Customer: Kjell Lemström Instructor: Anita Kasari

pics98

Design Document

Samuli Kaski
Timo Koskinen
Jani Leinonen
Mika Pennanen

Helsinki, on the 27th of July 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. It is designed to serve as a testing platform for new retrieval algorithms, and is aimed mainly at professionals engaged in such development. However, PICSearch can also be used as a conventional image cataloging program.

The initial version of PICSearch was developed at Helsinki University of Technology. This version made it possible to query a local image database based on a query image. In version 2 (developed at University of Helsinki, Department of Computer Science) network support was added to the system. The `pics98` project aims to fix the bugs still left in the system and introduces image caching and an external algorithm management program.

This document specifies the design on which the implementation of version 3 of the system is based. Only features that are new or have changed are documented. The design document of version 1 (see [1]) and the implementation specification of version 2 (see [2]) contain a complete description of the system. Section 2 of this document gives an overview of the system. Section 3 describes the messages that the server and client use to communicate with each other. Section 4 discusses the client program and section 5 describes the server. Section 6 deals with algorithm management (embedding new retrieval algorithms). Section 7 describes the restrictions imposed by the system and lists some guidelines heeded by the developers.

2 Overview of the System (JL)

PICSearch consists of three parts which are client, server and algorithm management program. Client and server communicate through a TCP/IP network (see Figure 1). The algorithm management program runs independently.

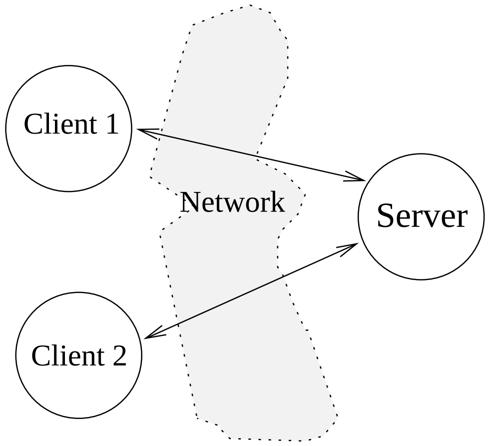

Figure 1: The model of the PICSearch client–server system.

2.1 Client

The client application consists of the graphical user interface, the program module and the cache (see Figure 2). The client program module and cache is implemented in C++ whereas the graphical user interface (GUI) implemented with Tcl/Tk scripts. The program module manages the communication with server. To accelerate the picture searches and minimize network traffic the client contains a cache where recently fetched thumbnail pictures are stored.

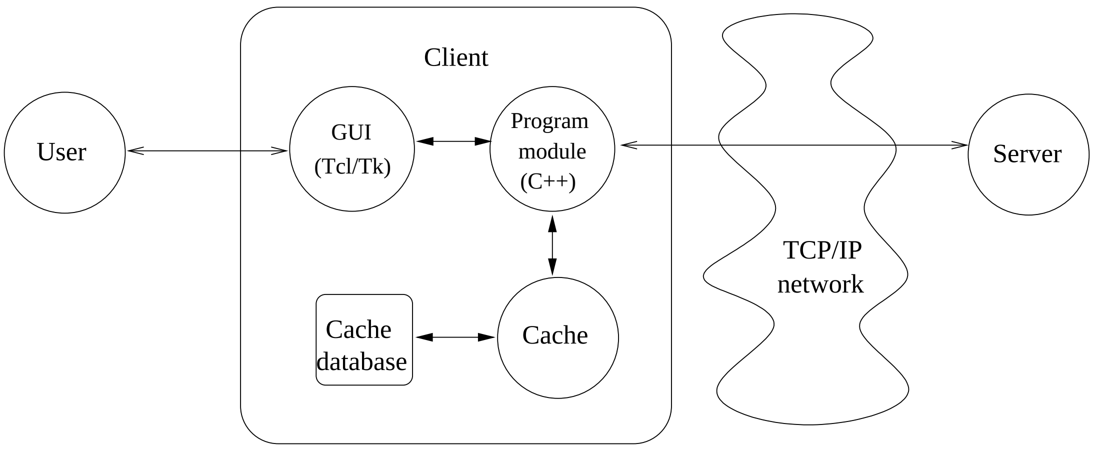

Figure 2: The model of PICSearch client.

2.2 Server

The PICSearch server manages picture and algorithm databases. It computes the signatures for each picture. It also retrievals the images (compares the signatures of the pictures). The server program is implemented in C++.

There is no user interface for the server. The server is used through a client.

The running server application consists of three types of processes. These processes are host process, database process and server process (see Figure 3).

