

DejaVista

Alper Çuğun

Leon van der Ree
Oliver Verver

Melle Sieswerda

26th September 2004

Abstract

Finding a required image from a library of clip-art is usually time-consuming and frustrating. One proposed solution is to have the user sketch his required image and have the system return matching images from the library.

This approach has been implemented with some success in the application BajaVista. It uses topological and geometric matching from the query image onto the library (or database) to return a set of candidate images.

In this paper we will discuss the assignment of improving BajaVista. The requirements posed to us and the research that was employed. Our final result, *dejaVista*, was built incorporating existing components in a new design. This design facilitates extensibility and the use of modern and open technologies. The existing classification and retrieval algorithms have also been improved in several places.

Foreword

This document was written during the course IN3700, *Bachelor Project* of Delft University of Technology. The objectives of this course are for students to get acquainted with working for a company by doing an internship. In this case the internship was followed at INESC, Lisbon, Portugal. INESC is a privately held non-profit organization. It is dedicated to scientific research, involving computer science and telecommunications.

Acknowledgements

We would like to thank our direct contacts Manuel João da Fonseca and Alfredo Ferreira for their help and guidance during the course of this project. We would also like to thank Joaquim A Jorge and Rafael Bidarra for arranging this project. Further we would like to thank everyone on the sixth floor of INESC for their help in various affairs and for making our stay pleasant and comfortable.

Thanks to Mr. Sepers, Mr. de Vries and other people at Delft University of Technology for making it possible to do a project such as this one abroad. Their guidance and their support in the preparations and during the project were invaluable.

Many thanks to the community of dedicated Open Source programmers for producing quality software and ensuring the existence of open standards for us to work with. Special thanks to:

- Apache Software Foundation ¹
- IBM Eclipse Foundation ²
- Free Software Foundation ³

This document was generated by L^AT_EX 2_ε

¹<http://www.apache.org/>

²<http://www.eclipse.org/>

³<http://www.gnu.org/>

Contents

1	Introduction	7
1.1	Sketch Based Retrieval	7
1.2	Previous work	9
1.2.1	CALI	9
1.2.2	Sbr-core	9
1.2.3	BajaVista	10
2	Assignment	11
2.1	Client/Server Architecture	11
2.1.1	Client / GUI	11
2.1.2	Server	12
2.2	Improving simplification	12
2.3	Improving SBR	13
2.4	Planning	13
3	System Design	14
3.1	Client	14
3.2	Server	16
3.3	SBR	16
3.4	Sequences	17
3.4.1	Query	17
3.4.2	Classification	18
4	Research	20
4.1	Information Management	20
4.1.1	MoinMoin	20
4.1.2	Subversion	21
4.2	Programming Languages	21
4.2.1	Java	21
4.2.2	C++	22
4.3	Communication	22
4.3.1	RMI	22
4.3.2	SOAP	23
4.3.3	Low level	23
4.3.4	XML-RPC	23
4.4	Image Processing	24
4.4.1	WMF	24
4.4.2	SVG	25

4.4.3	PNG	25
4.4.4	Drawing	26
5	Implementation	27
5.1	Client	27
5.1.1	Java Web Start	27
5.1.2	GUI	27
5.1.3	Communication	28
5.1.4	Gesture support	28
5.1.5	WMF preview	29
5.1.6	Selection	30
5.2	Server	30
5.2.1	Communication	30
5.3	SBR	31
5.3.1	Adjacency weights	31
5.3.2	Geometry hint	31
5.4	DrawingLib	33
5.4.1	Image Conversion	33
5.4.2	Simplification	33
5.5	Database	37
5.5.1	Database layout	38
5.5.2	DbBuilder	38
6	Evaluation	39
6.1	Testing	39
6.2	Conclusions and Future Recommendations	42
6.2.1	Implementation	42
6.2.2	Methodology	43
6.3	Scalability and Extensibility	43
6.3.1	Scaling	43
6.3.2	Extensibility	44
6.4	Process	44
6.4.1	Design	44
6.4.2	Organization	45
6.4.3	Language	45
A	Glossary	47
B	Geometry features	48
C	Images used for testing	50

List of Figures

1.1	Computation of the topology descriptor	8
2.1	Left: prior to simplification, Right: simplified image by Baja-Vista	13
3.1	Diagram of the original BajaVista application	15
3.2	Diagram of the System Architecture	15
3.3	Class Diagram of the Server	16
3.4	Class Diagram of the Client Application	17
3.5	Sequence Diagram of a Query	18
3.6	Sequence Diagram of the Building of the Database	19
5.1	Java Web Start Client	28
5.2	Using the adjacency weight to differentiate between two possibilities.	32
5.3	Graphical representation of the adjacency weights in the graph and matrix.	32
5.4	The geometry information is included in the inclusion link	32
5.5	Left: a wire model of the original image, Right: a wire model of the image after the RemoveSurroundingPrimitives method	34
5.6	Left: the original image with gradient, Centre: a wire model of the original image, Right: a wire model of the image after the ColorConcatPrimitives method	36
5.7	Left: a wire model of the original image, Right: a wire model of the image after the RemoveSmallPrimitives methode	36
5.8	Optimization of a primitive with the Douglas-Peucker algorithm	37
6.1	The Recall results of the tests	41
6.2	The Precision results of the tests	41
C.1	Images used for automatic testing	50

List of Tables

5.1 XML-RPC interface to the server	31
B.1 The 11 geometry features of CALI	49

Chapter 1

Introduction

With the rise of computers as instruments for creating text documents and presentations, large collections of Clip-Art have come into existence. Clip-Art is usually defined as pictures provided in a library for easy inclusion in publications.

These libraries are indexed by categories and users find the images they want by traversing these categories. As these collections, which are usually indexed by hand, generally consist of hundreds or even thousands of images, searching and managing them can be a lot of work.

Sketch Based Retrieval offers a solution to time consuming searches. Instead of browsing categories, the user draws a sketch and uses this sketch to find similar drawings in the Clip-Art collection. This method is first described in [Fons02] and later extended in [Fons04]. Our work is an extension of this method and uses software that was written by the authors of these papers.

In this chapter we will introduce the inner workings of Sketch Based Retrieval. We will then proceed to give an overview of software that was already developed by others. The rest of the document will describe subsequently the assignment we received, the research we did before we started and the system design we made. After these preliminaries are out of the way, we will go over the implementation and we will close off with an evaluation in which we will discuss the results we achieved as well as the process that led to those results.

1.1 Sketch Based Retrieval

Sketch Based Retrieval (SBR)¹ is a technique that can be applied to vector based *drawings*. It uses the fact that Clip-Art is mostly made up of polylines and polygons, to which we will refer as *primitives* from now on.

A drawing is first classified by describing it in terms of *geometry* and *topology*. Geometry describes the shape of an individual primitive whereas topology describes the spatial relation between the primitives in terms of *inclusion* and *adjacency*. After this classification is complete the collection of drawings can be searched by comparing the geometry and topology information.

¹For an explanation of terms and abbreviations see the Glossary in Appendix A

Geometry comparison Comparing the geometry of two drawings is fairly straightforward. Of course it would be possible to classify each primitive in a drawing as a *circle*, *square* or *triangle*, etc. This would, however, severely restrict the shape of each primitive. Instead we compute a *descriptor* for each primitive that is based on eleven features (see Appendix B). Primitives with a similar geometry will have a similar descriptor. Once this is done, we can use the Euclidean Squared Distance between two geometry descriptors as a measure of similarity.

Topology comparison Comparing the topology of two drawings is less straightforward. First we need to decide on a way to represent a drawing's topology. As topology uses inclusion and adjacency, using a graph is the first thing that comes to mind. A child-link would indicate inclusion and a sibling-link would indicate adjacency. The problem of comparing the topology of two drawings is now reduced to determining whether their topology graphs are isomorphic or nearly so.

Unfortunately the problem of determining (sub) graph isomorphism is in NP. To overcome this problem we determine the graph's adjacency-matrix and use its eigenvalues as a descriptor (as illustrated in figure 1.1). As was the case with geometry, graphs with a similar topology will have a similar descriptor and again we can use the Euclidean Squared Distance between two descriptors as a measure of similarity.

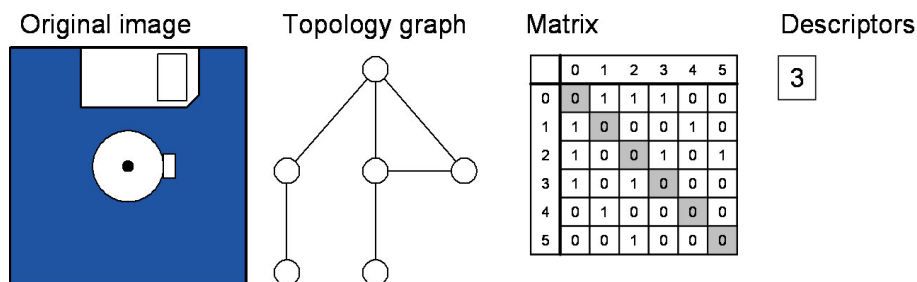


Figure 1.1: Computation of the topology descriptor

A drawing typically contains over a hundred primitives, this results in the same amount of descriptors. Now say we want to compare a sketch with ten primitives to a thousand drawings, each with hundred primitives. This would result in 1.000.000 comparisons. Clearly this is impractical for real use. We have devised two solutions to this problem.

- We try to simplify each sketch so that it only contains information relevant for recognition. This simplification consists of modifying or completely remove primitives based on their relevancy for the image. Primitives which are kept are then simplified themselves so they do not contain any superfluous points. These steps are performed during classification and greatly help reduce the computational burden during search.
- We reduce the number of possibilities for candidate images by first filtering the collection using a relatively simple pre-selection criterion.

1.2 Previous work

Building a project like this from the ground up takes a lot of time. However, a lot of code was already written and this made our task seem easy, at first. Unfortunately the majority of this code was undocumented and uncommented, so a lot of time was spent understanding the work of our predecessors. This almost nullified the advantage of not having to write the code ourselves. We will shortly discuss the different libraries that were available.

1.2.1 CALI

CALI is a calligraphic library written in C++ and developed by Manuel João da Fonseca. It is capable of recognizing shapes (e.g. *square*, *circle*) and gestures which are sequences of movements with a mouse or pen. Either predefined gestures can be used or gestures can be ‘learned’.

It also has the ability to describe the shape of a polygon by using a descriptor. This descriptor is based on 11 features (see Appendix B). The advantage of using a descriptor over a description like *square* is that it is not limited to predefined shapes.

Although CALI is a very powerful library once you know how to use it, getting that far may be a bit of a challenge. The documentation in the source code (as far as it is provided) does not provide enough help in understanding the precise functions of the classes and functions.

Fortunately the author of CALI was one of our direct supervisors. Whenever we did not understand something about CALI we could always ask him directly.

1.2.2 Sbr-core

The sbr-core is a program that is capable of Sketch Based Retrieval and uses the technique described in the introduction. It can be called with different parameters to either perform the classification of a new drawing –i.e. compute its geometry and topology descriptors– or to perform a query on an existing set of drawings –i.e. compare the descriptors of a drawing to a set of previously computed descriptors.

