



FindMyDouble

Delft, June 2001

Faculty of Information Technology and Systems, Delft University of Technology

D. Hendriksen, IF 9325091

R. Nachtegaal, IF 9594185

R. J. de Pauw, IF 9644077

Contents

Preface	2
Chapter 1. Introduction	3
Chapter 2. Procedures	4
2.1. Schedule.....	4
2.2. Course of events	4
2.3. Face recognition	5
2.4. Facial model	6
2.5. Search algorithm	8
2.6. Database connection.....	10
2.7. Scalability	10
2.7.1. Server set-up.....	11
2.7.2. Data traffic.....	11
2.7.3. Disk space requirements.....	12
2.8. Security.....	13
2.8.1. Current set-up.....	13
2.8.2. Threats to the system	13
2.8.3. Recommended set-up	13
Chapter 3. Design	15
3.1. Global design.....	15
3.1.1. Diagram.....	15
3.2. Database design.....	17
3.2.1. person table.....	17
3.2.2. data table.....	17
3.2.3. dd table.....	18
3.2.4. const table	18
Chapter 4. Programming	19
4.1. The three applications	19
4.1.1. ScreenApplet.....	19
4.1.2. PersonApplet.....	22
4.1.3. AdminApplet	23
4.2. Scripts.....	26
4.2.1. Upload.cgi	26
4.2.2. Update.cgi	26
Chapter 5. Recommendations	27
5.1. For online use.....	27
5.2. For a more automatic application	27
Chapter 6. Conclusion	29
Chapter 7. Literature	30

Preface

The FindMyDouble system is, at the time of this writing, starting to take shape in the form of several applications. All the pieces are falling in place and it is rapidly becoming a system too huge to finish in the short time we have left. The possibilities seem endless; the ever-growing to-do list is bloated with possible improvements. For now, we've tried to fix all the major and even some minor bugs but as you probably know, bugs are nasty little creatures. They attack at moments you least expect and can hide in places you've never even dreamt of. All in all it has been a process of trial and error. We have learned much, and we believe that we've achieved more than we could imagine at the onset of the project. On the other hand, if we knew then what we know now, we could have done so much more. The prototype described in this report can be found at <http://www.vergelijk.nl/~fmd/>.

Originally FindMyDouble started as a joke between 2 colleagues, Dr. Richard van Wezel and Drs. Bart Borghuis, both working at the Neurobiology Department of Utrecht University. They were having lunch together one day and came up with an idea for an application that would be able to compare a face with a set of other faces and tell which face in the set matches best: FindMyDouble. After a period of research, an apparently simple solution was found that would enable an application to do just that. This technique is based on scale-space theory and Gaussian filters. Dr. Richard van Wezel, also co-founder of Vergelijk.nl B.V., asked one of the programmers at Vergelijk.nl B.V. (Gijs Hollestelle) to build this application for him, since the application might also be used as promotion material for Vergelijk.nl B.V. It soon became apparent that the workload was too much for a single programmer to handle and since other more urgent work for Vergelijk.nl B.V. was pending, the project was suspended.

By then Robert-Jan de Pauw (a new programmer at Vergelijk.nl B.V. and student at the Delft University of Technology) had heard of this project. Since he and his fellow students, Rikkert Nachtegaal and Dennis Hendriksen, were looking for a project for their third year practical they decided to take on this project. That's when all the 'fun' began.

We'd like to thank Dr. Richard van Wezel and Drs. Bart Borghuis for coming up with the FindMyDouble idea in the first place, for the support, feedback and improvements, for being enthusiastic and for buying our food at the faculty restaurant.

We'd also like to thank Drs. Dr. L.J.M. Rothkrantz for all the useful information, feedback and ideas he gave us and for the pleasant conversations we had while driving to and from the city of Utrecht.

Further more, we'd like to acknowledge the contribution of Gijs Hollestelle. The upload script he wrote and the picture he gave us. The fun we shared while looking at his mug shot was invaluable.

Tjibbe de Jong has been so kind to provide us with a couple of FindMyDouble logos and banners.

Finally, we'd like to thank Marieke, Esther, Judith, Ruud, Fer, Antal, Timo and the people at <http://gathering.tweakers.net> for testing this system.

Chapter 1. Introduction

The purpose of this practical work is making an application for the web site <http://www.findmydouble.com>. Through this web site, visitors can search for their doubles by submitting their facial images. After submission, the application searches the image database for the most corresponding face. When the best match is found, the image is shown to the user.

Since this system will be part of a web site, it is required to run on as many platforms as possible. The user must be able to visit the FindMyDouble web site without having to install additional software. Therefore, we've chosen to develop the application as a standalone JAVA-applet^[7]. Another advantage of JAVA is that we've already had some experience gained during the courses IN137, IN138 and IN133.

This system will be a prototype where the user still has to do a lot of the work. The functionality that is not supported by this prototype is described in Chapter 5 Recommendations.

This system was made by order of:

- Neurobiology Department
Dr. Richard van Wezel is a member of the Neurobiology Department of Utrecht University and the Helmholtz Institute. In this group researchers are investigating how the brain processes visual information. A reason to participate in the FindMyDouble project is to use it as a tool to develop new models for face recognition on a biological basis.
- Vergelijk.nl B.V.
JeanPaul Soethout, Tjibbe de Jong and Richard van Wezel founded Vergelijk.nl B.V. in March 2000. At the comparison shopping site <http://www.vergelijk.nl> visitors can compare prices of books, compact disks, games, etc. Furthermore, services of internet-shops are compared to help customers make the best choice where to buy their products. At the moment Vergelijk.nl B.V. has expanded into Europe, as a joint venture in Finland, and with sites in Germany and England. Vergelijk.nl B.V. currently has 9 employees (programmers). The interest for the FindMyDouble project is for use as a marketing tool.

Chapter 2. Procedures

In this chapter the procedures and theories used in the FindMyDouble project are discussed. The course of events describes what has been done during this project. After that the face recognition process is described. The facial model, which defines the distances used for the double finding, is dealt with in paragraph 2.4 together with the weight model. The search algorithm, which is explained in paragraph 2.5, is used to find the best-looking person (the double) from the database. The algorithm uses the distances defined by the facial model to do so. Paragraph 2.6 deals with the database connection between the database server and the applications of FindMyDouble. The last two paragraphs deal with scalability and security issues in the FindMyDouble system.

2.1. Schedule

Date	Week	Week of work	Activity
05 feb	6	1	Read in
12 feb	7	2	Find out about Gabor filters
19 feb	8	3	Find out about Gabor filters
26 feb	9	4	Begin GUI
05 mar	10	5	Finish GUI, find out about data representation and the database
12 mar	11	6	
19 mar	12	College free week	--
26 mar	13	Exam week	--
02 apr	14	7	Implementation database connection and face-to-vector conversion
09 apr	15	8	
16 apr	16	9	
23 apr	17	10	Prototype
30 apr	18	Holiday	--
07 may	19	11	Optimise and automate the system
14 may	20	12	
21 may	21	13	Finish the documentation and prepare the presentation
28 may	22	14	
04 jun	23	College free week	--
11 jun	24	Exam week	--
18 jun	25	Exam week	Presentation at the Neurobiology Department of Utrecht University
25 jun	26	Exam week	--

Table 1: Planning

2.2. Course of events

The FindMyDouble project started on 5 February 2001. The first week was spent in the library collecting information concerning face identification, Gabor functions and PCA^[1],^[4],^[5]. During the second and third week of the project, an applet has been built, with the information found in the library and the data gathered from the Internet, which could draw two- and three-dimensional Gabor functions. Gabor functions (edge detectors) are used as compression of luminance values into contour values

that can be used in combination with Principle Component Analysis for face similarity measures. Also an application was made that can convert RGB to greyscale images. The output of the RGB-to-Greyscale application was used as input for the Gabor applet. After modifying this Gabor filter several times, to get a result in which only the edges were clear, the applet was finished.

