Université de Montréal

Improvements Brought to Graphical User Interfaces for Insurance Illustration Systems

Par

Imad Eid

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la faculté des études supérieures

en vue de l'obtention du grade de

Maître ès sciences (M.Sc.)

Avril 2000

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé :

'Improvements Brought to Graphical User Interfaces for Insurance Illustration Systems'

Présenté par :

Imad Eid

a été évalué par un jury composé des personnes suivantes :

M. Jean Vaucher	**President du jury**
Mme. Esma Aïmeur	**Directrice de recherche**
M. Houari Sahraoui	**Membre du jury**

Mémoire accepté le:

A place that I like…

Abstract

This thesis summarizes my experience during an internship at Logisil Consulting Inc. as part of my Masters in Computer Science. It also describes the projects I worked on, the challenges faced, lessons learned and where I failed or excelled the most in implementing tasks and features.

At Logisil, I worked on two projects and was responsible for the Graphical User Interface (GUI) of a Life Insurance Illustration System (LIIS). The GUI is the level of an application that allows the interaction of its heart and the end user. LIISs have a GUI that allows the interaction between the user (Agent) and the Calculation Engine (Calc. Engine) which is the heart of the application. GUIs for LIISs like most other applications should be user friendly and should allow the user to get to the end result, the safest, fastest and best way possible. Developing such systems is time and resource consuming. However maintenance costs can sometimes exceed development costs. LIISs can include bugs that sometimes are not easy to fix. Agents could ask for some sophisticated features that require research and analysis before implementation. In order to facilitate software maintenance and decrease costs, software-engineering techniques such as reverse engineering are implemented. From the analysis phase to maintenance, project management implements those techniques to optimize costs and code, especially in the development phase. I took advantage of software engineering techniques and of my analytical skills, which are research oriented, to develop and improve LIISs during my training period at Logisil. In the process of doing these tasks I had to apply software-engineering techniques and learned that some are crucial for a project's survival such as resource allocation and testing. The thesis presents the "golden rules" for GUIs and some software engineering techniques that were used. On an industrial level people talk about productivity which is a very important element for successful project planning. Resource attribution, work environment and conceptual choices are essential elements for successful projects.

Key words

Graphical User Interface (GUI), Life Insurance Illustrations Systems (LIIS), Automatic Language Switching (ALS), Hartford Templates, Compare To feature, Reverse Engineering.

Résumé

Ce mémoire résume mon expérience au cours d'un stage à "Logisil Consulting Inc." dans le cadre de ma Maîtrise en Informatique. Il décrit aussi les applications sur lesquelles j'ai travaillées, les difficultés rencontrées, leçons apprises ainsi que mes qualités et lacunes dans l'implémentation de taches et options dans ces applications.

A Logisil j'ai travaillé sur deux projets. J'étais responsable de l'interface graphique de systèmes d'illustrations d'assurance vie. L'interface graphique d'une application est le niveau qui permet l'interaction de son noyau avec l'utilisateur. Les systèmes d'illustrations d'assurance vie ont une interface qui permet l'interaction entre l'utilisateur (l'agent) et le moteur de calcul qui est le noyau de l'application. Comme pour la plupart des applications, l'interface graphique des systèmes d'illustrations devrait être conçu pour l'utilisateur et devrait lui permettre d'arriver aux résultats, d'une façon sure, rapide et simple. Le développement de tels systèmes, prend beaucoup de temps et de ressources. Cependant les coûts d'entretien peuvent dépasser parfois les coûts de développement. De tels systèmes peuvent inclure des problèmes qui ne sont pas toujours faciles à réparer. De même les usagers peuvent exiger des caractéristiques et options sophistiquées. Ceci incite des recherches et une analyse avant implantation, quitte à faciliter la maintenance. Des techniques en génie logiciel telles que 'reverse code engineering' sont implantées pour cette fin, et permettent en plus de diminuer les coûts d'entretien. De la phase d'analyse à l'entretien, la direction de projet se base sur ces techniques pour optimiser les coûts et le code, surtout dans la phase de développement. Je me suis servi de ses techniques et j'ai procédé à une phase d'analyse pour améliorer des systèmes d'illustrations pendant ma période de formation à Logisil. Afin d'exécuter ces tâches, j'ai utilisé des techniques en génie logiciel, dont quelques-unes sont cruciales pour la survie des projets telles que l'attribution des ressources et les procédures de tests. Ce mémoire présente les règles d'or des interfaces graphiques et quelques techniques en génie logiciel qui ont été utilisées. Dans le milieu industriel on parle de productivité qui est un élément très important pour la planification de projets prospères. L'attribution de ressources, l'environnement de travail et les choix conceptuels sont essentiels pour la réussite d'un projet.

Mots Clées

L'Interface Graphique, les Systèmes d'Illustrations d'Assurance Vie, le Changement automatique de Langue, Les Descripteurs de Hartford, l'Option de Comparaison , la Réingénierie de Code.

Table of Contents

. . . To My Parents

. . . And All My Beloved

Acknowledgements

I would like to thank my Research Director Prof. Esma Aïmeur for all her support. It's been a successful venture. I really value all you did for me.

I am honored to have Mr. Jean Vaucher as my jury's president and Mr. Houari Sahraoui as member.

I would also like to thank all the members of the computer science department at University of Montreal who made all this possible for me and who gave me the tools and the background I need to accomplish my dreams.

It's been an honor working with everybody (Staff and Friends) at the computer science department.

I would also like to sincerely thank my project manager on Hartford Life Mrs. Lorraine Pitre, our team manager Miss. Denise Tsakalaki and all the members of the LOGISIL family for making this period so joyful and fruitful for me. The University of Montreal family has grown at Logisil. This made me feel not very far from my second home.

Thank you Denise for taking such an interest in my training. It's amazing working at Logisil.

Thank you Sassine Abou Jaoude for introducing me to the Logisil family. You made my life easier. I will never forget that.

Thank you all.

List of Figures

Chapter 1. INTRODUCTION

In the computer industry, professionals are tending to go from one company to another looking for more benefits. This creates problems to organizations because they lose expertise that could affect the schedules and deliveries of their on-going projects. Specialized resources are hard to find and keep. This is where *Software Engineering* intervenes to make sure that projects are done on time by providing the required resources and applying the required project management techniques.

Logisil, like most companies in the industry, is confronted by this problem. But due to the awareness of it's managers and the good working atmosphere and benefits they provide to their employees, they are doing pretty well.

One area of Logisil's expertise lies in Illustration Systems for Life-Insurance Companies or Annuity Simulation Systems, which allow a broker (agent) to give a policy illustration to a client. Such systems are composed of four elements:

- Graphical User Interface (GUI)
- Calculation Engine
- Reports
- Business Rules

1.1. Logisil

Logisil <<Good people... in Good time>> [Logisil, 2000]

Since 1986, the company "Logisil Consulting Inc.", specialized in information technology, has been increasing it's know how. Analysis of needs, concrete computer science based solutions as well as the efficient implementation of these solutions has anticipated in developing this one of a kind expertise. This expertise varies from tailored application development of client-server technologies, to new object oriented tendencies as well as Internet.

Logisil team is composed of 80 computer science professionals, among which we number actuaries having a vast expertise in insurance and financial domains. This provides the company with a unique and highly prized expertise.

Logisil clients, mainly insurance and financial institutions, are also considered as partners, contributing to the company's growth and expansion.

These partners are among the leaders in the field. For example, IBM and Logisil hold very good relations. Logisil is a member of the programs Best Team and Solution Developer.

Logisil is a subcontractor for Navisys, a leader in the development of Insurance Illustration systems in the US. Navisys provide Logisil with specific application development tools as well as assistance and collaboration on different levels.

It is known in the life insurance field that all companies provide their agents and brokers with customized insurance illustrators. Since each broker is a company of its own, selling the products of multiple insurance companies, the illustration system should be easy to use, interesting, portable and reliable. The software does a simplified financial analysis of a potential customer and prints a report including recommendations as to the amount that should be invested, which could be taken as is, and included in a life insurance proposal. Proposals could be printed and handed to the client. The system also takes into consideration medical constraints, prints special documents that need to be filled out…

All systems are developed to function under Windows3.1, Windows95 or 98, and Windows NT platforms. They are developed using Borland C++ or Microsoft Visual C++, ECTA Report Macro Language and InstallShield.

Logisil is growing at the speed of light, because of it's professionalism, expertise, and it's team dedication.

During my training I was assigned to two projects and on my free time, I participated in other smaller projects. I also assisted in training new employees. It was a very active year for all of us at Logisil.

1.2. The GUI Team

Logisil *Graphical User Interface* (*GUI*) team is composed of a number of programmers and insurance consultants. The group has grown drastically in the last few months because of the growing demand on financial applications. This group handles insurance and private banking projects mostly for US and Canadian clients. Expansion plans are set to cover a bigger part of the globe.

Logisil might soon be handling clients in France, which is part of it's plan to cover Europe. The first application developed to be presented, as a demo in France is the bilingual version of Metropolitan life (Metlife) called *MetDemo*.

The GUI team is one of the biggest teams at Logisil. It is always in constant communication with the Engine team. We collaborate together to insure that the job is perfectly done.

The way projects are distributed in our department depends on the complexity and importance of the project, and on the revenues that it generates.

Some projects are handled by one or two persons under the supervision of a project manager and others have a team of five to ten people. The hierarchy of the GUI department is presented in figure 1 here below. I am at the (*) level.

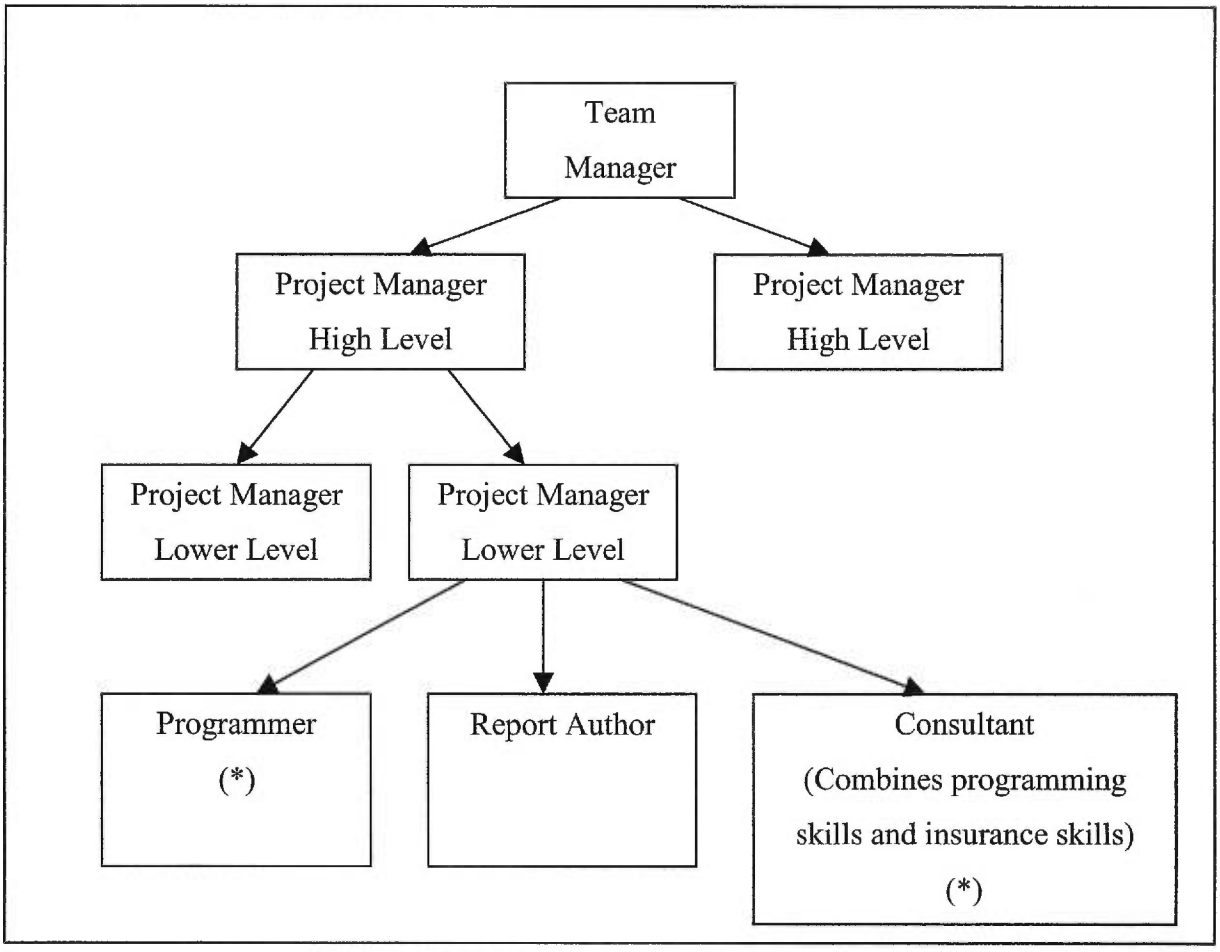

Figure 1. GUI department hierarchy.

Having introduced the company, I will give a brief description on the projects I worked on.

1.3. Projects I worked on

I worked on two major projects during this period. What was the purpose behind this work?

1.3.1. MetDemo Project

Metdemo is the French/English version of the Metlife project. This project was developed to be used as a demo of our illustration systems in France. It was my first project at Logisil after a three weeks training period. I was lucky enough to be assigned to this project and furthermore I was nominated to be the project manager.

This came as a sign of the confidence that our higher management confided in its employees. I was also given the task of the C++ programmer, had a report resource to assist me in the report translation and was introduced to an insurance consultant who offered his help in translating technical insurance terms. We also had a backup C++ programmer whose role was to jump in if things got out of control.

The purpose behind this project was to make the system bilingual. Starting from an English version of Metlife, we had to translate all the resources to French, make the system distinguish between English and French depending on the user's choice, and last but not least, we had to make the system switch from English to French at runtime and vice versa. We will introduce the problems and challenges that we faced in this project in the next section and detail them later on in our thesis.

The Metdemo system was presented in France. By summer 2000 Logisil might sign a contract which will be its first in insurance illustration systems in Europe.

This project involved software-engineering techniques that allowed us to respect our deadlines. Resource attribution and project management were mainly applied.

1.3.2. Hartford Life (Merlin) project

Merlin is an insurance illustration system customized for Hartford Life Insurance Company. I was assigned to Merlin right after finishing Metdemo. Merlin is an old project that started a few years ago at Navisys and it's maintenance and development were confided to a Logisil project manager. I was the C++ programmer for this project.

I was mainly confided the development of new features that were custom made depending on the client's needs (*Compare to feature* and *Template implementation*). Problem solving was also one of my tasks.

The purpose of assigning a trainee as the C++ main programmer was due to the fact that Merlin involved some innovations that required research and analysis. It also involved bugs that required analysis, deep debugging and code tracking.

1.4. Objectives behind each project

In this section we discuss my contribution to the projects.

1.4.1. Metdemo project

The objective behind the Metdemo project was to impress a potential French customer. Why a bilingual version? Simply because this version is supposed to be our gate to the European market. Obviously for a French customer a simple French version would have done the trick.

So why go through all the trouble of implementing a bilingual version that allows the switch between languages at runtime? Well, the idea was to impress the client. A simple French version would have provided the potential customer with an idea on our illustration system, but the version we presented helped the customer see what we were capable of doing, and to which extent we were ready to go.

1.4.2. Hartford Life (Merlin) project

Merlin enhancements and problem solving were part of our daily preoccupations. We tried our best and pushed ourselves to the extreme in order to satisfy our client.

Like most customized projects, Merlin involved innovations that came as a results of our client's requests in response to agents demands… Enhancements such as "compare to feature" came as a result of agents' repetitive demands for an option that allow them to compare the same data using a different product. The idea behind this feature is very interesting and useful. Agents can provide clients with different insurance plans by just three mouse clicks.

Templates were introduced to provide agents with defaults that contributed to reduce their workload.

Once defaults for particular repetitive cases are set, agents can use those defaults without having to go through the trouble of retyping all the data. The innovation was in the template implementation. I had to customize the templates to meet our client's needs. This enhancement will be discussed in details in the templates implementation section.

Another enhancement brought to Merlin was *code optimization.* Since it was an old project we had to do an extra step to make it compliant with Navisys and Logisil programming standards.

We also worked on reducing the bugs in the software. Complicated tests (regression tests) were done before releasing new versions to the field. Unfortunately, sometimes bugs were found in field versions. Bugs exist in all software applications. And a part of all programmers' job is to fix those bugs the best way possible without damaging saved data. We will discuss two major bugs that were fixed in Merlin. They were critical because they affected the system behavior. The first is the agent problem. This involved an Id problem that caused proposals to be linked to wrong agents… Second is the import/export problem, which made cases, become inaccessible.

After a brief description of the projects I worked on I will shed a light on the methods adopted.

1.5. Methods adopted

Conceptual and implementation details depend on the task to be accomplished. Nevertheless they have to comply with the "golden rules" set for GUIs. The "golden rules" are covered in chapter 2. They also have to follow software engineering techniques that are crucial for getting good results and making good decisions as to which implementation alternative to adopt and how to implement it. Analysis and research are the key that allow us to choose one specific solution, present among many, and confirm that it is the best.

Sometimes bad decisions are taken because of lack of analysis and understanding of what the client really wants, out of a certain feature. When I was implementing the templates on Merlin, the first version I came up with was a complete waste and I ended up re-implementing the whole feature so that it would meet our clients needs. Lack of specifications from our client's side contributed in this incident. But I should have known better. After all I was able to correct my mistake without affecting the flow of the project.

Code reengineering techniques were adopted during *Code Optimization*. Optimization was introduced to improve the application runtime. It also contributed in reducing maintenance costs. All these issues will be covered in details through the thesis.

In the remainder of the thesis, we will provide the reader with an overview on the GUI in general and Navisys/Logisil GUI in particular.

In chapter 2, GUIs golden rules as well as software-reengineering principles are discussed. Chapters 3 and 4, cover the two projects I worked on (MetDemo and Merlin). In the scope of our research, code simplification (restructuring) was also part of my involvement on Merlin. It is elaborated in chapter 5. Chapter 6 familiarizes the reader with the Logisil Software environment. Having a general understanding of the nature of the projects and technologies used, the user can finally get an idea of the time that such projects or features require to be accomplished. This is presented in chapter 7. So the user is left with an idea on the costs and productivity which are crucial on an organizational level. Chapter 8 finally ends this thesis by presenting the benefits I got from this internship.

Chapter 2. STATE OF THE ART

In terms of GUI, where are we and what are the perspectives? This issue as well as Software Engineering techniques are covered in this chapter.

2.1. Graphical User Interface (GUI)

It has been proven in the recent years that good, user interfaces produce corporate success stories and Wall Street sensations such as Netscape, America Online. They also, at an individual level change many people's lives: doctors can make more accurate diagnostics, children can learn more effectively, insurance agents can produce life insurance illustrations in a glance…

New technologies are used to improve the appeal and information content of user interfaces. We recall the sound (music and voice), three-dimensional representations, animation, and video. New techniques such as virtual reality may change the ways we interact with and think about computers.

The U.S. Military Standard for Human Engineering Design Criteria (1989) states the objectives of a GUI:

- Achieve required performance by operator, control, and maintenance personnel
- Minimize skill and personnel requirements and training time
- Achieve required reliability of personnel-equipment combinations
- Foster design standardization within and among systems

System engineering goals are stated as follows:

- *Proper functionality:* if the functionality is inadequate, it does not matter how well the human interface is designed. Excessive functionality is also a danger, and providing it is probably the more common mistake of designers, because the clutter and complexity make implementation, maintenance, learning, and usage more difficult.
- *Reliability, availability, security, and data integrity:* System reliability and availability are very crucial to whether the user will or won't be confident in using a system. Errors are not tolerated in this field.
- *Standardization, integration, consistency, and portability:* The Standardization of systems is essential because of the number of application that users are forced to

use. This would create room for errors if standards weren't applied, because users are forced to learn new methodologies constantly.

- *Schedules and budgets:* Careful planning and courageous management are needed if a project is to be completed on schedule.

The standard also lists five measurable human factors that are central to evaluation:

- *Time to learn:* How long does it a user to learn how to use the commands relevant to a certain task (benchmark task).
- *Speed of performance:* How long does it take to carry the tasks.
- *Rate of errors by users:* How many and what kind of errors do people make in carrying out the task?
- *Retention over time:* How well will the users maintain their knowledge after an hour, a day, or a week?
- *Subjective satisfaction:* How much did users like using various aspects of the system?

These factors could be thought of as principles that should endure as new user interface technologies emerge. The principles are so basic that even futuristic dialogue designs such as three-dimensional interfaces with DataGlove input devices, gesture recognition, and live video images will always have to take them into account as long as they are based on the basic paradigm of dialogues and user commands [Nielsen, 1990].

There is always room to improve the user interface. The cluttered displays, complex and tedious procedures, inadequate functionality, inconsistent sequences of actions, and insufficient informative feedback can generate debilitating stress and anxiety that lead to poor performance, frequent minor and occasional serious errors, and job dissatisfaction.

The following section will concentrate on the theories and rules as well as the strategies used in a successful graphical user interface.

