



Institut Supérieur de Technologie
Department of Applied Informatics

Final Study Work

from 10 February 2003 to 6 June 2003

at

caritas
L U X E M B O U R G

Development of an N-tiers distributed application:



Date:	18.06.2003
Version:	1.0
Author:	Robert FISCH
Tutor IST:	Steffen ROTHKUGEL
Tutor CARITAS:	Robert URBÉ
Spreading:	Caritas <ul style="list-style-type: none">• M. Erny GILLEN• M. Robert URBÉ
	IST <ul style="list-style-type: none">• M. Steffen ROTHKUGEL• CDC• SEE

0. PREFACE

0.1. Acknowledgments

I want to thank all employees of Caritas Luxembourg who I worked with, particularly:

- M. Erny GILLEN, president of Caritas Luxembourg, who gave me the possibility of doing my final study work.
- M. Robert URBÉ, representative manager of the Caritas Foundation, who was my tutor all along this time.
- Mme Claudine GOERES, Mme Danièle HENGEN, Mme Lydie KRECKÉ, Mme Viviane RENCKENS, Mme Sandra SCHOLER, Mme Caroline THEVES, Mme Pascale THULL, M. Frank KAYL and everybody else who was very patient with me.

From the side of the IST I am particularly grateful to:

- M. Steffen ROTHKUGEL, professor at the IST, who was my tutor.

Special thanks go also to:

- M. Armand KOPCZINSKI, also student at the IST, who did his practical semester at Caritas Luxembourg.
- Mme Romy FISCH, Mme Danièle HENGEN, Mme Chantal WEIS and M. Didier CODEN, who helped me revising this document.

0.2. Summaries

0.2.1. Summary

As final study work, I had to develop a maintainable and scalable, easy to use distributed client-server application that integrates dynamic code download.

This document describes how I proceeded to analyse the needs and the requirements, the way I worked out an adequate architecture and why I made different technological choices. It also points out what security problems had to be solved. Another important aspect was the implementation of a user-friendly graphical interface. Finally I developed a Delphi client application, which communicates via XML-RPC with a Delphi server and downloads DLLs for the latter one. The server is connected to a MySQL database.

0.2.2. Kurzfassung

Als Abschlussarbeit hatte ich eine einfach zu verwaltende und skalierbare client-server Applikation mit dynamischem Code download zu entwerfen.

Dieses Dokument beschreibt, wie ich vorgeing um die Bedürfnisse und Anforderungen zu analysieren, eine angemessene Architektur heraus zu kristallisieren und warum ich welche technische Wahl traf. Es weist auch darauf hin welche Sicherheitsprobleme gelöst werden mussten. Ein anderer nicht zu vernachlässigender Teil, bestand in der Entwicklung einer benutzerfreundlichen grafischen Oberfläche. Schließlich entwickelte ich eine Delphi Clientapplikation, die via XML-RPC mit einem Delphi Server kommuniziert und von dort DLLs downloadet. Der Server seinerseits ist mit einer MySQL Datenbank verbunden.

0.2.3. Résumé

Lors de mon travail de fin d'études, j'avais à développer une application distribuée client-serveur maintenable, évolutive et intégrant une fonction de téléchargement de code dynamique.

Ce document décrit mon procédé pour analyser les besoins et contraintes, la manière dont j'ai mise au point l'architecture et pourquoi j'ai fait différents choix technologiques. Il explique aussi comment les problèmes au niveau de la sécurité ont été résolus. Un autre point important était l'implémentation d'une interface graphique simple et facile à manipuler par les utilisateurs. Finalement j'ai développé une application client dans Delphi qui communique avec un serveur, lui aussi écrit en Delphi, via le protocole XML-RPC et télécharge des DLLs de ce dernier. Le serveur de son côté est relié à une base de données MySQL.

Table of contents

0. PREFACE.....	2
0.1. Acknowledgments	2
0.2. Summaries	3
0.2.1. Summary	3
0.2.2. Kurzfassung.....	3
0.2.3. Résumé.....	3
1. INTRODUCTION	8
1.1. The Final Study Work	8
1.2. Caritas Foundation Luxembourg.....	8
1.2.1. Introduction.....	8
1.2.2. Flowchart	9
1.2.3. Composition.....	9
1.2.3.1. Managing committee of the Caritas Foundation	9
1.2.3.2. Managing committee of the Caritas Confederation	10
1.2.4. Location & Map	10
1.3. The project	11
2. DESCRIPTION	12
2.1. Objective	12
2.2. Requirements	12
2.3. Procedure	13
2.3.1. Interviews	13
2.3.2. Validation	13
3. ARCHITECTURE	14
3.1. General architecture.....	14
3.1.1. Methodology	15
3.1.2. How it should work	16
3.1.3. Multithreading & Sessions.....	16
3.2. Client architecture	17
3.2.1. The Module Container	17
3.2.2. The Modules	19
3.2.3. Client class diagram.....	20
3.3. Server architecture	21
3.3.1. CARiDAS Method Server	21
3.3.2. WWW Method Server	21
3.3.3. Server class diagram	22

4. SECURITY.....	23
4.1. Client side.....	23
4.2. Server side	23
4.3. XML-RPC protocol	23
4.3.1. Definitions	24
4.3.2. Scenario.....	25
4.3.3. Possible attacks.....	26
4.3.3.1. Man-in-the-middle (Mallory)	26
4.3.3.2. System-kick-off (Freddy).....	26
5. CLIENT MODULES.....	27
5.1. User manager	27
5.1.1. Description	27
5.1.2. Definitions	27
5.1.2.1. Right	27
5.1.2.2. User	27
5.1.2.3. Group	27
5.1.3. Functionalities.....	28
5.1.3.1. Right	28
5.1.3.2. User	28
5.1.3.3. Group	28
5.1.4. Database structure	29
5.1.5. Class diagram	29
5.2. Address manager.....	30
5.2.1. Description	30
5.2.2. Definitions	30
5.2.2.1. Entity.....	30
5.2.2.2. Groups.....	30
5.2.3. Functionalities.....	31
5.2.3.1. Entity.....	31
5.2.3.2. Groups.....	31
5.2.3.3. Titles & Functions	32
5.2.3.5. Printing	32
5.2.4. Database structure	33
5.2.5. Class diagram	34
5.3. File manager	35
5.3.1. Description	35
5.3.2. Definitions	35
5.3.2.1. Directory.....	35
5.3.2.2. File	35
5.3.3. Functionalities.....	36
5.3.3.1. Directory	36
5.3.3.2. File	36
5.3.4. Database structure	37
5.3.5. Class diagram	37
5.4. Article manager	38
5.4.1. Description	38
5.4.2. Definitions	38

5.4.2.1. Items.....	38
5.4.2.2. Types.....	38
5.4.2.3. Categories.....	39
5.4.2.4. Places.....	39
5.4.3. Functionalities.....	39
5.4.3.1. Items.....	39
5.4.3.2. Types.....	39
5.4.3.3. Categories.....	39
5.4.3.4. Places.....	39
5.4.4. Database structure.....	40
5.4.5. Class diagram.....	40
5.5. Photo manager.....	41
5.5.1. Description.....	41
5.5.2. Definitions.....	41
5.5.2.1. Photo.....	41
5.5.2.2. Categories.....	41
5.5.3. Functionalities.....	42
5.5.3.1. Photo.....	42
5.5.3.2. Categories.....	42
5.5.4. Database structure.....	43
5.5.5. Class diagram.....	43
5.6. Donation manager.....	44
5.6.1. Description.....	44
5.6.2. Definitions.....	45
5.6.2.1. Donation.....	45
5.6.2.2. Account.....	45
5.6.2.3. Motive.....	45
5.6.2.4. Destination.....	45
5.6.3. Functionalities.....	45
5.6.3.1. Donation.....	45
5.6.3.2. Account.....	45
5.6.3.3. Motive.....	45
5.6.3.4. Destination.....	45
5.6.3.5. Export.....	46
5.6.3.6. Printing.....	46
5.6.3.7. Reports.....	46
5.6.3.8. Statistics.....	47
5.6.4. Database structure.....	48
5.6.5. Class diagram.....	49
6. CONCLUSION.....	50
6.1. The project.....	50
6.2. General conclusion.....	50
7. REFERENCES & ACRONYMS.....	51
7.1. References.....	51
7.2. Acronyms.....	51
8. INDEX.....	52

Table of figures

Figure 1: Caritas @ Luxembourg City	10
Figure 2: CARiDAS, general application architecture	15
Figure 3: CARiDAS, general client architecture.....	17
Figure 4: CARiDAS client, module container with dynamic start menu	18
Figure 5: CARiDAS client, module download & load	19
Figure 6: CARiDAS, client module class diagram	20
Figure 7: CARiDAS, server class diagram	22
Figure 8: CARiDAS client, login window	24
Figure 9: CARiDAS client, security alerts.....	25
Figure 10: CARiDAS client, security alert.....	26
Figure 11: User manager, main screen	27
Figure 12: User manager, edit user	28
Figure 13: User manager, edit group	28
Figure 14: User manager, database structure	29
Figure 15: User manager, class diagram	29
Figure 16: Address manager, main screen	30
Figure 17: Address manager, edit entity	31
Figure 18: Address manager, edit title.....	32
Figure 19: Address manager, label printing	32
Figure 20: Address manager, database structure	33
Figure 21: Address manager, class diagram	34
Figure 22: File manager, main screen.....	35
Figure 23: File manager, add file	35
Figure 24: File manager, add directoy	36
Figure 25: File manager, delete file confirmation.....	36
Figure 26: File manager, database structure	37
Figure 27: File manager, class diagram	37
Figure 28: Article manager, main screen.....	38
Figure 29: Article manager, insert type.....	38
Figure 30: Article manager, edit item	39
Figure 31: Article manager, database structure	40
Figure 32: Article manager, class diagram	40
Figure 33: Photo manager, main screen.....	41
Figure 34: Photo manager, search dialog	42
Figure 35: Photo manager, big preview	42
Figure 36: Photo manager, database structure	43
Figure 37: Photo manager, class diagram	43
Figure 38: Donation manager, main screen with donation capture	44
Figure 39: Donation manager, search for donations	45
Figure 40: Donation manager, reports and listing page.....	46
Figure 41: Donation manager, dynamic query interface	47
Figure 42: Donation manager, output graph.....	47
Figure 43: Donation manager, database structure.....	48
Figure 44: Donation manager, class diagram.....	49

1. INTRODUCTION

This document describes the different experiences I had during my final study work at Caritas Luxembourg during the period from the 10th of February 2003 to the 6th of June 2003. Its major part is dedicated to the project I worked on, which is a distributed system for different purposes.

1.1. The Final Study Work

Lasting at least 15 weeks, the final study work finishes off the second cycle of applied informatics. It consists of a project for an information technology engineer and has to concentrate around the specialisation the student chose. As opposed to the practical semester, it enables the future engineer to practice the skills he acquired during his studies. A non-negligible part of his work should consist of research.

The final study work needs to be completed within the scope of a company project, either within a research and development project at the company itself or within a public or private structure, in case the student wants to go to graduated school and realise a 3rd cycle.

1.2. Caritas Foundation Luxembourg

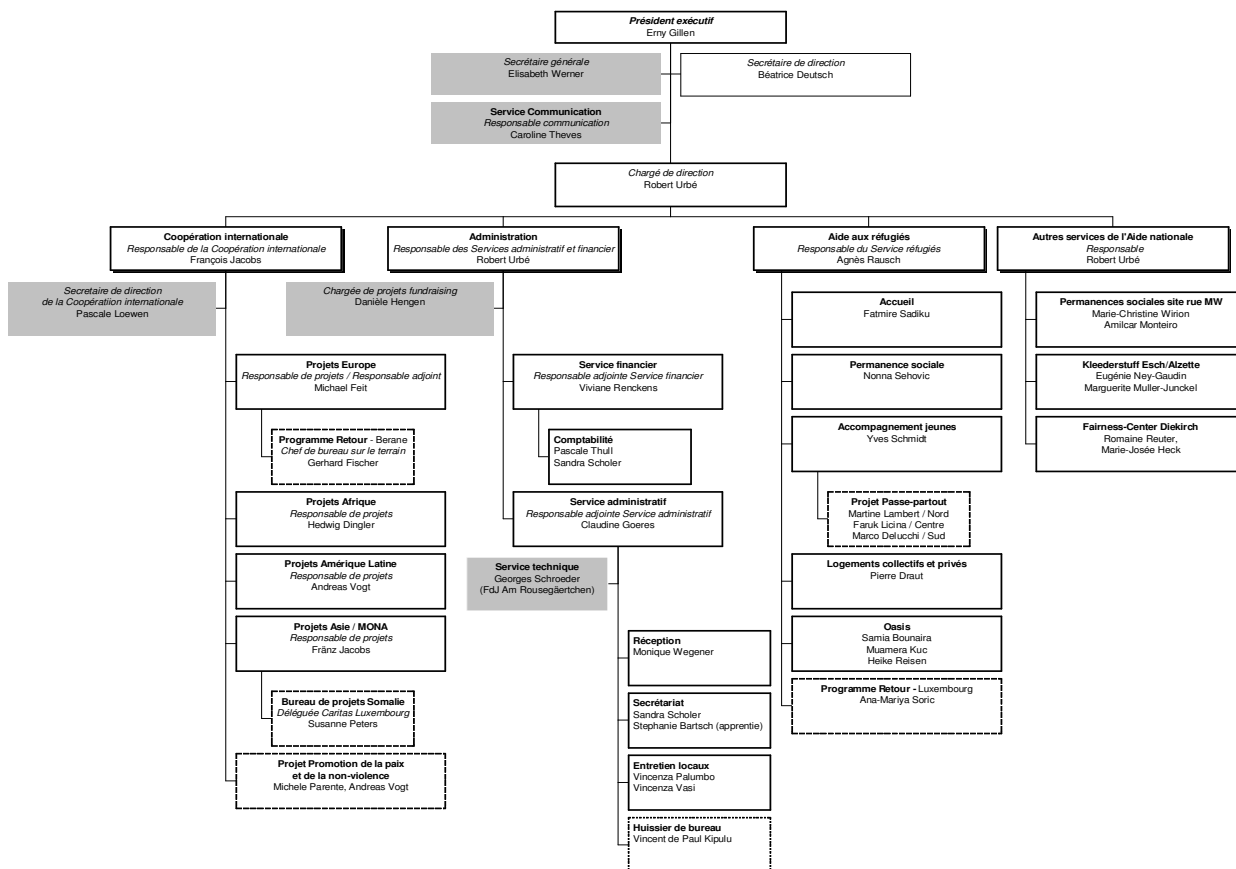
1.2.1. Introduction

The constitution of Caritas Luxembourg, as a federation of socio-caritative associations, goes back to 1932. Its activity, stopped during the German occupation, was devoted with the passing of years to the assistance of children, families, seniors, refugees and homeless people in the Grand Duchy of Luxembourg. After 1949 an international dimension, present in particular during the activities of the immediate post-war period, was confirmed with the establishing of Caritas Internationalis. This international confederation of which Caritas Luxembourg was co-founder, counts today 158 Caritas members working in 198 countries or areas all over the world.