The source code of sbr-core lacks any form of documentation nor is there any description of the global functioning of the program. The source is not commented at all. The only hint the source code gives about its functioning, is the usage description the executable prints when it is called without parameters. All Input/Output to the program (except, of course, the command line parameters) is done through files that reside on disk.

The program uses the library NBtree (Normalized Balanced Tree) to store descriptors. For a description of the NBtree format see [Fons02] [Fons03-1] [Fons03-2] [Fons03-3].

1.2.3 BajaVista

BajaVista is basically a graphical user interface for the sbr-core. Using BajaVista a user can draw a sketch with which to query the system. The executable sbr-core is then passed this sketch to query the database with it. The returned results are displayed so the user can select and save the image of his choice.

Image processing is done internally with classes. These classes are able to read Microsoft's WMF format and apply several simplification algorithms to improve both processing speed and relevance of retrieved results.

The source code of BajaVista is incomprehensible for anyone who is not Portuguese (and probably for anyone who *is* as well). It is not documented and the names of the variables, the names of the functions and the sparse comments in the source are all in Portuguese. Furthermore the code is a mess and poorly designed.

Chapter 2

Assignment

The assignment we received consisted of several parts. We will briefly describe the different parts of the assignment here.

2.1 Client/Server Architecture

First of all we had to rewrite the existing code (as described in section 1.2) and change it into a client-server architecture. In this scenario the client would facilitate in communication with the server and function as a graphical user interface (GUI). A by-product of this rewrite had to be the conversion of the existing program *sbr-core* into a library (which we have called SbrCore).

There were a couple of extra demands on both the client and the server, which are discussed in separate sections.

2.1.1 Client / GUI

The user interface had to consist of a thin client with a few important functions. The user had to be able to draw sketches by means of strokes and gestures. The client then had to facilitate the communication of these sketches to the server and any further interaction, like displaying the results returned by the server, in the image retrieval process.

Summarized, the client should:

- Enable the user to draw a sketch to use as query.
- Present the results of a query to the user.
- Enable the user to edit a sketch using gestures.
- Allow the user to use one of the results from a previous query as a sketch (which in turn can be edited like any other sketch).

A very important non-functional requirement for the client, besides ease of use, was ease of deployment. The user had to be able to install and run the application easily and securely. The client should also function on different platforms (e.g. Linux, Windows, OS X, etc.).

It was desirable that some of the editing commands were implemented as gestures, as long as they would be logical and self-explaining. We wanted the user to get started with the application as quickly as possible. Putting a function behind an obscure gesture where a button would be much clearer (e.g. *New*, *Query*) would only add confusion.

2.1.2 Server

The server processes requests from the client via a communication layer. Depending on the request it has to dispatch appropriate processing requests to its subsystems, collect the results and return them to the client.

Non-functional requirements can be posed on the response speed of the server. The server should not drop client requests when it is too busy handling other clients' requests. A typical client request should receive a reply within a couple of seconds. A longer delay would most certainly annoy the user.

The server also had to be cross-platform compilable. This meant that we could program on the Windows platform but could not for example use Windows system calls. It was also mentioned that the technology to be used had to be Open Source or compatible.

2.2 Improving simplification

To improve the results as well as to speed up the processing of queries, Baja-Vista employed several simplification algorithms. These were applied to the drawings in the Clip-Art collection, prior to performing classification.

Originally there were four simplification algorithms:

Vertex reduction After a WMF-file had been read, the image was simplified by an algorithm which removed vertices to close to one another. After that the Douglas-Peucker algorithm (see 5.4.2) was used to further decrease the vertex count.

Colour gradients The following step in the simplification process was the removal of surrounding primitives. If one primitive lies within another and their colours are alike enough the inner primitive is removed.

Contours Contours were removed using an algorithm that searches for parallel lines.

Small Area Finally primitives smaller than a certain threshold, relative to the size of the main polygon, were removed.

The effect these algorithms had on the classification process was dubious. In figure 2.1 you can see that what is supposed to be simplification can eliminate important pieces of the image. This is clearly not desirable.

This meant that the simplification could be improved in two ways. The current set of algorithms could be debugged and tuned to yield better results and we could come up with more ideas ourselves.

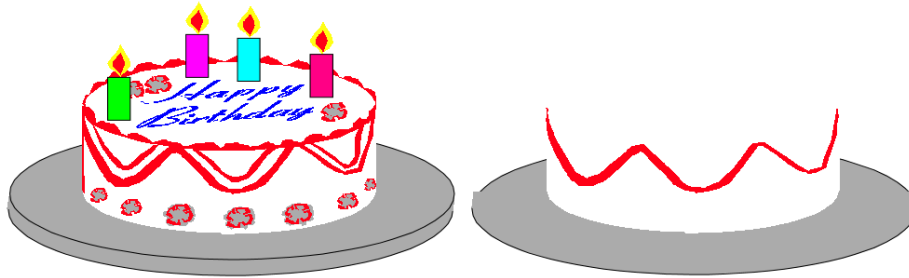


Figure 2.1: Left: prior to simplification, Right: simplified image by BajaVista

2.3 Improving SBR

Although the method of Sketch Based Retrieval –as described earlier– works, it lacks refinement. Using a topology-graph it is impossible to make a distinction between a *circle* including a *square* and a *square* including *circle*. Furthermore, two primitives are either adjacent (their borders touch) or they are not. There are no levels of adjacency like *adjacent*, *near* or *far*.

To solve these problems, we had to extend the SBR-mechanism by adding *geometry hints* and *adjacency weights* to the topology-graph.

2.4 Planning

Before the start of the project we received a draft planning from our supervisors indicating how they would like the project to proceed. This planning took into account a couple of weeks of time to start-up, analyze requirements and grasp existing source code. After that it set incremental targets for deliverables each week. The presentation of each deliverable was accompanied by a meeting with the entire team and both supervisors discussing progress, requirements and difficulties.

Chapter 3

System Design

The starting point of the system design was the previous application, BajaVista. The diagram 3.1 was made by us during the inventory phase of the project. The class names, the names of the methods, the names of the attributes and the few comments that were present were in Portuguese. These were decoded with the help of a dictionary and Oliver's knowledge of the Portuguese language.

The main problem with the BajaVista design is that pieces of functionality are not divided in modules. Responsibilities are not clearly separated and it may very well be the case that one piece does many things. The class *Imagem*, for example, is responsible for the reading and loading of WMFs from disk. It stores the WMF primitives it reads using the class *Primitiva*. However, *Primitiva* is not just used to represent a WMF-primitive, it is also responsible for reading certain WMF-commands from disk and interpreting them. Apart from the fact that this confuses people who are not familiar with the code, this mix of responsibilities makes it impossible to reuse *Primitiva* for, for example, the SVG-format, even though there would be no change of concept (both in WMF and SVG a primitive represents a polyline or polygon, with a border and a fill-colour).

For us to be able to make a new application, we had to functionally decompose what we had and change it into a new system (see 3.2). In this system the *dejaServer* class functions as a hub for the functional modules. It uses *dejaDatabase* for its persistent storage. It recognizes shapes and topology using the *SbrCore* which in turn uses CALI. Further search and classification is also done by using *SbrCore*.

3.1 Client

The client consists of two modules: *DejavistaApp* (named *Applet* in the design in 3.4) and *JCali* (a port of CALI).

The *DejavistaApp* is an application run by the user and can be used to query the system. Its user interface consists of a *SketchPane* on which the user can make his sketch, a *ButtonPane* with buttons for common actions and a *ThumbPane* on which the thumbnails of the query results are displayed. It has an internal *Sketch* representation which it can send to the server for querying using

its ClientCommunication interface. Using the results of the query it will then retrieve the thumbnails for the result images using a separate thread Thumb- Retriever.

JCali is a port of the CALI library which the DejavistaApp uses to determine if the user is drawing a gesture. It is a direct port of the C++ version.

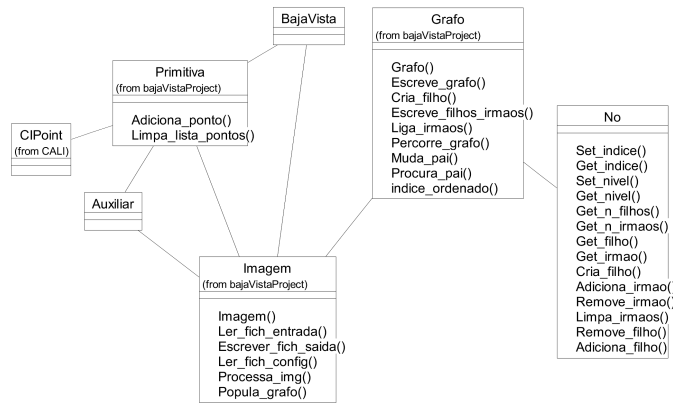


Figure 3.1: Diagram of the original BajaVista application

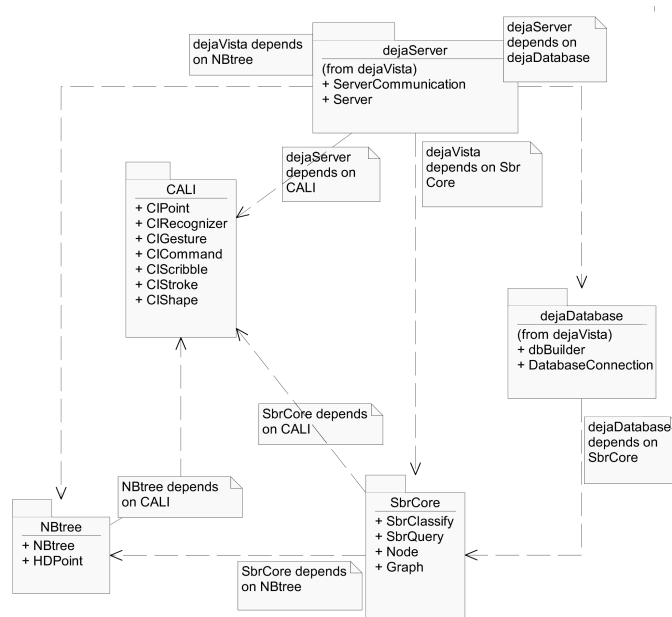


Figure 3.2: Diagram of the System Architecture

3.2 Server

Our primary concern while designing the server was to keep it as modular as possible. This way it would be easily extendable in the future. Still, a link exists between classes NBtree and Database. This dependency could have been removed by storing raw data (BLOBs) in the database. This was not a straight forward procedure and was not prioritized during implementation. We did however take this into account during the design process and it could still be easily implemented in the class DatabaseConnection.

We will summarize the modules used by the server (see fig. 3.3):

Database For storage of descriptors, Clip-Art, thumbnails and simplified Clip-Art drawings.

Drawing library For drawing simplification as well as for conversion between different image-formats.

SBR library Used for matching and classification of drawings.

Communication library For communication with clients.

