Customer: Kjell Lemström

Instructor: Anita Kasari

pics98

Implementation Specification

Samuli Kaski
Timo Koskinen
Jani Leinonen
Mika Pennanen

Helsinki, on the 10th of September 1998

Software Engineering Project, Group 3, Summer 1998
Department of Computer Science
University of Helsinki

Contents

1 Introduction (TK)

PICSearch is a development environment for content-based image retrieval algorithms. Its main function is to serve as a testing platform for experimental retrieval algorithms. PICSearch may, however, also be used as a conventional image cataloging system.

Version 3 of PICSearch is based on earlier versions of the system. Major features introduced with version 3 are caching of thumbnail images at the client and sending of algorithm parameter window scripts with the initialization message (previously these Tcl/Tk scripts were stored at the client.) Other enhancements include a redesigned graphical user interface with three alternative main window sizes and some new client options.

1.1 About This Document

This document describes how the features specified in the Design Document were implemented and explains the motivation behind these decisions. This document also includes a list of features that were implemented but are not mentioned in the Design Document, as well as a list of features that are in the specification but were not implemented. The results of module and system testing are also to be found on these pages.

2 Changes to the Requirement Specification and to the Design Document (MP&JL)

Matters which were defined in the Requirement Specification were taken into consideration also in the Design Document. So there were no changes between the Requirement Specification and the Planning Document. The changes took place at the realisation stage. In the following the report is from the changes that have been done to the Desing Document.

The changes in the project group in the planned working assignment have been divided to two parts, removed features and additions. The biggest removed feature was the interruption of AMP during the project for the taking to the end of the PICSearch v3.

2.1 Removed Features

The following sections of the design document have changed:

2.3 Algorithm Management Program - AMP
which totally removed. AMP was interrupted during the project.

3 Communication
Removed the AMP that has been demonstrated in the introduction.

5.2 Algorithms
which totally removed. The algorithms operate, such as in PICSearch v2.

6 Algorithm Management
which totally removed.

2.2 Additions

The following sections of the document have changed since the design phase ended:

3.1 Message Specification Changes
The Message types were changed during implementation.

For changes to existing messages from client to server, refer to Table 1.

Message	Change or implementation description
Message 02	**Start Query** - The message had to be changed to accomodate PICCache. At query time an array consisting of n * 2 longs (a long is 4 bytes on x86) is constructed that describes the contents of PICCache at that particular moment, n being the number of non-null elements stored in the cache.
Message 04	**Get More Thumbnails** - See Message 02.

Table 1: Modified Client→Server messages

3.1 Message Specification Changes
Added the new algorithm parameters (Tcl/Tk) to the init message. Also the server response (Message 22) to Message 02 or 04 was changed during implementation.

For changes to existing messages from server to client, refer to Table 2.

Message	Change or implementation description
Message 00	**Init** - One new additional field is required. The field contains the Tcl/Tk parameter window of the algorithm.
Message 22	**Query Results** - The message had to be changed to accomodate PICCache. The server sends a length of 0 bytes and skips the data section if the thumbnail to be sent is stored in PICCache.

Table 2: Modified Server→Client messages

3 Implementation Solutions

Reasons and means that have been used for realisation of the PICSearch v3.

3.1 Transmitting Algorithm Parameters to the Client (MP)

Because in the future the purpose that the client is independent of the algorithms will be, the transmitting of the parameters was solved in connection with the Init-message from the server to the client. The server tells own database information and algorithm information in which the Tcl/Tk files of the parameters are sent at the algorithms.

Transmitting parameter window has been realized by adding to the message the parameter area of the algorithms which has been realized with C++ language. Tcl/Tk files located in which it is the parameters of the algorithms have been realized on Tcl/Tk.

The parameter windows which are given to the algorithm are not added to the subwins.tk file of the gui index any more but the parameters of every algorithm to the alg index separately. Only to those algorithms on which there are parameters the own parameter files have to be made. The parameter files must be named as PicAlgParWinID in which ID is the ID mark of the algorithm.

