Customer: Kjell Lemström Instructor: Anita Kasari

OT4: pics99

Design Specification and Implementation of Corrections

Timo Immonen
Jussi Lindgren
Sampo Pyysalo

Helsinki March 31, 1999

Design Specification and Implementation of Corrections
Software Engineering Project
UNIVERSITY OF HELSINKI
Department of Computer Science

Contents

Version Number	Comments	Date/Author
0.1	document started	11.3.99 /TI
0.2	added secs 1,2 and 3	16.3.99 /TI
0.3	added sec 5	17.3.99 /JL
0.4	added sec 7	17.3.99 /JL
0.5	added sec db, comm and host	17.3.99 /SP
0.6	moved gui c++ to sec 5.5	18.3.99 /TI
0.7	added sec comm spec.	25.3.99 /SP
1.1	inspected and corrected	31.3.99 /TI

1 Introduction (TI)

PICSearch is a development tool for content-based image retrieval algorithms. It is aimed to serve as a testing platform for new retrieval algorithms. It can also be used as a conventional image cataloging program.

The initial version PICSearch was developed at Helsinki University of Technology. First version, **PICSearch v1.0**[1], was restricted to use a local image database only. Further development at University of Helsinki resulted in **PICSearch v2.0**[2], a version enhanced with rudimentary network capabilities. A third version of program was also developed at University of Helsinki. **PICSearch v3.0**[3] added a cache implementation for greater network efficiency and also featured an improved graphical user interface.

This document will introduce program changes and code corrections made during the first phase of the pic99 project. It also contains the design specifications for the second phase of the project, namely the plans to implement an independent search algorithm entry method and to replace existing database engine with a more capable free source implementation.

In section 2 PICSearch program is described. Section 3 addresses modifications made to the graphical user interface and the client software. In section 4 changes to the internal communication of the PICSearch software are described along with modifications to the server and host modules. Various removed unused code sections are listed in section 5. Sections 6 and 7 contain the design specifications of the algorithm entry module and the database module respectively. Finally, section 8 describes changes to the user interface.

2 Overview of the PICSearch Program (TI)

PICSearch program is structured in four distinct operational modules: The client software contains a graphical user interface. The host is the central point of the server side of the PICSearch. The server is an independent service provider process. One server is created by the host process per a connected client process. The database is a separate module which provides both servers and the host process with a database interface services (see Figure 1).

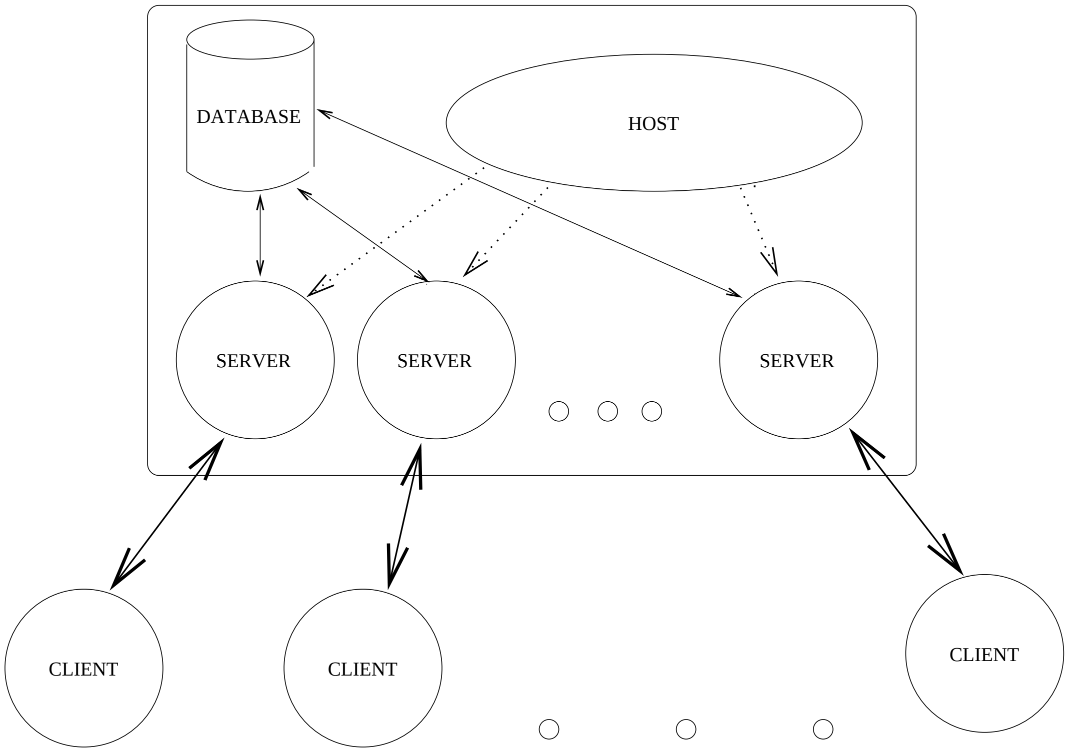

Figure 1: PICSearch fuctional modules (after client connection).

Normal operation of the PICSearch consists of a host process running on a server system. A user wishing to utilize PICSearch system starts a client front end of the PICSearch on his/her local, network capable workstation. On server side the host process monitors a well-known port address, and creates a socket connection when it receives client's request. It then creates a server process and assign it to the socket. Server process communicates with the connected client module and implements queries and other image database transactions. It also handles various system management activities such as adding a new search algorithm or a user account. A client package is also used as a PICSearch system console, and local connections with host/server and the client running all on a same system are supported.

The PICSearch system is implemented with C++ and Tcl/Tk for a Linux / UNIX platform.

Readers are advised to consult design requirements of previous projects [4] [5] [6], if specific implementation choices or general design questions need to be clarified.

3 Improvements for PICSearch GUI (TI)

The graphical user interface (GUI) of the PICSearch program is located in the client module. It is implemented with Tcl/Tk script language. Functionality of the user interface is mainly implemented in an accompanying C++ module. This module communicates with the Tcl/Tk interface scripts by setting a state variable. This variable is then monitored by Tcl/Tk event waiting. The interface initiates various actions made by the user by calling a single C++ function with appropriate arguments.

During the first phase of the project various small errors and inconsistencies of the graphical user interface were corrected. No major re-programming took place.

