Examensarbete

Design and implementation of a Function Block configuration tool

av

Saman Hadiani

LITH-IDA-EX--04/042--SE

2004-04-23
Examensarbete

Design and implementation of a Function Block configuration tool

av

Saman Hadiani

LITH-IDA-EX--04/042--SE

2004-04-23

Handledare: Arve Solie
Examinator: Nahid Shahmehri
Abstract

In high voltage substations there is an increasing demand for computerized equipment, and automation of operation and supervision. The future primary equipment will provide the possibility to collect more information and to provide better control for improved effectiveness.

Since ABB has got a large customer target, each product must be custom made to fulfil the customers (often from different countries) demands. Later these products will be configured by engineers who may not be familiar with the tools used at ABB. This is a very expensive and time consuming. To solve the problem, ABB was looking for a low cost, easy managed and more advanced replacement.

This report includes an investigation for a function block configuration tool based on Microsoft Visio. The tool is connected to a general database using ODBC for the latest ABB Substation Automation SW platform for protection in power transmission equipment.

The thesis report mainly contains a description of the functions and features that are requested. Then methods used to build the prototype is explained and at the end the result is presented.

At the same time the possibilities and problems that will arise from using ODBC based interface for control and monitoring of real-time systems were investigated.
# Table of Contents

1. **INTRODUCTION** ................................................................................................................................. 4
   1.1 BACKGROUND ................................................................................................................................. 4
   1.2 PROBLEM FORMULATION .............................................................................................................. 4
   1.3 DELIMITATION ............................................................................................................................... 4
   1.4 FORMULATION OF GOALS ........................................................................................................... 4
   1.5 METHODS .......................................................................................................................................... 5
     1.5.1 Validity of data and analyses .................................................................................................. 5
   1.6 OUTLINE OF THE REPORT ............................................................................................................. 6
   1.7 COPYRIGHT NOTICE ...................................................................................................................... 6
   1.8 DEFINITIONS .................................................................................................................................. 7

2. **SUBSTATION AUTOMATION** .............................................................................................................. 8
   2.1 INTRODUCTION TO SUBSTATION AUTOMATION ................................................................... 8
   2.2 CHOOSE PROTECTION, CONTROL AND MONITORING FUNCTIONS FROM THE SOFTWARE LIBRARY .......................................................................................................................... 8
   2.3 PROTECTION, MONITORING AND CONTROL .......................................................................... 9
   2.4 CONFIGURATION ............................................................................................................................ 9
   2.5 COMMUNICATION ........................................................................................................................... 9
   2.6 HUMAN SYSTEM INTEGRATION & INFORMATION HANDLING ............................................... 9
   2.7 THE 500 SERIES ............................................................................................................................ 10

3. **BASIC TERMINAL BLOCKS** ................................................................................................................. 12
   3.1 AFL COMPONENTS ........................................................................................................................ 12
   3.2 SOFTWARE DESCRIPTION ............................................................................................................... 12
     3.2.1 CAP 540 Application .................................................................................................................... 13
     3.2.2 Microsoft Visio............................................................................................................................ 14
   3.3 HARDWARE ....................................................................................................................................... 15
   3.4 SUMMARY ......................................................................................................................................... 15

4. **BUILDING BLOCKS OF THE ACT** .................................................................................................... 16
   4.1 VISIO HANDLERS ........................................................................................................................... 19
     4.1.1 Overview ..................................................................................................................................... 20
     4.1.2 Interface ....................................................................................................................................... 22
       4.1.2.1 Visio objects and COM interface .......................................................................................... 23
       4.1.2.2 Wrappers .............................................................................................................................. 24
       4.1.2.3 Where to start ........................................................................................................................ 25
       4.1.2.4 Returned Values .................................................................................................................... 25
       4.1.2.5 Arguments Passed to Visio Methods .................................................................................... 26
     4.1.3 Event Object .................................................................................................................................. 27
       4.1.3.1 Defining an Object .................................................................................................................. 27
       4.1.3.2 Receiving information .......................................................................................................... 28
     4.1.4 Handling Events .......................................................................................................................... 30
   4.2 DATABASE ROUTINES ....................................................................................................................... 34
     4.2.1 ODBC Driver ............................................................................................................................ 34
       4.2.1.1 Role of the DRM Driver ......................................................................................................... 35
       4.2.1.2 Linking ODBC Application to a Non-SQL Database ............................................................. 36
       4.2.1.3 Tasks of the DRM Driver ..................................................................................................... 36
       4.2.1.4 SimbraEngine Limits ............................................................................................................. 39
     4.2.2 MFC .............................................................................................................................................. 40
   4.3 DATA STORAGE .................................................................................................................................. 42

5. **CONCLUSIONS** ................................................................................................................................... 43
   5.1 COMPARISON BETWEEN RESULT AND GOALS ....................................................................... 43
   5.2 LEARNING’S FROM THE PROJECT ................................................................................................. 43
Table of Figures

FIGURE 3.1 EXAMPLE CONFIGURATION IN CAP 531 OF LINE PROTECTION TERMINAL................................. 13
FIGURE 3.2 BASIC AND ADVANCED TOOLBOX PACKAGE ................................................................. 13
FIGURE 3.3 MS VISIO ......................................................................................................................... 14
FIGURE 3.4 TERMINAL SETUP ........................................................................................................... 15
FIGURE 4.1 ACT OPTION MENU ....................................................................................................... 18
FIGURE 4.2 ACT MAIN PROGRAMMING BLOCKS ............................................................................. 19
FIGURE 4.3 THE INTERACTION BETWEEN VISIO AND YOUR CLIENT PROGRAM ............................... 21
FIGURE 4.4 VISIO OBJECT MODEL ................................................................................................. 22
FIGURE 4.5 COM CONVENTION ....................................................................................................... 23
FIGURE 4.6 THE INTERACTION BETWEEN A CLIENT EVENT SINK AND A VISIO SOURCE OBJECT ........ 29
FIGURE 4.7 CONNECTION TYPES ..................................................................................................... 34
FIGURE 4.8 ODBC LAYERS ............................................................................................................... 35
FIGURE 4.9 SIMBAENGINE OBJECTS ............................................................................................. 37
FIGURE 4.11 NODE CHAIN ............................................................................................................... 42

Appendices

APPENDIX 1 REQUIREMENTS ADDED DURING THE THESIS.
APPENDIX 2 SOFTWARE FLOW CHART.
APPENDIX 3 COPYRIGHT NOTICES.
1. Introduction

This master thesis has been carried out at ABB Automation Technology Products AB at the department of Substation Automation/DA. The department develops products for protection and control of high voltage substations and relays. Arve Sollie at ABB supervised the master thesis with a competent point of view. The Examiner at the department of Computer Science at University of Linköping is Professor Nahid Shahmehri.

1.1 Background

In high voltage substations there is an increasing demand for computerized equipment and automation of operation and supervision. If tasks are automated fewer errors will occur. Today ABB uses an aged tool to configure the terminals used in power transmissions. Since the costumers demand more sophisticated solutions, the configuration is getting more critical. Therefore ABB is interested in an easily managed configuration tool, so that the costumer can make his own changes and monitor the terminal for debugging or similar tasks.

1.2 Problem formulation

The purpose of the project is to make an investigation of a new configuration tool that will be used in ABB Automation Technology Products new power station. The main purpose is ease configuration through a graphical tool.

The project is carried out in three phases, the first one is to make an investigation, the second one to design a system solution for the new tool and final third one to implement and develop a prototype of the solution.

The investigation is based on a literature study, and information provided from ABB personnel, but also on own analysis. The result of the investigation is assembled in this thesis report.

1.3 Delimitation

The product of this thesis is only executable on Windows 95, 98 2000, ME and XP operating systems. Only MS Visio 2000 and 2002 GUIs are tested but the new MS Visio 2003 should also work error free.

1.4 Formulation of goals

To collect information about desired functions and features for the new configuration tool.
To make an implementation of the specified functions, and to test it either in laboratory or by computer simulations.
To document the work to be done in a thesis report, containing all of the above, and finally make a presentation of the material.
1.5 Methods

The thesis project started with a preliminary study of CAP 5.40 (see chapter 3.2.1), which is the tool used at the moment for configuration of terminals. The aim was to become acquainted with its functions, though not all of them were of great importance, and search for new functions to make the tool more automated and less complicated. The preliminary study ended with a specification of the thesis project and the goals of the project referred to in Appendix 1. A structured method was used to reach the goals:

- **Collection of information**
  A collection and study of literature and reference manuals were carried out to find information that could be useful. The focus at this point was not to understand all details, but to find useful information to understand the major parts of the system. This was done in three ways:

  1. Simply ask for information or get a reference where to find that information. People that, in that way, were contacted and provided information are listed in a special reference list at the end of the report.

