SIMULATION ENVIRONMENT
FOR A
REAL-TIME POWER SYSTEM SIMULATOR

by
ROBERTO ROSALES S.
Ing. Electrónico, Universidad Simón Bolívar, 1990

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
DEPARTMENT OF ELECTRICAL ENGINEERING

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
August 1997
© Roberto Rosales S., 1997
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical Engineering

The University of British Columbia
Vancouver, Canada

Date: Sept. 3, 1987
Abstract

Work is in progress at Power Systems Simulation Laboratory of The University of British Columbia to implement a computer-based general-purpose power system simulator called OVNI, for Object Virtual Network Integrator. In this report we present our contribution to this project, particularly, in the selection of hardware and software platforms, the specification of the simulation environment, and the implementation of the first development phase of such environment.

Selection criteria to choose a common hardware platform for an electrical network solver with real-time performance, and a fully interactive interface are explained. A formal long term specification, to be used in two development phases of OVNI's Interface, is presented. Design guidelines to implement a friendly and practical user interface are described, and the implementation of the first phase of OVNI's User Interface is explained.
# Table of Contents

Abstract .................................................................................................................. ii  
Table of Contents ..................................................................................................... iii  
List of Tables ........................................................................................................... vi  
List of Figures .......................................................................................................... vii  
Acknowledgments .................................................................................................... viii  
I. Introduction ........................................................................................................... 1  
   1. What is OVNI? ...................................................................................................... 1  
      1.1. Features .......................................................................................................... 1  
      1.2. Areas of application ....................................................................................... 3  
   2. Scope of this thesis ............................................................................................. 4  
   3. On terminology ................................................................................................... 5  
II. Hardware and Software Platforms ....................................................................... 6  
   1. Hardware ........................................................................................................... 6  
   2. Software ............................................................................................................ 8  
III. Specification of OUI, OVNI’s User Interface ......................................................... 10  
   1. Definition of OUI .................................................................................................. 10  
   2. Development, Operating and Maintenance Environments ................................ 11  
   3. External Interfaces and Data Flow ...................................................................... 11  
   4. Functional requirements .................................................................................... 12  
       4.1 Scope of the interface ..................................................................................... 12  
       4.2 Description stage .......................................................................................... 12  
          4.2.1 Appearance of the interface during the Description stage ......................... 13  
       4.3 Building a schematic .................................................................................... 16  
          4.3.1 To open a schematic page ........................................................................ 16  
          4.3.2 To add electrical components to the schematic ....................................... 16  
          4.3.3 Placing and moving Components on the schematic page .......................... 17  
          4.3.4 Setting of electrical parameters ............................................................... 18  
          4.3.5 Connecting Components ....................................................................... 18  
          4.3.6 Disconnecting Components .................................................................. 19  
          4.3.7 Setting node names ............................................................................... 19  
          4.3.8 Setting and removing probes ................................................................. 19  
          4.3.9 Setting and removing break points ......................................................... 20  
       4.4 Handling schematics ..................................................................................... 20  
          4.4.1 Saving the schematic ............................................................................... 20  
          4.4.2 Viewing and using Components from other schematics ......................... 20  
       4.5 Simulation stage ............................................................................................. 20  
          4.5.1 Types of simulations and modes of operation ........................................... 21  
          4.5.2 Appearance of the interface during the Simulation Stage ......................... 22  
          4.5.3 Network sanity check ............................................................................. 22  
       4.6 Running and controlling the simulation ......................................................... 22  
   5. Performance requirements ................................................................................. 23  
   6. Scope of first and second development phases .................................................... 24  

iii
IV. Implementation of OUI, first phase ................................................................. 26
  1. Design and implementation of OUI .............................................................. 26
    1.1 Design guidelines ..................................................................................... 26
      1.1.1 Look and feel .................................................................................... 26
      1.1.2 Structural considerations ................................................................... 28
        A. Production of a single executable file ................................................ 29
        B. Optimizing the use of System Resources .......................................... 31
        C. Designing for upgradability and code maintenance ....................... 32
    1.2 Software Architecture .............................................................................. 33
      1.2.1 Basic Functionality ............................................................................ 33
        A. Main Form or Main Window ................................................................. 34
        B. Schematic Area .................................................................................... 34
        C. Schematic sizes .................................................................................... 34
        D. Components .......................................................................................... 35
        E. Component Palette and Component List ........................................... 36
        F. DataForms ............................................................................................. 36
        G. Nodes and Branches ............................................................................ 37
        H. Full View Window ............................................................................... 38
        I. Printing and Print Preview mode ......................................................... 38
      1.2.2 Component library .............................................................................. 39
      1.2.3 Data handling ..................................................................................... 39
      1.2.4 Code organization .............................................................................. 40
  2. Implemented Features .................................................................................. 42
    2.1. Title Bar and size of Main Window ...................................................... 43
    2.2. Menus ..................................................................................................... 43
    2.3. Top Panel ................................................................................................ 47
    2.4. Speed Buttons ......................................................................................... 47
    2.5. Component Palette ................................................................................ 48
    2.6. Schematic Area ...................................................................................... 50
    2.7. Status Bar ............................................................................................... 52
    2.8. Full View Window .................................................................................. 53
    2.9. Component List ...................................................................................... 54
    2.10. Handling Components ......................................................................... 55
    2.11. DataForms ............................................................................................. 59
    2.12. Connections .......................................................................................... 61
    2.13. Printing .................................................................................................. 66
    2.14. File Management .................................................................................. 69
    2.15. Leaving the program ............................................................................ 71

V. Future Work .................................................................................................... 72

VI. Conclusions ................................................................................................... 75

VII. Bibliography ................................................................................................ 77

APPENDIX I Guide for adding new Components to the Interface .................. 80
  1. Getting organized ......................................................................................... 80
  2. Creating graphic symbols .......................................................................... 81
  3. COMPEDIT.EXE ......................................................................................... 82
3.1. What is COMPEDIT? ................................................................. 82
3.2. Program description ............................................................. 83
3.3 Using the program ................................................................. 85
4. Updating OUI’s code .............................................................. 85
5. Creating a DataForm .............................................................. 87
APPENDIX II  Glossary .............................................................. 89
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>OUI's Specification: Menu items and their options</td>
<td>14</td>
</tr>
<tr>
<td>Table 2</td>
<td>Specification of the Component Palette: minimum set of Components.</td>
<td>15</td>
</tr>
<tr>
<td>Table 3</td>
<td>Implemented features: File Menu item.</td>
<td>44</td>
</tr>
<tr>
<td>Table 4</td>
<td>Implemented features: Edit Menu item.</td>
<td>44</td>
</tr>
<tr>
<td>Table 5</td>
<td>Implemented features: View Menu item.</td>
<td>46</td>
</tr>
<tr>
<td>Table 6</td>
<td>Implemented features: Help Menu item.</td>
<td>46</td>
</tr>
<tr>
<td>Table 7</td>
<td>Implemented features: Pop Up Menus.</td>
<td>47</td>
</tr>
<tr>
<td>Table 8</td>
<td>Implemented features: Speed Buttons.</td>
<td>48</td>
</tr>
<tr>
<td>Table 9</td>
<td>Implemented features: Component categories</td>
<td>50</td>
</tr>
<tr>
<td>Table 10</td>
<td>Implemented features: Schematic Pop Up Menu.</td>
<td>51</td>
</tr>
<tr>
<td>Table 11</td>
<td>Implemented features: Component Pop Up Menu.</td>
<td>58</td>
</tr>
<tr>
<td>Table 12</td>
<td>Implemented features: Branch Pop Up Menu.</td>
<td>64</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Some OVNI applications and configurations</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Distribution of pages for each schematic size</td>
<td>35</td>
</tr>
<tr>
<td>Figure 3</td>
<td>A node made of several Branches</td>
<td>37</td>
</tr>
<tr>
<td>Figure 4</td>
<td>OUI, Main Window (no grid)</td>
<td>42</td>
</tr>
<tr>
<td>Figure 5</td>
<td>OUI, Title Bar</td>
<td>43</td>
</tr>
<tr>
<td>Figure 6</td>
<td>OUI, Main Menu</td>
<td>44</td>
</tr>
<tr>
<td>Figure 7</td>
<td>OUI, Main Menu -&gt; Drawing Area</td>
<td>schematic sizes</td>
</tr>
<tr>
<td>Figure 8</td>
<td>OUI, Schematic, Component and Branch Pop Up Menus</td>
<td>47</td>
</tr>
<tr>
<td>Figure 9</td>
<td>OUI, Component Palette</td>
<td>48</td>
</tr>
<tr>
<td>Figure 10</td>
<td>OUI, Status Bar</td>
<td>52</td>
</tr>
<tr>
<td>Figure 11</td>
<td>OUI, Full View Window</td>
<td>53</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Component List</td>
<td>54</td>
</tr>
<tr>
<td>Figure 13</td>
<td>OUI, Sample Components</td>
<td>55</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Sample Resistor DataForm</td>
<td>60</td>
</tr>
<tr>
<td>Figure 15</td>
<td>OUI, Deleting Branches</td>
<td>66</td>
</tr>
<tr>
<td>Figure 16</td>
<td>OUI, Print Preview Mode</td>
<td>67</td>
</tr>
<tr>
<td>Figure 17</td>
<td>OUI, Top Panel in Print Preview mode</td>
<td>67</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Drawing Component symbols and placing terminals</td>
<td>82</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Snapshot of COMPEDIT</td>
<td>84</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Updating OUI’s code, the files contain instructions to follow</td>
<td>86</td>
</tr>
</tbody>
</table>
Acknowledgments

I would like to thank several people for their contribution to this thesis:

- The Duggan family for taking me into their home and being patient with my unusual nocturnal programming habits. Thanks Shirley, Barry, Kelly, Dave, Mick and Alex!
- My family, back in Venezuela, for their care and long distance encouragement via e-mail.
- To Dr. J. R. Martí for his supervision and financial support,
- To Luis R. Linares for his enthusiasm, help and guidance, and
- To all my fellows in the UBC Power Group for their help.

Roberto Rosales S.
I. Introduction

1. What is OVNI?

OVNI, for Object Virtual Network Integrator, is a computer-based general purpose power system simulator that is being developed by the Power Systems Simulation Group of The University of British Columbia. It is an EMTP\textsuperscript{1} type simulator for real-time performance, with an interactive simulation environment.

OVNI is made up of two main programs: The Network Integrator (NI), and OVNI's User Interface (OUI). The integrator is OVNI's network solution engine. It is aimed to be a very-fast continuously-running solver. To take advantage of the local EMTP modeling expertise, the NI has been designed to support a library of models that can be upgraded or expanded. The core of the NI is currently under development as a Ph.D. thesis. The Interface (OUI), is a single unified environment where the user can: describe a network to be simulated, launch and control a simulation, observe simulation results and interact on-line with the simulation process. The development of OUI is treated in this report. The discussion that follows provides an explanation of OVNI's main features and of its potential areas of application.

1.1. Features

For many years, the EMTP has been the standard tool used by the power industry for analysis of transient phenomena. The EMTP solves a network at the level of its differential equations, by modeling each component with a discrete equivalent circuit; then the resulting equivalent network is solved in the time domain applying numerical methods to the simultaneous solution of all equations. This approach provides a complete and dynamic solution for variables in the network. In principle, a complete solution would be useful for different types of studies. In practice, however, limitations in computational tools and solution

\textsuperscript{1} Electromagnetic Transients Program
speeds, impose the use of other approaches based on quasi-steady-state solutions for non-transient analysis of large and complex networks.\(^2\)

OVNI uses the solution method of the EMTP, to implement the concept of a very-fast continuously-running simulator, which would allow the study of a wide range of phenomena and opens the door to several potential applications.

OVNI is also intended for real-time simulation. Real-time simulation of electrical networks is a hard real-time problem that imposes solution steps as low as 50\(\mu\)s for transient analysis. Considering the amount of computations involved, a rather natural but inconvenient approach to follow is the use of parallel-computer or supercomputer architectures, with the philosophy of matching increasing network complexity or decreasing solution time steps with extra computational power. This approach might be suitable for problems that involve a vast number of computations that can be performed simultaneously, and where the processing time is considerably more important than the response time. However, for the simulation of electrical power networks this approach is not optimum. One reason for this is that in the solution of networks, switching conditions can cause configuration changes, this requires breaking the system into sub-blocks and reassigning them to different processors, which in turn causes an overhead on execution time. Other causes of unwanted overhead are communication and exchange of data between different processors, and general coordination of the whole solution process.

The approach OVNI follows is to optimize the solution algorithms, and use hardware/software matching strategies to reduce overhead delays. The solution proposes to match the topological segmentation [1] of an electrical system to the architecture of a network of off-the-shelf computers.

Another important feature of OVNI is that of interactive simulation. Once the simulation is started the user can interact directly with the system on-line, without having to stop the simulation. Through a unified interface the user can make topological changes to the

---

\(^2\) For example, Transient Stability programs.
system, schedule events, and monitor electrical variables ‘on-the-fly’, as the simulation proceeds.

A final distinctive feature is the use of a single graphical and integrated interface for all types of simulations: an interface that operates in a Windowing Environment, and is the same for all stages and applications of OVNI, from the schematic description of an electrical network, to the execution and control of the simulation, and the plotting of results.

1.2. Areas of application

1. Stand-alone tool for power system analysis. For off-line/on-line, non-real-time simulation. Without real-time performance requirements, OVNI can be hosted by a minimum of one to two machines\(^3\). In off-line simulation the NI solves a network from time zero to a specific time, stops, and then shows results for all time steps in the range of the simulation. In on-line simulation the network is solved continuously and the user can interact with the circuit.

2. Stand-alone tool for power system planning. OUI has been conceived to support a library of generic components, that can be used to generate sub-circuits. With sub-circuits it is possible to implement blocks and elements of an existing power network.

3. Training tool for power system operators and educational tool for students. Using the minimum configurations of one or two machines, OUI can be installed in desktop workstations, under Windows95/WindowsNT, and be used as an interactive training tool. With an on-line simulation continuously running, OVNI provides a simulated breadboarding approach for the analysis of electrical power networks.

4. Online power system security and control. This application is based on the concept of OVNI’s synchronized simulation. Running on-line, OVNI is used to solve a network in parallel to a real system. The simulation is synchronized with the outside world by receiving, periodically (e.g. every 10 minutes), steady state values from the real system. The steady state values are used as initial conditions from which the NI will solve the

---

\(^3\) One machine is enough for off-line simulation and on-line simulation of simple networks. Two machines would allow for complex on-line simulations, with one machine hosting OUI and another the NI.
network until the next synchronization occurs. The aim of this is to provide an on-line simulator with the latest condition of the real-system, such that in the case that a fault occurs the simulator can be used to explore and suggest corrective actions. Work proposed in [2], aims to combine this type of simulation with the support of an expert system that can predict and correct possible faults. To do this, a fuzzy-logic expert system (FLEX) is used to make a preliminary assessment of a system security index. The simulator then is used to automatically validate possible faults, and possible corrective measures.

5. Online tool for power system wheeling. Performing also synchronized simulation, the simulator can be used to quickly evaluate the cost and effect of different energy routing alternatives.

6. Real-time equipment testing. With the simulator running fast enough to fake a portion of an electrical network, it can be connected to real equipment, through D/A and A/D converters, to study the behavior of that equipment as if it was under real circumstances. This is the most demanding application of OVNI, it devotes the simulator to one task, and requires custom platform configurations, on a network of off-the-shelf Pentium Pro-based workstations.

2. Scope of this thesis

Being OVNI a long term project, its implementation has to be broken down into several tasks and requires the work of several people. This report describes the work done, and results achieved, performing the following tasks:

1. Determine a convenient hardware and software platform for the NI and OUI. This involves analysis of hardware/software matching techniques, evaluation of platforms and selection of software tools.

2. Specify the goals and features of OUI. This involves: creating a specification that could be used as the basis for a consistent long term development of OUI, and testing of possible development approaches.

---

4 D/A Digital to Analog Converter, A/D Analog to Digital Converter.
3. Implement a first phase of OUI. Based on the specifications, the development of OUI was divided in two phases. Chapter III, Section 6, explains the scope of each phase.

3. On terminology

Throughout this report we made the effort of reducing the use of computer-related terms and acronyms. For each of such words we did use and thought needed some explanation, we provide a definition in Appendix II, where a Glossary of terms can be found. Words written in chapters I to VI, and Appendix I, using *Italic typeface* are contained in the Glossary. Concepts that were created for OVNI are defined in the text, and are not included in the Glossary. Some other terms are explained by a footnote.

The use of the mouse proves to be a matter of constant reference in the text. To shorten the explanation we use expressions as 'Right-Click' or 'Left-Click' the mouse to express 'Pressing the right or left button of the mouse'. 'Click the mouse' stands for 'press a key of the mouse', and select a control, or click a control, stands for 'Place the mouse cursor on top of that control and press the left button'.

Reference to keys on the computer keyboard is done enclosing the key letter with the '<>' characters. E.g., <Esc> represents the Esc key. To indicate a selection sequence for Menu items the format used is: Menu Name -> Menu Item | Option. E.g. select Main Menu -> Edit | Exit, means: on the Main Menu, select the Item labeled 'Edit', a Pop Down Menu will appear, from the Pop Down Menu select option labeled as 'Exit'.