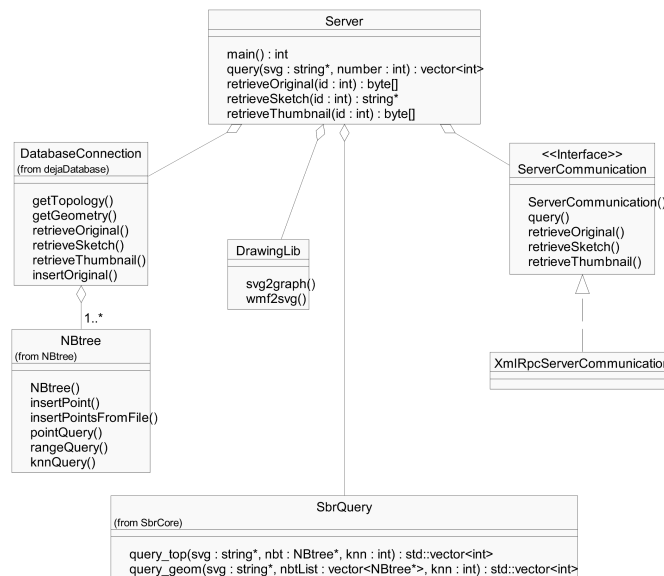


Figure 3.3: Class Diagram of the Server

3.3 SBR

The package SbrCore is separated into three different classes:

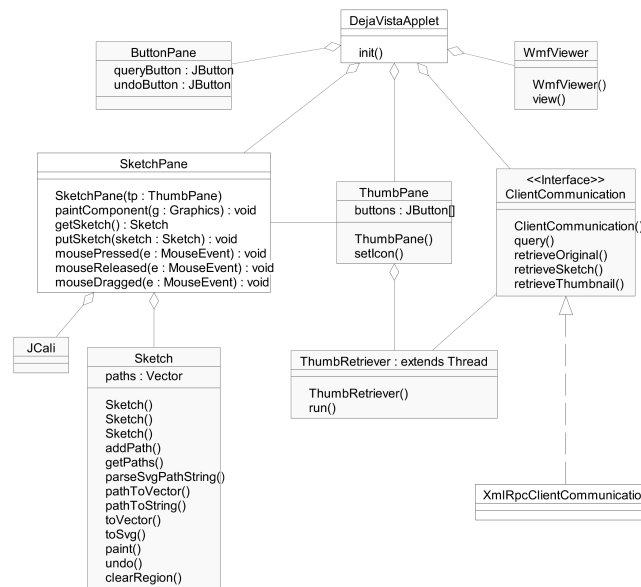


Figure 3.4: Class Diagram of the Client Application

SbrCore This class contains generic methods that are needed by both SbrQuery and SbrClassify. Separating classification and querying keeps the resulting executable as small as possible. Furthermore this way the query mechanism –currently first topology, then geometry– can be altered without affecting other parts of the system.

SbrQuery This class contains methods used in querying the system.

SbrClassify This class contains methods used in classifying images.

3.4 Sequences

3.4.1 Query

In figure 3.5 the sequence of actions when the user initiates a query is illustrated. The user makes a sketch and the DejavistaApp queries the server with that sketch (passed on as a string). When the query reaches the server, topological information is retrieved from the Database and a topological query ('query_top') is performed. The result of this topological query is a pre-selected set of candidate results. These candidates are then further evaluated using geometry. For each image the necessary geometrical information is retrieved (using 'getGeometry' with the integer id of the image). Then the candidates are further culled by querying (using the 'geom_query' function). The final results a list of image ids and relevance percentages are returned to the client.

The client then has to retrieve the thumbnails of the resulting images itself. For each thumbnail it wants to get it calls the 'retrieveThumbnail' method

with the correct id. This call is passed over the communication layer through the server’s internals until it reaches the database. The database returns the thumbnail image from storage which is then passed back to the client.

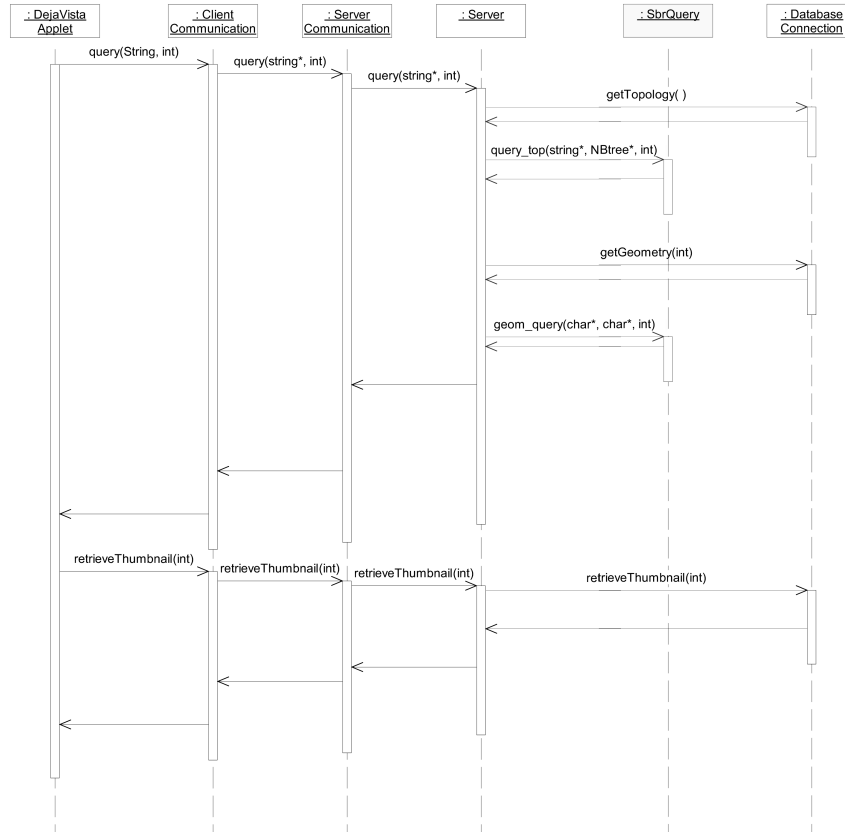


Figure 3.5: Sequence Diagram of a Query

3.4.2 Classification

In figure 3.6 the simpler sequence of actions when classifying an image is displayed. The dbBuilder retrieves the topology tree using ‘getTopology’ because the topological information of all images is stored in a single tree. The values of the new image have to be inserted into that tree.

It goes on to classify the image. This classification steps performs both the topological classification and the geometrical classification. The geometric information is stored separately for each image into the database (‘storeGeometry’). When the dbBuilder is finished it can also store the complete topology tree it now has in the database using ‘storeTopology’.

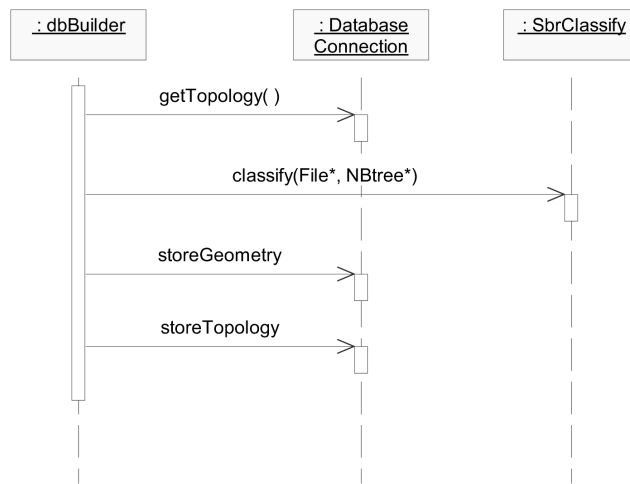


Figure 3.6: Sequence Diagram of the Building of the Database

Chapter 4

Research

After we had decided on a global design, we strode off in a quest for tools and libraries to make development easier and faster. The following chapter describes the many options we stumbled upon and explains the choices we made.

4.1 Information Management

During a project of this nature we are used to acquiring, processing and producing large quantities of information. We have employed two (for us) new applications to manage these information flows. We will discuss them briefly here.

4.1.1 MoinMoin

MoinMoin is a web-based tool to collaboratively organize a knowledge base.

The problem faced when doing research for a project is that this is usually an individual task. All members of the team go off on a different direction exploring possibly useful sources of information. Whenever information is found that is relevant this newfound knowledge has to be shared with the other project members.

Past experiences indicate that without a central place to share/store this knowledge it will not be used effectively. Usually someone who has found something lets the other members of the group know. They take note of the fact and forget it again if it is not immediately relevant.

A new way of information organizing which is gaining in popularity is the Wiki. This is a website with any number of interlinked pages. The thing that makes a Wiki different from a normal website is that anybody visiting it is allowed to edit/add pages using a browser. This fact, although it may seem dangerous, stimulates an open and dynamic exchange of ideas. MoinMoin is a specific implementation of a Wiki. Many others exist.

We had no previous experience with software like this except for having read some technical documentation from sites already using a Wiki. Those experiences led us to believe it would be a good idea to set one up and use it.

Installation and use both proved very simple. Mark-up of text is done via a special syntax (no HTML) which is easily learnt. The free modifiability of the web allows everyone to put whatever they think is important on it. Some organization has to be agreed on by everyone otherwise certain pages (e.g. the FrontPage) can become a horrible chaos where it is impossible to find things again.

The Wiki resulting from our project is still online¹. In it you can see the kind of information we recorded during the course of the project. Allowing so much information to be gathered in one place and making it available for everyone is one of the strongest suits of the Wiki concept.

4.1.2 Subversion

We also needed a tool to allow for us to share and version files easily. On previous projects at Delft we had to use the BSCW (Basic Support for Cooperative Work), a server supposedly enabling group work. This server only allowed uploading, downloading and versioning single files via a web interface. This is severely lacking for any kind of serious work.

A couple of us already had experience using CVS for version control. We quickly agreed that we would need similar functionality for this project. We chose Subversion over CVS because it was also free, it had superior features, it recently had had its 1.0 release and we had heard nothing but positive experiences about it.

Subversion, like CVS, is a command line network-oriented tool for version control which uses the copy-modify-merge paradigm. This means each user can check out a personal source tree, make his modifications on this tree and finally merge his changes back into the main trunk. Unlike CVS, Subversion is designed and implemented from the ground up to meet current version control needs. And it does meet those needs superbly. The process of setting up a server and using it is very adequately described in [Coll04]. Experienced CVS users do not really need a guide because the Subversion commands are very much like those of CVS.

The commands may not be that different but their use is. It was evident from the start that Subversion is very well designed and all common usage scenarios are well thought out. It provided flawless support for versioning as well as for common actions such as moving files and making tags. Besides a command line client which is more than sufficient for day to day use it also has several advanced tools for version control and merging as well as advanced integration into the Eclipse platform.

4.2 Programming Languages

4.2.1 Java

Our project's starting point was the already existing BajaVista application with its functioning core. One fundamental task was to pull the application which

¹<http://fun4me.demon.nl/moinmoin>

ran on a local workstation apart into two separate pieces: a client part to draw sketches and display images; a server part for the real work and the storage of data. The client would have to be easily and securely runnable by any end user who would like to use the application.

These requirements fit the Java programming language perfectly. Java is a mature and powerful programming environment for which a plethora of tools and third party libraries exists. Java is also widespread in industry and academia which means that a lot of people (including us) have a ready command of the language.

Other technologies which might have been able to fulfil our requirements were the .NET platform and the Mozilla platform. Both of which promise to enable remote deployment of content rich applications. The unprovenness of both these platforms and our lack of experience in either of them stopped us from further investigating these avenues.