The Confederation Caritas Luxembourg asbl obtained a new status in 1996 and gathers at the present time seventeen non-profit-making associations of Christian inspiration working in the social sector. Its aims is to represent its members as for their common interests like supporting and animating social work, including communication, synergies and innovative social work.

The Foundation Caritas Luxembourg, member of the Caritas Confederation, is particularly active in the assistance to the refugees and the urgent humanitarian aid and the development co-operation.

1.2.2. Flowchart



1.2.3. Composition

1.2.3.1. Managing committee of the Caritas Foundation

- Monseigneur Fernand Franck, Archevêque de Luxembourg, president
- Monsieur Erny Gillen, executive president
- Monsieur Ady Colas
- Monsieur Jean Kauffman
- Révérende Soeur Françoise-Elisabeth Scholtes
- Chanoine Mathias Schiltz
- Révérende Soeur Gemma Schmalen
- Monsieur Alphonse Wagner

1.2.3.2. Managing committee of the Caritas Confederation

- Monsieur Erny Gillen, president
- Monsieur Jean Audry
- Monsieur Ady Colas
- Monsieur Graziano Di Floriano
- Madame Ginette Faber
- Monsieur Henri Hamus
- Monsieur Ralph Hanck
- Monsieur Marc Hengen
- Monsieur Jean Kauffman
- Monsieur Michel Neyens
- Révérende Soeur Françoise-Elisabeth Scholtes
- Révérende Soeur Bertilla Schwalen
- Monsieur Alphonse Wagner
- Monsieur Victor Ziegler de Ziegleck
- Madame Suzette Zimmer-Maroldt
- Monsieur Romain Mauer
- Révérende Soeur Marie-Eugénie Knepper
- Révérende Soeur Emilienne Muller

1.2.4. Location & Map

Caritas Luxembourg
27, rue Michel-Welter
L-2730 Luxembourg

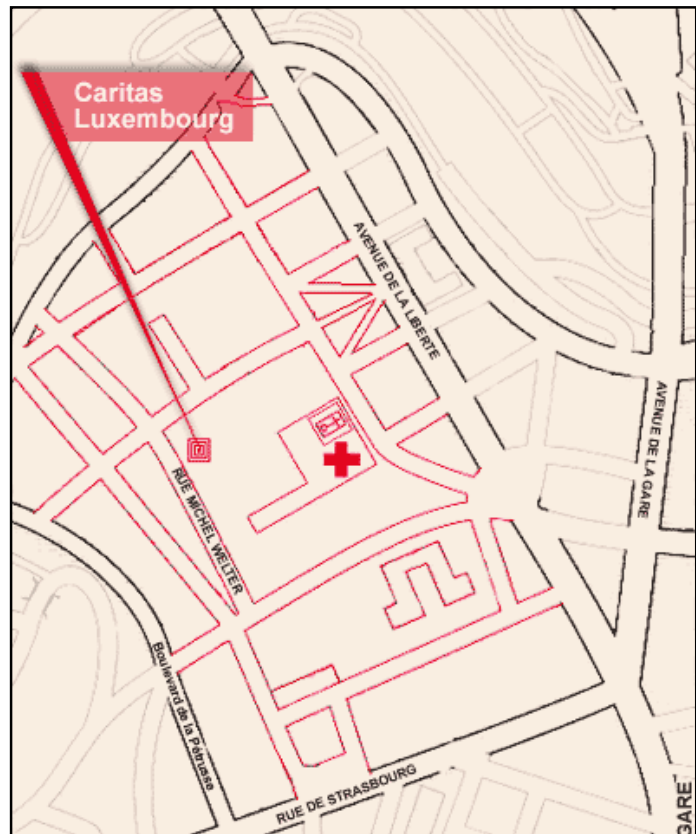


Figure 1: Caritas @ Luxembourg City

1.3. The project

The primary objective of the project was the development of a distributed client-server application which could be used for intern and extern management of different parts of the company.

I was in charge of analysing the needs of the organisation and work out a concrete application structure with high evaluative capacities.

The most challenging points to face were of course on one hand the dynamic code download part, which guarantees every connected client to possess the most recent software, and on the other hand the speed and performance as well as the response time the application should deliver.

2. DESCRIPTION

2.1. Objective

The primary objective of the project was the development of a distributed client-server application, capable of including dynamically different plug-ins, also referred to as modules in the following paragraphs.

The application had to be as dynamic as possible, in order to guarantee easy updates and evolution of its different parts. This is why a modular structure should be used, so that different modules could be easily added or modified at a later stage.

2.2. Requirements

From a technological point of view, the company did not have any requirements, but expressed a lot of other functional and non-functional needs:

- #1 First of all, compared to the existing system the company uses, the new application should pay attention to redundant data. Persistent entities should only exist once inside the system.
- #2 The new application should have improved speed and response time, as opposed to the existing application.
- #3 The client as well as server should run on the existing infrastructure of the company, which means Windows NT/2000/XP for the client and Windows 2000 for the server side.
- #4 Another very important aspect is the usability of the application. That is why it should be user-friendly, self-explaining and intuitive.
- #5 Finally, the development of the new application should be a low-cost solution, but without losing any performance.

2.3. Procedure

At this point I have to insert a personal note, which I think is very important. As known, I realized my final study work during the second semester of 2002/2003. During the first semester we had normal lectures at school, but, as a matter of fact, I already knew what project I would work on before school started.

I have to admit that I was unable not to think about this project during the first semester, that's why I tried to include most of our lectures in it, emphasizing the important points in order to improve the entire application.

2.3.1. Interviews

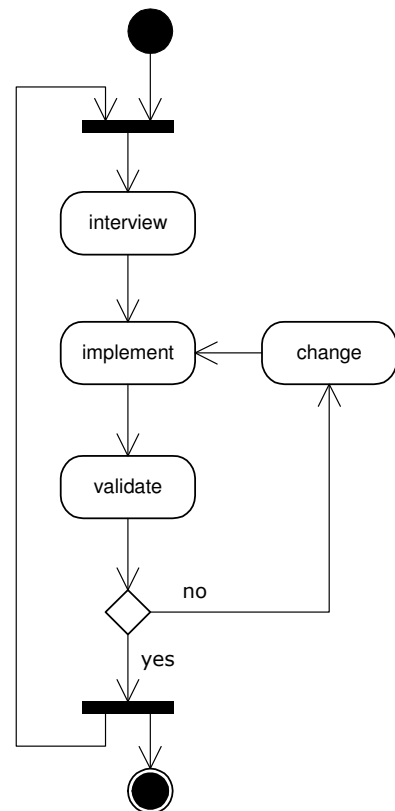
During many interviews with different employees of the company, I tried to determine their needs.

Some of these were large interviews, with a group of people, others small and quick ones. Due to the fact that the application was divided into different modules, I interviewed people all along the project.

2.3.2. Validation

During the whole project I absolved different mini-cycles, interviewing people, analysing system influences of the new needs, implementing a possible solution as a reusable prototype and having my work validated. One could say I did some kind of extreme programming.

This development cycle offered me also the advantage that I could freeze in a cycle for one module while I would work on another one. As a matter of fact, the employees of the company had to go on with their own work too, so they did not always have immediately time for me to answer my questions straight-away, but most of the time they put aside their work as soon as possible to help me design their new application.



3. ARCHITECTURE

3.1. General architecture

First of all, I had to design the general architecture, into which I could embed the modules. During this phase I already had to take decisions about what technologies to use. Especially trying to respect requirement #5 led me to the following choices:

- ▶ As the main program (client & server) should be really user-friendly (requirement #4) and do a lot of printing jobs, I decided to use Delphi 6 Personal Edition as development environment. This program is free for personal and non-commercial use.
- ▶ I decided to use a MySQL database, instead of MS-SQL, which is used by the existing application.

"The MySQL (TM) software delivers a very fast, multi-threaded, multi-user, and robust SQL¹ database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software. MySQL is a trademark of MySQL AB."²

Another advantage of using MySQL as main database server is that the Delphi application can connect to it via a native interface delivered directly by MySQL. So there would be no need to use MS-ODBC, which gives the system some more independency. Besides, MySQL can be downloaded for free.

- ▶ The last choice I had to make during this phase was about how to interconnect client and server. Here again I chose a free, extensible and open-source protocol: XML-RPC³. Beside the fact that it is really fast, XML-RPC implementations exist for nearly every development language⁴.

After this I assembled all information gained so far and obtained the architecture shown in the next figure.

Please note that at this point, the web part, developed by Armand Kopczinski during his practical semester, has been added later on. By connecting it to the rest, the architecture passed with success its first test in maintainability and scalability.

¹ Structured Query Language

² Description taken from the MySQL manual.

³ eXtensible Markup Language Remote Procedure Call

⁴ cf. <http://www.xmlrpc.org>

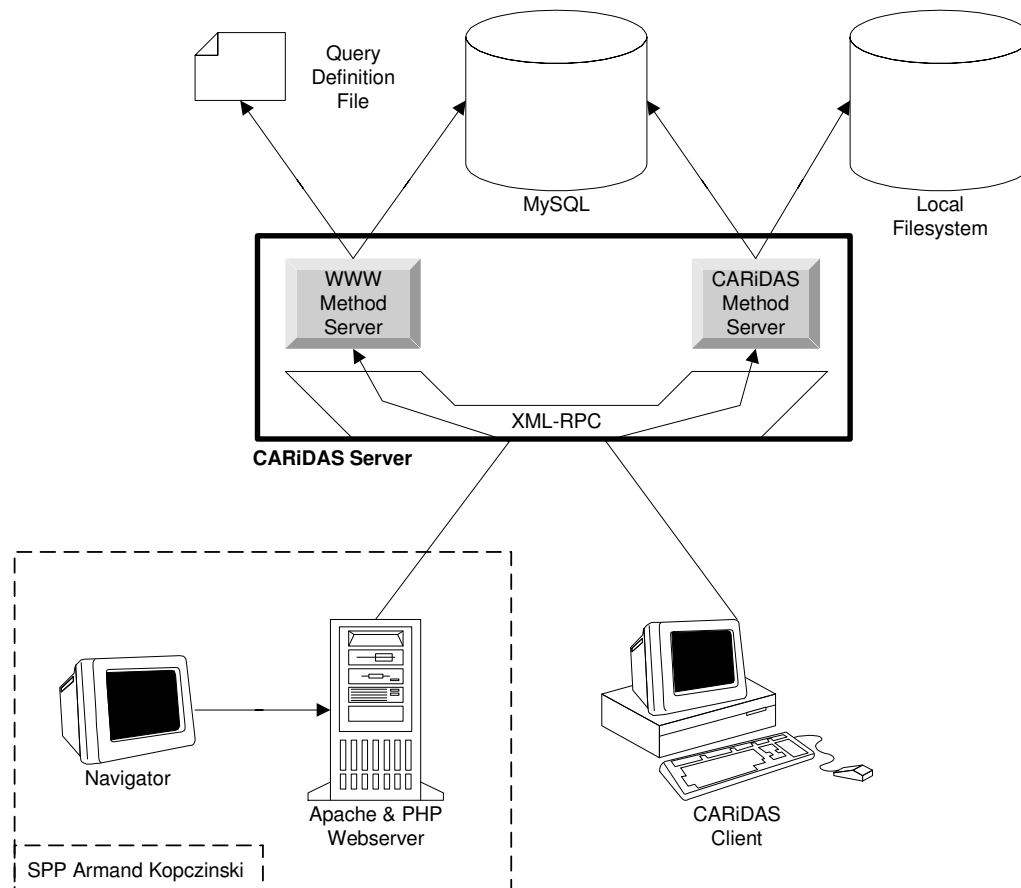


Figure 2: CARiDAS, general application architecture

During the next chapters I will go one step down and dive into the client and server applications. I will try to explain their respective inner working, especially the dynamic modules integration and the security aspects.

3.1.1. Methodology

From the moment when it was clear which way data would be sent between client and server, I did not bother about entities and how to make them persistent. I chose an object-oriented way for the functional parts of the client application and a document-oriented one for the data part. In fact, as I use the XML-RPC protocol to transmit data, each transmission can be interpreted as a document.

3.1.2. How it should work

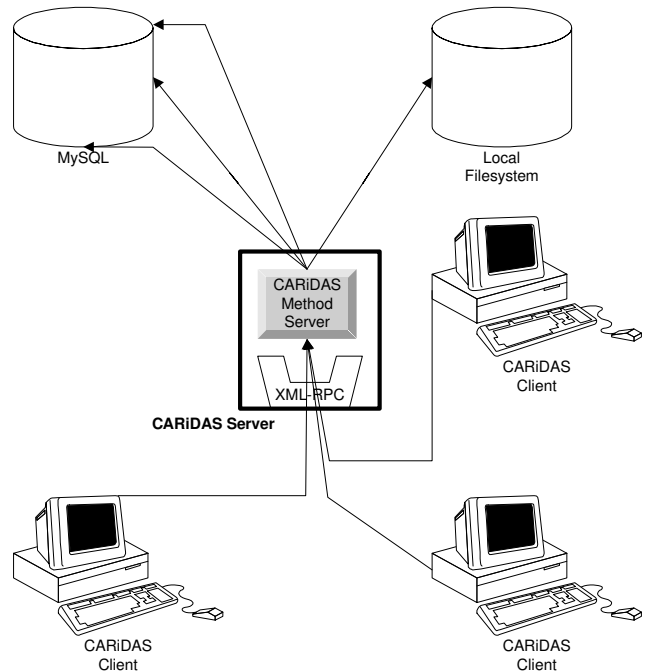
Let us suppose the database is up and the server is running too. When a client executes, first of all the user has to log onto the system, which means that he has to identify himself to the server. The user then enters his username and password and the client sends it to the server.

Within the server, his identity will be verified. Upon success, the server calculates the rights for the connected user and sends back to the client a list of all modules the user has access rights to. Once these modules are downloaded to the client, they will be integrated dynamically. For anything concerning security, please take a look at chapter four.

3.1.3. Multithreading & Sessions

What happens when there are multiple users who want to access the server at the same time? First of all, the XML-RPC server component implements a full featured multi-thread server, which means that multiple requests can be answered simultaneously.

As the database is also multi-threaded, the simplest way is to let it handle this itself, which is why the server opens a new database connection for each connected client. This also improves the query execution speed.