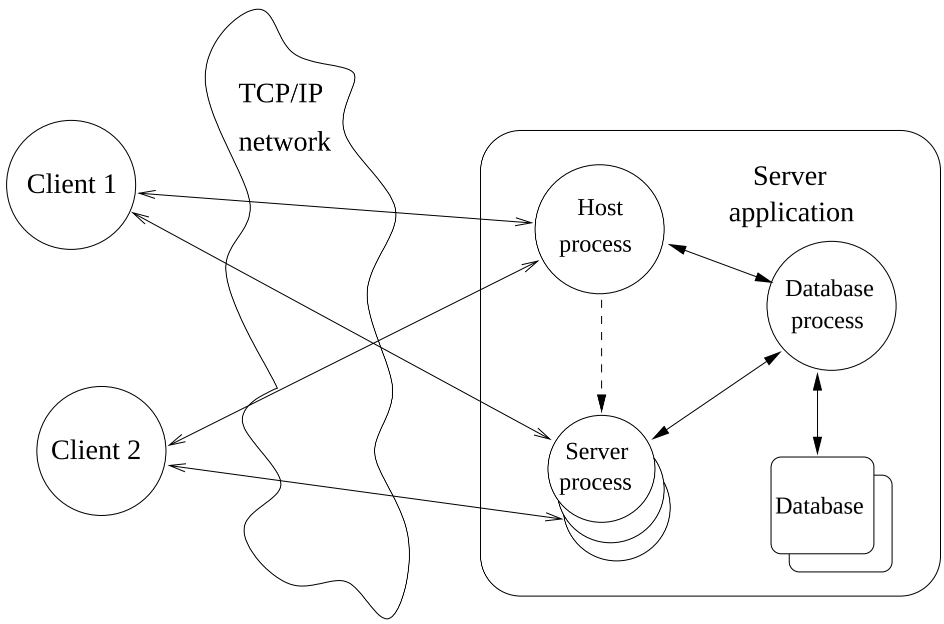

Figure 3: The model of the PICSearch server.

Host process listens to a predefined TCP port for connection requests sent by client applications. After receiving a request the host process checks for adequate user authorization before creating a dedicated server process for the client. Further communication from the client is handled by this dedicated server process on a separate TCP port.

Database process manages picture, algorithm and user databases.

2.3 Algorithm Management Program – AMP

Algorithm Management Program (AMP) is a totally new application for the PICSearch system. AMP is for adding and editing algorithms. It consists of graphical user interface and program modules. The graphical user interface is implemented in Tcl/Tk. Program module which does the actual work for algorithm management is programmed on C++. AMP makes changes in the source code of the server so it must be located on the same file system as the server itself.

3 Communication (JL)

PICSearch v3 is a distributed system which consists of client(s), server and Algorithm Management Program. Algorithm Management Program is a "stand alone program" which does not communicate with other programs. Client and server communicate over TCP/IP network using messages. These messages are specified in Implementation Specification of PICSearch v2 [2]. For a complete list of network messages between the client and the server, refer to Appendix A.

3.1 Message Specification Changes (SK)

Some of the network messages have to be changed because of the implementation of the client cache. For changes to existing messages from client to server, refer to Table 1.

Message	Change or implementation description
Message 02	**Start Query** - Two new additional fields are required. A field called **PICBOOL useResultSet** informs the server to use the provided array when deciding what pictures are to be sent over the network. The **PicArray resultSet** array describes what pictures are currently stored in the client cache, thus not needing to be sent over the network.
Message 04	**Get More Thumbnails** - As in Message 02, two additional fields are required. **PICBOOL useResultSet** and **PicArray resultSet**. The first informs the server to use the provided result set when deciding which thumbnail pictures are to be sent over the network. The second new field is the array itself describing the result set.

Table 1: Modified Client→Server messages

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

Message	Change or implementation description
Message 22	**Query Results** - Two new fields are required. **PICBOOL useResultSet** for acknowledging the client's request to use the provided array when deciding which pictures to send and the array **PicArray resultSet** itself. An entry in the array is either a picture object containing the picture data or a dummy object telling the client that the corresponding picture should be fetched from the cache.

Table 2: Modified Server→Client messages

4 The Client Program

The added functionality to the client program will be in the form of graphical user interface (GUI) changes and implementation of a configuration file and a client side cache entity. All these and additional problems that need to be corrected are described in the following subsections.

4.1 User Interface (TK)

The graphical user interface (GUI) of PICSearch is implemented with Tcl/Tk scripts. Communication between the main program and the GUI is handled via a C++ module called PicGUI. PicGUI implements an application specific Tcl command *transmit* that is used to send requests from the GUI to the main program. The *transmit* command has a variable number of parameters, the first of these always being the name of the requested command (see 4.1.1), the rest (if any) are parameters of the command. Depending on the command PicGUI calls other parts of the system to satisfy the request. PicGUI sends the results of the requests back to the GUI by setting Tcl variables. The trace mechanism provided by Tcl is used to monitor relevant variables for changes.

4.1.1 Parameters of the *transmit* Command

Table 3 includes details of all new or modified inputs for the *transmit* command. The first column of the table shows the name of the request and the second column shows the description of its parameters and whether the request is new to this version of PICSearch or a modified version of an existing request. (Usage from Tcl: `transmit request-name ?arg arg ...?` – note that argv[0] always equals "transmit" and argv[1] equals the name of the request, e.g. "createDatabase".)