3.1 Window Operation Corrections

In **PICSearch v3.0** several independent windows of the user interface were faulty. They caused a Tcl/Tk runtime error when selected again from menu, while already open. This could lead to inconsistent internal state and was confusing to the user. All windows were corrected to raise to the foreground when re-selected while already open.

Some windows were made modal, that is, rest of the GUI ceases to respond to user input while a modal window is open. Events generated on the rest of the GUI when a modal window is open are disregarded. Windows which were made modal were considered sensitive and important enough to warrant focusing user actions on them.

3.2 Selection Operation Corrections

File opening dialog of the open query image -operation was enhanced with a ready-made selection "All Image Files" which includes typical file name extensions of all supported file formats.

In **PICSearch v3.0** a most recently used servers -quick selection list was faulty. It recorded single connection to a local server as two separate entries. This was caused by erraneous spelling of a text label in client module, and has been corrected. The quick selection was also unrestricted. Server name was always added to the selection list regardless of whether it was already contained in the list. This was corrected and list is now restricted to a one entry per server. The list is not reordered, i.e. top of the list is not necessarily the most recently used server, it may be elsewhere in the list.

3.3 Removal of Unneeded Code

Several small segments of unneeded Tcl/Tk code were removed. These were typically debug -oriented code sections, and responses to non-existent interface features, presumably left over from previous versions of the interface. These removals are not marked in the code. They are tested not to affect the operations of the interface.

On one instance unneeded parameter array of a call to the C++ routines was removed and documented in the code. Also several magic numbers hard coded to Tcl/Tk scripts were replaced with constants or made redundant and removed.

4 Changes to the Client-Server Communication (SP)

A major part of the PICSearch program is dedicated to communication between the client and server processes over a network, and implementing this communication simply and efficiently is vital to the maintainability of the project. The current implementation is detailed below.

4.1 Overview of the Communication Implementation

The communication is almost completely implemented using a straightforward synchronous mode where the client issues a request and waits for the server to answer. During normal operation, no other messages are passed between the client and server.

This mode of operation does not currently pose a problem and will not be changed during the project. However, the program would obviously benefit from the chance to operate using 0 communication, and there is a chance that this will be implemented in the future. This is an important reason why the communications needs to be simple and clear to use.

A more severe issue than the operating mode (synchronous/asynchronous) is the implementation of the communication, which is currently almost totally unstructured. A typical operation consists of the client packing a number of unlabeled bytes to a PICMessage structure, sending this to the server and the server extracting them, trusting only on their order to find the original data. The PICMessage does not have any specific structure and the communication is therefore virtually as unchecked as sending the bytes over the network as raw data. This is obviously error-prone and makes the code hard to understand. In addition to this, it makes changing the protocol very difficult for any non-trivial modification.

In addition to the communication being unstructured, the PICMessage class that is used when sending messages over the network does not have a clear, well-thought out interface (it defines some 30 methods) and this makes any code concerned with communication hard to understand.

Changing the method of communication has therefore two important goals: To make the code clearer and easier to read, and to make changing the communication possible (if not entirely trivial).

4.2 Changes to the Communication Implementation

As the biggest problems with the communication have to do with its unstructured nature, this was addressed first. Being capable of expressing communication using structures instead of bytes essentially makes it possible to operate on a higher level of abstraction, making the code easier to read, write and maintain. While it would have been possible to create a number of objects with methods to write out their contents and construct themselves from a byte stream, this would have been frustrating (as there are 34 different PICSearch messages) as well as error-prone, and any changes to the communication would have required changing the read/write methods as well as the objects' structure. This alternative was rejected.

There exists a well-known and well standardized package that implements remote procedure calls (RPC) in a transparent way to the user. A critical part of making any such call is the ability to transfer data (structured as well as unstructured) over a network, and SUN RPC defines a format for such transfers, known as external data representation (XDR, see RFC 1832 for details). Both XDR and RPC use a special preprocessor program, rpcgen, to create routines to write and read structures to and from a network in an implementation- and architecture-independent way. Using the whole RPC system was not necessary for PICSearch as it already contains routines for communication and a well-defined communication protocol, so only the XDR part of the RPC package is used.

XDR defines a data definition language very similar in syntax to the C subset used to describe structures. A file describing the structures used for communication is written using this language, and the file is then processed using rpcgen, which creates a C header file describing those structures and routines that can be used to pass these structures over the network. Changing any of these structures requires only changing the data definition file and executing rpcgen again; The routines used to transport these structures will be changed to reflect the changes to the structures.