Security Protocol
XML-RPC
HTTP
Sockets

As mentioned before, the server opens a new database connection for each connected client. This implies that he has to maintain sessions. Because XML-RPC is a connection less protocol, it does not maintain any states. Beside this, the XML-RPC server component does not implement any kind of sessions, thus I had to do this job by myself. As a matter of fact the server maintains a list of all connected users. Problems about which session belongs to which user and how they are identified are solved by the security protocol I developed and which runs on top of the XML-RPC protocol.

3.2. Client architecture

The CARiDAS client, a Windows application written in Delphi, consists of two different parts:

1. The Module Container, which is the base application into which the modules (DLLs) are loaded.
2. The Modules themselves, which are containing the logic.

3.2.1. The Module Container

As shown in the figure below, the Module Container is the base application into which the modules (DLLs⁵) are loaded. Its primary function is to establish and handle a connection as well as the security protocol between the client and the server.

Depending on the user that logs on the system, the client downloads certain modules and loads them. In fact the client is an MDI⁶ parent form and each DLL contains an MDI child, which is loaded into the parent.

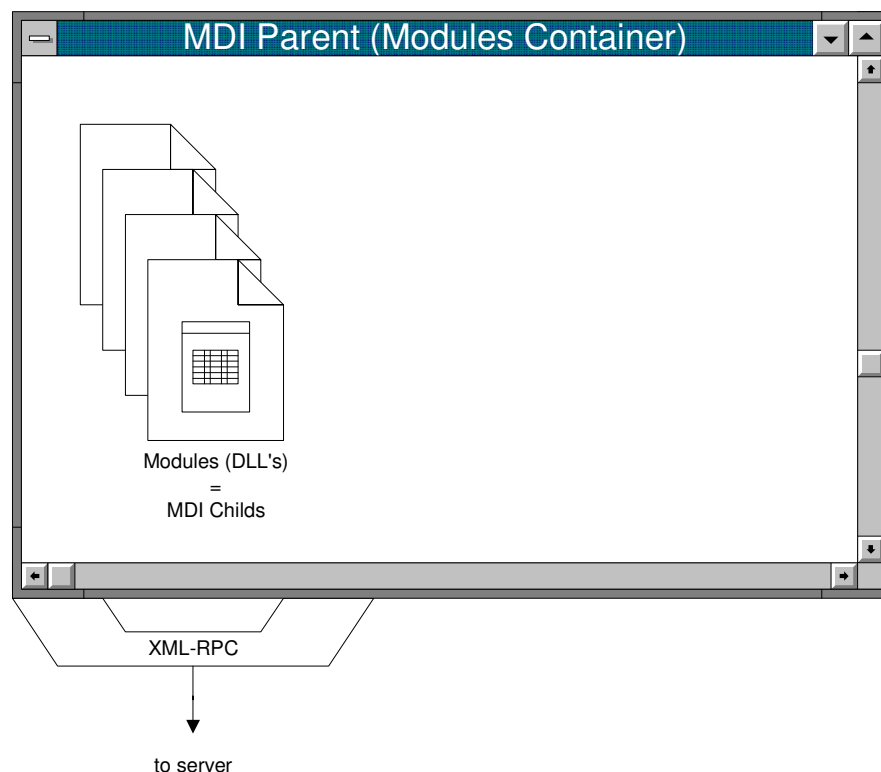


Figure 3: CARiDAS, general client architecture

⁵ Dynamic Linked Library

⁶ Multiple Document Interface

The MDI parent offers communication functions to the MDI children and handles most of the security checks.

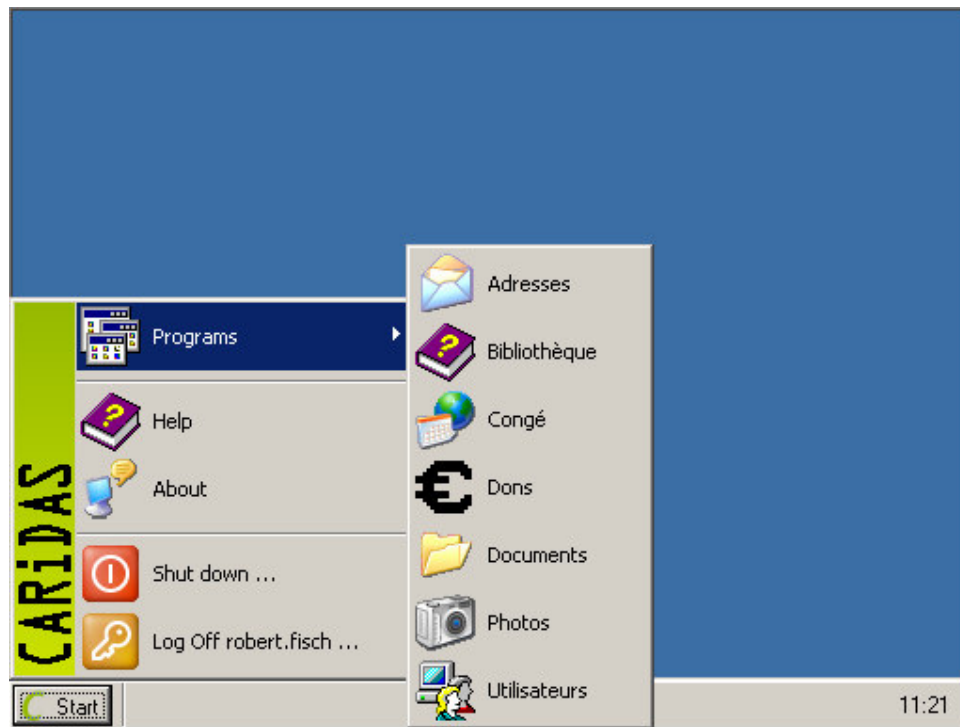


Figure 4: CARiDAS client, module container with dynamic start menu

Although this architecture seems to be quite simple, a lot of problems occurred during the phase of implementation. If loaded in a standard way, a loaded DLL is running in its own VCL⁷ environment, just the same way as the main application. In this case the main application and the different DLLs are built without runtime package, which means, that all code of any extra components is built into the executables.

At first I tried to pass system references from the main application into the DLLs. By doing this, I tried to join one or more VCL environments, but without success. The application compiled and loaded correctly, but while quitting, many memory errors occurred, which were related to the fact that the DLLs, as well as the main application, contained each one their own references to some common objects. When freeing up the memory, the DLLs first killed their objects, and then the main application tried again. As a result memory and null-pointer exceptions occurred. Worst of all, were the common system objects on which I had no influence.

So I had to try to make the main application and the DLLs run inside the same VCL environment. By compiling each part of the application with runtime packages I obtained the desired result. Runtime packages are external files

⁷ Visual Component Library

(.pbl), which contain the code for specific components. The fact that the main application and the DLLs are now executed inside the same VCL environment makes it possible to run the whole application exception-free. Another advantage of using external runtime packages is that, due to the fact that some code is put in common in these external files, the DLLs' size has been reduced significantly. This increases the download speed while loading them from the server.

3.2.2. The Modules

First of all, I want like to define the term module: a module is a DLL file where an MDI child is stored inside, which contains presentation and some business logic.

The DLL file itself only exports a set of functions and can be considered as a kind of wrapper around the MDI child. This means that all my modules have to export exactly the same functions. Actually, I do not think that it is possible to make inherit a DLL file from some ancestor.

One of the exported DLL functions, called by the base application, will add a new MDI child to it. How does this work? In fact, every module must follow some basic rules and must be known in a certain way to the base application. This explains why I defined a base module "BMDIChild", which is an abstract class. Any other module inherits from it and implements its own specialization (cf. figure on the next page). Going this way, all the base application sees, is a list of different MDI child windows which it can access via the methods and attributes defined by the abstract class.

The procedure is the following: The base application establishes a connection to the server and then loads a certain number of modules. Every loaded module has to be initialized with some values and references. This is done by passing a "BPluginVector" to every module. References to the modules themselves are held by the base application in a dynamic array.



Figure 5: CARiDAS client, module download & load

3.2.3. Client class diagram

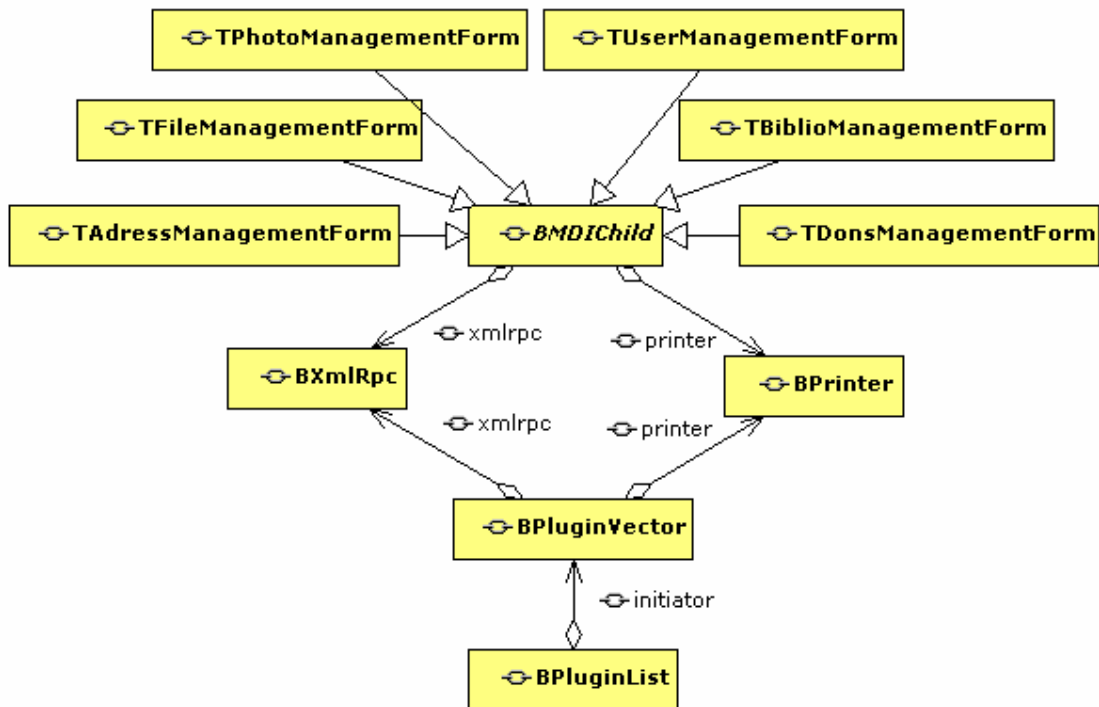


Figure 6: CARiDAS, client module class diagram

BXmlRpc is the common communication class, used by the application and all modules.

BPrinter offers a centralized printing management, but is not yet used by any modules.

3.3. Server architecture

As already shown in figure 2, the server consists of two different parts. Each of them is a fully functional XML-RPC server and runs on a distinct port. The main differences between them are explained below.

3.3.1. CARiDAS Method Server

This system serves methods to the client. Every call is secured and the user who requests it has to identify himself. A special security protocol (cf. chapter 4.3) makes this possible.

All of this system's methods are hard coded, by which I just want to point out that the server has to be recompiled in case one method changes or another one needs to be added. Methods are grouped by functionality inside method managers. Each manager inherits from an abstract manager class "BManager" which defines the minimal attributes and methods a manager needs in order to be incorporated into the XML-RPC server.

3.3.2. WWW Method Server

The WWW Method Server does not contain any inbound methods. All of them are based upon SQL queries, input and output parameters which are read directly from an external XML file upon server start-up.

As this part serves a PHP web interface, none of the exposed methods are secured, which means that anyone can execute them without identification. The fact that they are not secured at all may seem a bit too risky and is not fully correct. As a matter of fact, the server allows only SQL "select" statements to be executed. All other statements, which could change data, are rejected automatically by the server while reading the input file.

3.3.3. Server class diagram

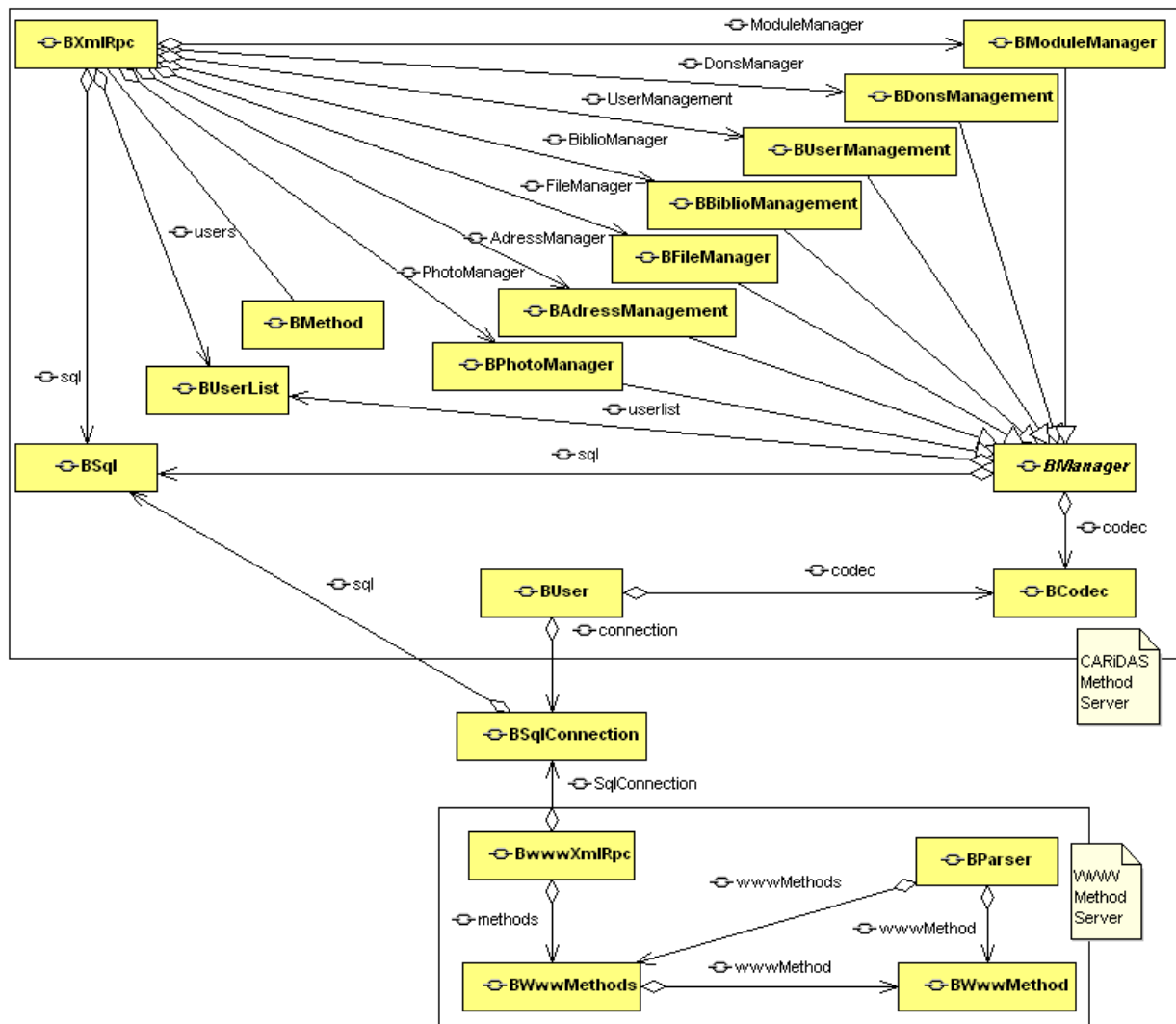


Figure 7: CARiDAS, server class diagram

4. SECURITY

In order to guarantee maximum security, CARiDAS implements security checks at three different levels:

4.1. Client side

Every user owns different rights. Depending on the rights the user has, the client application offers or denies access to different functionalities. It may also completely hide some of its parts, depending on the user who logs onto the system.