Parameters	Description
createDatabase	**Modified**, creates a new database
argv[2]	Database path
argv[3]	Database name
agrv[4]	Database info string
getDatabaseInfo	**New**, retrieves the description of a database
agrv[2]	Database name
setDatabaseInfo	**New,** sets the description of a database
argv[2]	Database name
argv[3]	Database info string
setAlgorithmInfo	**New**, sets the description of an algorithm
argv[2]	Algorithm name
argv[3]	Algorithm info string
setDatabaseAlgs	**New**, sets the algorithms used by a database
argv[2]	Database name
argv[3...n]	Algorithm names 1...n-2
scaleImage	**New**, scales an image to the requested size
agrv[2]	Image name
agrv[3]	Image index
agrv[4]	Width of the scaled image
agrv[5]	Height of the scaled image

Table 3: New and modified *transmit* inputs.

4.1.2 Changes to Methods

To facilitate the implementation of new GUI functionality (server hot list, database descriptions, image scaling, etc.) several new methods are needed and some existing methods have to be modified. This subsection describes these changes. Old and new method declarations are shown for each modified method followed by a description of how the functionality of the method will be changed.

Modified Methods in the `client` module

- `bool PICClient::CreateDatabase(char *path, char *databaseName)` becomes
  `PICBOOL PICClient::CreateDatabase(char *databaseName,`
  `char *databaseInfo)`

  The method creates a new picture database. The new parameter *databaseInfo* contains the description of the database. The database creation method is no longer given a separate *path* parameter.

- `void PICClient::GetAddress()` becomes
  `BaseString PICClient::GetAddress()`

  The method returns the HTTP address of the server instead of calling `PicGUI::setServerAddress()`.

- `bool PICClient::GetAlgorithmInfo(char* alg)` becomes
  `BaseString PICClient::GetAlgorithmInfo(char* alg)`

  The method returns the description of the algorithm instead of printing it by using `INFO_PRINT()`.

- `PICBOOL PICClient::GetDbAlgorithms(BaseString dbName)` becomes
  `BaseStringArray PICClient::GetDbAlgorithms(BaseString dbName)`

  The method returns the algorithm names of the database instead of calling `PicGUI::setSearchAlgorithms()`.

- `PICBOOL PICClient::GetAllDbNames(void)` becomes
  `BaseStringArray PICClient::GetAllDbNames(void)`

  The method returns the names of all databases instead of calling `PicGUI::setAllDbNames()`.

- `PICBOOL PICClient::GetDbNames(void)` becomes
  `BaseStringArray PICClient::GetDbNames(void)`

  The method returns the database names instead of calling `PicGUI::setDbNames()`.

- `PICBOOL PICClient::GetDbUserRights( char* dbName, char* userName)` becomes
  `BaseStringArray PICClient::GetDbUserRights(char* dbName, char*`
  `userName)`

  The method returns the user rights instead of calling `PicGUI::setDbUserRights()`. If an error occurs while getting the rights, a null `BaseStringArray` is returned.

- `PICBOOL PICClient::GetDbUsers()` becomes
  `BaseStringArray PICClient::GetDbUsers()`

  The method returns the database users instead of calling
  `PicGUI::setDbUsers()`.

- `void PICClient::GetInitData()` becomes
  `BaseStringArray PICClient::GetInitData()`

  The method returns the initialization data instead of calling `PicGUI::setInitData()`. The format of the return value is described in Table 4.

Array Component	**Description**	**Legal values**
1	Superuser rights	"0" or "1"
2	Can change password	"0" or "1"
3	Number of algorithms	Any BaseString containing a positive integer
4...n	Algorithm names, if any	Any BaseString
n + 1	Number of servers in hot list	Any BaseString containing a non-negative integer
n + 2...m	Server names, if any	Any BaseString containing a legal server name

Table 4: The `BaseStringArray` returned by `PICClient::GetInitData()`.

- `void PICClient::GetServerAdvertisement()` becomes
  `BaseString PICClient::GetServerAdvertisement()`

  The method returns the server ad instead of calling
  `PicGUI::setServerAdvertisement()`.

- `PICBOOL PicGUI::QueryProgressReport(long percent)` becomes
  `PICBOOL PicGUI::ProgressReport(PICULONG completed,`
  `PICULONG total)`

  The method can be used to display a progress report for any operation. The *completed* parameter describes how much of the *total* work of the operation has been done. The progress of a query operation, for example, can be reported by calling `ProgressReport` with parameters telling how many pictures have been scanned of the total number of pictures in the database.

- The parameter of type `PicIxCollect` received by the
  `PICBOOL PicMain::getParams(PicIxCollect &params)` and
  `PICBOOL PicMain::setParams(PicIxCollect &params)` methods
  is changed to the format described in Table 5.