4.2.2 C++

In our new setup most of the work done by the old BajaVista application would be moved to the server. BajaVista was written in C/C++ so it seemed a logical choice to migrate and refactor that code base to form our new server.

Another argument for the use of C/C++ was the often heard claim that the increased performance C++ offers would be essential to the kind of application we are developing. Converting and manipulating images and querying large datasets were going to be very intensive functions of the application to be developed.

One of the biggest drawbacks of picking C++ as a development language was the fact that none of us had sufficient experience developing in this behemoth of languages. We were not discouraged by this lack and started by reading the existing code to increase our understanding of the way the application was programmed. During this process we also began improving our knowledge of the language by reading the books by Bruce Eckel [Ecke00] [Ecke03].

4.3 Communication

We knew the client was going to be deployed remotely. This meant that a means of communication between the client and the server was going to be necessary. We investigated several possible solutions to this problem and finally picked XML-RPC as our solution. We will discuss the reasons why its alternatives were not found suitable to our problem and why XML-RPC did meet our needs.

4.3.1 RMI

Remote Method Invocation is a Java based technology which allows for remote Java applications to call functions on each other transparently. A Java class can expose its public methods to a remote caller. This seems ideal but the fact that RMI is limited to Java-to-Java made this unsuitable for our application.

4.3.2 SOAP

Simple Object Access Protocol (SOAP) is a “stateless, one-way message exchange paradigm” [W3C03] which enables communication and also interaction between several SOAP nodes. It is quite similar to RMI with the difference that SOAP is usable across a wide variety of languages. Like RMI setting up SOAP nodes and enabling them to communicate with each other is not immediately obvious and has some non-trivial overhead.

4.3.3 Low level

Another option would have been to write our own protocol for the communication using sockets on both sides. This would have provided us with full control of this part of the application.

We were not very inclined to follow this option firstly because we did not need full control over this specific non-core part of the application. Furthermore the many pitfalls in writing your own protocol and our lack of experience in low level network programming lead us to guess that any control we would win would be lost tenfold in implementation and debugging time.

4.3.4 XML-RPC

The one sentence explanation from the specification is that “XML-RPC is a Remote Procedure Calling protocol that works over the Internet” [Wine99] already says it all. This is well in tune with the goal of XML-RPC being as simple as possible.

XML-RPC is an intermediate protocol that enables calling functions and returning return values in a platform and language independently fashion over the network. This independence is guaranteed by translating function calls and returns to the intermediary XML format before sending them via HTTP. A function call `–name and parameters–` is unambiguously encoded as a well formed XML document containing a `<methodCall>` element. The receiving side unambiguously decodes this document to a function call and returns the return value to the function call in a similar XML document containing a `<methodResponse>` element.

We chose XML-RPC as the protocol for the application because of its simplicity of setting up, its usability and extensibility.

An XML-RPC interchange only has the following three settings: the server name or IP, the port to connect on and the path to the handling service so for example `http://google.xmlrpc.com:80/RPC2` would be a valid address. The server on that address has a public interface and has handler functions for all of its methods that can be called. These calls are nothing more than the correct XML written to the specified destination via HTTP. That XML is then mapped to the receiving client’s language’s data types and appropriate action is taken.

The biggest con to the use of XML as a data transport medium —and so in using XML-RPC— is its verbosity. This verbosity manifests itself firstly in wrapping the types to be transported in an XML envelope which is bigger than strictly needed as shown in the following example:

```
<?xmlversion="1.0"?>
```

```
<methodCall>
  <methodName>name</methodName>
  <params>
    <param>
      <value>
        <i4>1</i4>
      </value>
    </param>
  </params>
</methodCall>
```

Here transmitting a single integer parameter as a method call uses 131 characters of space. The transmitted SVG images are also in the XML format and so are also slightly more verbose than would be strictly necessary.

The application involves transmitting a lot of binary data (PNG and WMF images). This binary data has to be encoded in base64 for safe transmission which adds even more overhead.

We could ignore these surpluses and assume the overhead is negligible. Preliminary use (about 10 users) indicates that it is. Scaling the amount of users may be limited by the total of available bandwidth. A possible solution to this problem would be to enable end-to-end compression in the HTTP layer of XML-RPC. This approach will take a lot of load off of the connection between the client and the server. Instead it will shift the burden to the server's processing power.

4.4 Image Processing

The application under development will use images for several different purposes. The client needs to display the results from a query. This is likely to be done in the form of thumbnails. It also needs to be able to show the user a preview, to internally store the user drawn sketch and to be able to send this sketch to the server. Finally we need to take the image format, used by most Clip-Art collections, into account.

In this section we will give an overview of several available image-formats and the libraries that can be used to process these formats.

4.4.1 WMF

The image-format used by most Clip-Art collections is WMF (Windows Meta File). Since the collection of images readily available to us was in this format as well, dealing with this format was inevitable.

A WMF contains Windows GDI (Graphics Device Interface) calls. These calls contain instructions on how to draw an image on screen. On the Windows platform this is an extremely fast way of drawing and storing vector images. Other platforms however, do not implement the GDI and are therefore unable to display WMF without an intermediate conversion. Since the client is to be cross-platform, we could not assume native WMF displaying capability. We

thought of the possibility to use a library for Java to display WMFs, but unfortunately available libraries are either non-free or incapable of displaying WMF correctly.

Apart from this, graphic calls are not easy to manipulate. It was therefore desirable to be able to convert WMFs to a useful format and use WMF as little as possible.

The first library we found that was capable of reading and manipulating the WMF format for C++, was LibWMF. This library, which is open source and originally written for the Linux platform, looked very promising at first. However, because of dependency problems we were unable to include it into our project.

After some further investigation we found another library, called ImageMagick of which LibWMF is a component. ImageMagick contains a lot of other popular open source image libraries and combines those using a common interface.

ImageMagick can be used both under Linux and Windows and is easily included into all kinds of different programming languages. This library was much easier compiled and utilized in our project because of its lack of external dependencies. The fact that the `make/project-files` were included with the source-files was another much needed convenience.

The library contains methods to convert images to and from almost all image formats. Currently the only drawback is that it cannot export to a vector-based format yet. ImageMagick is still under development and this is something that still needs to be worked on.

4.4.2 SVG

It was clear from the beginning that Scalable Vector Graphics (SVG) had to be used in the communication of the query between client and server. This decision was made based on idealistic, rather than technical considerations.

SVG is a completely open-standard for vector-based 2D graphics and is defined using an XML grammar. When using Windows as operating system, it is not as fast as the WMF format. It is however, because of its open nature, very easy to implement. Furthermore it makes debugging easier as it uses XML to store the image data (XML data is stored as plain text).

One of the drawbacks of the SVG-format is the relatively large file size. This is caused by the overhead of using XML. This problem can be overcome by using the extended format SVGZ. SVGZ is a compressed version of the SVG format, using the GZIP compression algorithm.

4.4.3 PNG

Although most image-format needs on the server-side are solved by using WMF and SVG, we still needed a way on the client-side to display thumbnails and previews. There was the option of using a vector-based format and the option of using a bitmap-based format. Research almost immediately drifted

towards the last option as using a vector-based format would imply writing code to convert WMF to this format and Java has no native or open-source support for this, whereas C++ libraries to convert WMF to PNG were available.

Out of the several bitmap formats that were available, PNG (Portable Network Graphics) turned out to have the best size/quality ratio. Another advantage is the fact that it is an open standard, instead of a proprietary format.

4.4.4 Drawing

Since the ImageMagick library is still in active Development, their support for vector-based images is unfinished at the moment. It was therefore impossible to use the vector-based format that ImageMagick uses internally. Because of this lack of support for vector-based images in ImageMagick and the non-existence of other simple and useful libraries, we introduced our own Object-Oriented vector-based image format, which we simply called *Drawing*.

The class *Drawing* contains methods to add primitives (both lines and polygons, with their border- and fill-colour where applicable) and methods for simplification. By writing sub-libraries for other vector based image formats, like WMF and SVG, *Drawing* also created a way of converting images from the Clip-Art collection to user sketches. This made it possible to manipulate WMFs and sketches in the same way.

When support for vector-based images in ImageMagick has been improved, it is relatively easy to convert the internal vector-based format of ImageMagick, the *Drawable*, to our format, *Drawing*. This can be done by altering the code we created to convert WMF and SVG to *Drawing*.

Chapter 5

Implementation

In this chapter we will discuss the various sections of our implementation.

5.1 Client

5.1.1 Java Web Start

One of the requirements to the client was that it had to be easily and securely installable on a wide variety of operating systems. A nice bonus would be if it would update itself automatically. These requirements are usually met by using Java applets but these are becoming rapidly outdated and setting one up securely is a tremendous hassle.

As of version 1.2 of Java an alternative has become available called Java Web Start. As opposed to applets web start applications do not have to be run inside a web browser, instead they run directly inside Sun's Java VM. The application manager caches the application and updates it when it is changed on the server. Because the application is already local in most cases and you can create a shortcut to it wherever you want, this blurs the distinction between native applications and applications deployed over the Internet.

5.1.2 GUI

We started the implementation of the client by making the drawing canvas on which the user could draw a sketch. On the lower side of the canvas are the thumbnails of the results and on the right side are the buttons *Search*, *Undo*, *Clear* and *Help*. The *Undo* was first implemented with only one undo, but later replaced with a lengthy history of all stages of the drawing. The *Help* button displays some hints on how to use the application.

We also had to think of a way that enabled the user to do multiple things with a thumbnail; you can preview the drawing, you can save it to disk, and you can use it as a start for a new query. First we wanted to just give some sort of menu when the user clicked the thumbnail, presenting all the possible actions, but then we decided that dragging the thumbnail to the drawing canvas to edit it for a new query was more natural. If the user clicks on the thumbnail, the preview image is displayed with the option to save the WMF to disk.

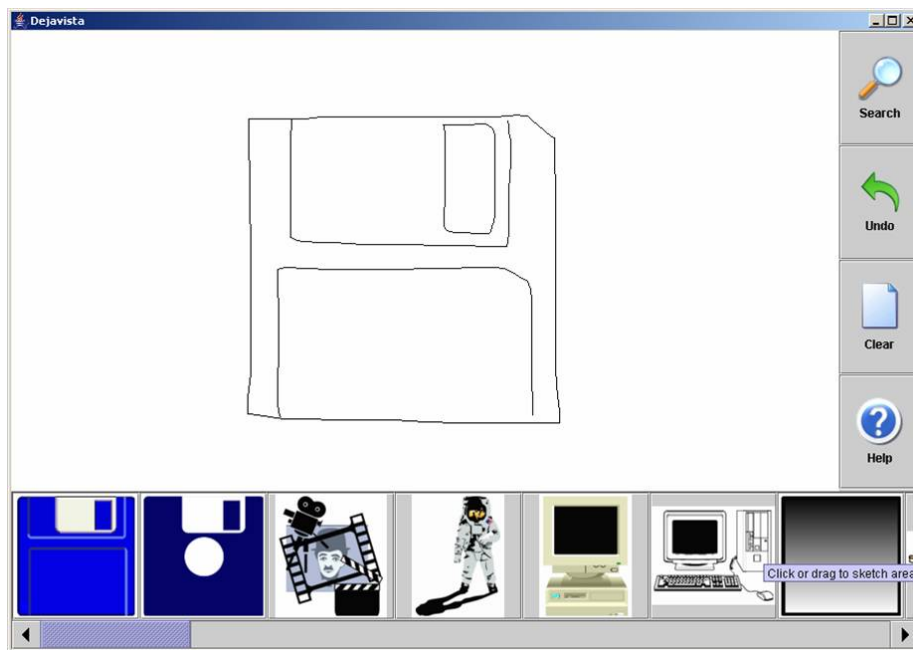


Figure 5.1: Java Web Start Client

5.1.3 Communication

The client –being *thin*– has to outsource as much as possible to the server. It only did very core GUI and drawing tasks itself. The functionality the client could access on the server will be treated shortly in 5.2.1. These server side functions have to be accessed using XML-RPC as already explained in 4.3.4.