II. Hardware and Software Platforms

The way OVNI has been conceptualized imposes a number of constrains on the choice of software and hardware tools for its implementation. OVNI is a distributed application, with real-time performance but attractive interface features. It also has to be modular and with a flexible architecture, to support different configurations for different types of applications. Figure 1 illustrates how OVNI adopts different configurations according to the application.

![Figure 1. Some OVNI applications and configurations.](image)

Among all possible applications, the most demanding requirements are imposed by the goal of achieving real-time simulation. Therefore, our analysis of hardware and software requirements had to begin from the top demanding application downwards to guarantee a consistent design. This task was the combined effort of three people, Dr. J.R. Martí, as project coordinator, L. R. Linares, as developer of the NI, and myself as developer of OUI.

1. Hardware

The keystone of OVNI's implementation is the approach chosen to achieve real-time performance. As explained in Chapter I, this approach is based on efficient solution algorithms and hardware/software matching strategies. Therefore the choice of hardware is related to those strategies and partly involves issues that are in the scope of the NI, and will not be discussed here. However one issue of relevance for both the NI and the Interface is the choice of a common computer hardware platform.
Choosing a standard computer platform for both, NI and OUI was a long task. Many factors contributed to make the choice particularly difficult:

1. The nature of both applications is radically different. For the Integrator, the main considerations were processing speed, hardware compatibility\(^5\), and programming language support. While for the Interface, the main concerns were convenient development tools, and graphical support. At the same time a way had to be found to communicate both applications.

2. The debate of RISC vs. CISC\(^6\) microprocessor architectures, was at a peak. New machines were constantly appearing in the market, to join in a benchmark race that continuously offered faster processing speeds. From our point of view the focus was on whether to choose a PowerPC, Alpha or Intel x86 based computer. For each machine, the right combination of peripheral devices, operating system, and development tools had to be found. This was particularly difficult because the rate at which new machines where appearing exceeded the rate at which new tools were offered.

3. A shift in bus standards was occurring. Different computers would support either VMEbus or PCI expansion slots\(^7\), with their correspondent drivers and development tools for some operating systems.

4. Our decisions about which programming language and operating system to use for the project were evolving as we gained experience.

5. Being OVNI a long term project, our decision had to provide for long term support.

Finally we chose Intel's Pentium-Pro 200MHz workstations, for which we could find a suitable combination of hardware peripherals, expansion slots and software tools. Further work done by other members of our research group [3], were based on these machines, and

\(^5\) Compatibility issues that involved transmission speeds on expansion slots, peripheral devices and communication data links, arose from hardware/software matching considerations.

\(^6\) RISC stands for Reduced Instruction Set Computer. CISC stands for Complex Instruction Set Computer.

\(^7\) VMEbus stands for VersoModule Eurocard bus. PCI stands for Peripheral Component Interconnect. Both are bus standards.
proved possible, for example, to model a 24-valve HVDC converter with a 50-100μs solution step.

2. Software

The choice of software tools was linked to the choice of a hardware platform, and as we considered different computer systems we had to shift our attention to the different software tools that were available. However a key point was the decision of adopting ADA as the language in which new models would be developed and added to the NI. This decision restricted the possibilities and provided a guiding focus. Finally a Real Time Operating System and an ADA compiler, for the platform chosen were selected. Details on this selection are in the scope of the NI, and are not discussed here.

For the Interface, the decision was made to develop it on Windows95 while we waited for Windows NT 4.0 to be launched. The reason for that is that Windows NT 4.0 interface is functionally identical to the Windows 95 shell. On the other hand, for desktop simulations the use of Windows 95 was convenient.

The next step was to select a development tool that could provide for:

1. Development in Microsoft’s Windows 95 with portability to Windows NT.
2. Extensive use of windowing objects such as windows, dialog boxes, icons, menus etc.
3. Robust and efficient development environment.
4. Integration of Data Base handling capabilities.
5. Support for Client-Server architectures

There is no shortage of tools for Windows development, and the market offers among others Visual Basic, Visual C++, Power Builder, Borland’s C++ and Pascal compilers, Borland’s Delphi, Symantec’s C++, SQL Windows, Clarion for Windows, Smalltalk. After considering several tools and testing some of them our decision was finally set for Borland’s Delphi 2.0. The following list points out some of the key attributes of Delphi:

---

* High Voltage Direct Current.
1. It is as Rapid Application Development Package (RAD) for Microsoft Windows 3.1 (Delphi 1.0 - 16 bit), Windows 95 and Windows NT (Delphi 2.0 - 32 bit).

2. It is a visual programming environment with a look very similar to Visual Basic’s.

3. It is a fully Object Oriented tool based on Borland’s Object Pascal language.


5. Includes a Database Engine for access to dBase, Paradox, local InterBase and other databases with support for OBDC (Open Database Connectivity).

6. Provides support for Client-Server development.

As it is discussed latter in this report, Delphi was used to develop the first phase of the Interface, and it proved to be a very useful and powerful tool.

To simplify the development of the Interface, we also looked for a library of Delphi components that would provide charting objects. We considered several of them and decided to acquire a library called TeeChart. Although not used for the first phase of the Interface, some preliminary, and successful, experiments were made with it in our group. The choice proved to be quite adequate and convenient since Borland decided to incorporate TeeChart in their new version of Delphi 9, that has recently been released to the market.

---

9 Delphi 3.0
III. Specification of OUI, OVNI’s User Interface

This section presents the specifications made for the development of OUI. These are long term specifications to be used by different programmers at different development stages. There may be several design issues that are not covered here or features that haven’t been foreseen at this time, however these specifications should provide a coherence and a solid base to organize the project in a long term basis.

At the specification stage there were no sketches nor graphical images to show how the program should look like, and the appearance of the Interface was left to the creativity of the programmer. Chapter IV (Implementation of OUI, first phase) provides snapshots of the program as it was finally implemented. To help the user visualize the descriptions in this chapter, references to the images of Chapter IV are provided.

Notice that the program described latter in this report represents the first development stage of OUI and does not implement all the requirements described by these specifications. On the other hand, those features that were implemented will be dealt with again in Chapter IV, and the reader might find some sections of the text redundant, however, this was considered necessary for completeness of this report. The scope of the implemented program is explained at the end of this chapter.

1. Definition of OUI

OUI (pronounced ‘wee’), stands for OVNI’s User Interface. It is a Graphical User Interface program that combined with the Network Integrator program (NI) implements the heart of OVNI.

Despite the fact that OVNI will cover a wide range of simulation types, and will provide different modes of operation (off-line, on-line, synchronized, real-time), the interaction with the user is to be done through a single unified environment. From the user’s point of view, the same program will be used to create a case, run a simulation interactively, plot curves, and save results. The same is true for real-time simulations where OVNI may be spread through an array of computers. OUI is the implementation of this integrated control environment.
2. Development, Operating and Maintenance Environments

The Development and maintenance environment is an Intel Pentium-Pro 200MHz Workstation, running Windows95. And the development tool is Borland’s Delphi 2.0, which is a fully object-oriented tool based on Borland’s Object Pascal language. These choices were explained in Chapter II.

The program is to run either on Windows95 or Windows-NT. Delphi 2.0 supports both operating systems, but if the programmer is to make direct use of system API \(^{10}\) calls he/she must make sure that the call is supported by both operating systems. For desktop simulations the program should support other PC platforms, but the developer should not concern himself with performance on those systems, and should focus on the use of at least Pentium-Pro based machines.

The output device is a SVGA monitor, and input devices are a keyboard and a two-button mouse. The program is to be updated constantly as new component models are added to OVNI. This is to be done in the same platform used for development, and for each update a new version is to be generated.

3. External Interfaces and Data Flow

The communication with the user is done through the keyboard and the mouse. Some applications of OVNI require OUI to receive data from other sources. For instance, in power system planning a database of existent equipment might be used. To do so, OUI will rely on Delphi’s built-in database management capabilities. Database management will be used also for online power system security and control applications, where the interface will be periodically fed with actual data from a State Estimator.

For real-time simulations, the information from A/D converters will be passed to OUI by the NI. The communication protocol between OUI and the NI is based on a Client-Server scheme. Both programs are developed with different compilers, different languages, and at

\(^{10}\) Application Program Interface, is the interface by which an application program accesses operating system and other resources. A few differences still exist between Win32 API and Windows NT specially in graphics functions.
times could run on different platforms. Therefore a communication protocol is necessary to connect both programs. From the point of view of OUI, Delphi’s *Dynamic Data Exchange* and *Client-Server* objects are enough to implement this link. The problem is more complex from the side of the NI, but after a preliminary analysis of the functions and message formats used by Delphi the problem is though to be solvable.

OUI must also save information in files. For this purpose, at least the following file formats are to be supported: OVNI’s proprietary formats, *MicroTran*’s data case and output files, standard EMTP, and a *free-form exchange format*.

4. Functional requirements

The operation of the interface is presented here from a functional perspective, i.e., which are the particular features provided, how do they interact with the user, and how are they displayed. The following description assumes that the program runs under Windows95 or Windows NT, and a mouse with two buttons is available.

4.1 Scope of the interface

There are two main activities in the operation of the interface. First, to help the user create a description of the electrical network to be simulated (Description stage). Second to launch, interact and control the simulation (Simulation stage). OVNI’s GUI provides an integrated environment for both stages, i.e., the same program provides control in every stage. This is also true for all types of simulation: off-line, on-line, synchronized on-line, and real-time.

4.2 Description stage

During the Description Stage the user builds a schematic representation of a network using a drag and drop approach. Several electrical-Component symbols can be selected with the mouse from a library and can be placed on a drawing page (or Schematic Area). Once the Component is on the page, it can be moved, rotated, renamed, deleted, copied or connected to another Component. Double-clicking on the Component launches a window (DataForm) with fields to input parameter information and simulation choices.
The Schematic Page area can be set to one of four different sizes with or without a guiding grid of dots. Once the schematic has been built, the user can save it into different formats, e.g., OVNI, Microtran, standard EMT, free format. Also any circuit can be saved as a 'sub-circuit' and be imported into another schematic as a black box.

4.2.1 Appearance of the interface during the Description stage

The following are feature guidelines:

- When the user runs the program a full screen-size window opens. The window can however be resized by the user, either by dragging the borders or by pressing the resize icon in the title bar. The program is smart enough to save these settings for the next time it is executed.

- The title bar of the window shows the name and version of the program, plus the Minimize, Maximize, and Close buttons. It is important to boldly show the version number as different versions would have more or less Components in the library.

- A Menu on top of the window provides the following options (some of these functions can also be accessed from Speed Buttons):

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New</td>
<td></td>
<td>Creates a new blank schematic</td>
</tr>
<tr>
<td>Open</td>
<td></td>
<td>Loads a schematic file</td>
</tr>
<tr>
<td>Reopen</td>
<td></td>
<td>Shows a list with names of last files used</td>
</tr>
<tr>
<td>Save</td>
<td></td>
<td>Saves the current schematic</td>
</tr>
<tr>
<td>Save Results</td>
<td></td>
<td>Saves simulation results</td>
</tr>
<tr>
<td>Save As</td>
<td></td>
<td>Saves current schematic under a new name</td>
</tr>
<tr>
<td>Print Preview</td>
<td></td>
<td>Shows a screen preview of the schematic as it will look when printed.</td>
</tr>
<tr>
<td>Print</td>
<td></td>
<td>Prints either the schematic, element parameters, or both</td>
</tr>
<tr>
<td>Exit</td>
<td></td>
<td>Exits the program</td>
</tr>
</tbody>
</table>

11 For the implemented appearance see Fig. 4, Chapter IV.
<table>
<thead>
<tr>
<th>Edit</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>Undo the last action</td>
</tr>
<tr>
<td>Redo</td>
<td>Redoes the last action</td>
</tr>
<tr>
<td>Cut</td>
<td>Cuts selected Components from schematic</td>
</tr>
<tr>
<td>Copy</td>
<td>Copies selected items</td>
</tr>
<tr>
<td>Copy to clipboard</td>
<td>Copies to clipboard the image of selected area in schematic</td>
</tr>
<tr>
<td>Paste</td>
<td>Pastes Component previously copied or cut</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes selected Components from schematic</td>
</tr>
<tr>
<td>Select All</td>
<td>Selects all Components from schematic</td>
</tr>
<tr>
<td>Drawing Area</td>
<td>Adds additional space to the schematic page in multiples of one page</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>View</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid ON/OFF</td>
<td>Shows or hides grid on schematic area</td>
</tr>
<tr>
<td>Show all</td>
<td>Shows a reduced full-view of the whole schematic</td>
</tr>
<tr>
<td>Schematic</td>
<td>Shows a dialog box with an alphabetical list of all Components</td>
</tr>
<tr>
<td>Component List</td>
<td></td>
</tr>
<tr>
<td>Other cases</td>
<td>Opens a window to other schematics where all edit functions are supported but simulation is not possible</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Schematic</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Starts drawing of a connection</td>
</tr>
<tr>
<td>Add probe</td>
<td>Adds a probe element, to display simulation output</td>
</tr>
<tr>
<td>Add event</td>
<td>Programs occurrence of an event in the simulation</td>
</tr>
<tr>
<td>Make Sub circuit</td>
<td>Creates a sub-circuit with current a selected portion of the schematic</td>
</tr>
<tr>
<td>Show Sub circuit</td>
<td>Displays the schematic inside a sub-circuit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Starts simulation process, asks for time step, and optionally tmax</td>
</tr>
<tr>
<td>Stop</td>
<td>Stops simulation</td>
</tr>
<tr>
<td>Pause</td>
<td>Pauses simulation</td>
</tr>
<tr>
<td>Rewind</td>
<td>Allows rewinding to a particular time step</td>
</tr>
<tr>
<td>Fast forward</td>
<td>Allows fast forward after rewinding</td>
</tr>
<tr>
<td>Add Breakpoint</td>
<td>Sets a logical state condition to stop the simulation</td>
</tr>
<tr>
<td>Sanity Check</td>
<td>Performs a Validity Check of the schematic</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Help</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help Topics</td>
<td>Opens a Tabbed Help window with options:</td>
</tr>
<tr>
<td></td>
<td>Contents, Index, Find</td>
</tr>
<tr>
<td>About</td>
<td>Shows a window with name, program version and copyright notice.</td>
</tr>
</tbody>
</table>

Table 1. OUI's Specification: Menu items and their options.
The menu structure does not change or merge with other menus at any moment. The options not available are dimmed and the string of the Undo and Redo commands adapt to the action to be performed e.g. ‘Undo Delete Component’.

The Menu bar is followed by a top panel that contains speed buttons, page tabs and a palette of Components with tabbed pages.

The speed buttons are grouped in three sections, one for File Menu related actions (New, Open, Close, ...), the second for some schematic drawing actions (Cut, Paste, Delete, Add probe, ...), and the third for Simulation Menu related actions (Run, Pause, Stop, Rewind, ...).

The Component Palette is used to access and select different electrical Components. Each tabbed page displays a palette of speed buttons to select different Components.

The tabs and the Components shown include (on a first implementation):

<table>
<thead>
<tr>
<th>Basic Elements</th>
<th>Resistor, Capacitor, Inductor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compound Elements</td>
<td>Transformer, Pi-Circuit, Transmission Line, HVDC Converter</td>
</tr>
<tr>
<td>Control Blocks</td>
<td>none</td>
</tr>
<tr>
<td>Electronic Devices</td>
<td>Diodes, Thyristors, Triacs, GTO's</td>
</tr>
<tr>
<td>Sources</td>
<td>Voltage Source, Current Source, Dependent Voltage Source, Dependent Current Source.</td>
</tr>
<tr>
<td>Switches</td>
<td>Time-controlled switch.</td>
</tr>
<tr>
<td>Meters</td>
<td>Voltmeter, Ammeter, Fluxmeter, Wattmeter</td>
</tr>
</tbody>
</table>

Table 2. Specification of the Component Palette: minimum set of Components.

More Components will be added to this library in future versions.

At the bottom of the window there are two status panels. One to show coordinates on the schematic page, the second to display messages to the user. The messages are constantly appearing providing hints related to the current situation.

The rest of the window (between the top panel and the status panel) is the Schematic Area. This is were the schematic is to be drawn. The schematic page is scrollable and
has a white background and a (visible or invisible) position grid associated. By default the size of a new schematic page is one screen (this is the area visible for drawing); when printing, this fits the dimensions of a letter-sized page in landscape position. The user can expand the drawing area by adding additional pages using the Menu option: Edit -> Drawing Area. There are only four options available: one page, two pages in an array of 1x2 (one row X two columns), two pages in an array of 2x1, and four pages in an array of 2x2\textsuperscript{12}.

4.3 Building a schematic

4.3.1 To open a schematic page

The user can either create a new schematic or modify a schematic that was previously saved. The steps to create a new schematic are: select File -> New. This opens a blank unnamed schematic page. To open an existing schematic file: select File -> Open. The program is aimed to support the following circuit specification formats: OVNI, Microtran, standard EMTP, and free-form exchange format.

Every schematic page has an invisible grid associated that casts the position of the elements in the schematic. The grid is invisible by default but can be made visible, or invisible again, by selecting View -> Grid ON/OFF. However the grid cannot be disabled, nor resized.