Component	Description	Legal values
1	Confirm changes in image editor	"0" or "1"
2	Copy images to database physically	"0" or "1"
3	PicGUI debug messages	"0" or "1"
4	PicCore debug messages	"0" or "1"
5	PicAlgorithm debug messages	"0" or "1"
6	PicDb debug messages	"0" or "1"
7	Default width of a new image	"20"-"1024"
8	Default height of a new image	"20"-"768"
9	WWW browser name	Any string
10	Number of query result images	**TO BE DEFINED!**
11	Number of thumbnails received at a time	**TO BE DEFINED!**
12	Minimum match percentage	"0.0"-"100.0"
13	Ask algorithm parameters	"0" or "1"
14	Show server advertisement at connect time	"0" or "1"
15	Scale query image	"0" or "1"
16	Main window size	"1", "2" or "3"
17	Main window X position	**TO BE DEFINED!**
18	Main window Y position	**TO BE DEFINED!**
19	Thumbnail width	"128"
20	Thumbnail height	"128"
21	Maximum number of open query images	Any string containing a positive integer
22	Cache size (number of thumbnails)	Any string containing a positive integer
23	Image editor name	Any string containing a legal path and filename

Table 5: The `PicIxCollect` parameter received by `PicMain::getParams()` and `PicMain::setParams()`. Options in the lower half of the table are new to PICSearch v3.

New Methods

- `BaseString PICClient::getDatabaseInfo(char *dbName)`

  The method receives a database name as a parameter and returns the description string for the database.

- `PICBOOL PICClient::setAlgorithmInfo(char *algName, char *info)`

  Sets the description of the given algorithm. Returns `boolFalse` if an error occurred, `boolTrue` otherwise.

- `PICBOOL PICClient::setDatabaseAlgs(char *dbName, BaseStringArray dbAlgs)`

  Sets the search algorithms that are applicable to the given database.

- `PICBOOL PICClient::setDatabaseInfo(char *dbName, char *info)`

  Sets the description of the given database. Returns `boolFalse` if an error occurred, `boolTrue` otherwise.

- `PICBOOL PicMain::scaleImage(Picpicture origImg, PicPicture &scaledImg, PICULONG width, PICULONG height`

  The method receives a source image (*origImg*) and scales it to the dimensions specified by *width* and *height*. The resulting image is returned in *scaledImg*. The method returns `boolTrue` if the scaling completed successfully, `boolFalse` otherwise.

4.2 Client Cache (SK)

The cache is an intermediate storage for pictures. Once a picture has been received from the server, it is stored in the client cache. If any further queries occur for the same picture, the cache will function as the source instead of the server.

4.2.1 Description

The cache is defined in the PICCache class (see subsection 4.2.3 for the definition). The PICCache object maintains a static set of data structures which are allocated at program startup after the GUI has been initialized. Since the cache is not dynamically sized, it is necessary to appropriately size it. Testing will determine the appropriate default size for the cache. The size of the cache is determined by the amount of thumbnail pictures stored in it, which is a configurable option in the client configuration file.

4.2.2 Data Structures

A thumbnail picture is identified by several attributes, of which an id number and a database id give it a unique handle within one server. This handle is used as a hash key in an allocated static array consisting of pointers to cache objects. The cache objects define pointers to other cache objects so that it is possible to create a last recently used (LRU) list and to avoid hash collisions. When disconnected from a server the cache will be purged to assure consistency. The cache object is defined in the CacheObj class (see below for definition).

If the hash function returns a hash key pointing to a slot which already has one or more cache objects, a descision must be made what to throw out of the corresponding chain. A balanced descision making of looking at times the cache object has been used and at the last usage time of the object will determine which object will be thrown out of the chain. The chain size is limited in order to avoid one chain becoming too long thus making the structure comparable in performance to a linked list.

The main advantage gained by implementing the client cache comes from the picture queries caused by scrolling of the result picture set with the GUI. This leads to a situation where it becomes less important to optimize the picture queries in a situation where the user establishes connections to several servers (disconnecting from the old server first).

```
class CacheObj {
public:
    CacheObj(); // constructor
    ~CacheObj(); // destructor

    CacheObj *next; // next in chain, NULL if last
    CacheObj *prev; // previous in chain, NULL if first
    static long databaseId; // .. for hashing
    static long pictureId; // and ..
    static time_t inserted; // insert time
    long hitCount; // for cleanup
    PicPicture Pic; // same format as in the database
}
```

4.2.3 Implementation

The PICCache class is not derived from any other object. It is part of the data section in the PicMain object. The cache is created from `PicMain::guiHasStarted()` and destroyed at program (GUI) exit.

```
class PICCache {
```

```
private:
    static long arraysize; // configurable
    static const CacheObj **array; // allocated at startup
public:
    PICCache(); // dummy constructor
    ~PICCache(); // dummy destructor
    buildCache(); // constructor, creating array
    buildCache(long size); // create array of n
    destroyCache(); // clean up
    PicPicture &getThumb(long databaseId, long pictureId);
    static bool putThumb(long databaseId, long pictureId, PicPicture &pic);
}
```

4.2.4 Communication

The client program uses the cache entity simply by calling public methods provided by it. In normal operation the client simply asks the cache to supply it a thumbnail picture. If the cache has a copy of the requested thumbnail, it is returned.