The first appointment with Drs. Dr. L.J.M. Rothkrantz took place February 27th. During this project he is our supervisor at the faculty Information Technology and Systems at the Delft University of Technology. This conversation changed the approach to this project dramatically. The advice was to begin with a simple prototype that relied upon user interaction to establish any initial parameters. This approach was different from the original objective that the computer accomplishes this automatically. The working on the filters was stopped and started working on a primitive interface that could be used as a foundation for a more complex and automated application, since the time for this project was limited. The goal was that the prototype had to be as simple as possible, with a large amount of user interaction. During the making of this prototype, quite some JAVA security-features had to be worked out. It appeared a new compiler version was needed because the one used contained a nasty bug (user input wasn't handled correctly in some cases). To make everything a bit more efficient, the jobs were split up in three parts: the interface itself, the database and the connection between the applet and the database and a zip applet to zip pixel-data before sending it from the client to the database thus saving bandwidth.

By then it was time to come up with an algorithm that was able to express the likeliness of two faces. After working it out on paper first, it was implemented and integrated with the other components. The prototype was now ready for testing. During this phase the prototype was improved and the result given when finding doubles was evaluated. After extensive testing, the points that have to be selected by the user were modified, so it would give better results. Using this new point-set, weights to each of the fifteen vectors were added thus improving the double finding by emphasising the static distances. In the mean time an administration applet was developed. It is made to browse through the database with a graphical interface and to make modifications to the weight-system etc. In the last two weeks the usability of the interface was improved and the double-browser is introduced, which enabled the user to search for second, third, fourth, etc. best doubles.

2.3. Face recognition

In the ideal situation, the user has to do nothing but selecting an image to upload and fill in the personal information. After this, the program will automatically come up with the best matching person. The process of face recognition, also needed for face comparison, consists of three steps.

The first step is locating the face within the image. This step is often done with the help of a neural network.

Step two: certain parts of the face, such as the mouth and the nose, must be located within the face. Mathematical techniques like "Gaussian"- or "Gabor"-filters are used most often for the implementation of this step. The filtering can be done for a few orientations and scales (scale-space theory). By using these filters, essential parts of a face are relatively easy to find. During the first three weeks of this project these filtering techniques have been researched, it has shown us this is a very complex part of the face recognition system.

The final step of the process is determining the likeness of two facial images for which two often-used methods exist. The first one, a statistical method called Principle Component Analysis (PCA), correlates the luminance pattern of the image. The second technique uses a “Gaussian”- or “Gabor”-filter to process the images. The output of these filters can be correlated (possibly in combination with PCA) to get the most corresponding picture.

But since the system is only a prototype, these three steps aren't implemented. A third option to calculate the likeness of two images is used: by combining distances between the points the user clicked in the second screen, every image gets its own vector with data used to compare images. The distance between two of these vectors represents the equality of two faces.

2.4. Facial model

The input for the 'double-find' algorithm consists of data collected from user input. The example face shown below has points numbered from A to Q. The user is asked to click these points on their own photo using the example face as a reference model.

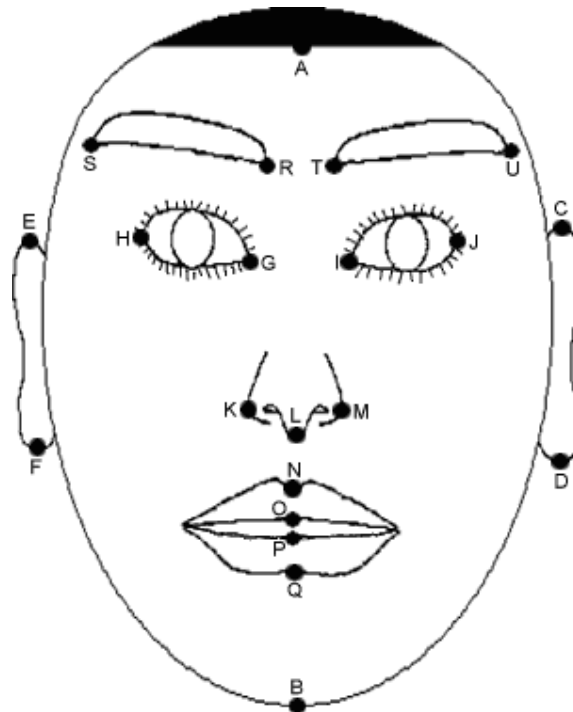


Figure 1: Example Face

15 Distances were chosen using these 21 points. Each distance has a weight-factor. The first version of the model gave every distance an equal weight. There are three factors influencing the weights. The first factor deals with the staticness of the face. A distance is said to be completely static when it doesn't change when the facial expression does change. The second factor is the amount of 'wrong-clicking' in the user input and its influence on a distance. Some points are easy to 'wrong-click'. For example: try clicking the ears of someone with long hair covering the ears. The third factor is the importance of the distance in the facial model. Some distances represent

characteristics of a face that are more important than others when finding a double. All weights lie between 0 and 2. The bigger the weight the more important the distance is for the 'double-find' calculation. The weight-model was optimised by trial-and-error. Doubles were calculated for a set of pictures with different weights. The weights for each distance described below resulted in the best doubles. All of the 15 distances and their corresponding weights will now be discussed:

Distance 1. Height of the head, without the mouth

'[A-N] + [Q-B]': It's one of the most important distances in this model, because the length of the head is characteristic for a person. Furthermore it's a completely static distance, meaning that for example the expression of the face doesn't influence the distance. If the user clicks wrong (he clicks a little above/below/left or right of where he is supposed to click) the impact on the result isn't that big. The weight of this distance is 1.7.

Distance 2. Width of the head

'[(Average of C and D) - (Average of E and F)]': Roughly said, this is the distance from the 'middle' of one ear to the 'middle' of the other ear. Although the width of the head is pretty characteristic for a person the amount of 'wrong-clicks' is big. For example: it's hard to select the right points of both ears if hair is covering them on the photo. The expression of the face doesn't have much influence on this distance. The weight for this distance is 1.0.

Distance 3. Space between the eyes

'[G-I]': Users won't make much wrong-clicks when selecting these two points. The expression of the face doesn't influence the distance much either. The weight of this distance is 1.2.

Distance 4. Average width of the eyes

'[Average of ((G-H) + (I-J))]: Different size of the eye is visually a big difference in the face. The user input is reliable. That makes the high weight. The weight of this distance is 1.5.

Distance 5. Width of the nose

'[K-M]': The facial expression doesn't have any influence on the distance. The width of the nose is an important distance in the face model. It's, for example, more important than the height of the lips. The weight of this distance is 1.3.

Distance 6. Height of the nose

'[L- (average of (G-I))]: This is one of the most important distances in the face model. The beginning point of the nose is defined at the level of the lower two eye corners. Furthermore, the height of the nose is very static. The weight of this distance is 1.7.

Distance 7. Average height of the ears

'[Average of ((C-D) + (E-F))]: The amount of 'wrong-clicks' is big (see Distance 2) which can have a major impact on this distance. The weight of this distance is 0.5.

Distance 8. Space between the eyebrows

'[R-T]': It can be difficult to tell in a person's face where the eyebrows end, which can result in bad placement of the points. The weight of this distance is 0.9.

Distance 9. Distance between the eyebrows and the nose

'[L- (average of (R-T))]: Clicking points R and T can be difficult on some photo's which results in a lot of 'wrong-clicks'. That decreases the weight for this distance. Furthermore the facial expression can have effect on the position of the eyebrows (for example the expression amazement will result in the eyebrows being placed higher in the face). The weight of this distance is 0.9.

Distance 10. Location of the ears with regard to the eyes (vertically)

'[(average of (F-D))-(average of (G-I))]: It's the same problem as mentioned before regarding the ears. The expected amount of 'wrong-clicks' is big in points F and D. Points G and I have less 'wrong-clicks'. The weight of this distance is 0.7.

Distance 11. Average height of the lips

'[Average of ((N-O) + (P-Q))]: The facial expression influences this distance. For example smiling will appear to decrease the height of the lips. Also, the amount of 'wrong-clicks' is very high because of the short distance between the lips. The reason why this distance is part of the facial model is because the heights of the lips are characteristic for a person. The weight of this distance is 0.4.

Distance 12. Average length of the eyebrows

'[Average of ((R-S) + (T-U))]: The length of the eyebrows is influenced by a change in the facial expression. The begin- and endpoint of the eyebrows are sometimes difficult to select, so the amount of wrong clicks is more than average. The average length of the eyebrows is an important distance in the face model. The weight of this distance is 1.0.

Distance 13. Distance from the corners of the eyes to the nose

'[Average of ((L-J) + (L-H))]: It's a very static distance. The point of the nose is often difficult to select, resulting in much wrong clicks. The importance of this distance in the face model is normal. The weight of this distance is 1.4.