2.1.1. Object-Action Interface (OAI) model.

Graphical User Interfaces (GUI) consist of a visual representation of objects and actions. The emphasis is now on the visual display of user task objects and actions. Doing object-action design starts with understanding the task. That task includes the universe of real-world objects with which, users work to accomplish their intentions and the actions they apply to those objects. Task action start from high-level intentions that are decomposed into intermediate goals and individual steps. This would allow the designer to create metaphoric

representations of the interface objects and actions. Then the designer must make the interface actions visible to users, so that users can decompose their plan into a series of intermediate actions, such as opening a dialog box, all the way down to a series of detailed keystrokes and clicks. So the theory of decomposing complex problems into several smaller problems so that they become manageable can be applied. This would force designers who support professionals to be knowledgeable in the domain they are supporting.

The interface objects and actions are also decomposable. Next is an example that illustrates how this hierarchical decomposition is done: in writing a business letter using computer software, users have to integrate smoothly their knowledge of the task object and actions and of the interface objects and actions. They must have the high-level concept of writing (task action) a letter (task object), recognize that the letter will be stored as a document (interface object), and know the details of the save command (interface action). Users must be fluent with the middle-level concept of composing a sentence, and must recognize the mechanisms for beginning, writing, and ending a sentence. Finally, users must know the proper low-level details of spelling each word (low-level task object), and must know where the keys are for each letter (low-level interface object) [Schneiderman, 1998].

Interface objects and actions help in making the low-level syntactic details disappear in the systems. This is of great importance since it accelerates the retention process of the user, who doesn't have to learn combinations of keystrokes for every system that he will eventually end up using.

An example of an OAI is the Microsoft Windows interface presented in figure 1. We can also see the hierarchical levels (high, middle, and low) that are behind these icons. For each one a series of actions is associated. The retention process is accelerated enormously by this GUI because the desktop theme is common to almost everybody.

2.1.2. Golden Rules of User Interface Design

In his book "The Elements of User Interface Design" [Mandel, 1997], Mandel presents the "golden" rules of interface design which we found very pertinent and therefore used in our work.

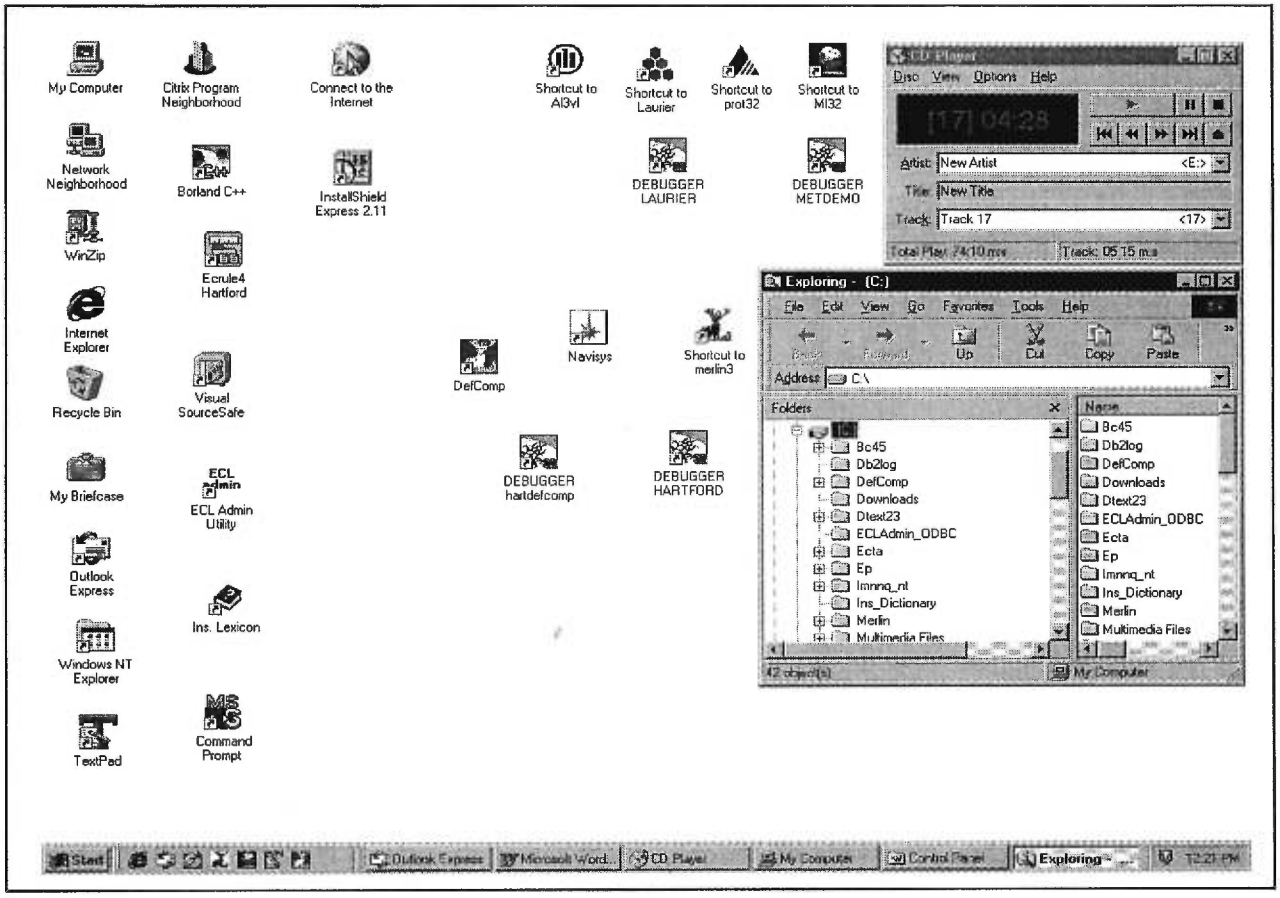

Figure 2. Windows NT OAI.

2.1.2.1. Golden Rule One: Place Users in Control

It all depends on whether the user is a novice or a professional. Sometimes it would be best not to give him full control (it is like driving a car or taking the bus).

Wise designers let users do their work for them rather than attempting to figure out what they want. This is possible through behavioral observation. So the interface would allow the users to go where they want to go and how they want to get there.

There are ten principles that place users in control:

- *Use modes judiciously:* an example would be on the difference between using the insert mode and replace mode.
- *Allow users to use either the keyboard or mouse:* one of the Key Common User Access (CUA) design principles is that users must be able to do any action or task using either the keyboard or the mouse.
- *Allow users to change focus:* users shouldn't be forced to complete predefined sequences. They should be given options to cancel or save and return to where they left off.

- *Display descriptive messages and text:* throughout the interface use terms that users can understand, rather than system or developer terms.
- *Provide immediate and reversible actions and feedback:* Every product should provide undo and redo actions to users.
- *Provide Meaningful Paths and exits:* users should be able to relax and enjoy exploring the interface of any software product, even industrial strength products. The Internet is the best example where navigation is the key. It helps people figure out the interface and become experienced very quickly.
- *Accommodate users with different skill levels:* expert users shouldn't be treated like casual users. In other words they should be provided with macro commands allowing them to perform actions that usually take casual users too many steps in one step.
- *Make the user interface transparent:* By including a trash or shredder (for deletion) as well as other meaningful objects, the user is reaching right through the computer and manipulating the objects directly. That's one of the aspects of transparency. Users should be free to focus on the work they are trying to perform, rather than translating their tasks into the functions that the software program provides.
- *Allow users to customize the interface:* Operating systems offer a great deal of customization for interface elements.
- *Allow users to directly manipulate interface objects:* The interface must be explorable. Users should feel comfortable picking up objects and exploring dragging and dropping them in the interface to see what happens.
- *At least let users think they're in control:* A well-designed interface can comfort and entertain users while the computer system is completing a process. It gives a "status" feedback during long operations.

2.1.2.2. Golden Rule Two: Reduce Users' Memory Load

The capabilities and limitations of the human memory system are behind the idea of making computer systems that store and remember information for users. Humans aren't good at remembering things, so programs should be designed with this in mind.

There are nine principles that reduce users' memory load:

- *Relieve short-term memory:* Users shouldn't be forced to remember and repeat what the computer could and should be doing for them (E.g. a client name that should be used in a coming screen…)
- *Rely on recognition, not recall:* Users should be provided with lists and menus containing selectable items instead of fields where users must type in information without support from the system.
- *Provide visual cues:* Whenever users are in a mode, or are performing actions with the mouse, there should be some visual indication somewhere on the screen that they are in that mode. (E.g. the mouse pointer might change to show the mode or the current action.)
- *Provide interface shortcuts:* Once users are familiar with a product, they will look for shortcuts to speed up commonly used actions.
- *Promote and object-action syntax:* Xerox PARC developers specified the object-action when they built the Star user interface in the late 1970s. Application and system features were to be described in terms of the objects that user would manipulate with the software and the actions that the software provided for manipulating object [Johnson et *al.* 1989]. Consistent implementation of object-action syntax allows users to learn the relationship between objects and actions in the product.
- *Use real world metaphors:* Once a metaphor is chosen for an interface, it should be followed consistently throughout the interface. A metaphor could be extended, but should not be broken.
- *Use progressive disclosure:* Always provides easy access to common features and frequently used actions. Less common features could be hidden and user should always be allowed to navigate to them. Secondary information could be affected to secondary windows so the main window wouldn't have to include everything.
- *Promote visual clarity:* Visual design principles of human perception should be applied, such as grouping items on a menu or list, numbering items, and using headings and prompt text. Graphic designers and book designers are skilled in the art of presenting appropriately designed information using the right medium.

2.1.2.3. Golden Rule Three: Make the interface consistent

Consistency is a key aspect of usable interfaces. One of its major benefits is that users can transfer their knowledge and learning to a new program if it is consistent with other programs they already use. However if consistency principles and guidelines don't make sense in a certain working environment they shouldn't be followed.

There are five principles that make the interface consistent:

- *Sustain the context of the users' tasks:* Users should be provided with points of reference as they navigate through the interface (E.g. windows titles, navigation maps and trees…). They should also be provided with cues that help them predict the results of an action (E.g. dragging an object and dropping it on another should trigger a certain mouse behavior depending on the result of whether the other object could or not accept it…).
- *Maintain consistency within and across products:* Learning how to use one program should provide positive transfer when learning how to use another similar program interface. When things that look like and should work the same in a different situation don't, users experience negative transfer. This can inhibit learning and prevent users from having confidence in the consistency of the interface.
- *Interface enhancements and consistency:* New enhancements should make users only learn a few behaviors or techniques. This is provided if the interface is consistent. Users should not be forced to unlearn trained behavior because it is much more difficult than learning new behavior.
- *Keep interaction rules the same:* If by design, results might be different from what users expect, inform them before the action is performed. Give them the option to perform the action, or cancel the operation, or perhaps perform another action.
- *Provide aesthetic appeal and integrity:* A pretty interface can't cover up for a lack of product functionality. Users don't just want "lipstick on the bulldog", they want a visually pleasing interface that allows them to get the job done.
- *Encourage exploration:* Interfaces today and in the future must be more intuitive, enticing, predictable, and forgiving than the interfaces we've designed to date. The explosion of CD-ROM products and Internet browsers, home pages, and applets have exposed the user interface to a whole New World of computer users. It's time

we moved past user-friendly interfaces to user-seductive and fun-to-use product interfaces, even in the business environment.

During my training I worked on the Logisil/Navisys GUI system which is designed by a group of highly experienced engineers and specialists in insurance illustration systems. This state of the art GUI is shown in figure 3.

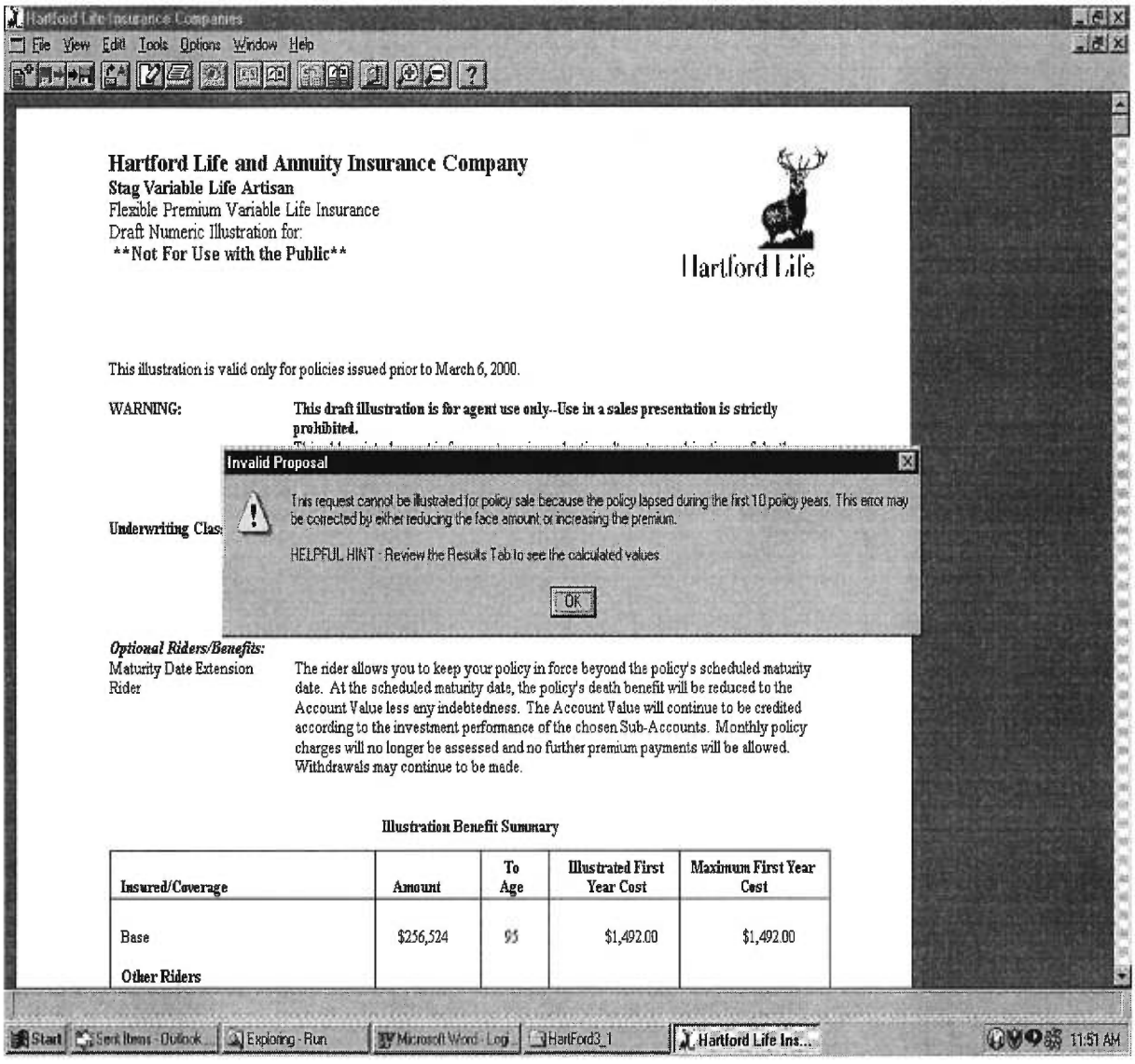

Figure 3. A Significant GUI message.

2.2. Code reengineering

Code reengineering is producing new software source code without changing the overall system function [Pfleeger, 1998].

2.2.1. Data restructuring

A program with weak data architecture will be difficult to adapt and enhance. In fact for many applications data architecture has more to do with long-term viability of a program than the source itself.

When data structure is weak (e.g. flat files are currently implemented when relational approach would greatly simplify processing), the data are reengineered.

Because data architecture has strong influence on program architecture and the algorithm that populate it, changes to the data will invariably result in either architectural or code-level changes [Pressman, 1997].

Breuer and Lano defined the code reverse engineering process in 1991. This process is described in figure 4.

The first activity in reverse engineering is to understand the code. Overall functionality of the entire system must be understood before more detailed reverse engineering work occurs. Benchmarks as well as specifications are used to test if nothing has been broken in the system functionality after code clean up.

It is important to keep specifications up to date because specifications that are written early in the life history of a program are never updated. As changes are made, the code no longer conforms to the specifications. I would like to mention to this regard that when I first started my training I used to change things without updating the specifications. When I had to do code reengineering (simplification) I ended up realizing that what I did was wrong because I needed up to date specification to be able to test if I didn't break anything in my regression tests.

A technique called program segmentation [Ning et *al.* 1994] has been suggested. It consists on grouping code having the same functionality, then re-packing to a new module. This was previously done at Logisil and had to be re-implemented to some functions.

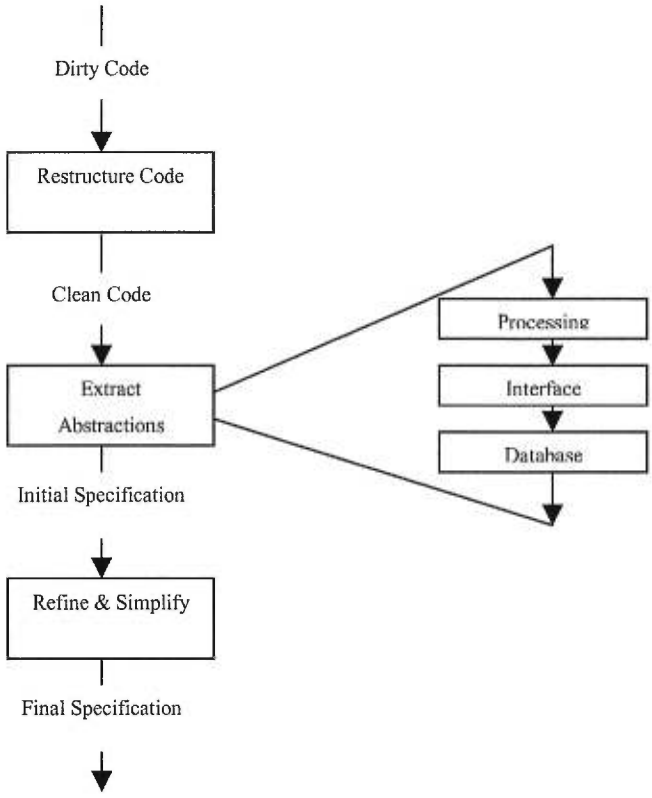

Figure 4. Reverse engineering process [Breuer & Lano, 1991].

The following approach was suggested for reverse engineering class [Breuer & Lano, 1991].

- Identify flags and local data structures within the program that record important information about global data structures (e.g. file).
- Define the relationship between flags and local data structures and the global data structures.
- For every variable within the program that represents an array or file, list all other variables that have a logical connection to it.

2.2.2. Reverse engineering user interfaces

In order to redevelop UI the structure and behavior of the interface must be specified: three basic questions must be answered as reverse engineering of the UI commences [Merlo et *al.* 1993].

- What are the basic actions that the interface must process (e.g. action associated to a mouse click).

- What is a compact description of the behavioral response of the system to these actions.
- What is meant by a replacement or more precisely, what concept of equivalences of interface is relevant here?

In order to implements the templates in the Merlin project these questions had to be answered. We unfortunately had to implement this feature twice. First time it was a Navisys consultant and I. It didn't meet the clients' expectations. So finally I ended up re-implementing this feature. It is important to mention that this feature was implemented without detailed specifications from the client. This kind of relieves us from baring the full responsibility of having to do the same thing twice. But the Hartford project has and always will be a special project unlike other projects where tasks are split between Logisil and Hartford Company. Other projects are directly supervised by Navisys and this leaves no room for error as to Hartford where the client is always present and sometimes changing his mind about certain things. This open room for errors but there has always been and will always be a full understanding from both parts on certain issues (see templates implementation, section 4.1.).

2.2.3. Restructuring (optimizing)

Software restructuring modifies source code and / or data in an effort to make it amenable to future changes. In general, restructuring does not modify the overall program architecture. It tends to focus on the design details of individual modules. If the restructuring effort extends beyond module boundaries and encompasses the software architecture, restructuring becomes forward engineering.

A number of benefits can be achieved when software is restructured [Arnold, 1989]:

- Lead to programs that have higher quality, better documentation and less complexity; conformance to modern software engineering practices and standards.
- Reduces frustration among software engineers who must work on the program thereby improving productivity and making learning easier.
- Reduces the effort required to perform maintenance activities.
- Makes software easier to test and debug.

We experienced this in the Hartford project after doing our code optimization and clean up. Removing redundancies and grouping common code at the parent level makes maintenance and adding new features an easier task (see chapter 5).

2.2.4. Code Restructuring

It is performed to yield a design that produces the same function but with higher quality then the general program (code optimization => application runtime optimization).

2.2.5. Data Restructuring

Data restructuring was taken into consideration in the templates implementation and will be subject to our interest in the conversion from flat files to Database.

Before data restructuring can begin a reverse engineering activity, analysis of source code must be conducted. All programming language statements that contain data definitions, file descriptions, I/O and interface descriptions are evaluated. The intent is to extract data items and objects, to get information on data flow, and to understand the existing data structures that have been implemented. This activity is sometimes called data analysis [Ricketts et *al.* 1989].

2.2.6. Forward engineering of user interface

A model for reengineering user interfaces exists [Merlo et *al.* 1995].