Implementation of the XML-RPC communication on the client was done using the Apache XML-RPC module¹. Using this library consisted of adding the correct JAR to the project, instantiating the `XmlRpcClient` class like this: `XmlRpcClient xmlrpc = new XmlRpcClient ("http://localhost:8080/RPC2");` and then calling its `execute` method with the remote methodname. This entire procedure was just as easy as expected.

5.1.4 Gesture support

One of the requirements for the client application was support for gestures for certain actions. Though implementing basic gestures should not be too difficult, implementing generic gesture support in a flexible way was a task deemed beyond the scope of this project.

Fortunately one of our supervisors, Manuel João da Fonseca, had already developed a library which did exactly this. A major problem for us was the fact that this library was written in C/C++ and our client was to be written in Java and deployed remotely.

¹<http://ws.apache.org/xmlrpc/>

We solved this problem by porting the entire CALI framework (about 30 source files) to Java. The fact that C++ and Java are syntactically and structurally very similar made this task much easier than we initially thought. The porting process consisted of pre-processing the .cxx and .h files into a single .java file. After the pre-processing we had complete information in each .java file and duplicate declarations of functions had to be cleaned up and all the information had to be put in one place. Having done all of this the process was pretty straightforwardly removing all the *strange* C++ operators, pointers (*), references (&), new and delete keywords etc. The Eclipse IDE has a Java syntax-checker as you type. This made our work a lot easier, reducing it to simply making all the red squiggly lines disappear.

Porting towards a garbage collected language is certainly much much easier than porting away from one would be. We did have to write a lot of casts to port the templated CIList data structure to a Java equivalent. If we could have used *Java 5* with its auto-casting facility the templates could have been saved with seemingly identical code.

For the pre-processing we used a very basic Python script which still left a lot of boring manual work. We could have expanded that first to make further changes before manually making it conforming Java code. Passing the C++ code through a source code formatter before porting would also have saved us some work in the Java version. We did most of the work by hand.

The functional quality of the resulting Java code surpassed our expectations. Before porting we could not be sure of the functioning of CALI. The framework was unfamiliar to us and we had not field tested it for ourselves. We also did not have any unit-tests which we might have ported as well and run on the Java version. The final Java version after porting and compiling contained a few runtime errors which were easily debugged. After that it was ready to be incorporated into the client application and do its work.

This work consisted mainly of recognizing the delete gesture. This gesture first deleted the portion of the sketch covered by the convex hull of the gesture. This was expanded to delete an entire stroke if part of it fell inside the delete gesture. Later on we tweaked the delete action to improve its recognition. The idea of removing CALI and implementing the recognition of the delete gesture ourselves was rejected because we wanted to keep the option of adding more gestures open.

5.1.5 WMF preview

The client also had to display the WMF as a preview before saving it, so we investigated some ways of doing this. We found some libraries that could do that, but they were usually not free and they could not display all WMFs correctly. We also ported the WMF viewer part from BajaVista, but that was not able to display all WMFs correctly as well. Finally we used ImageMagick on the server-side to convert the WMF-format to PNG-images. These PNG-images were used as thumbnails and previews of the WMFs and were returned upon request. We did however use our own WMF viewer a couple of times to display the structure of WMFs while creating the WMF simplification methods for the server-side.

5.1.6 Selection

Another decision was how to select a part of the drawing. The first idea was to use the right mouse button for this action. This was rejected because *dejaVista* was meant to be used with a drawing tablet or a tablet PC with a pen. Some of the pens used by these devices do not have a (substitute for the) right mouse button. Using buttons also goes against the sketching paradigm we were using all this time. Another option which was rejected was to enter selection-mode by making a gesture or clicking a button. Finally we decided to use press-and-hold to enter the selection mode.

When something is selected, you can drag it to another place, delete it by using the delete-gesture onto the selection, or crop it by making the delete-gesture outside the selection. To unselect, the user has to press-and-hold and release the mouse button without selecting anything.

5.2 Server

5.2.1 Communication

In the Chapter Research (4.3.4) we explained the reason for choosing XML-RPC over other means of communication. We implemented the XML-RPC communication using a LGPL C++ library called *XmlRpc++*². This library enables making an XML-RPC server using C++ classes –the class names are the method-names. After the initial setup is done the server starts its main event loop and services XML-RPC requests as they come in by dispatching the *execute* method of the proper class.

Using the server we have now developed a client with the functionality of retrieving vector drawings by sending sketches. But because all the server's functions are publicly exposed (for the complete interface see table 5.1) it is easy to write another client which connects to the same server but offers a different range of functionality. One possibility would be to implement an image library that displays thumbnails.

The methods as specified in table 5.1 have the following description:

Query Performs a query on the system (for the sequence diagram of this action see figure 3.5) passing it an SVG representation of the query sketch. The received sketch is first simplified using *DrawingLib* and then passed to the *SBR* library to perform a topological query. Based on this query, the server retrieves the geometrical descriptors needed from the database and performs another call to the *SBR* library, this time to perform a geometrical query. The result of this query is an array of ids of images that are similar and their respective dissimilarity to the query. The results are in ascending order by dissimilarity (so the id of the most similar/relevant result comes first).

RetrieveThumbnail Retrieves the small thumbnail of the image with the given id as a PNG image.

²<http://xmlrpcpp.sourceforge.net/>

RetrievePreview Returns a thumbnail image of the specified width and height of the image with given id.

RetrieveOriginal Returns the image file and the name of the file corresponding to the given id.

RetrieveSketch Returns a string containing a simplified SVG representation of the original image.

Additional methods have already been added to the server due to changing requirements. More can be added when the need arises.

Method name	Parameters	Return value
<i>Query</i>	string with SVG	array of structs with members <i>id</i> (integer) and <i>diss</i> (double)
<i>RetrieveThumbnail</i>	integer with imageid	base64 data
<i>RetrievePreview</i>	integer with imageid, integer with the width, integer with the height	base64 data
<i>RetrieveOriginal</i>	integer with imageid	struct with members <i>imagename</i> (string) and <i>imagecontent</i> (base64)
<i>RetrieveSketch</i>	integer with imageid	string

Table 5.1: XML-RPC interface to the server

5.3 SBR

The implementation of the SBR library mainly consisted of converting the original program, as developed by Manuel João da Fonseca, into a library. Apart from this a couple of enhancements were made to the computation of the topology descriptor.

5.3.1 Adjacency weights

In order to differentiate between a drawing with two squares which are close together and a drawing with two squares that are far apart, we introduced the notion of the *adjacency weight*. Where an adjacency-link was either present or not, we now compute the (normalized) distance between to primitives and use this value for the adjacency-link. This process is illustrated in figures 5.2 and 5.3.

5.3.2 Geometry hint

To make it possible for the SBR method to recognize the difference between a *square* containing a *circle* and a *circle* containing a *square*, we introduced the notion of the *geometry hint*. We changed the computation of the topology-graph, so that the inclusion-link contained more information than either inclusion or

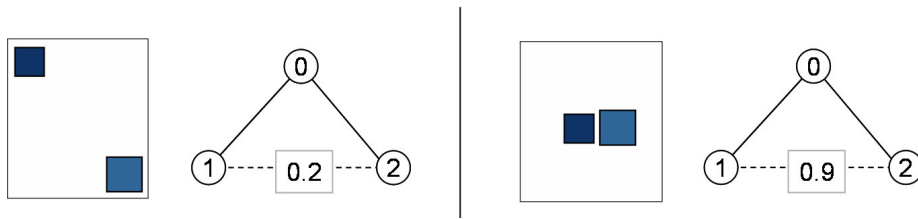


Figure 5.2: Using the adjacency weight to differentiate between two possibilities.

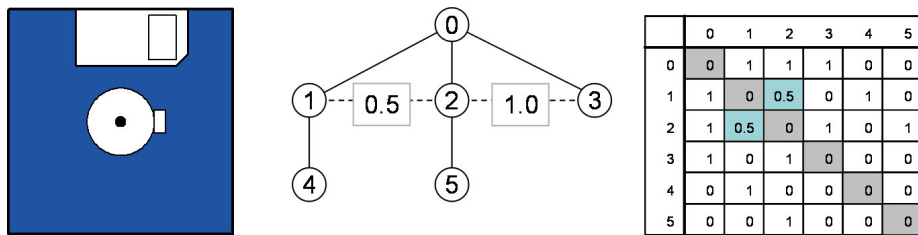


Figure 5.3: Graphical representation of the adjacency weights in the graph and matrix.

no inclusion (see figure 5.4). For each primitive that is included, a single-value geometry descriptor is computed and used instead of the original one (which indicated-inclusion).

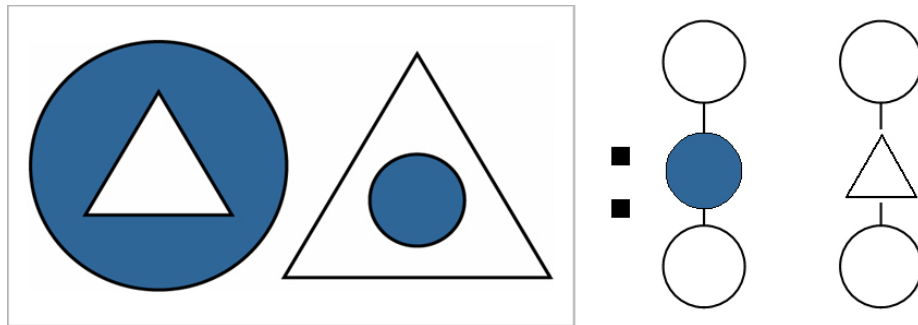


Figure 5.4: The geometry information is included in the inclusion link

The new descriptor is computed in the following way. First the *normal* geometry descriptor is computed. Then the first number of this descriptor is added once, the second number is added twice, the third number thrice, etc. This results in a single-value descriptor that is in no way unique, but should improve the results.

5.4 DrawingLib

The library DrawingLib is capable of reading, manipulating and converting all the image formats we use in our application. It functions as a wrapper around both external libraries and our own classes, some of which are based on code that already existed. This is done in such a way that the number of currently processable image-formats, as well as the available ways of manipulating (i.e. simplifying) images, is easily extendable in the future.

In this chapter we will explain how we implemented image conversion and discuss the importance of simplification as well as the simplification techniques we employed.