When building a schematic it may be useful to have access to other schematics. To do so, user can select: View -> Other Cases. This opens a window to the schematic with all Edit options available, but no support for simulation. Using Copy and Paste it is possible to import Components from one case to the other.

4.3.2 To add electrical components to the schematic

There are two ways to find and select a particular Component. The first way is to look for the graphical symbol that represents the Component in the Component Palette tabs \textsuperscript{13}. To select the Component the user must click on the symbol with the mouse; then drag the Component and release it on the Schematic Page by clicking the left mouse button.

\textsuperscript{12} For a graphic description see Fig. 2, Chapter IV. For the implemented appearance see Fig. 7, Chapter IV.
The second way is to look for the Component in an alphabetical list of all available Components\textsuperscript{14}. To do so the user must select from the Main Menu the option \textit{View} -> Component List. This shows a Component list dialog box with an empty edit field and a list box with all Components names and their symbols. Once the user clicks a ‘Select’ button (in the dialog box) the Component is attached to the mouse and can be dropped on the Schematic Page.

In both ways of selecting a Component, when the Component is attached to the mouse the \texttt{<Esc>} key can be used to cancel the selection and release the cursor; and the object can be rotated in increments of 90° by clicking on the right mouse button.

\textbf{4.3.3 Placing and moving Components on the schematic page}

The Components are originally placed at the location where the user dropped them on the schematic. On the schematic page the Component is represented by a graphical object that shows its electrical symbol and electrical terminals\textsuperscript{15}.

Any Component can be selected by pressing the left mouse button when the mouse cursor is on top of it. Several Components can be selected simultaneously with the mouse by pressing \texttt{<Ctrl>} first and then left clicking on one of them, or by drawing a selection box around them. Also all Components can be selected simultaneously using the Menu option: Edit -> Select All.

Any Component can be moved around the schematic area by dragging its graphic symbol. If the Component is dragged near the visible borders of the page, the scroll bars are adjusted automatically to span the page. Copies of Components can be done using menu options ( Edit -> Copy, Edit -> Cut), in such case the copy made is another element with the same electrical parameters but with a different name. The Menu option Edit -> Copy to \textit{Clipboard}, makes a copy of only the graphical image of the Component as the user sees it on the schematic page; no parameters or model information is copied. This is useful for importing schematic drawings to other applications.

\textsuperscript{13} For the implemented appearance see Fig. 9, Chapter IV.

\textsuperscript{14} For the implemented appearance see Fig. 12, Chapter IV.
To delete a Component the user can either select the object and press <Del>, or choose the menu item Edit -> Delete.

4.3.4 Setting of electrical parameters

Double-clicking with the left mouse button on any Component opens a Component’s Data Form. This is a tabbed Dialog Box where the user selects simulation models to be used and inputs parameter information. The fields and options of these dialog boxes are different for each Component.

The dialog box can show different tabbed pages corresponding to different models for the Component. The last selected tab is the one in effect for the simulation, although the information in it doesn’t override information set in any other tab.

The basic parameter values of each Component are shown on the graphical symbol, e.g., R=10K. Any other relevant information appears on the status panel whenever the cursor is on top of the object, e.g., ‘Frequency Dependent Model’, for transmission lines.

Each Parameter Dialog Box performs a preliminary sanity check of the data the user introduces. If there are any errors the wrong data is not saved and an error message is shown.

4.3.5 Connecting Components

To connect Components the user must enter a drawing mode, then select the terminals of the Components by pressing the left mouse button with the cursor placed on top of a terminal’s image. Using the cursor keys (or using the mouse) the user can draw a line from the terminal. The line extends or retracts only in vertical and horizontal directions following the page grid.

When the line reaches another terminal a connection is established if the mouse button is released, otherwise there is no physical connection although the line crosses the terminal.

A connection may also be established when a Component is dropped such that any of its terminals overlaps with another terminal on the schematic.

15 For the implemented appearance of some Components see Fig. 13, Chapter IV.
In any case a Component can be pulled away from the connections just by dragging the Component on the page. It is also possible to move a Component around the page and keep its connections. In such case the connection lines extend elastically. To do so, the user must press the <Shift> key while dragging the selected Component(s).

If the user wishes to reshape any lines after moving, it will be necessary to delete and redraw them. All the lines (as well as the Components) are drawn in XOR mode to show overlapping.

4.3.6 Disconnecting Components

Any connection line can be deleted. To do so, the user must select the line with the mouse and press Del, or chose Delete from the Edit menu. If two Components are connected by overlapping terminals any of them could be dragged away with the mouse.

4.3.7 Setting node names

Node names are a feature associated with connections. To set a name, double-clicking on a connection line opens an edit box. Once the label is displayed it can be dragged on the screen. If the connection is made with overlapping terminals (no visible line), there is in fact a connection line associated but of zero length, thus double-clicking the terminals will open the edit box.

All nodes are initially assigned default names. These names are overridden by any name set latter.

4.3.8 Setting and removing probes

Probes are Components that specify the simulation data to display. They are placed and removed the same way other Components are. A probe can be (on a minimum implementation) a voltmeter, ammeter, fluxmeter, or wattmeter. When the user places a probe on the schematic a small graphic object appears that must be attached to a terminal or node. Double-clicking the object opens a form where different output formats and variables that can be selected. During the simulation stage double-clicking a probe launches a window where the results for the selected variables are plotted.
4.3.9 Setting and removing break points

The support for using breakpoints in the simulation comes from the NI (OVNI's solver). The details on how this will be done are not defined yet, but it is clear that the Interface should provide for an easy way to set logical breakpoints when the simulation is paused. The sequence could be: 1) the user sets a logical breakpoint condition with the simulation stopped, 2) OUI sends schematic and breakpoint information to the NI, 3) the simulations starts and the NI sends results to OUI, until the breakpoint condition occurs, 4) then OUI receives a message from the NI and prompts a message to the user.

4.4 Handling schematics

4.4.1 Saving the schematic

To save the schematic the user must select File -> Save or press a Save Speed Button on the top panel. The schematic can be saved in different formats, e.g. OVNI, Microtran, standard EMTP, and free-form exchange format. For the OVNI format the schematic is saved in a file with the extension *.ovs. This file contains not only the graphical representation of the schematic but also all the parameters and options introduced in the DataForms.

4.4.2 Viewing and using Components from other schematics

Some users might want to copy Components from one schematic to another. To make this operation easier the menu command View -> Other Cases opens a window to another schematic page. This window only allows for schematic drawing actions but does not support simulation. The user can copy and paste Components from one schematic page to another. Every copy, paste and delete operation involves not only the image of the elements but the parameters as well. It is also possible to use a whole schematic as a black-box sub-circuit in another schematic. This will allow users to create their own libraries of sub-networks.

4.5 Simulation stage

After drawing or loading a schematic the user can initiate the Simulation Stage. During the Simulation Stage the Interface launches the Network Integrator program to perform the simulation, and communicates using a Client-Server scheme, with the Integrator as the Server.
Once the simulation is running it is possible to interact with it, to add and remove Components 'on the fly', as in a breadboard experiment. All this while displaying output plots that update on-line. Furthermore the simulation can be paused, rewound, fast-forwarded, stopped, and restarted interactively.

4.5.1 Types of simulations and modes of operation

OVNI is aimed at providing a common tool for different types of simulation, namely: steady-state power flow, stability analysis and fast transient studies. All this using the same solution method. It also has the advantage of offering different ways or executing those simulations, that is different ways for the user to control the simulation. These 'ways of execution' are called Simulation Modes. The modes provided are:

- Off-line simulation: The Integrator solves the network from time zero to a specific time, stops, and then shows results for all time steps in the simulation.

- On-line simulation: The Integrator solves the network continuously allowing for live interaction with the circuit, and providing a continuous stream of data that is plotted and scrolled as if viewed with an oscilloscope. The interaction includes not only control over Component parameters, but also the possibility of adding and removing Components.

- Synchronized simulation: This is a refinement of the on-line mode, where OVNI is used to solve a network in parallel with a real system while receiving periodical updates of steady state values from the real system. The aim of this is to feed OVNI with initial conditions coming from a State Estimator program.

- Real-time: This is another refinement of the on-line mode where the performance of the simulation is fast enough to fake an electrical network while connected to actual equipment. The behavior of the equipment can then be observed and studied as if it was connected to the real system. For real-time applications the interaction of the Interface with the circuit is restricted to those situations that do not involve sudden user modifications of network topology.

The Client-Server approach of the Interface and Integrator allows for the possibility of having the Interface and the Integrator on the same or on different machines. For non-real-
time simulations one single machine can be used to host both the Interface and the Integrator. However, for real-time simulations a network of several machines to add extra computational power, may be necessary to comply with time constrains. In every case the interface is the same.

4.5.2 Appearance of the interface during the Simulation Stage

The appearance of the interface during the Simulation Stage is the same as during the Description Stage except for the following details:

- Menu options which are not available during the simulation are dimmed.
- The background color of the schematic changes from white to black.
- Double-clicking on the scope symbols opens a window where data is plotted.

4.5.3 Network sanity check.

This action can be selected from Simulation -> Sanity check; it is also performed whenever the user selects to run the simulation. If there are any errors the user is notified with a dialog box. The nature and details of the sanity check are not defined yet.

4.6 Running and controlling the simulation

To run the simulation the user can select Simulation -> Run or press the Run Speed Button on the Top Panel. A sanity check is performed internally and the simulation program is launched. Many details about the interaction with the simulation are not defined yet, as they depend on the implementation of the NI: however, the basics of that interaction should allow for:

- Pause: To pause the simulation, either by selecting Simulation -> Pause, or pressing a Pause Speed Button on the Top Panel.
- Stop: To stop the simulation, either by selecting Simulation -> Stop, or pressing a Stop Speed Button on the Top Panel.
- Rewind: To rewind plotted results, even while the simulation is running. To do so, the Interface keeps a memory pool of data. As the user rewinds, by pressing a button, the plots
scroll in the opposite direction, but as soon as the user releases the \textit{button} the plots proceed to show the latest results. Rewinding can also be done after stopping or pausing the simulation.

- Fast Forward: After rewinding it is possible to fast forward. This makes sense only if the simulation has been stopped or paused.

- Breakpoints: If a certain breakpoint condition is attained, the NI pauses the simulation and sends a message to the Interface, which prompts a message to the user.

- Adding and removing probes on-line: It actually will cause the NI to pause the simulation, and then re-start at the last point of it issuing new output data.

- Adding and removing elements: Components and Branches can be added or removed on-line. In that case the Interface pauses the simulation, sends to the NI the new configuration, and the NI retakes the simulation from the point it was paused, but with the new configuration to simulate the response to such change.

- Modifying parameters and events: A Component parameter or a breakpoint condition can also be changed on-line. The NI pauses, receives the new data and retakes the simulation from the last point.

- Viewing simulation results: Double-clicking on a probe should open a form with a scrolling plot. For every probe there is a form.

- Controlling the synchronization: For synchronized simulations, the user must be able to modify the rate at which OVNI and an external \textit{State Estimator} program communicate.

5. Performance requirements

The most demanding requirements of OVNI fall on the design of the NI. Particularly, timing constrains in the range of 50-100µs are required to achieve real-time simulation. The Interface, on the other hand, has to deal with speeds that are related to human reaction times. As a general rule, the design of the Interface must provide reasonable reaction times on a Pentium Pro based machine. During the simulation stage, however, some complications arise.
The problem here is how to synchronize scrolling plots with data output from the NI. To simplify the problem, the NI will send the data to a memory pool from which the Interface can read it at different speeds. This still leaves unsolved issues about how to relate the size of the pool to different reading speeds. A final solution has not been chosen yet, but the requirement set is that the scrolling must be smooth, while displaying several signals at the same time.

Accuracy and volume of data depend also on the NI. The Integrator uses signed 32bit variables\(^{16}\), therefore, the Interface must use the same type of variables for the parameters fields in the DataForms. The results plotted are also in the same format. Other than that, since the Interface performs no calculations, there are no accuracy concerns. The volume of Components that can be used in a single schematic depends on the amount of available memory. If OUI and the NI are hosted by different machines, there will be different memory limits for each program. However, it is very unlikely that memory limit problems could arise from the use of these programs, since the objects in OUI consume only a small amount of memory.

6. Scope of first and second development phases.

As indicated, the development of OUI is a long term project which has been divided into two phases. The goals of the first phase are to implement:

- General look and feel of the interface during the description stage, with basic functions to handle a library of electrical components, and with a good base of routines and objects to allow for upgrades.
- A library of components that is easy to upgrade and simplifies the development of new components.
- Basic properties of Components: placement, movement, rotation.
- DataForms in a way to simplify their design for future Components.
- Connections between Components.

\(^{16}\) Signed 32 bit variables cover numbers in the range of: -2\(147483648\) .. 2\(147483647\).
• Printing and print preview options.
• A window to show a reduce view of the whole schematic
• File saving and loading of schematics.

The main goals of the second phase are to implement:

• A basic library of Components with their DataForms
• Online help and create a help file
• Improvements to general features
• Support for different file formats
• Simulation Stage
• Probes, and plotting of results
• Communication with the Network Integrator

Only the first phase is in the scope of this thesis. The results are explained in the next chapter.
IV. Implementation of OUI, first phase

1. Design and implementation of OUI

This section explains the way OUI was technically devised. The discussion here covers design guidelines, program architecture and some practical implementation issues.

Two important concepts handled in this section are ‘look and feel’ and ‘software architecture’. When talking about Graphical User Interfaces (GUI), the term ‘look and feel’ is used to refer to the appearance and operation of the program. Look and feel is a broad concept that extends from the visual appearance of the program to every other feature that defines the way the program interacts with the user.

Generally, the term ‘software architecture’ refers to an abstract description of the program that breaks the problem into functional blocks, and defines the way those blocks behave and communicate with each other. Ideally this is done at a level of abstraction that is beyond the choice of a programming language and development tools. However, GUIs are very dependent on the operating system (OS) they run in and very much restricted by the development tool used to create them. This makes difficult to specify a practical software architecture without making reference to either the OS or development tool chosen. Therefore, in this section we make constant use of terms related to Windows and Delphi programming. These terms are explained in Appendix II, Glossary.¹⁷

1.1 Design guidelines

The design guidelines we followed can be divided into two groups, one related to the look and feel of the program, and the other to the structural aspects of the implementation.

1.1.1 Look and feel

The Look and feel of the interface was implemented following these guidelines:

---

¹⁷ A description of software development tools was presented in Chapter II, Section 2.
1. The program must look attractive: The general rule is to pay attention to aesthetics and simplicity. Use as much as possible standard 3-D Windows95 controls and keep general conventions, e.g., for the names of File Menu and the location of the OK Button. For custom controls, such as Components, Branches and customized controls, keep as close as possible to Windows style and don’t overcharge the images. Avoid fancy fonts.

2. The program must feel attractive: As much as possible avoid excessive response delays. If a task takes time, display an Hour-glass icon. Allow for use of shortcuts, and the <Tab> key but do not add fast-keys. Take advantage of the mouse, e.g., to implement drag and drop for Components, etc. But to draw connections, which is an action that requires accuracy, use the arrow keys. Some users might complain about this but it will avoid them errors.

3. The program must be simple to understand: Avoid clustering of controls, specially of Speed Buttons, Check Boxes and Radio Buttons. Keep all Menu options always visible, even when unavailable, and limit the number of nested sub-menus to two. Use hints, and on-line help, and when performing a sequence of steps, e.g., to connect two Components, do not expect the user to guess or explore for the next move. To help find Components add a Component Palette with Components classified by categories in tabbed pages. To simplify use of Menu options add Pop Up Menus with most common commands.

4. The program must capture the user’s attention: OVNI is a serious application that requires attention and by no means an interactive multimedia experience. As default, launch the Main Form maximized to cover distracting windows, icons and images on the background, but also allow for resizing if the user wishes to. Avoid distractions caused by excessive use of Message Dialog Boxes, changes in the image or sounds, unless you wish to draw the attention to a particular point in the window. Also keep clear focused controls or any other area where the eyes of the user would be.

5. The program must be predictable: Keep as close as possible to standard Windows conventions. And choose intuitive icons and cursors. Represent electrical components with standard Electrical Symbols. Avoid situations where the user must guess the next action to take.
6. The program must predict actions: When possible predict where the next point of focus will be. In panels with many input fields (as in DataForms) make the <Enter> key switch the focus to the next Control (as the <Tab> key does). In Open and Save Dialog Boxes, provide always as default path the last path the user chose. In List Boxes with too many objects implement incremental-alphabetic search. In Menus place more frequent actions at the top and add Pop Up Menus with most common commands.

7. The appearance must be consistent on different types of screens and video resolutions. And the application must be capable of handling different types of printers.

8. The program must be not too customizable: Design the program using Windows’ default appearance scheme (particularly the MS Sans Serif font), but allow for custom changes made in Windows’ Control Panel to apply. Other than that, limit custom options to the minimum (if possible don’t even provide an Options Menu), and save into file any selected option.

9. Provide constant help: Design for on-line help. Although not in the scope of this first stage of the project, add help buttons where necessary too simplify the work latter. Use hints to identify buttons, Components and Branches, and add a Status Bar to show options or directions whenever a specific action is required.