- Understand the original interface and the data that move between it and the remainder of the application. If a new GUI is to be developed, the data that flow between the GUI and the remaining program must be consistent with the data that currently flow between the character-based interface and the program.
- Remodel the behavior implied by the existing interface into a series of abstractions that have meaning in the context of a GUI. A redesigned interface must still allow a user to exhibit the appropriate business behavior.
- Introduce improvements that make the mode of interaction more efficient. The ergonomic failings of the existing interface are studied and corrected in the design of the new GUI.
- Build and integrate the new GUI. The existence of class libraries and fourth generation tools can reduce the effort required to build the GUI significantly. However integration with existing application software can be more time-consuming. Care must be taken to ensure that the GUI does not propagate adverse side effects into the remainder of the application.

One thing that I like to mention before ending this chapter is that all projects I worked on involved software engineering principles:

- Project management concept

- Project planning
- Estimations
- Resources
- Risk management
- Project scheduling and tracking
- Software quality assurance
- Software testing methods
- Documentation
- Performance
- Regression testing

Metdemo is a demo based on the Metropolitan Life project. What was the purpose behind this demo and what challenges did we as a team face to accomplish this demo and me in particular? This will be the subject of the following chapter.

Chapter 3. BILINGUAL VERSION OF METLIFE (METDEMO)

Metdemo is the bilingual version of an insurance related project "Metlife", which allows agents to create Insurance Illustrations to the clients of the Metropolitan Life Insurance Company. The goal behind Metdemo is to create a bilingual application that allows users to change from one language to the other at runtime.

We started from a modified version of Metlife, which was adopted to fit the size of a demo. We had to translate all the resources (from English to French). This was a long process that involved three people, working non-stop for a month.

Having the resources translated, we had to find a way to make the system switch between languages (change from English to French and vice versa) at runtime.

In the first version, users had to restart the application so that they change languages. The trick is to access the "ML.ini" file where some specific variables are changed depending on the specified language.

Below are blocks of code used for this purpose, as well as a description of each block.

```
//---------- Modifications brought in to make the system bilingual ------
// Change language to english
        EV_COMMAND( CM_ENGLISH, CMChangeLang ),
        EV_COMMAND_ENABLE( CM_ENGLISH, CMEnglishEnable ),

// Change language to french
        EV_COMMAND( CM_FRENCH, CMChangeLang ),
        EV_COMMAND_ENABLE( CM_FRENCH, CMFrenchEnable ),
```

- EV_COMMAND( ) and EV_COMMAND_ENABLE( ) are menu related functions.
- EV_COMMAND( ) is used to handle the event (CM_ENGLISH or CM_FRENCH) of choosing a language (English or French) from the main menu.
- EV_COMMAND_ENABLE( ) enable the menu item…
- CMChangeLang is the function that handles changing the language from English to French and vice versa.

Figure 5 shows the toolbar menu option that is associated with these events and functions.

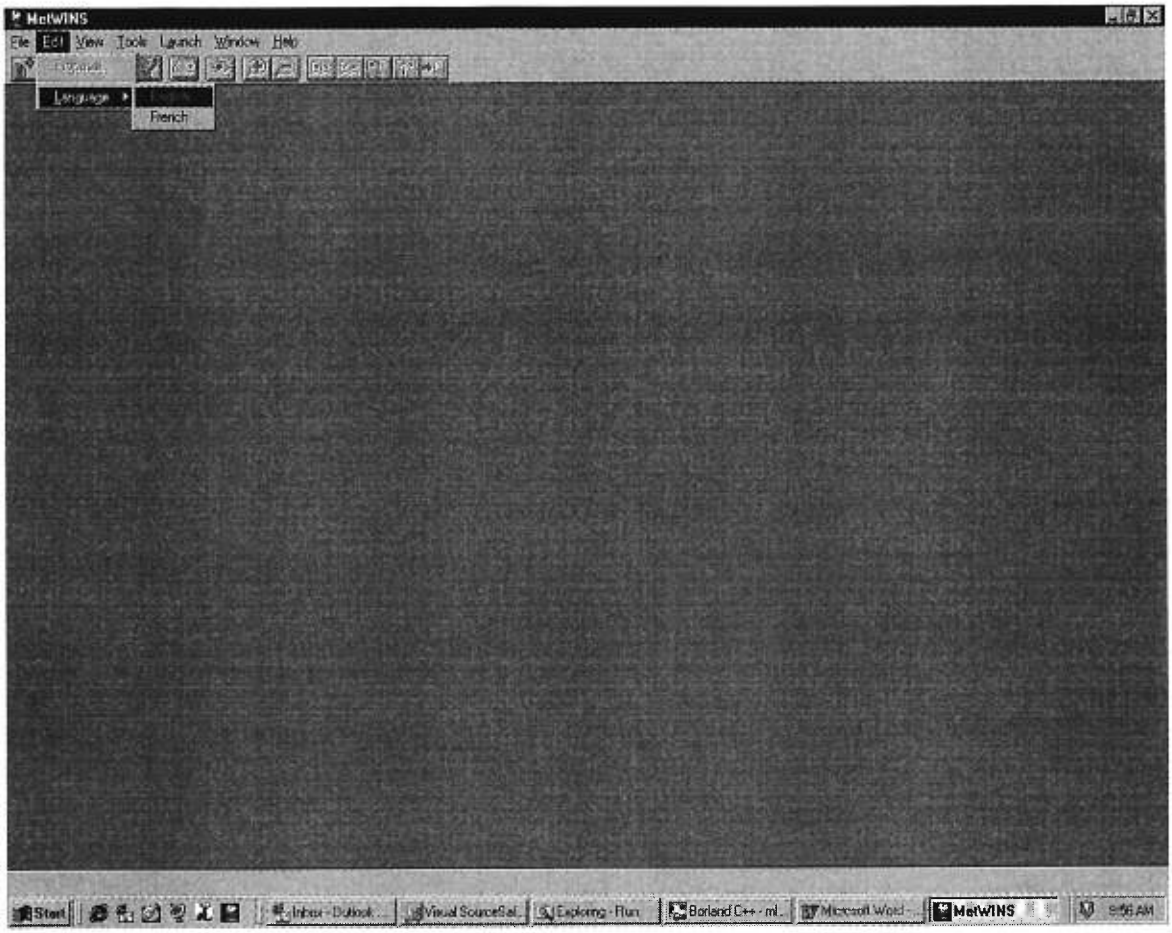

Figure 5. Metdemo Toolbar Menu (shows enabled language option).

I had to find a way to specify to the application which resources to load. I created a flag in the INI file (ML.ini). This flag is tested when the application is first loaded and each time the user chooses to change the language from the menu bar, an event is launched and handled by a function that tests the value of the INI flag and loads the adequate resources.

The following lines of code handle "ML.ini" file, where specific variables were added in order to test the user's choice. Depending on whether he/she chooses English or French the value of the Language variable is 1 for English and 3084 for French. (These values where assigned respecting a certain standard issued by our partner "Navisys" in the United States.)

```
[APP]
Language=1
ReportLang=1
```

The value of "Language" is affected to "sLang" (which is a string defined in our application just for the purpose of including the value of Language) using GetPrivateProfileString( ) [API, 1999], and GetDLocale()->Load( ) functions.

Knowing the value of sLang, special tests on which files and DLLs to be loaded are required depending on the user's choice.

```
// Modif. for Bilingual ----------------------------------------
char sLANG[80];
GetPrivateProfileString( "APP", "Language", "", sLANG, 79, INI_NAME );
GetDLocale()->Load( atoi( sLANG ) );
//Each time you change the librairy check the index he can be 1 or 2 or...
if( atoi( sLANG ) == LANG_CANFRENCH )
        GetDLocale()->SetTypecodeIndex( 1 );
GetDLocale()->Save();
// Instantiate a Rule Session
if( GetDLocale()->GetLanguage() == LANG_CANFRENCH )
        pRuleSession = new GAppRuleSession ("MLFR.EBR");
else
        pRuleSession = new GAppRuleSession ("MLENG.EBR");

pResourceDll = NULL;
pMDIFrame = NULL;
```

- GetPrivateProfileString() allows reading from an "ini" format file.
- GetDLocale() points to a "Rec" (special memory block) which is used to store parameters related to the chosen language.

Since the content of Listboxes is generated at runtime by pointing to type code set elements in the .TCC file created by ECL Admin (we will explain this in details in the chapter that deals with Logisil software environment), we had to find a way to distinguish between English and French element description. This was handled by using an index. It is set to "0" for the English version and "1" for the French version. SetTypecodeIndex(1) is used to modify the index of the type code sets so that it points to the second element definition (which in this case is the French version).

Depending on the chosen language the corresponding rule session is instantiated. After that the resource and frame pointers are initialized to NULL.

Figure, 6 presents how the application handles the language switching for this first version as well as the second version, which is also, discussed in this chapter.

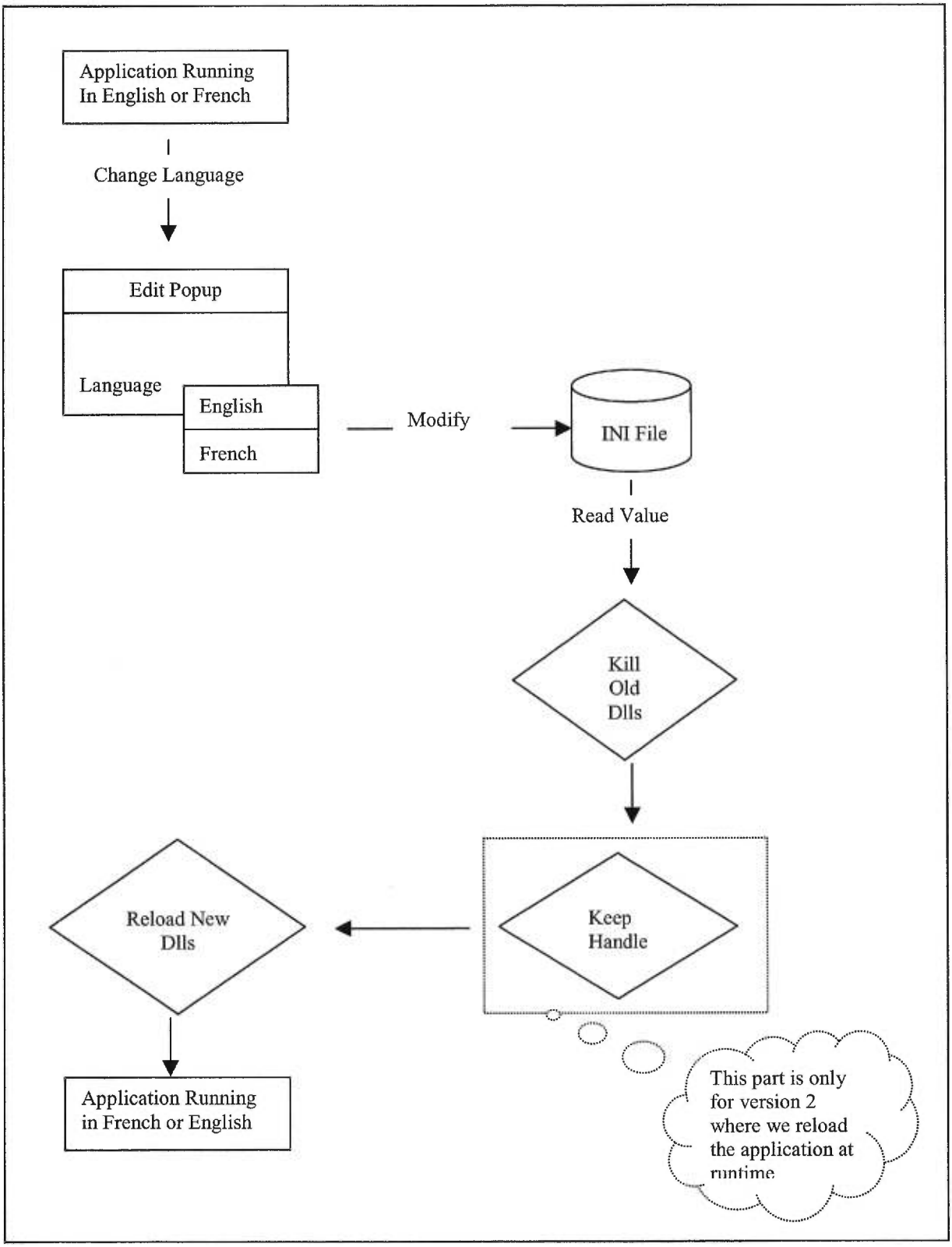

Figure 6. Metdemo Diagram.

The second version raised a bigger challenge. We had to reload the resources at runtime. We had to find a way to delete (unload) the French Dlls and reload the English Dlls and vice versa. This could be done by using the Object windows related procedures (LoadLibraries() & FreeLibraries()). We modified (InitInstance()) in the libraries so that it handles the bilingual version. LoadLibraries() & FreeLibraries() were added to this function which is always called by object windows when the application is executed. The major challenge was to reinitialize and load all the application parameters. The modifications brought to this function allowed it, on language change, to free the DLLs of the current application language and load the ones of the user specified language.

InitInstance() is Called by ObjectWindows when the application is initialized. If this method is overridden, the base class method must be called. This method calls, in order, **InitResourceDll**(), **InitDocManager**(), **InitControlDocument**(), **InitSession**(), the inherited **InitInstance**(), and then the **LogOn**() method of the session (initialization method). It launches a CM_STARTUP event, which is used to trigger the startup dialog.
After initializing the application the correspondent resource DLL is loaded using:

```
if( GetDLocale()->GetLanguage() == LANG_CANFRENCH )
{
return pResourceDll = new DOwlDllLoader( "ML32RCF.DLL" );
}
else
{
return pResourceDll = new DOwlDllLoader( "ML32RC.DLL" );
}
```

Unfortunately for this version we couldn't finish the help of the French application. In order to fill in this gap we included a dialog that showed the message "under construction!".

```
//-------- help dialog for the french version ------------------------------
void GApplication::CMHelpSearch()
{
if( GetDLocale()->GetLanguage() == LANG_CANFRENCH )
{
Ddialog dHelp( pControlDocument, GetMainWindow(), "HelpConstruction" );
dHelp.Center();
dHelp.Execute();
}
InvokeHelp( HELP_PARTIALKEY );
}

//---------------------------------------------------------------------------
void GApplication::CMHelpContents()
{
if( GetDLocale()->GetLanguage() == LANG_CANFRENCH )
{
DDialog dHelp( pControlDocument, GetMainWindow(), "HelpConstruction" );
dHelp.Center();
dHelp.Execute();
}
InvokeHelp( HELP_CONTENTS );
}

//---------------------------------------------------------------------------
void GApplication::CMHelpUsingHelp()
{
if( GetDLocale()->GetLanguage() == LANG_CANFRENCH )
{
DDialog dHelp( pControlDocument, GetMainWindow(), "HelpConstruction" );
dHelp.Center();
dHelp.Execute();
}
InvokeHelp( HELP_HELPONHELP );
        }
```

The following function handles the event that is triggered when the user chooses the change language option from the menu.

```
void GApplication::CMChangeLang()
{
char s1[150];
char s2[150];
if( GetDLocale()->GetLanguage() == LANG_CANFRENCH )
{
strncpy( s1, "Vous êtes sur le point de changer le langage de votre application (en
Anglais).\n\nEtes vous sûr?", 149 );
strncpy( s2, "Changer langage" , 149 );
}
else
{
strncpy( s1, "You are about to change your application language (to French).\n\nAre you sure?",
149 );
strncpy( s2, "Change language" , 149 );
}
char buffer[7];
char sLANG[80];
GetPrivateProfileString( "APP", "Language", "", sLANG, 79, INI_NAME );
if( MessageBox( s1, s2, MB_YESNO|MB_ICONQUESTION ) == IDYES )
{
if( atoi( sLANG ) != LANG_ENGUSA ) // (1)
{
itoa( LANG_ENGUSA, buffer, 10 );
ChangeLanguage( buffer );
}
else // (1)
if( atoi( sLANG ) != LANG_CANFRENCH ) // (2)
{
itoa( LANG_CANFRENCH, buffer, 10 );
ChangeLanguage( buffer );
}
else // (2)
{
MessageBox( "Language not handled!!!", "ERROR!" );
                itoa( LANG_ENGUSA, buffer, 10 );
                ChangeLanguage( buffer );
}
}
}
```

This function stops the user and asks him if he is sure he wants to change the language. If he confirms (validates) his choice, ChangeLanguage() is called. This function will redo all the steps that are required to reinitialize the system, and set all the parameters of the chosen language, then re-executes the application using ShellExecute().

```
//---- In order to reload with language change ----------------------
void GApplication::ChangeLanguage( char* buffer )
{
WritePrivateProfileString( "APP", "Language", buffer, INI_NAME );
char sLANG[80];
GetPrivateProfileString( "APP", "Language", "", sLANG, 79, INI_NAME );
        GetDLocale()->Load( atoi( sLANG ) );

//Each time you change the library check the index he can be 1 or 2 or...
if( atoi( sLANG ) == LANG_CANFRENCH )
GetDLocale()->SetTypecodeIndex( 1 );
GetDLocale()->Save();

//Check for Home Office Mode
char sHOPW[80];
GetPrivateProfileString( "SYSTEM", "HOPW", "", sHOPW, 79, INI_NAME );
bHomeOfficeMode = !strcmp(sHOPW, "OKTOEDIT");

// Instantiated a Rule Session
if ( pRuleSession )
        delete pRuleSession;
if( GetDLocale()->GetLanguage() == LANG_CANFRENCH )
        pRuleSession = new GAppRuleSession ("MLFR.EBR");
else
        pRuleSession = new GAppRuleSession ("MLENG.EBR");
if ( GetPrivateProfileInt( "SYSTEM", "Disable Conversion", 0, INI_NAME ) )
        DisableDataConversion( TRUE );
pMDIFrameTemp = pMDIFrame;

//------------------ Reload the application -------
HWND pParamWindow;
HINSTANCE hStartup;
hStartup = ShellExecute(pParamWindow, "open", "ML32.exe", "", "", SW_SHOWMAXIMIZED);
// Delete the old Pointer = clean the old application before language change
delete(pMDIFrameTemp); }
```

PS: In the first version, CMChangeLang( ) only changed the values of the variables in the "ML.ini" file. The user had to restart the application to switch languages.

ChangeLanguage( ) is the function where:

- The type code sets index is reset. (GetDLocale()->SetTypecodeIndex(1)).
- The rules are reset. (pRuleSession = new GAppRuleSession ("MLFR.EBR")).
- The old frame is deleted:
  - pMDIFrameTemp = pMDIFrame;
  - delete(pMDIFrameTemp);
- The resources are reloaded at runtime. This is done by ShellExecute() [Newsgroups, 1999] which reopens the application. This force a call to InitInstance().

The following functions enable or disable the English or French Menu Items depending on the current language.

```
//------------------------------------------------------------------------------
void GApplication::CMEnglishEnable( TCommandEnabler& ce )
{
char sLANG[80];
GetPrivateProfileString( "APP", "Language", "", sLANG, 79, INI_NAME );
if( atoi( sLANG ) != LANG_ENGUSA )
        ce.Enable( TRUE );
else
        ce.Enable( FALSE );
}
//------------------------------------------------------------------------------
void GApplication::CMFrenchEnable( TCommandEnabler& ce )
{
char sLANG[80];
GetPrivateProfileString( "APP", "Language", "", sLANG, 79, INI_NAME );
if( atoi( sLANG ) != LANG_CANFRENCH )
        ce.Enable( TRUE );
else
        ce.Enable( FALSE );
}
```

What was described in a few pages is the fruit of five months of hard work. Details on the terminology used in this chapter will follow. The second version was also known under *Automatic Language Switching (ALS)*.

This project helped me develop managerial skills. It made me realize that sometimes decisions are taken based on feasibility and time constraints. So the adopted solutions might not be the best ones available. On a technical level, the adopted solution proves that changing from one language to the other is done by simply deleting the resource DLL that points to the current language (e.g. ML32RCF.DLL) and reloading the DLL of the selected language (ML32RC.DLL). This is done at runtime and it also involves:

- Deleting the current application frame
- Resetting the rules session
- Reloading (re-executing) the application using ShellExecute(...)

ShellExecute(...) allows re-executing the application from within, so that all required initializations are done, and without losing the handle of the application. It's an automated process that only requires a few mouse clicks.

Next I will introduce Hartford Life project and the features that I added to this project.

Chapter 4. HARTFORD LIFE PROJECT (MERLIN)

Hartford Life involved the implementation of several features. The first feature was templates implementation. Templates are default values provided by Hartford to their agents to help reduce their workload. Templates implementation involved a few complications that we were able to overcome as a result of a mutual agreement that we reached with our client. This is due to the good relation that is established between our management and Hartford general management.

The second feature is "Compare to", which allows agents to compare results of different insurance products and concepts using the same client data by a simple three mouse clicks procedure.

Work on Merlin also involved problem solving. The agent problem involved a mismatch between physical ids and memory ids, which made the agents, get assigned to the wrong proposals.

4.1. Templates Implementation

The templates provide users (agents) with default values. Instead of having to input data related to cases that are more likely to be reproduced with multiple clients, users can save this data and just select the corresponding template. Two sorts of templates are introduced: the market templates and the strategy templates. The market templates are related to concepts while strategy templates are related to products. So templates help reduce repetitive tasks.

The way templates are implemented in the Navisys system (system that is considered to be the base ground for all projects) restrict their portability.

In the Navisys illustrator both market and strategy defaults are saved in the proposal files. This creates certain limitations because templates are saved using the same files that store proposal related information.