The changes have taken place to the client as well as the server.

3.2 PicGUI (TK)

The graphical user interface of PICSearch is implemented with Tcl/Tk scripts. These scripts communicate with the PicGUI module via an application specific Tcl command `transmit`. Once a `transmit` command is intercepted PicGUI searches an array of function pointers for a matching function (one with the same name as the first argument to `transmit`.) If one is found and `transmit` was given the correct number of arguments, the function pointed to by the pointer is called with the parameters given in the call to `transmit`. Otherwise an error message is returned to Tcl.

The motivation behind implementing `transmit` as an array of function pointers was to make the source code as compact as possible. In version 2 the implementation used a lengthy if-else construction.

The following is a complete list of `transmit` commands and their arguments. The command to be executed by `transmit` is always its first argument, for example the Tcl command `transmit addUser name password` asks PicGUI to add a new user. The list below also tells which method the function calls to fulfill the request.

`transmit` Commands and Their Arguments

- **acceptRights**
  Calls `PICClient::AcceptRights()`.

- **addUser** user name, password
  Calls `PICClient::AddUser()`.

- **batchInsert** filename
  Calls `PicMain::batchInsert()`.

- **browseResults** number of first image
  Calls `PicMain::browseQueryResults()`.

- **cancelQuery**
  Calls `PICClient::StopQuery()`.

- **changePassword** user name, new password, old password
  Calls `PICClient::ChangePassword()`.

- **changeSearchType** search type name
  Calls `PicMain::changeSearchType()`.

- **changeUserDbAuthorization** user name, database name, authorization
  Calls `PICClient::changeUserDbAuthorization()`.

- **PICClient::changeUserServerAuthorization** user name, server authorization, can change password flag
  Calls `PICClient::changeUserServerAuthorization()`.

- **closeQueryImage** image list index
  Calls `PicMain::exImageClosed()`.
- **connect** host name, user name, password
  Calls `PICClient::Connect()`.
- **createDatabase** db name, db algorithms, db description
  Calls `PICClient::CreateDatabase()`.
- **deleteImage** id of the image to delete
  Calls `PicMain::deleteImageFromDb()`.
- **disconnect**
  Calls `PICClient::Disconnect()`.
- **getAlgorithmInfo** algorithm name
  Calls `PICClient::GetAlgorithmInfo()`.
- **getAlgorithmParams** algorithm name
  Calls `PicMain::getSearhAlgParams()`.
- **getAllDbNames**
  Calls `PICClient::GetAllDbNames()`.
- **getAllUsers**
  Calls `PICClient::GetDbUsers()`.
- **getClientOptions**
  Calls `PicMain::getParams()`.
- **getDatabaseAlgNames** database name
  Calls `PICClient::GetDbAlgorithms()`.
- **getDatabaseInfo** database name
  Calls `PICClient::getDatabaseInfo()`.

- **getDatabaseNames**
  Calls `PICClient::GetDbNames()`.

- **getImage** image id, destination image name, image list index
  Calls `PicMain::readImageFromDb()`.

- **getServerAddress**
  Calls `PICClient::GetAddress()`.

- **getServerAdvert**
  Calls `PICClient::GetServerAdvertisement()`.

- **getServerOptions**
  Calls `PICClient::GetServerOptions()`.

- **getThumbnail** name of thumbnail, name of destination image, image list index, image id, image name
  Calls nothing.

- **getUserDbAuthorization** user name, database name
  Calls `PICClient::getUserDbAuthorization()`.

- **getUserServerAuthorization** user name
  Calls `PICClient::getUserServerAuthorization()`.

- **getSignature** algorithm name, image list index
  Calls `PicMain::getSignatureImage()`.

- **guiStarted**
  Calls `PicMain::guiHasStarted()`.

- **insertImage** image name, image list index, update flag
  Calls `PicMain::insertImageToDb()`.

- **openQueryImage** filename, destination query image name, image list index, update flag
  Calls `PicMain::loadImage()`.