For a flow chart representation of how the cache is used see Figure 4.

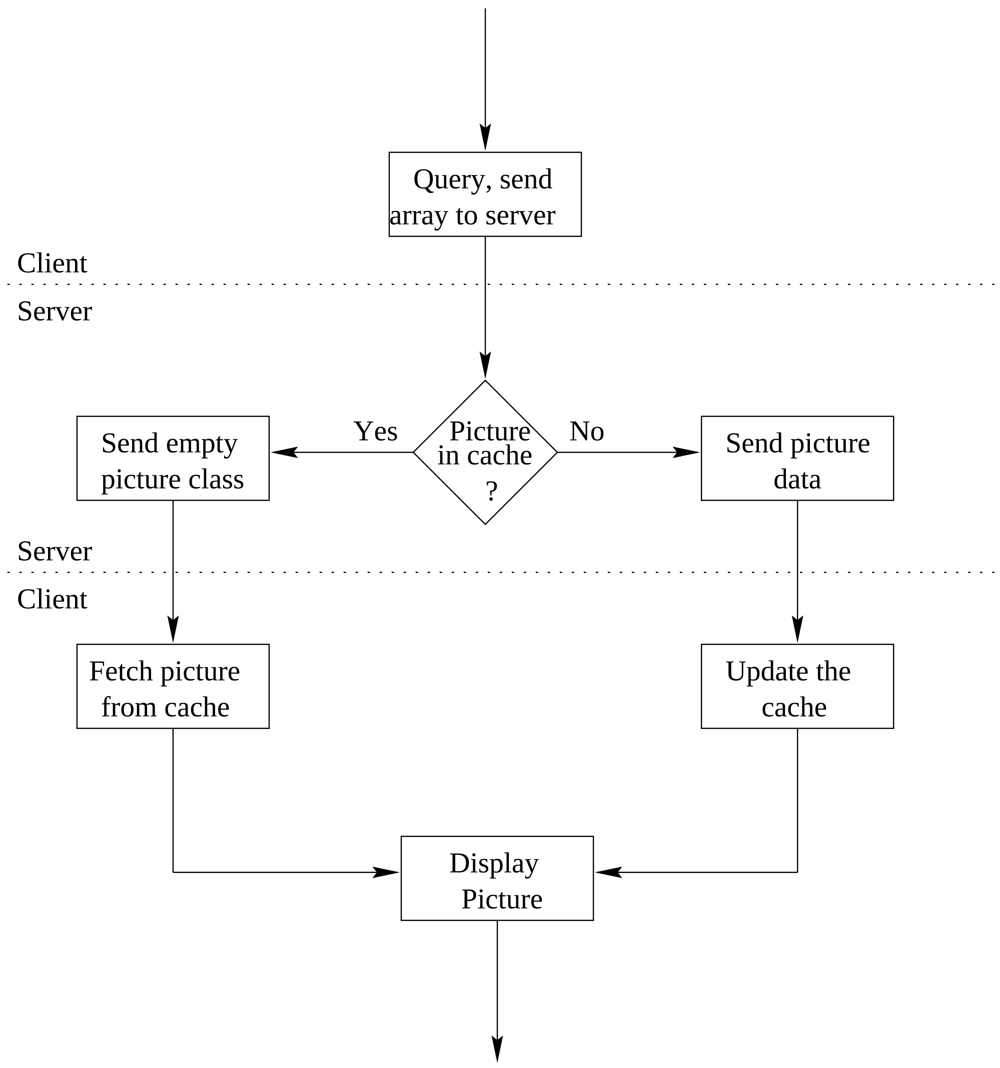

Figure 4: Algorithm for picture queries

4.2.5 Interface Changes

Sofar the client has simply requested the server to send it a set of pictures, starting from position N and giving a range of M. In order for the server to know which pictures are stored in the client cache, the client will provide an array describing the cache content when doing server queries. Only pictures not stored in the client cache will be actually sent over the network. In place of pictures that are already stored in the client cache the server places empty picture objects. This informs the client that the corresponding picture should be fetched from the cache.

Both a modified `PICClient::StartQuery()` and a modified `PICClient::GetMoreThumbNails()`

are required. Both methods differ from old versions of these methods in that they will pass along the array describing the contents of the client cache.

```
class PICClient {
    // .. //
    static PICBOOL StartQuery(BaseString algName, PicArray &genOpts,
PicPicture &pic, PICBOOL useResultSet, PicArray &resultSet)
    static PICBOOL GetMoreThumbNails(int numOffFirst, PicArray &genOpts,
PICBOOL useResultSet, PicArray &resultSet)
    // .. //
}
```

4.3 Client Configuration (SK)

The PICSearch client program has several attributes which affect how the graphical user interface (GUI) appears and functions. In addition to the GUI settings, the client configuration file has settings also for other client specific options such as the option which determines how many pictures fit in the client cache.

The initial settings and settings defined by the user must be saved for later use and retrieved when the client program is invoked.