4.3 Client-Server Communication Specification

The following listing describes in the XDR data definition language the structures used for communication between the PICSearch client and server processes. The structures are thoroughly commented, but due to the precise technical nature of the specification the comments are not always self-explanatory and fully understanding them requires familiarity with the basic PICSearch concepts.

```
typedef unsigned int algorithmID_t;  /* Type of algorithm ID */
typedef unsigned int databaseID_t; /* Type of database ID */
typedef unsigned int pictureID_t; /* Type of picture ID */

/* Permitted values for the client's server authority level */
enum ServerAuth {
  NoAccess,/* no access to server  */
  UserAccess,/* normal user access  */
  SuperUser /* superuser priviledges*/
};

/* Permitted values for the client's database authority level */
enum DatabaseAuth {
  DB_NoAccess,/* no access to database*/
  DB_ThumbAccess,/* thumbnail read access*/
  DB_ReadAccess,/* picture read access */
  DB_WriteAccess /* full write access */
};

/* Describes a single picture database to a client */
struct DBInfo {
  databaseID_t ID; /* unique ID of the DB */
  DatabaseAuth authLevel; /* authority level to DB*/
  string name<>; /* the name of the DB */
  algorithmID_t algs<>;   /* available algorithms */
};

/* Describes a single algorithm to a client */
struct AlgInfo {
```

```
  algorithmID_t ID; /* unique ID of the alg */
  string name<>; /* the name of the alg  */
  string description<>; /* describes the alg */
  string algParamWin<>; /* Tcl/Tk parameter win */
};

/* A clumsy workaround to allow arrays of strings */
struct strStruct {
  string str<>;
};

/* A value with a name associated with it */
struct namedValue {
  string name<>;
  string value<>;
};

/* Describes a client's access rights to a single DB */
struct DBRights {
  databaseID_t DBID;
  DatabaseAuth auth;
};

/* Describes a user's access rights to a server */
struct UserRights {
  string name<>; /* the name of the user */
  ServerAuth authLevel; /* serv authority level */
  bool changePW; /* allowed to change PW?*/
  DBRights DBs<>; /* access to each DB */
};

/* Describes a thumbnail */
struct thumbNail {
  unsigned int size; /* size in bytes */
  opaque data<>; /* thumbnail data */
  pictureID_t picID; /* associated picture ID*/
```

```
  string paramStr1<>; /* parameter string 1  */
  string paramStr2<>; /* parameter string 2  */
  unsigned int rank; /* pic's rank in query */
  float matchPercent; /* pic's match% in query*/
};

/* description of client-server messages. Unless otherwise noted, the */
/* messages are sent from the client to the server. The message numbers */
/* were assigned during previous PICSearch projects. A number of */
/* messages are mentioned but not specified using XDR because they */
/* contain no data. */

/* former message00. sent by server to initialize client after connecting. */
struct Init_msg {
  ServerAuth authLevel; /* server authority level */
  bool changePW; /* allowed to change PW? */
  DBInfo DBs<>; /* describes DBs */
  AlgInfo algs<>; /* describes algorithms  */
  string serverAd<>; /* server advertisement */
  string serverHTTP<>; /* server HTTP address */
};

/* former message01. sent by server to terminate a connection. */
struct Hangup_msg {
  unsigned int time; /* time of hangup */
};

/* former message02, sent to start a query. */
struct Query_msg {
  databaseID_t DBID; /* DB to query */
  algorithmID_t algID; /* algorithm to use */
  strStruct algParams<>; /* parameters to algorithm */
  unsigned int thumbMax; /* total pics wanted as answer */
  unsigned int thumbNum; /* num of thumbs to send at once*/
  float minMatch; /* min allowed match percentage */
  unsigned int qPicID; /* ID of picture to use in query*/
```

```
  unsigned int qPicDB; /* DB the query picture is in */
};

/* former message03, sent to terminate a query in progress. */
/* struct stopQuery_msg { }; */

/* former message04, sent to request more picture thumbnails. */
struct moreThumbs_msg {
  unsigned int firstThumb; /* rank of first thumbnail */
  unsigned int thumbNum; /* number of thumbnails wanted */
};

/* former message05, sent to insert a picture to a database. */
struct insertPic_msg {
  string fileName<>; /* name to save picture by */
  unsigned int picSize; /* size of the picture */
  opaque picData<>; /* picture data */
  string paramStr1<>; /* picture parameter string 1 */
  string paramStr2<>; /* picture parameret string 2 */
  databaseID_t picDB; /* database to put picture in */
  pictureID_t picID;   /* ID to give to the picture */
};

/* former message06, sent to request a picture from a database. */
struct getPic_msg {
  databaseID_t picDB; /* database the picture is in  */
  pictureID_t picID; /* ID of the picture */
};

/* former message07, sent to delete a picture from a database. */
struct deletePic_msg {
  databaseID_t picDB; /* database the picture is in */
  pictureID_t picID; /* ID of the picture */
};

/* former message08, sent to disconnect from a server. */
```

```
/* struct disconnect_msg { }; */

/* former message09, sent to add/remove a number of algorithms in a DB. */
struct modifyAlgs_msg {
  databaseID_t targetDB; /* database to operate on */
  algorithmID_t addAlgs<>; /* algorithms to add */
  algorithmID_t delAlgs<>; /* algorithms to remove */
};

/* former message10, sent to request user information from a server.  */
/*struct getUsers_msg { };*/

/* former message11, sent to change the access rights of a number of users. */
struct changeRights_msg {
  UserRights users<>; /* new access levels of users */
};

/* former message12, sent to add a user to a server. */
struct addUser_msg {
  string name<>; /* new username */
  string passW<>; /* new user's password */
};

/* former message13, sent to remove a user from a server. */
struct deleteUser_msg {
  string name<>; /* username of user to remove */
};

/* former message14, sent to change a single user's password. */
struct changePW_msg {
  string name<>; /* username identifying user */
  string newPW<>; /* new password */
  string oldPW<>; /* old password */
};

/* former message15, sent to create a new database. */
```

```
struct createDB_msg {
  string directory<>; /* directory to create DB in */
  string DBName<>; /* name to give to new DB */
  algorithmID_t algs<>; /* algorithms available in DB */
};

/* former message16, sent to add an existing DB to server's DB list. */
struct addDB_msg {
  string directory<>; /* directory the DB is in */
  string DBName<>; /* DB name to use in DB list */
};

/* former message17, sent to remove a database from server's DB list. */
struct removeDB_msg {
  databaseID_t DBID; /* ID of database to remove */
};

/* former message18, sent to physically delete a DB. */
struct deleteDB_msg {
  databaseID_t DBID; /* ID of database to delete */
};

/* former message19, sent to request server options. */
/* struct getServOpt_msg { }; */

/* former message20, sent to set a number of server options. */
struct setServOpt_msg {
  namedValue opts<>; /* options names and new values */
};

/* former message21, sent by server to give the progress status of a query. */
struct progReport_msg {
  float queryStat; /* percentage of query done */
};

/* former message22, sent by server. Query result thumbnails. */
```

```
struct queryResults_msg {
  struct thumbNail nails<>;   /* thumbnails */
};

/* former message23, sent by server to inform that a query has no matches. */
/*struct queryNoMatch_msg { };*/

/* former message24, sent by server as a general error when a request fails. */
/*struct requestFailed_msg { };*/

/* former message25, sent by server to inform that query results are over. */
/*struct queryNoResults_msg { };*/

/* former message26, sent by server as a general OK when a request succeeds. */
/*struct opComplete_msg { }*/

/* former message27, sent by server to give a picture to a client. */
struct picData_msg {
  string fileName<>; /* name of the picture */
  unsigned int picSize; /* size of the picture */
  opaque picData<>;   /* picture data */
  string paramStr1<>; /* picture parameter string 1 */
  string paramStr2<>; /* picture parameter string 2 */
  pictureID_t picID; /* picture ID */
  unsigned int picWidth; /* picture width */
  unsigned int picHeight; /* picture height */
};

/* former message28, sent by server to give the progress status of an alg. */
struct algProgress_msg {
  bool opType; /* add/remove operation? */
  algorithmID_t algID; /* algorithm in progress */
};

/* former message29, sent by server to describe users. */
struct userData_msg {
```

```
  UserRights users<>;     /* user information */
};

/* former message30, sent by server to describe a new user. */
struct newUser_msg {
  UserRights user; /* user information */
};

/* former message31, sent by server when an incorrect password is given. */
/* struct passwFail_msg { }; */

/* former message32, sent by server when a database is added. */
struct newDB_msg {
  DBInfo DB;   /* database information */
};

/* former message33, sent by server to describe current server options. */
struct servOpts_msg {
  namedValue opts<>; /* option names and values */
};

/* former message34, sent by server when a picture is added. */
struct newPic_msg {
  pictureID_t picID; /* picture ID */
};
```

5 Code Clean Up and Other Modifications (JL)

5.1 Removal of v1db

The obsolete version 1 database has been removed from the client. Any possible leftovers are accidental and should be removed.