Hartford general management thought of templates as a dynamic means of providing users with defaults. So they planed to create templates and include them in their new release (Templates version). If templates were saved in the proposal file, a new release would make the users lose their saved proposals. Since templates don't use most of the fields that are usually used in the proposal record, this creates space loses.

The solution was to create separate files for the templates. We actually created market files and strategy files that were added to the already existing proposal files. This maximized record space usage.

Changes were done to several files in the application, we recall:

```
//App.cpp
#ifdef USE_STRATEGY
DEFINE_DOC_TEMPLATE_CLASS( GProposalStrategyDocument, GProposalStrategyView,
StrategyTemplate );
StrategyTemplate gStrategyTemplate( "Proposal Strategy", "*.STY", 0, "STY", dtAutoOpen | dtAutoDelete
);

DEFINE_DOC_TEMPLATE_CLASS( GMarketStrategyDocument, GMarketStrategyView,
MarketTemplate );
MarketTemplate gMarketTemplate( "AM Strategy", "*.MKT", 0, "MKT", dtAutoOpen | dtAutoDelete );
#endif

// Function that gets called for the startup dialog and defines the actions associated to the user selection

void GApplication::NewStartup()
        {
        switch( GNewProposalDialog( "ProposalNewStrategy" ).Execute() )
                {
                case IDOK:
                        if ( pControlDocument->PeekInt( GD_ActionNew ) )
                                {
                                switch( pControlDocument->PeekInt( CT_PropType ) )
                                        {
                                        case TC_PropIndiv:
                                                {
                                                SalesStrategy = FALSE;
                                                SalesMarket = FALSE;
                                                DRec tempkey( "PKEY" );
                                                tempkey.PokeString( KS_PlanId,
                                                pControlDocument->PeekString( IS_NewPlanId ) );
                                                OpenNewDocument( GetTemplateType( & tempkey,
                                                TC_Proposal, FALSE ).c_str() );
                                                MainWindow->PostMessage( WM_COMMAND,
                                                CM_EDIT );
                                                }
                                                break;
                                        case TC_PropGroup:
                                                OpenNewDocument( GetTemplateType( NULL,
                                                TC_Group, FALSE ).c_str() );
                                                MainWindow->PostMessage( WM_COMMAND,
                                                CM_EDIT );
                                                break;
                                }
                        }
                        else if ( pControlDocument->PeekInt( GD_ActionExisting ) )
                                {
                                switch( pControlDocument->PeekInt( CT_PropType ) )
                                        {
```

```
                                        case TC_PropIndiv:
                                                MainWindow->PostMessage( WM_COMMAND,
                                                CM_FILEOPENPROPOSAL );
                                                break;
                                        case TC_PropGroup:
                                                MainWindow->PostMessage( WM_COMMAND,
                                                CM_FILEOPENGROUP );
                                                break;
                                        }
                                }
                        break;

                case GP_OpenAgent:
                        MainWindow->PostMessage( WM_COMMAND, CM_FILEOPENAGENT );
                        break;

                case GP_OpenBusiness:
                        MainWindow->PostMessage( WM_COMMAND, CM_FILEOPENBUSINESS );
                        break;

                case GP_AddBusiness:
                        MainWindow->PostMessage( WM_COMMAND, CM_FILENEWBUSINESS );
                        break;

                case GP_AddAgent:
                        MainWindow->PostMessage( WM_COMMAND, CM_FILENEWAGENT );
                        break;

                case GP_AddMarket:
                        {
                        SalesMarket = TRUE;
                        MainWindow->PostMessage( WM_COMMAND, CM_FILENEWMARKET );
                        }
                        break;

                case GP_AddStrategy:
                        {
                        SalesStrategy = TRUE;
                        MainWindow->PostMessage( WM_COMMAND, CM_FILENEWSTRATEGY );
                        }
                        break;

                case GP_OpenMarket:
                        SalesMarket = TRUE;
                        MainWindow->PostMessage( WM_COMMAND, CM_FILEOPENMARKET );
                        break;

                case GP_OpenStrategy:
                        SalesStrategy = TRUE;
                        MainWindow->PostMessage( WM_COMMAND, CM_FILEOPENSTRATEGY
);
                        break;
                }
        }
```

Basically the code displayed above shows us how documents, views and templates related to a concept and a product are defined. We can also see the associated events that are related to opening an existing template (GP_OpenStrategy, GP_OpenMarket) and creating a new one (GP_AddStrategy, GP_AddMarket).

"LoadDefaults" function was modified by adding a break (return TRUE) so that template values are not overridden. "LoadDefaults" handles loading default values to fields.

Changes were also made to "Startup.cpp" where we defined the tables (DMemRecSets) that were related to showing the corresponding templates for selected products and concepts in the startup screen.

```
//Startup.cpp
                case GY_NewMarketCombo:
                        pMarketBox = TYPESAFE_DOWNCAST( Context.GetWindowPtr(),
                        DQueryComboBox );
                        PRECOND( pMarketBox );
                        pMarketBox->SetDataSource( & Markets );
                        pMarketBox->SetTemplate( "~KS_MSaveAs" );
                        break;
```

Two files (Market.cpp and Strategy.cpp) were created. They handled the classes in charge of the documents (Data Warehouses) related to the market and strategy templates.

Here below some bits of code taken from Market.cpp. They basically handle linking the general proposal document to the corresponding advanced market related document.

```
//---------------------------------------------------------------------------
// GMarketStrategyDocument -- Implementation
//---------------------------------------------------------------------------
GMarketStrategyDocument::GMarketStrategyDocument( TDocument * parent ) :
        FMarketStrategyDocument( parent )
        {
        }
//---------------------------------------------------------------------------
GMarketStrategyDocument::~GMarketStrategyDocument()
        {
        }
//---------------------------------------------------------------------------
DRec * GMarketStrategyDocument::GetMainRec()
        {
        PRECOND( pProposalDocument );
        PRECOND( pProposalDocument->GetAMDocument() );
        FAMDocument * pAMDocument = TYPESAFE_DOWNCAST( pProposalDocument-
>GetAMDocument(), FAMDocument );
        PRECOND( pAMDocument );
        return pAMDocument->GetAMProp();
        }
//---------------------------------------------------------------------------
```

Another obstacle we had to overcome was the need to keep the templates in future releases. This could cause losing defaults created by the agents. The idea behind that was to provide users who didn't install the first release, with the defaults they were supposed to get had they installed the templates version.

To solve this problem we came with the idea of creating a C++ program (Class) that checks if the templates are already installed. If they are, the program disregards copying the required templates files. If not the program copies those files to the cases folder, specified in the application initialization file.

```
//--------------------------------------------------------------------------
// batchApp
// Function that ovewrites Product and Strategy file Templates if they are empty
//--------------------------------------------------------------------------
void batchApp::FileUpdate ()
        {
        struct ffblk ffblk;
        char * ini_path;
        char szDataPath[ 80 ];
        int iCDDrv;
        OFSTRUCT ofStrSrc;
        OFSTRUCT ofStrDest;
        HFILE hfSrcFile, hfDstFile;
        char sSrcFile[ 100 ];
        char sDstFile[ 100 ];
        ini_path = searchpath( "Merlin.ini" ); //Search for the INI file
        //Get the Data path from the INI file
        GetPrivateProfileString( "PATHS", "DataPath", "", szDataPath, sizeof( szDataPath ), ini_path );
        //Check if the files are there and if not or if their size indicates that they are empty
        //copy them to the data folder "cases"
        BOOL done;
        iCDDrv = getdisk();
        setdisk( GetDrv ( szDataPath ) );
        chdir( szDataPath );
        // We only test Market.id3
        // If this file doesn't exist or its size is = 8196 (empty) we will copy
        // Market.id3, Market.ik3, Mkey.id3, Mkey.ik3 from the Installation cd to
        // our current cases folder
        done = findfirst( "Market.id3", &ffblk, 0 );
        if ( ( !done && ffblk.ff_fsize == 8196 ) || done )
                {
                // Market.id3
                strcpy( sSrcFile, GetInstDrv( iCDDrv ) );
                strcat( sSrcFile, "\\Templates\\Market.id3" );
                strcpy( sDstFile, szDataPath );
                strcat( sDstFile, "\Market.id3" );
                hfSrcFile = LZOpenFile( sSrcFile, &ofStrSrc, OF_READ );
                hfDstFile = LZOpenFile( sDstFile, &ofStrDest, OF_CREATE );
                LZCopy( hfSrcFile, hfDstFile );
                LZClose( hfDstFile );
                LZClose( hfSrcFile );
                }
        //Same thing for the rest of the files
        ...
```

To make our templates similar to the basic system we had to find a way of removing the report background (View) that is generated each time we created a new template. To do this we did the following:

When (string GApplication::GetTemplateType(DRec * pRec, int nDocType, BOOL bHidden)) is called, BOOL variable is set to True (it specifies if we want a hidden view or not). But the problem is that if this BOOL variable is set to true there wouldn't be a view to pick up the message posted by CM_EDIT: *MainWindow->PostMessage(WM_COMMAND, CM_EDIT).* So the result would be a blank screen since there is no view to pick up the message. The solution is to call a specific function that handles the event. So instead of posting a message, this particular function is called as follows (EditMarket(-1L)).

```
BOOL GApplication::EditMarket( DRecId id )
{
BOOL bMarketDocOK = FALSE;
DDocument * pMarketDoc;
string szDocType;
//Save Current Product, and set Product type to NONE so we recognize this as a Market template....
int nSaveProduct = pControlDocument->PeekInt( CT_NewProduct );
pControlDocument->PokeInt( CT_NewProduct, PRODUCT_NONE );
//Valid ID passed, try to Open existing Market Doc
if( id > -1L )
        {
        //Look for the exisiting document in the PKEY table
        DRecSet * pPropKey = SESSION.GetPKeyTable();
        if ( pPropKey->Exec( id ) ) // ODBC - Recreate the set with the required //ID in it
        if ( pPropKey->Goto( id ) ) //Position yourself on the existing Market Doc //record
                {
                //Get the Document type with a Hidden View
                szDocType = GetTemplateType( pPropKey, TC_Proposal, TRUE );
                pMarketDoc = OpenExistingDocument( id, szDocType.c_str() );
                bMarketDocOK = TRUE;
                }
        }
//If no valid Document found, Create new Market Doc
if( bMarketDocOK == FALSE )
        {
        //Set a temporary PKey Record so we can create a new MarketDoc
        DRec tempPKey( "PKEY" );
        tempPKey.PokeString( KS_PlanId, pControlDocument->PeekString( IS_NewPlanId ) );
        //Get the Document type with a Hidden View
        szDocType = GetTemplateType( & tempPKey, TC_Proposal, TRUE );
        pMarketDoc = OpenNewDocument( szDocType.c_str() );
        }
        //Now, show market template dialogs
        GAdvancedDialog d( TYPESAFE_DOWNCAST( pMarketDoc, GProposalDocument ),
        APP.GetMainWindow(),"", TRUE );
        AddAdvanceMarketTab( d, pMarketDoc );
        int nRez = d.Execute();
        if ( nRez == IDOK )
          pMarketDoc->Commit(); //Save Changes to disk
```

```
        delete  pMarketDoc; //Done with document, delete it.
        //RESET PRODUCT TYPE TO CORRECT ONE
        pControlDocument->PokeInt( CT_NewProduct, nSaveProduct );
        return nRez;
}
```

This function creates a new Market document.

The AddAdvancedMarketTab is the function where the tabs are added to the dialog (Tabs such as Insured and Case Input can be viewed in figure 7).

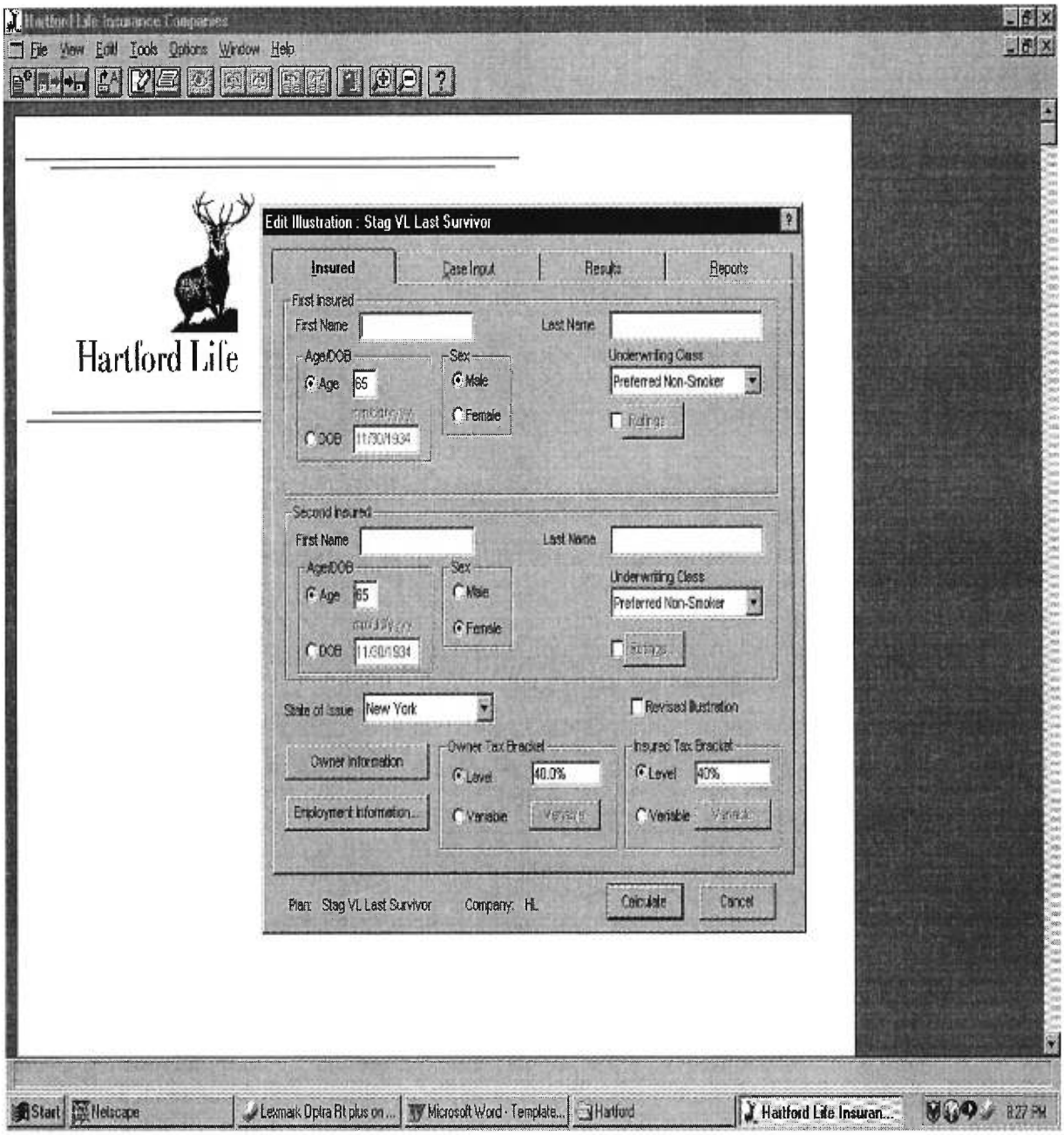

Figure 7. Merlin Tabs.

In the previously displayed code, I first experienced a crash when I tried to delete a document. After analyzing I reached the following deductions:

- If you are deleting a document and you suddenly crash you better:
  - Make sure that you didn't ADD a container to the Data Manager.
  - This container could be static or have another type... It will get attached to your document so when you delete the document the container is still there and leads to a crash!
- So you better delete the container first then delete your document.

This could be compared to memory leaks in C++.

Figure 8 shows us how a document can be related to multiple containers that ought to be deleted before deleting the document which itself is composed of multiple containers. It illustrates the separation between the document and the container.

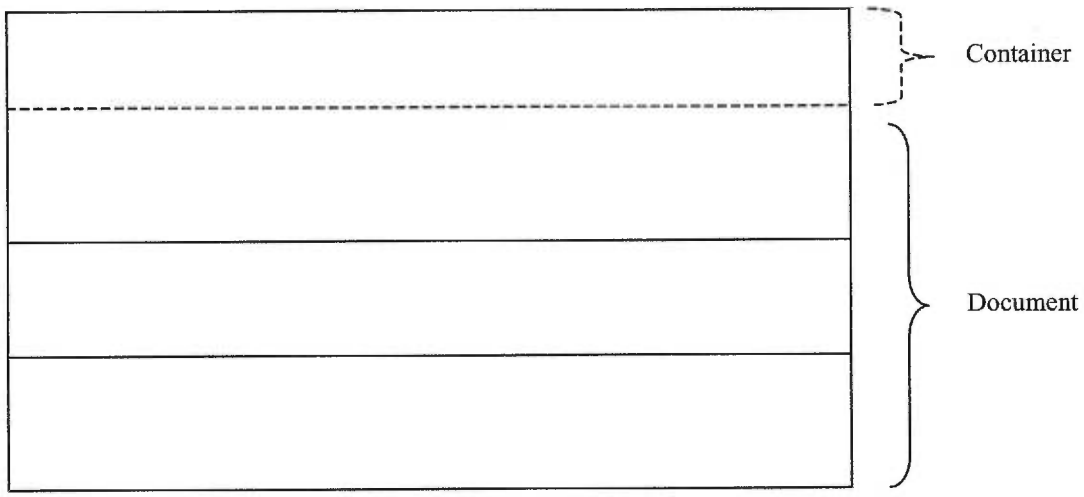

Figure 8. Documents & Containers.

So always delete containers before deleting your document:

```
//Declaration
DDocument * pStrategyDoc;
GAdvancedDialog * pAdvancedTab =
                    new GAdvancedDialog( pDoc, APP.GetMainWindow(),          "", TRUE );

//First, Delete the tab Dialogs
//Note the Document MUST be deleted second, in case the Dialogs have attached
//any temporary datamanagers to it (e.g. DREC may have been added by the dialogs)
delete  pAdvancedTab;

//Now, Delete the Document
delete  pStrategyDoc; //Done with document, delete it.
```

My second assignment was the implementation of the compare to feature, which was highly regarded by our client.

4.2. "Compare to" feature

Agents wanted a feature that allows them to view the results of a particular insurance case using a different insurance concept and product. In other words, after selecting a certain product from the startup dialog and adding all the required information related to the insured, the desired benefits, and riders… the agents wanted to be able to change the insurance product and concept so that they compare results using the same inputs. A dialog was created for this purpose allowing users to change the concept and related product. Results were displayed using the previously selected inputs. The behavior of this feature is presented in figure 9.

The complexity of this option prevail in the fact that after generating reports for a specific product, unsaved information has to be used to generate reports with a different product. Both reports have to be kept active allowing the user the do a visual screen comparison and choose to save, print or cancel any of the active insurance illustrations.

After doing the required analysis, we ended up with two solutions.

- The first was to try to recreate a new instance of the object (Document that contains all the information related to the insurance illustration "GproposalDocument") and just change the concept and product and recalculate the results. The problem in implementing this solution was the unfeasibility of recreating a new instance of a complicated object in C++. Java allows the recreation of object instances using a specific function but not C++.
- The second solution was similar to the first but instead of trying to recreate a new instance of the generated object, we saved it, reloaded it using a new instance where we replaced the concept and product with the selected ones, and redisplayed the results by launching a calculate event.

It took us a couple of weeks to discover and create functions that allowed us to attend the required results. Here below the code used for this feature. Some of the functions aren't displayed because of the Copyrights that protect Navisys libraries code.