Distance 14. Distance from the corners of the eyebrows to the nose

'[Average of ((L-S) + (L-U))]: This distance isn't very static due to the varying vertical location of the eyebrows. Their location in the face is different concerning the facial expression. This distance hasn't got a high importance in this model. The weight of this distance is 1.1.

Distance 15. Location of the mouth with regard to the eyes (vertically)

'[N- (average of (G-I))]: This distance is very static and has a normal importance in this model. The weight of this distance is 1.4.

2.5. Search algorithm

Besides the challenge of writing a semiautomatic face comparison algorithm, another big challenge lies in designing and implementing the database and the search method. Even when the database contains an enormous amount of facial images, searching for the best matching image must be done very quickly. Considering the quantity of the data, the most ideal situation would be sorting the data linearly, this way the best matching picture can be found without checking every single image in the database.

The vectors, consisting of 15 numbers, can't be sorted linearly. Sorting them would require some sort of weight mechanism with which an index could be created for every vector in the database. These weights must be chosen in such a way, that

sorting on this index would give us a person's best doubles. This is quite impossible, because of the complexity of the distance vector.

To minimise this problem, an algorithm has been designed. It can be split up into two parts: one as an initial procedure to build an average vector and sort the database, and one to search the database for the best double, which must be done on every new user.

Average vector

At first, calculate an average vector of all vectors in the database. Then, for all vectors, the distance to this average vector is being calculated, and stored in the database. Next the database is sorted on this distance, only the vectors with a distance to the average vector less than two times the deviation plus the average distance will be used to calculate the new average vector. This way, the 'nonsense' images will be left out of the calculation. This new average vector is used to recalculate the distance between the average vector and each vector in the database. This way all vectors get mapped onto a one-dimensional line. Although images that look like each other have approximately the same distance, images with approximately the same distance don't have to look like each other. This is because mapping a vector onto a one-dimensional line causes loss of information regarding the direction of the vector.

Searching

The next part of the algorithm checks for the best double without checking every single vector in the database. These are the 9 steps of this algorithm:

- 1) Calculate the distance **R** between the average vector (**v_avg**) and the current vector (**v_curr**).
- 2) Find the **x** vectors of which the distance to **v_avg** (**d**) is the closest to **R**.
- 3) Calculate for each of these vectors their distance to **v_curr** and call it **d_curr**.
- 4) Find the vector (**v_closeby**) closest to **v_curr**. Its distance **d_curr** is called **A**.
- 5) Store the distances **d** of the two vectors lying closest to and furthest from **v_avg**, respectively in **dmin** and **dmax**.
- 6) If **dmin** > (**R-A**) select all the vectors with a distance to **v_avg** between **dmin** and (**R-A**) into set **set1**. Else **set1** stays empty.
- 7) If **dmax** < (**R+A**) select all the vectors with a distance to **v_avg** between **dmax** and (**R+A**) into set **set2**. Else **set2** stays empty.
- 8) Now create a complete set **set_complete** with all **y** elements of **set1**, **set2** and the vector **v_closeby**.
- 9) Find in **set_complete** the vector with the smallest distance to **v_curr**.

This way, only (**x + y**) vectors have to be correlated with the current vector, instead of the total amount of vectors in the database.

A good value for **x** has to be found because if **x** is taken too small, the distance **A** will be very small and so a large **y**. But if **x** is chosen too large, there will be checked way too many vectors in the first place.

Trial and error will lead to a reasonable **x**.

The first step deals with calculating the distance of the current vector to the average vector, so the position in the database is known. Then select a number of vectors that are in the neighbourhood (based on the distance to the average vector) of the

current vector as described in step two. Step three: calculate for each of these vectors the distance to the current vector. This value indicates the likeness between the two corresponding faces.

The vector with the shortest distance to the current vector is the vector that corresponds to the best matching double ⁽⁴⁾. It's not known if this is the best double in the database. There may be a better match in the vectors just outside the selection. That is because two vectors with an approximately same distance to the average vector, not necessarily lie close to each other (they can even be opposite). The one thing known is the closest match in the first selection. This distance is added to the distance of the current vector to the average vector and subtracted of this value. Because the distances from the average vector to the current vector and to the best double can differ at most the distance to each other, the best double must be between these two values. After this select all vectors from the database with a distance to the average vector between these two values, except the vectors of the first selection. It's for sure that a possible better double must be in this selection, so calculate for every vector the distance to the current vector. The vector with the shortest distance must be the best double.

2.6. Database connection

MySQL^[9] is used as database server, because it's already running on the server FindMyDouble will be running on. To establish the connection between the server and the client (the FindMyDouble application) JDBC is used^[8]. JDBC is the JAVA API for executing SQL queries, which supports the following three functions:

- Establish a connection with the database.
- Send SQL statements.
- Process the results.

JDBC supports both a "two-tier" and a "three-tier" model to access the database. When the "two-tier" model is used, the JAVA-applet communicates directly with the database. To do this a JDBC driver is needed to communicate with the database management system used. The SQL statements are sent to the database and the corresponding results are sent back directly to the applet. The database may be situated on the local machine, as well as on a different machine. This model is called client/server configuration. The client is the machine of the user (the applet) and the server is the machine on which the database is running.

When the "three-tier" model is used, all SQL statements are sent to a "middle-tier", which sends it to the database. The database processes the commands and sends the results back to this "middle-tier". There they are sent back to the client.

In this application, the "two-tier" model is used with the mm.mysql driver v2.0.4^[9].

2.7. Scalability

The following chapter tries to answer questions like: "How many users can we serve on a 10mBit half-duplex connection? Is there a way to improve this?" and "How many users can be stored on the current server? Is there a way to improve this?"

Although system memory is a very important issue, estimating the impact on the system memory of for example 200 users working on the system at the same time has not been done due to a lack of time and resources.

2.7.1. Server set-up

The server the FindMyDouble system is tested on has the following set-up:

CPU: AMD Athlon 800 MHz
Memory: 384 MB SDRAM
Hard disk: Western digital 20 GB 7200 rpm
Operating system: FreeBSD 4.2
HTTP server: Apache/1.3.12 (Unix)
Database server: MySQL database server v 3.23.33

2.7.2. Data traffic

To calculate the average amount of data traffic, an average user was imitated. This means the average size of each data type that gets sent over by either the server or the client application was used for this calculation. For example, currently 200 pictures were uploaded. The total size of these pictures is 3,116,032 bytes. An average user would upload a picture of $3,116,032/200 \approx 15$ kB. The following table shows the amount of data traffic for each operation. Because an average user won't use the AdminApplet, the amount of data generated by the use of the admin applet is left out of the picture.

Data type	Amount of data (kB)
ScreenApplet	
Uploading the image	15
Downloading the JAVA classes	171
Downloading the image	15
Downloading vectors set 1 (if the database would contain 100.000 entries)	30
Downloading vectors set 2 (if the database would contain 100.000 entries) ¹	25
Inserting personal information	58
Downloading double information	58
Subtotal	372
PersonApplet	
Downloading classes	148
Downloading personal information (1*58)	58
Downloading double information (10*58)	580
Subtotal	786
Total	1,158

Table 2: Scalability, current situation

According to this calculation it takes about 1.1 MB per person to use the ScreenApplet and the PersonApplet. This is a large amount of data traffic for a single run but since it is spread over an average of 4 minutes per run², the average amount of data traffic per run is only 4.8 kB/s.

If our server connection was 10 Mbit/s half-duplex we could theoretically serve $1,000/4.8 \approx 210$ users synchronically.

¹ Because of the small size of our database it was not possible to get an accurate estimation. We think however that the number of vectors in this set will keep on growing logarithmic.

² It would take an experienced user about 4 minutes to complete a single system run. This was tested by 3 experienced users using 3 different connections.

If all the images would be saved as JPEG's on the server and not in the MySQL database (see paragraph 5.1: For online use), the table would look like this:

Data type	Amount of data (kB)
ScreenApplet	
Uploading the image	15
Downloading the JAVA classes	170
Downloading the image	15
Downloading vectors set 1 (if the database would contain 100.000 entries)	30
Downloading vectors set 2 (if the database would contain 100.000 entries)	25
Inserting personal information	0
Downloading double information	15
Subtotal	270
PersonApplet	
Downloading classes	148
Downloading personal information (1*15)	15
Downloading double information (10*15)	150
Subtotal	313
Total	583

Table 3: Scalability, optimisation 1

Now the traffic is less than 0.6 MB, a reduction of almost 45%. The average amount of data traffic per run would be 2.4kB/s. If the server connection was 10 Mbit/s half-duplex the server could theoretically serve $1,000/2.4 \approx 420$ users synchronically.