- **query** algorithm name, id, filename, name, description1, description2, image name,...
  Calls `PicMain::query()`.
  If there are several query images the arguments id through image name are repeated for each image. As of v3 PICSearch only supports queries using one query image.

- **quit**
  Calls `PicMain::exitProgram()`.

- **removeDatabase** database name, mode
  Calls `PICClient::RemoveDatabase()`.
- **removeUser** user name
  Calls `PICClient::RemoveUser()`.
- **saveQueryImage** filename, image list index, overwrite flag
  Calls `PicMain::saveImage()`.
- **scaleImage** source image name, destination image name, destination width, destination height
  Calls `PicTools::scaleImage()`.
- **setAlgorithmParams** algorithm name, parameters
  Calls `PicMain::setSearhAlgParams()`.
  Algorithm parameters are given as one string.
- **setClientOptions** options
  Calls `PicMain::setParams()`.
  Options are given as one string containing a Tcl list whose elements are the actual option values.
- **setDatabaseInfo** database name, description string
  Calls `PICClient::setDatabaseInfo()`.
- **setImageAttributes** image list index, new attributes
  Calls `PicMain::setImageAttributes()`.
  Attributes are given as one string containing a Tcl list whose elements are the actual attribute values: name, description1, description2, width, height, sizeChanged.
- **setNumOfDbFrames** number of database frames
  Calls `PicMain::setNumOfDbFrames()`.
- **setServerOptions** maximum number of connections, disconnect timeout in minutes, server's http address, server advertisement text
  Calls `PICClient::SetServerOptions()`.

3.3 PICCache (SK)

The client side cache was implemented as part of the program module that controls everything the client does. It's main structure is part of `PicMainSharedData` in `PicMain` and it is initialized at program startup.

The PICCache itself can be (and is) sized through a configuration parameter. This parameter tells PICCache how many thumbnails it is allowed to store, thus allocate memory for. Specifying a value of 0 for the cache size has special meaning, it will switch the cache off. The cache can even be switched off while the client is running.

For the description of the PICCache data structure, refer to Figure 1 below.

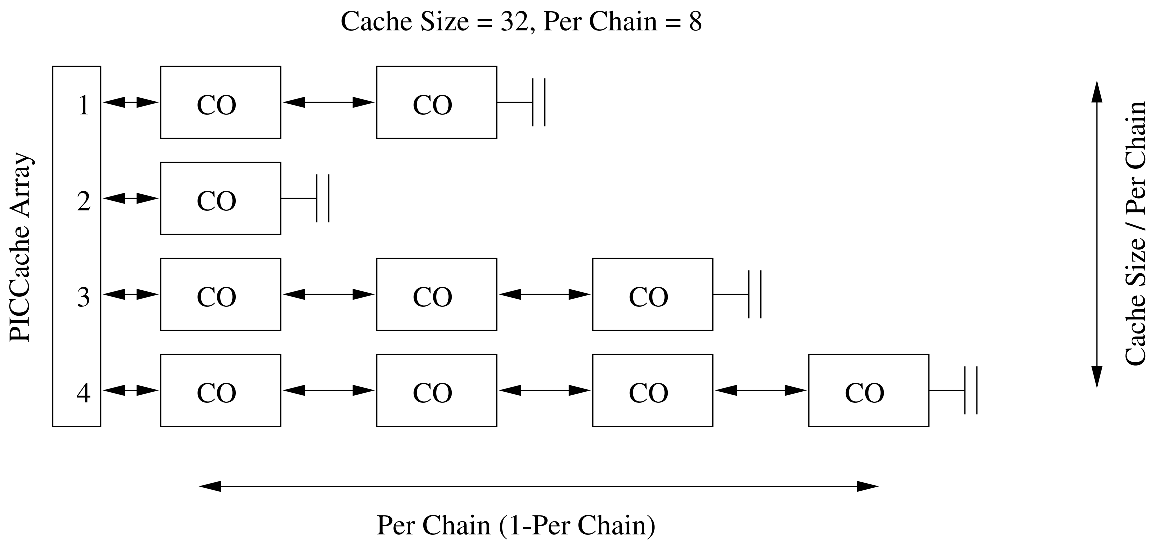

Figure 1: PICCache Data Structure

As one can see from the Figure 1, PICCache implements a data structure which uses a simple open hash algorithm with collision chains.