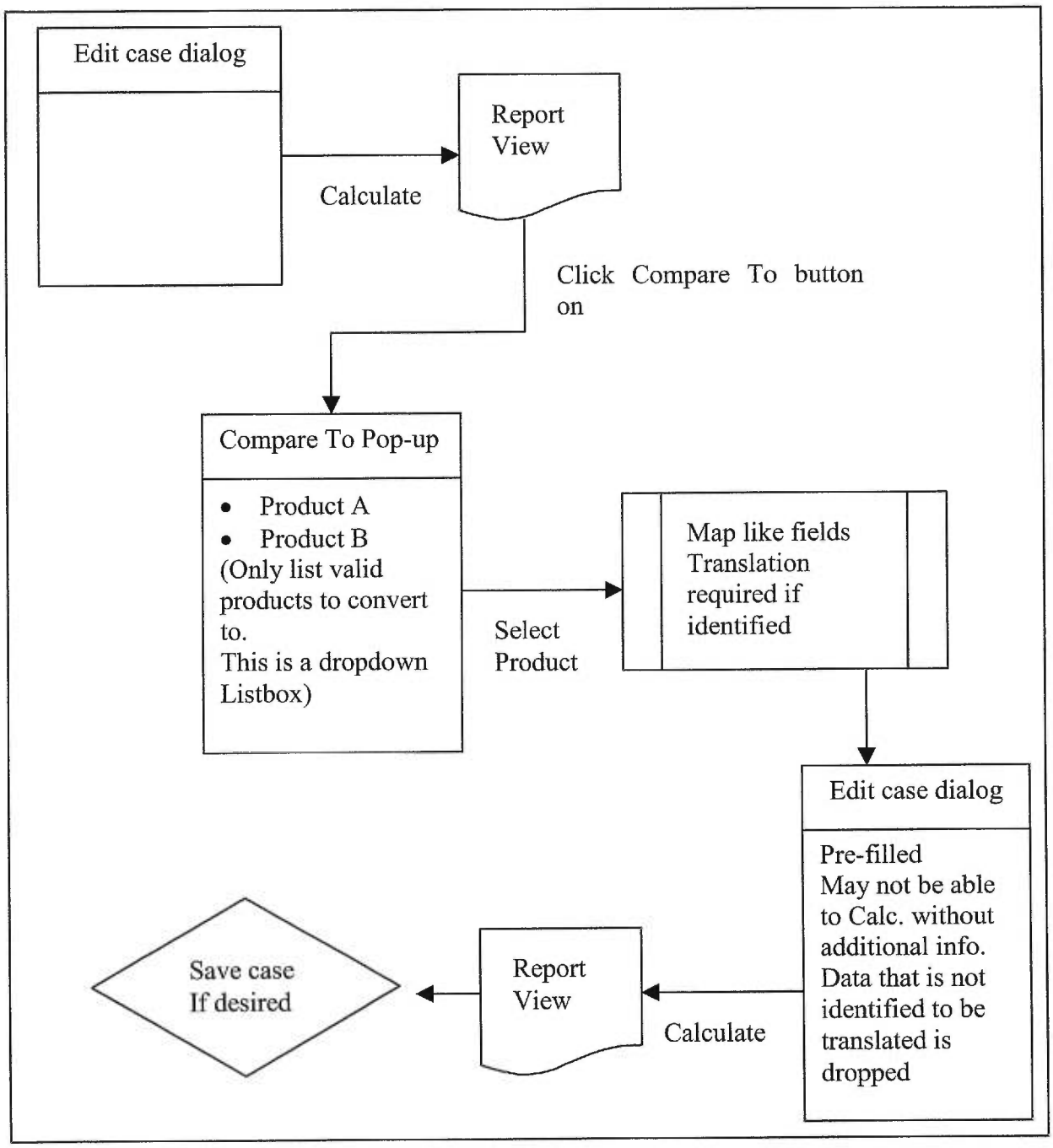

Figure 9. Compare To feature behaviour.

We had to modify and add the following functions in the "Proposal.cpp" file.

```
//-------------------------------------------------------------------------------------------
// I.E. 03/11/99 Implementing the Compare To, this leads us to change the
// handling of the Save As function to just allow to save the proposal with another name ...
//-------------------------------------------------------------------------------------------
BOOL GProposalDocument::SelectSaveAs()
        {
        string str = PeekString( KS_SaveAs );
        if ( ! PeekString( KS_SaveAs ).length() )
                PokeString( KS_SaveAs,
                        PeekString(KS_FirstName) + " " + PeekString(KS_LastName) );
        if (GetDocPath() == 0)  // never saved
                {
                if ( GSaveAsDialog( this, "ProposalSaveFirst" ).Execute() == IDOK )
                        return TRUE;
                }
        else
                {
                if ( GSaveAsDialog( this, "ProposalSaveAs" ).Execute() == IDOK )
                        return TRUE;
                }
        PokeString( KS_SaveAs, str );
        return FALSE;
        }
```

"SelectSaveAs" was modified because in previous versions it allowed users to save an illustration using a different product and once they reloaded it (reopened the proposal), it appeared with the saved product and corresponding results. The modifications to this function make it just rename a saved illustration.

"SelectCompareProduct" is the main function that handles the events related to the "Compare to" feature. The code in this function is unorthodox in the sense that it doesn't really comply with the standards and conventional methods adopted by Logisil and Navisys.

Next the code of "SelectCompareProduct" is displayed with comments that allow the reader, who is familiar with our methods and way of coding, to understand how this feature is implemented.

```
//-------------------------------------------------------------------------------
// I.E. 03/11/99 Compare To product allows to view the current proposal with a
// different product ...
//-------------------------------------------------------------------------------
BOOL GProposalDocument::SelectCompareProduct()
        {
        // Keeping old info so that they won't be lost during this process
        string sOldConcept = pRuleSession->GetConceptTable()->
                                                PeekString( CS_GroupReportClass );
        string sOldName = PeekString( KS_SaveAs );
        string sOldPlanId = PeekString( KS_PlanId );
        string sOldClass = pRuleSession->GetPlanTable()->PeekString( CS_DocDialogClass );
        string sOldProdName = pRuleSession->GetPlanTable()->PeekString( CS_ShortPlanDescription );
        int nOldConcept = PeekInt( KT_Concept );
        int nOldProduct = PeekInt( KT_Product );
        //Test to see if the Proposal is already saved
        if( ! PeekString( KS_SaveAs ).length() )
                PokeString( KS_SaveAs, "  " );
        else
                PokeString( KS_SaveAs, PeekString( KS_SaveAs ) + "  " );

        if ( GSaveAsDialog( this, "ProposalCompareProduct" ).Execute() == IDOK )
                {
                ChangeToSaveAs();       // make it a new proposal
                Commit();               // save proposal
                long lOldId = PeekLong(KL_PKEYId);
                SetAutoStartup( FALSE ); // disable startup dialog
                TMDIFrame * pFrame = TYPESAFE_DOWNCAST( APP.MainWindow, TMDIFrame );
                PRECOND( pFrame );
                TWindow * pOldMDIChild = pFrame->GetClientWindow()->GetActiveMDIChild();
                PRECOND( pOldMDIChild ); // get pointer to old MDI Child Window
                pOldMDIChild->SetWindowText(sOldProdName.c_str()); // Assign the first window a
                                                                   // Title
                APP.GetRuleSession()->PlanGoToPlanID( PeekString( KS_PlanId ) );
                long lTempKey = GetPrimaryKey();
                SetPrimaryKey( -1L );   // to trick IApplication if not we would get a
                                        // message saying that is is already opened...
                GProposalDocument * pDocument =
                                TYPESAFE_DOWNCAST
                                        (
                                        APP.OpenExistingDocument
                                                (
                                                lTempKey,
                                                APP.GetRuleSession()->GetPlanTable()->
                                                PeekString( CS_DocDialogClass ).c_str()
                                                ),
                                        GProposalDocument
                                        );

                //Force a delete of the temporary saved proposal
                SESSION.GetPKeyTable()->Goto( lOldId );
                SESSION.GetPKeyTable()->Delete();
                SESSION.GetProposalTable()->Goto( lOldId );
                SESSION.GetProposalTable()->Delete();
```

```
        if( pDocument )
                {
                pDocument->SetAutoStartup();
                // initialize the required reports for this proposal
                DRecSet * pReportTable = SESSION.GetReportTable();
                DBlobClipBuffer blob;
                pReportTable->Exec( "" );
                pReportTable->Rewind();
                //Get the reports that are specific to the selected product
                while( pReportTable->Skip() )
                        if( APP.IsReportValid( pDocument->PeekInt( KT_Concept ),
                           pDocument->PeekString( KS_PlanId ),pReportTable ) )
                                {
                                int nRepReq = pReportTable->PeekInt( Rpt_IT_ReportReq );
                                if( nRepReq == TC_RptReqAll ||
                                   nRepReq == TC_RptDefaulted )
                                        blob.Put( (long) pReportTable->GetId() );
                                }
                pDocument->Poke( IO_PrpReports, blob );
                APP.MainWindow->PostMessage( WM_COMMAND, CM_CALCULATE );
                //This section resets the first document
                this->ChangeToSaveAs();          // So that we can save again
                this->SetPrimaryKey( -1L );      // Trick the IApplication
                this->TDocument::SetDocPath(0); // Call SetDoc from Object Windows so that
                                                 // we set the Path to 0 in order to call the
                 // SelectSaveAs function (Called from the libraries)
                this->SetDirty(TRUE);    // This will activate the X (close document) on the
                                         // top right of the screen
                this->PokeString( KS_PlanId, sOldPlanId ); // So that we re-assign old Plan
                                                           // Because we are loosing it
                this->PokeInt( KT_Concept, nOldConcept );
                this->PokeInt( KT_Product, nOldProduct );
                this->GetRuleSession()->PlanGoToPlanID( this->PeekString( KS_PlanId ) );

                // This is where we reset the second one
                pDocument->ChangeToSaveAs();
                pDocument->SetPrimaryKey( -1L );
                pDocument->TDocument::SetDocPath(0);
                pDocument->SetDirty(TRUE);
                pDocument->GetRuleSession()->PlanGoToPlanID( pDocument->
                                                        PeekString( KS_PlanId ) );
                // Assign the second window a title
                TMDIFrame * pFrame = TYPESAFE_DOWNCAST( APP.MainWindow,
                                                        TMDIFrame);
                PRECOND( pFrame );
                TWindow * pOldMDIChild = pFrame->GetClientWindow()
                                                        ->GetActiveMDIChild();
                PRECOND( pOldMDIChild ); // get pointer to old MDI Child Window
                pOldMDIChild->SetWindowText(pDocument->
                                GetRuleSession()->GetPlanTable()->
                                        PeekString( CS_ShortPlanDescription ).c_str());
                }
        return FALSE;
        }
PokeString( KS_SaveAs, sOldName );
return FALSE;
}
```

The SetDocPath (indicates the path of the corresponding Document) function as well as SetDirty (indicates if set to true that the frame top left cross option is available) can both be found in [Borland, 1991].

In the file "propview.cpp" we had to add the following so that we associate an event to the Compare To option. When this option is selected from the menu toolbar a message is posted and then caught by the application. The function CMChangeProduct is associated to the event CM_PRODUCTCOMPARE that is related to this feature.

```
EV_COMMAND( CM_PRODUCTCOMPARE, CMChangeProduct ), // I.E. 03/11/99 Compare To feature
EV_COMMAND_ENABLE( CM_PRODUCTCOMPARE, CMChangeProductEnable ),
```

Sometimes features are added to please the client. This is the fun part of the job. Other times there are bugs in the system and the client isn't happy. This is when we should intervene in a very professional and fast way. This is when the client knows if he could or could not count on us.

4.3. Agent problem / Import Export problem

The problem is related to agent Ids. This was produced when trying to open a certain proposal. The proposal was assigned to the wrong agent. Opening existing proposals is handled using QueryComboBoxes that are related to DMEMRecSets (memory records specially adapted in the Navisys libraries).

A QueryListBox was required in order to manipulate the Agents (selection & sorting) for Hartford project. For QueryListBoxes the control is given a data source (usually a DMemRecSet or a RecSet). We had to distinguish between a QueryComboBox and a QueryListBox.

It is specified in the *ECL Class Library Programmer's Reference* at page 57 [ECL, 1999] that if the Id of the third record in the DMemRecSet is 22, and the user selects the third item, the value of the QueryComboBox field will be 22L; this is not the case for a QueryListBox.

So the problem was the discrepancy between the Ids of the DMemRecSet (specially introduced to sort the QueryListBox) and the Corresponding RecSet (related to the Agents table).

The solution was to modify the code in STARTDLG.CPP so that the Ids always match. In the Class constructor we tested to see if the active agent was properly set and if not we affected it with the first agent in the Table.

In the SetupControl function (called when a dialog is built, for each field or control) of the QueryListBox field the following was done:

- Test to validate the active Agent (necessary in the case of deletion…)
- If Agent not valid it would be affected with the first agent in the Table.
- In order to sort the Agents (in our case by last name) a DMemRecSet is created by copying data from the Agent's table.
- It is important, since the Ids are not the same in the Agent table and Agent set, to test if the Set Id corresponds to the Current Id. This is done while data is being copied to the Agent set. The context will be affected with the same Id which is = to the position of the Agent in the QueryListBox.
- The set is sorted afterwards.

In the ControlChange function (called for each field in a dialog when the control changes from one field to the other) we took the Agent Id by Poking (KL_AKEYId) to the CurrentAgent. This Id is the same as the one in the Agent table.

So we learned that:

- If you delete a record in a DRecSet, holes are created in the DRecSet. But since a DMemRecSet is always recreated in the memory, these holes will eventually disappear creating a discrepancy between the Ids of the DRecSet and the Ids of the DMemRecSet. So RecIds and Temporary SetIds are not handled in the same manner.
- It is important to know that the KL_AKEYId is the same for both the DRectSet and the DMemRecSet.
- The Ids after doing a sort are not reassigned. This is very good or else we would have had another problem.
- Sort can only be done through sets (DMemRecSet).

Another problem was that for certain agents the related proposals weren't showing in the open proposal dialog.

This is because the AgentID in the QueryListBox is not the same as the AgentID in the ComboBox.

The code before the fix was like this:

```
BOOL GOpenProposalDialog::AllowKey( DRec * pKey, int nCurrFilter )
{
if ( ( PeekLong( GY_AgentFilter ) != 0L ) &&
  ( pKey->PeekLong( KL_AgentId ) != PeekLong( GY_AgentFilter ) - 1L ) )
return FALSE;
...
```

So the filtering wasn't done properly. After the fix the code changed to:

```
BOOL GOpenProposalDialog::AllowKey( DRec * pKey, int nCurrFilter )
{
long lAgtTable = pKey->PeekLong( KL_AgentId ); // Agent Key related to pKey Table
//Position in AKey on the Filtering agent to get the id
AgentKeys.Goto(PeekLong( GY_AgentFilter ));
long lAgtFilter = AgentKeys.PeekLong(KL_AKEYId); // Agent Key Used for Filtering
//the Proposals
BOOL bTest = PeekLong( GY_AgentFilter ) != 0L; // is True if not all Producers are
//selected
// bTest1 Is False when AgentId in combobox == AgentId in QueryListBox
BOOL bTest1 = AgentKeys.PeekLong(KL_AKEYId) != pKey->PeekLong( KL_AgentId
);
if ( bTest && bTest1 ) // F & F == T  as well as T & T = T So Return False
        return FALSE;
//Ps: If either bTest Or bTest1 is False => One of the conditions is True so Continue
//Continue = Fill the QueryListBox
...
```

This function is called for each proposal so if the proposal meets the filtering criteria it is selected and if not we break. If we continue we fill the QueryListBox.

The Import/Export problem is related to Ids conversion. Flat files were considered because data was saved in flat files in Merlin while other projects involved databases. Since Ids are generated by *ECLAdmin* (special tool developed by Navisys and covered in the Logisil software environment chapter), we had to verify if our problem was directly related to ECLAdmin, the application itself, the way the conversion was handled or the flat files. After covering all the possibilities, we finally discovered a table duplication problem generated in ECLAdmin, which directly affected previously, saved proposals. This was also present in other projects but it wasn't triggered yet. Our finding was beneficial to other projects and it ended in an important improvement to ECLAdmin.

A small bug in ECLAdmin caused this problem. This taught us to question how to approach bugs. Sometimes the simplest things could cause serious and critical application misbehavior. It took us a while to solve this one. On a personal level it helped me understand the whole process of Ids conversion and how the links are made.

Working on Merlin made us refer to the GUI "golden rules". Golden rule two, especially relieving short-term memory, was the trigger of the "Compare to" feature.
Software engineering techniques, such as data restructuring, were applied in templates implementation.
This experience made me realize the importance and seriousness of working on real projects with deadlines. Bad analysis and implementation decisions can affect the whole project as well as the future and reputation of a company. Maintaining good relations with the clients and defining responsibilities are essential for successful projects.

What impact does code reengineering have on a project?

Chapter 5. CODE SIMPLIFICATION

We, at Logisil, used *code optimization* to refer to *code simplification*. This is possible, knowing the difference between code optimization (which doesn't involve source code modifications) and code simplification (reengineering).

Code optimization included two steps:

- Step 1 involved *code clean up*. By code clean up we mean both code display (comments…) and excluding redundancies, in other words trying to enjoy to the max object oriented features.
- Step 2 involved code optimization. Code optimization is the fact of arranging code or even rewriting blocks of code in order to reduce the application's response time. This is known as code reverse engineering.

Code clean up also involved two-steps. The first was code display, which intended to improve the shape and not the content. For that I followed Logisil standards. I must admit that this was a repetitive task. The only benefits of such a task is that it allows you to prepare for the second step because you would have gone through all the code (line by line). An example on how code should be is the following:

```
void MyFunction( int param )
    {                                  | DO NOT
      Code();                          | USE THIS
.....                                  | STYLE!!!
    }                                  |


    void MyFunction( int param )       | CORRECT WAY
    {                                  | OF PLACING BRACES.
      Code();                          | ALWAYS USE THIS
.....                                  | METHOD !!!
    }
```

The second step was trying to group all the common features (validations…) of the child classes (or objects) in the higher level (parent class or object).

Code optimization involved efforts in removing unnecessary validations and tests. It also involved grouping these tests and arranging blocks of code.

This increased the application's speed, and was appreciated by the users. It is important to note that sometimes, code optimization could be a complicated procedure especially if the client sets specific response times for certain tasks.

The application as a whole, included around 1,500,000 lines of code among which I had to clean about 100,000 and optimize around 5000. This wasn't very obvious and serious testing was required in order to verify that the system functionality was not affected. If we refer to the code reengineering section, we recall that most of the requirements needed to perform this task are very important since modifying application source code could lead to serious modifications in it's behavior. The person or team in charge of doing code optimization must have system specifications, good knowledge of the application and must implement testing strategies in order to verify that the application's behavior is respected. Regression tests are required in such cases.

This task made me realize the seriousness of working on big projects and the implications of code modifications. Regression tests were done after code optimization and we discovered that some system functionality was broken. Testing is crucial in the software development process.

Next we will introduce the Logisil software environment used for developing insurance illustration systems.

Chapter 6. LOGISIL SOFTWARE ENVIRONMENT

In this section we will talk about the tools that are used for GUI development

6.1. Project Management (SourceSafe)

Microsoft Visual SourceSafe is a version control system for team development of software applications [SourceSafe, 1998]. Version control systems track and store changes to a file so developers can review a file's history, return to earlier versions of a file, and develop programs concurrently. Microsoft Visual SourceSafe does this using reverse delta technology, and stores only the changes to a file, not each complete version of the file itself. And, unlike other version control systems, Visual SourceSafe is project-oriented.

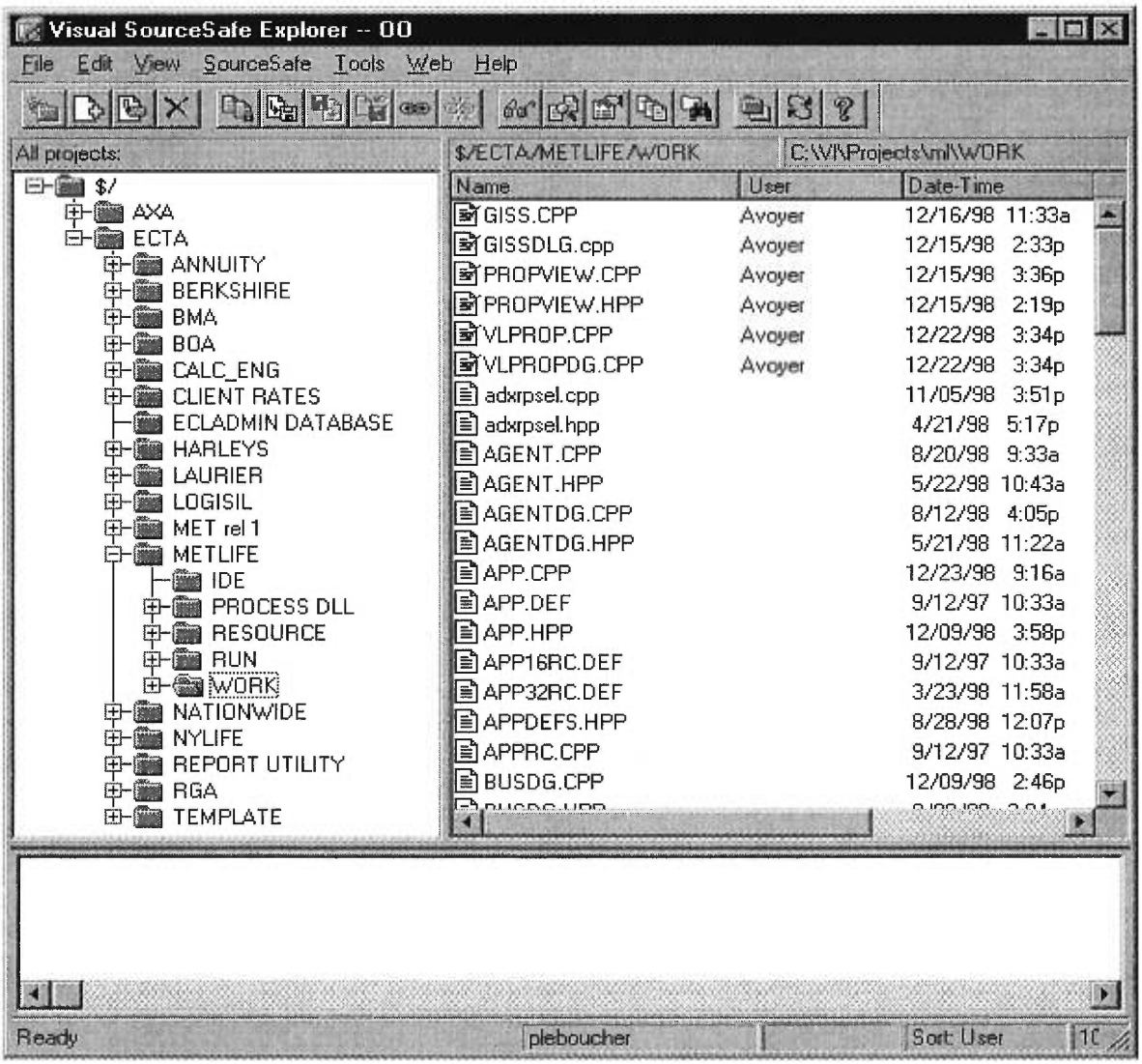

Figure 10. Microsoft Visual SourceSafe.