  2. A narrow study of books about Database systems and object oriented programming for useful information.

  3. Search on the Internet, mainly on ABB and Microsoft sites, for information about Microsoft Visio.

- **Discussions with coworkers**
  Several discussions took place where the aim was to find a solution about how to approach sub problems. These discussions were prepared in the same way as the preceding point, collection of information.

- **Discussions at development forums**
  Discussions with other developers on the net for idea exchange and performance improvements.

- **Implementation**
  Programming of the prototype and then verification and validation of the function in lab or alternatively with computer aid.

1.5.1 Validity of data and analyses

To ensure that the collected data and that the analyses are accurate, the following measures will be made:

- **Circulation for comment.**
  Let the people, that have knowledge in the specific area and/or functions read and comment the report, in the form of a circulation for comment.

- **Implementation and test of a solution.**
  To make a prototype implementation and to test by computer aid, to see if the functions works as assumed. The testing is only made on Windows operating systems.
• References to documents.
  Information gathered from documents are listed at the end of the report. Some
documents used is however only for ABB internal use, and can not be referred to in this
thesis report.

1.6 Outline of the report

• Chapter 1 - Introduction
• Chapter 2 - Substation Automation
  This chapter describes the products developed at SA.
• Chapter 3 - Basic terminal blocks
  Describing the role of the ACT.
• Chapter 4 - Building blocks of the ACT
  Main methods and routines used during development-time.
• Chapter 5 - Conclusions

1.7 Copyright notice

Some information and pictures in this report is taken from different documents, pamphlets,
information from the Internet, and from information provided by contact persons in ABB and
Microsoft. The accuracy of the information is only assured by reference to these papers and
persons.

Therefore to be permitted to use, copy and distribute this is hereby granted on the condition
that each copy contains copyright notices presented in appendix 3 in its entirety and that no
part of the documentation is used for commercial purposes but restricted to use for
information purposes within an organization.
### 1.8 Definitions

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT</td>
<td>Application Configuration Tool</td>
</tr>
<tr>
<td>ADO</td>
<td>ActiveX Data Object</td>
</tr>
<tr>
<td>AFL components</td>
<td>Application Function Blocks. Components/Function Blocks developed at ABB/SA to configure terminals.</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CAP 540</td>
<td>Configuration tool used at ABB today.</td>
</tr>
<tr>
<td>DDE</td>
<td>Dynamic Data Exchange</td>
</tr>
<tr>
<td>DSN</td>
<td>Data Source Name</td>
</tr>
<tr>
<td>FB</td>
<td>Function Block</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GCC</td>
<td>GCC is the compiler system of the GNU environment</td>
</tr>
<tr>
<td>HiDraw</td>
<td>Software for implementing electrical components</td>
</tr>
<tr>
<td>HSI</td>
<td>Human System Integration</td>
</tr>
<tr>
<td>IEC</td>
<td>Inter-Equipment Communication</td>
</tr>
<tr>
<td>IED</td>
<td>Intelligent Electronic Device</td>
</tr>
<tr>
<td>LON</td>
<td>Local Operating Network</td>
</tr>
<tr>
<td>Matlab</td>
<td>Mathematical software</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open DataBase Connectivity</td>
</tr>
<tr>
<td>OLE</td>
<td>Object Linking and Embedding</td>
</tr>
<tr>
<td>OPC</td>
<td>OLE for Process Control</td>
</tr>
<tr>
<td>PID</td>
<td>Product Information Database</td>
</tr>
<tr>
<td>SA</td>
<td>Substation Automation</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SimbaEngine</td>
<td>Kernel of the ODBC driver</td>
</tr>
<tr>
<td>SPA</td>
<td>Specific bus protocol used at ABB</td>
</tr>
<tr>
<td>Substation</td>
<td>Transformer station</td>
</tr>
<tr>
<td>Terminal</td>
<td>One of the REx 5xx terminals</td>
</tr>
<tr>
<td>VS C++</td>
<td>Visual Studio C++</td>
</tr>
</tbody>
</table>
2. Substation Automation

In this chapter contains an introduction to Substation Automation concepts and products.

2.1 Introduction to Substation Automation

Substation Automation includes protection, monitoring and control for transmission substations, power plants, transmission, distribution and industry applications. Solutions range from single function units to fully integrated, comprehensive, high-performance substation automation systems.

With ABB Substation Automation a great deal of attention has been paid to making the system user friendly, to cut down training time, to rationalize procedures and to reduce the possibility of misunderstanding and thereby reducing the risk for human error. In addition, high operational reliability, due to extensive automatic self-supervision, means that maintenance is considerably reduced.

2.2 Choose Protection, Control and Monitoring Functions from the Software Library

The customer can choose among a large number of protection, monitoring, control and communication features to tailor make his own terminal to adapt and fulfill the requirements that are made. The application library contains type-tested software modules (function blocks) that are developed for the electrical power system.

Microprocessor-based techniques make it possible to combine many functions into one physical unit. The traditional relay or protection function is combined with many other functions within a single terminal unit. The integration of functions within one multiprocessor environment means that the terminal plays a key role in the ABB Substation Automation concept. The system is based on the concept of distributed intelligence, where each terminal handles all information for one or several bays. In addition, each terminal is equipped with self-supervising functions to monitor conditions, values and performance.

Today the configuration of the terminal is carried out with an PC-based tool, CAP 540. This tool allows the user to configure the terminal using graphic symbols and the configuration is later uploaded through an ftp-client program. This tool is going to be replaced by MS Visio GUI library and the application configuration tool implemented during this thesis.

2.3 Protection, Monitoring and Control

The object terminal software library consists of several type-tested function blocks. Each function block is specifically designed to meet stringent requirements for dependability and security. All function blocks can easily be combined into complex functions by using the old graphical PC-based tool, CAP 540 or the newer MS Visio. A set of standard, logical building blocks, like AND-gates, OR-gates, timers, etc., ensures ease in designing conditional logics.
2.4 Configuration

MS Visio will be used throughout all stages of a project, from engineering to setting, testing, commissioning, documentation, and maintenance. MS Visio will provide for the GUI needed to make this tool user friendly. After the thesis the user can remove and add connections between different function blocks in order to achieve desired functionality. A number of free logical elements (AND, OR, Timers, etc.) enables configuration of different requirements for customer-specific solutions. Various function blocks can be combined either as pre-determined or custom-designed schemes. This means that an output signal from one function block (e.g. start of breaker failure protection from trip logic or block of distance protection from fuse failure supervision) can be used as an input signal to another function block. External signals can be used to block or enable a specific function, change setting group etc. A monitoring function offers an on-line check of all internal signals in an object terminal. This function offers the user a powerful help, providing a window into the terminal, where the user can see the changes in a signal status.

2.5 Communication

The object terminal supports multiple communication ports: a front PC connection and two rear serial communication ports, one port with ABB SPA bus and another port with Local Operating Network (LON)\(^1\) interbay bus communication. The Object Terminals may be used in an ABB Substation Automation system or in a third party Substation Automation system. In the latter case a DDE server or an OPC server may be used to access the Object Terminals via the LON interbay bus. Information is available, as time-tagged events with 1 millisecond resolution. Load and fault data, and complete oscillographic disturbance recordings are also available. Any function and parameter can be programmed and changed via remote communication.

2.6 Human System Integration & Information Handling

Each terminal has a database. This secures all information locally, such as function blocks in use, handles disruptions in DC supply, and don’t rely on any central memory. The database acts as a buffer for event signals. These signals consist of time-tagged on-line oscillographic disturbance recordings, fault locator values and other information. Event recording is not limited to internal events, it can also be used to store any status change in any of the binary inputs. Therefore, for example the terminal can also be used to monitor switchgear equipment. A number of terminals can be looped over a fiber optic bus and connected directly to a station PC and/or a telephone modem.

---

\(^{1}\) A local operating network consists of intelligent devices, or nodes, that are connected by one or more communications media and that communicate with one another using a common protocol. Nodes are programmed to send messages to one another in response to changes in various conditions and to take action in response to messages they receive. The nodes on a LON may be thought of as objects that respond to various inputs and produce desired outputs. Linking the inputs and outputs of network objects enables the network to perform applications. While the function of any particular node may be quite simple, the interaction among nodes enables a local operating network to perform complex tasks. A benefit of local operating networks is that a small number of common node types may be configured to perform a broad spectrum of different functions depending how they are linked in a network.
From any remote location, it is possible to access any station and any terminal. Available information includes everything stored in each terminal, even load and fault data. To verify the performance, it is also possible to use the recorded current and voltage values during a fault for a “playback” via a computerized relay test set and simulator program.