10. Be forgiven with the user: Always provide an escape and cancel paths to commands. Avoid error messages that could make the user feel bad, particularly because most users will redirect those feelings to the program. Use error messages for errors in the execution of the program but use warning messages for errors made by the user; and when possible, replace warning messages by information messages (a type of Dialog Box message in Delphi) which would feel more didactic than penalizing.

1.1.2 Structural considerations

There were three initial decisions that determined how the program was to be structured and that identified critical aspects in the coding stage.
A. Production of a single executable file

Delphi's Compiler produces a single executable file for distribution. This file is a 100% stand alone application\(^\text{18}\) that combines a basic footprint of inherited behavior for Controls, the application's code, and all the resource information (e.g., appearance of controls, bitmap images, etc.). However, in Windows programming there is the possibility of developing an application to make use of shared DLL's. If so, DLL files have to be distributed with the application. The advantages and disadvantages of using DLLs were weighted and the decision was made not to use them and produce a single executable file. The discussion that follows explains that decision.

DLL stands for Dynamic Link Library. A Dynamic Link Library, is a module of code, data or resources that is linked to the programs of an application when they are loaded rather than at compilation time. The main purpose of DLLs is the sharing of code. In fact all Windows programs rely on the use of DLLs: USER, GDI and KRNL (for user interface, graphics and program management).\(^\text{19}\) DLLs can contain functions for the use of several applications or can be designed for one particular application. They can contain routines, code, resources or a mixture of them, e.g. a Dialog Box.

The advantage of writing custom DLLs is that a program can be made modular. This simplifies updating the application since the code can be organized such that the updates are done by replacing the DLLs rather than the entire application.

OVNI is intended for successive updating as new Components are added to it. Therefore the use of DLLs seems attractive to implement a library of components. The interface would then consist of a main application program that could load different types of libraries depending on the version of the DLL installed. On the other hand the disadvantages of using DLLs in this application are:

---

\(^{18}\) Except for database applications, in that case the distribution must include the application's executable file plus Borland Database Engine.

\(^{19}\) Dynamic Link Libraries are also used in other operating systems, e.g. SunOS.
1. DLLs are not required in OVNI to share code with any other part of it, or any other application. The code for OUI could be fully implemented in Object Pascal (as it was) without the need for DLL libraries, and the communication between the Interface and the Integrator involves the passing of data and not the sharing of code.

2. Regarding the use of a DLL as a stand alone library, our experience with other applications has lead us to conclude that keeping track of different versions of DLLs is prone to cause problems. Since the code of the interface will be upgraded as new features are added in, a mixture of new versions of the main program and versions of the Component Library can cause confusion.

3. The amount of information required in the Component Library is small compared with the size of the program. However, although Delphi executable files are reputed for their big size, it is possible to design OUI to be quite small if the coding is done carefully (less than 700KBytes at this stage). Therefore, updating the entire application is not cumbersome.

4. The difference between two Component Libraries relates to which one is more up to date; and is not a matter of each library offering absolutely different sets of Components. Thus modularity of the library is not really essential.

5. One of the areas of application of OVNI is as a stand alone simulation environment for training and educational purposes. For those cases, the program is to be distributed with a minimum amount of files, in order to offer the simplest possible installation.

6. A mistake writing a DLL, as for example, calling a function with the wrong parameters, can not be detected by Delphi’s compiler, and would make the application crash.

7. Several technical difficulties, among them: In DLLs the Pascal-type-of string is not supported. Therefore string information must be handled with a type of string that is not native to Delphi’s compiler; Over-range access of variables can occur with DLLs; Only very simple data types can be used as function parameters.

Considering these disadvantages, DLLs were not used, and the generation of one single executable was taken as a rule. At some points in the development of the program several problems were encountered that could have been solved easier using DLLs (as, for example,
how to intercept all mouse messages). However the effort was concentrated on avoiding those solutions, and other successful approaches were found.

B. Optimizing the use of System Resources

Optimizing the use of resources, has several implications as the term resources is broadly applied to different items. A resource can be the amount of memory still available. It can also be a read-only module of data stored in an executable code file that is loaded into memory whenever is needed by the program. It can be an object used by the graphics system (GDI), or also available system settings, like windows handles, etc.

There are several reasons to be concerned about the use of resources.

1. As OUI provides four schematic sizes and carries an increasing amount of Component types, each one with a respective DataForm, it is an important design problem how to make the performance of the program independent of the number of Components. It cannot be acceptable that to draw a schematic with a few Components, other available Components would cause an overhead on the time the program takes to load and execute (i.e., as the library of Components increases, the application’s performance downgrades). Neither should the number of DataForms in use become a burden for the system, nor the files the program generates be excessively large, nor the requests to the OS for new handlers and graphical resources be too frequent.

2. For some OVNI applications the Interface and the Integrator are to be hosted by the same computer. Then both programs would be competing for processing time, OS resources and memory. Performance constrains are more demanding for the integrator than for the Interface, and therefore OUI has to be designed to give as much free space as possible to the Integrator.

3. Also, in non-real-time desktop simulations several other applications, e.g., a word processor, could be running simultaneously, and OUI should not be felt to be affecting the execution smoothness of other programs.

These considerations were taken into account when devising the software architecture.
C. Designing for upgradability and code maintenance.

Guidelines were chosen here to simplify the task of adding new Components to the Interface, to provide for a solid platform for further development, and to help future programmers maintain the code.

As explained in Chapter I the goal in this phase of the development of OUI is to implement all basic program tasks (except for writing a help file and communicating with the Integrator), such that in another phase the program can be enhanced. Therefore, one of the basic rules was: to keep in mind the problems and needs of future programmers.

A considerable amount of effort was spent to achieve this goal. The architecture of the program was devised to simplify the task of adding new Components to the Interface. A guide for adding new Components was written (See Appendix I) and a utility, COMPEDIT, was made to help in that purpose. Many features were implemented in part to provide a base for the next development phase of OUI. Methods of general use were written, custom controls were developed for future usage. The program contains guiding comments, debugging variables, routines, controls and error messages to help debugging; it also has a developer mode in which more Components and menu options used for debugging become visible.
1.2 Software Architecture

In organizing the structure of the program there are four different issues to consider:

1. Basic Functionality: How to implement the general tasks of the interface, e.g., how to implement Components, nodes and DataForms, according to the specifications and the design guidelines.

2. Component Library: How to create a library of components that can be easily upgraded, does not affect performance, and is allo*linked to the executable file.

3. Data Handling: How to communicate with the NI, and in which format to save schematic files.

4. Code organization: Number of units, modules etc.

The answers to these issues are briefly discussed here.

1.2.1 Basic Functionality

The use of Object Oriented Design greatly simplified the implementation of OUI. In fact the operation of the program relies on a collection of objects and their interactions.\(^{20}\)

The question of how to implement the functionality became a question of which objects to use/create, and how. It is however unnecessary, and unproductive, to organize a project of OUI's complexity in an object-per-object base. Rather than that, the approach followed was to conceptualize a few elements in charge of the main tasks and then make sure that they could be implemented with the tools, and objects available. Being OUI a Graphical User Interface all of these elements are visible, and some became basic definitions on which the structure of the interface is based.\(^{21}\)

The basic definitions introduced in OUI are:

---

\(^{20}\) Most of the objects used came from Delphi's Component Library, others are custom objects created for this project.

\(^{21}\) There are in fact, at this stage, in OUI, some one hundred and eighty (180) objects from which eight (8) are custom objects, and a few more are created at run time. COMPEDIT uses 25 objects.
A. Main Form or Main Window

When the program is loaded, a single main window is opened, this window is also called the Main Form. Although at some times other windows are shown, the Main Window is the heart of the application from where all commands are available and on which the schematic is drawn. The Main Form contains an area for menus, speed buttons, a Component Palette (defined below), a status bar and a scrollable drawing area. The Main Window can be built using Delphi objects with no major complications.

B. Schematic Area

The schematic area is a blank area where the user can draw a schematic. The size of it can be changed, and the area is scrollable, both by the user or automatically. On the schematic area the user places Components, moves Components and draws connections. The schematic can be saved into a file, and read from it.

The schematic area is in fact an Image object placed on top of a scrollable object. The implementation, however, involved the creation of a new scroll box object, and presented severe flickering problems that were solved combining the use of Delphi objects with special properties in custom objects for Components and Branches. Eliminating blinking was an arduous problem, for which several different approaches were tried.

C. Schematic sizes

There are four possible sizes for the schematic area. Different sizes are made up of different numbers of pages; each page is placed horizontally and its dimensions are proportional to a Letter size page (8 1/2 x 11in). Figure. 2 shows the distribution of pages for each of the four possible schematic sizes. Table 2 shows the distribution of pages, and their position vs. schematic size.
Figure 2. Distribution of pages for each schematic size.

Internally, the program identifies each size and page position by a number, as shown in Fig 2. For the user these names are transparent. The unit page, is sometimes called **Unit Size Page**. The most important implication of changing sizes is that for bigger dimensions the program uses more resources and becomes slower when scrolling. These problems were reduced as much as possible using Delphi objects.

D. Components

Components are the electrical elements the user uses to build a schematic, and the term is used in the same sense as for electrical components. For every Component the NI must provide one or several simulation models, and the Interface a symbolic representation. The term Component is applied in OUI to refer to the symbolic representation.
A Component in OUI is a *graphical object* that can be selected from a library and placed on the Schematic Page. It can be *dragged* over the page and rotated. Every Component has terminals that are sensitive to *mouse events* and an associated DataForm to input simulation data. Implementing Components required the creation of a new object that was called TElement. Among other properties, TElement is capable of reading Component information from a Windows *Resource File*, it provides mouse *event handlers* for *hotspots* used to implement the Component terminals and creates its own DataForm.

**E. Component Palette and Component List**

From these elements, the user can select a Component to place on the schematic area. The Component Palette, as defined in the specifications, is a tabbed *palette*. The Component List is a *floating form* with a *list box* and *incremental alphabetical search*.

Both, the Component Palette and the Component List, had to be devised to handle an increasing amount of Components. The code was designed to simplify this task. The images used for a same Component on both elements are shared to reduce the use of *graphical resources*. An as a new Component is added to the Component Palette the Component List automatically loads the new image and name. This is all implemented using Delphi components.

**F. DataForms**

The DataForm is a *floating window* where the user types-in parameter data and simulation choices. If there is more than one simulation model for that Component, the DataForm has *tabbed pages* with each page containing a different set of input fields. For each type of Component there is a type of DataForm.

As the number of Components increases, so will increase the number of DataForms. DataForms drain more system resources than Components. To reduce this problem, DataForms were implemented such that they are *instantiated* only when needed, and when closed they are destroyed. A DataForm is created by a Component and communicates with it through the OS *messaging system*. Another implementation problem was to design the DataForms such that it would be easy for future programmers to create new ones. DataForms
are derived from a custom Form called TDataForm, and several custom objects were created to implement input controls.

G. Nodes and Branches

A Node is in fact a collection of lines on the schematic that are linked together. Since there is no way to determine the number and location of lines the user will add or remove to a node, lines cannot be merged into one single Node element because the application and the user will both need information and control on every particular line.

To draw connections one custom object called TNodeBranch was created. All nodes in OUI are drawn with straight lines, of which, each is a Branch. Branches only have two ends, keep track to whom each end is connected, and have a set of properties and methods that make it easy to draw & move them. TNodeBranches represent lines, but the object is in fact bigger than the space taken by the line. This is useful in order to widen the area in which mouse events are detected.

During the design of TNodeBranch, a terminology was created to help define the object and its properties Figure 3 shows one node made of several Branches and illustrates some of the terms (internally topological points 4, 7 and 8 constitute the same electrical node).

![Figure 3. A node made of several Branches.](image)

Terminology:

1. Branches are all non segmented lines in the node. Branches can be:
   
   A Dead End, if one side connects to an element. e.g., 1.
A Bridge, if both sides are connected to other Branches, e.g., 2 and 3.
A Link, if it's a bridge with both sides connected to intersection points, e.g., 3.

2. Edges are any of the two extremes of a Branch. Edges can be:
   - Forks, knobs: are any intersection of more than two lines, 7 is fork, 4 is knob.
   - Corners: an intersection of two lines, e.g., 6.
   - Terminals: connects to an element, e.g., 5.

3. Intersection point: is wherever two or more Branches meet, e.g., 4, 6, 7.

In OUI all the Branches are drawn in straight angles (horizontally or vertically), so for every intersection point there can be only 4 Branches connected, i.e., for every Branch edge there can be maximum 3 other Branches connected, or one corner, or one element.

H. Full View Window

The Full View Window is a floating form that provides a reduced view of the whole schematic, such that if the Schematic size is bigger than the screen or than the Main Form it can be fully viewed. The portion of schematic visible on the Main Form is represented on the Full View Window by a focus rectangle.

The Full View Window was implemented with Delphi components. The only special considerations were caused by synchronization problems with scroll events on the Schematic Area, and drawing updates on the Main Form.

I. Printing and Print Preview mode

To supply different printing formats, a print preview mode is offered. After selecting Print Preview the appearance of the Main Form changes and shows a scaled view of the printout, as well as some controls to change printout formatting. Delphi's printing objects are a great help to implement printer handling. Printout formatting is up to the programmer, and required quite an amount of work. Several complications arose from the fact that fewer than needed properties are available in Delphi's printing objects.
1.2.2 Component library

Since OVNI has to allow for easy incorporation of additional Components, the approach chosen to load the Component’s information is the following:

When OUI starts running, in the initialization part of the code it reads a resource file containing information for every Component. The information includes the total number of Components, and particular data as: type of Component, location and names of terminals and symbol image.

The resource file is built using a batch file that calls Borland's Resource Compiler. It basically appends several files to the resource file 'COMP.RES'. The files appended are *.BMP files for the images, and *.STR files for the rest of the information.

All the information in a .STR file is very simple to define except for the location of the hotspots; so to simplify the process of adding hotspots on the terminals and update the batch file a utility called COMPEDIT was written. The operation of COMPEDIT is explained in Appendix I, Section 3.

COMPEDIT guides the user to input information, creates the *.STR file, updates file COMP.BAT and is capable of executing the batch file to update the resource file COMP.RES. The information contained by file COMP.RES is linked to OUI’s executable file and is read from there only when the program requires it. In this way a library of Components can be created that doesn’t take space in RAM memory. When the user places a new Component on the schematic area, the program loads to memory the resource information related to that Component, and discards that information after use.

1.2.3 Data handling

Although the communication with the NI is out of the scope of this phase of the project, it was necessary to specify some basic aspects of it to assure that it could be implemented later.

\[\text{\textsuperscript{22} Windows Bitmap files}\]
\[\text{\textsuperscript{23} Custom format to store raw data and strings}\]
The communication between the Interface and the Integrator follows a *Client-Server* approach. The Interface launches the Integrator, sends it data with the configuration of the network and waits for output data to plot. The communication is *full-duplex* as the Interface controls the simulation process and must be capable of notifying the user of changes to the configuration or to the parameter values.

As the Integrator can be located in the same machine, for desktop or simple on-line simulations, or in a different machine, for real-time or complex on-line studies, the *Client-Server* approach has to support communication in a distributed architecture. Delphi provides support for Client-Server design, but since the NI is being developed using a different compiler the issue was whether both programs could communicate using Delphi’s scheme. The conclusion was that this communication is possible using the Windows *Dynamic Data Exchange* protocol. The format used to pass the information to the Integrator has not been defined yet, and depends on the development of the preprocessor of NI.

Another issue about data handling is what format to use when saving the schematics in a file. The format chosen is custom, an allows to save the graphical information of the schematic with a minimum of data. This format is of use only for OUI, and although different formats might be implemented in the future, it will always be required to graphically reconstruct the schematic. One important aspect of the format is that it only makes distinction between Components and Branches. The saving and reading routines were made independent of the amount of data a Component needs. This means that to add a new Component the programmer only has to follow the rules specified to create new DataForms, and the program will make use of that code to save the Component.

1.2.4 Code organization

The code is dispersed among a few units. Unit Main.pas takes much of the burden and implements the Main Window. Unit Init.pas is used to store initial information and is the file where modifications must be made when a new Component is added to the library. There are two other basic units, one to implement the Full View Window, FullView.pas, and another for the Component List, CompList.pas. For every new DataForm, a unit and a form are added to the program. In contrast with other forms (Full View Window, Component List) DataForms
are created and destroyed dynamically, so it is necessary to let the compiler know that these forms are not to be *instantiated* on execution.\textsuperscript{24}

CompEdit was developed apart from OUI and is not for distribution with OVNI. It has a supporting structure of directories where bitmap files (for the symbols) and intermediate resource files are saved. These directories and their contents are explained in a file called 'What is all this.txt', that was saved with COMPEDIT.

In every unit, methods and functions were organized in sections according to the object they are related. The code was structured with care and in a consistent manner. There are comments all over the files explaining different steps of the program. Units for new custom objects have a header of comments about their features, methods and properties. All new objects were placed in a new tab of *Delphi's Component Palette*, except for OVDaframForm (which is used to create new DataForms).