4.2. Server side

The security provided by the server side does not differ much from the client side one. Thus, depending on the rights the user has, the server side starts or denies execution of the requested method.

4.3. XML-RPC protocol

As XML-RPC is a "connection less"⁸ protocol, I had to implement a special security protocol in order to guarantee that no sensible data is transmitted in clear text over the connection and that the client users do not have to identify themselves to server each time they request a method to be executed. The last reason was the major one while conceiving the security protocol.

⁸ A connection less protocol is a protocol which does not use a persistent connection. This means that for every request a new connection is being established and closed after transmission of the response.

4.3.1. Definitions

Before describing in detail the scenario of how the client establishes a secure connection with the server, the different elements that will occur are enumerated below:

- username: Each user has a unique username (usr) which is being used to identify the user. The username is known by the user and is also stored inside the database.
- password: The password (pwd) is the secret key that is only known to user. Only its hash is being stored inside the database.
- session key: Each time a user connects to the server, a new session key (SK) is generated. This session key will be transmitted to the client who may then use it.
- session ID: Each time a user connects to the server, a unique session ID (SID) is being generated. It is used by the server in order to quickly find a connected user.
- chip: The chip (C_x) is a random key which is generated each time a user executes a method. The new chip is sent to the client. If the client does not possess this chip at his next request, he will be kicked out of the system. Not possessing the chip at the next request means that something went wrong or that someone tried to break into the system.

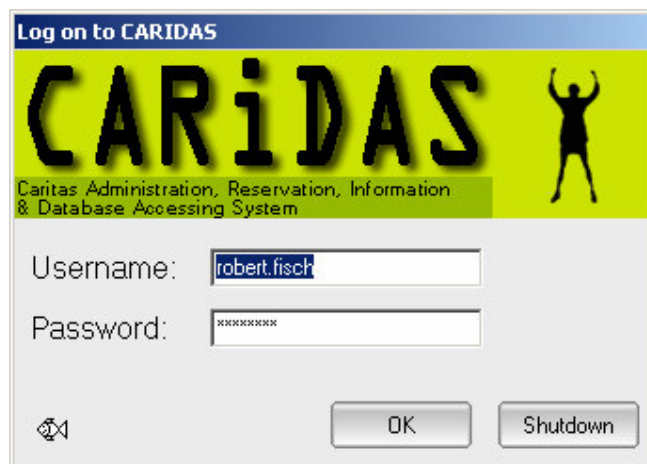


Figure 8: CARiDAS client, login window

4.3.2. Scenario

Bob (client)

(server) Alice

Bob first announces himself

---| announce (usr) |--->

Alice receives the request,
looks up whether 'bob' is an
existing user, retrieves the hash
of his password, generates a
session key, a SID and a chip C_0 .

<---| $\{SID\}_{[pwd]}$, $\{C_0\}_{[pwd]}$, $\{SK\}_{[pwd]}$ |---

As Bob is the only person who
knows his password, he now can
decode the content of Alice's response.
He now holds every element to login.

---| login(usr, SID, $\{C_0\}_{[SK]}$) |--->

Upon this login request Alice verifies
that a given SID exists and is bound
to the given username. After this she
can verify the sent chip. At this moment
Alice considers the user as being logged.

<---| $\{C_1\}_{[pwd]}$ |---

If everything is OK,
Bob receives a new chip
and is now logged on.

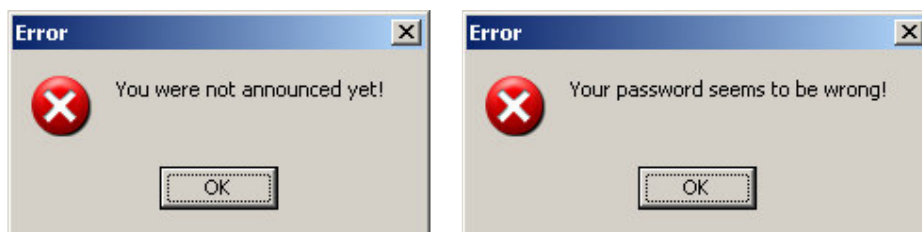


Figure 9: CARiDAS client, security alerts

4.3.3. Possible attacks

4.3.3.1. Man-in-the-middle (Mallory)

Except the case where Mallory knows the password of the client he wants to impersonate, the man-in-the-middle attack will not work because, Mallory cannot decode neither the session key nor the chip sent by Alice.

This is why he could only relay communication, but he can still catch or modify data sent as clear text.

4.3.3.2. System-kick-off (Freddy)

During this attack Freddy tries to execute a method on the server, but provides a wrong chip. As a result the attacked user is kicked out of the system and has to log on again before continuing to work.

In order to perform this attack, Freddy has to know the username and the actual SID of the user he wants to attack. Even though the SID is a random generated key, this is not very hard to find out, because username as well as SID are sent as clear text to the server. Even if they were encrypted, Freddy could simply copy them and resend a request containing a false chip.

Because multiple logins by the same user are permitted, username and SID have to be sent as clear text to the server in order that the latter is able to get the right session key to decode the chip.

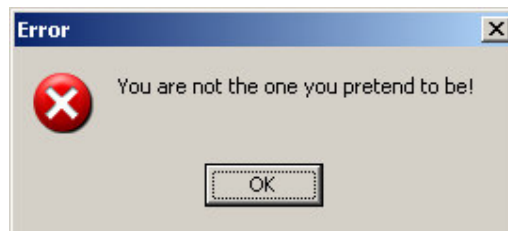


Figure 10: CARiDAS client, security alert

5. CLIENT MODULES

Before reading the next pages, I have to point out that some functionality is really straight forward. In fact this applies to all "dictionaries", which are tables storing data used over and over again inside the module(s). Users have the possibility to add, edit or remove items from these dictionaries.

5.1. User manager

5.1.1. Description

The user management module performs all the tasks related to the management of rights, users and groups.

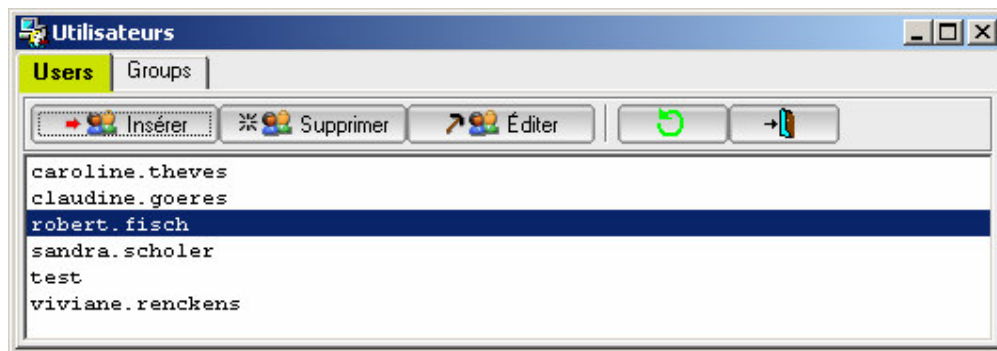


Figure 11: User manager, main screen

5.1.2. Definitions

5.1.2.1. Right

A right is a special entry, related to one or more modules. Possessing or not possessing a right may result in gaining or being denied access to different functionalities of the related modules. Rights are possessed by a user or by a group. Because rights are always system related, they cannot be modified by a system administrator.

5.1.2.2. User

A user is an internal representation of an external actor of the system. He is defined by his username, password and a collection of rights. Users may be part of one or more groups. Its collection of rights is defined by the union of the users own rights with the rights of all the groups he is part of.

5.1.2.3. Group

A group is an ensemble of distinct users. A group can have one or more rights.

5.1.3. Functionalities

5.1.3.1. Right

Because a right is always system or module related, it can not be modified in any way by a system administrator.

5.1.3.2. User

Users can be freely inserted, edited or deleted by a system administrator. They can be added to or removed from a group. They can also obtain different rights.



Figure 12: User manager, edit user

5.1.3.3. Group

Groups can be inserted, edited or delete freely by a system administrator. They can obtain different rights and users can be added or removed from them.

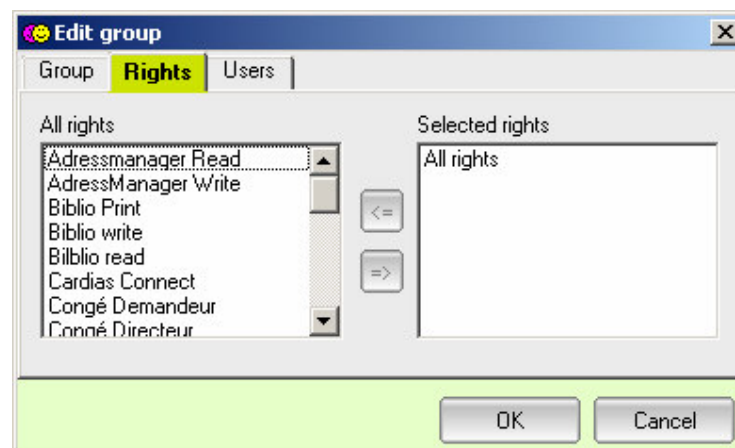


Figure 13: User manager, edit group

5.1.4. Database structure

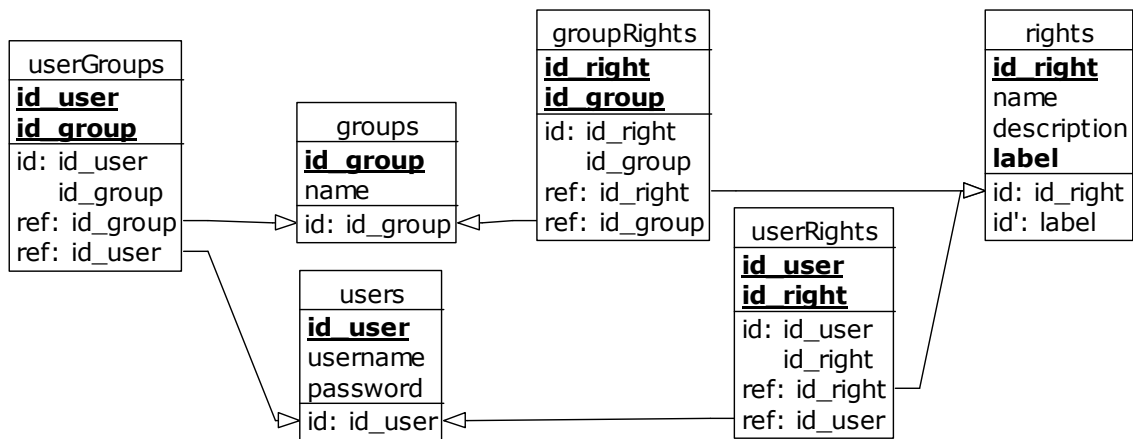


Figure 14: User manager, database structure

5.1.5. Class diagram

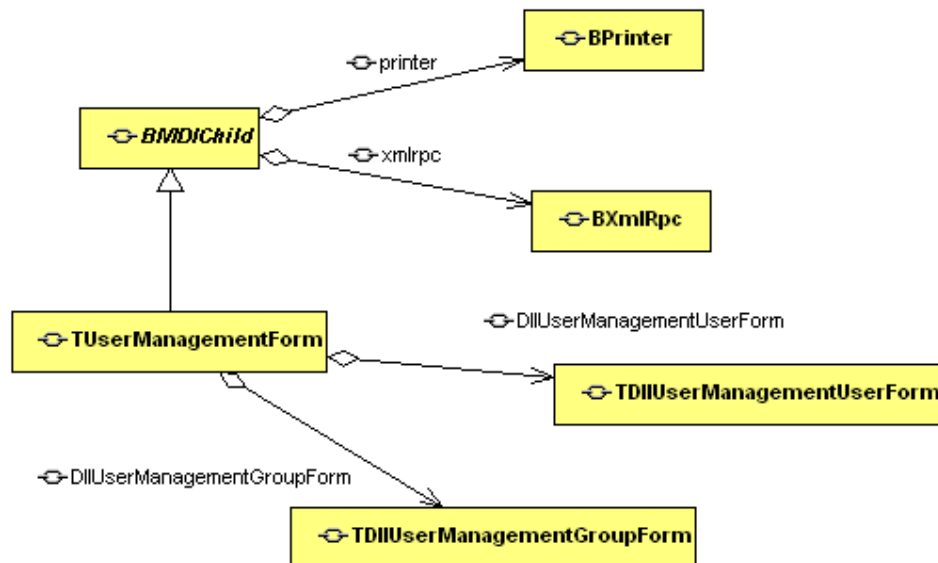


Figure 15: User manager, class diagram

5.2. Address manager

5.2.1. Description

The address management module offers all functionalities related to manage addresses for different purposes.

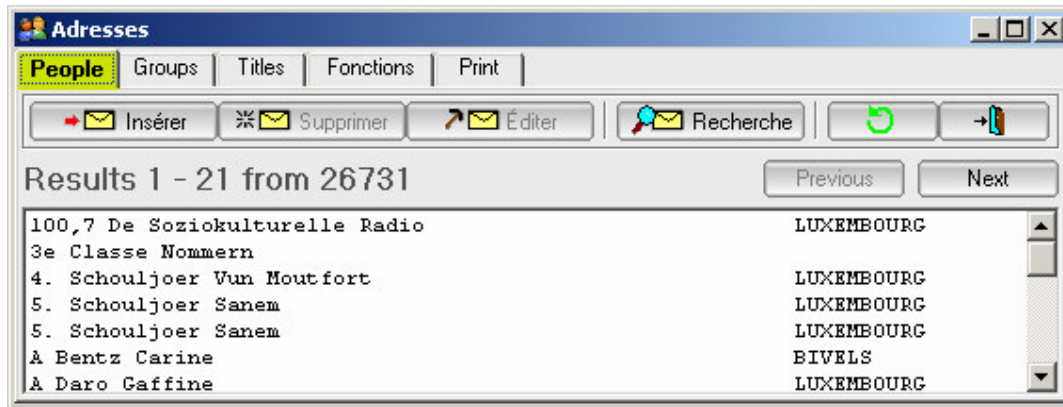


Figure 16: Address manager, main screen

5.2.2. Definitions

5.2.2.1. Entity

An entity can be either a physical person or an organization. Every entity has an address and a certain number of different attributes.