### 2.7 The 500 series

The 500 series terminals cover all needs for the protection, monitoring and control of overhead lines and cables, transformers, breakers and busbars on all voltage levels.

**Line Distance Protection Terminals: REL 501, 511, 521, 531, 551 and 561**

The REL 5xx terminals can be applied to overhead lines and cables in solidly earthed, as well as in isolated or high impedance earthed networks. Terminals are different in areas such as sensitivity, robustness and more.

**Object Terminal for Railway Systems: REO 517**

The REO 517 terminal can be applied to single-phase contact lines or two-phase transmission lines for railway systems, both in directly earthed and impedance-earthed systems. The main protection facility is a full-scheme distance protection device featuring three impedance-measuring zones with quadrilateral characteristic. A sudden current change function is used to distinguish between load and fault conditions.

**Disturbance recorder and Fault locator Terminal: RES 505**

The disturbance recorder enables evaluation of different events within the power system and is an important part of a station monitoring system.

**Overcurrent and Earthfault Protection Terminal: REL 505**

The REL 505 terminal can be applied as a back-up protection for overhead lines and cables. Both directional and non-directional over current and earthfault protection are included.

**Breaker Protection Terminals: REB 551**

The REB 551 breaker terminals available in different variants provides breaker-related functions associated with protection, such as breaker failure protection, auto-reclosing, synchronism and energizing check, and pole discordance.
Control Terminals: REC 561
REC 561 is used for control and supervision of circuit breakers, disconnectors, and earthing switches in any kind of switchgear and bus-bar arrangement. Functions such as interlocking, auto-recloser, and synchrocheck are also included. The terminal is suitable for various configurations – from cost-effective solutions with a high degree of integration, to optimized, highly available configurations. One control terminal can handle the supervision and control of three High Voltage bays, with a separate synchrocheck for each bay.

Transformer Protection Terminal: RET 521
RET 521 is a multi-functional transformer protection terminal suitable for fast and selective protection and control of two- and three-winding transformers, auto-transformers, generator-transformer blocks and shunt reactors. The RET 521 has a compact design for protection and control, with internal programmable logic.

General Differential Protection Terminal: RED 521
RED 521 is a high-speed general differential protection. The terminal is suitable for fast and selective protection of bus-bars, T connections, auto transformers, reactors, generators and big motors.
3. Basic terminal blocks

This chapter is supposed to give important insight about the terminals and their main blocks. One terminal consists of three main parts:

- AFL Components
- Software
- Hardware

3.1 AFL components

Application Function Library contains components that are developed at ABB and are the heart of all terminals. An AFL component is usually developed in Matlab and/or HiDraw and is basically C-code, this part is the most time consuming since engineers develop custom made functions for different purposes. The goal is to have a reusable and common AFL with high quality for all Substation Automation products. These components are then used with MS Visio or CAP 540 to build new applications.

3.2 Software description

To connect AFL components with terminal hardware, an interface is required. This interface, which goes under the name “Dragon”, is a very complex system. Dragon includes an operating system and other necessary software. This is where the thesis enters and the purpose is the find a good way to configure the terminal with different components. The configuration tool ACT (Application Configuration Tool) is installed on a remote machine and the when ready the new configuration will be sent to the Object Terminal.

ACT

Most people respond to visual images in the reports and e-mails we see every day, even when the accompanying text does not grab our attention. Images such as charts, tables, process flows, floor plans, Venn diagrams, and so, on use text and symbols to convey information at a glance. You could call this type of image an information graphic, and a good one can clarify an idea and help you to understand even complex concepts more quickly. Therefore engineers at ABB were looking for a smart tool that could graphically help and ease configuration of terminals. CAP 540 and Microsoft Visio are two tools based on the concept of drawing. CAP 540 is somewhat older software with specific purpose in electrical business and MS Visio is new software from Microsoft corp. that is user-friendlier, easier to adapt, provides a nice GUI and many other similar advantages.

Below the two configuration tools are briefly explained.
3.2.1 CAP 540 Application

CAP 540 has been used throughout all stages of normal operation from engineering, parameter setting, commissioning and power system disturbance evaluation. It can also be used to clear front indications, switch setting groups and other parameters. The engineering work is done off-line on a PC.

The default configuration in the terminals can be adapted to the customer’s needs with the configuration tool. The configuration consists of function blocks, logic gates and timers. The functions blocks included in a terminal are available in a library of functions, where the engineer can pick a function and connect it according to the requirements. The configuration tool offers a compilation check to help the engineer to make a correct configuration.

The monitoring function provides an online check of all internal signals in a terminal. It offers a window into the terminal, where the commissioner can see all changes in signal status. It is also used by the control system constructor that for example adapts the interlocking logic to the switchgear configuration of the station.
With the parameter-setting tool, parameters from the terminal can get read, edited and rewritten. In addition, a simple monitoring function with access to service values is included. The simple monitoring functionality helps uploading several power system values like currents, voltages and frequency. Service values include list of terminal events, current status of internal signals, self-supervision, LED status, etc. The tool can also monitor communication for the line differential function and display internal measurements from functions like the automatic reclosing function.

3.2.2 Microsoft Visio

Visio is a business and technical drawing and diagramming program that anyone can use to communicate concepts, procedures, product information, specifications, and more. Visio is designed to help convey information visually, without requiring that you know how to draw. Visio does this by giving solutions to the diagramming needs: ready-to-use templates that set up a page appropriately and open stencils that contain predrawn shapes or function blocks. For example, Visio includes several flowcharting solutions. These advantages are useful but not used during this thesis since many of them are not applicable or simply does not ease our work. Visio is used only as visualization program.

Starting with the Basic Flowchart template, Visio displays a new, blank drawing page and opens several stencils that contain the shapes needed to create a flowchart. Draging a shape from the stencil onto the page, which has a grid that helps align the diagram.

![Visio drawing page and stencil with components](image)

**Figure 3.3 MS Visio.**

This process is referred to as "drag and drop." That is the fundamental idea behind everything in Visio. Perhaps even more important is the idea of connecting shapes. When draging additional shapes onto the page to create a diagram such as a flowchart, special lines called connectors connect the shapes and stay attached when shapes are moved around. By arranging and connecting shapes on the page, one can rapidly assemble a diagram that can be pasted into another document, such as a report or presentation slide; save as a Web page; or print.
Shapes in Visio look a lot like clip art. In fact, shapes have built-in intelligence; their "smarts" makes them work in uniquely appropriate ways. For example, auto-routing lines are used to connect process shapes in a flowchart. When moving a process shape, all the lines stay connected and reroute around other shapes as necessary.

The shapes are smart in other ways, as well. For example, a door shape in an office layout can swing in or out by using a single command; valve shapes can rotate into place automatically on a pipe; milestone shapes can shift position on a timeline as dates adjust. Maybe these feathers are not necessary in an electrical scheme but the fact that all data and parameters building a component can be hidden behind every shape and that they are accessible during the drawing considerably simplifies the configuration work.

From the facts above we see that Visio is a multipurpose software, implemented to adapt to as many businesses as needed. All the included libraries are useful but they don’t include advanced and complex components that are used at ABB to configure their terminals. Most of the components used in the terminals build at SA are also defined and developed at SA.

So the base requirement from Visio is to produce stencils/libraries with components/function blocks. These blocks will provide users with a graphical interface that will ease configuration of terminals.

### 3.3 Hardware

The Base500 platform provides an environment for applications to operate and form products. The terminal hardware is currently Pentium III based, using the VxWorks operating system. It is a single process, multi-threaded system, where each thread may enter several states. The platform supports other operating systems and CPUs as well, such as PowerPC and Win32 systems.

### 3.4 Summary

AFL components which are the building foundation of terminals are connected to the hardware by Base 500 software. MS Visio and the ACT will be a part of this software.

![Figure 3.4 Terminal setup.](image-url)
4. Building blocks of the ACT

This chapter explains main blocks of the software and the functionality implemented.

Below all the functions implemented are described briefly and in appendix 2 a simple flowchart presents the course of action. Underneath the following functions there must be an understanding of how Visio objects are created and managed. Aspects such as Event handling, COM objects and object wrappers are highly important. Therefore descriptions of these methods are included in this report.

A wrapper application is designed, implemented and documented that will start a Visio instance and connect to the Visio application through the Visio Automation (COM) interface. The following operation modes are implemented.

- AFL Component Stencil builder
- From a database containing a description of application function blocks (AFL) definitions, a VISIO stencil are generated and contains as master shape for each AFL type which includes:
  - AFL name and type ID
  - Inbound connection point description with name, data type and optional semantics.
  - Outbound connection points description with name, data type and optional semantics.
  - Settings/Parameters for component with identifier, textual name, type and the default value.