The following files have been deleted:

- cmain/inc/main2db.hxx
- cmain/src/main2db.cxx
- v1db/inc/PicDbEngine.hxx
- v1db/src/PicDbEngine.cxx
- v1db/inc/PicDb.hxx
- v1db/src/PicDb.cxx
- v1db/

References to the deleted files have been removed from the following Makefiles:

- Makefile
- src/Makefile
- src/client/Makefile
- src/cmain/Makefile
- src/gui/Makefile

The client code has been modified to accommodate the changes.

- src/cmain/inc/mmain.hxx
  Removed PicMain::doReorganize()
  Removed PicMain::doDumpDb()
  Removed #include <main2db.hxx>
  Removed #include <PicDb.hxx>
  Removed PicMainDbUser db from all classes
  Removed readParamsFromDb()
  Removed writeParamsToDb()
- src/cmain/inc/mthreads.hxx
  Removed #include <main2db.hxx>
  Removed dbConn from sendResult() parameters
  Removed 'PicMainDbuser db' from all classes
  note: sendResult() may be obsolete

- src/cmain/src/mthreads.cxx

  Removed dbConn from SendResult(). Function may be obsolete.

  Gutted obsolete PicQueryThread::threadFunction(), it now does nothing.

  Gutted obsolete PicQueryThread::queryLoop()

  Gutted obsolete PicDbUpdateThread::RemoveOldEntries()

  Gutted obsolete PicDbUpdateThread::insertAlgNames()

  Gutted obsolete PicDbUpdateThread:doSignatures()

  Gutted obsolete PicDbUpdateThread:threadFunction()

- src/cmain/src/mtools.cxx

  Removed warnings of implicit declarations of system calls read() and write() by adding #include <unistd.h>

- src/cmain/src/maininit.cxx

  Removed "PicMainDbUser PicMain::db"

  Removed db references from PicMain::guiHasStarted() and PicMain::serveChild()

- src/cmain/src/mdbfuncs.cxx

  Removed PicMain::readParamsFromDb()

  Removed PicMain::writeParamsToDb()

- src/cmain/src/mmisc.cxx

  Removed PicMain::doReorganize()

  Removed PicMain::doDumpDb()

  Removed references to db.disconnect()

  Removed references to db.releaseDb()

5.2 Removal of PicImageProcAlg

Image processing has been delegated to an user-selectable software package, which can probably meet modern image manipulation needs better than the few simple functions offered by PICSearch. Thus, the PicImageProcAlg -class and related code has been removed from the system as obsolete.

The following files have been deleted:

- alg/src/PicImageProcAlg.cxx
- alg/inc/PicImageProcAlg.hxx

- alg/src/PicNegAlg.cxx
- alg/inc/PicNegAlg.hxx
- alg/src/PicLeeAlg.cxx
- alg/inc/PicLeeAlg.hxx

Several modifications have been done, as follows:

- smain/src/mmisc.cxx
  Removed PicMainSharedData::getImageProcAlg()
- cmain/src/mmisc.cxx
  Removed PicMainSharedData::getImageProcAlg()
- smain/inc/mshared.hxx
  Removed references to class PicImageProcAlg
- cmain/inc/mshared.hxx
  Removed references to class PicImageProcAlg
- smain/inc/mmain.hxx
  Removed PicMainSharedData::getImageProcAlg()
  Removed PicMain::getImProcAlgParams()
  Removed PicMain::processImage()
- cmain/src/maininit.cxx
  Removed initializer for PicImageProcAlg from PicMain::PicMain()
  Removed PicImageProcAlg startup from PicMain::guiHasStarted()
- smain/src/maininit.cxx
  Removed initializer for PicImageProcAlg from PicMain::PicMain()
  Removed PicImageProcAlg startup from PicMain::guiHasStarted()
- cmain/src/mainalgs.cxx
  Removed PicMain::getImProcAlgParams()
- cmain/inc/mmain.hxx
  Removed ImageProcAlg #include statement
  Removed PicMain::getImProcAlgParams()
  Removed PicMain::processImage()

- cmain/src/mainpics.cxx

  Removed PicMain::processImage()

- smain/src/mainpics.cxx

  Removed PicMain::processImage()

- gui/src/PicGUI.cxx

  Removed PicGUI::setImageProcessingAlgorithms()

  Removed PicGUI::tr_processParams()

  Removed reference to tr_process()

- gui/inc/PicGUI.hxx

  Removed PicGUI::setImageProcessingAlgorithms()

  Removed PicGUI::tr_process()

- Makefile, alg/Makefile

  Removed references to PicImageProcAlg files

5.3 Removal of Compile-time Warnings

Compiling **PICSearch v3.0** with warnings enabled caused endless amounts of complaining, efficiently hiding any true problems. Work has been done to remove these warnings from the system using "-Wall -Wno-unused" in various Makefiles. We do not wish to see the unused-type warnings, because there are a lot of functions whose parameters are not used in all instances of the function, but are still necessary to some. Also, we did not want to start changing lots of function interfaces at this point.

Removed warnings from "cmain/"

- src/cmain/inc/mshared.cxx

  Changed cache variables defined with "long" to "PICULONG"

  Changed cacheArrayDescr longs to PICULONGs

- src/cmain/src/maininit.cxx

  Changed elementsInChain() to PICULONG

  Changed getHashKey() to PICULONG

  Changed chainCollisions() to PICULONG

  Changed getArrayDescr() memory allocation to PICULONG

- src/cmain/src/mmisc.cxx

  Changed setParams().oldCacheSize to PICULONG

  Changed reInitializeCache().size_shift to PICULONG

- src/cmain/src/mtools.cxx

  Added (ssize_t) casting to sendMasterPrint() to remove warning

  Added (ssize_t) casting to sendMasterQueryAdvance() to remove warning

  Added (ssize_t) casting to sendMasterQueryResults() to remove warning

  Added (ssize_t) casting to sendThreadCommand() to remove warning

  WARNING: switch in sendThreadCommand() doesn't handle one possible enumeration. May cause trouble.