When initialized, PICCache has a minimal size. The collision chains will grow with time assuming enough new, not yet stored, thumbnails are received from the server. After PICCache has been initialized with a non-null size, the most used operations it offers, are getThumb and putThumb. These methods are used for looking up a thumbnail in the cache and respectively storing one in it. Duplicates won't be stored, but it would never happen anyway because of the way the queries are done.

Each query and moving of the result set scrollbar will cause PICCache to construct an array describing it's contents in full and send that array to the server. The benefit of doing things this way is that we avoid further round trips. The downside of it is that the array is sent each time, meaning all it's non-null members will be sent over the network. Since two values (longs, each 4 bytes on x86) give a thumbnail a unique handle (and a hash key), this means that a full cache of 100 thumbnails will result in $(100 * (2 * 4)) = 800$ bytes to be sent over the network on each query or scrolling of the result set.

The benefit of PICCache comes from the saved thumbnail sends. When the server receives a request to send additional or new thumbnails relating to a specific query, it also knows what not to send because the client passed the array describing the contents of PICCache at the time of that particular query.

The implementation itself can be found in the following source code files:

- cmain/inc/mshared.hxx
- cmain/src/maininit.cxx

Additionally changes have been made to the following source code files to accommodate PICCache:

- cmain/src/mmisc.cxx
- client/src/PICQuery.cpp
- client/src/PICClient.cpp
- server/inc/PICServer.h
- server/src/PICServer.cpp
- server/src/Query.cpp

3.4 Client Configuration (SK)

It was necessary to allow client parameters to be saved and re-read into the client at program startup. Earlier versions of PICCache implemented this functionality by using the server to store the parameters. This didn't suite our needs since we needed the parameters before any connection was established to a server.

We wanted a simple configuration file that could be edited with a simple text editor, thus we went for a ASCII file. The configuration file is parsed for sections (marked with brackets []), variables and their respective values. Variables are separated from values by the is equal to character '='. A short example is included below:

picsearch.cfg

[gui]
resolution=
windowpos=
..

As you can see from the example above, the configuration is named picsearch.cfg. At program startup three different locations are scanned for the presence of the configuration file, in the following order: $PICSEARCH/, $HOME/ and in the current directory. Settings are always stored to the same file on program termination. If the configuration file doesn't exist, a default configuration will be created in one of the above directories. Where the default configuration file be saved depends whether either of the $PICSEARCH/ and $HOME/ environment variables exist. The $HOME/ or the current directory is your likely choice.

The parameters, stored in the `PicParams` object, are part of `PicMainSharedData` in `PicMain`. The implementation itself can be found in the following source code files:

- cmain/inc/mshared.hxx
- cmain/src/maininit.cxx

The following source code files have been changed to use the new parameters:

- client/src/PICClient.cpp
- client/src/PICQuery.cpp
- cmain/inc/main2db.hxx
- cmain/src/main2db.cxx
- cmain/inc/merrors.hxx
- cmain/src/mdbfuncs.cxx
- cmain/inc/mmain.hxx
- cmain/src/mmisc.cxx
- cmain/inc/mquery.hxx
- cmain/src/mquery.cxx
- cmain/inc/mthreads.hxx
- cmain/src/mthreads.cxx

4 Module Testing

4.1 Algorithm Parameters (MP)

The table below describes the tests the Init message was subjected to and their results.

Test	Result
Sending `Init` without parameters	OK, some algorithms could be without parameters.
Sending `Init` within parameters	OK, client can use the parameters of algorithms.

Table 3: Module tests of PicGUI.

4.2 PicGUI (TK)

The sole purpose of the PicGUI module is to create one new Tcl command (`transmit`) and process invocations of this command. The table below describes the tests PicGUI was subjected to and their results.