Function block stencil Builder is implemented and added. This builder creates a function block stencil based on the AFL stencil where the user can build new function block masters from the AFL stencil masters. The user is be able to:

- Assign a function name and type ID to the FB master.
- Create FB internal connections (relative, hidden)
- Verify type and semantics of connection terminals.
- Define externally visible connection points
- Define externally visible settings/parameters and define default value overrides.
- Reposition the AFL's graphics representation in order to provide a clear visual representation of the FB.
- The wrapper application can verify type and signal semantics before a connection is accepted.
Product template builder is implemented. The Product template builder can from a database containing a description of FB and AFL instances in a specific product class, as well as information about the task in which they are executed, create a VISIO product template that contains the maximum configuration for the product. The product template contains:

- A VISIO page for each task. The page provide task properties such as taskID, cycle-time and task priority.
- On each page, an instance of each FB or AFL master shape from the relevant stencil with type and instance information is instantiated.
- All relative, internal connections are expanded to an absolute path.

An offline configurator is designed and implemented. Based on existing terminal-specific drawings, or a product specific template, a terminal specific drawing is opened or created. All output->input connections from the template shall be generated. The user is be able to:

- Create new connections between existing FB and AFLs externally visible connection points. Verify type and semantics of connection terminals.
- Edit all externally visible settings.
- Remove unused FB and AFLs.
- Reposition FB and AFLs on the page to create a suitable graphical representation of the configuration.
- The VISIO wrapper is able to:
  - Prevent the user form adding shapes that are not in the template.
  - Create a terminal specific database containing:
    - View all tasks and the task properties (from page with AFL or FB)
    - View all AFLs contained in the product instance and the position on the page.
    - Absolute connections between AFL inputs/outputs.
    - Settings for all AFL's (default or product specific)

An RCS/VSS managed terminal specific database loader file (XML) from the generated database is produced.

Online configurator. The VISIO wrapper reads the configuration from the terminal database using the ODBC interface and create a drawing based on the AFLs and connections in the terminal and the stencil masters in the AFL stencil created in 1. The settings shall be stored in the PropertySheet of the AFL shape instance.
Online monitor mode is implemented. A Visio document is create, no editing can be performed, but connections are colored according to the value of the output terminal that they connected to. Connection between analog terminal points displays the actual value. The ShapeSheet containing the settings for the component can be updated to reflect the actual value of the setting in the terminal. It is possible to change the settings editing the Custom Properties of the shape. The new values can be transferred to the terminal

Performing the described tasks should be done easily without confusing the user. Therefore a startup menu is implemented and the functions above are sorted into different categories.

At the moment there are six categories implemented:

1. Open: Simply opens an already existing stencil/function block or a drawing/configuration.
2. Upload: This function will copy a terminal database to a local database. This action is done because of three important reasons. First to reduce traffic over the network, second to reduce time to receive data and last to avoid collisions (when two users try to write to the same database).
3. Create FB: Software will read PID-database and create the stencils necessary for configuring the terminal.
4. Reverse Engineer: Present configuration from the terminal is fetched and visualized for altering.
5. Debug: Does the same action as Reverse Engineer but no altering is allowed. At the same time connections will be colored depending on their analog or digital value.
6. Save and create XML: The new configuration will be saved on the database and sent to terminal. An XML-tree will be created for other tools.
The thesis software consists of three major blocks and they are implemented using COM and C++ which are explained later. The interaction between the blocks should be error free to make the software stable.

![Diagram of ACT main programming blocks]

**Figure 4.2 ACT main programming blocks**

In the main-loop of the first action is to start MS Visio and instances of data-storage, database and Visio handler classes. In order to gather information from the terminal and other databases, locally or remotely, some routines are needed. These routines are mainly implemented in the Database Routine block. It consists of functions for building SQL-commands, which is the language used for communication with the terminal. There are also methods that are responsible for processing the SQL-query and parsing the received information for further use.

When information is delivered from the Database Routines block it has to be visualized somehow in the MS Visio window. Later the user will alter the information/configuration to achieve his goal. Visio Handlers-block contains methods for communication between the ACT and MS Visio. These methods are responsible for awareness of actions taking place in the main window.

The Data Storage-block acts as a temporary storage space containing important information concerning used function blocks and the state of the software.

These three parts are the main blocks and implemented during the thesis.

Rest of this chapter explains vital information about how the code is designed and implemented to achieve the functionality described above. Still it is important to state that browsing the code will give the best insight of design and functionality and the information below describes methods and principals that should be followed.

### 4.1 Visio Handlers

Here Visio Handlers (Visio Event handling methods) are described.
4.1.1 Overview

This subchapter discusses how CommonView handles actions taking place in the GUI - such as invoking menus, scrolling windows, moving the mouse, closing an application and so forth.

The CommonView window environment is event-driven: the user directs the flow of the program by choosing menus, entering text and clicking various options. When something happens (an event) - such as a mouse click - the windowing system must determine what caused it and which window it occurred in.

The program must determine what action to take in response to the event, in other words, how to handle the event. This involves developing a range of functions to cope with actions such as selecting menu options, clicking on controls like RadioButtons, pressing the keyboard or entering text. Each event, therefore, has an associated 'event handler'.

When an event occurs, CommonView calls the associated event handler. For example, the function Menu Command() is invoked when the user chooses an item on a menu. (This function typically analyses which option have been chosen and processes it accordingly).

These actions will be processed by the ACT and the according reaction will take place.

Implementing an Event Handler

CommonView's Event Handlers are provided as virtual functions in the Window class. Since they are virtual functions, it means specific behavior can be written for them.

There is no need to introduce extra code to invoke the tailored event handler - CommonView and C++ automatically take care of that. If the function is not tailored, CommonView calls its default event handler whenever an event is generated.

The default implementations of event handlers are part of CommonView's dynalink library and are automatically brought in at the linking stage. CommonView requires the Exec() function to be called to set off this event handling mechanism.

The arguments passed to each event handler are the object representations of the event, based on CommonView classes. In effect they contain the crucial information concerning the nature of the event - the Who, What, Where and When aspects. Although this data is private, the event object's member functions provide the necessary access to obtain their values.

Working with an object's events differs from working with an object's properties and methods in one key respect: Properties and methods are both defined and implemented by Visio; events are defined by Visio, however, the programmer is responsible for writing the code to implement them.

The following illustration shows how one program calls Visio to implement properties and events, and how Visio calls the program to implement events.
An event has both a subject and a source, which are typically different objects. The subject of an event is the object to which the event actually happens. For example, the subject of a ShapeAdded event is the Shape object that was added.

The source of an event is the object that produces the event. Most events have several potential sources. The source object you choose determines the scope in which the event fires—the higher the source object in the object hierarchy, the greater the scope. For example, if the source is a Page object, the ShapeAdded event fires whenever a shape is added to that page. If the source is the Application object, the ShapeAdded event fires whenever a shape is added to any page of any document that is open in the Visio instance.

Obviously, the more often an event fires, the more likely it is to affect the performance of your solution. Therefore it is essential to think about the scope in which the event is handled.

*Figure 4.3 The interaction between Visio and your client program*
4.1.2 Interface

The Automation interfaces on MS Visio objects are defined in Visio.h. This file is a part of MS Visio SDK and can be downloaded from the internet. Visio.h contains a standard COM interface definition for each Visio object.

MS Visio also provides services in the form of wrapper classes that simplify programming Visio using C++. A wrapper class is so called because it encapsulates, or "wraps" the code involved in certain tasks, such as getting and releasing interface pointers and working with strings. The basic benefit of using these classes is that they keep track of AddRef and Release calls for the program, using C++ constructors, destructors, and assignment operators. When appropriate, these wrapper classes also automatically wrap any arguments or return values with a wrapper class.

Figure 4.4 Visio Object Model
4.1.2.1 Visio objects and COM interface

The objects that MS Visio exposes are Component Object Model (COM) objects. The concepts of an interface on an object and a reference to an interface are fundamental to understanding COM.

To illustrate an interface on an object and a reference to an interface, here is a simple example, expressed in pseudo code:

```pseudo
ipAppObj = <reference to an interface on a Visio Application object>
//Get documents collection
ipDocsObj = ipAppObj->Documents()
//Get first document
ipDocObj = ipDocsObj->Item(1)
```

Given a reference to an interface on a Document object, the program can obtain, in like fashion, a reference to an interface on a Page object, and then a Shape object, and so on.

![Figure 4.5 COM convention](image)

The program state after this code executes is shown in figure 4.5, which uses the common conventions for showing COM objects. The controlling program has obtained references to interfaces on three objects exposed by Visio. The arrows are the references, the circles are the interfaces, and the boxes inside the Visio instance are the objects.