5.2.2.2. Groups

Entities can be regrouped. There are two different types of groups:

- User-defined groups
These groups can freely be inserted, edited or deleted by an end user who has the respective right.
- System groups
Contrary to the user-defined groups, system groups cannot be modified by any person. The logic of different modules may depend on the existence of a system group.

5.2.3. Functionalities

5.2.3.1. Entity

A user with the respective rights has the possibility to insert, edit or delete an entity. Each entry can be added to one or more groups. The user may also search for entities.

The screenshot shows a software window titled "Edit address" with a yellow smiley icon and a close button. It has two tabs: "Adresse" (selected) and "Groups".

Contact section:

- Titre: A dropdown menu showing "Monsieur".
- Nom: A text input field with a light green background.
- Prénom: A text input field containing "Bob".
- C/O: A text input field.
- Fonction: A text input field containing ".....".
- Entreprise: A text input field.

Adresse section:

- BP: A text input field.
- N°: A text input field.
- CP: A text input field with a light green background, followed by a right-pointing arrow.
- CC: A text input field containing "L".
- Rue: A dropdown menu.
- Localité: A dropdown menu with a light green background.
- Pays: A dropdown menu showing "Luxembourg".

An "Error" dialog box is overlaid on the "Edit address" window. It has a red "X" icon and the text: "You forgot to fill in some required fields!". It has an "OK" button.

At the bottom of the "Edit address" window, there are "OK" and "Cancel" buttons.

Figure 17: Address manager, edit entity

5.2.3.2. Groups

As already mentioned before only user-defined groups can be managed through the use of this module.

5.2.3.3. Titles & Functions

In order to maintain a constant naming scheme, titles and functions are stored separately. The graphical interface allows the inserting, editing and deleting of titles and functions.

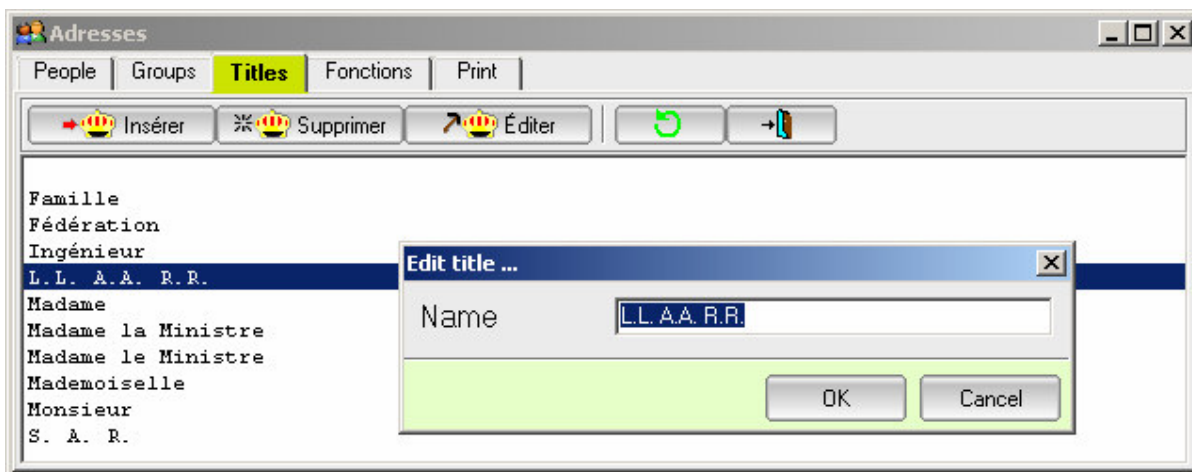


Figure 18: Address manager, edit title

5.2.3.5. Printing

This module allows printing address labels for the content of one or more groups.

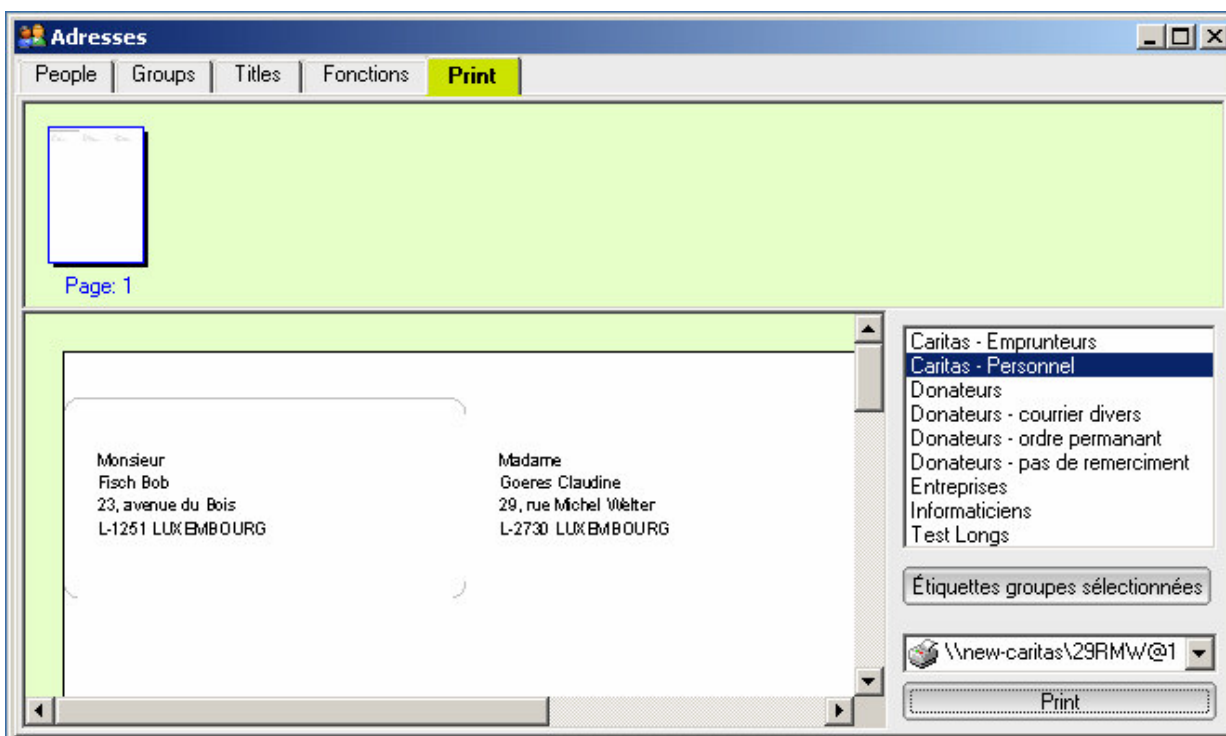


Figure 19: Address manager, label printing

5.2.4. Database structure

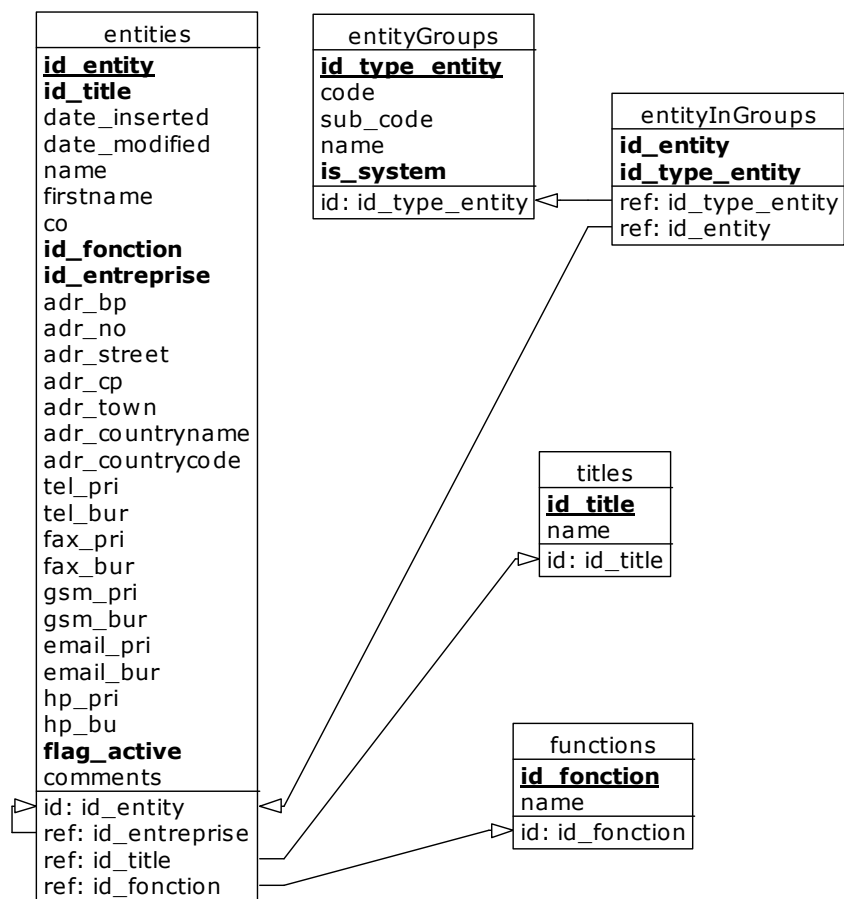


Figure 20: Address manager, database structure

5.2.5. Class diagram

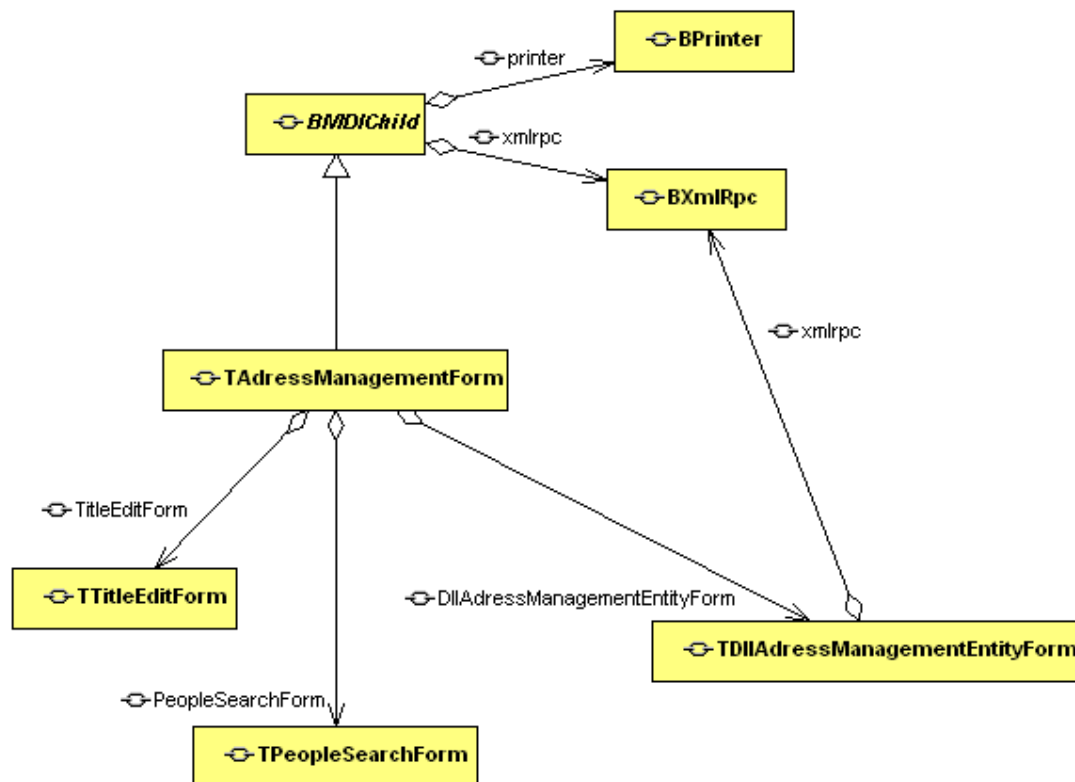


Figure 21: Address manager, class diagram

5.3. File manager

5.3.1. Description

The file management module offers the possibility to store files hierarchically on the server.

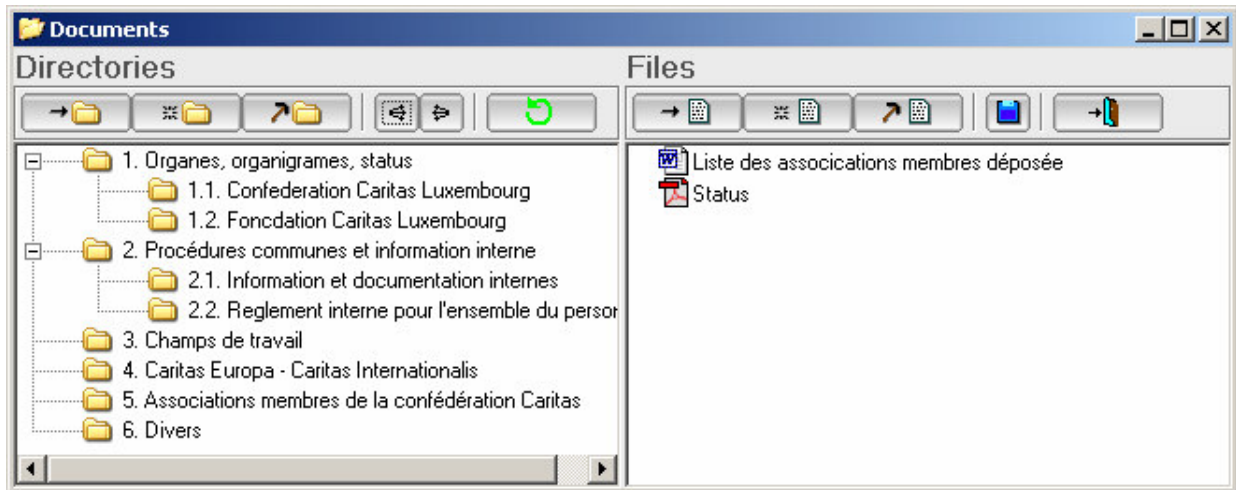


Figure 22: File manager, main screen

5.3.2. Definitions

5.3.2.1. Directory

A directory can contain other directories and files. It is either a root directory or it is contained inside another directory.

5.3.2.2. File

This can be any type of physical file. Files are contained inside directories.