One last improvement one might want to look at is compressing the class files. The JAR format was used to test this. If the class files would be jarred it would reduce the size of the classes needed for ScreenApplet to 92kB and the size of the classes needed for PersonApplet to 80kB thus reducing the overall size to 437kB. The average amount of data traffic per run would be 1.8kB/s. If the server connection was 10 Mbit/s half-duplex the server could theoretically serve $1,000/1.8 \approx 560$ users synchronically.

2.7.3. Disk space requirements

The current database structure consists of 4 tables: person, data, dd and const. The person and data tables contain 201 entries, the dd table contains 67 entries and the const³ table contains 2 entries. The average size of a row in the person table is 58kB, in the data table is 0.14kB and in dd is 0.14kB. MySQL limits the maximum size of the person table to 4GB⁴, the maximum size of the data table to 332GB and the maximum size of the dd table to 348GB. So on a 20GB hard disk the maximum size of the double database would be limited by the maximum size of the data table (4GB). The maximum number of people stored on this system would be $4GB/58kB \approx 72.300$.

³ The const table will not be used in this calculation since its size is static and very small. Therefore it will not influence the result of this calculation.

⁴ Tables containing BLOB types may not be larger than 4GB in MySQL databases.^[9]

If all the images would be saved as JPEG's on the server and not in the MySQL database (see paragraph 5.1: For online use), $15.00\text{kB} + 0.03\text{kB} + 0.14\text{kB} + 0.14\text{kB} = 15.3\text{kB}^5$ would be needed per person and the maximum size of the person table would be 332GB. On a 20GB hard disk about 1.4 million people could be stored.

2.8. Security

In this chapter the weak links in the current security set-up are identified and possible solutions to these system threats are found.

2.8.1. Current set-up

The weakest link in the current set-up is the database connection. The current set-up has 2 different user accounts, the fmdadmin and the fmdclient account.

The fmdadmin account allows one to select, insert, modify and delete everything in the FindMyDouble database. This account is used to administrate the database. The fmdclient account allows one to select and insert everything in the FindMyDouble database. The fmdclient cannot modify nor delete the tables.

2.8.2. Threats to the system

The AdminApplet uses the fmdadmin account to administrate the database but since login name and password have to be supplied by the user, this account poses no real threat to the security of the system. AdminApplet itself poses a slight threat to the system since it contains a hard coded link to update.cgi (the script that recalculates the average vector and the distances). The execution of this script asks for a lot of system resources since it needs to readjust the vector distance for every single double in the database. A malicious user could decompile the AdminApplet, fetch the link and bring down the system. Another obvious problem area is that both the ScreenApplet and PersonApplet make use of the fmdclient account. This requires that the login information (including password) be hard coded into the source. The offshoot of this is that by obtaining the source code and/or decompiling the applet, one may easily obtain access to the fmdclient account. Although it is not possible to modify the table entries, one would not want a user to be able to fill the database tables with false entries. Another problem occurs if the auto update recommendation on page 27 in paragraph 5.1 would be implemented. Since the fmdclient account is not able to modify a table entry, the PersonApplet would not be able to update a user's double-info. Using the fmdadmin account for updating this data is NOT an option.

2.8.3. Recommended set-up

Solutions to the problems found in 2.8.2. Threats to the system are mentioned here.

2.8.3.1. Update.cgi

The solution to this problem is simple. Modify update.cgi so that one has to supply the login name and password used in AdminApplet in order for it to execute.

⁵ The size of the index tables was not taken into consideration.

2.8.3.2. fmdclient account

The solution to this problem proves to be a bit more tedious. The first step in solving this problem is removing the insert privilege from the fmdclient account. This way a malicious user cannot fill the database with false entries. The second step is making a database “account” for each and every user. This is not a real MySQL database account but a mere entry in a table. The entry consists of a user id (automatically created by the database) and a randomly created password. The password is created by upload.cgi (the script that handles the uploading of the person’s personal info and the person’s facial picture). Upload.cgi passes the person’s id and corresponding password on to ScreenApplet. The last step is the creation of a new script called db.cgi that allows users (ScreenApplet and PersonApplet) to insert, modify and delete their own entries in the database if the script is provided with the user’s id and password.

Chapter 3. Design

Before the programming, a decent design must be made. The design of the program is explained in the first paragraph of this chapter. After that the database design is discussed.

3.1. Global design

In this paragraph the design of all three programs (Screen, Person and Admin) of FindMyDouble is shown.

3.1.1. Diagram

This is globally the cohesion of the complete work. This diagram is only a global description of the real programs, not every loose box is a different (program-) class. Some classes cover different boxes, and some boxes are in different classes.

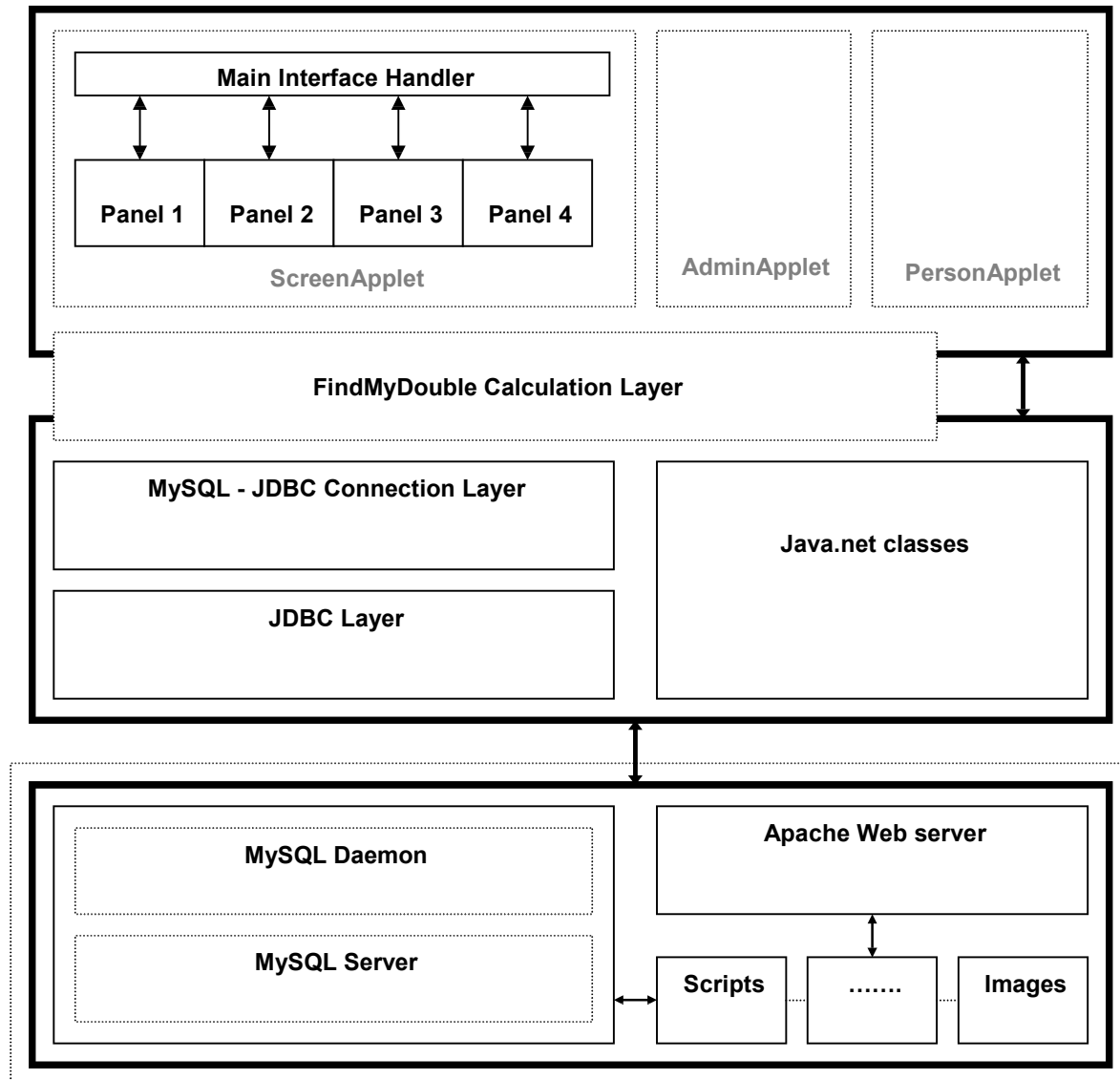


Figure 2: Global Design Diagram

GCH
UOE
SMUR

3.1.1.1. Functional description

The inner workings of the applets can be described by the use of this model. For the realisation of this application a 3 layered model was chosen. The system consists of the following layers:

- 1) the GUI (Graphical User Interface) layer
- 2) the COM (Communication) layer
- 3) the SERVER layer

The calculation mid-layer is both part of the GUI and COM layer. It can be seen as the glue between both layers.

