKC          UND2.1L
TTF2.SYSTEM       R_MON          UND2.1R
TTF.VAC           BCM            UND2.2L
TTF.RF            MAINS          UND2.2R
TTF.UTIL          PHASE_MON      UND2.3L
TTF.FSM           BPM            UND2.3R
TTF.DIAG          BPM.HOM        UND3.1L
TTF.KRYO          BPM.ELECTR     UND3.1R
TTF.KRYO_B        ADC            UND3.2L
TTF.SYSTEM        BLM            UND3.2R
FERMI.RF          BLM.CLOCK      UND3.3L
HERA.KRYO         BLM.ALARM      UND3.3R
HERA.PKTR         BLM.PULSER     UND4.1L
HERA.COUPLER      BLM.HV         UND4.1R
HERA.PVAK         CLOCK          UND4.2L
```

```
DOSE.BEAM.ALARM diff. to the value before
DOSE.BEAM.DIFF_TIME time correction value in s
DOSE.BEAM.RATE_OLD2 mean to 2 value back
DOSE.BEAM.RATE_OLD2.DESC mean to 2 value
DOSE.BEAM.RATE_OLD2.HIST history of mean to
DOSE.BEAM.RATE_OLD2.FILT history data filter
DOSE.BEAM.RATE_OLD2.EGU engineering units y
DOSE.BEAM.RATE_OLD5 mean to 5 value back
DOSE.BEAM.RATE_OLD5.DESC mean to 5 value
DOSE.BEAM.RATE_OLD5.HIST history of mean to
DOSE.BEAM.RATE_OLD5.FILT history data filter
DOSE.BEAM.RATE_OLD5.EGU engineering units y
DOSE.BEAM.RATE_OLD20 mean to 20 value bacl
DOSE.BEAM.RATE_OLD20.DESC mean to 20 valu
DOSE.BEAM.RATE_OLD20.HIST history of mean t
DOSE.BEAM.RATE_OLD20.FILT history data filter
DOSE.BEAM.RATE_OLD20.EGU engineering units
DOSE.MEAN.DESC description
DOSE.MEAN.HIST history
DOSE.MEAN.FILT history data filter
DOSE.MEAN.EGU engineering units y-scale
DOSE.MEAN.DIFF_TIME time correction value in s
DOSE actual dose in Gy
DOSE.BEAM (only beam operation) actual dose in
DOSE.MEAN mean value of dose in Gy
```

```
Location "TTF2.DIAG/DOSE/UND1.1L/DOSE.MEAN" not found in database. Skipping.
done
```

Figure 9: Screenshot from gtk_db_doocs application

Potential future improvements of CDB

- Implementation of access restrictions (protects against accident modification of data caused by human fault)
- Design of new application to manage database records (define new objects, move objects to locations)
- GUI improvements (sort, multi choice, storage/readout of selected locations to/from file)
- Implementation of new data type: table of floats (this data type is requested by the DOOCS users but still not defined in DOOCS API)
- Visualization of data relationships between calibration parameters and servers (presentation in graphical diagram which server uses/modifies which parameter)

Bibliography

[1] DOOCS Home Page, http://tesla.desy.de/doocs/doocs.html
[2] ORACLE Database, http://www.oracle.com/database/index.html

Appendix A

example Makefile of `cdb` project (located in *cdb* DESY CVS directory)

```
CXX=CC
LIBS=
CXXFLAGS=-g -I/opt2/oracle/db_client9i/rdbms/demo/ -I/opt2/oracle/db_client9i/rdbms/public -I
/doocs/doocssvr1/MAIN/Epics-R3.13.1/base/include -I/doocs/doocssvr1/lib/include -DDEBUG
LDFLAGS=-g -L/opt2/oracle/db_client9i/lib -locci9 -lclntsh -lTTFapi -L/doocs/doocssvr1/lib
#-dn gives static linking
all: cdb

main.o: main.cpp cdb.h
	$(CXX) $(CXXFLAGS) -c main.cpp
cdb.o: cdb.cpp cdb.h
	$(CXX) $(CXXFLAGS) -c cdb.cpp

cdb: main.o cdb.o
	$(CXX) cdb.o main.o -o cdb $(LDFLAGS)

cdb_eq_client.o: cdb_eq_client.cpp cdb_eq_client.h
	$(CXX) $(CXXFLAGS) -c cdb_eq_client.cpp

clean:
	rm *.o cdb
```

Appendix B

example Makefile of *t_db_doocs* and *gtk_db_doocs* project (located in *gtk_db_doocs* DESY CVS directory)

```
#DOOCSROOT=/doocs/doocssvr1
#DOOCSARCH=solaris2
#include $(DOOCSROOT)/doocs-versions/$(DOOCSARCH)/CONFIG
#include $(DOOCSROOT)/$(DOOCSARCH)/DEFINEDOOCSROOT