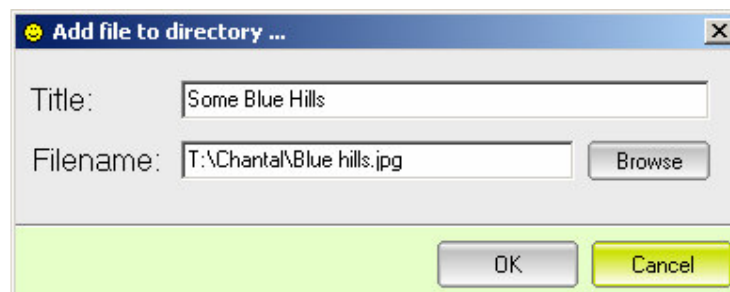


Figure 23: File manager, add file

5.3.3. Functionalities

5.3.3.1. Directory

Depending on their rights, users can add, rename or delete a directory. A non-empty directory cannot be removed.

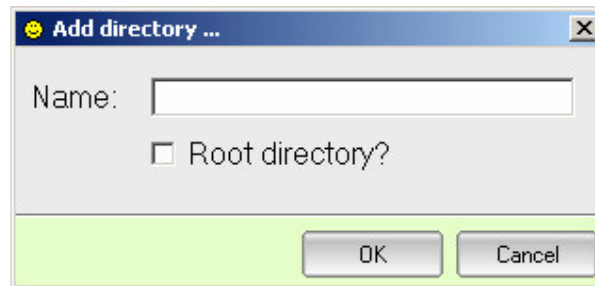


Figure 24: File manager, add directoy

5.3.3.2. File

In addition to the user's possibility to add, rename or delete files. They may also download them to their local disk.

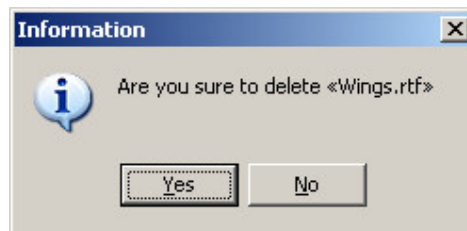


Figure 25: File manager, delete file confirmation

5.3.4. Database structure

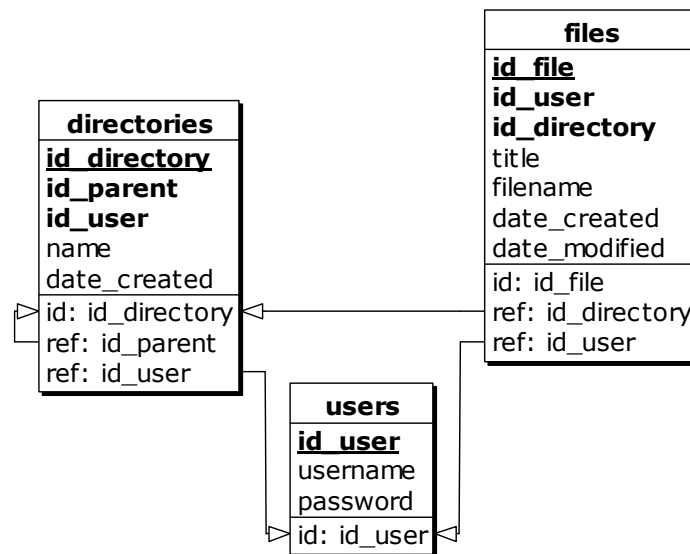


Figure 26: File manager, database structure

5.3.5. Class diagram

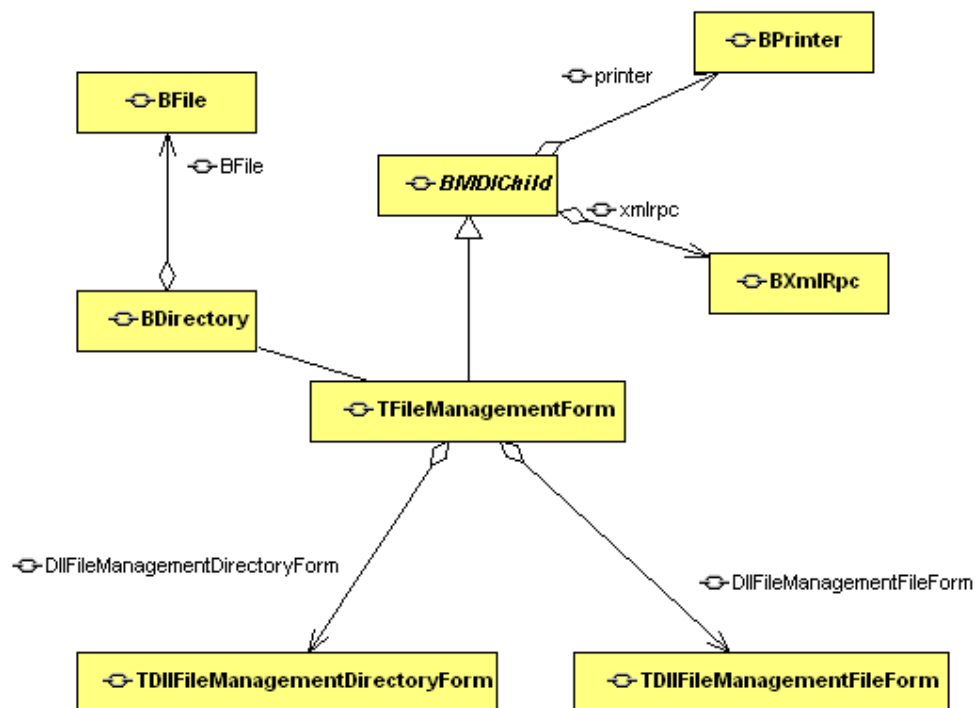


Figure 27: File manager, class diagram

5.4. Article manager

5.4.1. Description

The article management module offers the possibility to manage different articles, ordered into categories.

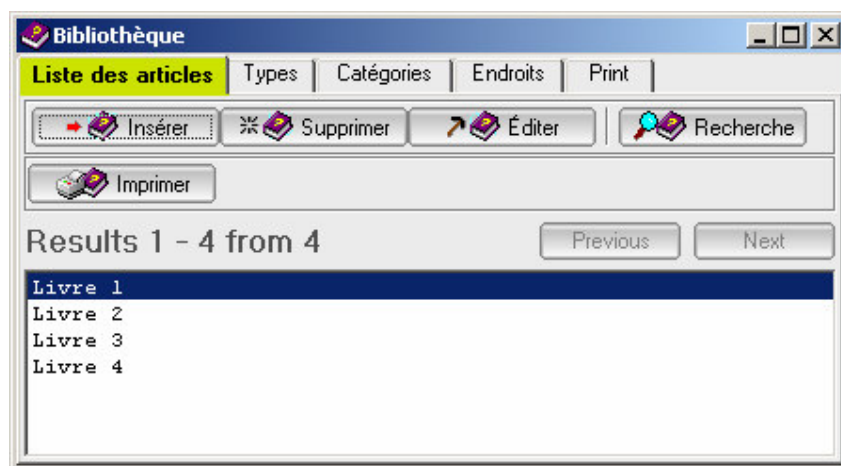


Figure 28: Article manager, main screen

5.4.2. Definitions

5.4.2.1. Items

An item represents a physical article with all its attributes. It is of a certain type and stored in a certain place.

5.4.2.2. Types

Each item is of a certain type (CD, DVD, Book, Journal ...).

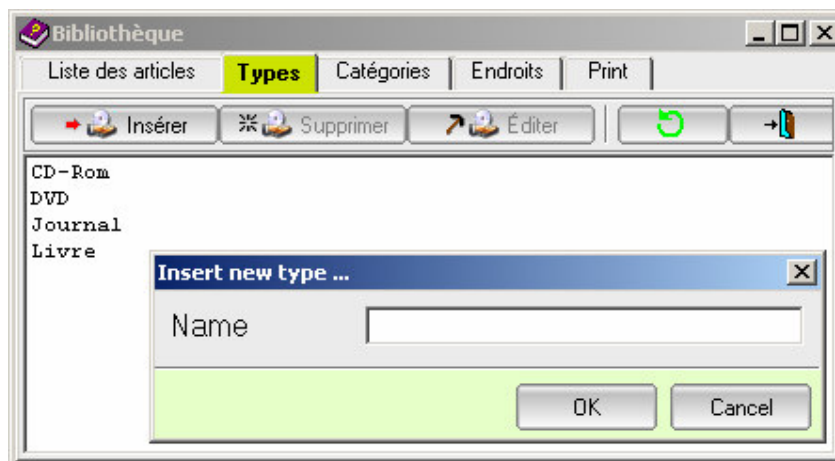


Figure 29: Article manager, insert type

5.4.2.3. Categories

Each item is held inside a category.

5.4.2.4. Places

This represents the physical place, office or bookshelf, where the article is usually stored inside the company.

5.4.3. Functionalities

5.4.3.1. Items

The users can add, edit or remove items. For each item the module offers also to print an overview sheet. The user may also search for articles.

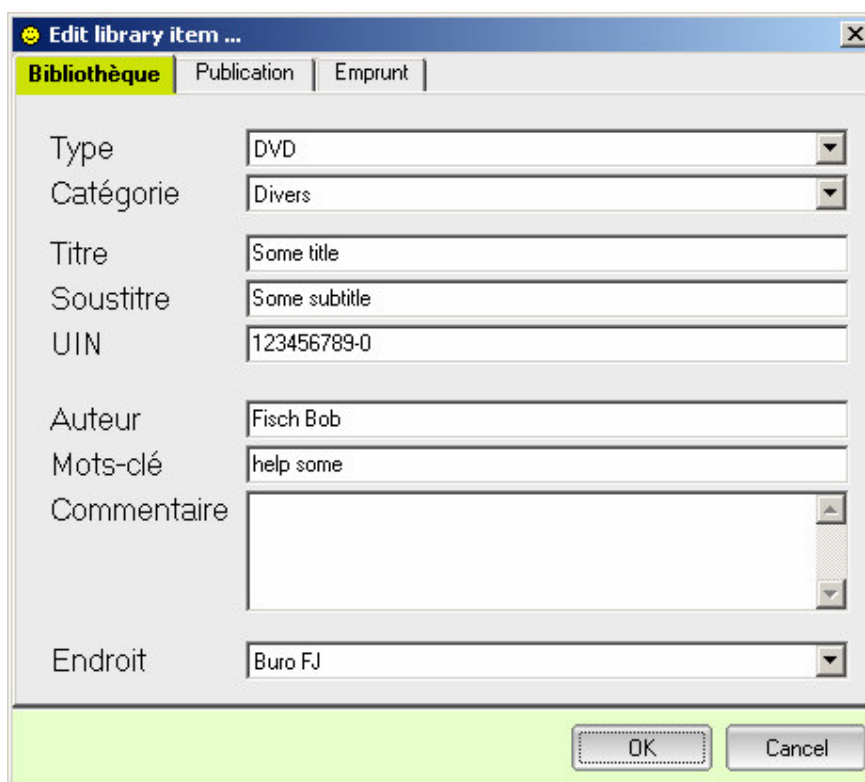


Figure 30: Article manager, edit item

5.4.3.2. Types

Types can be inserted, edited or removed.

5.4.3.3. Categories

Categories can be inserted, edited or removed.

5.4.3.4. Places

Places can be inserted, edited or removed.

5.4.4. Database structure

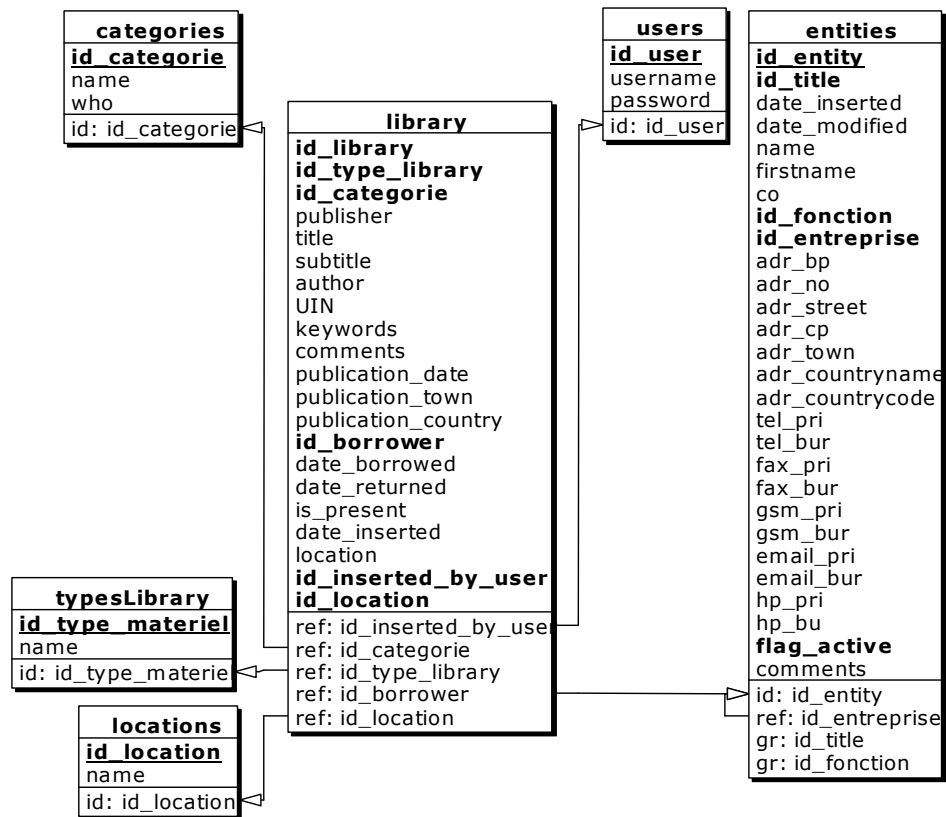


Figure 31: Article manager, database structure

5.4.5. Class diagram

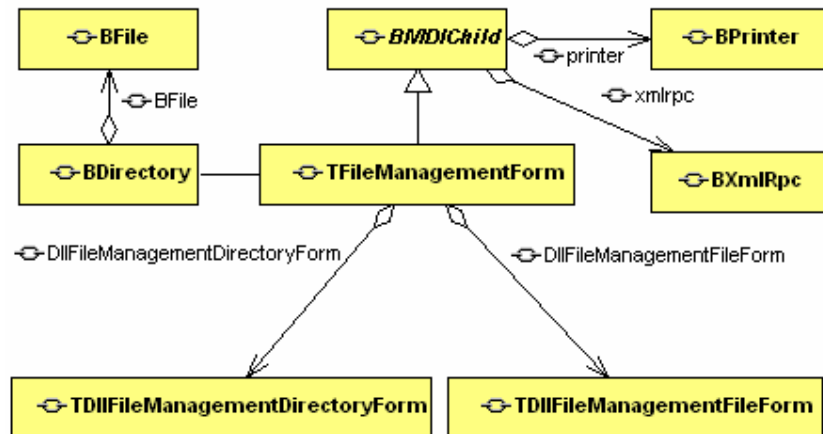


Figure 32: Article manager, class diagram

5.5. Photo manager

5.5.1. Description

The article management module offers the possibility to manage different articles, ordered in categories.