3.1.1.2. GUI Layer

The GUI Layer is the only layer that's completely different for each applet.

- ScreenApplet
The GUI layer consists of the Main Interface Handler and 4 panels. The Main Interface Handler handles the traffic between the panels and the traffic between the panels and the calculation mid-layer. Each panel represents a step in the process and each has its own screen components (buttons, canvases, textfields, etc.). The reason the use of these panels was chosen was to make the source code maintainable.
The main task of the GUI layer is guiding the user through the process. It should provide help at critical moments and thereby improve the quality of user input. At the time of this writing four steps can be differentiated:
 - Step 1, the user has to select the face within the uploaded image
 - Step 2, the user has to click the facial points displayed in the example picture.
 - Step 3, the user input is processed and passed on to the calculation mid-layer. The results are calculated.
 - Step 4, the results are displayed.
- AdminApplet
The GUI layer of the AdminApplet is completely in one class. There are a few different tasks what the AdminApplet must do:
 - Showing a person corresponding to the ID chosen
 - Showing the double of this person
 - Updating userinfo like names, emailaddresses, etc.
 - Recalculating a persons doubles of the person
 - Only check for better doubles
 - Recheck the complete database with persons
 - Deleting a person, and all references to this person (if this person is the double of another one, this doublelist must be updated)
 - Checking and changing the weights of the 15 distances in our distance-model (see Paragraph 2.4: Facial model)All these tasks are done by the AdminApplet with the help of the calculation layer for the recalaculation of the doublelist and the updating of the distance-weights. The connection to the 'MySQL - JDBC Connection Layer' is needed for almost every action. The java.net classes aren't used by this program.
- PersonApplet
This PersonApplet is used to show the user all information filled in, and the doubles. Also it will be used to let the users adjust their personal information, or just remove themselves from the database completely. The only userinput

now is the browsing thru the doubles, so only the database connection is needed in this GUI layer.

3.1.1.3. Calculation Mid-Layer

The calculation mid-layer can be seen as the glue between the GUI and the COM layer or rather between the GUI and the MySQL – JDBC Connection layer since some traffic may come to pass between the GUI layer and the java.net classes without the calculation layer interfering. The calculation layer uses the user input and the data in the database to calculate the best set of doubles. The MySQL – JDBC Connection layer is its interface to the database.

3.1.1.4. COM Layer

The COM layer connects the GUI layer with the SERVER layer. Its task is to provide an interface to the MySQL database and to the Apache webserver. In order to accomplish this feat, it uses JDBC for the database connection and the java.net classes for the webserver connection. The database connection is utilised by the GUI layer to accomplish both the retrieval and storage of data and images associated with the doubles. The webserver connection is used by the GUI layer for the retrieval of images, such as the uploaded facial image and the example picture used in panel 2 of the GUI layer. The GUI layer also uses it to execute certain perl scripts.

3.1.1.5. SERVER Layer

The server is used for the storage of data and images associated with the doubles. The MySQL database can be accessed by both the JDBC layer in the COM layer and the serverside scripts. The scripts are used to clean up the mess left by previous application runs and to recalculate and update certain values stored in the MySQL tables.

3.2. Database design

The database consists of 4 tables: *person*, *data*, *dd* en *const*.

3.2.1. **person table**

In the table *person* all persons with their photo are stored. All input is stored in this table, and also the zipped photo. That leads to the following columns:

- **id** (integer): same value as the corresponding data.id
- **name** (varchar(100)): person's name
- **country** (varchar(100)): person's country
- **email** (varchar(100)): person's e-mail address
- **dob** (date): person's date of birth
- **gender** (char(1)): person's gender ('m'/'f')
- **notify** (char(1)): Whether the user wants to be notified if there is a better match ('y'/'n')
- **zipphoto** (mediumblob): The zipped photo

3.2.2. **data table**

In the table *data* the distances in the face and the distance to the average vector are stored. That leads to the following columns:

- **id** (auto incremental integer)

- v1 (float): first distance in the face
- ...
- v15 (float): last distance in the face
- d (float): distance between the vector (v1, ..., v15) and the average vector
- d_id (int): id of the best double found
- sdf (float): distance to the best double
- lastidchecked (int): last id checked to find the best double

3.2.3. dd table

In the table *dd* the double data, the ids of and the distances to the 10 next best doubles are stored. The best double, and the distance to it, is stored into *data*. The table *dd* consists of the following columns:

- **id** (integer): same value as the corresponding data.id
- did1 (integer): id of the first double
- d1 (float): distance from the vector to double did1
- ...
- did10 (integer): id of the last double
- d10 (float): distance from the vector to double did10

3.2.4. const table

In the table *const* consists of the constant values, like the average vector. That leads to the following columns:

- **id** (auto incremental integer)
- name (varchar(100)): name of the constant
- value (text): value of the constant

Chapter 4. Programming

This chapter describes all the programs and scripts that are used in the FindMyDouble project. First the three graphical user interfaces (GUIs) are described. In paragraph **Fout! Verwijzingsbron niet gevonden.** the UML diagrams for these three GUI classes can be found. These GUIs are the graphical shells built around the core classes. The core classes are described in paragraph **Fout! Verwijzingsbron niet gevonden.** with their corresponding UML diagrams. The distinction between the GUI and core classes was made so it's easy to use the systems create another GUI for (one of) the programs without having to change the core classes. All the calculation should be done in the core classes. The task of the GUI classes is dealing with user input and interpreting the output of the core classes. The last paragraph of this chapter contains information about the various scripts and server side programs used in the FindMyDouble project. The source of all classes comes along as an appendix with this report.

4.1. The three applications

The functionality of the three applications, ScreenApplet, PersonApplet and AdminApplet is described here. Also the required user-input and the layout is discussed.

4.1.1. ScreenApplet

The ScreenApplet consists of the main Screen class and four ScreenPanel classes. These five classes form the Graphical User Interface (GUI). This paragraph describes the steps a user has to go through in order to participate in the FindMyDouble project.

4.1.1.1. Part 1: Enter personal information and upload an image

The first page shown to the user when visiting the FindMyDouble web site contains a form. This form requires some personal information regarding the user. Figure 3: ScreenApplet Part 1 shows the current layout.

You can upload your image here

Images are accepted in all standard image formats (JPEG, GIF, PNG, TIFF etc)

Please enter your name:	<input type="text" value="Al Pacino"/>
Please enter your email address:	<input type="text" value="al_pacino@jail.com"/>
Please enter your date of birth:	<input type="text" value="4"/> <input type="text" value="September"/> <input type="text" value="1951"/>
Please enter the country you're from:	<input type="text" value="United States"/>
Please select your gender:	<input checked="" type="radio"/> Male <input type="radio"/> Female
Do you wish to be notified if there is a better match?	<input type="radio"/> Yes <input checked="" type="radio"/> No
Choose a photo to upload	<input type="text" value="C:\alpacino.jpg"/> <input type="button" value="Browse"/>
<input type="button" value="Upload"/>	

Figure 3: ScreenApplet Part 1

The user fills in the fields and chooses a picture to upload. When the upload button is pressed the upload script generates a random key consisting of 20 letters and uses

this key to create a name for the image. An example would be: “ANBFOIDHTBQDUPNMIWCQ-orig.jpeg”. The tasks of the upload script are further described in paragraph 4.2.1. The image is uploaded to the server and is given the name generated by the script. The reason for uploading the image to the server has to do with the security policy of Java. Java is only allowed to display files that come from the same location the applet is from. After the image has been uploaded the parameters (filled in the form) are passed to the applet and the applet is loaded. When the ScreenApplet is executed the image of the user is displayed on the first screen of the ScreenApplet.