  Added return(boolTrue) to listenToMaster() to remove warning

- src/cmain/src/mthreads.cxx

  Added 3 extra parentheses to extractParams() to remove warnings

- src/cmain/src/mquery.cxx

  Changed PICSIZE nowSize to PICUSHORT.

  Changed member initializers in PicQueryArray() to match declaration order

Removed warnings from "smain/src/*"

- smain/src/mtools.cxx

  Removed warning about read() and write() by adding #include <unistd.h>

  Removed warnings about comparisons in read() and write() by adding (ssize_t) castings

  Added return(boolTrue) to the end of PicTools::listenToMaster()

- smain/src/maininit.cxx

  Added return(boolTrue) to PicMain::startGUI() (OBSOLETE FUNCTION)

  Changed PicMain::PicParams() initializers to match the .hxx declaration order

- smain/src/mquery.cxx

  Changed PicQueryArray::PicQueryArray() initializers to match the .hxx declaration order

  Added (PICSIZE) cast to PicQueryArray::add() to remove warning about signed<->unsigned comparisons

Removed warnings from "alg/src/*"

- alg/src/PicSunRaster.cxx

  Changed "i" and magicNumber to PICULONG to avoid signed<->unsigned warnings

  In ReadPicture(), "colorMap" might have been used uninitialized, fixed

  In ReadPicture(), unsigned char c was used uninitialized in Black and White -reading. It is now initialized to zero, but the read might have problems. No more than before, though.

- alg/src/PicImageFormat.cxx

  Changed "bytesToRead" to PICLONG in PicPPM::readPicture(), because the previous value of unsigned caused it to be set always to zero in a loop. It was compared with "<0" (== always true with unsigned).

- alg/src/PicExtSemgAlgs.cxx

  Added parenthesis to computeSegments() to assignment used as truth value

- alg/src/PicPicture.cxx

  Changed PicPicture::PicPicture() initializers to match declaration order

  Added parenthesis to writeToFile() to assignment used as truth value

5.4 Changes to the Host (SP)

All changes apply to files PICHost.cpp and PICHost.h. Functions and variables in class PICHost are referenced without the class scope specifier (ReadRequest() instead of PICHost::ReadRequest()) when unambiguous.

- Removed variables from the PICHost object: pidSuper and superIn. This modification disables a previous limit of only one superuser at a time.
- Removed useless functions that were cluttering up the interface from the PICHost object: IsSuperuser(), IsFirstSuperuser() and dropRights()
- Removed code referencing variables and functions mentioned above.
- Manually inlined funcs sendACK(), sendNAK1() and sendNAK2() and removed them, as the 2 lines per each are clearer than the calls.
- Removed from PICHost::Main() a net.CloseHostSocket() call that was placed after a for(;;) loop with no breaks. Placed similar call to the PICHost destructor.

- Removed from ReadRequest() lines reserving memory for userName and passWord, as this memory was immediately thrown away.
- Restructured comments in PICHost.h
- Removed from PICHost.cpp old, commented-out blocks.
- Fixed a bug in RemoveServer. It did not use its parameter but instead always removed the last process that was created.
- Removed unused and useless function IsAccess().
- Concatenated AddServer() to end of CreateServer() since these two logically go together. Removed private variable process that was only used by these to for communication.
- Removed private communication variables userName, passWord, changePassword and authLevel. added params to ReadRequest, GetAuthorization and CreateServer to pass the same information.
- Removed declaration for non-existent function AnonAccess.
- Manually inlined and removed single-comparison func IsFreeSpace.
- Removed declaration for non-existent constructor.
- Made a temporary improvement to the hard-coded maximum client limit by defining ABSMAXCLIENTS and using it instead of constant.
- Replaced hard-coded superuser and anonymous names and passwords with named constants.
- Added class UserInfo to represent a user.
- Modified ReadRequest, GetAuthorization and CreateServer to take a single UserInfo parameter.
- Replaced the fixed-size array that used to hold the server process ids with a standard template library list of UserInfo structures, which contain the process ids and some other useful info.

5.5 Removal of Unneeded GUI -Related C++ Code (TI)

The C++ component of the graphical user interface of the PICSearch is mainly a software interface. It receives messages from the client component of the program, and forwards them to the Tcl/Tk script driven part of the GUI. It also initiates appropriate actions on behalf of the GUI when it receives a state change message from the GUI.

During the first phase of the project following methods of the C++ component of the GUI were found to be unneeded, and were removed:

- gui/src/PicGUI.cxx and gui/inc/PicGUI.hxx

  Removed PicGUI::simpleDialog (BaseString type, BaseString name, BaseString text, BaseString bnames)

  Removed PicGUI::setPictureData(PicPicture &pic)

  Removed PicGUI::sendQueryAdvance(PICUSHORT percentage)

- cmain/src/mtools.cxx and cmain/inc/mtools.hxx

  Removed PicTools::sendMasterQueryAdvance(PICUSHORT percentage)

- smain/src/mtools.cxx and smain/inc/mtools.hxx

  Removed PicTools::sendMasterQueryAdvance(PICUSHORT percentage)

The removals have been tested and found not to affect the operations of the GUI. Respective method calls and routine initiations were also removed from different parts of the PICSearch program source code.

6 External Algorithms (JL)

PICSearch v3.1 features external search algorithm support. Each algorithm is an external library, which is linked to the server at run-time. This approach makes it possible to add/remove/modify algorithms while the system is online.

Server no longer has any internal, hard coded algorithms. All algorithm information is kept in the database and given to the client on request.

Server and client must undergo several modifications in order to support this approach.

6.1 Algorithm Libraries

Libraries are handled using the `dlopen()` interface. Due to the nature of the interface, the libraries cannot use any functions whose code resides outside the library file. Thus, all functions the libraries intend to use (including the class member functions!) must be added to the build of each library. A bare-bones algorithm skeleton containing all required code will be provided for the algorithm developer.