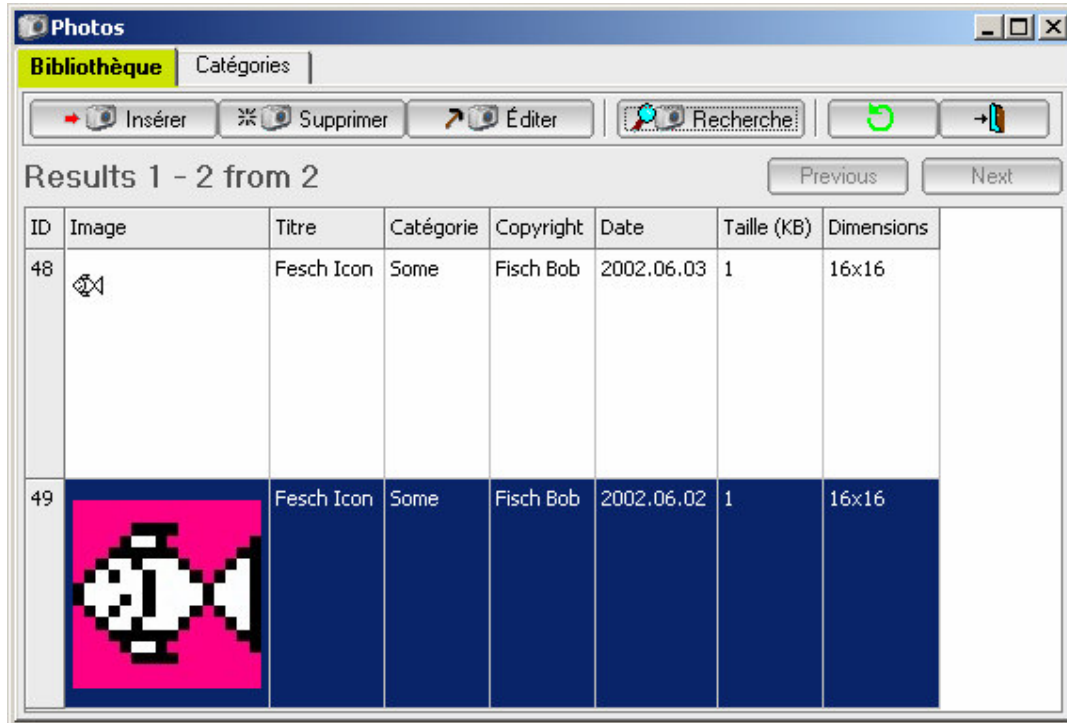


Figure 33: Photo manager, main screen

5.5.2. Definitions

5.5.2.1. Photo

A photo is an image file of one of the following types:

- BMP
- JPG
- ICO
- EMF
- WMF

5.5.2.2. Categories

Each photo is held inside a category.

5.5.3. Functionalities

5.5.3.1. Photo

The users can add, edit or remove photos. The user may also search for photos. This module uses an advanced search method, allowing advanced search options like plain text search and logical operators.

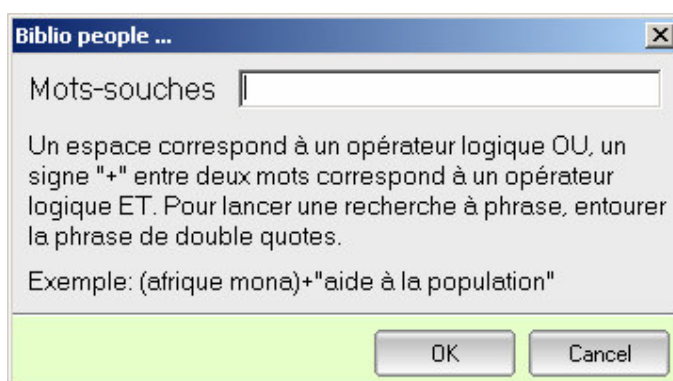


Figure 34: Photo manager, search dialog

For each photo a thumbnail is generated during the upload to the server. This reduces the download time while the client retrieves a list. Full size images may be downloaded on request.



Figure 35: Photo manager, big preview

5.5.3.2. Categories

Categories can be inserted, edited or removed.

5.5.4. Database structure

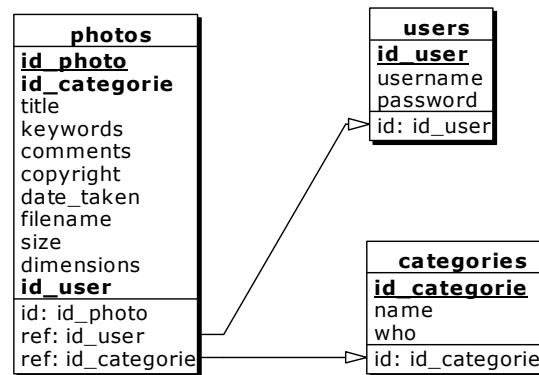


Figure 36: Photo manager, database structure

5.5.5. Class diagram

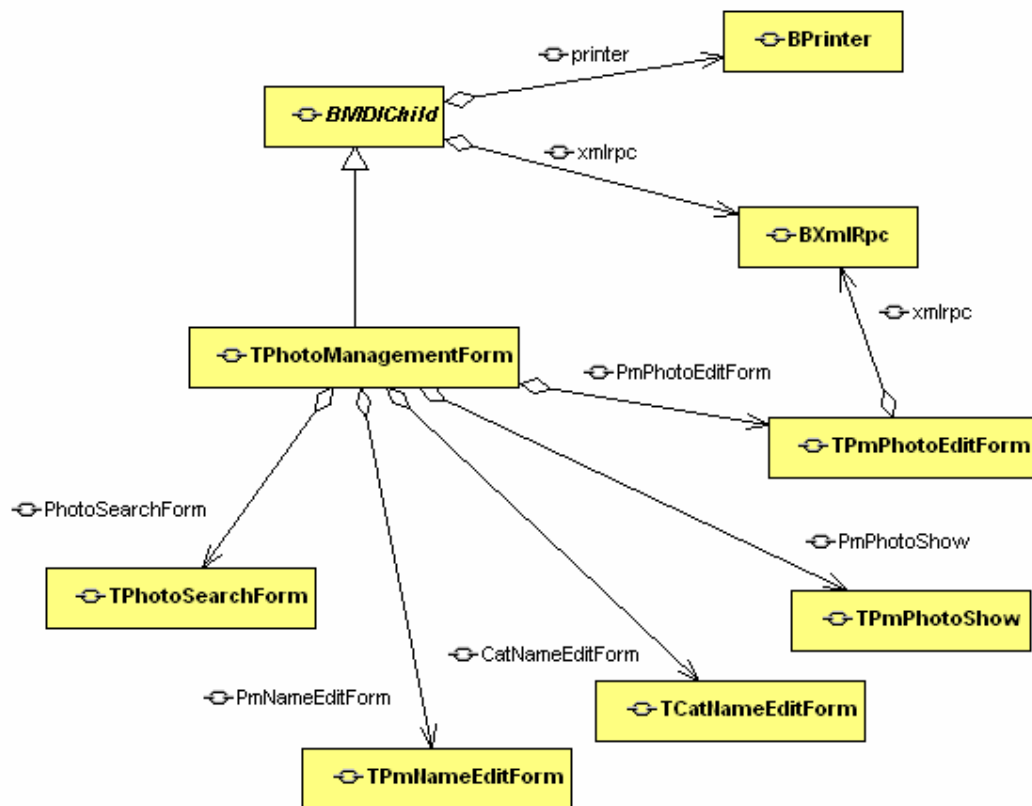


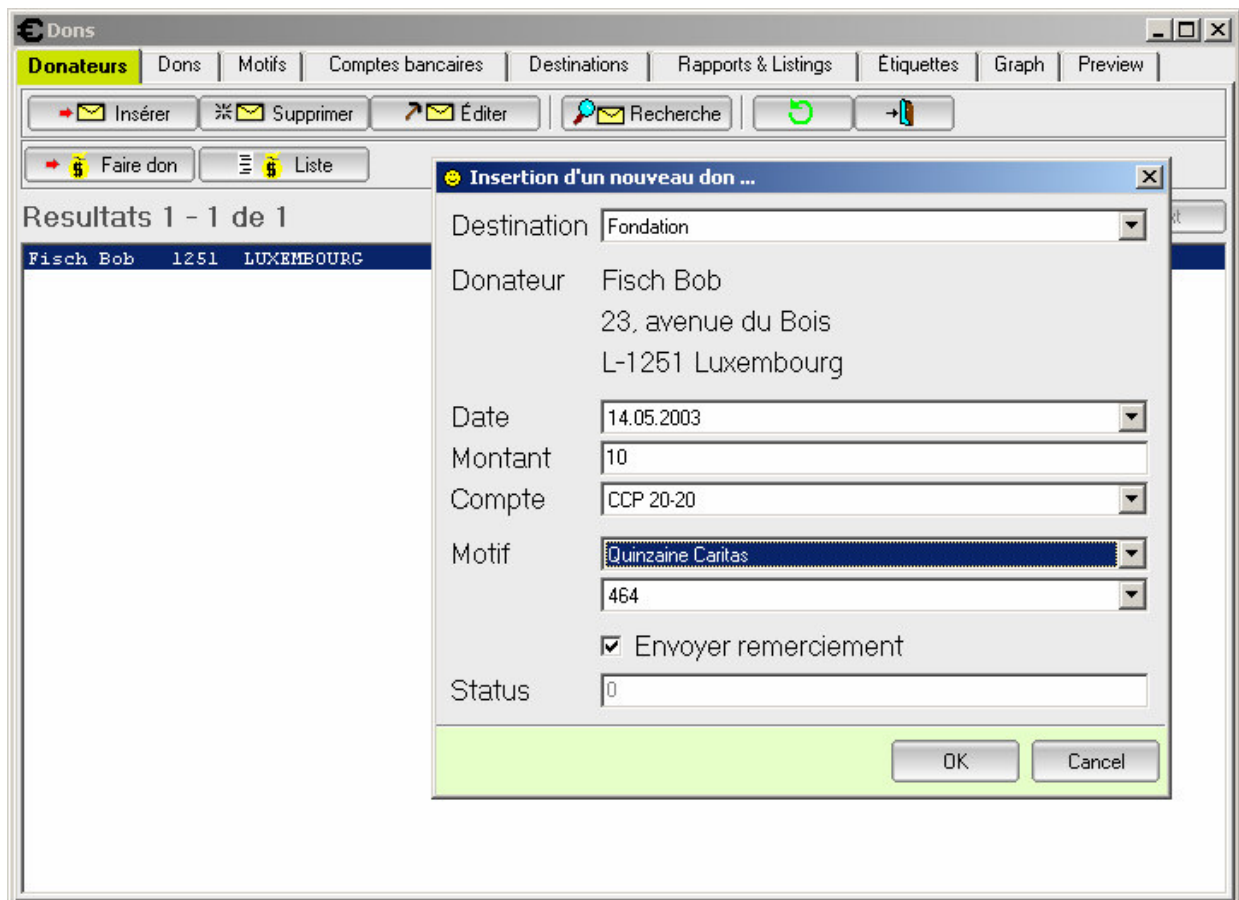
Figure 37: Photo manager, class diagram

5.6. Donation manager

5.6.1. Description

The donation manager is not limited to managing donations, but it has a wider range of tasks to achieve. It is actually the biggest module of the application.

As this module is much related to the address management module, it allows also the inserting, editing or removing of entities from the system.



The screenshot shows the 'Dons' application window. The main menu includes 'Donateurs', 'Dons', 'Motifs', 'Comptes bancaires', 'Destinations', 'Rapports & Listings', 'Étiquettes', 'Graph', and 'Preview'. The 'Donateurs' menu is currently selected. Below the menu, there are buttons for 'Insérer', 'Supprimer', 'Éditer', 'Recherche', and 'Faire don'. The 'Faire don' button is highlighted. The main area displays 'Resultats 1 - 1 de 1' and a list of donors with the entry 'Fisch Bob 1251 LUXEMBOURG' selected. A dialog box titled 'Insertion d'un nouveau don ...' is open, showing the following fields:

- Destination: Fondation
- Donateur: Fisch Bob
23, avenue du Bois
L-1251 Luxembourg
- Date: 14.05.2003
- Montant: 10
- Compte: CCP 20-20
- Motif: Quinzaine Caritas
- 464
- ☒ Envoyer remerciement
- Status: 0

The dialog box has 'OK' and 'Cancel' buttons at the bottom right.

Figure 38: Donation manager, main screen with donation capture

5.6.2. Definitions

5.6.2.1. Donation

A donation is a certain amount of money that has been sent by someone (= an entity) at a certain moment.

5.6.2.2. Account

This is the account on which the money arrives.

5.6.2.3. Motive

A motive is the purpose for which someone donated money.

5.6.2.4. Destination

This is the destination of the money inside the organisation.

5.6.3. Functionalities

5.6.3.1. Donation

The user can add donations to the system, search for them, and, if he has enough rights, edit them.

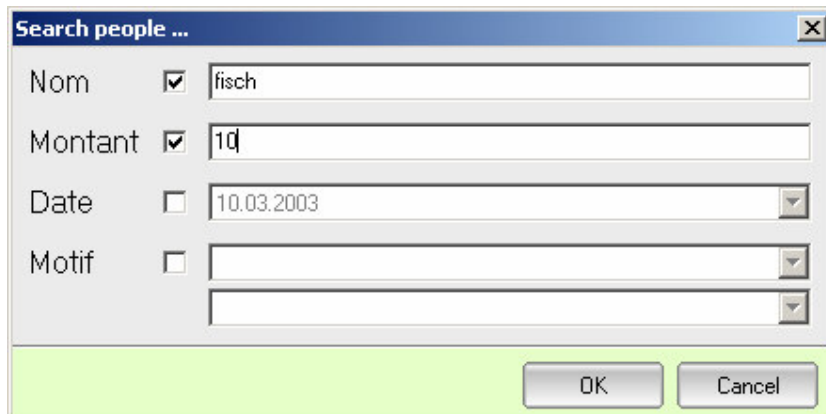


Figure 39: Donation manager, search for donations

5.6.3.2. Account

Accounts can be inserted, edited or removed.

5.6.3.3. Motive

Motives can be inserted, edited or removed.

5.6.3.4. Destination

Destinations can be inserted, edited or removed.