5.4.1 Image Conversion

To make it possible to process the WMF-images stored in the Clip-Art collection, we had to combine some existing libraries and add some code of our own.

We used the ImageMagick library to convert the WMF images to bitmap images (PNG) for the preview.

Because of the lack of write support for vector-based images in ImageMagick, we had to write our own code. The code from BajaVista was already capable of reading WMFs, but only processed them partially and it stored the data in a very large and inefficient class. This class was not extendable for use with other image formats. It would have been a waste to not salvage as much as possible though, so we created a new class by extending and altering the BajaVista code. This new class is also capable of converting WMF to our own internal format, *Drawing*.

For the SVG-format we created a similar class. It is capable of reading and writing SVG-images by converting them to and from our internal format. This class makes it possible to convert images from the Clip-Art collection to send them as a sketch to the client and to receive sketches from the client to compare them.

The use of the internal format to represent user sketches as well as Clip-Art collection images, ensures that all the images will be processed in the same way and is our first step towards creating a comparison that is as good as possible.

5.4.2 Simplification

The internal format *Drawing* is not only used to convert WMF to Drawing to SVG, but also contains methods to simplify itself. There are two reasons for simplifying a drawing from the Clip-Art collection:

1. It speeds up processing.
2. It yields better results.

Why simplification speeds up processing is obvious. Primitives consisting of a smaller number of vertices are easier to process. This goes for drawings with fewer primitives as well.

The reason why simplification yields better results is less straightforward. Clip-Art in WMF format is usually created in a very complex way, using hidden layers to create shadows, multiple polygons with slightly changing colours to create gradients, two different polygons to create a single polygon with a border, etc. For this reason, the internal structure of a Clip-Art image will differ enormously from a sketch drawn by a user. Simplification tries to overcome this difference by making Clip-Art look like a sketch as much as possible.

We will describe the different algorithms we employed in separate sections. Note that the order in which the algorithms are applied is very important.

Removing surrounding primitives

The first step of the simplification process is the removal of surrounding primitives. During this step, primitives that surround a smaller, but similar shaped primitive, are removed. This is done to get rid of lines that accentuate the surroundings of an object (e.g. borders and shadows), which the user would not draw anyway. For an example result of this process see figure 5.5 where all double lines from the grill of the car and some of the wheels have been removed.

A primitive is considered a surrounding primitive when at least 90% of its vertices are within range l from a vertex of the other primitive, where l is 0,6% of the length of the diagonal of the total image-size. These values have been found by trial and error and gave the best results on a series of test images.

When a surrounding primitive is a border (i.e. has the exact same size, a border-colour but no fill-colour), the border of the inner primitive is assigned the colour of the border primitive which is then removed. This is done to preserve as much information as possible, while reducing the number of vertices and removing unimportant primitives. We try to reduce the vertex count as much as possible, to speed up the rest of the simplification process.

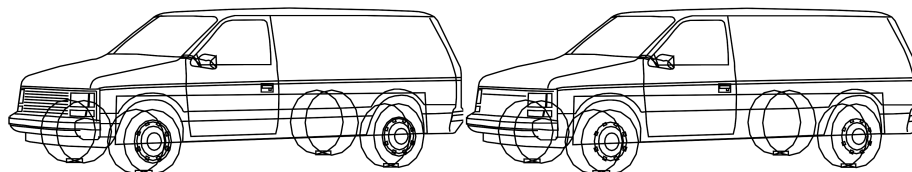


Figure 5.5: Left: a wire model of the original image, Right: a wire model of the image after the RemoveSurroundingPrimitives method

It is important for the simplification process to begin with this step, because of the following. Say we have two congruent primitives, one slightly bigger than the other. This implies that the vertices that make up the primitives are

also fairly close together. If we would first try to reduce the vertex count by the *ReduceVertexCount* step, this congruency can no longer be guaranteed.

After some testing, however, we discovered that without this reduction, this step would take too long to finish. To solve this problem we included a pre-ReduceVertex call, which only removes vertices which are very near (0,075% of the main diagonal of the total image) to other vertices. This way surrounding vertices can still be found and removed, while keeping the processing time within reasonable limits.

The *RemoveSurroundingPrimitives* step has to be done before the *ColorConcatPrimitives* step, because tests made clear that there otherwise could be some mistakes when adding, instead of removing, shadows from an object. And finally the *RemoveSmallPrimitives* step can only be performed after the *ColorConcatPrimitives* step, to avoid the removing of small parts of primitives which should be attached to other primitives in the image, to represent an object.

colour concatenate primitives

This step combines adjacent and overlapping polygons, by checking for borders, strict adjacency (i.e. their edges touch) or inclusion and matching colour. Depending on the combination of these parameters, a choice is made to leave the polygons untouched, to combine them or to perform an intersection.

When an overlapping primitive with a different colour, a border or both, is strictly adjacent to the primitive below (i.e. it sticks out), the primitive is cut out.

If its colour *is* the same and it has no border, the primitives are concatenated (i.e. a Boolean union operation is performed).

If it has a different colour or a border but is completely included it is left untouched, since a primitive, at least in the way we have defined it, cannot contain any holes.

If the overlapping primitive is completely included and does not have a border, it might be a gradient (as is the case in figure 5.6), and so we determine the difference in colour between the primitive and the overlapping primitive. If it has either the same or a slightly different colour, the overlapping primitive is removed in favour of the primitive below.

In this case, a polygon is said to have the same colour as the one it overlaps when their hue differs no more than 1,5%, their saturation no more than 20% and their intensity no more than 60%. These values have been found by trial and error. We chose to convert the colours from the RGB-format to the HSI-format, because it would make it easier to find and combine primitives with gradients.

Both the combining of overlapping primitives and the removing of gradients are done with the help of an external library called GPC³ (Generic Polygon Clipper). The end result is a drawing that is more similar to the way a user would draw it.

The newly created primitives will sometimes contain a lot of superfluous vertices, due to the unification and differentiation methods. This leads to an

³<http://www.cs.man.ac.uk/aig/staff/alan/software/gpc.html>

enormous slowdown of the simplification process. Therefore the pre-ReduceVertex method is performed once every 20 processed primitives and in this case the optimization value is raised until the vertex count of a primitive is below the threshold of a thousand vertices.

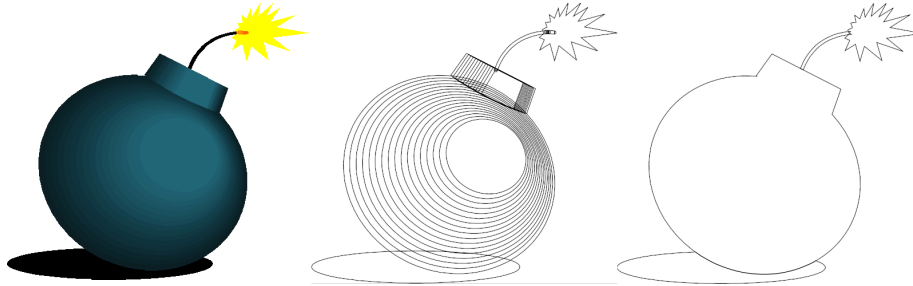


Figure 5.6: Left: the original image with gradient, Centre: a wire model of the original image, Right: a wire model of the image after the ColorConcatPrimitives method

Remove small primitives

Then the *RemoveSmallPrimitives* function follows, with which we try to further reduce the number of primitives in the drawing. This is done by removing those primitives, which are too small to be of any importance. This speeds up the comparison of images, because the number of geometry descriptors will be reduced. An example of this method can be seen in figure 5.7.

Small primitives are polygons with a surface of at most 0,025% of the total image size, or polylines with a length of at most 20% of the diagonal of the total image.



Figure 5.7: Left: a wire model of the original image, Right: a wire model of the image after the RemoveSmallPrimitives method

Reduce vertex count

Finally, we try to reduce the vertex count by removing vertices which are very close to each other and the ones which are of little or no importance for the shape of the primitive. This is done by calculating the Euclidean squared distance between a vertex and its preceding vertex and with the help of the Douglas-Peucker algorithm.

The Douglas-Peucker algorithm is an intensive algorithm and that is why we first reduce the vertex count with the help of the Euclidean distance. When two vertices are closer together than a threshold of 0,2% of the main diagonal of the image, one of them will be removed. After that the Douglas-Peucker algorithm will further reduce the number of vertices. The Douglas-Peucker algorithm reduces the vertex count by searching and removing all vertices which will not make a relevant contribution to the shape of the primitive. An example run of this algorithm is visible in figure 5.8. It first selects the first vertex of a primitive and the one furthest from the first. Then it recursively searches for the vertex which is furthest from the line which makes up the new primitive. If this distance is greater then a threshold value the vertex gets included into the primitive and the lines of the new primitive are updated. This goes on, until the distance to all the vertices are within the threshold and the new primitive is optimally simplified.

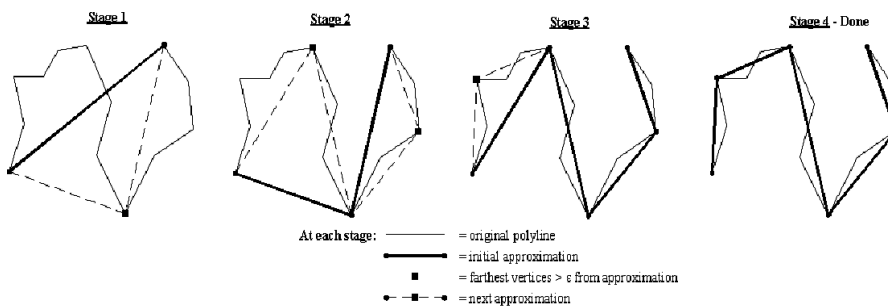


Figure 5.8: Optimization of a primitive with the Douglas-Peucker algorithm

5.5 Database

Our application needed a way to store heterogeneous data and retrieve it reliably. Not favouring the monolithic approach our predecessors took, a core design goal was to separate the data storage and retrieval from the real logic of the application. To achieve this all data access is done, using a level of indirection via the class called DatabaseConnection. This class's external interface consists of only those functions needed for storing and querying data. Internally it is free to implement the data storage in any way it pleases, as long as it satisfies persistency across multiple invocations.

We implemented the Database class used by the DatabaseConnection as a bunch of flat files in the file system. These files are identified by the integer id of the image to which the data in the file belongs. This solution is conceptually

very simple and the performance is just adequate for the number of images we are dealing with and the query loads posed to us.

We did not want to add the complexity of a real DataBase Management System (DBMS) when the file system sufficed. And we were all unfamiliar with using a DBMS for an application like this with very simple queries but with high throughput needs.

5.5.1 Database layout

The file system based database consists of one file in which data is stored for all images *topology.tree* and files in separate folders storing various data for each image.

Topology.tree is a file which contains the topological information for each image in the database. This file is actually a BerkeleyDB database file, written to disk by the class NBtree.

The other files are:

xx.original.txt is a text file containing the path to the original indexed image. Used when the system needs to return the original image to the user.

xx.geom.tree is a binary BerkeleyDB file containing the NB-tree of the geometrical descriptors.

xx.sketch.svg is an SVG file containing a simplified sketch representation of the indexed image. Used when the user wants the sketch of an existing image.

xx.thumb.png is a PNG thumbnail of the indexed image. Used for displaying small representations of candidate images.