\textsuperscript{24} How to do this is explained in Appendix I.
2. Implemented Features

In this section we list the features implemented in the first phase of OUI's development. This is mostly a descriptive explanation of the current ‘look and feel’ of the Interface, and does not include programming details.

When the program is launched, a window (Main Window) appears on the screen. Figure 4 is a snapshot of the Main Window.

The Main Window contains some important elements, which are pinpointed on Fig. 4. OUI's Specifications\(^\text{25}\) provided a general explanation of what are these elements, and their

\(^{25}\) Chapter III, Section 4.
function. The following outline describes the behavior of each element and their parts, as they have been implemented.

2.1. Title Bar and size of Main Window

- The title bar shows the program version and the name of the file currently open. The default name, for a new empty schematic, is 'Untitled' (see Fig. 5).

- The title bar has on its upper right corner a Minimize, Maximize/Restore and Close buttons. The default window size is maximized. The size and proportion of the elements were chosen to fit the size the schematic takes printed on a ‘letter-size’ page in Landscape orientation. Since the default size is maximized, to resize the user has to click on the restore button and resize with the mouse dragging the edges of the window. When the resize button is clicked, the window shrinks with its upper left corner placed in the upper left section of screen. This design assumes that way assuming that most users will prefer to have the window’s upper left corner in that area, to have room to expand the window towards the right lower end of the screen.

- There is a limit for the smallest size the main window can be resized to, the reason for this limit is to always keep in view the main Menu options and Speed Buttons.

- If the schematic is closed using the Close Button and the user made any changes to the schematic since the last Save, the program asks whether to save or discard those changes.

2.2. Menus.

- There are four Menu objects in the program, one Main Menu bar at the top of the Main Window (just bellow the title bar) and three Floating Pop Up Menus. The Main Menu is always visible, except when the program is displaying a Print Preview. There are four items in the Menu Bar, each one representing a pop down menu with several options. All
options and items are visible at all times, and when an option is not available it is dimmed. Figure 6 shows the Main Menu with the ‘View’ Pop Down Menu visible.

![Main Menu](image)

**Figure 6. OUI, Main Menu.**

- On the *Menu* bar the mouse cursor has always the default shape of a pointing arrow, even when the shape is different in other parts of the Main Window, e.g., when drawing connections or placing Components. The Main Menu items and their options are:

  **File Menu item**

  - Discards current schematic and creates a new empty one.  
  - Opens and loads a schematic from a file.  
  - Saves current changes into last selected file.  
  - Saves current schematic in a file specified by the user.  
  - Makes program enter Print Preview mode where printout formatting can be performed and a preview of the printout is shown.  
  - Opens a *Print Dialog Box*.  
  - Exit the program. If there are unsaved changes a confirmation message is displayed.

  *Table 3. Implemented features: File Menu item.*

  **Edit Menu item**

  - Shows an attached sub-menu with four choices to set the size of the Schematic Area.

  *Table 4. Implemented features: Edit Menu item.*

---

26 See section 2.14 of this chapter for more details.

27 See section 2.13 of this chapter for more details.
A schematic height and width are always multiples of a Unit Size Page height and width, such that different sizes are just different layouts of Unit Size Pages. Figure 7 shows the options in the Drawing Area sub-menu.

Figure 7. OUI, Main Menu -> Drawing Area | schematic sizes.

The options select sizes of 1x1, 1x2, 2x1, 2x2 from top to bottom. Changing sizes do not mean flipping pages into a new layout. For example changing from size 1x2 to size 2x1 does not mean that page in position II (see Fig. 2) is moved to position III. In fact, page II is discarded and a new page is created for III. If there are Components or Branches in a page that is about to be discarded a message appears explaining that resizing cannot be performed. The message also tells the position of the conflicting elements.

The menu options in the Drawing Area are graphical symbols rather than text: they also have a check mark to indicate which one is currently in use, and are highlighted as they receive the focus. When the schematic size is changed, the scroll bars are moved to the origin and the grid state (ON or OFF) is maintained. If the full View Window is visible then the image in it also reflects the change in size and scroll position.
View Menu Item

- Makes the grid visible or invisible.
- Displays the Full View Window. \(^{28}\)
- Displays the Component List. \(^{29}\)

Table 5. Implemented features: View Menu item.

The position of Components and Branches is always snapped to a casting grid of points. This is always true whether the grid is visible or not. New Schematics have the grid visible by default, but the state of the grid is saved when a schematic is saved into a file. The user has no control over the granularity of the grid. The grid is only visible on the schematic page, and it is not shown in the Print Preview Window nor in the Full View Window. The label of this menu option changes with the current grid selection. If the grid is visible the label is ‘Grid OFF’ and when the grid is not visible the label is ‘Grid ON’. Grid points are covered by Components and Branches, but when a connection is been drawn, grid points are shown through the new lines with a slight change in color.

Help Menu item

- Not implemented. Intended to open a Windows Help File.
- Displays an About Box with brief information about the program. The current message is quite simple and it will grow as new phases of OVNI are implemented. It could also show a logotype for the project.

Table 6. Implemented features: Help Menu item.

- The three Pop Up Menus are floating menus that are displayed when the user clicks the right mouse button on top of certain objects. These objects are the Schematic Page, Components, and Branches. All Pop Up Menus are disabled when: 1) A Component has been selected from the Component Palette or the Component List and it has not been placed yet (i.e. when the cursor shape is a hand holding a Component), or 2) when a

\(^{28}\) See section 2.8 of this chapter for more details.

\(^{29}\) See section 2.9 of this chapter for more details.
connection is being drawn (i.e. when the cursor shape is either a pointing index or a hand holding a pencil). When the pop up menus are disabled, the right-mouse button is used to cancel the current operation (same effect as the <Esc> key).

The Pop Up Menus and their options are:

<table>
<thead>
<tr>
<th>Menu</th>
<th>Triggering action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schematic Pop Up Menu</td>
<td>right click on a free point on the Schematic Area. 30</td>
</tr>
<tr>
<td>Component Pop Up Menu</td>
<td>right click on a Component. 31</td>
</tr>
<tr>
<td>Branch Pop Up Menu</td>
<td>right click on a Branch. 32</td>
</tr>
</tbody>
</table>

Table 7. Implemented features: Pop Up Menus.

Fig.8 shows the three Pop Up Menus:

![Schematic Pop Up Menu](image1)

![Component Pop Up Menu](image2)

![Branch Pop Up Menu](image3)

Figure 8. OUI, Schematic, Component and Branch Pop Up Menus.

2.3. Top Panel

- The Top Panel has a cluster of Speed Buttons on its right side and a Tabbed Palette on the left (see top of Fig. 4). The Top Panel is always visible, except when the program is in Print Preview mode.

2.4. Speed Buttons

- There are 7 Speed Buttons so far. They are all shortcuts to options available in other parts of the interface. Each button can be identified by an icon and a hint. A short explanation is displayed in the Status Bar every time the mouse is on top of a Speed Button.

---

30 See section 2.6 of this chapter for more details
31 See section 2.10 of this chapter for more details
32 See section 2.12 of this chapter for more details
The shortcuts are:

<table>
<thead>
<tr>
<th>Speed Button</th>
<th>Equivalent action</th>
</tr>
</thead>
<tbody>
<tr>
<td>🖼️ New File</td>
<td>Main Menu -&gt; File</td>
</tr>
<tr>
<td>📦 Open</td>
<td>Main Menu -&gt; File</td>
</tr>
<tr>
<td>☝️ Save</td>
<td>Main Menu -&gt; File</td>
</tr>
<tr>
<td>📹 Print</td>
<td>Main Menu -&gt; File</td>
</tr>
<tr>
<td>📹 Print Preview</td>
<td>Main Menu -&gt; File</td>
</tr>
<tr>
<td>☐ Connect</td>
<td>Schematic Pop Up Menu</td>
</tr>
<tr>
<td>📚 Show All Schematic</td>
<td>Main Menu -&gt; View</td>
</tr>
</tbody>
</table>

Table 8. Implemented features: Speed Buttons.

2.5. Component Palette

- The Component Palette has a Toggle Check Box, Tabbed Pages and Speed Buttons. Figure 9 shows a snapshot of the Component Palette.

![Component Palette Snapshot](image)

Figure 9. OUI, Component Palette.

All OVNI Components are available from the Component Palette or from the Component List. On the Component Palette Components are grouped by categories in different tabbed pages. In the Component List Components are displayed alphabetically in a list. The Component Palette is on the top panel and is always visible, except when Print Preview is selected. The names for each element category are indicated on the top edge of the tabs.
• To view the Components on a Tabbed Page the user must click the edge of the tab. If the Main Widow is resized such that some tabs cannot be visible then two push buttons, with a right and left arrow icons, appear on the upper right corner of the Component Palette. These buttons slide all tabs in the available area to make all of them reachable without resizing the Main Window.

• Each Component is represented by an icon and a name. The name appears on a hint window when the mouse is placed on top of the element.

• To place a Component on the schematic, the user must follow two steps: 1) Select a Component, either from the Component Palette or from the Component List; 2) Place the Component on the Schematic. On the Component Palette Components are selected by clicking on their Icon. The mouse cursor then changes to the shape of a hand holding a Component to indicate that the Component has been selected. To place the Component the user must left-click on the schematic. After dropping the Component, a schematic symbol for the Component appears on the page, with its upper left corner located at the point where the cursor was when the mouse button was pressed.

• A message appears on the Status Bar when a Component is selected.

• To Cancel a Component selection, the user can either press <Esc> key or the right mouse button. During selection, all pop up menus are disabled to free the right mouse button for canceling the selection. As soon as the selection is canceled the right button is re-assigned to the Pop Up Menus.

• Toggle Check Box: The user can check this box with the mouse. The default state is unchecked. The Check Box controls whether the Component Speed Buttons operate as Toggle Buttons or as Push Buttons. When unchecked the user must select from the palette one Component at the time for every single Component he/she wishes to place on the schematic. Some times a user might need to place several copies of one type of Component. In such case the successive mouse movements to select the same Component would be redundant. To avoid this situation, the user can check the Toggle Check Box, and then select a Component. The Component will remain selected to place as many copies of it as desired until any of the following actions occurs: The Check Box is unchecked;
another Component is selected; or the Selection is Canceled, either by pressing the <Esc> key or the right mouse button. If another Component is selected and the toggle box is still checked, then that Component is selected for multiple copies.

- Component Categories: In successive versions, OVNI is intended to include an increasing number of Component Models, some of which may not be foreseen at this moment. To simplify locating an arbitrary amount of models, the Component Palette was divided into categories, with a tab assigned to each one. More Tabs can be added in the future if necessary. At this point, only 3 sample elements are implemented (Resistor, Capacitor and Inductor); however, the tabs and buttons for a basic set of Components have been implemented. The chosen Categories and their Components are:

<table>
<thead>
<tr>
<th>Tab Name</th>
<th>Description</th>
<th>Actual Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Elements</td>
<td>Basic passive elements</td>
<td>Resistors, Inductors, Capacitors</td>
</tr>
<tr>
<td>Compound Elements</td>
<td>Built from other elements</td>
<td>Transformer, Pi-Circuit, Transmission Line, HVDC Converter.</td>
</tr>
<tr>
<td>Control Blocks</td>
<td>Elements that perform control functions</td>
<td>None</td>
</tr>
<tr>
<td>Electronic Devices</td>
<td>Solid State elements</td>
<td>Diode, Thyristor, Triac, GTO.</td>
</tr>
<tr>
<td>Sources</td>
<td>All types of power sources</td>
<td>Voltage Source, Current Source, Dependent Voltage Source, Dependent Current Source</td>
</tr>
<tr>
<td>Switches</td>
<td>Switching devices</td>
<td>Time-Controlled Switch</td>
</tr>
<tr>
<td>Meters</td>
<td>For scopes</td>
<td>None</td>
</tr>
</tbody>
</table>

*Table 9. Implemented features: Component categories.*

### 2.6. Schematic Area

- The Schematic Area is where the circuit schematic is drawn. The Schematic Area has four important features: Size, Grid, Pop Up Menu, Scroll Bars and one important property: the schematic drawn on it can be saved into a file.

- Default appearance of the Schematic Area is an empty 'Unit Size Page' schematic with Grid.
• The size of the schematic area can be modified. There are four possible sizes, all multiples of a unit page size. The unit page is called ‘Unit Size Page’ and it was chosen to be in a 1:1 scale with a Letter-size paper page. Sizes available are arrays of: 1x1, 1x2, 2x1, 2x2 unit pages. To change the size user must select: Main Menu -> Edit | Drawing Area | one size. This was explained with more detail in section 2.2 of this chapter.

• The Grid is a guiding array of points that covers all the schematic area. The grid can be set to be visible or invisible, but the location of Branches and Components is always snapped to the grid even when the grid is not visible. The granularity of the grid is fixed. To display the Grid ON or OFF, user must either: select Main Menu -> View | Grid ON/OFF or, on a free spot of the schematic area, click the right mouse button: this will display the Schematic Pop Up Menu from which the option Grid ON/OFF is available. The grid can be set ON or OFF at any moment, and the state of the grid is saved with other information whenever the schematic is saved. Mouse movement is not snapped to the grid. The Grid is not shown neither in the Print Preview window nor in the Full View window.

• Schematic Pop Up Menu: Right clicking on any free spot of the schematic area makes the program display a floating pop up menu. This Menu is called the Schematic Pop Up Menu. It contains a set of commands that are considered to be of frequent use. All commands in the Schematic Pop Up Menu are also available at other parts of the program. The items in the Schematic Pop Up Menu, and their shortcuts, are:

<table>
<thead>
<tr>
<th>Menu option</th>
<th>Same action as ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>• Connect Speed Button on top panel</td>
</tr>
<tr>
<td>Grid ON/OFF</td>
<td>• Main Menu -&gt; View</td>
</tr>
<tr>
<td>Show All Schematic</td>
<td>• Main Menu -&gt; View</td>
</tr>
<tr>
<td>Component List</td>
<td>• Main Menu -&gt; View</td>
</tr>
<tr>
<td>Save</td>
<td>• Main Menu -&gt; File</td>
</tr>
</tbody>
</table>

Table 10. Implemented features: Schematic Pop Up Menu.

• Scroll Bars to span schematic area: The Schematic Area has a vertical and horizontal scroll bars. The scroll bars are used to span the schematic on the Main Window. The size of the
sliding bar reflects the portion of the page that is not visible. e.g. for a small non visible area in the vertical direction the bar occupies almost all of the vertical scroll bar strip. As the size of the schematic is changed, the size of the bar is adjusted. The page can be spanned by dragging the focus rectangle of the Full View Window. The schematic scrolls automatically whenever the user drags a Component or draws a connection. In all cases the Full View Window Focus rectangle moves to reflect the new position.

2.7. Status Bar

- The Status Bar is located at the bottom edge of the Main Window.
- It has two panels: the left panel displays the mouse position, in pixels, on the schematic page. The origin of the coordinates is located at the upper left corner of the schematic page and the value of X and Y increase as the point moves to the right and down on the screen. Mouse location is shown as a means of measuring dimensions and locating points for those users who wish to do so. It is not a necessary feature for the interface but at times it shows to be handy. The two panels can be seen in Fig. 10.

![Figure 10. OUI, Status Bar.](image)

- The Status bar right panel is used to show hints. Some elements of the program display a fly-over hint indicating their name or giving some clue as to their purpose. In the same way, some elements display a larger hint on the status bar to advise the user with possible choices or instructions. The intention of this is to provide directions for all actions that might not seem natural to the user. For example, after selecting a Component from the Component Palette, the Status bar shows the following message: 'Click Left Button on Schematic to place Component, Click Right Button or press <Esc> to cancel selection'.

---

33 See section 2.8 of this chapter for more details
2.8. Full View Window

- The full view window shows a non-editable reduced view of all the schematic on a small window that always stays on top of the schematic page. To display the Full View Window user must either Click on the ‘Show All Schematic’ Speed Button on the Top Panel or select Main menu -> View | Show All Schematic or select Show All Schematic from the Schematic’s Pop Up Menu. The Full View Window cannot be resized but it can be closed or minimized. If the window is already visible but minimized it will be restored to its original size.

![Focus rectangle](image)

*Figure 11. OUI, Full View Window.*

- The dimensions of the window reflect the dimensions of the schematic area. For a size of 1x2 the window is twice as wide as it is for a schematic of size 1x1. Whenever the schematic’s size is changed the dimensions of the Full View Window are updated.

- The window shows a scaled image of the schematic without the grid. On that image no elements can be selected. If the size is bigger than one page (1x1), dashed lines show the limits of each page. This is useful in order to see the distribution of elements when printing in several pages or also to check if a page can be discarded.

- A red focus rectangle shows the portion of the schematic that is visible on the Main Window. The rectangle can be dragged thus panning the schematic on the Main Window,
and readjusting the position of the scroll bars. If the Main Window is resized, the focus rectangle is also resized, proportionally and simultaneously. If the window is dragged, closed, or reopened, the window will appear at the last location. The first time the window is opened it appears on the center of the screen.

2.9. Component List

- Provides a way to select Components alphabetically. All Components available in the Component Palette are listed in the Component List.

![Component List](image)

*Figure 12. Component List.*

- To display the Component List, the user can either select main menu -> View | Component List or select Schematic Pop Up Menu -> Component List.