Understanding projects is the key to understanding Visual SourceSafe. It is helpful to think of a project as similar to a file system folder. Like a folder, a project is a collection of files that you create and maintain. Also, like a folder, a project is hierarchical; that is, you can place a subproject under another project, another subproject under that one, and so on.

While it is useful to think of a project as analogous to a folder, it is important to remember that it is not quite the same thing. Unlike a folder, every Visual Source project keeps a detailed record of its history, can be deleted and then recovered, and can share one file with many other projects.

There is always one current project, which Visual SourceSafe Explorer displays above the file list. One can change what is displayed by navigating the project list. Most commands that are carried out act on files and subprojects inside the current project.

Figure 10 here above shows a project for ECTA Company in the ECTA folder. There are several different projects like "BERKSHIRE", "BMA", "LAURIER", "METLIFE", etc. In the Metlife project there are subfolders with the same organization as programmers have on their hard drives.

So basically Microsoft Visual SourceSafe is a tool that is used by programmers so that they can safely work on the same projects, modify files and easily merge their changes to other changes that were made on the same file. Its allows the project manager to monitor who did what, when and how? And if a certain change breaks the smooth execution of the project one can always go one step backwards and get back to a previous version…

6.2. Borland C++ & Visual C++

All the excitement surrounding the Java programming language might give you the idea that C++ should be placed at the head of the endangered species list. But whatever the future may hold, C++ is still arguably the best way to get top performance and access to cutting-edge features on today's Microsoft Windows platforms, including Windows 98 and Internet Explorer 4.0. Microsoft certainly thinks so. The new release of Microsoft Visual C++ 6.0 Enterprise Edition illustrates why C++ is likely to continue playing a strong role in today's Internet and database applications for the enterprise. This package offers a compelling array of new features that. It also offers unprecedented ease-of-use features for those just starting out with C++ and the *Microsoft Foundation Classes* (*MFC*), as well as significant improvements for enterprise development [Richard, 1999]. At Logisil we use Visual C++ for

all our new projects and some of the old projects that were programmed using Borland C++ are now in the process of being converted to Visual C++.

Borland is based on *Object Windows Library* (*OWL*) while Visual C++ is based on MFC. OWL is a framework (for both C++ and Pascal) for developing applications that will run under the Microsoft Windows operating system [Borland, 2000]. Borland includes Resource Workshop, which provides a consistent user interface that makes it easy for users to switch from one Window to another. It is mainly used for designing the GUI interface. By designing we mean drawing the fields... We also use it for declaring our error messages.

OWL Variable-Length Decoder (*VLD*) [Babel, 2000] allows compiling Borland C++ code using Visual C++. This is very powerful since some old projects are being converted to Visual C++.

6.3. Ecta Class Library (ECL) Administrator (Admin.)

ECL Admin. is used to create the data model. Mostly, we use ECL Admin to define persistent storage variables. It is also being used to define the GUI controls, the labels, the error messages, the result fields and the set of elements contained in a combo box. ECL Admin. assigns a specific ID to each control you define. Therefor, instead of using the IDs, you can use the name you gave the control in your code (which makes it easier to understand). Before being able to populate the dialogs, you must create the data model your system will use. The Base data model is already part of the template system.

The data model in ECL Admin. is hierarchically split into containers (Tables that group information related to a specific topic). By splitting our data (containers, sets, etc.) into smaller components we can include smaller chunks of information without having to include everything. Not all users require Advanced Market support, and should not be required to carry the overhead associated with this application, however, if the client requires Advanced Markets support, access is available to the already defined containers and sets. This does not eliminate work associated with the hook up and customization of the "standard Advanced Markets" model. However, a significant piece of the work can be eliminated.

When a dependency is established on a component, the component will appear in the tree view. You can browse (but not edit) all of the data associated with the component. You will be able to "extend" the definition of a container or a set defined by that component. Additionally, your controls/fields will be able to reference any sets, elements, containers or other fields defined by that component. This relationship allows you to interact with the

component almost as if it is yours, with the exception that component data cannot be edited. Defining a control in ECL Admin. is very easy. All you have to do is create it using the popup screen here below (figure 11). So basically you only have to specify the data type and the container where the field is stored. Containers are a broad category of data, with controls being the individual items stored inside the container. The items stored in containers are data entry fields and GUI controls. These controls are displayed on the dialogs you create to allow users to convey information to your illustration system.

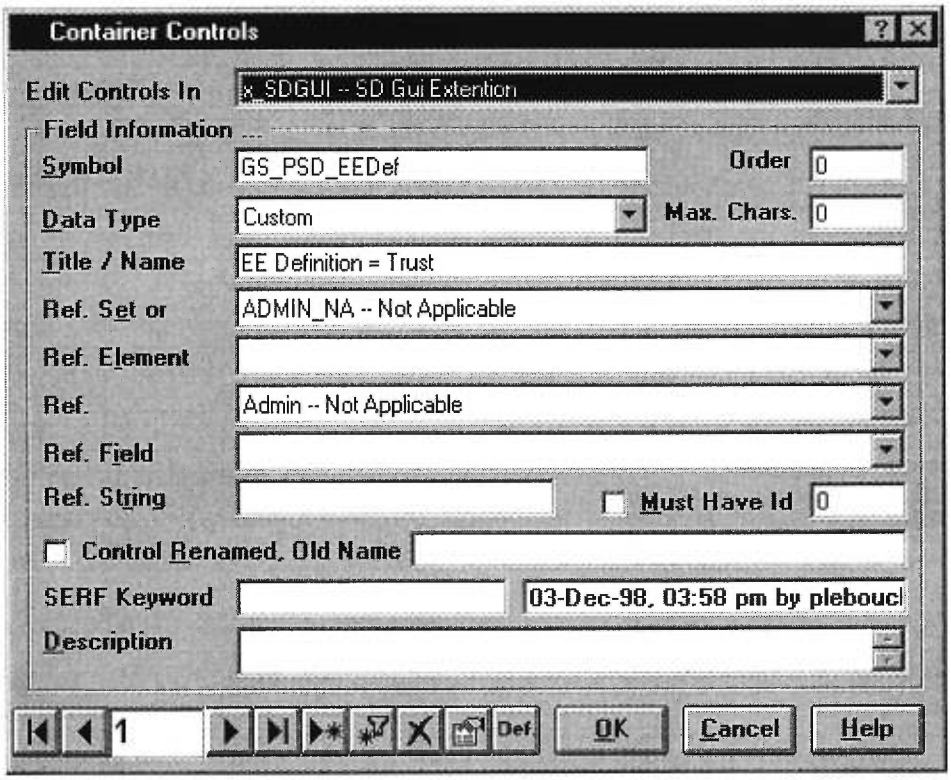

Figure 11. ECL Interface.

After finishing an ECL Admin. session there are four files that need to be generated and in certain projects they could be more (Projects that have CALC Engines coded in Pascal). In order to generate these files we use the export command which pops up the dialog in figure 12 here below.

There is one CTL file called ProjectName.CTL, which contains all of the containers that you have established a dependency upon. The .CTL file contains run-time information that is read by an ECL application. This information describes the format of each container

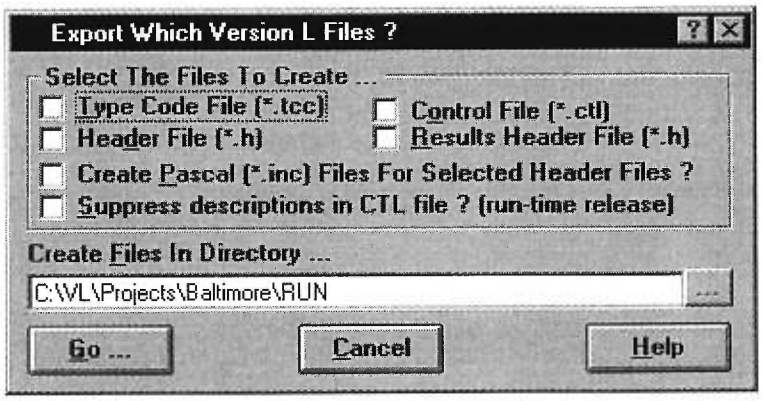

Figure 12. Exporting files in ECL Admin.

and the fields/controls that are associated with the container. In addition to the basic description of a container, information is stored in the CTL that sets the default values, minimum and maximum values for a field and formatting information. In summary, this file communicates all of the information that you provide in ECL Admin. that describes a field to the ECL application.

The .TCC file contains run-time information that is read by an ECL application. This information describes each set and the language specific text associated with each element. You extend a set to add and suppress elements. Set views are used to create a variation in the element labels for the elements that exist in a set (or set extension). What does this have to do with the TCC file? All applicable set extensions are resolved when the TCC file is written out. Second, additional sets are generated that match the set views that are described by the component. The end result of this process is a TCC file that will contain all of the sets (and views) that have been defined in your component and any component that your component is dependent upon.

The compiler uses the header files. They are also used in the case of Resource Workshop to automatically associate the fields to their Ids in ECL Admin.

The .INC files are used in particular projects where the engine is coded using Pascal programming language. They are the link between some of the Ids used at the GUI level an engine Ids.

So ECL Admin. is mainly used for managing the data model. All persistent, GUI, or result fields are declared using this very efficient home made tool. Figure 13 here below gives a general idea on how ECL Admin. is integrated and how it interacts with the other project modules.

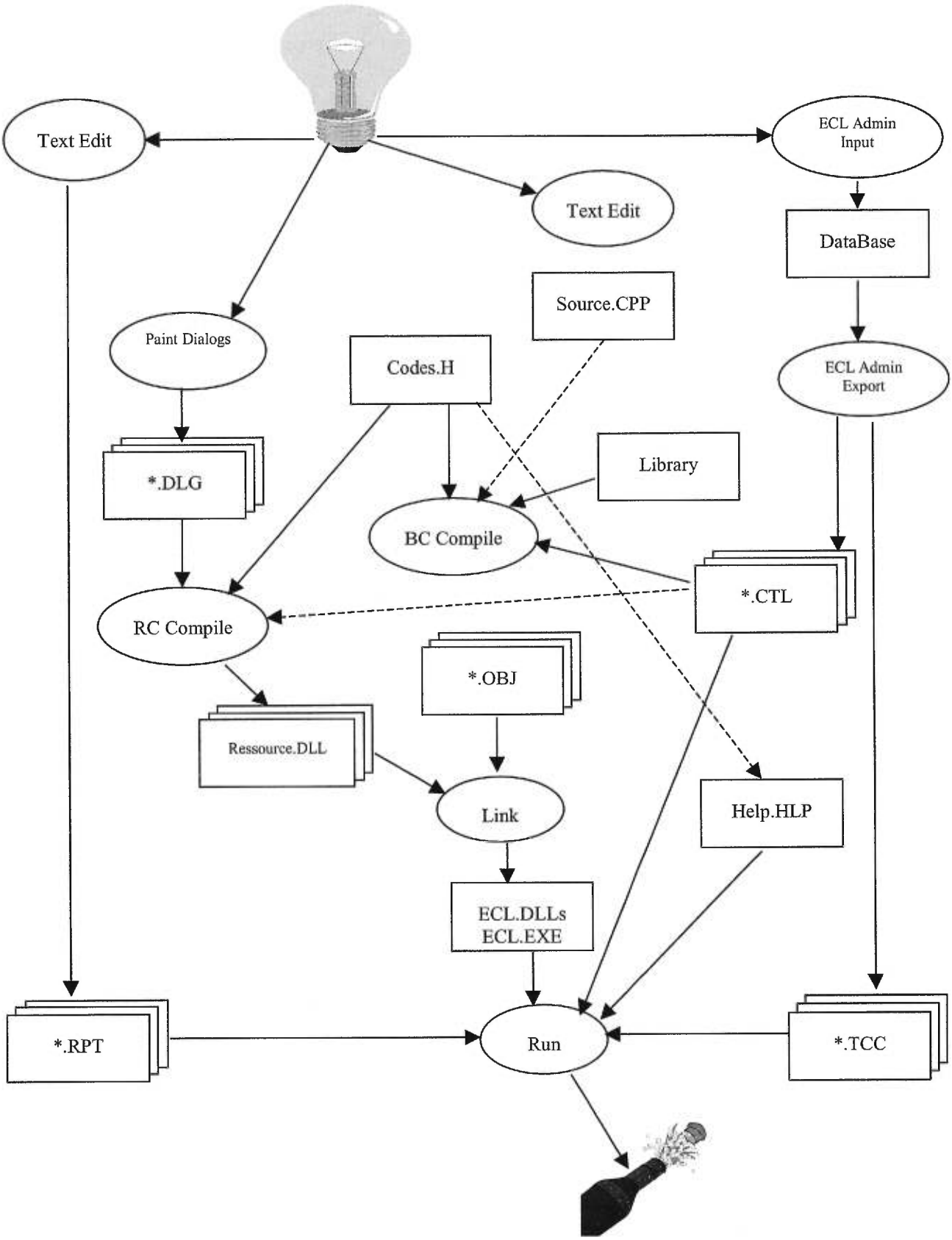

Figure 13. Interaction of ECL Admin. and the other system tools [Training, 2000].

The reports are created by a text editor and are used at runtime to display the results. Dialogs are compiled as well as the source code with the libraries and are all linked with the ECL DLLs. Help is independently created as well as the engine which is also presented as a DLL. All these parts interact together to give the insurance illustration system.

6.4. Business Rules

The *rules* are a shared database both used by the GUI and the Calculation. The ECTA Business Rules Database contains many of the rules and parameters, which define a life insurance policy form with it's associated coverage (base, riders and benefits) as presented, in figure 14 here below.

The Rules Database contains these tables:

- *Engine table*
- *Jurisdiction table*
- *Plan table*
- *Coverage table*
- *Benefits table*
- *Fund table*

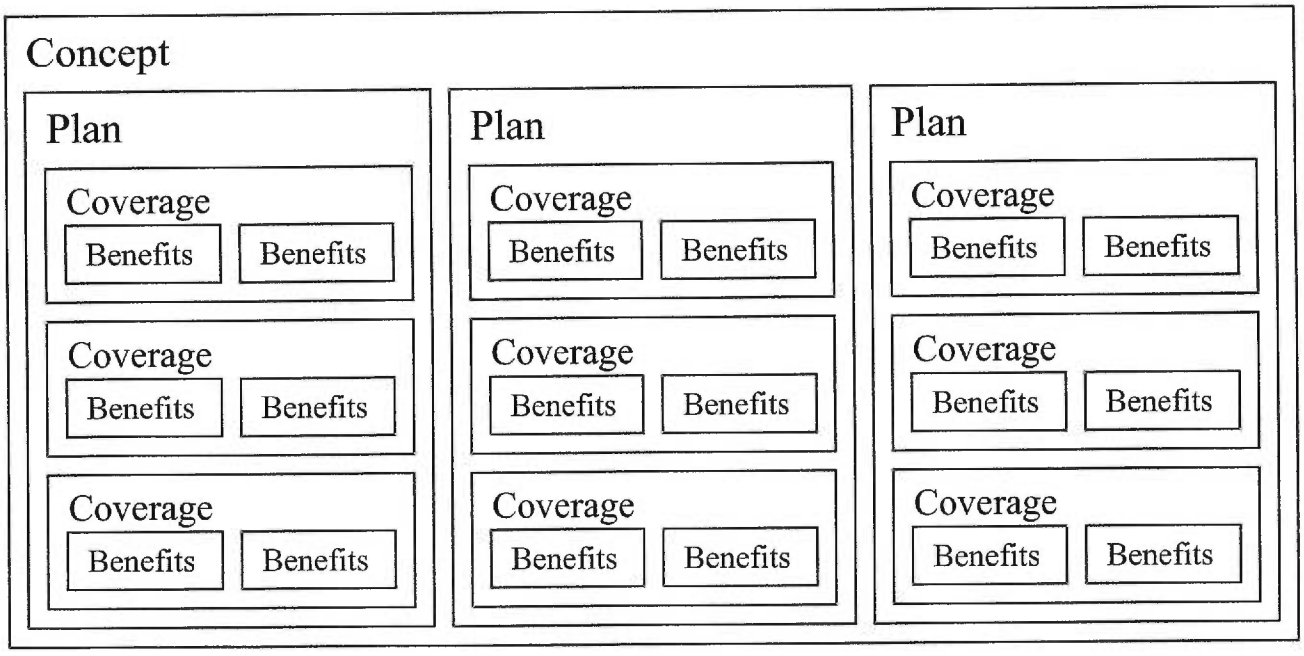

Figure 14. Components of the Navisys illustration system.

The Rules are used to externalize the Business Rules. You don't need to recompile when the rules change (using the .EBR). So it makes it easy to change values once the system is delivered. Also, it helps in developing the application since you only need to put the values in one place. Here are the components included in the rules:

- Concepts: A concept defines the way you sell the products.
- Plans: This defines the policy forms (insurance policies) within product families such as Whole Life, Universal Life, Term, Variable Life, etc for the insurance company.
- Coverage: This defines the base coverage and rider coverage available to make up plans.
- Benefits: This defines benefits, which can apply to multiple coverage.
- Funds: This defines the rules associated with investment funds available to variable type policies.
- Jurisdictions: This defines the rules associated with the states or jurisdictions where the insurance company is authorized to transact business.
- Engines: This contains records that point to and define which product calculation engine DLLs are available for this insurance company.

Navisys has created specials methods to read or write data stored in ECL Admin. For example PeekString() and PeekDouble() to peek (read) strings and doubles, PokeString() and PokeDouble() to poke (write) strings and doubles. So en example of how a value is read from the rules would be as follows:

```
String strPlanId;
strPlanId = GetRuleSession()->GetPlanTable()->PeekString( CS_PlanId );
```

6.5. CALC Engine & *"Serf-mapping"*

The CALC Engine is like a black box for GUI programmers. The interaction between the GUI and the CALC Engine is done via *"Serf mapping"*. The way this works is presented in figure 15 here below. All CALC programmers have solid mathematics background because their work involves a lot of insurance formulas... There is a distinction between variables used in the GUI and in the CALC Engine. The reason behind this distinction is making both teams independent. The CALC Engine person can make his engine work without having to worry about the GUI person. The only bad thing about that is having to do extra work for mapping GUI and CALC variables but it was proven to be efficient.

The *"Serf-mapping"* is used to pass and receive data from and to the engine so that the engine can do calculations. This data communication is done via keyword files. They are files that contain Keywords. A keyword is a place where you read or write data: it's like a drawer. A Keyword file contains several drawers. The engine reads data from keywords in the Keyword files, does the calculation and stores results in other keywords so that the GUI can display them (.SRF and .SRX files).

The .SRF file is generated using the DoPreCalcProcessing() function. In this function subclasses are created and correspondent values are set and mapped to the engine variables. This is a communication protocol between the engine and the GUI.

The DoPostCalcProcessing() is where the results are taken from the engine and mapped back to the GUI. This method is used to read data from calculation result vectors. Results are taken from the .SRX file. So as previously stated figure 15 gives a general idea on how all this process is done.

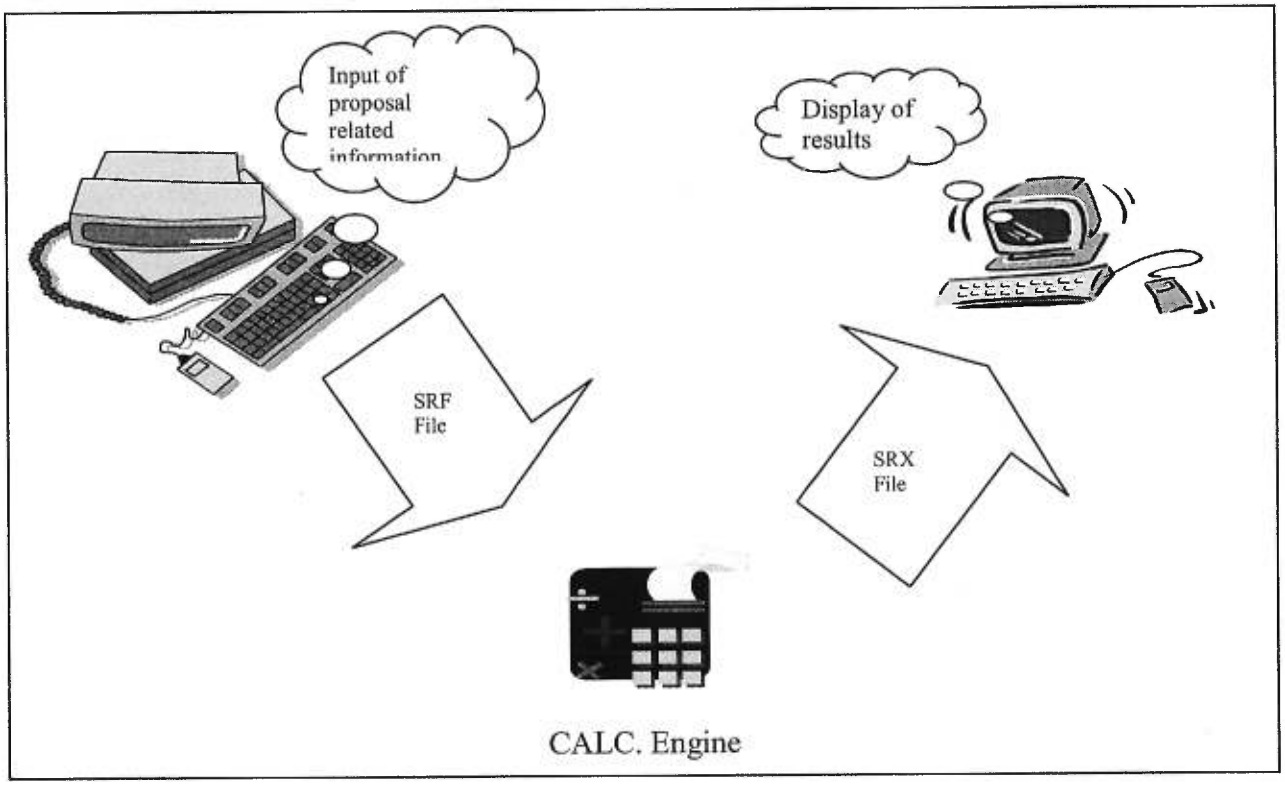

Figure 15. Communication between the GUI and the CALC. Engine.