For PICSearch server, every external algorithm library must provide the following outside-accessible functions:

setParams(PicIxCollect &params)

This sets possible parameters got from Tk parameter window. The parameters will be followed until new ones are assigned. It is up to the library how the parameters are interpreted.

compare(PicSignature &firstSig, PicSignature &secSig)

This function compares two signatures and outputs a match rate from 0 to 1000.

computeSignature(PicPicture &pic)

This will calculate a signature from the picture given as a parameter.

algName(void)

This will return the name of the algorithm.

algInfo(void)

This will return some other human-readable information about the algorithm. It is up to the algorithm developer to decide what information is provided.

External algorithms will not provide an ID or any type information. An unique ID is assigned by the server when the algorithm is added and type information (drawing, sketch, etc) is given by the user on add.

6.2 Algorithm Class

[alg/inc/PicSearchAlgorithm.hxx]

All virtual functions of the algorithm class are changed to function pointers to support the library interface.

Algorithm class already has an unique ID, but it has not been used in a very unique way. From now on, all algorithms will be matched by their IDs only. Notice that the ID is assigned by the server, not by the algorithm developer.

6.3 Algorithm Related Changes to the Database

In the main level of the database will reside a table for algorithm information. It must contain at least the following elements:

- **ULONG Alg_ID**

  An unique identifier for this algorithm.

- **string Type_information**

  Some extra information about the algorithm. It is still unsure if (and how) this will be used.

- **string Path_to_the_library**

  A file system path to the library which provides the functions of this algorithm.

- **string Tk_parameter_window_code**

  If algorithm has a parameter window, this is the Tk code provided to the client GUI.

- **int Out_of_Order**

  This flag implies that the algorithm has been disabled from the system, and should no longer be used.

- **ULONG Replaced_With_ID**

  This algorithm ID tells that some other algorithm has taken over the functions of this one.

In addition, every picture database will contain an algorithm specific signature table, consisting of (picture_ID, signature) pairs. The name of the table will match the unique ID of the algorithm.

6.4 Algorithm Related Changes to the Server

Algorithm initialization

[smain/src/maininit.cxx:PicMain()]

Algorithm initialization is implemented here.

For each algorithm available in db, the following initialization is performed.

1. Algorithms data is read from the database

2. The library is opened
3. Object representing the algorithm class is created
4. Function pointers are assigned from the library
5. Other data is assigned to the object
6. Algorithm object is added to the algorithm array

Algorithm name will be queried from the library with algName() function. Other algorithm information (e.g. quick help) will be asked from the algorithm library with algInfo() function.

Subsequently the algorithm information is sent to the client.

The opened library will be closed on server exit, or when server notices that the library should no longer be used.

Creating signatures

[smain/src/mmisc.c/PicMain::calcAllSignatures(PicPicture &image)]

This function goes through the available algorithms and calculates their signatures using the library functions. No significant changes will be done to the existing function code.

Comparing signatures

[server/src/Query.cpp!PICServer::Query()]

Before each query, server will check from the database if the used algorithm is still valid, and Out_of_Order has not been set.

If algorithm has received Tk parameter information from the client, it will be set with library function setParams().

All entries of the used picture database will be scanned and their signatures read. Every signature fetched from the database will be compared with the library function compare().

6.5 Algorithm Maintenance

Algorithm maintenance is done through the client GUI with suitable privileges. Client GUI will provide an algorithm maintenance menu, listing the operations Add, Delete and Modify, along with an 'Available Algorithms'-list.

In all operations, all files indicated by user provided file paths are copied to the actual algorithm directories. After the server completes its operation, user provided temporary files can be removed.

Removing algorithms

In the client GUI, user chooses the algorithm to be removed from a list. An ALG_REMOVE message is sent to the server which processes it as follows:

Algorithm will be marked as Out_of_Order to the database. Servers will no longer provide service based on this algorithm. The queries already running will be finished in normal fashion.

When a server notices at the beginning of a query or insert that an algorithm has been removed, it will notify its own client. Client can request update of the algorithm info.

Removed algorithm, its signature table and the library file will be physically removed only when the host starts the next time.

Adding new algorithms

In the client GUI, user fills a New Algorithm form. It should have the following items:

- **Path to the algorithm library file**

  This is a path to the compiled library file on the servers file system.

- **Path to the Tk parameter window code**

  This is a path to the Tk parameter window code on the servers file system.

- **List of algorithm types**

  User chooses the algorithm type from {Basic, Sketch, Drawing, Texture, Color, Combination}. These types are defined in "cmain/src/maininit.cxx".

Other information will be queried from the library by the server. An ALG_ADD message (containing the data specified above) is sent to the server, and server processes it as follows:

New algorithm is given an unique ID. The library file provided by user will be copied to actual algorithm library directory, and renamed according to the unique ID. This approach prevents old and new algorithm library files from colliding.

The optional Tk parameter window code is read into memory and the algorithm is added to the database.

Only the server (and client) which added the algorithm will immediately recognize the added algorithm. Other servers can gain knowledge of the added algorithm only by restarting. This means a reconnection from the client side. This is mostly because of the synchronous nature of PICSearch message system.

Activation of algorithms

Simply adding a new algorithm does not yet make it available in any picture database. User must select "Database management" from the client GUI, and activate the new algorithm for a specified database. Server will be sent a request to calculate the signatures of this algorithm for the specified database.

The client and its GUI must be updated to support this.

Modifying existing algorithms

Modification of an existing algorithm will be done with inserting a new algorithm and removing the old one.

In the client GUI, user chooses which algorithm will be replaced with a new library file, and informs if the signatures must be recalculated, or if it is just sufficient to copy them. User also provides a path to the new Tk window code, if the old is not suitable anymore. An ALG_UPDATE message is sent to the server and processed as follows:

First, the new algorithm is added. If signatures do not need to be recalculated, server can copy the old signature table to a new one. Signature table cannot be renamed, because the old one is probably still in use.

After that, server can remove the old algorithm by marking it in database with Out_of_Order -flag and a proper Replaced_with_ID value.

7 Changes to the Database (SP)

The PICSearch database will be thoroughly revised during the second phase. The following sections describe three major changes.