4.1.1.2. Part 2: Selection of the head in the image

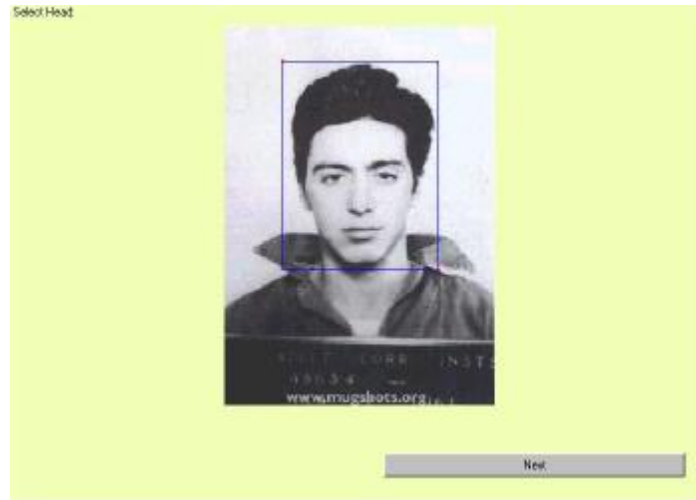


Figure 4: ScreenApplet Part 2

In this screen the user selects the head from the image by dragging a selection box over the image. When the head is selected, the selection can be changed by dragging one of the two points to a new location or by making a new selection. When the user is satisfied with the selection made, the 'Next' button will show the next screen containing the persons enlarged head.

What actually happens when the Applet is loaded is that applet draws the first panel on the rectangular area above the 'Next'-button. The picture is converted to an array of pixels and is drawn onto a canvas. It is resized if necessary in order to fit on the panel. When the user makes a selection, a new selection box is drawn with every mouse move. This means that some pixels in the array are updated with pixels representing part of the rectangle and the whole array is redrawn on the canvas. The begin- and endpoint of the selection are saved with every move of the mouse. Clicking the 'Next' button hides the first panel and displays the second panel.

4.1.1.3. Part 3: Selecting the Points

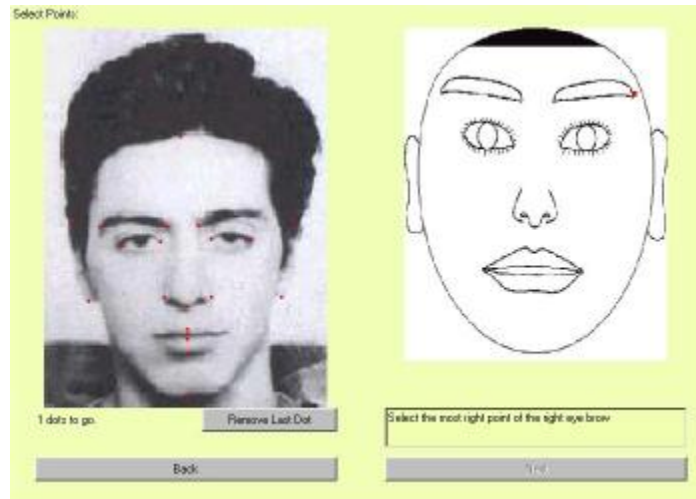


Figure 5: ScreenApplet Part 3

In this screen the user is asked to place points on the image he selected before. On the right of the screen there's an example face. If the user isn't satisfied with the selection of the photo placed on the left, he can press the 'Back' button and make a new selection in the first screen. In this second screen, a red dot appears on the example face with text in the label describing exactly where the user has to click. The user clicks on the picture at the same place as the red dot on the example face. The text in the label gives some explanation about the location of this point and it is to help the user clicking the right spots. After the user has clicked the point, another point appears on the example face with other text in the label describing where to click. This way the user selects 21 points (the amount of points can be changed in the software if there's need to). There's a counter below the user's image indicating how many dots the user has yet to place. The user also has the possibility to remove the last dot set on the image or drag dots when they haven't been placed exactly where the user wanted them placed. When the user is satisfied with the points set, clicking the 'Next' button will take him to the next screen.

On a lower level the location of all the dots are saved in an array. Removing a dot will result in removing one element of the array. Dragging a dot to a new location results in changing the co-ordinates of a dot in the array. Of course the dot counter is changed when a dot is removed or added. Only when all dots are set the 'Next' button is enabled.

4.1.1.4. Part 4: Finding your Double

The only thing asked of the user in this part of the applet is to have patience. To satisfy his hunger for information a text area is filled with status information.



Figure 6: ScreenApplet Part 4

What's actually happening is that the applet passes the distances, calculated from the coordinates of the dots, to the core of the application and waits till it retrieves the location where to find the double image. The next screen is loaded without any input from the user. The user is still feeling overwhelmed by the impressive amount of status information when he suddenly realizes the last screen of ScreenApplet is being displayed.

4.1.1.5. Part 5: The Double

On the right of the screen you see the picture of the user (in this case Al Pacino after being arrested) and on the right side you see his double. Both with the personal information they entered displayed below their photos.

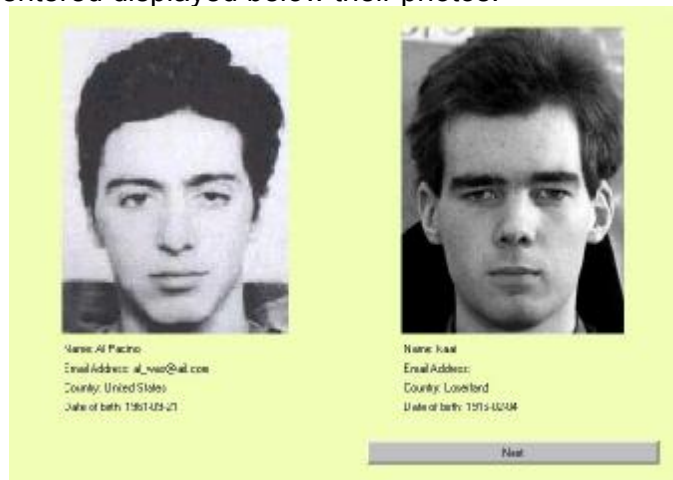


Figure 7: ScreenApplet Part 5

After pressing the 'Next' button you'll be forwarded to a web page with the PersonApplet, which is discussed in the next paragraph.

4.1.2. **PersonApplet**

The PersonApplet displays the image and the personal information of both a user and a double on the screen. Each time the applet is loaded it checks for new doubles

of the person. At the lower-right corner of the screen there's a button used to display the person's next best double.

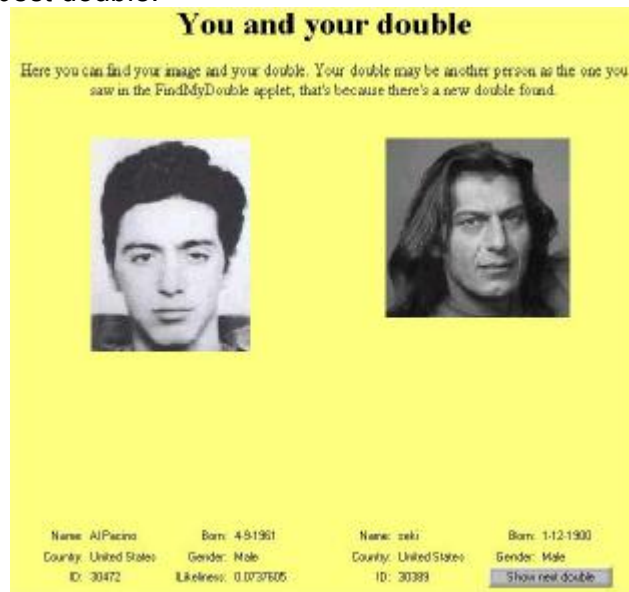


Figure 8: PersonApplet

The PersonApplet has two main functions. The first function is to show the other doubles to the user. Clicking the 'Show next double' button does this. The user with the next best likeliness is displayed instead of the current double. This option can be repeated up to 10 times (this number can be changed). The reason there is a maximum is because the likeliness is getting smaller for every double. It also prevents users from viewing very much doubles, which would cost a lot of bandwidth. When the user presses the 'Next' button on the last screen of ScreenApplet, described in the previous paragraph, the PersonApplet is displayed with (most likely) the same double as in the last screen of ScreenApplet.