CXXFLAGS=-g -I/opt2/sbin/local/sfw/lib/glib/include/ -I/opt2/sbin/local/sfw/include/ -I
/usr/local/include/glib-1.2 -I/home/mwojtow/cdb/ -I/opt2/oracle/db_client9i/rdbms/public -I
/opt2/oracle/db_client9i/rdbms/demo/ -I/doocs/doocssvr1/lib/include -I/doocs/doocssvr1/MAIN/Epics-
R3.13.1/base/include -I/doocs/doocssvr1/lib/include/xalanc -I/doocs/doocssvr1/lib/include/xercesc
LDFLAGS=-lrt -mt -lgtk -L/usr/local/lib -lgdk -lglib -lTTFapi -g -L/doocs/doocssvr1/lib
LDFLAGS_NOGUI=-mt -g -L/opt2/oracle/db_client9i/lib -L/doocs/doocssvr1/lib -lTTFapi -locci9 -lclntsh
gtk_db_doocs: gtk_db_doocs.o
	CC -o gtk_db_doocs gtk_db_doocs.o ${LDFLAGS}
	chmod u+x gtk_db_doocs
gtk_db_doocs.o: gtk_db_doocs.cpp
	CC -c gtk_db_doocs.cpp ${CXXFLAGS}
get_property: get_property.o
	CC -o get_property get_property.o ${LDFLAGS}
get_property.o: get_property.cpp
	CC -c get_property.cpp ${CXXFLAGS}
gtk_clist_ex: gtk_clist_ex.o
	CC -o gtk_clist_ex gtk_clist_ex.o ${LDFLAGS}
gtk_clist_ex.o: gtk_clist_ex.cpp
	CC -c gtk_clist_ex.cpp ${CXXFLAGS}
t_db_doocs: t_db_doocs.o ../cdb/cdb.o
	CC -o t_db_doocs t_db_doocs.o ../cdb/cdb.o ${LDFLAGS_NOGUI}
t_db_doocs.o: t_db_doocs.cpp
	CC -c t_db_doocs.cpp ${CXXFLAGS}
clean:
	rm *.o gtk_db_doocs
```

APPENDIX C

Database structure of CDB

```
SQL> desc structure_tbl;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 LOCATION_COL                                       VARCHAR2(255)
 OBJECT_SERIAL_NO_COL                               VARCHAR2(50)
 OPERATION_COL                                      VARCHAR2(30)
 TIMESTAMP_COL                             NOT NULL TIMESTAMP(6)

SQL> desc objects_tbl;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 OBJECT_SERIAL_NO_COL                               VARCHAR2(50)
 OBJECT_TYPE_COL                                    VARCHAR2(255)
 OPERATION_COL                                      VARCHAR2(255)
 TIMESTAMP_COL                             NOT NULL TIMESTAMP(6)

SQL> desc values_tbl;
Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PROPERTY_NAME_COL                                  VARCHAR2(50)
 VALUE_STRING_COL                                   VARCHAR2(4000)
 OBJECT_TIMESTAMP_COL                               TIMESTAMP(6)
 TIMESTAMP_COL                                      TIMESTAMP(6)
 FACTOR_COL                                         NUMBER
 UNITS_COL                                          VARCHAR2(30)
 OPERATION_COL                                      VARCHAR2(30)
 VALUE_NUM_COL                                      NUMBER
 VALUE_NUM_2_COL                                    NUMBER
 VALUE_NUM_3_COL                                    NUMBER
 VALUE_NUM_4_COL                                    NUMBER
 CLOB_COL                                           CLOB
 TYPE_COL                                           NUMBER

SQL> desc access_history_tbl;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 LOCATION_COL                                       VARCHAR2(255)
 PROPERTY_COL                                       VARCHAR2(255)
 DOOCS_ADDRESS_COL                                  VARCHAR2(255)
 TIMESTAMP_COL                                      TIMESTAMP(6)
 OPERATION_COL                                      VARCHAR2(10)
```

Appendix D

contents of *cdb.h* header file

```
#ifndef __cdb_h
#define __cdb_h
#include <string>
#include <vector>
#include <occi.h>
#include <eq_client.h>
using namespace oracle::occi;
using namespace std;
class cdb
{
  Environment *env;
  Connection *conn;
  string doocs_address;    //client identification
  int database_reconnect();
  int save_to_access_history(string location, string property, bool read);
  bool set_property(string location, string property, bool is_double, double dvalue, string svalue);
  int get_property(string location, string property, double *ret_double, string *ret_string);
  public:
    string get_string_property(string location, string property=NULL);
    double get_double_property(string location, string property=NULL);
    int get(EqAdr *, const EqData *const src, EqData *dst, unsigned history=0);
    // parameter history says which historical data you want to take
    // 0 means present, 1 previous, 2 previous previous ....
    bool set_property(string location, string property, double value);
    bool set_property(string location, string property, string value);
    int set(EqAdr *, EqData *const src, const EqData *const dst);
    vector<string> get_locations_list();
    cdb(string doocs_address);
    ~cdb();
};
#endif
```