Chapter 7. EVALUATION

Productivity, costs… are common words used in the industry these days. Projects should meet deadlines. Eventually a project will end, but will it meet the deadline and will it be lucrative for the company? These aspects should always be taken into consideration besides the technical side, which is one of the major constraints in the software industry.

In the following section we try to combine the human factor in terms of hours spent on a project, and it's expected time so that we define productivity and costs.

We will provide the reader with information on the estimated time (also referred to as Effort) and the effective time for each project (Metdemo and Hartford Life). This will be backed up with explanations on why there are differences between the two.

7.1. MetDemo

This project as we already mentioned was intended for a demo of our products in Europe. So it was an inside financed project. In other words it was important for us to try to finish within, if not, before the fixed deadline. We were four persons to work on this project, and were accorded four months.

Since I was in charge of this project I was expected to do most parts, so the other three human resources were assigned to assist me especially in the translation process which was a very long process. The project finally took five months.

We notice that it took more then four months to do the project, but less time in terms of human resources. The extra month we took wasn't critical because the demo was to be done after 6 months from the day we started the project. The total effort of each person allow us to get to the following chart (figure 16) which shows in terms of individuals the estimated and effective time (effort).

Total Effort (Time) / Individual

	Effective Total Effort	Estimated Total Effort
Person A	750	600
Person B	37,5	150
Person C	37,5	75
Person D	75	150

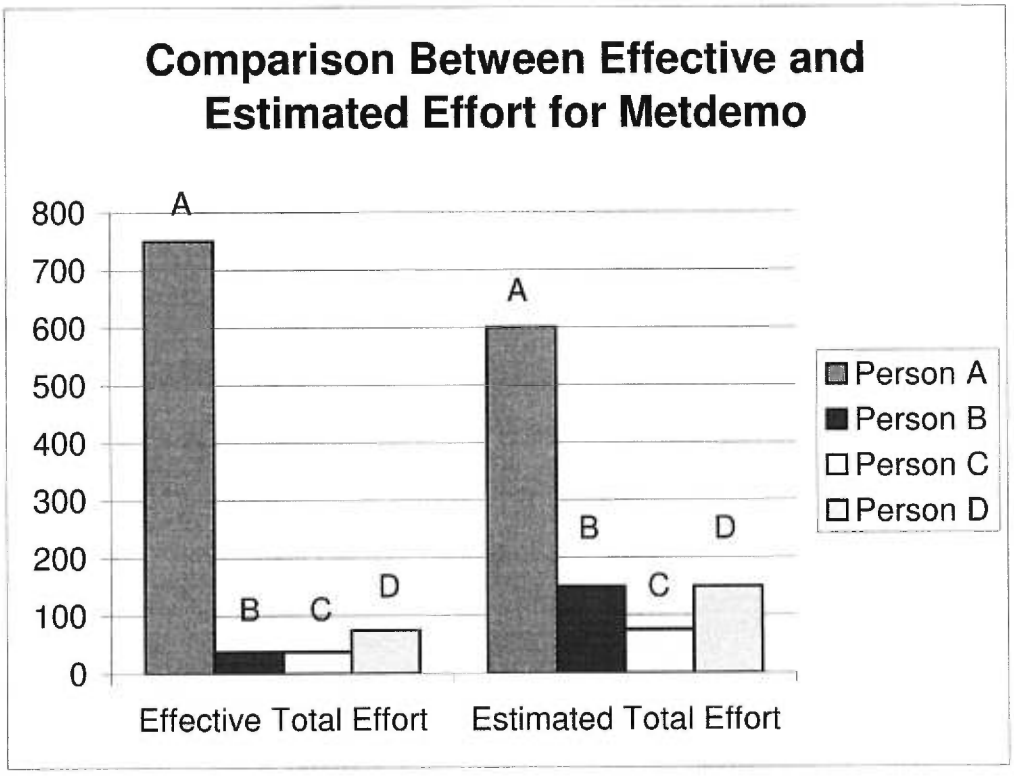

Figure 16. Difference between Effective and Estimated Effort (Metdemo).

Totals per individual allow us to measure the overall productivity. When the effective effort is less then the estimated, it means that the productivity is more then average. It is interesting to mention that the differences between estimated and effective time on a resource level were due to other project requirements. Since I was in charge of this project I was the only person fully dedicated to the project, and that's why I overlapped my estimated time by covering for others.

7.2. Hartford Life (Merlin)

We were three people on this project and were assisted at a certain point by the advanced markets team. First estimates were done but we ended up with an extra month because of the special features that our client asked for along the way.

It was very important to meet the deadlines for the features and functionality that were mentioned in the contract.

I ended up being the main C++ programmer on this project. This didn't stop me from doing analytical work that was required for each feature. The project finally took six months.

We notice that it took more then five months to do the project. More time in terms of resources. The extra month we took wasn't critical, it was beneficial for everybody. Our client was happy and aware that the delay was caused because of the additional functionality he asked for along the way.

Total Time / Individual

	Effective Total Effort	Estimated Total Effort
Person A	900	750
Person B	900	750
Adv. Markets Team (T)	300	300
Person C	150	150

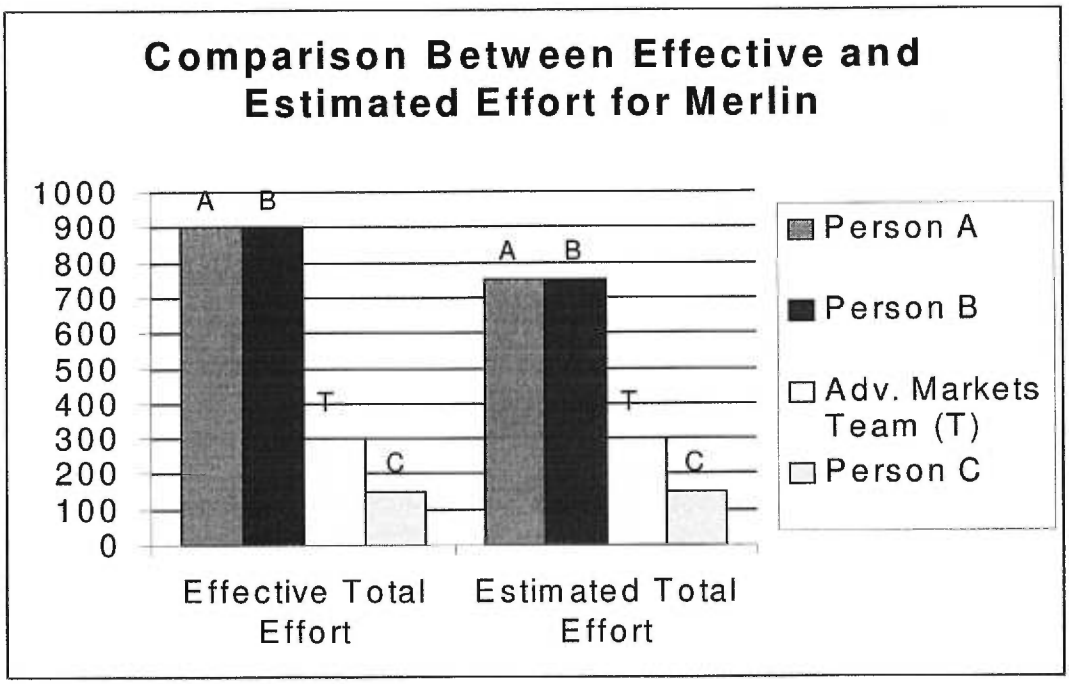

Figure 17. Difference between Effective and Estimated Effort (Merlin).

The difference between the estimated and effective time is normal in this project because it involved lot of communication with the client.

In this chapter we compared the estimated and effective effort required for two projects. We realize that they differ, depending on the nature of the project. A demo for internal use cannot be approached the same way a real contract is. The approach is different because a contract depends most of the time on the client and because of the legal implications that might occur if the project doesn't meet the deadline.

An internal project that doesn't meet the deadline can only cause financial loses in terms of efforts while a contract can take the firm to court, and ruin it's reputation.

Chapter 8. CONCLUSION

This thesis summarized my experience during an internship at Logisil Consulting Inc. as part of my Masters in Computer Science. During this year I worked on two projects: Metdemo for five months and Merlin for seven months. I've tried to step back, take a general look on my training, and came up with a few essential lessons that I've learned.

I have exposed problems that I solved, enhancements that I brought and features that I participated in developing. It is important in this whole process to focus on the lessons learned, obstacles faced, which sometimes I was able to overcome and unfortunately other times wasn't. Why is it important, because humans are supposed to learn lessons from their experiences so that the next time they are confronted to a similar situation they can handle it the right way. Also because this helps in increasing a person's productivity which is crucial for his survival in the industry. Next is an overall resume of what this thesis handled as well as the lessons learned through this experience.

Metdemo project was a big challenge for me and the solution I came up with wasn't even the best. I was confronted by having to switch from one language to the other at runtime. The way I solved this is by deleting the DLLs and reloading the application. I didn't lose the handle after deleting the DLLs, which allowed me to re-execute the application and load the DLLs that are associated with the chosen language. The best solution that I could foresee is to reload the DLLs without re-executing the application.

This would make the transition from English to French and vice versa transparent to the user. I wasn't very far from this solution but due to time constraints I had to drop it. I must admit that I was disappointed of not being able to do what I thought was best. But in the industry people ask for solutions within time constraints. Of course things have to be done the best way possible, but first they have to be done on time. I learned to focus more on solutions and analyze the problems and / or features that I have to implement before getting to the implementation phase. The time I wasted doing two versions for this project could have been invested in just one. The first version only allowed the user to switch from one language to the other after restarting the system by referring to flags in the INI file. The second version is the current version, which reloads the DLLs at runtime. This version could have been better if it included this transparency aspect which I almost accomplished. My supervisor nevertheless appreciated the current version and the feedback after the demo was very positive.

Basic software engineering knowledge was required for this project. We had to keep in mind that the actual performance of the system wasn't to be affected by our modifications. Regression testing techniques were used to make sure that our version was error free. Not doing regression tests properly can have serious consequences. For example, Seligman [Seligman, 1997] and Trager [Trager, 1997] reported that 167,000 Californians were billed $667,000 for unwarranted local telephone calls because of a problem with software purchased from Northern Telecom (Nortel). So another lesson that I learned and am still learning day after day is the importance of testing. I had tendencies of fixing bugs and testing the fix without focusing on the side effects that it could bring to the application. Regression tests are mostly done before a field release to be sure that every single validation and detail is there.

In Merlin, my second project, all concepts related to project management such as scheduling, resources, quality assurance, testing… that were previously mentioned in the state of the art chapter, were used in the intention of making the development and field implementation of this project as smooth as possible. This project was directly financed by our client and room for error wasn't really tolerated.

As previously mentioned through our thesis, Merlin was a project directly supervised by our client. Working on this project made me develop some managerial skills in order to deal with our client. Situations involved: acquiring information, replying to a client's request… We made sure that our client got what he wanted because he was always reasonable with his requests, which made working on this project a pleasure.

Working on this project made me develop debugging techniques which are essential for bug fixing [Debug, 2000]. It also made me realize the importance of code reengineering. This latter affects maintenance, application runtime… To pass from an unstructured code to a structured code there are three steps (Reverse engineering, updating internal specifications and regenerating the new system) that need to be achieved [Bohner, 1990]. As to the lessons that I learned while working on Merlin I can categorize them by problem or feature developed.

- The agent problem made me realize the difference between DMEMRecSets and DrecSets, which wasn't mentioned in any reference manual. It made me also realize the impact of a bug on the whole application. I made other colleagues benefit from my discovery by issuing a memo that explained the whole issue.
- The Import / Export problem made me realize the importance of first trying to find easy solutions before starting to complicate things. What I mean by that is that sometimes the

solution is right there and it's easy but you don't see it. I must admit that it took us a while to find all the problems that were related to this nightmare which affected our field version. I say we because this problem involved senior consultants because of the impact it had on our client. It made me also realize that if something gets corrupted it might not be easy to fix, or even impossible to fix, especially if it's a flat file. This also showed us the gravity if doing mistakes in this domain.

- Templates implementation was where I really improved things to make our client happy. It's all discussed in the templates implementation section but I can say to this regard that sometimes you cannot follow others even if they are more experienced then you. Analyze, first then implement because if you don't, you might have to do the same thing twice.
- Compare to feature made me explore deeper and deeper the higher level of the application (ECL level). I must say that this is the feature that I did with zero mistakes on an analysis or implementation level. I guess I learned from my previous mistakes. This feature required originality. Since it was completely new to our system.
- Code simplification (reengineering) made me learn some software engineering techniques and realize the importance of a well-structured code.

I had the opportunity to work on projects involving more then one million lines of code. This widened my scope and made me want to explore more and more the upper level of the class hierarchy. These projects were for real and included features that were required by our clients and really counted for our company's reputation. I was fortunate enough to be able to satisfy both our customers and our management. This is due to my solid analytical background, which University of Montreal contributed the most in it's development. Analysis as well as research were required for most of the tasks I worked on, and due to the vast collaboration of my research director during this period we got the best results out of these two critical steps in software development. I must also admit that the projects I worked on during my studies at University of Montreal, even if they were at a lower scale, in terms of lines of code, taught me a lot on how to approach a problem, analyze it and find the best solution.

GUI principles and rules such as performance, consistency… were always considered during our analysis because we had to make sure that our modifications wouldn't affect negatively the rest of the application. Since our application and interfaces complied to most of

these rules and principles, we as analysts and developers had to make sure to maintain our standards.

To resume the lessons I learned during my training period I can say that the most valuable things were analyzing and testing. They both involve software engineering, which is the key for software development and maintenance.

I wish I could improve runtime language switching to make it really transparent to the user. This could be done in other domains and applications. The challenge is to keep the data intact.

As to what is next, well we have major contracts with US and Canadian life insurance companies. This will make us busy for a while. As far as I'm concerned, Hartford life contract doesn't end before end of summer 2000. I have a training to give for our clients developers, new features to implement and products to add. Another good news is that our client is considering going on the web. This should be a great opportunity for me to learn insurance web applications, which is an expertise that is actually being developed at our premises in Montreal and in Ambler. We call it the Java, HTML, XML expertise. This is the key for future technologies.

Bibliography

[Arnold, 1989]: Arnold, R.S. (1989). Software restructuring. Proceedings IEEE, April 1989.

[Bohner, 1990]: Bohner, S. A. (1990). Technology Assessment on Software Reengineering, Technical Report, CTC-TR-90-001P. Chantilly, VA: Contel Technology Center.

[Borland, 1991]: Borland Object Windows Reference Guide (1991). Vers. 2.5, California.

[Breuer & Lano, 1991]: Breuer, P. T. & Lano, K. (1991). Creating Specifications from Code: Reverse Engineering Techniques. Journal of Software Maintenance: Research and Practice, Wiley, Vol. 3, P. 145-162.

[ECL, 1999]: Ecta Class Library Reference Guide (1999). Navisys Corporation. Ambler, Connecticut.

[Johnson et *al.* 1989]: Johnson, J., Roberts, T. & Verplank, W., Smith, D., Irby, C., Beard, M. & Mackey, K. (1989). The Xerox Star: A Retrospective. *IEEE Computer.* Vol. 22, No. 9, P. 11-29.

[Mandel, 1997]: Mandel, T. (1997). The Elements of User Interface Design. John Wiley & sons, Inc.

[Merlo et *al.* 1993]: Merlo, E. et *al.* (1993). Reverse engineering of user interfaces. Proceeding working conference on reverse engineering, IEEE, Baltimore, M.D., May 1993, P. 171-178.

[Merlo et *al.* 1995]: Merlo, E. et *al.* (1995). Reengineering User Interfaces. IEEE software. January 1995. P. 64-73.

[Nielsen, 1990]: Nielsen, J. (1990). Traditional dialogue design applied to modern user interfaces. *Communications of the ACM.* Vol. 33, No. 10, P. 109-118.

[Ning et *al.* 1994]: Ning, J. Q., Engberts, A., Kozaczynski, W. (1994). Automated support for legacy code understanding. Communication of the ACM, Vol. 37, No. 5, P. 42-49.

[Pfleeger, 1998]: Pfleeger, S. L. (1998). Software Engineering Theory and Practice. Prentice Hall Inc, New Jersy.

[Pressman, 1997]: Pressman, R. S. (1997). Software Engineering a Practitioner's Approach (Fourth edition). Mcgraw-Hill companies Inc.

[Ricketts et *al.* 1989]: Ricketts, J. A., Delmonaco, J. C., Weeks, M. W. (1989). Data reengineering for application systems. Proceedings of conference software maintenance. IEEE, P. 174-179.

[Seligman, 1997]: Seligman, D. (1997). Midsummer madness: New technology is marvelous except when it isn't. Forbes, September 8, P. 234.

[SourceSafe, 1998]: Microsoft Visual Studio Developing for Enterprise (1998). Microsoft Corporation.

[Trager, 1997]: Trager, L. (1997). Net users overcharged in glitch. Interactive Week, September 8.

[Training, 2000]: Logisil GUI Training Document (2000). Logisil Inc. Montreal, Quebec.

URLs

[API, 1999]: API Online. http://www.sourcevault.com/win32api/GetPrivateProfileString.htm.

[Babel, 2000]: http://www.telecomm.uh.edu/links/babel96b.html#V.

[Borland, 2000]: http://www.borland.co.uk/bcppbuilder/productinfo/competitive.html.

[Debug, 2000]: Debugging C++. http://www.cod.edu/people/faculty/lawrence/debug11.htm.

[Logisil, 2000]: Logisil home page. http://www.logisil.com/index.html.

[Newsgroups, 1999]: Newsgroups. http://www.borland.com/newsgroups/ngsearch.html.

[Richard, 1999]: Richard, V. (1999). Published as PC Tech Feature in the 1/19/99 issue of *PC Magazine*. http://www.zdnet.com/pcmag/pctech/content/18/02/tf1802.001.html.

Appendix A. Development Standards and methods

Appendix A handles development standards and methods used by Navisys and Logisil programmers.

Programming Standards

Programming standards are here to make code cleaner, readable and maintainable. Typically the parameter declaration list for a method or function should have a one or two-letter prefix on each parameter that is indicative of its type. References do not need a prefix. Also, member variables of classes should use the same naming convention.

Constructors often take parameters that are used to initialize member variables. Since there is a potential conflict with the names of the parameters and the names of the member variables, constructor parameters should be prefixed with the capital letter 'A'.

Two types of comments should be used inside blocks of code. The first is for short comments that apply to a specific statement. On the same line as the statement, the developer can put a comment delimiter '//', and the comment. This style should only be used if the comment does not pass the right margin. For longer comments, or those that apply to whole sections of code, the comments should appear flush with the code surrounding it. A line should separate the comment from the code both above.

Class declarations should be preceded by comments that clearly delineate them from their surroundings. To increase readability, always separate individual methods or functions from one another by the addition of comment lines. For the picky ones, they can add a description of the function's purpose.

One must avoid using ambiguous variables, serving many different functions. This will cause confusion during debugging and maintenance, when the meaning of the variable may no longer be clear.

All code should be easy to follow, and logically grouped together. One should avoid breaking up the logical flow of the code by inserting unrelated segments in between. Also, always avoid creating dependencies in unrelated functions and classes.

GUI Standards

GUI standards are regulated by copyrights and are mostly tailored to fit the clients needs. Each project has its own taste and flavors but there are always criteria to be respected in this regard. We've already discussed in our state of the art section how Logisil GUI met the golden rules. We would like to emphasize on the fact that the common ground to all projects is that they all use the same tools, same validation functions… The only difference is in the interface, which depends on the client. At this level there are also efforts to make a unique interface for various projects. This interface will always lack features that are specific to big projects but the idea behind this is to provide an interface that will allow at least to create an insurance illustration using a company's specific calculation engine. Since each company has its own products and services it is natural that for each company there would be a particular CALC Engine. Winflex, which is a GUI standard interface, previously introduced in our state of the art section, was developed for specific clients who wanted to make their products easier to use. Certain brokerage companies also imposed on our clients to do the requirements to have Winflex because they were trying to provide their agents with an easy tool that allows them to sell insurance illustrations.