4.3.1 File Format

The configuration file consists of several entries divided into reasonable categories. A setting in the configuration file consists of a variable and the value of that variable. The file is parsed at program startup. The logical content of the configuration file is as follows

picsearch.cfg

[gui]
resolution=
windowpos=
..
[runtime]
platform=
performance=
nwperf=

..
[used servers]
server1=
server2=
..
[user]
logonuser1=
..
[cache]
size=
..

4.3.2 Implementation

The functions for retrieving and setting the client options can access the same variables which the GUI uses. These functions must also be able to parse the ASCII format configuration file and provide some level of error detection.

In addition to the retrieval of the client specific settings we must also store the settings when requested to do so by the client program. Only the window position should be automatically saved at program exit. The client parameters are set during program startup in the PicMain object, more precisely in the constructor of the `PicMain::globData` object (globData is defined as PicMainSharedData).

The PicParams object contains all the parameters that are used by the client program. The new parameters will be added there.

```
class PicParams {
    // .. //
    BaseString db[10];
    PICULONG resx;
    PICULONG resy;
    PICULONG posx;
    PICULONG posy;
    BaseString platform;
    PICULONG perfindex;
    PICULONG nwperf;
    BaseString serverlist[10];
    BaseString user[10];
    BaseString os_user[10];
```

```
    PICULONG cache_size;
    // .. //
}
```

The methods for reading and writing settings from and to the configuration file are defined as new methods to the PicParams class.

```
class PicParams {
    // .. //
    PICBOOL readSettings();
    PICBOOL writeSettings();
    // .. //
}
```

Both methods have access to the same global data as PicParams, thus not requiring any parameters. The `PicParams::readSettings()` is called before the GUI is displayed and along the same logic `PicParams::writeSettings()` is called before the GUI is terminated.

4.4 Bug Fixes for Client (JL)

- Space characters in the names of databases cause the program to make two (or more) distinct databases. This will be corrected by covering the space characters with underline characters.

5 The Server Program

The changes that will be done to the server program by the `pics98` developers cover the network communication between the client and the server, the server databases and algorithms and their handling. Of these changes all but the first is covered in the following subsections. For changes in the network communication layer, refer to Section 3.

5.1 Server Databases (SK)

5.1.1 Current Implementation

The PICSearch v1 and v2 programs have straightforward databases, that consist of binary files which have an known amount of records and indexes pointing to these records. The PICSearch server program version 3 will continue to use this ideology, however increasing the overall database query performance by developing better routines for the handling of the indexes and their relating data.

For a representation of the current implementation see Figure 5.

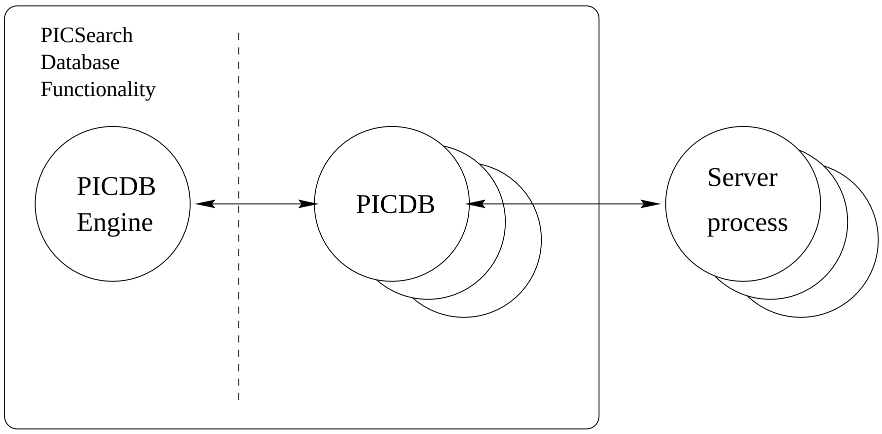

Figure 5: Current database implementation

5.1.2 Changes to Current Implementation

Database reads can be optimized at the application level by implementing read ahead, writes can be optimized by implementing delayed writes and seeks can be optimized by combining several seeks in a less costing matter by implementing a seek table.

The problem with these techniques is that most modern operating systems, especially Linux, already use these techniques, thus implementing them at the application level could even bring performance down. Testing will define how far these operations should be optimized at the application level. All of the changes will be implemented to the database engine, PICDBEngine. The affected methods which will be changed are listed below.

```
class PICDBEngine {
```

```
    // .. //
    long PICDBEngine::CreateKey( PICDBOperation& o )
    int PICDBEngine::Insert( PICDBOperation& operation )
    int PICDBEngine::Remove( PICDBOperation& operation )
    int PICDBEngine::Fetch( PICDBOperation& operation )
    int PICDBEngine::Update( PICDBOperation& operation )
    int PICDBEngine::Pack( PICDBOperation& operation )
    char* PICDBEngine::GetTableName( long database, long table )
    char* PICDBEngine::GetTableIndexFile( long database, long table )
    char* PICDBEngine::GetTableDataFile( long database, long table )
    int PICDBEngine::CreateDatabaseIdentifier( PICDBOperation& operation )
    int PICDBEngine::ReadDatabases()
    // .. //
}
```

After a response has been given to the calling instance (a PICDB process), the database engine can resume with the optimization work while constantly polling for new requests.