COM provides many kinds of interfaces, such as those that support persistent data storage. A COM interface pointer refers to data that represents the object that owns the interface. An interface also refers to an array of functions that perform the actions defined in that interface for that object. After referencing to an interface on an object, methods defined in that interface for that object can get called.

The interfaces that Visio exposes are dual interfaces. In a dual interface, the first entries are identical to the entries in a standard IDispatch interface, the principal interface used to implement Automation. The IDispatch methods are followed by entries that correspond to the methods and properties exposed by the object. A dual interface has methods or properties that can be called either indirectly through IDispatch methods or directly through the "dual" methods.
IDispatch functions define a protocol that allows *late binding*—that is, binding that occurs at run time—between Automation controllers and Automation servers. However, if an Automation server provides a type library and implements dual interfaces (as Visio does), it enables *early binding*—that is, binding that occurs at compile time. This typically results in improved performance by the Automation controller, because the program makes fewer calls at run time to invoke a method.

4.1.2.2 Wrappers

To use the wrapper classes, Visiwrap.h is included in the project source files. This file is a part of MS Visio SDK and can be downloaded from the internet. The wrapper classes obey the following conventions:

Wrapper class names use the CVisio prefix. For example, the wrapper class for a Page object is CVisioPage.

Properties are accessed through methods that use the *get* prefix to read the property or *put* to set the property. For example, the methods for the Name property are getName and putName. (In Visio.h, the corresponding methods include an underscore between the prefix and the method name: get_Name and put_Name.)

Helper classes such as VBstr and Vvariant are defined in Helpers.h. VBstr is a class that simplifies working with BSTR variables, which are used to pass strings through Automation and VVariant class simplifies working with VARIANT variables, which are the Automation counterparts of C++ unions.

**Definition of wrappers**

The Visio.h file defines the objects exposed by Visio in standard COM interface declaration syntax. The wrapper classes defined in Visiwrap.h call the methods of these interfaces. The wrappers are already implemented and can be downloaded from Microsoft homepage and compiled in the code to access the object more easily.

Every object exposed by Visio has a similar declaration in Visio.h. Various macros in the declaration, which are not shown in this example, allow Visio.h to be included in either C or C++ source files.

Because every Visio interface derives from IDispatch, every Visio interface has the following methods:

- QueryInterface
- AddRef
- Release (for IUnknown)

These methods are followed by:

- GetTypeInfoCount
- GetTypeInfo
- GetIDsOfNames
- Invoke (for IDispatch)
4.1.2.3 Where to start

The ACT-application begins with starting an MS Visio instance and then vaoGetObjectWrap starts up the instance by allocating an applicationobject. To do anything with a Visio instance, you need an Application object, which is what vaoGetObjectWrap gets.

```c
CvisioApplication app;
if (VAO_SUCCESS != vaoGetObjectWrap(app))
    //Error handling
    goto CU;
```

The vaoGetObjectWrap function calls the vaoGetObject function, which is declared in Ivisreg.h and implemented in Ivisreg.cpp. The services defined in Ivisreg.h are for working with a Visio instance provide the necessary means to launch a new Visio instance or establish an Application object for the active Visio instance.

4.1.2.4 Returned Values

This part is to make the reader more familiar with the returned values.

Every method declared in Visio.h is specified to return an HRESULT that indicates whether the method has executed successfully. The HRESULT returned by a method declared in Visio.h is passed along by the equivalent method of the corresponding wrapper class defined in Visiwrap.h.

If a method succeeds, it returns NOERROR. A common practice is to check a method's result by using SUCCEEDED(hResult).

Object reference

Many methods return an object reference as their output value. This value is a COM interface pointer, which, like any interface pointer, must eventually be released. If using wrapper classes, the value returned is an object of another wrapper class in which the interface pointer is packaged. When the object goes out of scope, the Visio interface pointer it holds is automatically released. If not using wrapper classes, the interface pointer is held directly by the software written, which must explicitly release the pointer at the appropriate time.

If a method that returns an object reference fails, the output value again depends on whether wrapper classes are used or not. If using wrapper classes, the program still gets an object of the appropriate wrapper class, but the interface pointer held by the object is NULL. Calling the IsSet function on that object will return FALSE. If not using wrapper classes, the interface pointer is NULL, so the program can simply check for that.

Even if the method succeeds, the program might still need to check the output parameter. For example, if ActiveDocument is called when no documents are open, it returns an HRESULT of success and a NULL interface pointer (wrapped or not). The reasoning here is that an error did not occur—having no documents open is a perfectly valid state for which the caller should account. The various Active* methods behave in this manner, and verifying that their output values are not NULL before proceeding is necessary. The various Item and Add methods, however, always return a non-NULL reference if they succeed.
String
Several methods return a string to the caller. The Shape object's Name property (getName of CVisioShape or get_Name of IVShape) is one example. All strings passed to or returned by Visio methods are of type BSTR, which consists of 16-bit (wide) characters in Microsoft Win32 programs. The Visio engine allocates the memory for the strings it returns, and the caller is responsible for freeing the memory.

The wrapper classes, defined in Visiwrap.h, take care of freeing memory for strings. If the program do not use the wrapper classes, however, SysFreeString is called to free any string returned by a Visio instance.

4.1.2.5 Arguments Passed to Visio Methods

Passing arguments to Visio methods is straightforward:

- Integer arguments are declared as short or long, depending on whether they are 2-byte or 4-byte values.
- Floating-point arguments are declared as double.
- Boolean values are passed as short integers or as VARIANT_BOOL.
- Arguments that are object pointers, such as BSTR or VARIANT, merit further discussion.

Object pointer
Some methods take object pointers, and some require a pointer to a specific type of Visio object. The simplest way to pass an object pointer to a method is to pass a reference to an object of the appropriate wrapper class.

The interfaces defined in Visio.h declare object pointers as the corresponding interfaces. For example, Visio.h declares GlueTo as taking a pointer to an IVCell interface. Because the Drop method is not restricted to a particular object, Visio.h declares Drop to take an IUnknown interface, which means Drop takes a reference to any object.

String
Any string passed to a Visio instance must be of type BSTR. The helper class VBstr, defined in Helpers.h, is a convenient way to pass strings to Visio instances. VBstr allocates memory for the string when it is created and frees the memory when the VBstr is destroyed. If VBstr is not used, SysFreeString is called to free the memory that has been allocated for strings.
Variant

Some Visio methods take arguments that aren't constrained to a single type. For example, if you pass an integer 5 to the Item method of a Documents collection, it returns a reference to the fifth document in the collection. If you pass a string that is a document name to the same method, however, it returns a reference to a document of that name (assuming that the document is open).

COM defines a data structure known as a VARIANT for passing such arguments. The helper class, VVariant defined in Helpers.h, is a convenient way of passing a VARIANT to a Visio instance. For example, compare the following two statements:

```csharp
hr = pages.Item(VVariant(1L), page);
hr = masters.Item(VVariant("Position"), master);
```

The first statement passes 1 (an integer) to the Item method of a Pages collection. The second statement passes "Position" (a string) to the Item method of a Masters collection.

4.1.3 Event Object

A MS Visio Event object pairs an event with an action. Either to run a Visio add-on or to notify an object in the software (also called a sink object or an event sink) that the event occurred. The software declares Event objects that describe the events to handle and tell Visio what action to take. When that event occurs, the Event object fires, triggering its action.

This part of the code is the very essential, since without any knowledge about the actions taking place in the Visio window there can not be any reaction from the software. MS Visio SDK provides with necessary information and skeleton to be able to implement Event Objects. The first thing to do is determine the type of Event object that the solution requires. What is the source object? What events are needed to receive? How will the solution respond to the events it does receive? Once these decisions are made, an Event object can be created that runs an add-on or receives notifications.

4.1.3.1 Defining an Object

Before creating an Event object, one needs to decide the following:

- The scope that the Event object should fire in.
- The event or events to receive.
- The actions to perform when the event occurs—run an add-on or send a notification to an object in an already running program.

The scope

The scope determines the source object whose EventList collection the Event object is added to. Any object that has an EventList collection can be a source of events. For performance reasons, add the Event object to the EventList collection of the lowest object in the hierarchy that can fire the Event object in the scope.
Triggers
To indicate the event to handle, the \textit{event code} has to be added. When an Event object fires, the Visio engine passes the event code for the event that actually occurred, even if the Event object's event code indicates multiple events.

\begin{verbatim}
    evtList.AddAdvise(visEvtAdd + visEvtShape, sinkObj, "", "")
\end{verbatim}

Actions
The action determines which method is used to create the Event object, Add or AddAdvise. After deciding what the source object should be, the Event object can be created by adding it to the EventList collection of the source object.