Test	Result
Creating `transmit`	OK, a call to `PicGUI::start` registers `transmit` as a valid Tcl command.
Calling `transmit` with a valid command name and the correct number of arguments	OK, all functions in the array `PicGUI::transmitFuncs` get called properly: no unused functions.
Calling `transmit` with a valid command name and an incorrect number of arguments	OK, `PicGUI::transmitCmd` returns an error to Tcl, except for `transmit query` which has a variable number of arguments.
Calling `transmit` with an invalid command name	OK, `PicGUI::transmitCmd` returns an error to Tcl.

Table 4: Module tests of PicGUI.

4.3 PICCache (SK)

In this section you will find test results for the methods that construct the functionality of PICCache. Only methods which take parameters, and thus the program execution could be influenced by the user, were tested.

- **void PICCache::PICCache(long)**
  This method is the constructor for PICCache. The parameter is the value of how many thumbnails should be stored in the cache. The method itself isn't doing any bounds checking because it only gets called from PicMainSharedData's constructor and the checking is done there. The method has been tested and found to be working.

- **bool PicMainSharedData::reInitializeCache(long)**
  This method is used when the cache size is changed while the client program is running. A sanity check for the new size is done against MIN_CACHE_SIZE and MAX_CACHE_SIZE as is in PICCache::PICCache(long). The method has been tested to be safe.

- **long PICCache::getHashKey(long, long)**
  This method is responsible for returning a hash slot for the thumbnail in question. The picture id and database id of the thumbnail are given as parameters. Tested with reasonable values, overflows in the hash function are possible, relies on the server to make sane id numbers.

- **long PICCache::chainSize(long)**
  This method is used for returning the size of the collision chain of the hash slot in question. Method has been tested to be safe.

- **long PICCache::elementsInChain(long)**
  This method is used to return the true size of the collision chain. Returns only the number of non-null members. Method has been tested to be safe.

- **long PICCache::chainCollisions(long)**
  This method is used for returning the amount of collisions for this particular chain. Method has been tested to be safe.

- **bool PICCache::isThumbInCache(long, long)**
  This method is used for returning information to the caller whether the thumb-

nail in question is already stored in the cache or not. Method relies on the server for providing sane id numbers. Method has been tested to be safe.

- **bool PICCache::putThumb(long, long, const PicPicture &)**
  This method is used for inserting a new thumbnail to the cache. The method relies on the server to provide valid id numbers. Method has been tested to be safe.

- **bool PICServer::isInPICCache(long, long)**
  This method is used in the server for telling if the thumbnail to be sent is either found or non-existent in PICCache. The method relies on the id numbers to be sane, which in turn have originated from the server. Method has been tested to be safe.

- **PicPicture *PICCache::getThumb(long, long)**
  This method is used for returning the thumbnail in question to the caller. Again, this method also relies on the server to provide sane id numbers. Method has been tested to be safe.

4.4 Client Configuration (SK)

In this section you will find test results for the methods that construct the functionality of `PicParams`. Only methods which take parameters, and thus the program execution could be influenced by the user, were tested.

- **void PicParams::PicParams(char *)**
  This method is the constructor for PicParams and gets called on program startup. The parameter is the configuration filename and since the user can't give a runtime value for it, this method is safe to string overflows. The method has been tested and found to be working.

- **void PicParams::PicParams(const PicParams &)**
  This method is one of the constructors of the PicParams object. It takes a reference to another PicParams object and uses that as source. As the original object is a valid PicParams object this method is safe. The method has been tested and found to be working.

- **void PicParams::restoreSettings(const PicParams &)**
  This method is for restoring settings from another PicParams object. Same applies here as above, the source is a valid PicParams object, thus the method is safe. The method has been tested and found to be working.

- **PicParams &PicParams::operator=(const PicParams &)**
  This method is a special C++ operation, it is an assignment method. Same applies here as for the previous two methods. The source is a valid PicParams object thus the method is safe. The method has been tested and found to be working.