5.5.2 DbBuilder

The dbBuilder is an application which uses the same libraries as the main application (meaning DatabaseConnection, SbrCore, DrawingLib, etc.). It makes all the calls needed to compute topological and geometrical descriptors (i.e. classify a drawing), image thumbnails and simplified sketches. It then stores the generated data in the database. As a result, the program does not have to be rerun until the Clip-Art collection changes.

Chapter 6

Evaluation

The project is not yet mature. This is mainly because of too little incentive for growth and a lack of market pressure. In short the project goal has implicitly been to the stage of a 'proof of concept' and that is as far as it got. What we offer currently is a robust and fully functional prototype using modern technology and a clear design.

6.1 Testing

Testing was mainly done in a subjective manner by clicking through the application manually. Results we perceived as being somewhat similar gave us an indication that the development of the server was proceeding in the right direction. This method of testing of course is very rudimentary and can hardly be considered indicative of anything in the system.

An improvement in testing was achieved after constructing a set of images with very simple and very different geometries and topological relations. In this set improvements in the algorithms we used were very visible. This was a good indication that the changes we were making into the system were headed in the right direction.

This form of automated testing though can never be indicative for real life usage, especially when the drawings contain a lot of topological information. During classification, multiple topology descriptors are calculated, one for each sub graph. This makes it possible for the system to retrieve an image showing a number of items when the input is only one of these items. When a query is made, only a single topology descriptor is computed for the received sketch.

Normally this works fine, because most users will only draw the part they consider interesting or important. In this form of testing, however, the input consists of an entire image, instead of the part a user would consider relevant. So when the system is queried, a drawing of a single car might return another drawing showing three cars. The drawing showing the three cars, on the other hand, is very unlikely to be matched with the drawing of just the single car.

High server load was simulated by running a Python script which continuously fired all possible requests to the server (in parallel if needed). Running this test gave us an indication of response time under heavy load (far heavier than was realistic actually). We used it to test if all the functions were working properly and that there were no stability, memory or other performance problems.

Finally we also needed to perform quantitative tests on precision and recall using various settings of the server.

Quantitative testing Another Python script tested the recall and precision values (on a result window of 10 images). For these tests it used a prepared set of images (see Appendix C) divided into –for a human observer– clearly visible classes. The script requested the sketch representation of each image from the server. These sketches then were returned to the server as queries. Depending on the results the server returned to each image the recall and precision were calculated and averaged for the entire test run.

Recall is the percentage of the possible correct images returned.

$$\text{This is } \frac{\text{correct images returned}}{\text{all correct images}}$$

Precision is the correctness percentage of the return results.

$$\text{This is } \frac{\text{correct images returned}}{\text{total number of images returned}}$$

Parameters These test runs were done on various parameters of the server and query settings to see which combination of settings returns the best result. The results of these tests are visible in figures 6.1 and 6.2.

The horizontal axis of both figures indicates the weight of how the query is evaluated between topology and geometry. A low value (to the left) means that only topology is used and no geometry. The higher the value the more influence is given to the geometric information.

Each of the series are specified as follows:

Series 1 Tested without adjacency weights and without geometry hint

Series 2 Tested with adjacency weights and without geometry hint

Series 3 Tested without adjacency weights and with geometry hint

Series 4 Tested with adjacency weights and with geometry hint

Series 5 The same as the previous one with one improvement.

We had made the mistake to put the geometry hint on the diagonal of the adjacency matrix. This put the results completely off. After feedback from our instructors we removed the values there and put them in the inclusion link of the adjacency matrix.

Series 6 The same as the previous one with one improvement.

We had forgotten to normalize some values. This also had a detrimental effect on the adjacency matrix we were told. These are the results after normalization.

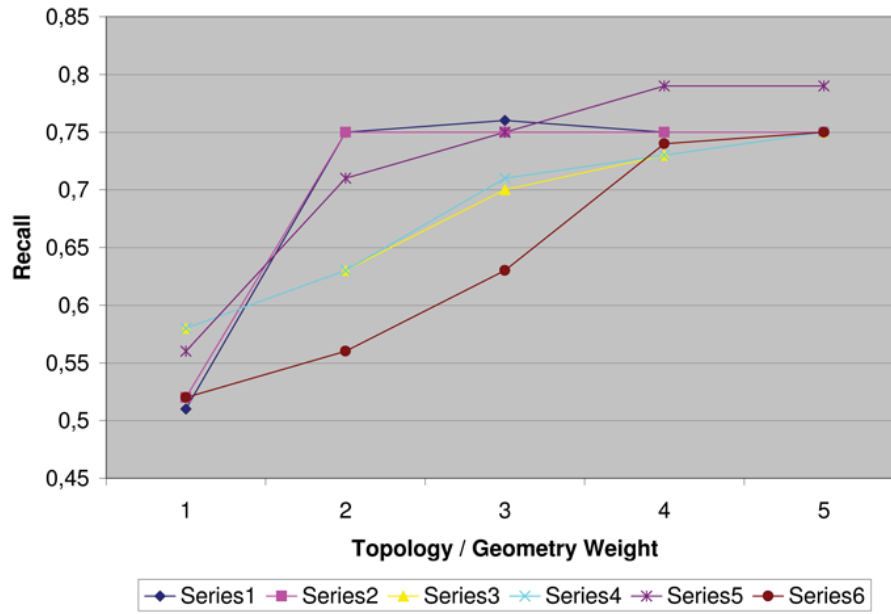


Figure 6.1: The Recall results of the tests

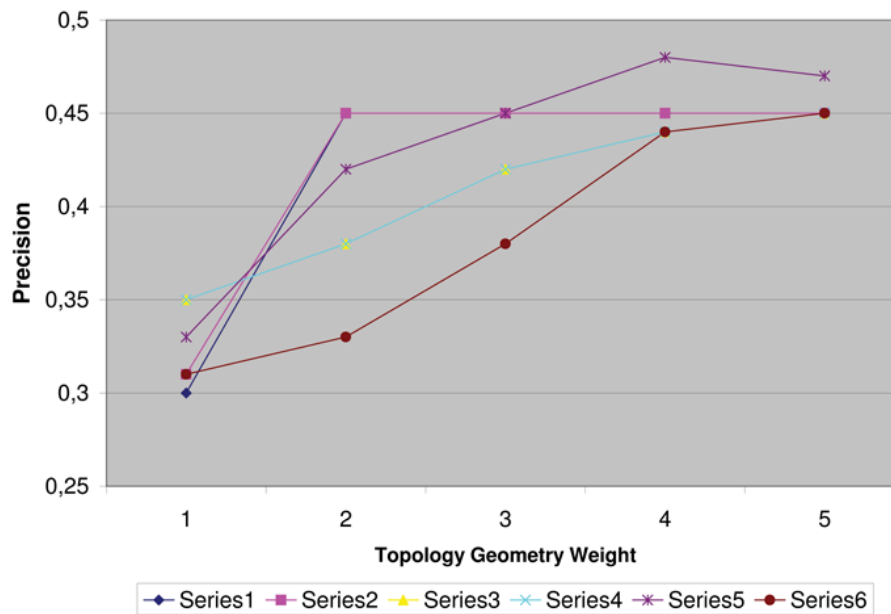


Figure 6.2: The Precision results of the tests

Result The results indicate that series 5 using query parameter 4 yields the best results.

This means that the final ‘improvement’ (in series 6) normalizing the values does not provide improvements. Using geometry hints and adjacency weights does provide better results. The query is passed a reasonably high value indicating that geometry and topology are both used to evaluate similarity.

6.2 Conclusions and Future Recommendations

6.2.1 Implementation

SVG We could include reading and classifying SVG images as well either by writing our own adapter to the Drawing class or by waiting until ImageMagick has better support for vector-based images.

Compression The trade-off between cpu-load and bandwidth could be examined more closely. For example currently flat XML is being sent between the client and the server with flat representations of the SVG image format. Either the SVG images could be compressed to become SVGZ images or the entire communications layer could be zipped (‘deflated’) at the HTTP layer. Both of these size reductions would cost the client and the server cpu-time. This could be desirable depending on the kind of load.

Database Implement the database layer to use a high-grade Database Management System. This might increase throughput provided the DBMS has a near optimally tuned cache.

The filesystem approach currently is fundamentally limited. Most filesystems have a strict boundary on the number of files they can store in a single directory. A DBMS would not have such problems – except maybe at numbers as high as 2^{32} .

If this system would be deployed using those quantities of images, another advantage of a professional DBMS would be its support for replication and clustering. Using this approach the system could be setup in a n -tier model and each tier could be replicated as much as needed.

Single-threaded The server’s request handling facility is currently single-threaded. This is because nobody had any experience in programming cross-platform multi-threaded C++ and we thought it would cost too much time to make that work.

Currently the server completely processes a request before it handles another one. If the load on the server is not too big it usually handles its requests fast enough and does not drop any new incoming requests. For increased loads this would have to change to a non-blocking approach using async-IO or multi-threading. This way new requests could always be accepted as fast as they come in. Processing would take more time but no requests would be dropped.

6.2.2 Methodology

The approach we currently use for image classifying and retrieving does work and returns relevant results. We feel however that the approach we are currently using would be more suited to CAD drawings. This is because most CAD drawings are simpler and topology is more important.

If the approach is not suited to the drawings, we could work the other way around and make the drawings suited to the approach. There is still a lot to gain by improving the simplification. Unfortunately most of the generic and to us obvious simplification heuristics have already been implemented. These heuristics do provide some simplification but do not go as far as to provide a pure sketchical representation of the original image.

From a human perspective these simplifications lack the quality of only keeping what is most important to a human. This human perspective could be inserted into the simplification routine by using tests and tuning with the help of an eye-tracker. Using the eye-tracker it would be possible to register what the user finds most important in a certain drawing. This information could be saved and used in some way to simplify the images to only save what a user finds important.

The user could also be educated –so to say– by giving more feedback. One idea is to display the returned results with in highlights those parts which have matched parts of the query image. This would tell the user how the system has interpreted his input.

6.3 Scalability and Extensibility

6.3.1 Scaling

During design, scalability was a constant concern. Many users connecting to the server at the same time can cause diminished performance. This is easily solved by creating a pool of servers each having a copy of the same data. The statelessness of the protocol makes it possible for each subsequent request to be sent to any server in the pool. A central access point can then distribute the requests to the servers balancing the load across the pool.

The scaling of the quantity of images and accompanying classification data is also possible but somewhat harder to do. The current file system based database is a major bottleneck because its performance requires repetitive hard disk access. That problem can be solved by reading all data into memory, but that option is not as scalable as desired. The best solution would be to rewrite the database layer of the application to make use of a real relational database (e.g. Oracle, PostgreSQL). This way the application can do what it needs to do while not worrying about data access and storage –which is handled by a special application. The database servers then can be tuned and replicated independently for increased performance.

6.3.2 Extensibility

The key of making it possible to extend this application in the future is modularity. We tried to make it as easy as possible to improve this application in the future, by building it from many independent modules. This makes it easy to extend libraries and thereby improve the functionality of the application.