- The Component List has a scrollable List Box and an Edit Box. The List Box shows a list of all Components, with their icons, alphabetically ordered from the top to the bottom of the list. There are two ways to look for a Component in the Component List. One way is to scroll the List Box with the mouse and look for a Component name or icon. The other way is to type the name of the Component in the Edit Box: as letters are typed in the List, the program performs an incremental search and scrolls the List Box to select the
Component whose name is the closest match to the input text. Once a Component has been found, the user must click select if he/she wishes to place it on the schematic. Selecting a Component from the Component List has the same effect as selecting it from the Component Palette.

- The Component List cannot be minimized or resize, but it can be closed. If moved, closed, or reopened it re-appears at its last location. Also, it always stays on top of the main Window. The first time the Component List is opened it appears on the center of the screen.

2.10. Handling Components

- How to place Components on the Schematic was explained in sections 2.5 ‘Component Palette’ and 2.9 ‘Component List’.

- Identifying a Component and its terminals: Figure 13. Shows three sample Components.

![Figure 13. OUI, Sample Components.](image)

Components on the schematic area are represented by a symbol. One particular symbol is assigned to each type of Component. The symbols correspond to generally used electrical symbols, although for some Components some new proprietary symbols might be used (e.g. for an HVDC Component). Different Components can have a different number of terminals, which are represented in the symbol by small circles. All these symbols are built by the programmer and the user has no control over their appearance. Construction of these symbols is discussed in Appendix I. In principle two Components of the same type look
the same on the schematic. The Component name is displayed on a fly-over hint window whenever the mouse is set on top of the Component. The name of the terminals is also shown in the same way whenever the mouse is on top of a terminal. The hint window for terminals has a different background color from that of all other hint windows in the program. Beware that two Components of the same type have terminals with the same names. For example, a Component named R1 has a pair of terminals called ‘k’ and ‘m’, while on the same schematic a Component called R2 also has a pair of terminals called ‘k’ and ‘m’. Terminal names are independent of the Component name, or the name of the nodes they are connected to: their name reflects an internal property of the Component itself. The name of a Component is also visible at the top of any open DataForm.

• Naming Components: When a Component is placed on the schematic, it receives a default name. This name is made of a prefix that indicates the type of Component and a number. For example, R1, L2, C23. The number is increased by one for each type of Component every time a new Component of that type is added. Deleting a Component does not decrease the counter which is also saved with other information when a schematic is saved into a file.34 However, the user can rename a Component to any name. To do so, the user must right click on the Component to open the Component Pop Up menu, then select the option ‘Rename’. An input Dialog Box prompts the user to type in a new name. Any character available in the keyboard can be used for the name, including blank spaces. Only an empty string is not accepted. In that case the name remains the same and no error message is displayed. If the new name is already in use by another element (Component or Branch) then a warning message is displayed and the name is not changed. The name is not case sensitive.

• Moving: Any Component with all terminals still unconnected can be moved anywhere in the Schematic Area. This is also true for any rotation of the Component. As soon as one terminal is connected, the Component can no longer be moved. But if all connections are deleted the Component is free to move again.

34 See Chapter V, for details on how to improve this naming mechanism.
To move a Component the user must Right Click on it and, keeping the mouse button down, drag the symbol to a new location. While the mouse button is down the cursor shape is changed into an open hand (symbolizing dragging); the image of the Component stays in its original place but a dashed rectangle representing the bounds of the Component moves with the cursor to show the progress of the dragging action. When the object is dropped (mouse is released) the Component Image is moved to the new location. As the cursor approaches the edges of the visible section of the schematic, the scroll bars move automatically. The faster the user approaches the edges the faster the scrolling. This allows to reach any location on the entire Schematic Area beyond the visible area. Once in place, releasing the right mouse button concludes the dragging and places the object in the new location. Since Components are always snapped to the grid the final location is not exactly the same one where the mouse is released but the closest grid point to that location. The relative position of the mouse inside of the Component while it is dragged is also considered when placing the Component in its final location: if the user drags the symbol from a mid point in the Component the image is placed with that same midpoint close to the location where the mouse was released.

Restrictions to selecting and dropping a Component: A Component can be dragged from any point on its symbol except the area that encloses the terminals. This is so to avoid confusions with connecting Components and discourage any attempt of performing connections while moving Components. A Component can be dropped at any point of the Schematic Area and, therefore, it can overlap another Component or any Branch.

If, while dragging, the mouse is placed on any spot outside the Schematic Area, even outside the main Form, the cursor shape changes to a crossed circle (NoDrop symbol) to indicate that the Component cannot be dropped in that location. If the mouse is released outside the valid area dragging is canceled and the Component remains in the original position. If the Component is dropped such that the mouse is still inside the Schematic Area but some part of the Component trespasses an edge, then the final position is chosen such that to keep all the Component inside the Schematic Area.
• If the Full View Window is open, the position of the newly moved Components is shown in it as soon as the Component is dropped.

• Bring To Front: Placing and moving of Components allows overlapping. In such case, to bring a particular Component to the front, the user must right click on it. If the Full View Window is opened the action is also reflected there. Branches cannot be brought to the front, this is so to avoid confusion where several Branches cross each other.

• Component Pop Up Menu: Components have an associated Pop Up Menu. The user may right click on a Component to open the Component Pop Up Menu. Any point on a Component, including terminals, are valid points. The right mouse button is also used to cancel a Component selection (when user selects a new Component from the Component palette or the Component list), or to cancel drawing of a connection, in those cases, the Component pop Up menu is disabled. The items of the Pop Up menu are:

<table>
<thead>
<tr>
<th>Menu option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rename</td>
<td>To rename a Component. Enabled at all times.</td>
</tr>
<tr>
<td>Rotate</td>
<td>To right rotate a Component in steps of 90°. This option is disabled if at least one terminal of the Component is connected.</td>
</tr>
<tr>
<td>Show DataForm</td>
<td>Displays the Component’s Data Form. Has the same effect as double clicking on the Component.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the Component from the schematic.</td>
</tr>
</tbody>
</table>

*Table 11. Implemented features: Component Pop Up Menu.*

• Rotating: A Component that has all terminals unconnected can be rotated. To do so, the user must select Component Pop Up menu -> Rotate. This rotates the Component to the right in steps of 90°. Rotating a Component can cause it to overlap other Components or Branches, if so, it is up to the user to move the Component away. If a Component is too close to an edge of the schematic, rotating it can result in part of the Component trespassing the edges. If so, it is also up to the user to relocate the Component. Rotation of a Component is saved with other data whenever the schematic is saved into a file.
• Deleting: A Component can be deleted provided that none of its terminals are connected. To delete a Component user must select: Component Pop Up Menu -> Delete. Unlike the Rotate Item in the Pop Up menu, Delete is not disabled when a terminal is connected, but if the user selects it a warning message is displayed. This was made so to make the situation more transparent and help the user understand he/she should disconnect all terminals first. This action is irreversible and causes the loss of all data in the Data Form. Deleting a Component does not change the counter used to assign default names to new Components of that type.

2.11. DataForms

• Description: Every Component on the schematic has an associated DataForm. Figure 14 shows a sample DataForm. A DataForm is a window where the user provides parameter and simulation information. DataForms have the same appearance for Components of the same type, except for the name of the Component displayed on the top, and the information the user feeds in. To display a DataForm the user can either double click on a Component or select: Component Pop Up menu -> Show data Form.

• By default a data Form appears the first time in the center of the screen, but if moved it will reappear in the new location. All data Forms share the following characteristics:

  • Tittle bar with OVNI’s icon, the text ‘DataForm’ and a close button.
  • They cannot be resized, minimized nor maximized.
  • A Top panel with a label showing the type of the Component and its name.
    E.g. Resistor C1.
  • A lower panel with Save, Cancel and Help Buttons.
  • An image showing the Component’s symbol and terminal names.
  • Tabbed pages with input objects. Each tab corresponds to one model supported by OVNI. Information in all tabs is always saved but only the last selected tab is used for the simulation.

59
• The input fields, and the number and name of the tabs will vary for different Components.

• Using Data Forms: The program has at this point 3 sample Data Forms. One for each of these Components: Resistor, Capacitor and Inductor. Fig. 14 shows the Sample Resistor DataForm. A DataForm opens the first time with default values, this includes a default selected tab. If the user selects the ‘Save’ Button all changes to fields, tabs and final position of the DataForm are saved and the Form is closed. If the user selects ‘Cancel’ all changes to the fields, tabs and position are discarded and the form is closed. The Help button has no function at this point. Clicking on the close icon of the title bar is equivalent to selecting the ‘Cancel’ Button except for a warning message asking the user if he/she wishes to Cancel or not the Data Form.

![Sample Resistor DataForm](image)

**Figure 14. Sample Resistor DataForm**

• The sample DataForms are very simple but implement the basic behavior of data Forms and provide a new type of edit box and a new type of list box. The special features of the new edit box are:
  • Only accept numbers, '!', '-' and system keys (<Enter>, <Home>, arrows, etc.)
  • If the user presses <Enter>, it switches focus to next element.
  • It takes only one '!', and if there is no number at the left side of the '!', it inserts a '0'.
  • If the user presses '-' it toggles sign.
• If the user presses the <Backspace> key to erase a '.' and there is a '0.' In the sequence, the action erases both
• The default string is ‘0.0’
• Formats all input to two decimals

The new List Box handles Greek symbols and switches focus if user press <Enter>.

• As already indicated, all the information in the fields of a Data Form, including the tab selected and the position of the form on screen are saved when the schematic is saved into a file. Also, the fact that the information in the DataForm is saved or canceled is taken into account when the program decides if there have been modifications to the schematic worth saving.

2.12. Connections

• Drawing a connection: To draw a connection the user must draw with the mouse all the Branches that will link the two connecting points. A connection can be made to link terminals, Branches, or terminals with Branches. To start drawing a connection the user must either click the ‘Connect’ Speed Button at the Top Panel or select: Schematic Pop Up menu -> Connect. At that point if the schematic is empty a message appears explaining that there are no Components to connect and the Connect Speed Button (if it was pressed) graphically pops out. If a connection is possible, the mouse cursor changes into a hand with a pointing index and a message in the status bar explains to the user that he/she must select, with the mouse, a starting point for the connection. The starting point can only be a Component terminal or a Branch. Therefore the user must place the cursor on top of one of those elements and select it with the mouse. To show when the mouse is on top of a terminal a fly-over banner appears whenever the mouse is on top of any of them. To show that the mouse is on top of a Branch, the Branch is highlighted (color changes to red) when the mouse is on top of it. From a programming point of view the sizes of Branch thickness and a terminal area are bigger than the image the user sees, this facilitates mouse selection of the objects. To indicate the starting point for drawing the connection the user must click left mouse button. After clicking the mouse, the cursor changes into the shape of a hand
holding a pencil horizontally. If the point selected as start is a terminal, all terminals of that Component are highlighted (covered by a red square) and remain in that condition until the drawing is finished or canceled. If the point selected is on a Branch, a blue knob is drawn on that point to indicate point of selection: if the drawing is canceled the knob is erased, and if a connection is performed successfully the knob is kept to indicate a point of electrical connection.

- Next step, after selecting a starting point, is to draw vertical and horizontal lines to link that point to a target point. The lines are drawn only with the cursor keys. Moving the mouse displaces the cursor but does not change the last drawing point. The lines can expand and contract only in the horizontal and vertical directions and in steps that match the grid spacing. It is possible for the user to change his/her mind while drawing by going back on his/her steps with the cursor keys. As the direction in which the lines are drawn changes from Up to Down the shape of the mouse cursor is also changed into a hand holding a pencil up or down, accordingly. The hotspot of the mouse is also shifted from its upper left corner to the lower left corner. The lines are drawn freely on any empty space of the schematic page, and are not allowed to pass over a Component or to trespass the edges of the schematic. Neither it is possible for a link been drawn to cross over itself (this was considered as messy drawing). To distinguish between the connection that is been drawn and others already existing, the drawn one is highlighted with red color. If a new line is about to cross over another Branch a message prompts asking the user if wishes to connect to that point. If the answer is ‘Cancel’ the whole drawing is aborted; if the answer is ‘No’ the line is advanced to that point and while the cursor is on top of the crossing point the mouse shape is changed into a hand holding a pencil horizontally (which proved to be a handy way of showing crossing of lines and cursor). A similar message is shown if the next point is about to enter a Component Terminal. The only difference between both messages is that for a Branch the Node Name is displayed while for a Component the names displayed are those of the Component and the terminal. Generally, the message window appears in the center of the screen, however if the target point is in an area that could be covered by the message then the window appears in another location. This was made so because at that point it might be important for the user to visually check the connection.
he/she is about to make. Answering ‘Yes’ to any of the two messages establishes a connection. A new line crossing over a Branch does not show the message if it is moving back on its steps, but as long as the user moves the line on top of a Branch, even the same one, the message is displayed. At any point in this process the drawing can be canceled either pressing the <Esc> key or the Right Mouse Button. If so, all new lines are erased and the starting element is deselected.

- If a target point is selected: All red lines are replaced by blue Branch lines, and the starting point element is deselected. Depending on the target point, the program performs different actions.

- If source and target are both free Component Terminals, this is a new node. A new default name is generated for that node and is suggested to the user in an input dialog box. The user is here free to accept that name or type in any other name. After that, node information in saved in all Branches making the link, and in both Components.

- If one point already belongs to a node but the other not, then information about the existing node is saved in all new Branches and the newly connected element.

- If two points that belong to two different nodes are connected together, a message appears on screen prompting the user to type in a new name. A name for the target node is proposed by default, but the user is free to type in any other name. In this situation, two nodes are merging, and although the name of either of the two nodes might be a reasonable choice, the user can select any other name. If a new name is selected the two original names will be replaced by the new one; e.g., if the connecting starting point is a Branch belonging to ‘Node4’ and the target is a Branch of ‘Node2’, the program will prompt the following message: Type in new name: ‘Node2’. ‘Node 2’ is the default name in the message window. If the user selects ‘Node2’, then all Branches and terminals belonging to ‘Node4’, no matter their location, will be updated as belonging to ‘Node2’. ‘Node2’ becomes then a name not used in the schematic. If the user types in ‘Node4’ as the new name, all terminals and Branches of ‘Node2’ will be updated to ‘Node4’, and ‘Node2’ becomes a name not used. If the user types in as new name ‘My Node’ then all references to ‘Node2’ and ‘Node4’ are changed to ‘My Node’. Any new name is valid
unless it conflicts with the name of a Component or another node not involved in the connection. For example, if the same schematic has another node named ‘Node A’, then ‘Node A’ is not a valid entry, or if there is a Component called ‘R1’ then ‘R1’ is not a valid entry. If an invalid entry is made, a warning message is issued with an explanation and the user gets a new chance to type in a new name.

- If any of source or target points are on a Component that had all terminals free, from then on the Component is not allowed to be moved or rotated unless all terminals are first freed.

- Default names for new nodes are built with a prefix ‘Node’ and a number that is increased sequentially as more nodes are created, e.g., ‘Node1’. If by any chance the user renames an Element with a name of that sort (unlikely to happen but possible), the program is smart enough not to that name as a default name.

- Checking for name conflicts is not case sensitive, but if the user types in a node name with a different casing, e.g., NODE1, and a previous reference was Node1, then the new case is updated in all references to that node.

- If, when prompted to type in a name, the user selects ‘Cancel’ or closes the window, the connection is canceled.

- At any point where the new link attaches to a Branch a knob is drawn.

- Branch Pop Up Menu: Nodes are made up of Branches, and for all Branches there is an associated Branch Pop Up Menu. To display this Menu the user must place the mouse on top of it and click the right mouse button. If mouse location is correct the Branch is highlighted and a floating Pop Up menu appears. There are two items in the menu: Rename Node and Delete. Clicking on any point of the screen, pressing <Esc> or selecting any of the options in the Pop Up menu will close it.

<table>
<thead>
<tr>
<th>Menu option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rename Node</td>
</tr>
<tr>
<td>Delete</td>
</tr>
</tbody>
</table>

- To rename the Node a Branch belongs to.
- Deletes a Branch from the schematic.

Table 12. Implemented features: Branch Pop Up Menu.
• Rename Node: Whenever the mouse is on top of a Branch a *fly-over banner* with a text appears. This text is the name of the node to which this Branch belongs. What the rename command does is not to change the name of a particular Branch, but the name of the node that Branch is part of. Therefore, renaming a Branch affects any other Branches and Component terminals that are connected to that node. To rename a node, the user must select: Branch Pop Up Menu -> Rename Node. A message dialog box appears on screen showing the actual name. The user can type in any other name as long as it is not in use by another node or Component. If the name is in use a warning message is issued and the renaming process is ended.

• Delete: To delete a Branch or a portion of it, user must select: Branch Pop Up menu -> Delete. A Branch is any straight segment of a connection that can be identified by two extreme points. Branches are created as the user draws lines while connecting elements, but there can be more Branches on the schematic than just the lines drawn with the cursor keys. This is so because every time a junction is created (a link to an existing Branch) a Branch is broken into two smaller segments (i.e., Branches).