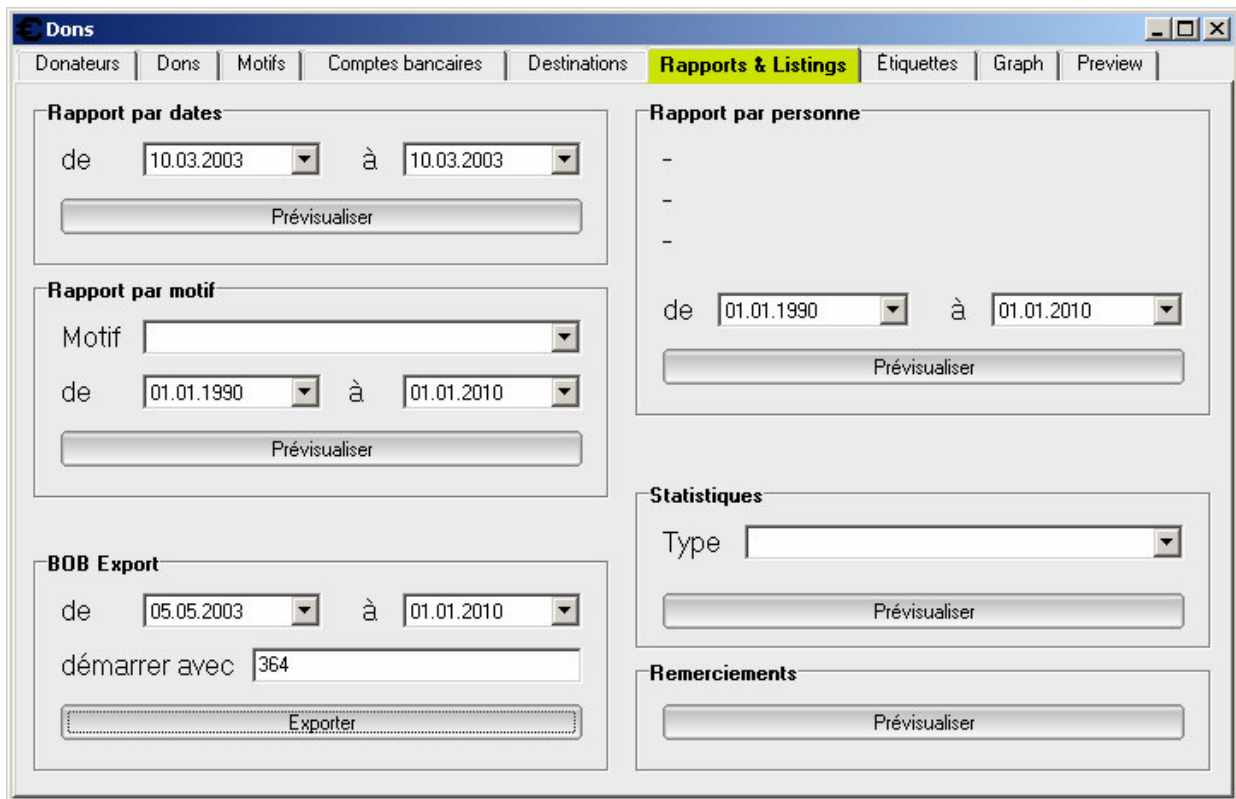


Figure 40: Donation manager, reports and listing page

5.6.3.5. Export

Donations can be exported to an accounting management application called BOB. The export is done via a text file with a predefined structure.

5.6.3.6. Printing

This module allows different printings. First of all simple reports can be printed, but also address labels, graphics and tables. The user can also directly print onto "thank-you" letters.

5.6.3.7. Reports

This module offers some predefined reports:

- report of all donations between two dates
- report of all donations between two dates for a certain motive
- report of all donations of an entity between two dates

Dynamically defined reports are available through the statistics functionality.

5.6.3.8. Statistics

Statistics are defined dynamically. They are described in an XML file that is located on the server and downloaded into the application. Depending on the input description, the module produces a user input window and catches some information.

Statistiques - Intersection entre plusieurs projets (par nom)

Motifs

- "Weihnachtsfreude verschenken"
- Afghanistan - progr. nutritionnel femmes et enf.
- Afghanistan, aide d'urgence cc. tremblement de terre
- Afrikahilf 1996
- Aide à Caritas Kisangani / Zaïre
- Aide à Kisangani - Crise Zaïre
- Aide à Taiwan - séisme 1999
- Aide au Bangladesh

entre le 14.06.2002

et le 14.06.2003

OK Cancel

Figure 41: Donation manager, dynamic query interface

Next the query is composed. The server executes it and sends back the output, which is then formatted, depending on the description in the XML file. Four types of output formats exist; they can be:

- in form of a list
- in form of a table
- in form of a graph
- in form of an Excel output file

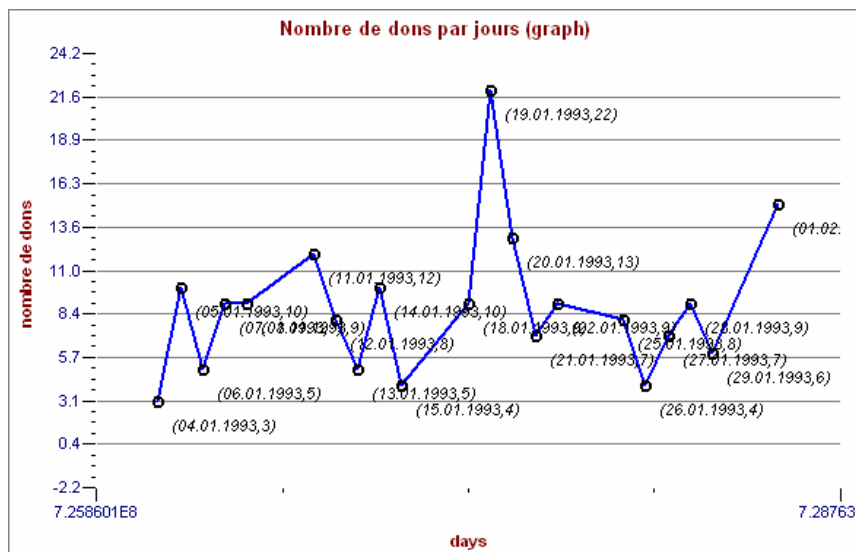


Figure 42: Donation manager, output graph

5.6.4. Database structure

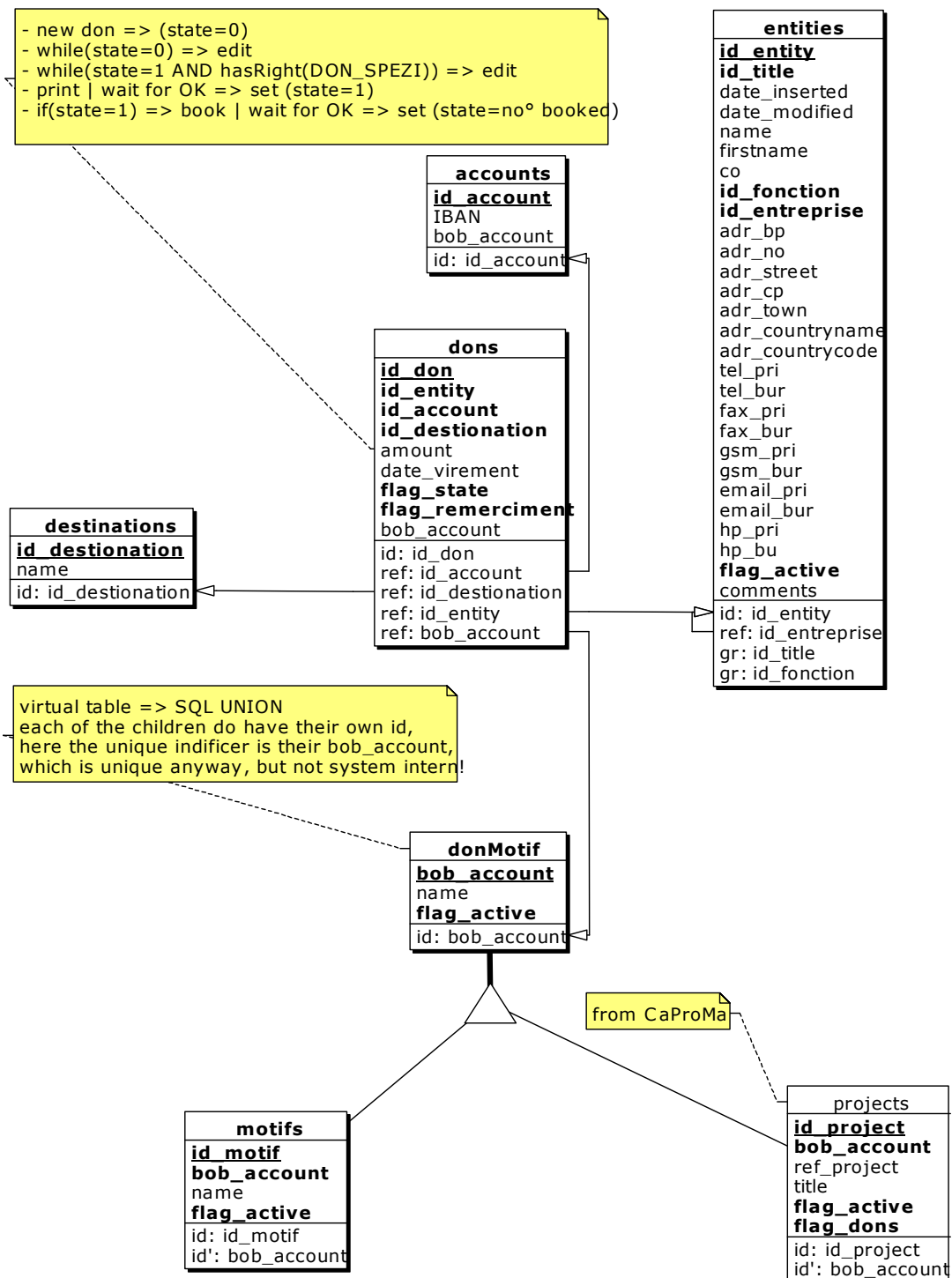


Figure 43: Donation manager, database structure

5.6.5. Class diagram



Figure 44: Donation manager, class diagram

6. CONCLUSION

6.1. The project

All the way through, this project fascinated me. I found the dynamic side, with the downloaded code integration at run time, especially challenging. Even during harder periods, when I was dealing with a lot of memory run-time errors and pointer exceptions, it kept me going on.

This might not be the best place for me to mention it, but it gave me a sense of achievement when finally, after several days of trying, I got something to work the way I wanted it to.

But there were other parts too, which turned it into a kind of adventure, doing a lot research and trying out many possibilities until I found a way to make it operational. Data transmission problems (due to some errors in the XML-RPC component I used), memory leaks and pointer exceptions were just some of the problems encountered. It was not always easy to find a solution.

Whichever way I look at it, this was a really interesting and fascinating project.

6.2. General conclusion

I am very satisfied with the experience I gained during my final study work. I was given the possibility to work inside an organisation, assuming responsibility and learning about its needs and requirements. Employees welcomed me with open arms, so it was a pleasant work.

Concerning my technical knowledge, this final study work offered me an opportunity to contribute my acquired knowledge to this project. I was able to fill gaps concerning maintainability of applications and long-term planning. Furthermore I was shown again how important good communication between application developer and application end-user is.

Last but not least, I have to admit that, contrary to the opinion I had last year after my practical semester in a web design company, now I can imagine very well to work in this sector. I suppose the reason for this change of mind is the fact that last year I was surrounded by computer specialists and that my work was somehow always the same. During my final study work, the people around me had all different jobs, different interests and my work was a lot more varied. This made it much more interesting for me.

7. REFERENCES & ACRONYMS

7.1. References

Caritas Luxembourg	http://www.caritas.lu
Delphi Components	http://www.torry.net
▶ AdvStringGrid	http://www.tmssoftware.co
▶ BarMenu	http://www.bluecave.net
▶ DCPCrypt	http://www.cityinthesky.co.uk
▶ StatusBar	http://www.delphifreestuff.com
▶ dxButton	mhoffmann@apriori.de
▶ elegantMDI	http://www.zecos.com
▶ Indy	http://www.nevrona.com/indy
▶ lbButton	http://www.leif-bruder.de
▶ mxNativeExcel	http://www.maxcomponents.net
▶ GifImage	http://www.melander.dk/delphi/gifimage
▶ UJGraph	ujhs@aol.com
▶ wavePanel	http://www.zecos.com/maxspc
▶ XMLParser	http://www.destructor.de
▶ XmlRpc	http://www.codepunk.com
General Search Engine	http://www.google.com
MySQL	http://www.mysql.org
XML-RPC Protocol	http://www.xmlrpc.org

7.2. Acronyms

DLL	Dynamic Linked Library
HTTP	HyperText Transport/Transfer Protocol
MDI	Multiple Document Interface
ODBC	Open Data Base Connectivity
RPC	Remote Procedure Call
SQL	Structured Query Language
VCL	Visual Component Library
WWW	World Wide Web
XML	eXtensible Markup Language

8. INDEX

A

architecture 3, 14, 15, 17, 18, 21

C

CARiDAS 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26
client 17, 18, 19, 21, 24, 25, 26

Caritas 1, 2, 8, 9, 10, 51

class diagram 20, 22, 29, 34, 37, 40, 43, 49

client . 3, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 23, 24,
25, 26, 42

container 17, 18

D

database 14, 16, 24, 29, 33, 37, 40, 43, 48

Delphi 14, 17, 51

DLL 17, 18, 19, 51

M

maintainability 14, 50

management 11, 20, 27, 30, 35, 38, 41, 44, 46

manager .. 2, 21, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49

MDI 17, 18, 19, 51
child 17, 18, 19
parent 17, 18

module(s).... 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 27,
28, 30, 31, 32, 35, 38, 39, 41, 42, 44, 46, 47

MS-ODBC 14

MS-SQL 14

MySQL 14, 51

P

PHP 21

protocol 14, 15, 16, 17, 21, 23, 51

R

resquest(s) 16, 21, 23, 24, 25, 26, 42

S

security 3, 15, 16, 17, 18, 21, 23, 25, 26

server 3, 11, 12, 14, 15, 16, 17, 19, 21, 22, 23, 24, 25,
26, 35, 42, 47

sessions(s) 16, 24, 25, 26

SQL 14, 21, 51

U

user(s) 3, 12, 14, 16, 17, 21, 23, 24, 25, 26, 27, 28, 29,
30, 31, 36, 39, 42, 45, 46, 47, 50

V

VCL 18, 51

W

WWW 21, 51
method server 21

X

XML 14, 15, 16, 21, 23, 47, 50, 51

XML-RPC 14, 15, 16, 21, 23, 50, 51
server 16, 21