To run an add-on or other external program, create an Event object using the Add method of the source object's EventList collection.

To send a notification, create an Event object using the AddAdvise method of the source object's EventList collection.

The following example is copied from the code and shows how you might create an event sink, get and set objects, and filter events that are passed to the event sink:

\begin{verbatim}
    vsoApp.EventList(vsoEList);
    CoCreateAddonSink(ReceiveNotifyFromVisio, &ipSink);
    vsoEList.AddAdvise((short)(visEvtShape | visEvtAdd), VVariant(ipSink), VBstr(_T("")), VBstr(_T("")), vsoAnotherEvent);
    ipSink->Release();
    ipSink = NULL;
\end{verbatim}

4.1.3.2 Receiving information
After creating an Event object, the information can be received by querying its properties. In addition, the EventInfo property of the Application object provides more information about certain events after they occur. For example, after the ShapesDeleted event fires, the name of the deleted shapes is received from the EventInfo property.

Because there is only one EventInfo property for potentially many events, the event to get handled must be specified when EventInfo is received. To do this, the event's sequence number (which Visio passes as the third argument when it calls VisEventProc on the corresponding sink object) is passed. If there is no additional information for the event you specify, EventInfo returns an empty string.

Notifications from an Event object
An Event object that sends a notification is created using the AddAdvise method of the EventList collection and can send a notification to a program that is already running. Creating this kind of Event object is done in the following ways:
An object is defined in the program to receive the notification when it is sent. This kind of object is sometimes called a notification sink or sink object.

An event procedure (VisEventProc) in the sink object to handle notifications is written to receive the event.

The program creates instances of sink objects and the Event objects in the Visio instance at run time. Because this kind of Event object uses references, it cannot be stored with a Visio document and must be created each time the program runs.

The following diagram shows how a program interacts with objects in Visio to receive event notifications.

In this diagram, pSource is a reference to the source object in the Visio instance. This is used to get a reference to the source object's EventList collection, which is assigned to pEvtList.

The program uses pEvtList.AddAdvise to create the Event object, which is assigned to pEvt. With AddAdvise, the program passes a reference to the sink object that the Visio instance sends the notification to when the Event object fires.

**Sink Object**

A sink object is a non-Visio object, which is defined to receive the notifications that the Visio instance sends. At a minimum, the sink object must be programmable (that is, it must support the Automation IDispatch interface) and must expose an event procedure named VisEventProc. The sink object can be altered with whatever additional functionality that makes sense for the software.
VisEventProc

In the class module, an event procedure called VisEventProc is written to handle notifications when they are received from Visio. Visio does not require the design of event handler to be in a particular way. Using any technique for branching within the procedure, and depending on the number and category of events the program will handle is authorized, and a different sink object for each event may be defined. When an Event object fires, the Visio instance calls the VisEventProc procedure for the corresponding sink object.

The event procedures implemented in the software alerts when shapes or connections are added or deleted from the drawing page. These shapes/connections will be examined to decide whether the operation is allowed or not. If not, a roll back function will take the configuration to the most recent accepted state.

Lifetime

Event objects created with the AddAdvise method persist until the Event object is deleted with the Delete method or all references to the source object are released, including references that are held indirectly through a reference to the source object's EventList collection or to an Event object in the collection.

When the Visio instance terminates, it issues a BeforeQuit event, which the program should handle by releasing all its references to source objects or performing any other cleanup tasks. After the Visio instance issues BeforeQuit, it releases all its references to sink objects in the program.

4.1.4 Handling Events

The topics in this section describe how to receive Visio events that were described in the section before. Here as well MS Visio SDK provides the developer with information and skeleton to be able to define and implement Sink objects. Although this protocol is specific to Visio, a C++ program do not need to implement entire event-set interfaces; instead, the C++ program can register for just the events of interest rather than every event in an event set.

Implementing

The sink object in the C++ program must be a COM object that exposes the IDispatch interface. The IDispatch interface must supply a method called VisEventProc that has the following signature:

```c
STDMETHOD(VisEventProc) ( 
    WORD wEvent, // Event code of the event that is firing 
    IUnknown FAR* ipSource, //Pointer to IUnknown on object firing the event 
    DWORD dwEventID, //The ID of the event that is firing 
    DWORD dwSeq, //The sequence number of the event 
    IUnknown FAR* ipSubject, //Pointer to IUnknown on event subject 
    VARIANT VextraInfo //Additional information 
); 
```
When AddAdvise is called to create the Event object, a pointer is passed to the IUnknown or IDispatch interface on the sink object.

Event objects created with AddAdvise persist until the Event object is deleted, all references to the source object are released, or the Visio instance is closed. If CVisioAddonSink is used in a Visio library (VSL), its unload handler must call Event.Delete.

The example below is copied from the code and gives insight on how different events are caught and handled. The example describes the actions performed when a new pages is added to the collection.

```c++
HRESULT STDMETHODCALLTYPE ReceivePageAddNotifyFromVisio (  
    IN  IUnknown* ipSink,   // ipSink [assert]  
    IN  short nEventCode,   // code of event that's firing.  
    IN  IDispatch* pSourceObj,  // object that is firing event.  
    IN  long nEventID,   // id of event that is firing.  
    IN  long nEventSeqNum,  // sequence number of event.  
    IN  VARIANT pSubjectObj,  // subject of this event.  
    IN  VARIANT vMoreInfo,  // other info.  
    OUT VARIANT* pvResult)  // return a value to Visio for query events.  
{
    LPVISIOPAGE  pPage = NULL;
    LPVISIODOCUMENT  pDoc = NULL;
    CStr  Oname;
    CVisioPages  pages;
    CVisioShape  sheet;
    CVisioCell  vsoCell;
    short  cellNr;
    short  *p_cellNr = &cellNr;
    p_cellNr = &cellNr;

    if (fin) { //if the finish flag is set
        if (SUCCEEDED(pSubjectObj->QueryInterface(IID_IVDocument,
            (LPVOID *) &pDoc)))
        {//and we have the active document
            CVisioDocument  doc(pDoc);
            CVisioPage  page;

            //get pages in the document
            doc.Pages(pages);
            pages.Item(Variant(1L), page);
            //get page name and sheets below
            page.getName(Oname);
            page.PageSheet(sheet);

            //and add appropriate page properties
            sheet.AddNamedRow(visSectionProp, VBstr("ApplicationId"),
                0, p_cellNr);
        }
    }
}
```
Libraries

A MS Visio library (VSL) is a special dynamic-link library (DLL) that is loaded by the Visio engine at run time. A VSL can implement one or more Visio add-ons, which are programs that use Automation to control Visio instances.

An add-on implemented by a VSL can interact with Visio objects in exactly the same fashion as an add-on implemented by an executable (EXE). An add-on implemented in a VSL has performance and integration advantages over one implemented in an executable program— one reason being that a VSL runs in the same process as the Visio instance. You cannot, however, run a VSL from Windows Explorer as you can an executable program.
Advantages

All else being equal, a VSL runs faster than an executable program. Because a VSL is a DLL, it is loaded into the process space of the Visio instance that is using the library. Calls from a VSL to a Visio instance do not cross a process boundary, as is the case when an executable program calls a Visio instance.

In addition, because a VSL runs in the same process as a Visio instance, it is much easier for it to open a dialog box that is modal to the process in which the Visio instance is running. When two executable files (an add-on and a Visio instance) are running, it is difficult for one to display a dialog box that is modal with respect to the other. An add-on executable program can display a dialog box, but the user can click the Visio window and change the Visio state while the dialog box is open.

Because of specific architecture demands the add-on (ACT software) is implemented as an executable file.
4.2 Database routines

Database interfaces are the user’s way to access the database. An interface normally contains functions for connecting to the server, making queries, performing transactions, and making changes to the database schema. However different interfaces can be specialized on specific types of operations, e.g., SQL is used mainly for schema modifications and queries, while a C/C++ API normally are specialized for transactional access. Figure 4.7 shows a typical interface configuration for an embedded database system.

![Figure 4.7 Connection types](image)

The most basic interface is the native interface. This interface, which is targeted for a specific programming language, e.g., C or C++, is often used by the application running on the embedded system. Microsoft has developed the Open Database Connectivity (ODBC) interface in the late 80’s. This interface uses the query language SQL, and is today one of the largest standards of database interfaces. The advantage with ODBC is that any database that supports ODBC can connect with any application written for ODBC. ODBC is a low-level interface, that requires more implementation per database access then a high-level interface like ActiveX Data Object (ADO) interface. The ODBC interface cannot handle non-relational databases very well. Main benefits, beside the interchangeability, is that the performance is generally higher than high-level interfaces, and the interface gives the possibility of detailed tuning.