A Branch can play different roles in a node, depending on what elements are connected to its extremes, and according to those roles Branches and their extremes are classified into different types\(^{35}\). Deleting a Branch has different consequences depending on the role the Branch played in a node. Any deleted Branch can cause the deletion of other Branches attached to it. This propagates deletion of Branches until a junction point or a terminal is found. The effect of this is to delete a whole connection arm and not leave any Branch hanging on the schematic with an unconnected end. The following image shows an example:

---

\(^{35}\) See Chapter IV, section 1.2.1.
Figure 15. OUI, Deleting Branches.

Notice how deleting the Branch in the upper left side caused the deletion of another Branch.

- Deletion of Branches is irreversible. If a whole node is deleted, the name of the node is available for new nodes or Components. If deleting a Branch frees a Component from being connected in all of its terminals, then the Component can again be moved or rotated.

2.13. Printing

There are two commands related to the printing of schematics: Print Preview and Print.

- Print Preview: A schematic can be printed in several ways, i.e., with different sizes, title text, paper orientation, etc. Selecting among these alternatives is called here 'Print Formatting'. Other options like number of copies, printer used, etc. are more related to printer commands and are quite standard in Windows applications. To perform print formatting and see a scaled preview of the final printout select Main Menu -> Edit | Print Preview. When the program is in preview mode the appearance of the main Window changes. The title bar and the status bar stay in place but the Top panel is replaced by a new panel and the schematic page is replaced by an scaled image of the printout (See Fig. 16). The size of the scaled image depends on the dimensions of the Main Window. If the Main Window is resized while the program is in Print Preview mode, the size of the
reduced image is adjusted to fit the area available, but always keeping the correct ratio of page width / page height. This is true for all printout orientations.

Figure 16. OUI, Print Preview Mode.

- In principle the scaled image shows the schematic (or a part of it) inside a black frame, with a title text in the lower side of the frame. Other features like Page Number, or Date and Time can be added also. The Top Panel contains a set of buttons, Check Boxes, etc. that control Print Formatting. Also, in Print Preview mode all Pop Up Menus are disabled. A description of the options available follows.

- Print Preview Top panel: Figure 17 shows the distribution of controls on this panel.

Figure 17. OUI, Top Panel in Print Preview mode.
• ‘Print’ Button: displays a Print Dialog Box to select the number of copies and printer. It has the same effect as selecting the Print Speed Button on the Top Panel or to select Main Menu -> Print.

• ‘Prev Page’ and ‘Next Page’ buttons: If the check box labeled ‘Fit all in one page’ (in the ‘Options’ section of Print Preview Top Panel) is unchecked, then the schematic is printed into several pages. The size of the schematic is always a multiple of a Unit Size Page that fits in a scale of 1:1 the dimensions of a letter size paper page in landscape orientation. Therefore, to print in a scale of 1:1 the whole schematic is mapped into several pages of which each corresponds to one Unit Size Page section of the whole image. The preview image shows one page at a time. Prev Page and Next Page shift the page displayed. Both buttons are enabled only if there is a next, or previous, page to display, otherwise, they are disabled. As the user switches from one page to another the program takes some time to prepare the new image. To reduce that delay if the user is frequently switching pages, the program has some built in logic that makes the process faster.

• ‘Back’ Button: Exits Print Preview and saves all changes made.

• ‘Cancel’ Button: Cancels all new changes and exits Print Preview.

• Page Tittle: The tittle text appears always centered at the lower side of the printout and it is the same text for every page of the same schematic. The default text is always ‘Untitled’, but can be changed by typing any new text into the Tittle edit box. If the text is so long that it will overlap either the page number at the left side (if available) or the data and time label at the right side (if selected) a warning message appears on screen.

• ‘Font’ Button: Clicking this button changes font settings for all text shown in the lower side of the printout. This affects the tittle text, time and date label and page number. Different colors, styles, font types and sizes can be selected. If any of these changes makes the tittle overlap over the page number or time and date labels, a message is shown on screen.

• Options Group Box: Encloses several options that are selected by check boxes or radio buttons. These options are:

36 See Chapter IV, section 1.2.1.
• 'Fit all in one page': If checked, the program shrinks the schematic printout to fit in only one page. In this case, the orientation of the page is under control of the program and the landscape and portrait orientation radio buttons are disabled. Sizes: 1x1, 1x2, 2x2 are fitted in landscape orientation. Size 2x1 is fitted in portrait orientation. Title and date and time text are still displayed but the page number is replaced by a 'Full View' label. The Prev Page and Next Page buttons are disabled too.

• 'Add date and time': If checked, current date and time are printed in the lower left side of the page with same font as the title text.

• Landscape Orientation and Portrait Orientation: Available if 'Fit all in one page' is not selected. These radio buttons are mutually exclusive and control the orientation of the printout.

• The status bar on the left panel shows information about the current page in view while the right panel shows information about possible actions.

• Print: Available through Main Menu -> Print, Print Speed Button of Top Panel or Print Button of Print Preview Top Panel. This command displays a Print Dialog Box to select the number of copies and printer and to send the printout to a printer. It is a standard print dialog box of the Windows OS. Some advanced options on it (in the property section) might conflict with the settings in print preview. For example, selecting Portrait Orientation when print preview has Landscape Orientation selected will cause a conflict. This conflict can not be avoided if the program is to be kept compatible with all Windows Print Dialog Boxes.

2.14. File Management

This section explains how to create a new schematic, how to save a schematic into a file, and how to read a schematic file. At this point the application only saves schematic information in a format of its own; the files are terminated by the extension '.ovs'.

• New schematics: By default OVNI opens with a new empty schematic. A new schematic can also be created at any time by selecting main Menu -> Edit | New ... If there are changes to save in the current schematic, a confirmation message is shown and the schematic can be saved before creating the new one. New schematics have a size of 1x1, a
visible grid and are named 'Untitled'. After creating a new schematic the save option of
Main Menu -> Edit is disabled until a change is made to the schematic.

- ‘Save and Save As ... ‘: ‘Save As’ is intended to save a schematic under a new name. For
new, unsaved schematics this action is necessary to create a new schematic file. For
schematics that have been read from a file, this option would save that schematic into a
new file with the changes done and will switch the focus of the program to that new file.
Selecting ‘Save As ...’, as well as ‘Save’, for new schematics opens the Save As Dialog
Box. This is a standard windows Save As box, with the focus originally set on the program
directory. As the directory is changed during different ‘Save As’ actions the focus switches
to the last directory selected. ‘Save As’ is available only in Main Menu -> File | Save As
....

- The ‘Save’ command is available either in Main Menu -> File | Save or in the Schematic
Pop Up menu -> Save. Unless the schematic is a new schematic, ‘Save’ just updates the
information in a file. The file name is the same shown in the title bar of the program. If the
schematic has never been saved, ‘Save’ opens the Save As Dialog Box. The program keeps
track of when new modifications have been done to the schematic after the last save, and
this is reflected in the enabling and disabling of the ‘Save’ option in both Main Menu ->
File and Schematic Pop Up Menu.

- ‘Open’: To open and load a schematic file, select Main Menu -> File | Open. This opens a
standard Windows95 Open Dialog Box The focus is originally set on the program’s
directory but switches to the last selected directory in successive ‘Open’ actions. After
selecting the file the integrity of it is checked; if an error is found, a message is shown
describing the problem. Once a schematic file has been successfully loaded, the name of it
is shown in the title bar. If there is another schematic on the Schematic Area when the
‘Open’ command is issued, the program shows a message asking whether to save the
previous schematic, unless there have been no modifications since the last save.

- For both Open Dialog Box and Save As Dialog Box the file search filter is set to (*.ovs),
but it can be changed to (*.*) . If a file is saved with a different extension there is no
complain from the program and it can also, latter, be loaded. However it will be up to the
user to remember the extension. If the file name to be opened does not exist, a message is shown explaining that the file cannot be found, and the Open Dialog Box stays open for the user to input a new name. If the file selected for a ‘Save As’ action exists, a message asking whether to overwrite it is shown.

2.15. Leaving the program

- There are two ways to quit the program, one is to select Main Menu -> Exit, the other is to click on the close button at the right of the title bar. In both cases, if there is a schematic in view that has not been saved since last changes were made to it, a message is displayed asking the user whether he/she really wants to leave without saving first. If the user selects ‘Yes’, the program is closed and the last changes are lost. If the user selects ‘No’, the message window is closed and the Exit command is aborted.

- The program considers that a modification has been made since the last time the schematic was saved when: 1) a new Component or Connection is made; 2) a Component or Connection is deleted; 3) a Component is moved; 4) A Component or Node is renamed; 5) A change is made to a DataForm.
V. Future Work

There is much work still to be done, in different aspects of OVNI's implementation. Most of it corresponds to other phases of the project, like the development of the NI or the implementation of the second phase of the Interface. There are also tasks that can be done to improve the work we present in this report. Some of these tasks are:

- Build a Component Library: At this point OUI has the functionality to support a library of Components, but only offers three sample and provisional Components. To build a Component Library requires one programmer to follow the instructions we provide in Appendix I. The process is already established and only requires Delphi programming for the implementation of DataForms.

- Use the actual Interface as a shell for Microtran. There are several reasons to do so:
  
  1) The connection of the NI with the Interface will take some time and it is scheduled for the second phase of the project. Meanwhile if the Interface was adapted to Microtran, it would be constantly used and tested. This would provide for an internal testing loop, where bugs could be detected and suggestions generated.

  2) Microtran would benefit from a new interface. At this point, in Microtran the network to be simulated is described using a program called MTDATA. In this program the user makes a written description of the network using the format of old computer punched-cards. OUI would provide for a tool to build a network graphically. To execute Microtran, a program called MT has to be executed and to plot results a program called MTPLOT is used. OUI could be used to launch theses programs, just acting as a shell.

  3) To define a Component Library the first step is to decide which Components to add to the library, what properties to add to the DataForm, etc. Implementing the same set of Components as Microtran provides a basis for these decisions and is close to the set of Components OVNI will eventually support.

  4) Supporting a library of Microtran Components requires to write routines to generate Microtran case files which is one of the formats OVNI plans to support as well
• Improve on actual features, like:

  • 1) Drawing of connections is currently done with the use of the arrow keys. Although it was decided that this method is safer than allowing the use of the mouse, some users might feel strongly against it. If so, mouse support could also be added.

  • 2) Add multiple selection for objects, such that many Components could be moved or deleted at the same time.

  • 3) Every time the image of the schematic is shrunk to show a reduced view of it, like in Print Preview or FullView Window, the resulting image is incomplete. This is particularly notorious with lines, which may show or not depending on their position. This is a common problem in Windows applications, and it is caused at the level of the Win32 API. A solution would be to implement a custom shrinking routine, that could take into account the position of lines.

  • 4) When resizing the main window, the controls on the top panel blink. This is another common problem among Windows applications. There are two possible solutions to it: one is to not show the contents of the window while resizing, and just use a frame rectangle. However this might conflict with similar settings from the program Windows95 Plus. Another solution would be to detect the resizing of the window and make all controls invisible while it lasts.

  • 5) To create new Components and nodes, the program assigns default names based on internal counters, e.g., Resistor1, Resistor2, Resistor3, Capacitor1. If the user deletes one Component or node, the number used by the name of that element is not reused. For example, in the previous sequence if Resistor2 is deleted and a new resistor is created the result is: Resistor1, Resistor3, Resistor4, Capacitor1. Some users might, depending on the circumstances, prefer that number to be reused, while some other users might prefer all names to be reassigned sequentially. There is no unique and definitive way for the program to follow. The decision taken then was not to reuse a number, while eventually, in the future, implement the following: Add to the Main Menu an option to list in a table, by names all Components and nodes. Add to this table some facilities to
reassign and change names, for example, by dragging strings in the columns, supporting multiple selection, etc.

- 6) When a Branch is deleted, such that it breaks a node into two different nodes, the program keeps the same name for both new nodes, and it is up to the user to rename them. This is a problem that needs to be solved. To do so, after deleting the Branch, the program should highlight both new nodes and issue a message on a dialog box requiring the user to rename at least one of the new nodes, and offering an input dialog box with as many input text boxes as new nodes.
VI. Conclusions

The main conclusion derived from this project is that it has been possible to find a common hardware environment to develop and host a real-time network solver and a fully-graphical interface. Dell Pentium-Pro 200MHz machines were chosen to comply with the following requirements:

1. Provide fast execution times for real-time performance. Work done by other members of our research group [3] using these machines, achieved solution step times for 24-valve HVDC converters in the border of 50 μs. This has been possible due to the fast execution times and superscalar pipeline structure of the Pentium Pro processor.

2. Provide for a common platform for the development of OVNI’s User Interface, OVNI’s Network Integrator, and EMTP type models. Dell Pentium Pros are currently in use in our group for all these tasks, and are also used as general purpose workstations. Since the machines are in fact PCs platforms, they can be used for standard PC applications as well as for the development of OVNI. The off-the-shelf nature of the machines has also allowed the use of diverse software development packages, as Delphi, Borland C++, and ADA 95, and is warranty of future software support and upgrades for all development tools used in OVNI’s implementation.

3. Provide common interconnection standards to implement a distributed approach to real-time simulation, using as much as possible off-the-shelf devices. In that sense, the devices and development tools to build a first close loop real-time application of OVNI have been purchased, and the implementation has been recently started.

4. Support Windows NT as the platform for OVNI’s User Interface. The approach followed, which showed to be quite successful, was to start the development of the Interface using Windows 95 while waiting for Windows NT 4.0 to be released, in that way two OS’s with the same look and feel are supported, and the execution of the project was not delayed.

Regarding the implementation of OVNI’s User Interface (OUI), the following are important conclusions:
1. Borland Delphi has shown to be an adequate selection for the implementation of OUI. The RAD (Rapid Application Development) approach of Delphi made possible to write some fifteen thousand lines of code in a period of nine months. One feature of the implementation that speaks well of Delphi as development tool is that most of the problems solved in this first stage of OUI tackled issues that are not of common support by any visual programming environment (neither by Delphi), but Delphi proved to be flexible and useful enough to help solving those issues. We have no doubt that the results shown in this report could have not been achieved in a shorter time by the use of any other visual programming tool, nor could have been done better.

2. At this point the first development stage of OUI has been executed, and the program provides for the basic functionality to either start the second stage, which relates the interface to the Network Integrator, or build a generic interface for other network solving programs. In that respect it is advisable to make use of the code to build a shell for Microtran. The reasons for that are discussed in Chapter V.
VII. Bibliography


APPENDIX I

Guide for adding new Components to the Interface
Guide for adding new Components to the Interface

Adding New Components to OUI is a process of several steps. It involves creating graphical symbols for the new Component, defining terminal information, creating a custom DataForm and making the Component available from the Component Palette. In the future, when the program is capable of saving schematics into different formats it will also involve writing format conversion routines.

As OVNI evolves and new models are developed it may occur that different individuals will be in charge of adding new Components to the interface. To simplify the task, the process has been reduced, as much as possible, to a sequence of easy-to-follow steps. A utility has been written to simplify an otherwise tedious step and the source code contains instructional comments to avoid mistakes. Only one phase, creating a DataForm, involves programming, but this has been reduced as much as possible to the use of existing objects; also the main code is provided as a template, custom objects of common use were developed and instructions are given.

This section provides the guidelines to add a new Component to OUI. The text was written here with two purposes in mind: first, to illustrate how the program was designed to be easily upgraded, and second, to provide a guide for future OVNI developers. For that reason the reader is addressed here as if he/she were to add his/her own Component.

Beware that adding a Component to OVNI also involves adding a new simulation model to the NI: that process is not described here.

1. Getting organized

At some stages of these instructions you will have to provide names for certain properties related to the Component. When that moment arrives it helps to know what are those properties about and to have names ready for them.

For every new Component you have to decide on:
1. An Icon that represents the Component, and on which tab of the Component Palette it will fit. Does it fit any of the existing tabs? Or would it need a new tab category?

2. A symbol that will represent the Component on the schematic. How many terminals are required? Where to place them on the image?

3. A string identifier CompName. This is a short string of maximum 5 characters that is used as a prefix for several objects and properties related to the Component. Some examples are: ‘R’ for resistors, ‘L’ for inductors, or ‘TL’ for transmission lines.

4. A Form identifier FormName. This is the name for the associated DataForm: it is a string of maximum 10 characters. Some examples are ‘RForm’ for resistors, ‘LForm’ for inductors, ‘TLForm’ for Transmission Lines.

2. Creating graphic symbols

Every Component is represented by two graphical symbols. One is an icon image used on the speed buttons of the Component Palette and the Component List, the other is the electrical symbol used on the schematic area.

- Icon symbol: Use Delphi’s Image Editor to create a Bitmap of 20x20 pixels. Use previous images as a guide, you will find them under ‘OVNI\Images\*20x20.bmp’. Save the new image in that same directory.

- Electrical Symbol: Use Delphi’s Image Editor to create a new Bitmap. The size can vary but the height and width are limited to a maximum of 150 pixels, also make sure you choose sizes multiple of 10, this is to make all corners of the image fit on a grid point. Use previous images as a guide, you will find them under ‘...\COMPEDIT\Images\*.bmp’. If possible the image should be based on the IEEE standard representation for that particular Component, but it doesn’t have to be exactly the same. Try to make the symbol easy to recognize, and simple. Do not sacrifice clarity for creativity but try to make a symbol that is pleasant to look at. Also keep the convention of colors for lines and terminals; notice that a terminal square is of 10x10 pixels. The only complication drawing the symbol is that the
terminals cannot be positioned anywhere: terminals must match fixed locations on OUI's grid. Fig. 18 illustrates this problem.