7.1 Changes to the Database Implementation

The current PICSearch database is based on a special-purpose system that was implemented solely for use in PICSearch. It is a bare-bones service that implements simple indexing and basic database operations, but lacks several vital features for a real database:

- The database system never really deletes anything. Records in the database are marked "deleted" upon deletion but never physically removed from the database.
- The slots in the database that are marked "deleted" after deleting records are never reused by new records that are inserted to the database. This causes the database to grow constantly.
- The two problems mentioned above would not necessarily be serious if the database could be reorganized (built again in a fashion that removes the holes), but the database reorganization feature is broken in a way that can not be trivially repaired.
- The indexing system is implemented poorly, requiring repeated linear lookup of the records in the index
- The database locking scheme is lacking in features.
- The database system as a whole is inefficient. This is a serious problem as the database is a major bottleneck in the overall performance of the system.
- The database system interface is unusable, requiring intimate knowledge of the system for even the simplest operation and tedious use of PICMessage-class objects for each query and update. This easily leads to several pages of obscure code each time the database is used.

These and associated problems led to the decision to remove the current database in its entirety and replace it. Several options were considered and to best avoid the same problems associated with the current database it was decided that an external database library would be used. The GDBM (GNU database manager [7]) library was chosen. It is distributed under the GNU General Public Licence and is therefore freely available. It is also relatively low-level, efficient and easy to use.

Replacing the old database with the new is a major undertaking and a significant part of the second phase of the project. In addition to removing the old code and fixing references to the old database, some higher-level functionality must be implemented around the GDBM system to make it simple and safe to use in a multiple-server environment.

In addition to switching to GDBM, portions of the database system will be implemented as simple formatted textfiles. See section **??** for details.

7.2 Changes to the Image Database Design

In addition to changing the database implementation, the database structure design will be revised. The following describes the current design, what is wrong with it, and finally the new design.

Each PICSearch server may provide access to several image databases (referred to as databases from this point), each of which has the same structure (see Figure 2). The figure is a simplification of the actual database structure in that it does not show the tables' keys or reference keys as part of the tables' data. These are represented by lines connecting the tables. Refer to the documentation of previous PICSearch projects for a detailed description of the database table fields.

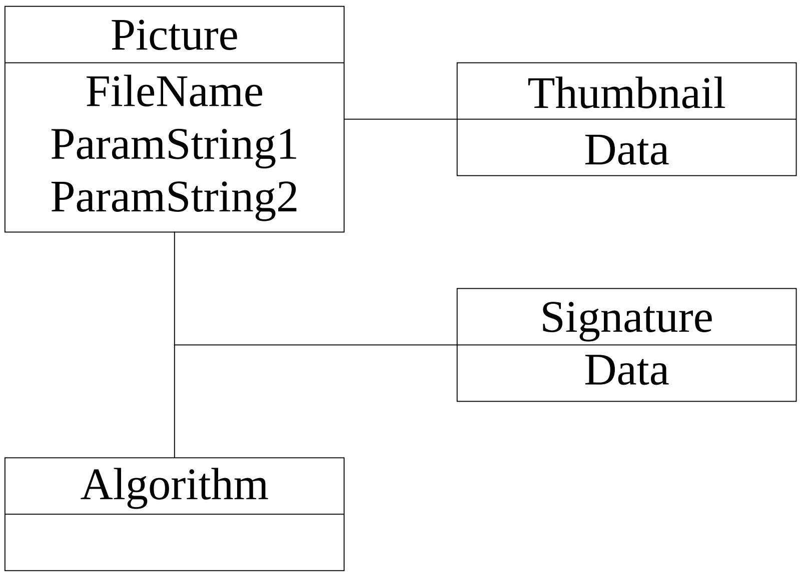

Figure 2: The current image database structure.

The database structure design has several shortcomings that can be easily fixed during the database system rewrite:

- The algorithm table has no data except its key, a single integer value. There is no reason to have a table with no real data.
- The thumbnail table has a single data item (other than its key), a thumbnail of a single picture. In addition to this, each thumbnail table entry is associated with exactly one picture table entry. There is no reason to keep these two tables separate.

- The signature table holds a single signature for each picture/algorithm combination. This table is scanned through during each query operation. Therefore it is important for the overall efficiency of the server to be able to find the relevant signatures quickly. With the current design, in a database with N algorithms only $1/N$ of the entries in this table are relevant to any single query.

The database design will be changed in a way that will fix these problems (see Figure 3). In the new design, the thumbnails are part of the picture table, each algorithm has a table containing all its signatures, and the separate algorithm table has been removed. A list of all available algorithms is still contained in a higher-level database that describes all the picture databases and their associated algorithms. This new design fixes all the problems mentioned above and simplifies operations that need to add, remove or modify algorithms in a database.

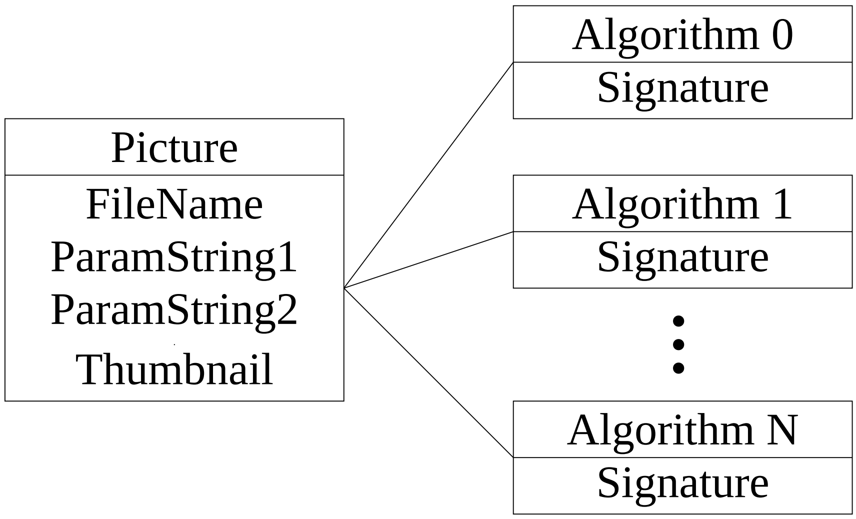

Figure 3: The new image database structure.

7.3 Changes to the Server Database

As mentioned above, a single server may provide access to a number of databases. To make this possible PICSearch has a two-level database structure (see Figure 4).