- **PICBOOL PicParams::isOptSec(char *)**
  This method is used for telling whether the string given as parameter is a section in the configuration file. Method isn't vulnerable to string overflows as long as the source string has enough storage and is null terminated. The method has been tested and found to be working.

- **PICBOOL PicParams::isOpt(char *)**
  This method is used for telling whether the string given as parameter is a option line in the configuration file. Method isn't vulnerable to string overflows as long as the source string has enough storage and is null terminated. The method has been tested and found to be working.

- **PICBOOL PicParams::getOpt(char *, char *)**
  This method is used for getting the option from the string given as parameter. Used together with isOpt(). The Method isn't vulnerable to string overflows as long as the source string has enough storage and is null terminated. The method has been tested and found to be working.

- **PICBOOL PicParams::getOptVal(char *, char *)**
  This method is used for getting the option value from the string given as parameter. Used together with isOpt(). The Method isn't vulnerable to string overflows as long as the source string has enough storage and is null terminated. The method has been tested and found to be working.

- **PICBOOL PicParams::getOptValString(char *, char *)**
  This method is used for getting the option value from the string given as parameter. Used together with isOpt(). The only difference to getOptVal() is that this method allows spaces in the option value. The Method isn't vulnerable

to string overflows as long as the source string has enough storage and is null terminated. The method has been tested and found to be working.

- **PICBOOL PicParams::addServer(char *)**
  This method is used for inserting the last successful server address. The data structure is LRU based and only overflows could result from BaseString limitations. The method has been tested and found to be working.

- **PICBOOL PicParams::addUser(char *)**
  This method is used for inserting the logon user used for the last successful connection a PICSearch server. The only overflows could result from BaseString limitations. The method has been tested and found to be working.

- **PICSHORT PicParams::getOptSec(char *)**
  This method is used for returning to the caller in which section the parsing of the configuration currently takes place. Method is used together with isOptSec(). The method has been tested and found to be working.

5 System Testing (MP)

5.1 Client

Tested a user interface and its operation in the property so SuperUser as Ordinary User. Setting the right of the anonymous user does not work. Short cuts of the user interface does not work.

The user interface has been tested by the menu.

Menu Item	**Test&result**
connect	**SuperUser/Ordinary User** - OK, tried to make connection to different kind of servers like Localhost 123, Local Host, cs.helsinki.fi and LocalHost. If the server does not exist, error message will be `Connection failure: server is down or does not exist.` Hot list of servers is working unstabile, not major thing, but the names of the servers could somethimes change.
disconnect	**SuperUser/Ordinary User** - OK, make the disconnection correctly.
Database management	**SuperUser** - .
User management	**SuperUser** - `Add user` OK, `Remove user` OK, `Change PW` OK, tested several times, `Database Authorization` OK, tested several times.
Change Password	**SuperUser/Ordinary User** - FAIL, changes works correctly only for SuperUser.
Quit	**SuperUser/Ordinary User** - OK.

Table 5: Server

Menu Item	**Test&Result**
Insert Query Image into Database	**SuperUser/Ordinary User** - OK, tried several times and different images.
Delete Selected Result Images from Database	**SuperUser/Ordinary User** - OK, tried several times.
Batch Insert Images	**SuperUser** - OK.

Table 6: Database

Menu Item	**Test&Result**
Available Images	**SuperUser/Ordinary User** - OK, tested several times, works perfectly.
Open Query Image	**SuperUser/Ordinary User** - OK, tested with several pictures.
Close Query Image	**SuperUser** - OK.
Save Query Image As	**SuperUser** - OK, tested with several pictures.
Get Selected Thumbnail	**SuperUser** - OK.
Get Selected Image	**SuperUser** - OK.
Image Attributes	**SuperUser** - OK.
Image Editor	**SuperUser** - OK, remember to check `Options` and its `EditClient Options` the location of WWW Browser and Image Editor.

Table 7: Image

Menu Item	**Test&Result**
Edit Client Options	**SuperUser/Ordinary User** - OK.
Edit Server Options	**SuperUser** - OK.