4.2.1 ODBC Driver

The ODBC driver that is used is called memdb and is under development at ABB, still it is necessary to explain its functionality.
4.2.1.1 Role of the DRM Driver

The Database Record Manager Application Programming Interface (DRM API) is a C language interface that performs low-level database access services required by SimbaEngine. SimbaEngine depends on the DRM. In ODBC terminology, this is a single-tier driver, where all executable code resides in the current environment. All the program logic necessary to handle requests from an affront-end application is contained within the driver itself, including an SQL interpreter and database engine. Drivers built with SimbaEngine can also be used over a network, using client/server capabilities of SimbaEngine.

The following diagram shows the relationship between an ODBC-compliant application, SimbaEngine, the DRM driver, the Database Record Manager, and a non-SQL database.

![Figure 4.8 ODBC layers](image-url)
4.2.1.2 Linking ODBC Application to a Non-SQL Database

Each layer shown in Figure 4.2.1.2 has the following role in accessing database files:

An application accesses a database through ODBC API calls, some of which include SQL statement as arguments.

The Driver Manager (in Windows, odbc.dll) loads the ODBC driver for the application and then passes requests to the driver and results to the application, via ODBC function calls.

SimbaEngine processes ODBC function calls, parses SQL statements, and generates an optimal plan for accessing the database data. SimbaEngine then makes a number of lower-level calls to the DRM driver through the DRM API. SimbaEngine maps SQL data types to data types acceptable to the DRM driver.

The DRM driver presents a relational view of the native database for SimbaEngine, accesses database files either directly or indirectly through the services provided by the Database Record Manager, and returns results to SimbaEngine. The DRM driver reports any error conditions to SimbaEngine via the DRM API:

The Database Record Manager (provided by the database vendor) accesses database files directly and performs low-level services required by the DRM driver.

4.2.1.3 Tasks of the DRM Driver

SimbaEngine makes calls into the DRM API to perform the following tasks:

- Connecting to a data source
- Defining data, including:
  - retrieving meta data (data dictionary information) such as table and column names, data types, and index availability
  - creating and deleting tables and indexes in the native database
- Manipulation data, including:
  - retrieving data from database tables
  - updating database tables (adding new records, deleting records, and updating existing records)
  - moving sequentially through a table in both non-ordered and ordered (keyed) ways
  - locking data as appropriate in multi-user environments
- Disconnecting from a data source
- Performing general utility functions, including:
  - retrieving extended error string messages
  - comparing two string for equality
- Processing transactions (optional), including:
  - logging, committing and rolling back transactions
  - setting the transaction isolation level
- Processing push-down optimization (optional), including processing joins and filters that are pushed down by the SimbaEngine.
Conceptually, SimbaEngine associates four types of “object” with these tasks: sessions, databases, tables, and columns. These objects are arranged in the following hierarchy under each ODBC-compliant application:

![Figure 4.9 SimbaEngine objects](image)

**Session**
This is a connection to the DRM driver. When an application tries to establish a connection to the database through the ODBC, API call SQLConnect or SQLDriverConnect, SimbaEngine makes a number of DRM API calls into the DRM driver and a session is established. An application may initiate multiple sessions and multiple applications may be active at the same time.

**Database**
This is the grouping of the tables that an application accesses during an active session. Each session may access multiple databases. A database is usually a directory of tables, a file containing tables, or any other grouping of tables appropriate to the native database.
Table

A table contains one or more columns. SimbaEngine recognizes real, and virtual join tables. A real table contains actual data, stored locally or on a file server and in a format specific to the native database. A virtual table is created by the DRM driver through a DRM API function call and contains data dictionary information required by the SimbaEngine. To access data in a database, SimbaEngine requires meta data—that is, data dictionary information about tables, column, data-types and indexes. It is the responsibility of the DRM driver to organize this information into virtual tables for SimbaEngine.

A virtual table created by the DRM driver is not an actual table, so SimbaEngine does not call CreateTable or OpenTable function calls. However, to SimbaEngine, this table looks like a real table in the sense that SimbaEngine can move through it using DrmMove and retrieve column information.

A virtual join table (not to be confused with a virtual table) is a temporary table produced by a DRM driver in response to a push-down join.

Column

Each column in a real table or a virtual join table is stored in a native database-specific data type. The DRM driver maps this native database-specific type to the applicable SimbaEngine DRM data type. Each column in a virtual table is retrieved by SimbaEngine in a specified format.

SimbaEngine obtains unique identifiers for these objects from the DRM driver through specific DRM API function calls. These identifiers are used as arguments in subsequent DRM API function calls. The identifiers are allocated by the DRM driver and they are typically implemented as pointers to memory which the DRM driver uses internally to store information on the associated object.

For example, a column identifier might point to a structure containing state information, the column’s name, the data type of the column, and the width of the column. SimbaEngine does not modify the identifiers or any of the memory addressed by the identifiers. SimbaEngine expects table and column identifiers to be unique across multiple databases and database identifiers to be unique across multiple sessions.

DRM API Components

The basic components of the DRM API are:

- Function calls
- Support for SQL data types
- A pre-defined set of return codes and support for extended DRM error codes
- A mechanism for passing data to a DRM driver, or retrieving data from a DRM driver, through input and output buffers.
4.2.1.4 SimbaEngine Limits

The following limits apply to SimbaEngine:

- Number of rows: 2 billion
- Number of select list columns in a query: 300
- Number of columns in a table: 5 000
- Size of column: 2 gigabytes
- Number of connections: limited by memory
- SQL statement length: 16 K
- Number of statements per connection: limited by memory
- Table name length: 64 characters
- User name length: 128
- Column name length: 64 characters
- Table qualifier: maximum length limited by the operating system
- Number of joined tables: limited by memory
- Number of views SimbaEngine can store: 255

Concurrency

The timeliness of data, dynamic or snapshot, is determined by whether or not execution of a query results in sort. Queries with distinct, group by or order by will result in a sort by SimbaEngine, unless an index exists that satisfies the required ordering.

For those queries which do not result in a sort by SimbaEngine, the data fetched will be from the data files. For those result in a sort by SimbaEngine, the data fetched will be from a temporary table. The temporary table is built from the required data in the original data file that at SQLExecute time.

Temporary tables greater than 64K will be written to a temporary file in the directory indicated by the application user’s TEMP environment variable. If there is not enough disk space on the user’s system to write the temporary file, the sort will fail.
4.2.2 MFC

So far ODBC-driver provides the connection between the database (remote or local) and the local PC on which the software is running. However connecting the ACT-application to ODBC-driver needs an interface. The answer is Microsoft Foundation Classes (MFC) and therefore an database control class was implemented. The classes in MFC, taken together, constitute an “application framework”, the framework of an application written for the Windows API.

The ODBC-class in MFC contain a CDatabase object that represent a connection to a database source, through which any software can operate on the data source. A data source is therefore a specific instance of data hosted by some database management system (DBMS).

![Connection map](Figure 4.10)

CDatabase class members such as Open, Close, ExecuteSQL, IsOpen etc. are used to implement such a portal for communication between ACT and data source. For further information see MFC library reference [7].

Different functions are implemented to read, write or alter different tables in the database when changes occur. These functions basically build appropriate SQL-commands with appropriate data which are run through the ODBC-driver.

One example is InitDb, which opens a connection to a specified database. The function is described below:

```cpp
bool DB::InitDb(CString dsn, bool loc_rem){
    CString openBase="DSN=newdsn;UID=\#;PWD=\"", openCommand;
    bool success=true;
    openCommand = openBase;
    openCommand.Replace(\"newdsn\", dsn);

    if(!loc_rem) //check if it local or remote database
        MessageBox(GetDesktopWindow(), \"Enter login and password for local database.\", \"Local DB\", MB_OK);
```

40 (55)
try{ //check if local DB is open
if(!dBase_Local.IsOpen())
    //if not, open and lock it
    dBase_Local.OpenEx(openCommand);
    success = true;
}
catch(CDBException* e){
    //if failed, return error messages
    MessageBox(NULL, e->m_strError, "failed to open DB",
    MB_OK);
    e->Delete();
    success = false;
}
}

else{
    MessageBox(GetDesktopWindow(), "Enter login and password for
terminal database.", "Terminal DB", MB_OK);
    try{ //check if remote DB is open
        if(!dBase_Terminal.IsOpen())
            //if not open and lock it
            dBase_Terminal.OpenEx(openCommand);
            success = true;
    }
catch(CDBException* e){
        //if failed, return error messages
        MessageBox(NULL, e->m_strError, "failed to open DB",
        MB_OK);
        e->Delete();
        success = false;
    }
}
return success;

Other functions can be found in the code. These are:
_InitDB():bool
_GetInfoFromDB():Database
_Connect():bool
_AddCompToDB():bool
_AddConnPtsToDb():boolean
_AddParamToDb():boolean
_AddConnToDB():boolean
_CloseConnection():boolean
_LoadComponents():boolean
_LoadParams():boolean
_LoadSettings():boolean
_LoadInputs():boolean
_LoadOutputs():boolean
_LoadConnections():boolean
_CopyDb():boolean
4.3 Data storage

The software needs some temporary storage space. To store, data linked nodes are used. These nodes represent components and contain vital information such as number of used components, active pages or simply to store parsed data from database for specific component types.