The second function of PersonApplet is updating the double information, when the user visits the page. The PersonApplet can be loaded for every person by passing along the ID of a user to the script. This way every user can return to their page regularly, checking for new double information or, as described above, viewing other doubles.

4.1.3. AdminApplet

In the AdminApplet all administrator tasks can be performed. It's possible to view, update and remove users, one can view and update the weights for the distance-model and there is a possibility to recalculate double information of any user.

4.1.3.1. Logging in

To use the AdminApplet, you have to login into the system. This must be done in the first screen, at which two text fields for the username and password and a button to login.

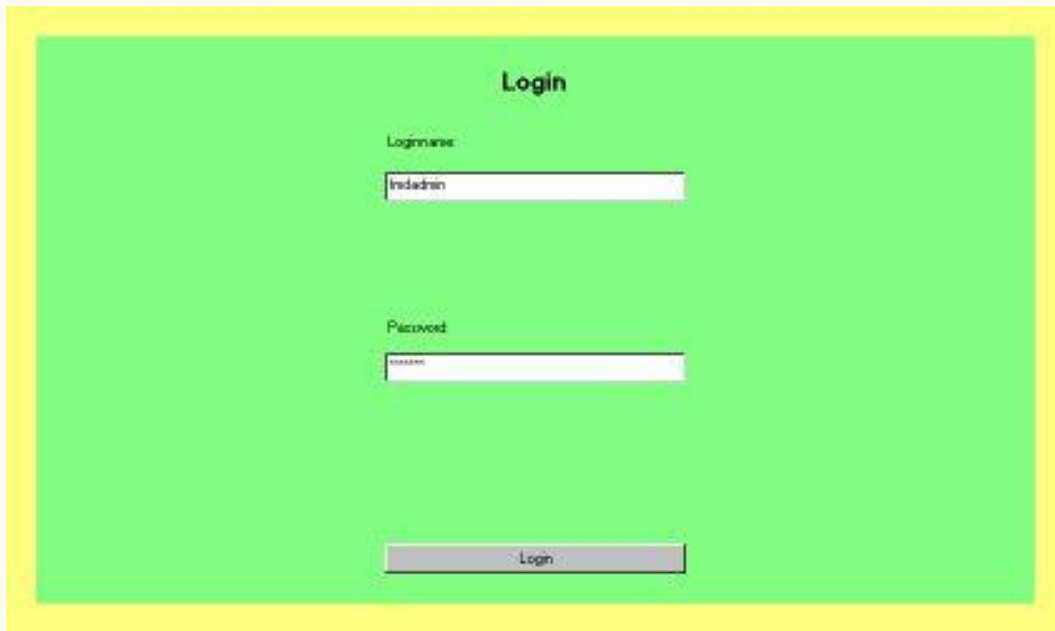


Figure 9: AdminApplet: Login

When the 'Login' button is pressed, the system is going to check the supplied credentials and it will download the list of ID numbers for the next screen. If the username and password combination is not valid, a warning will be shown.

4.1.3.2. Viewing and updating users and recalculating double information

After the correct login, the following screen will be shown, which contains possibilities to update user information, recalculate double information and show other users.

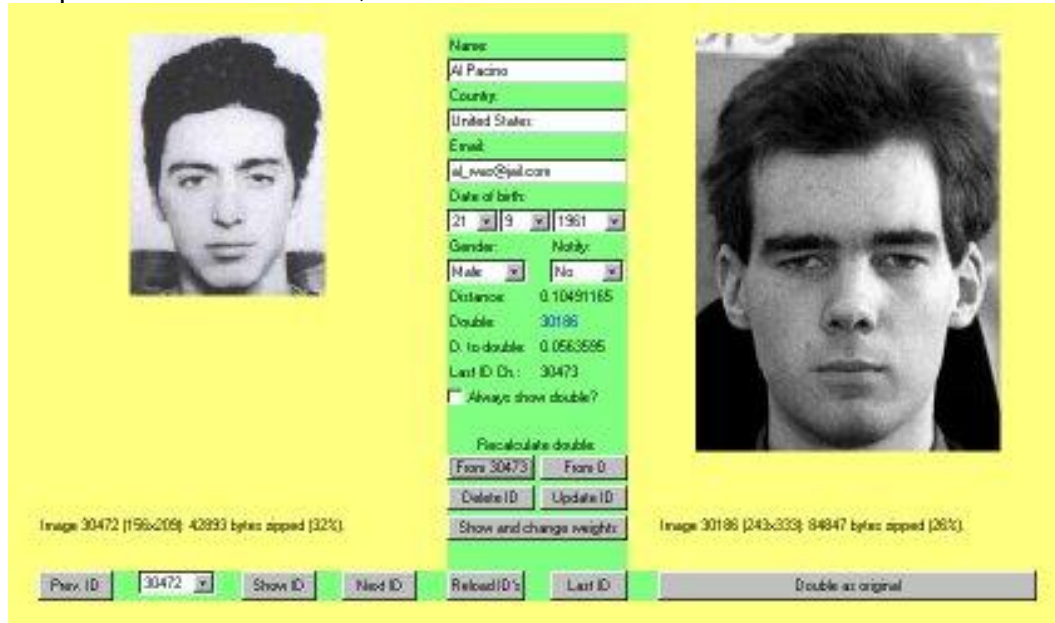


Figure 10: AdminApplet: View

On the left the user is shown who's ID is chosen in the choice below. The image on the right won't be there directly. It will only be shown when the blue label with the ID of the double is clicked, or if the checkbox 'Always show double' is checked. In the

middle there is a small panel with the information of the user and five buttons. When one clicks the 'Update ID' button, all information in the editable fields will be uploaded to the database, overwriting the information there. This way, all user information can be changed in one update action. Some of the information isn't editable, because it's the result of the double calculation. It can only be updated by pressing the recalculation buttons. The left button is to find some possible new doubles in the database since the last check (the button's label 'From 30473' means all IDs until 30473 are checked for doubles). The right button will destroy all double information and rebuild it from all persons in the database.

On the bottom there are another 6 buttons and one choice. Pressing the first button ('Prev. ID') will show the previous ID in the database. The choice contains the IDs in the database, and will start with 'Prev' (to see the previous up to 100 IDs) and ends with 'Next' (to see the next up to 100 IDs). When selecting an ID, clicking the 'Show ID' button (directly right to the choice) will show the image and the information on the screen. It also shows the image of the double when the 'Always show double' checkbox is checked. The 'Next ID' button shows the information and image corresponding to the next ID in the database. Pressing the 'Last ID' button will reshown the person that was visible before the currently visible person. The large button on the right is the 'Double as original' and shows the currently shown double as the original person, with all the information and even this double's double (if the checkbox 'Always show double' is checked).

4.1.3.3. Deleting users

By clicking in the previous screen on the 'Delete ID' button, this screen will be shown. It consists of the previous screen, but with a small window instead of the image that will be deleted.

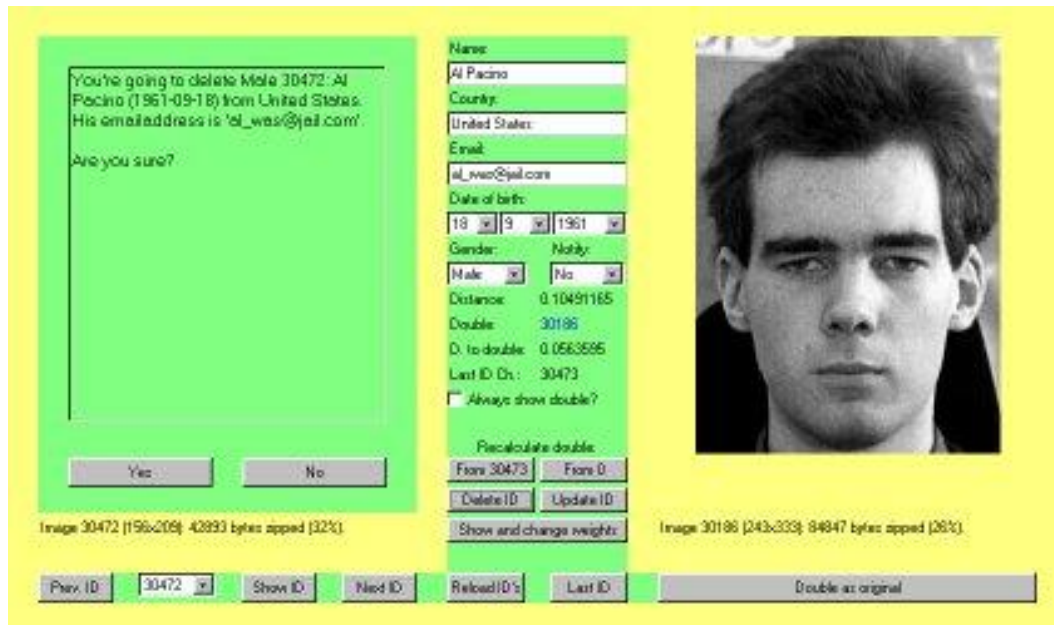


Figure 11: AdminApplet: Deleting users

There is a field that contains a question whether or not this person must be deleted, and two buttons. Pressing the 'Yes' button deletes the message and the 'No' button will remove small window and returns to the previous screen.