5.1.3 Database Architecture

The `pics98` developers will attempt to keep the database architecture identical to that of PICSearch v2. The changes implemented to the algorithm handling will however result in additional flags in some of the database records. Unless some already present entries in the data structures can be removed, additional fields will have to be added to the structures. This would result in tables, indexes and databases themselves to become incompatible at the binary level with the v1 and v2 databases.

5.2 Algorithms (MP)

The server database contains the algorithms which can be used for the picture search. The started server checks from the external file (flag.amp) if algorithms have been added, removed or changed. If algorithms have been modified, the server computes the signatures out of the pictures for the new algorithms. The PICDB manages the database management operations such as additions, removals and changes. The algorithms are added to the server by using Algorithm Management Program (AMP). The server informs of the algorithms available to the client by sending information about the algorithms.

5.2.1 Adding Algorithms

The started server notices that the algorithms have been added. The server does not compute a new signatures to the databases but leaves computation later. The server notifies the new algorithms (which have not been computed) to the client. The new algorithms will be computed to the database when the user chooses to use a new algorithm.

5.2.2 Removing Algorithms

The server notices if algorithms have removed in which case it removes the signatures of the corresponding algorithms.

5.2.3 Updating Algorithms

The server notices that the algorithms have been modified in which case it recounts to the changed algorithms signatures to all the databases. Before it begins to renew signatures to calculate, it removes the old signatures of the algorithm and only it forms the new.

5.3 Bug Fixes for Server (JL)

- ID and name searches will be corrected to give right results.
- The changing of the rights of anonymous user will be made available.

6 Algorithm Management

6.1 The User Interface (TK)

The graphical user interface of the Algorithm Management Program (AMP) is implemented using an approach identical to that used to create the GUI of the main program (see 4.1). The only differences are that the C++ module implementing the *transmit* command used by AMP is called AMPGUI and the parameters to this command are different.

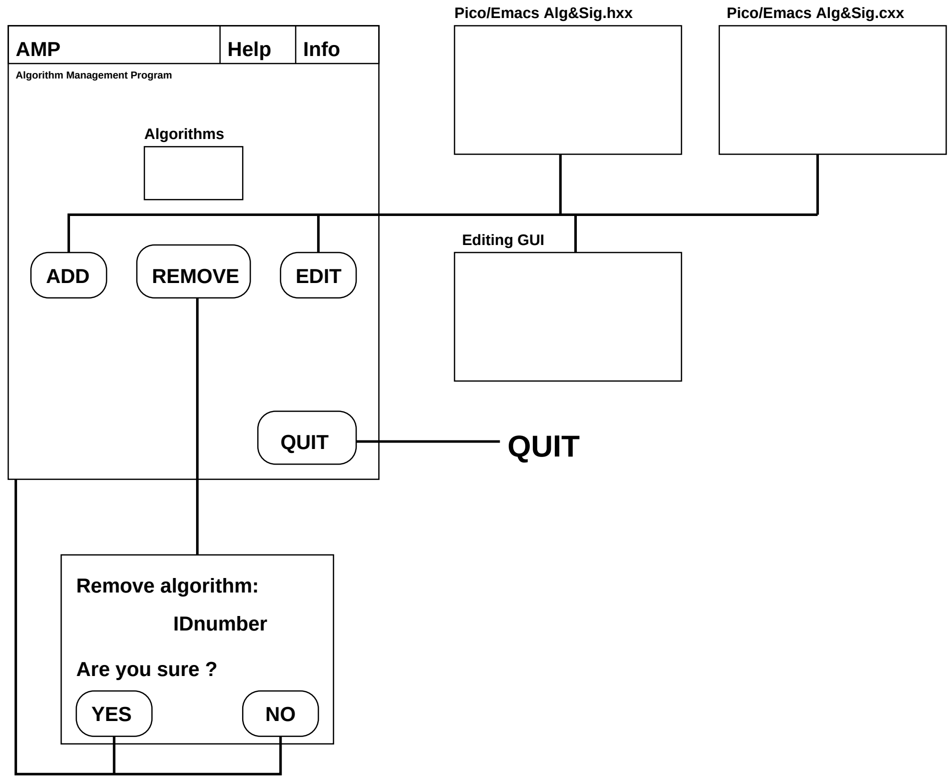

Figure 6: The User Interface of AMP

Parameters of the *transmit* Command

Table 6 describes the inputs accepted by the *transmit* command of AMP. The first column of the table shows the name of the request and the second column describes its parameters.

6.2 Realisation Logic (MP)

Algorithm Management Program (AMP) can be used to facilitate updating of the algorithms. AMP is separate, external programme. New algorithms can be added by simply deriving a new C++ class from a common base class redefining a couple of functions and recompiling the system.