An example of this is the way you can improve the number of image-formats which can be processed from a Clip-Art collection. The modularity of our application makes it possible to simply write a new sub-library capable of converting the desirable image-format to the internal Drawing format. You just have to include this new library to our application and thereby extend its capabilities, without having to worry about the simplification and comparison processes.

In some cases it might even be preferable to replace complete parts of the application. The current set-up of the application makes it possible to completely replace libraries without having to alter the other libraries.

One example of this has already been mentioned: the database integration. At the moment we use our own database, but the connection to the rest of the application is made by the `DataBaseConnection` class. This makes it possible to completely replace our database implementation with a wrapper to a real DMBS.

6.4 Process

6.4.1 Design

Our design work in this project consisted of two main stages in the first couple of weeks. These stages are summarized briefly in the following sections and the issues that rose are discussed.

Inventory Our design was started by taking a thorough inventory of the existing code and comments. The fact that a lot of the comments and even some of the code identifier names were written in Portuguese made this a painstaking process. The information extracted in this way was captured in UML class diagrams so it could be kept as reference material.

System design After the inventory the existing software had to be refactored into the application we wanted to build. This refactoring was preceded by an extensive design stage in which lots of UML diagrams were produced. A selection of these diagrams is included in this report (see [3.2](#) [3.3](#) [3.4](#) [3.5](#) [3.6](#)).

We adhered to the standard design methods taught to us during the course of our education. This is very briefly summarized as generating several documents and lots of diagrams in which all the intentions are encompassed. This formal design method may not have been optimally suited to this project. Producing and updating a lot of diagrams is a lot of work.

Some of the diagrams we made may have been made solely because it is customary to make them. A lot of the others certainly did have a use in capturing requirements and system layout for us. When the implementation is well

under way the diagrams are not used a lot anymore because by then everyone is mostly familiar with them.

A problem is that requirements may change during implementation. Updating all the diagrams to reflect the changes is usually a lot of work. Nobody is thinking about diagrams anyway by this stage. Because of these reasons they are usually not updated. The time spent on maintaining design documents would generally be better used attaining the goal of the project which is to produce software, not specifications.

6.4.2 Organization

Although the hierarchy between the people working at INESC is quite pronounced, the relationship between us and our supervisors was quite relaxed. They made it clear there was work to be done, but we were left free as to how we would accomplish this.

We chose a very informal method of cooperation without any form of hierarchy or function distribution, but most importantly without regular meetings, apart from the weekly meetings we had with our supervisors. If one of us had a problem or design issue, he would discuss this with a teammate. At irregular times, someone would worry about the overall progress and a meeting was called to discuss this.

This way of communicating had not been possible if we had been in separate rooms and this method of cooperation had not been possible if not all of us had felt a certain responsibility towards the success of the project.

The division of work was accomplished in much the same way. Everyone just picked an area of interest. Through informal communication redundancy was avoided.

6.4.3 Language

The decision to develop a major part of the language in C++ in retrospect turned out to be nothing shy of a small disaster.

The first problem was coping with the already existing code. This code was not set up in a modular fashion, responsibilities bled back and forth. The coding style was completely non standard keeping a bad middle between C and C++ and revealing a lack of understanding of the C++ language and the Standard Template Library. Various memory leaks were present which had to be ferreted out and solved. The installable application as we got it would not run out of the box but had to be tweaked. And finally running the results returned by the application were very poor on several points: performance, simplification and querying.

We started out with good intentions tearing this code base apart into functionally usable parts and developing the separate parts of the system. This went well enough even though we too were not very experienced in C++. We tried to make up for this lack of experience with a willingness to learn and do the right thing but this willingness slowly evaporated.

Being used to high level languages such as Java and Python, seeing how the simplest tasks had to be performed in the most elaborate and circumspect of

ways, discouraged some of us. The lack of features in the C++ standard library for file system path manipulation, advanced string processing etc. is baffling to programmers who want to get work done instead of implementing the same basic constructs over and over again.

Development proceeded slowly finding ways around these problems as they occurred. After some individual development the time came to integrate the separate parts into a whole. This integration process and especially the linking of several C++ libraries into a single application was so slow it seemed almost an impossible task. It did succeed finally but only after wasting about 3 person weeks on this task alone and simultaneously stunting development for the other team members.

Seeing the difficulties it caused would lead us to the conclusion that the C++ programming language is not suited for modern day programming unless there are some highly experienced developers part of the team. If we had realized this earlier we could have steered our development in one of two alternate directions:

Wrapping Wrap the existing code so it can be called from higher level languages. Java's JNI or Python's Boost facility would both be very suited to this approach. The success of this approach would be hampered in part by the obscurity and lack of modularity of the original code. To wrap code successfully the functionality being wrapped and its dependencies have to be clear.

Porting The successful port of CALI already demonstrated that the path of porting things from C++ to Java is wide open. Recent developments [Lewi] also make Java give C++ a run for its money performancewise. This means porting the entire code base to Java would have been a very viable option. Automating the process in part and adding human checking the pure port would maybe take the four of us a week. Another week then could be spent decomposing the code in functionally separate sub-parts and adding unit tests after which real development could begin.

Appendix A

Glossary

DRAWING A vector based image.

DBMS DataBase Management System

GEOMETRY Describes the shape of a primitive. It differentiates between a square and a circle for example.

GESTURE A movement made with the pen (when using a tablet) or with the mouse, indicating a command.

GUI Graphical User Interface

IDE Integrated Development Environment

JAR Java Archive

PNG Portable Network Graphics. An image format that uses lossy compression, very similar to the well known JPEG format. Its main advantage over JPEG is the added alpha channel which enables users to specify transparency.

PRIMITIVE Either a polygon or polyline.

STROKE The digital equivalent of a stroke made with a pencil on a paper.

SBR Sketch Based Retrieval. Retrieving vector images by comparing their topology and geometry to one another.

SVG Scalable Vector Graphics. An image format, defined by W3C. It was introduced as an open vector based format, primarily targeted for use on the Internet.

TOPOLOGY Describes the spatial relation between the primitives in a drawing. It does so by keeping track of inclusion and adjacency. If a primitive A encapsulates primitive B in its entirety it is said that primitive A includes primitive B. If A does not include B but if their borders do touch, A and B are said to be adjacent to each other.

WMF Windows Meta File. An image format using Windows GDI (Graphics Device Interface) calls to “paint” an image on screen.

Appendix B

Geometry features

Feature	Observation
$\frac{Perimeter_{ConvexHull}^2}{Area_{ConvexHull}}$	This feature measures the similarity to a circular shape.
$\frac{Height_{EnclosingRectangle}}{Width_{EnclosingRectangle}}$	The aspect ratio measures the thinness of shapes.
$\frac{Area_{ConvexHull}}{Area_{EnclosingRectangle}}$	This set of features measures the similarity to a rectangle.
$\frac{Perimeter_{ConvexHull}}{Perimeter_{EnclosingRectangle}}$	
$\frac{Area_{LargestQuadrilateral}}{Area_{EnclosingRectangle}}$	
$\frac{Area_{LargestQuadrilateral}}{Area_{ConvexHull}}$	
$\frac{Perimeter_{LargestQuadrilateral}}{Perimeter_{ConvexHull}}$	
$\frac{Area_{LargestTriangle}}{Area_{ConvexHull}}$	This set of features measures the similarity to a triangle.
$\frac{Perimeter_{LargestTriangle}}{Perimeter_{ConvexHull}}$	
$\frac{Area_{LargestTriangle}}{Area_{LargestQuadrilateral}}$	
$\frac{TotalLength_{Shape}}{Perimeter_{ConvexHull}}$	This feature measures the complexity of shapes, i.e. it measures the length of strokes within a limited area (convex hull).

Table B.1: The 11 geometry features of CALI

Appendix C

Images used for testing

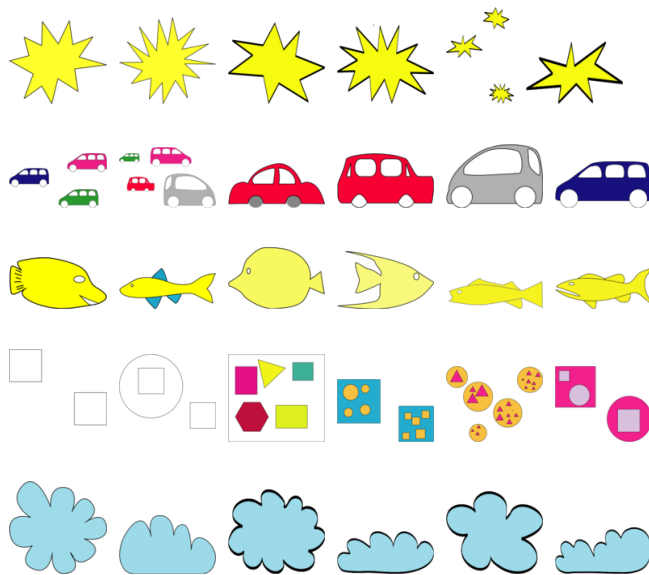


Figure C.1: Images used for automatic testing

Bibliography

- [Coll04] Ben Collins-Sussman, Brian W. Fitzpatrick & C. Michael Pilato: *Version Control with Subversion* (Copyright © 2000, 2001, 2002, 2003, 2004 CollabNet, Inc.)
- [Ecke00] Bruce Eckel: *Thinking in C++ 2nd Edition Volume 1* (<http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html>)
- [Ecke03] Bruce Eckel: *Thinking in C++ 2nd Edition Volume 2* (<http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html>)
- [Fons02] Fonseca M.J. and Jorge J.: *Towards Content-Based Retrieval of Technical Drawings through High-Dimensional Indexing* (Proceedings of the 1st Ibero-American Symposium in Computer Graphics (SIACG'02), pages 263-270, Guimares, Portugal, Jul 2002)
- [Fons03-1] Fonseca M.J. and Jorge J.: *Indexing High-Dimensional Data for Content-Based Retrieval in Large Databases* (Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA 2003), pages 267-274, Kyoto, Japan, Mar 2003)
- [Fons03-2] Fonseca M.J. and Jorge J.: *Indexing High-Dimensional Data for Content-Based Retrieval in Large Databases* (Technical Report, INESC-ID, IMMI, Jan 2003 (Extended version of DASFAA'03 paper))
- [Fons03-3] Fonseca M.J. and Jorge J.: *Towards Content-Based Retrieval of Technical Drawings through High-Dimensional Indexing* (Computers and Graphics, 27(1), pages 61-69, Jan 2003)
- [Fons04] Fonseca M.J., Barroso B., Ribeiro P. and Jorge J.: *Retrieving Vector Graphics Using Sketches* (<http://w5.cs.uni-sb.de/%7Ebutz/events/sg04/CRV/30310065.pdf>)
- [FSF98] *GNU Make Manual* (GNU Free Documentation License, Copyright © 2000 Free Software Foundation, Inc.)
- [Lewl] J.P.Lewis: *Performance of Java versus C++* (<http://www.idiom.com/zilla/Computer/javaCbenchmark.html>)
- [W3C03] *SOAP Version 1.2 Part 0: Primer* (W3C Recommendation 24 June 2003) (<http://www.w3.org/TR/soap12-part0/>)

- [Wine99] Dave Winer *XML-RPC Specification* (© Copyright 1998-2003 User-Land Software) <http://www.xmlrpc.com/spec>