4.1.3.4. Weight administration

On this screen, the fifteen weights of the fifteen distances will be visible. Also two buttons below these text fields are placed.

Weight	Value
w1:	1.7
w2:	1
w3:	1.2
w4:	1.5
w5:	1.3
w6:	1.7
w7:	0.5
w8:	0.9
w9:	0.9
w10:	0.7
w11:	0.4
w12:	1
w13:	1.4
w14:	1.1
w15:	1.4

Buttons: Update, Back

Navigation: Prev. ID: 30472, Show ID, Next ID, Reload ID's, Last ID, Double as original

Figure 12: AdminApplet: Weight administration

These fifteen fields are uploaded to the database if the 'Update' button is pressed. This means there is no need for a press on the 'Update' button for every distance that must be changed. If the 'Back' button is pressed, this screen will be removed and the normal screen with the image and the rest of the information of the user is shown.

4.2. Scripts

The scripts described in this paragraph are upload.cgi and update.cgi.

4.2.1. **Upload.cgi**

This script has 2 major tasks:

- Accepting personal information concerning the user.
- Generating a random filename.

The script checks if the user information seems correct by using perl's regular expression engine. If the information seems correct it is passed on to the applet along with the randomly chosen filename. The filename is the name the image is stored with.

4.2.2. **Update.cgi**

First, this script recalculates the average vector using the vectors in the database. Next, it recalculates the distance of each vector from the average vector. It does this by using weights to emphasize the importance of each normalized vector distance.

Chapter 5. Recommendations

Due to the limited time we had for the FindMyDouble project, we weren't able to implement all the features we wished to implement. Also, the more functionality you add, the more functionality you want, so it will never be 'finished'. Here are our recommendations for a better working system that requires less user-input and adds more functionality. The first paragraph describes the points that must be implemented before this can be used online. Paragraph 5.2 contains the points that lead to a much more automatic application, less sensitive to errors.

5.1. For online use

- A mail script that sends an email containing the URL of the user's personal page. This page shows the data entered, and the pictures of the user and the double. When a user visits this page, the applet will update the user's double-info. This is done by searching for better doubles in the people who are new in the database since the last visit.
- Browse thru the doubles in the admin applet.
- Allow users to update their personal information in the personApplet. For example if a user has a new e-mail address.
- Allow users to remove themselves from the database.
- Show tool tips when the mouse pointer is held over a Dot. The tool tips show the same information about the Dot as the information placed in the label.
- A script that's able to crop uploaded pictures and save them as JPEG on the server. This way we are saving disk space and bandwidth. The first saving is because JPEG images are smaller (take less disk space) than the zipped image we use now. The second saving is because the client doesn't have to upload the cropped image since the image is cropped on the server by this script.
- Implement a mess cleaner. This script will be able to delete all the uploaded images, temporary stored on the server. It will run a few times a day depending on the number of uploaded images and leave all the images not older than 10 to 15 minutes. The users those images belong to may still be in progress. The mess cleaner will become partly obsolete when the script described in the previous point will be implemented.

5.2. For a more automatic application

- Automate the selection of the head from the photo in the first screen of the applet to reduce user input. Our idea would be to use a Gabor function on the photo. This will enhance the contours of the face in the image.
- Automatically identify the points from the facial model in a photo. This'll be the most difficult change to implement, especially if you expect the results to be the same or better then the current results with user input. Applying filter techniques and a neural network that can train itself are some of the ideas worth looking at when automating the selection of points. The ideal situation would be when all the points are selected automatically; a situation in which some points are automatically selected and the user selects the rest of the points is more realistic.
- Use constraints to control user input. For example: some points can only be placed in some areas of the picture or must be placed above or below another point to make a point valid. Erroneous points must result in a warning.

- Use a neural network to manage the weight model. The user can select the best double in his list of doubles. This way the neural network is trained for better weight-factors.

Chapter 6. Conclusion

The primary goal of this project, designing and implementing a system able to find someone's double has been achieved. The current prototype offers the possibility to upload a photo and find the corresponding double with some help from the user. Extra features like finding multiple doubles and administration of the system have been implemented. Other features that have not been implemented include implementation of a face-localisation algorithm and a self-adjusting weight model.

In a project this big it is not easy to estimate the time needed to implement the separate system components. Unforeseen problems, like language restrictions and compiler bugs, can become major bottlenecks. Also Java turned out to be not as 'platform independent' as Sun declares.

A decent design is a must when building a system this large. Modifying a design afterwards, because it does not exactly describe the functionality needed, may take up hours, days or even weeks of work. Therefore it is important to leave room for modifications in the design. When designing a system, make sure to define the system components in such a way that the dependency between them is as small as possible. By doing this you make sure that a change of source code in one component does not affect every single other system component. Flexibility is increased allowing easy addition and removal of parts of the system.

Seemingly trivial components, like the Gabor filter, turned out to be a lot more complicated than was thought at first. This kind of problems can be foreseen by a thorough analysis of the problem at hand. The analysis enables one to estimate the time needed to build each system component.

Enabling three persons to work on the same source code proves to be a tedious task. We lost a lot of time because work done by one was overwritten by another. Systems like Revision Control System (RCS) and Source Code Control System (SCCS) may help in managing the source code of a large project.

Looking back at four months of hard work, frustration and eureka's we are proud of the prototype we implemented. A lot of joy was found in the fact that the people who tested the system enjoyed working with it and were amazed by the results they got. We hope you enjoyed the show and will try the system for yourself.

Chapter 7. Literature

- [1] Peter J.B. Hancock, Vicki Bruce, Mike A. Burton. *A comparison of two computer-based face identification systems with human perceptions of faces (posted in Vision Research 38, pp 2277-2288)*. 1998.
- [2] L.C. Jain, U. Halici, I. Hayashi, S.B. Lee. *Intelligent Biometric Techniques in Fingerprint and Face Recognition (CRC Press International Series on Computational Intelligence)*. June 1999.
- [3] M. Pantic, L.J.M. Rothkrantz. *Expert system for automatic analysis of facial expressions (posted in Image and Vision Computing 18, pp. 881-905)*. 2000.
- [4] R. Mehrata, K.R. Namuduri, N.Ranganathan. *Gabor filter-based edge detection (posted in Pattern Recognition Vol. 25, No. 12, pp. 1479-1494)*. 1992.
- [5] R. Panda, B.N. Chatterji. *Gabor Function: An Efficient Tool for Digital Image Processing (posted in IETE Technical Review Vol. 13, Nos. 4&5, July-October 1996, pp 225-231)*.
- [6] Larry Wall, Tom Christiansen, Randal L. Schwartz with Stephen Potter. *Programming Perl, Second Edition*. O'Reilly & Associates, September 1996.
- [7] David Flanagan, Mark Grand, Pat Niemeyer, Josh Peck, John Zukowski. *Java Reference Library on the web*. O'Reilly & Associates. 1996-1997.
- [8] *JDBC™ Guide: Getting Started:*
java.sun.com/products/jdk/1.1/docs/guide/jdbc/getstart/introTOC.doc.html
- [9] David Axmark e.a. *MySQL Reference Manual: <http://www.mysql.com/doc/>*. 1995-2001.
- [10] Mark Matthews. *mm.mysql-2.0.4 – documentation*.
- [11] Rob Pooley, Perdita Stevens. *Using UML*. Harlow, Essex: Addison Wesley Longman Limited, 1999.