Winflex

Winflex was created by Lifelink systems with the collaboration of Navisys. Briefly Winflex interacts with our system via two files. It's kind of the same scenario as the serf mapping. So basically Winflex would generate a certain file that is passed to our system. This is done through a procedure called the handshake. The handshake is done when Winflex generates the file (with the corresponding information related to the insured…), and passes the hand to us. In this case we refer to Merlin because it is the first project to have Winflex (other projects are following). Once the hand is passed to Merlin, it will execute in a silent mode and does the following:

- First thing is mapping Winflex values.
- Second is passing these values to the engine.
- Third is taking the engine's values and generating the reports.
- Fourth is returning certain values to Winflex.

After doing these four steps Merlin return the hand to Winflex which displays the reports that were generated by Merlin and which were passed to Winflex as image files. Winflex also generate particular reports by using the values that Merlin passed in the forth step.

The problem with Winflex is that it requires a lot of work from Merlin side. Basically we do more work then the Winflex application.

ECTA Libraries

ECTA libraries are the Navisys libraries that include the parent classes of most of the common classes that we use in our projects. The children of these classes include all the extra tests that are needed on an application or single project level. The class hierarchy is presented in figure 18. The libraries are very useful since they include 70% of the system functionality. Sometimes we could be limited by the libraries because in a way they set for us some standards and functions that we cannot ignore and totally overwrite but by doing this they help keep the code maintainable on a project level. For more details on the libraries hierarchy please refer to appendix A. It includes charts of the libraries functions and elements.

Debugging methods

Are mainly used for bug fixing. Depending on the debugger used (Visual C++ or Borland C++) there are certain keywords and certain tricks to be known to make this task as efficient as possible. Mainly in the case of a crash one can refer to the stack and then put breakpoints…

There is a special file called (KTrace.TXT) which is mainly used for report problems. If there is a crash at the reports level (written using a macro language) we set the KTrace flag to true and can trace where the crash happened.

Ecta also introduced some debugging message (DTDEBUG Message) that can be activated by setting the (ECTAMESSAGE) flag in the system environment to Y.

We basically covered in those two sections the main tools and software used by Logisil and Navisys GUI programmers. As previously exposed most of the tools are custom made to make the programmers tasks as easy as possible. But even if, the system is very complex and big that it takes time for getting to really manage it.

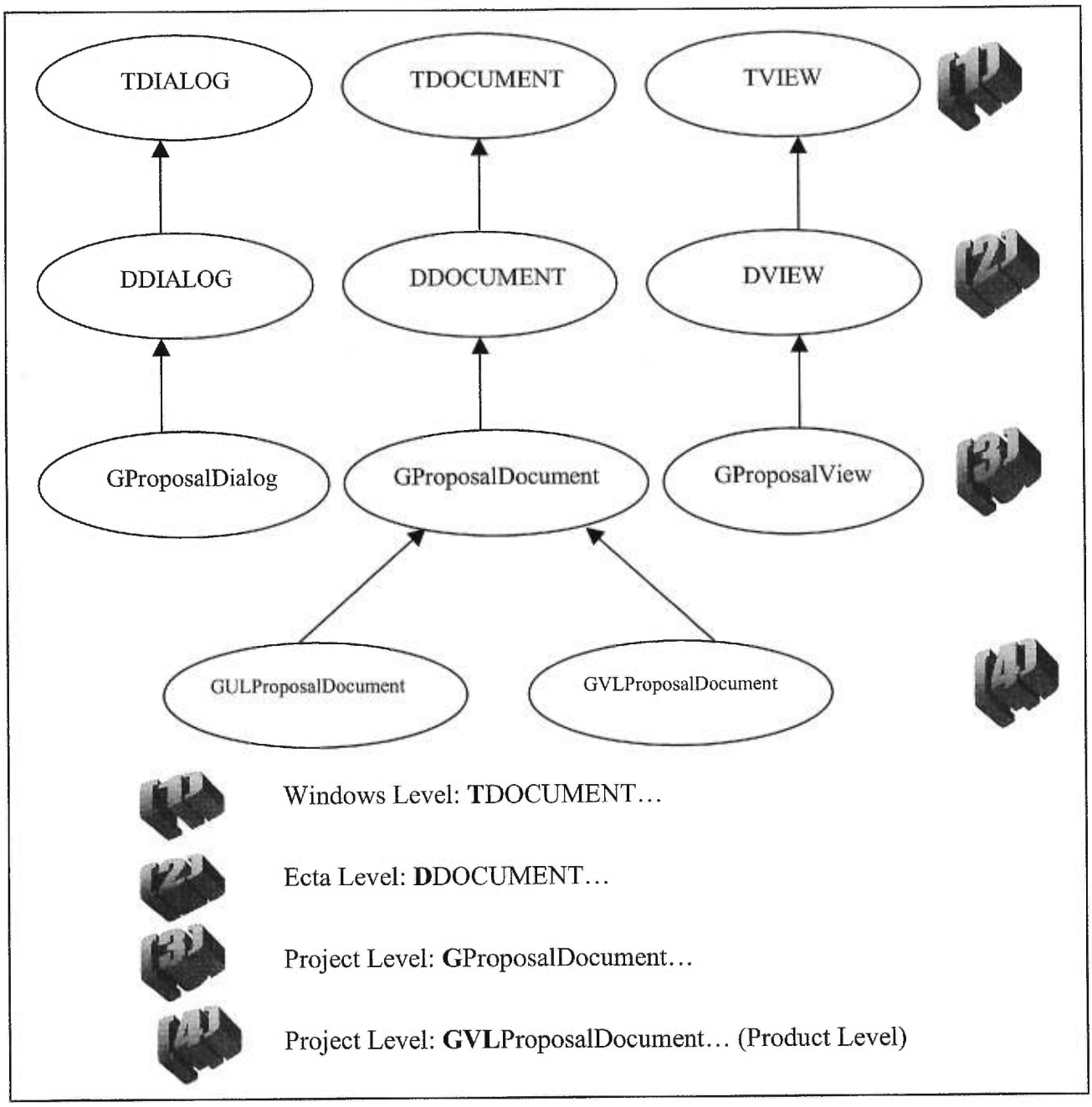

Figure 18. General Class Hierarchy Diagram.

Appendix B. Navisys Libraries Hierarchy

Logisil/Navisys Gui structure is composed of a Document, a Dialog and a View that are represented in the figures (19,20,21) here below.

Figure 19 shows the levels that are derived from DDataManager which as the name indicates manages data storage. This is where Peek and Poke functions are defined as well as all the other related storage functionality.

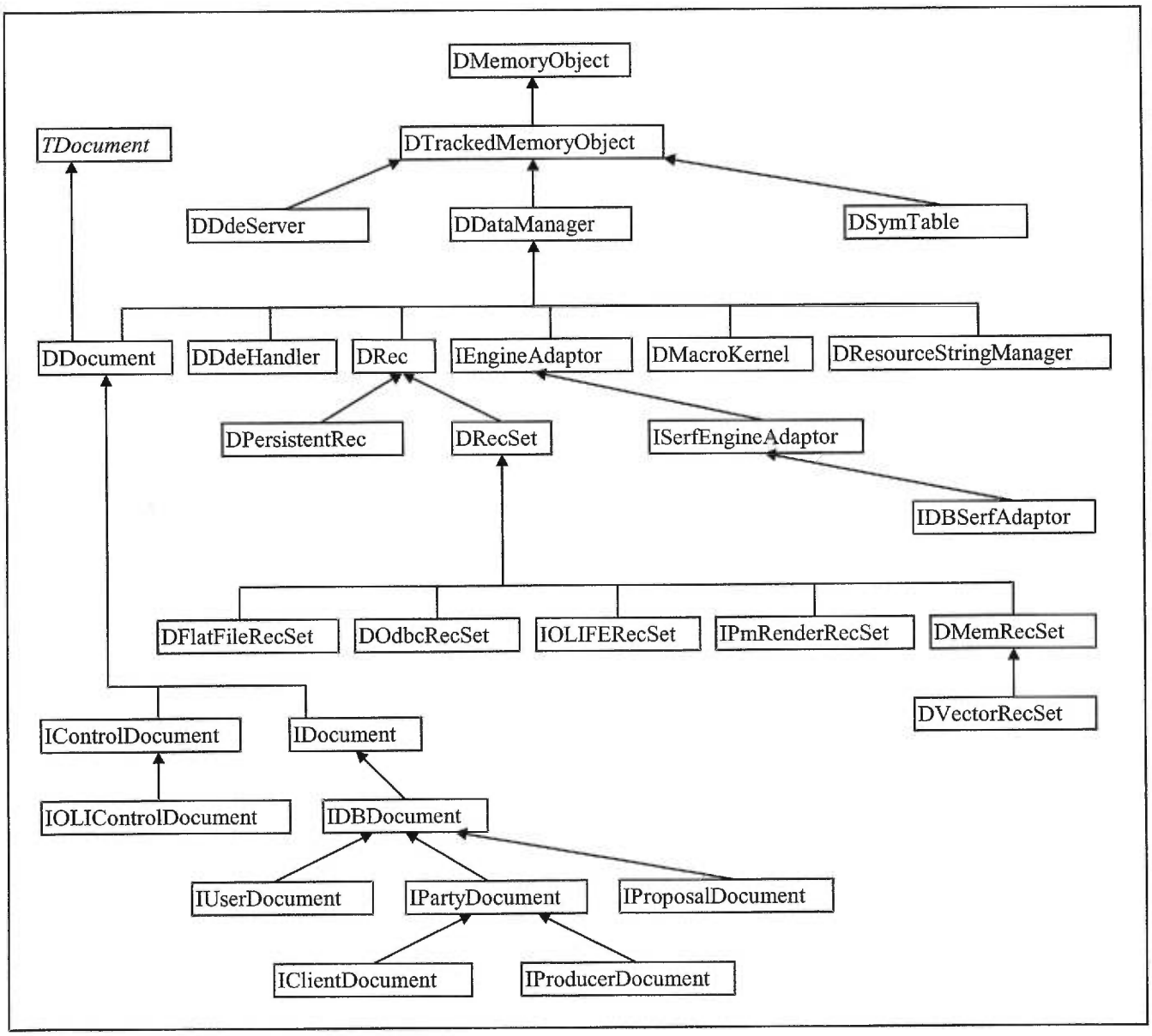

Figure 19. Hierarchy of Navisys storage system.

As for the Document well it kind of a warehouse that will contain data related to a proposal. It could contain fields (Integers…) as well as blocks of information (blobs…).

Figure 20 presents the classes that contain the dialog constructors. We start from TDialog to DDialog and we reach the application level.

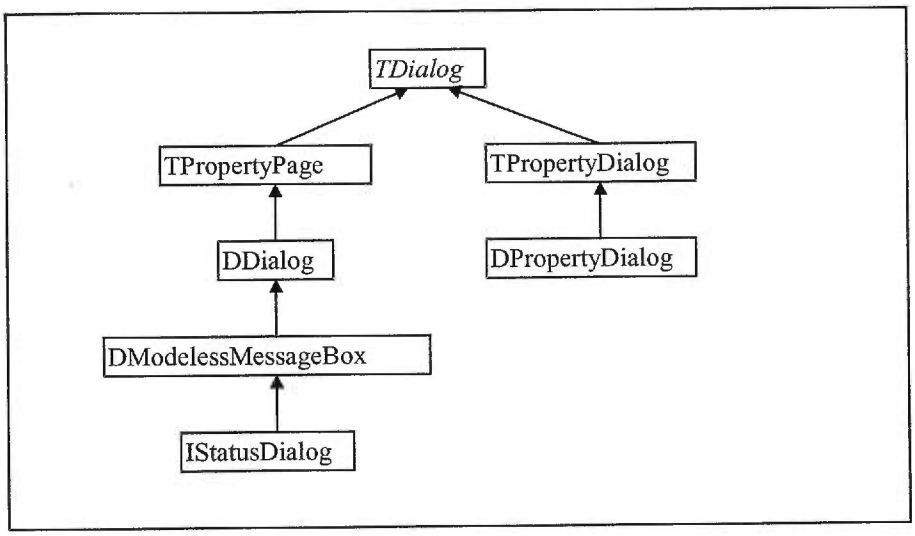

Figure 20. Dialog hierarchy.

Figure 21 illustrates certain properties like assigning the canhit property to fields in reports… are defined… It is basically related to the report view…

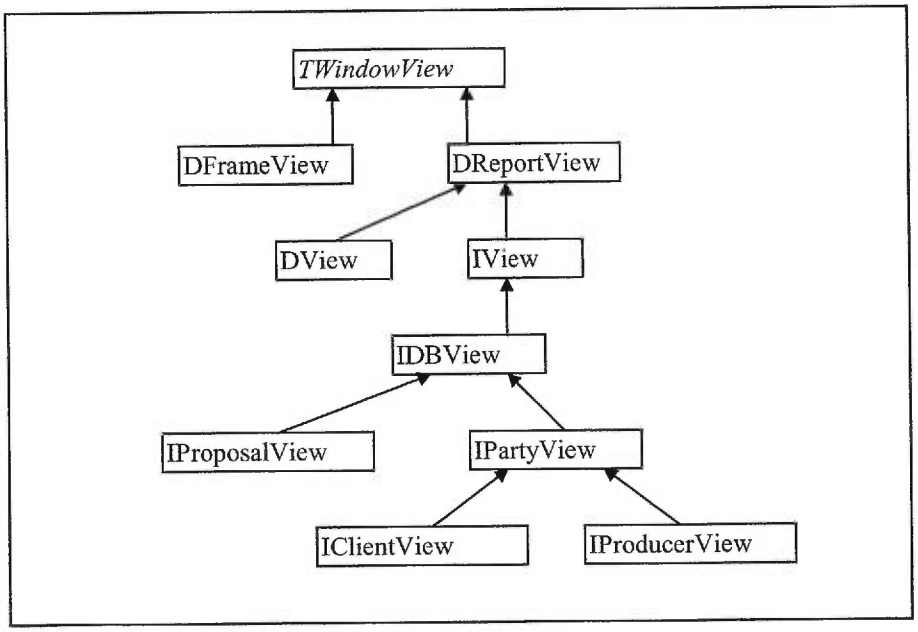

Figure 21. View level.

In the following we show the application hierarchy (figure 22) as well as the data types (figure 23) and controls (figure 24).

Figue 22 presents all the application initializations and where DLLs are assigned...

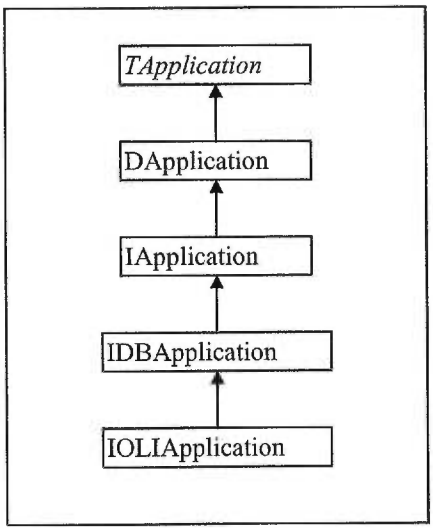

Figure 22. Application Hierarchy.

Figure 23 is where we can see all the data types that are predefined in ECL Admin. We also see some of the pointers that could be used to reference certain types (DblobClipBuffer).

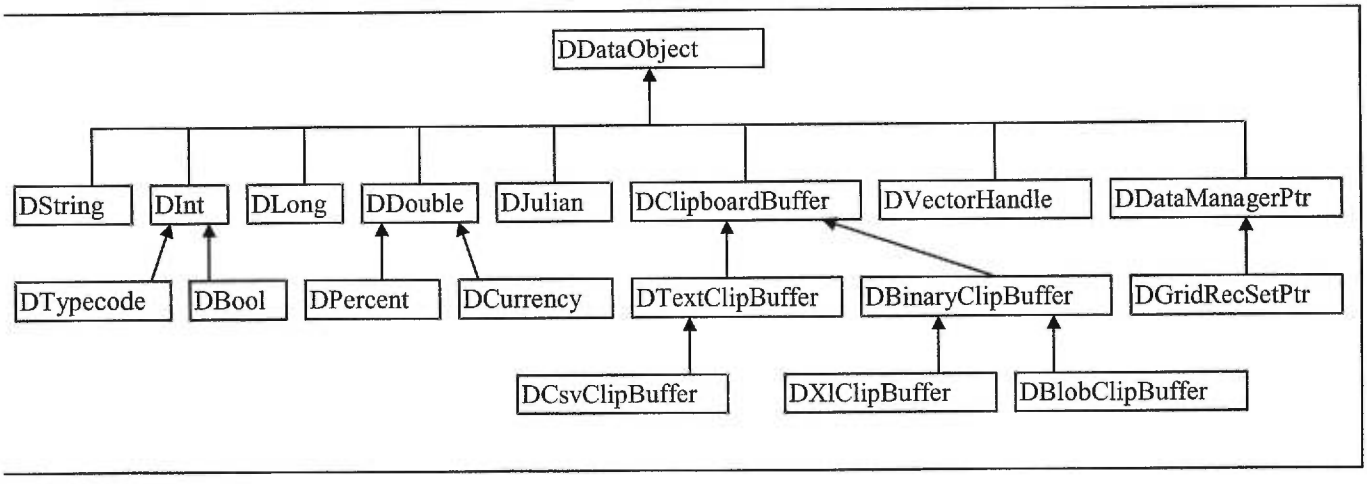

Figure 23. Data Types Hierarchy.

Figure 24 presents control types. All these types could be found in ECLAdmin and they would be related to fields in Resource Workshop.

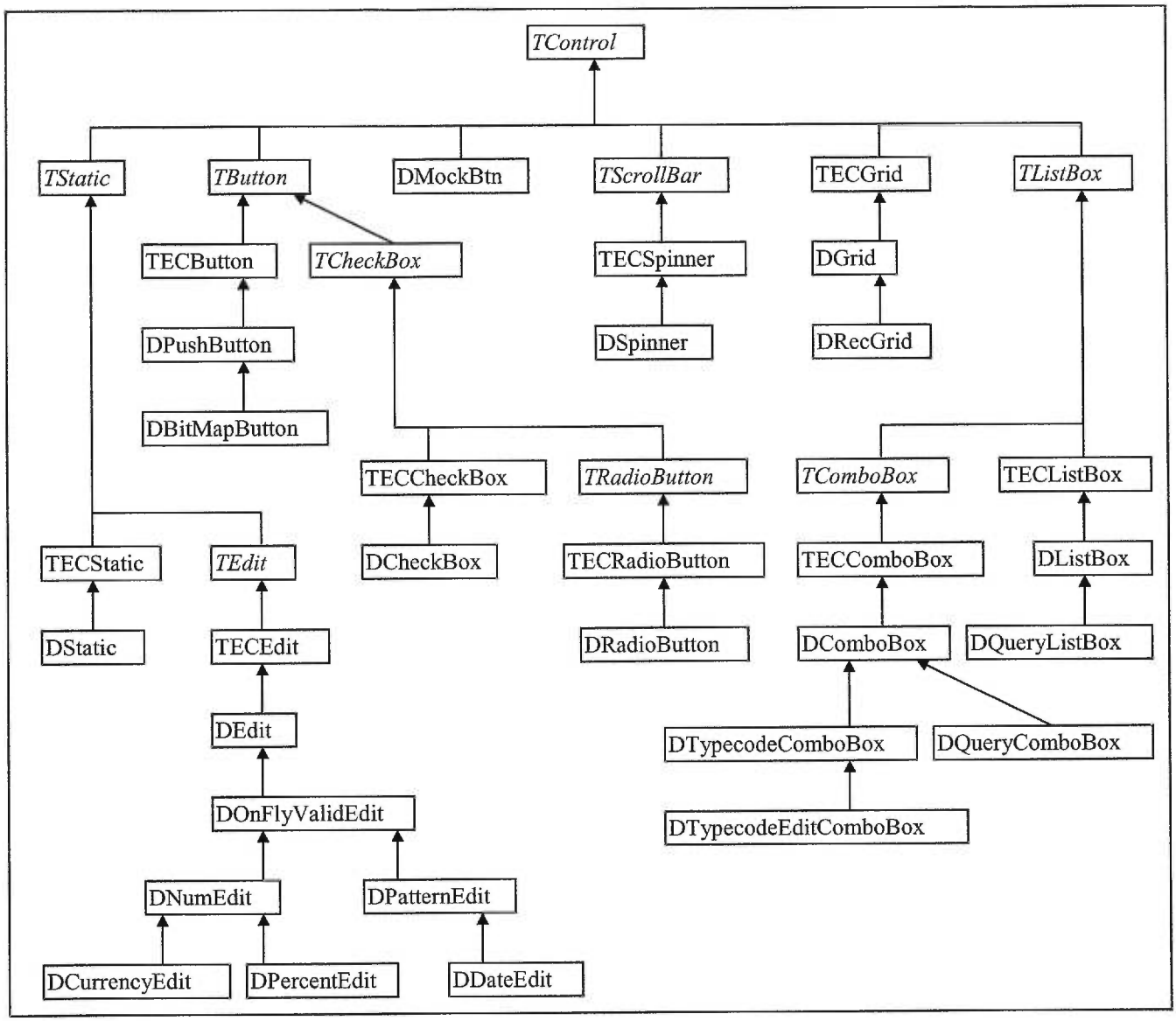

Figure 24. Control Types.