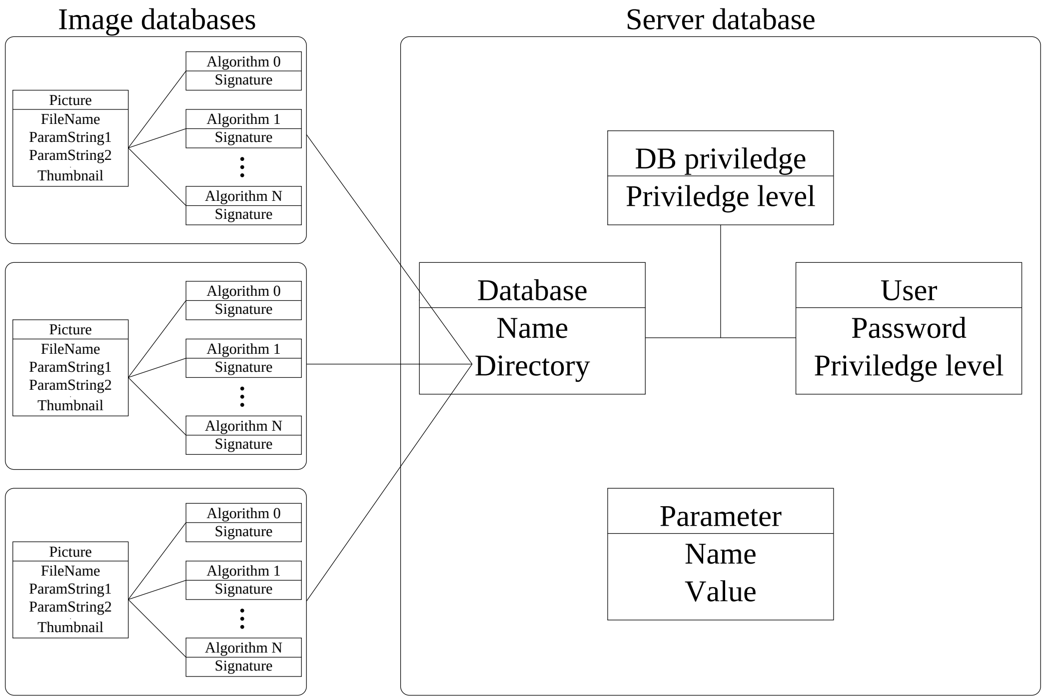

Figure 4: The new PICSearch database structure.

The lines connecting the `Database` table's `Directory` field to the `Image` databases are not actual database references. Instead, the `Directory` field holds the name of a subdirectory to the server database. Each of these subdirectories contains a single image database.

The `Database, Database privilege, User` and `Parameter` tables are currently implemented with the same database system as the image database. These tables are simple, seldom referenced and rather small, and the system would benefit in usability from a simpler implementation. That is why these tables will be implemented as simple formatted text files that can be easily edited by hand. A minor part of the second phase will be creating a text file parser and a simple interface for it.

8 Changes to the User Interface (TI)

PICSearch graphical user interface (GUI) will need to be updated during the second phase of the pics99 project to reflect the changes in other parts of the program. No major changes to the GUI are needed.

A new menu entry will be available under main menu `server` for algorithm maintenance. It will let the user to carry out all algorithm maintenance tasks described in section 6. A new algorithm window will be used to get all the necessary information input from the user.

A new menu entry will be available under main menu `server` for requesting re-initialization of the client. User may wish to re-initialize the connection in order to get access to the most up-to-date algorithm information.

Glossary

Following terms have been used in this document. Some meanings may be project-specific.

algorithm

In this document, algorithm references to a content-based image comparing algorithm. Basically, these algorithms just decide how closely two pictures resemble each other. PICSearch uses these to compare a user-provided query image with the pictures in the database.

bug

An error in the source code of a program. A bug is generally considered to be an error which does not stop the program from being compiled, but instead causes it to function erratically, stop functioning, or to function in ways not considered appropriate.

dead code

A useless section of the source code of a program, ie. a section which does not get executed when program is run, or which duplicates operations already implemented in some other code section.

debugging

Removing errors (see: bug) from the source code of a program.

gutted

In this document, gutting means the removal of operative code, while the function definition is left, returning a simple value in the fashion of "boolTrue".

hard coded

A value used by a program in its calculations is said to be hard coded if it is explicitly embedded in the source code. These values typically include some numerical limitations of the program, for example maximum number of permitted users could be hard coded as 99, as opposed to using a constant MAX_USERS_PERMITTED or some variable.

signature

Every algorithm produces a signature out of a source image. The signature contains those things the algorithm thinks important in the source image. Signatures of two different pictures are used to check for resemblance, not the pictures themselves.

References

[1]
K. Lemström, J. Korte, P. Kuusi, P. Kyheröinen and P. Päiväkumpu:
PICSearch - A platform for Image Content-based Searching Algorithms, (Technical Report C-1997-48), proceedings of the Sixth International Conference in Central Europe on Computer Graphics and Visualisation 98 (WSCG'98), edited by Vaclav Skala, Chzech Republic, February 9–13, 1998, pp. 222–229.
`http://www.cs.helsinki.fi/ klemstro/publications.html`

[2]
P. Kuusi, K. Lemström, P. Kanerva, S. Koponen, S. Männynsalo, J. Riihimäki, J. Teinilä and T. Virtanen:
A Client-Server Tool for Image Retrieval Applications, In: Proceedings of the HeCSE Workshop on Emerging Technologies in Distributed Systems, Helsinki University of Technology, Department of Computer Science and Engineering, pp. 45–46 (Research Report A50). Otaniemi, Finland. March 1998.

[3]
S. Kaski, T. Koskinen, J. Leinonen and M. Pennannen:
pics98, Implementation Specification, Helsinki, Finland, September 10, 1998.
`http://www.cs.helsinki.fi/group/pics98/`

[4]
J. Korte, P. Kuusi, P. Kyheröinen ja P. Päiväkumpu:
PICSearch-projektin (PICSearch v1) vaatimusmäärittely, Teknillinen korkeakoulu, 1996.

[5]
P. Kanerva, S. Koponen, S. Männynsalo, J. Riihimäki, J. Teinilä ja T. Virtanen:
NetPICSearch-projektin (PICSearch v2) vaatimusmäärittely, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 1997.
`http://www.cs.helsinki.fi/group/netpics/VM/VaatM12.html`

[6]
S. Kaski, T. Koskinen, J. Leinonen ja M. Pennannen:
pics98, Vaatimusmäärittely, Helsingin yliopisto, Tietojenkäsittelytieteen laitos, 1997.
`http://www.cs.helsinki.fi/group/pics98/`