Parameters	Description
getAllAlgNames	Retrieves the names of all the algorithms available at the server
no parameters	
addAlg	Adds a new algorithm. Note that if the algorithm has no parameters then only argv[2] exists
argv[2]	Name of the algorithm to be added
argv[3]	Algorithm parameter 1: a Tcl list containing the parameter's name, type (bool, int, double, string), default value and optionally minumum and maximum values
argv[4...n]	Algorithm parameters 2...n-2
removeAlg	Removes an algorithm
argv[2]	Name of the algorithm to be removed
changeAlg	Update an algorithm
argv[2]	Name of the algorithm to be updated
argv[3...n]	Algorithm parameters (if any), just as in *addAlg*

Table 6: Inputs accepted by *transmit*.

6.2.1 Adding Algorithms

Before the use of the addition function, AMP will ask the name of the new algorithm. The function gets the new algorithm as a parameter.

The addition function

```
void add(char name);
```

Operation of the addition function:

- copy PicNameAlg.cxx to name.cxx
- copy PicNameAlg.hxx to name.hxx
- open name.hxx by Pico/Emacs
- open name.cxx by Pico/Emacs
- open PicAlgDefs.hxx by Pico/Emacs
- editing parameters
- add to the external flag file (flag.amp) information of the new algorithm

6.2.2 Removing Algorithms

Before the use of the removal function, the user has to choose the algorithm from the list. The function gets the new algorithm as a parameter.

The removal function

```
int remove(int name);
```

Operation of the removal function:

- remove name.cxx
- remove name.hxx
- remove "#define name ID" from PicAlgDefs.hxx
- add to the external flag file (flag.amp) information of the removed algorithm

6.2.3 Updating Algorithms

Before the use of the revision function, AMP will ask the name of the changed algorithm. The function gets the changed algorithm as a parameter.

The revision function

```
void change(char name);
```

Operation of the revision function:

- open name.cxx by Pico/Emacs
- open name.hxx by Pico/Emacs
- open PicAlgDefs.hxx by Pico/Emacs
- add to the external flag file (flag.amp) information of the the changed algorithm

6.2.4 Quitting of Algorithm Management Program

If nothing has been done to the algorithms, AMP will quit without recompiling the system. The system will be recompiling by AMP, if something has added, changed or removed.

7 Restrictions and Guidelines (JL)

7.1 Standards and Programming Languages

Implementation language is mainly C++ except the user interfaces are impelemented on Tcl/Tk (7.6/4.2).

7.2 Programming Style and Naming Practises

Names of the **classes** are written with a capital initial letter. If the name of the class includes several words, each one will be written with a big initial letter. The names of all the classes begin with the letters, "Pic".
For example:

```
class PicSignature : public PicObject
```

Names of the **variables** are written with a small initial letter. If the name of the variable includes several words, these will be written with a capital initial letter.
For example:

```
PicSignature *computedSignature = NULLPTR;
```

Macros are written totally with upper-case letters. If a name includes several words, they will be written, either as connected together or separated with underline characters. typedef are written either as macros or such as the names of the class.
For example:

```
#define NULLPTR 0
#define MAKE_PICANYCLASS(className, classIdName, base)
typedef unsigned short int PICUSHORT;
typedef PicMyClass::CaseSensitivity PicCaseSensitivity;
```

Indents are 4 space characters in their size.

References

[1] **Pyry Kuusi, Jouni Korte, Pekka Päiväkumpu and Pasi Kyheröinen:** *"PICSearch - Technical Specification", Helsinki University of Technology*, 1996 *(in Finnish)*
`http://www.hut.fi/~pkuusi/oht/documents/te.html`

[2] **P. Kanerva, S. Koponen, S. Männynsalo, J. Riihimäki, J. Teinilä, T. Virtanen:** *"Implementation Specification of PICSearch v.2", University of Helsinki, Department of Computer Science*, 1997
`http://www.cs.Helsinki.FI/research/picsearch/v2/index.html`

A Appendix

Network Messages

Message 00 - Init
Message 01 - Timeout

Client→Server Messages

Message 02 - Start Query
Message 03 - Stop Query
Message 04 - Get More Thumbnails
Message 05 - Insert Picture
Message 06 - Get Picture
Message 07 - Delete Picture
Message 08 - Disconnect
Message 09 - Add/Remove Algorithms
Message 10 - Get Users
Message 11 - Change User Rights
Message 12 - Add User
Message 13 - Remove User
Message 14 - Change Password
Message 15 - Create Database
Message 16 - Add Database
Message 17 - Remove Database
Message 18 - Delete Database
Message 19 - Get Server Options
Message 20 - Set Server Options

Server→Client Messages

Message 21 - Query Progress Report
Message 22 - Query Results
Message 23 - No Query Matches
Message 24 - Failed To Complete Request
Message 25 - No More Query Results
Message 26 - Operation Complete
Message 27 - Picture Data
Message 28 - Algorithm Progress Report
Message 29 - Userdata
Message 30 - New User Rights
Message 31 - Password Incorrect
Message 32 - New Database
Message 33 - Server Options
Message 34 - New Picture ID