Table 8: Options

Menu Item	**Test&Result**
On Active Server	**SuperUser/Ordinary User** - OK.
On Selected Database	**SuperUser/Ordinary User** - OK, no information from database (yet).
On Selected Algorithm	**SuperUser/Ordinary User** - OK, print the information to the info screen.
On PICSearch	**SuperUser/Ordinary User** - OK.

Table 9: Info

Menu Item	Test&Result
Help	**SuperUser/Ordinary User** - OK, not in use (yet).

Table 10: Help

Client - testing of the windows and scroll bars of the user interface	SuperUser/Ordinary User
Query Results	**SuperUser/Ordinary User** - OK.
Info	**SuperUser/Ordinary User** - OK.
Databases	**SuperUser/Ordinary User** - OK.
Algorithms	**SuperUser/Ordinary User** - OK.
Query Image	**SuperUser/Ordinary User** - OK.

Table 11: Help

6 Performance Analysis (SK)

This section tries to give the reader a understanding of what the changes done by the `pics98` developers mean in terms of program performance. The following sub sections go through the components of the PICSearch software that have been affected by the changes.

6.1 Client - Program Module

The client itself was mainly changed to include new functionality. Not much has changed in the logic how the client module works. It still gets called from PicGUI and it receives messages from the network. Since no real changes were done in the functionality, it is acceptable to say that the performance has stayed the same. Only the memory footprint has become a bit larger.

6.2 Client - GUI

The GUI has been totally rewritten. The most noticeable speed increase comes from the fact that the way PicGUI uses the client has been changed. It no longer has complicated function calling, instead it uses return values. No formal measurements have been done but all `pics98` developers have agreed that the new GUI feels snappier and even if it really isn't, at least it's appearance has been improved to a great extend.

6.3 Client - Communication

The implementation of PICCache has had the most impact on the communication between the client and the server. The client's communication performance has gone down fractionally because on each query the array describing the cache contents has to be sent over to the server. The impact on performance is though quite minimal.

6.4 Server - Program Module

The server has been changed in order to know about PICCache. When doing database queries, just before the server will send result thumbnails to the client, it will go through the array that describes what is stored in the client side cache. When a thumbnail that is to be sent over the network is found in the cache it won't be send. The overhead going through the array is minimal. The scan is linear up to the hit or miss, since it is sorted.

6.5 Server - Database

The `pics98` developers had hoped to bring up database performance but failed to deliver. The database just won't scale, thus the performance of it has remained the same.

6.6 Server - Communication

It is in the communication from the server to the client that most new gained performance can be accounted to. Since the server isn't sending anymore thumbnails that the client has stored, a great fraction of the network load has been removed. The cache works quite well when it's properly sized and the user does several queries with different result sets on the same server. No formal measurements have been done but the cache statistics are shown in the GUI info screen for deeper analysis.

Now that the server stores the TCL parameter windows for all algorithms, they need to be sent over to the client. These parameter windows are sent with the init message that is sent only once when the connection to the server is established. This leads to minimal performance degradation.

6.7 Conclusion

Counting all changes that affect program performance, good or bad, the conclusion is that PICSearch v3 is more powerful and efficient then it's predecessor. Although the `pics98` developers have no statistical data or mathematical prove of this, it is a observation we have made during the project and the implementation phase in general. Feel free to evaluate it yourself.

7 Further Development and Visions for the Future

PICSearch v3 is already functionally quite a versatile programme. The most important improvement targets are the facilitation of the control of the algorithms and the improvement of the control of users.

From the point of view of the future it would be important to think who use the programme. If the users are mainly the developers of the algorithms, tools for the easy developing of the algorithms should be offered to them. On the other hand if the users are so "end users", easy searches from the database should be offered. Also for superuser should be offered an easy administration of the end users.

Thinking about the maintenance and expandability, it would be useful to design and to program the whole PICSearch again. As it is now the database realisation is not suitable to the searches which are directed to massive databases.

A Appendix