![Node chain diagram]

Figure 4.11 Node chain

Operations such as add, read and delete are implemented for using the structure. During the thesis, suggestion was made to implement a balanced three and reduce the search time while looking for an element. But since the structure is not going to contain many elements such feature would be unnecessary. But it would be wise to have this in mind when the number of elements increases.
5. Conclusions

5.1 Comparison between result and goals

When this thesis project started, the goals and the expected results were vague. The project's problem formulation and delimitation were not strictly set. But the project developed during the first month of work and functions that are examined in this thesis was decided once half of the project time had passed. Overall the work has brought an understanding of substation control, and substations in general.

One of the goals was to make an investigation about whether the existing ODBC driver could be used to monitor this real-time system. It showed however, that the driver is very poorly documented and most of the information is based on discussion with co-workers, directly from testing the driver or examining the code. Some information was found from seminar papers, but the subject felt old compared to the information provided from ABB sources.

This new concept has made an impact on co-workers at ABB/SA, even though this software is only a prototype and doesn’t include all functionality to replace the previous tool (CAP 540) today. After approximately 12 weeks a demonstration was held for all involved people with great response. Shortly the news reached Finland where ABB’s actual tool-development unit is placed. Therefore a videoconference took place and the result was again presented.

After demonstrations, many new requirements were added. People started to see new vision using MS Visio in different business application and MS Visio had to be examined under the microscope. These new visions put the project out of balance and time-consuming researches for additional requirements took place even though they were not a part of the original requirements. But it is very likely that this project will stay alive after the thesis project is ended and Visio will be adapted to suit the business plan from ordering a product to configuring and developing new components.

As mentioned above, time was “spend” on new requirements but still the thesis was finished with all original requirements implemented.

5.2 Learning's from the project

The project has been satisfying in the way that it has given knowledge in an area that has been new to me. It has also given more knowledge in control terminals and the secondary equipment on the level below, like circuit breakers, and sensors and actuators.

The project has been using different software libraries. New to me was MFC, which has made it much more simple to implement the functions. The library is used both for ODBC connection and for notification boxes.

Using Visual Studio C++ 6.0, which was a new tool, caused some problems. During my time at Linköpings universitet all programming has been done on Solaris machines and therefore Emacs and gcc felt like a better tool at the beginning of the project. But since the software is intended for Microsoft programs, some advantages when using VS C++ appeared.
5.3 Further work

Even though all requirements were implemented, this software is not yet a substitute to the already existing CAP 540. Some additional work such as adding more error/exception handling, making it user-friendlier, etc has to be done. Implementing new pioneering functions would add great value when marketing the software.

At the beginning of the project it was decided to use ODBC- interface. Using this low-level interface requires more implementation per database access compared with a high-level interface such as ADO (ActiveX Data Object). Generally a low-level interface performs better than a high-level, but since ADO is based on COM the advantages using it would be of greater value.

The storage manager saves data in lists without any specific order, and searching large trees are expensive. Due lack of time the planned binary tree-algorithm has not been implemented.

Furthermore the code may need some kind of clean up and optimization. Since the project included several different blocks, the code was left as it was when it executed correctly. This does not mean that the most optimal code is implemented. Since this project only aims at a prototype, its main purpose was only to confirm the possibility of creating such a tool.

Here are some other tasks that are either started or should be implemented:

- Tool should be able to handle several configurations at the same time. This task has been started but not finished.
- Master properties should be implemented. These will control setting for all components on the drawing page.
- Names for in/outputs should be visualized.
- Print settings, when a drawing doesn’t fit in one page.
- To make Metadef run on different ports, or to be able to access the functions directly.
6. References

This thesis report is based on the following documents:

Literature:


World Wide Web:


This thesis report is based on information provided from following persons:

[a1] Arve Sollie, ABB Automation Technology Products AB.

[a2] Lasse Kinnunen, ABB Automation Technology Products AB.

[a3] Johan Sälj, ABB Automation Technology Products AB.

[a4] Hans Karlsson, ABB Automation Technology Products AB.

[a5] Carl Byman, ABB Automation Technology Products AB.
APPENDIX 1

Requirements added during the project

1. Stand-alone:
   The software shall not be integrated with other applications. It is possible that this software will be part of a universal configuration tool and therefore cannot depend on other software.

2. Version handling:
   This new function was desired shortly after the first presentation. The function will show the differences graphically between the present configuration on the target terminal and the newly made configuration made with ACT.

3. Business system:
   After the first presentation, the business unit at ABB saw opportunities to ease ordering procedures by using MS Visio. Therefore, a prototype, which could handle their requirements, was implemented. This software is not presented in this thesis report due it was not a part of the thesis and was implemented mostly on residual time.
APPENDIX 2

Navigation Menu

Add-on context

MS Visio context

Open drawing/FB

Copy DB

Create FB

Reverse engineer

Save & make XML

Enter Path

Enter DSN to local DB

Enter DSN to PID DB

Enter DSN to DB

Write to local DB

Open DB session

Open DB session

Open DB session

Open DB session

Get relative info

Read DB

Read DB

Put new info

Local DB created

Generate comp. from DB

Generate comp. from DB

Create XML tree

Visualize in Visio

Generate connections from DB

Upload parameters & settings

Visualize in Visio

New drawing page or stencil

Changes made in the configuration

Throw registered events to handler

Close/Save

New drawing page or stencil

Changes made in the configuration

Throw registered events to handler

Close/Save

New drawing page or stencil

...
APPENDIX 3

COPYRIGHT NOTICE. Copyright © 2003 Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 U.S.A. All rights reserved.

MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE INFORMATION CONTAINED IN THE DOCUMENTS AND RELATED GRAPHICS PUBLISHED AS PART OF THE SERVICES FOR ANY PURPOSE. ALL SUCH DOCUMENTS AND RELATED GRAPHICS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS HEREBY DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS INFORMATION, INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF INFORMATION AVAILABLE FROM THE SERVICES.

THE DOCUMENTS AND RELATED GRAPHICS PUBLISHED ON THE SERVICES COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN. MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED HEREIN AT ANY TIME.

IN NO EVENT SHALL MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF SOFTWARE, DOCUMENTS, PROVISION OF OR FAILURE TO PROVIDE SERVICES, OR INFORMATION AVAILABLE FROM THE SERVICES.

COPYRIGHT NOTICE. Provider information/Impressum © Copyright 2003 ABB. All rights reserved

ALL INFORMATION PUBLISHED ON THIS WORLD WIDE WEB SERVER IS PROVIDED AS IS WITHOUT ANY REPRESENTATION OR WARRANTY OF ANY KIND EITHER EXPRESS OR IMPLIED INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES FOR MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. ANY ABB DOCUMENTATION MAY INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES AND ADDITIONS MAY BE MADE BY ABB FROM TIME TO TIME TO ANY INFORMATION CONTAINED HEREIN.
APPENDIX 4

Code (see separate folder ABB_Visio_Final)
Design and implementation of a Function Block configuration tool

Abstract

In high voltage substations there is an increasing demand for computerized equipment, and automation of operation and supervision. The future primary equipment will provide the possibility to collect more information and to provide better control for improved effectiveness. Since ABB has got a large customer target, each product must be custom made to fulfill the customers (often from different countries) demands. Later these products will be configured by engineers who may not be familiar with the tools used at ABB. This is a very expensive and time consuming. To solve the problem, ABB was looking for a low cost, easy managed and more advanced replacement. This report includes an investigation for a function block configuration tool based on Microsoft Visio. The tool is connected to a general database using ODBC for the latest ABB Substation Automation SW platform for protection in power transmission equipment. The thesis report mainly contains a description of the functions and features that are requested. Then methods used to build the prototype is explained and at the end the result is presented. At the same time the possibilities and problems that will arise from using ODBC based interface for control and monitoring of real-time systems were investigated.

Nykkelord

High voltage, ODBC, COM, Visio, database