Unfortunately, the Image Editor does not provide a grid feature. To know where the grid points are, keep in mind that the upper left corner of the symbol will be placed on a grid point and from there on a grid point is located every 10 pixels away.

Again, use previous images to help you place the drawing or draw your new symbol on a copy of file: ‘...\COMPEDIT\Images\grid150x150.bmp’. Save the bitmap under; ‘...\COMPEDIT\Images’.

3. COMPEDIT.EXE

3.1. What is COMPEDIT?

All Components in OUI are instances of the same graphical object. What helps the program distinguish one Component from the other is a set of properties stored in a Windows Resource File called Comp.Res. For every new Component the following information has to be linked to the resource file:

1. Component Type (CompName),

2. DataForm Name (FormName),

3. Image,
4. Number of Terminals,
5. Location and name of terminals.

To do this by conventional ways is a very tedious task and prone to cause mistakes. To solve this a utility called COMPEDIT was developed. COMPEDIT is a smart shell that guides the programmer to specify Component information and links that information to file Comp.Res. It is a program aimed for OVNI developers and therefore it is not to be distributed with OVNI.

The resource file is generated using a Resource Compiler from Borland. To interface COMPEDIT and the Resource Compiler a batch file is used. Therefore COMPEDIT, strictly speaking, is a smart editor for the batch file that can also invoke the Resource Compiler. The program adds new entries to the batch file, but it does not edit previous entries, nor creates a new batch file. The program is friendly, though aimed at programmers, and provides constant help through each step. It was developed with Delphi into one single executable file.

3.2. Program description

COMPEDIT.EXE is located in a directory called CompEdit. It has a supporting structure of sub-directories and files that is explained in the file ‘_What is all this.txt’. That file also provides information about how to use the program. Figure 19 is a snapshot of COMPEDIT.

The main features of the program are:

- At all times the right hand panel provides online information as the focus moves through the different input fields. The program uses an original way of implementing the online help without using a Windows Help File, which also allowed the linking of the help information and the code into one single executable file.
- The button labeled Help provides, at any time, general information about using the program.
- Buttons and Edit Boxes are enabled or disabled according to available options.
OVNI's Component Editor

1. Read text on Memo panel and follow the instructions

2. Enter full path location of file COMP.BAT
   D:\Delphi Projects\CompEdit\comp.bat

3. Enter a name for the component: VS

4. Enter name of the Data Form: VSFORM

5. Enter name of BMP file with component image:
   D:\Delphi Projects\CompEdit\Images\WS.bmp

6. Add a Hotspot on top of each terminal

Image File (Image : String)

Requirements:
- It must be the name of the bitmap file that stores the component's schematic symbol to be used in OVNI,
- There is no restriction on the file name length and path,
- The bitmap's maximum Height or Width is 150 pixels.

Example:
- d:\Delphi Projects\OVNI\Images\R.BMP
- d:\Delphi Projects\OVNI\Images\ResistorImage.BMP

Explanation:
For each type of component OVNI displays a schematic symbol using a graphical object. All this symbols are stored in bitmap files in resource file 'COMP.RES'. Therefore this graphical information is packed with OVNI.EXE and there is no need to keep bitmap files in subdirectories.

When the user adds a Component to the schematic, OVNI creates a new instance of the graphical object and copies the corresponding bitmap stated by 'Image'.
• The user must provide a path for a *batch* file called ‘COMP.BAT’. This file is reused for successive versions of OVNI and therefore has to be preserved.

• The actions are organized into steps numbered from 1 to 6.

• Steps 3, 4 and 5 require the user to input names of CompName, FormName, and symbol image file. The image file is the bitmap created in the previous section and saved in: ‘...\COMPEDIT\Images’

• Step 6, ‘Adding HotSpots’, is the biggest advantage that COMPEDIT offers compared with creating the resource file by hand. For OVNI to know the location of each terminal and its name the user must place on top of each terminal a hotspot. With COMPEDIT the hotspots can be placed with the mouse and snapped to the correct grid locations. The online information guides the user through these steps and explains how to choose and input terminal names.

• Once a Component has been defined, the user must press the Button ‘Comp. Done’. The Component can be discarded or saved. If saved, it is added to the batch file.

• Pressing the Button ‘Run Comp.bat’ executes the batch file which in turn updates the resource file ‘COMP.RES’.

### 3.3 Using the program

Launch COMPEDIT and follow, one by one, the instructions the program provides. If you are adding several Components just press the button ‘Run Comp.Bat’ at the end of the session. After quitting the program, make a copy of the new file COMP.RES and place it in the source directory of OUI.

### 4. Updating OUI’s code

Adding a new Component to the Interface requires some changes in OUI’s source code. Using Delphi’s IDE open unit ‘Init.Pas’. This is where OVNI gets from initial information

---

37 See section 2 of this chapter.
about how many Components there are, what are their names, and where to find their DataForms. Every new Component added to OVNI involves some changes in this unit. Scroll down the file and you'll find explanations about what to change and how.

```
{***** STOP HERE! Assign the name of any new DataForm to FormRefAr E.g., if you are adding a DataForm called MyForm:
FormRefArray[next number] := TMyForm;
this will not work if you did not increase the value of MaxNumber0

FormRefArray[0] := TRForm;  {Hard code the form names here.}
FormRefArray[1] := TLForm;
FormRefArray[2] := TCForm;
```

Figure 20. Updating OUI's code, the files contain instructions to follow.

Save your changes. Open unit CompList.Pas, this file implements the Component List Window. Two modifications are required here. Follow the instructions provided at the head of the file. Save your changes.

Finally to add the new Component to the Component Palette, open the unit ‘Main.Pas’. To place a SpeedButton for your new Component follow these steps:

- To select the tab on the Component Palette: With the Object Inspector select the component Notebook1. With the mouse, select the property ActivePage (first in the list). A Combo Box list will become available. From the list select the name of the tab you are looking for.

- To add a SpeedButton for the Component: With the cursor on the Object Inspector press the key <F12>. This will show the MainForm. On the Component Palette, copy a SpeedButton (Click on it and select Edit -> Copy) and place it at the right of all SpeedButtons (Select Edit -> Paste and drag the new button to the correct position). To play with the position a pixel at the time, change the values in properties Top and Left. This will create a new SpeedButton object with the correct event handler.
On the object Inspector change the name of the SpeedButton (name property) to a string of the form: Name := CompName + 'SpeedButton'. For example, for Resistors CompName is ‘R’ and name is ‘RSpeedButton’. For transmission lines if CompName is ‘TL’ Name will be ‘TLSpeedButton’. Now, update the image on the SpeedButton. Click on property ‘Glyph’. An open dialog will appear to select the Bitmap file with the image. This is the image you created in section 2 of this chapter and saved as: ‘OVNI\Images\*20x20.bmp’. Save your changes to Main.Pas, compile and run the program to see your new Component in place.

5. Creating a DataForm

The last step is to create a DataForm for your newly placed Component. This requires programming with Delphi. How complicated or simple the development of a DataForm can be depends on how many fields, models, etc. it has. With Delphi’s IDE, open the file ‘UnitRForm.pas’. This is the unit that implements the DataForm for resistors, ‘TRFORM’. The file was intended as a template to build on top of it more complicated DataForms.

To create a new DataForm based on TRFORM, select the unit ‘UnitRFORM.pas’. Select save as and type in a new name. Rename the form using the Object Inspector and then proceed to change other details in the form. Remember to add the unit to the project, move it to Available-units, and add it to the uses clause of ‘Initial.pas’. In unit ‘Initial.pas’ increase by the value of the constant ‘MaxNumberOfForms’, and in the initialization section hard code the form.

E.g., FormRefArray[2] := TCForm;

By now you will have a new Form that implements all the basic behavior of a DataForm. Your task is to add more components to it to implement the input fields and to extend procedures StoreData and LoadData to handle the information you want to save. Read the code in the form you just copied and follow the instructions on it.
APPENDIX II

Glossary
Glossary

This appendix provides a brief glossary of those terms that we thought needed some explanation. Words written in chapters I to IV, and Appendix I, using *Italic typeface* are contained in this Glossary.

Many definitions in this glossary are taken, in a partial or complete way, from other sources. When that is the case, a reference to the source is made at the end of the definition.

3-D Control

Control that has a three-dimensional (3-D) look. Controls can have either 3-D or 2-D flat appearance.

About Box

*Dialog Box* that displays a Copyright notice, Company and/or authors' names, version number, and other program information. It is generally available from the Help menu.

Ada

Named after Ada Lovelace, ADA is a Pascal-descended language, designed by Jean Ichbiah's team at CII Honeywell in 1979, made mandatory for Department of Defense software projects by the Pentagon. The original language was standardized as "Ada 83", the latest is "Ada 95". Ada is a large, complex, block-structured language aimed primarily at embedded applications. It has facilities for real-time response, concurrency, hardware access and reliable run-time error handling. In support of large-scale software engineering, it emphasizes strong typing, data abstraction and encapsulation. The type system uses name equivalence and includes both subtypes and derived types. Both fixed and floating-point numerical types are supported. Control flow is fully bracketed: if-then-elsif-end if, case-is-when-end case, loop-exit-end loop, goto. Subprogram parameters are in, out, or inout. Variables imported from other packages
may be hidden or directly visible. Operators may be overloaded and so may enumeration literals. There are user-defined exceptions and exception handlers.

An Ada program consists of a set of packages encapsulating data objects and their related operations. A package has a separately compilable body and interface. Ada permits generic packages and subroutines, possibly parametrised.

Ada programming places a heavy emphasis on multitasking. Tasks are synchronized by the rendezvous, in which a task waits for one of its subroutines to be executed by another. The conditional entry makes it possible for a task to test whether an entry is ready. The selective wait waits for either of two entries or waits for a limited time. [17]

**Alpha, DEC**

A family of advanced RISC-based, 64-bit CPUs from Digital Equipment Corporation. The first model introduced in early 1992 was the 150MHz 21064-AA, considered equivalent to a Cray-1 on a single chip.

The current Alpha processor has a clock speed of 600MHz. [15]

**API, Application Program Interface**

The interface (calling conventions) by which an application program accesses operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the kernel (or other privileged utilities) to ensure the portability of the code. [17]

Windows API, is composed of all the calling routines provided by three main DLLs: User, Kernel and GDI. User DLL handles the windowing system and controls. Kernel handles core tasks like memory allocation, task switching and process management. GDI handles drawings and graphics.

The 32-bit version of Windows API is called Win32. It is based on the 16-bit Windows 3.x API. Win32 was originally developed for Windows NT and is mostly the same API for Windows 95. [4]
**Batch file**

A text file containing operating system commands that are executed automatically by the command line interpreter. In Unix, this is called a "shell script" since it is the Unix shell which includes the command line interpreter. Batch files can be used as a simple way to combine existing commands into new commands. "autoexec.bat" is the best known example of an MS-DOS batch file. [17]

**Bitmap**

Approach for graphic file formats, in which the image is stored as a collection of dots (pixels) the same way they would be displayed on screen or a printer. In contrast Drawing file formats, store information about how the image is created rather than how it looks. Bitmaps files are larger files than Drawing files, and are resolution dependent. Some popular Bitmap formats are TIFF, PCX and BMP. Windows make extensive use of the BMP format, and provides a Bitmap graphic object that can create, store or manipulate (paint, scroll, rotate, scale) an image in a disk file. [8]

**Borland C++**

An ANSI C and C++ compiler from Borland for DOS and Windows applications. It is turbo C-compatible and its debugger supports Windows programs written in Microsoft C. It includes application frameworks for Windows (Object Windows) and DOS (Turbo Vision). [15]

**Borland's Resource Compiler**

A resource compiler is development tool used to create Resource files. Delphi provides two 32-bit resource compilers that a programmer can use to translate .rc script files to .res binary resource files. These programs are: brcc32.exe, which creates a resource file, and brc32.exe which creates and binds a resource file to an executable code file. [4]
Buttons

*Graphic object* that represents an electrical *push button*. Generally buttons are rectangular boxes with a label that specifies what the button does. The user selects a button either by clicking on it with the mouse, or with the keyboard, using either mnemonic access characters (such as Alt-H for Help) or keyboard shortcuts (such as X for Cancel). Windows buttons can also be selected using the <Tab> or <Shift-Tab> keys to move the input *focus* to the button and then press the space bar to "click" it. 3-D Buttons appear as if they are pressed and released when the user selects or deselects them. [8]

Cancel path

Is a sequence of user commands to cancel the last action done. Usually the <Esc> key and the right mouse button are used to cancel any action that requires a sequence of steps and that has not been completed yet. A Cancel path can also be provided by a Cancel *Button*, or any other input Control. The term Cancel path is very similar to Escape path, and the only difference between them comes from those situations when a Cancel option implements an undo type of command.

Check Boxes

Square shaped *graphic object* used in groups to present the user with several options from which more than one can be selected at the same time. Check boxes control individual choices that can be turned on or off. When off, the option box is blank; when on the option box has an X in it. The user can select more than one choice or no choices at all. [8]

CISC, Complex Instruction Set Computer

A processor where each instruction can perform several low-level operations such as memory access, arithmetic operations or address calculations. The term was coined in contrast to Reduced Instruction Set Computer.

Before the first *RISC* processors were designed, many computer architects were trying to bridge the "semantic gap" - to design instruction sets to support high-level languages by providing "high-level" instructions such as procedure call and return, loop instructions such as
"decrement and branch if non-zero" and complex addressing modes to allow data structure and array accesses to be compiled into single instructions.

While these architectures achieved their aim of allowing high-level language constructs to be expressed in fewer instructions, it was observed that they did not always result in improved performance. For example, on one processor it was discovered that it was possible to improve the performance by NOT using the procedure call instruction but using a sequence of simpler instructions instead. Furthermore, the more complex the instruction set, the greater the overhead of decoding an instruction, both in execution time and silicon area. This is particularly true for processors which used microcode to decode the (macro) instruction. It is easier to debug a complex instruction set implemented in microcode than one whose decoding is "hard-wired" in silicon.

Examples of CISC processors are the Motorola 680x0 family and the Intel 80186 through Intel 486 and Pentium. [17]

Clarion

A fourth generation language for MS-DOS.

Client-Server

A common form of distributed system in which software is split between server tasks and client tasks. A client sends requests to a server, according to some protocol, asking for information or action, and the server responds. There may be either one centralized server or several distributed ones. This model allows clients and servers to be placed independently on nodes in a network, possibly on different hardware and operating systems appropriate to their function, e.g. fast server/cheap client. [17]

Combo Box

A graphic object, that combines an editable text field and a list box. Combo boxes are useful when the application requires user input and can display a list of possible responses. The user can type a response in the text box if the correct one is not in the list. There are two types of Combo Boxes: standard and drop down according to which type of list they include. Standard
list boxes display all the text at once, while drop down list boxes only show one item at the time but when the user selects the list it extends to show several options at the same time. [8]

Components

Components are the building blocks of Delphi Applications. Although most components represent visible parts of a user interface, components can also represent non-visual elements in a program, such as timers and databases.

Controls

All the visible objects through which the user interacts with the application, e.g., Buttons, Radio buttons, Speed Buttons, Check Boxes, List Boxes, Text Boxes, Group Boxes, Combo Boxes, Scroll Bars, etc. In Delphi programming the term Control has a more precise meaning. Controls are objects, descending from the class TControl, these objects have the capability to display themselves and enable interaction with the user at design time.

Cursor

A small bitmap used as a pointer on the screen and that can be moved with the mouse. The mouse cursor can have different shapes (bitmaps) depending on the action that is performed. For example, the default cursor shape, an arrow pointing up, can be changed in to an Hour Glass cursor, small glass clock, to indicate that an application is busy and cannot respond to user commands.

Data type

In programming languages, every variable has a type. A variable type circumscribes the set of values it can have and the operations that can be performed on it. For example, Integer, Real or String are data types. [11]

dBASE

A DBMS from Ashton-Tate Corporation and the language used by it. The first release was dBASE II, ca 1980. There never was a ‘dBASE I’. Later versions: dBASE III, dBASE III+, and dBASE IV. Visual dBASE is a graphical development environment. [17]
**DBMS, Database management system**

A suite of programs which typically manage large structured sets of persistent data, offering ad hoc query facilities to many users. They are widely used in business applications. Commercial examples include Ingres, Oracle and Sybase.

A database management system (DBMS) is an extremely complex set of software programs that controls the organization, storage and retrieval of data (fields, records and files) in a database. It also controls the security and integrity of the database. The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data. [17]

**Delphi’s Component Palette**

All Delphi components from the VCL (Visual Component Library) are available from the Component palette. The component palette is a tabbed palette that contains icons representing Delphi components, that the user can drop onto a form. The Component palette provides eleven groups of objects and it is possible for the user to add to the palette its own custom objects.

**Delphi’s IDE**

Delphi’s development environment is called ‘Delphi’s IDE’ (for Integrated Development Environment). The Delphi IDE is divided into four main portions: the Code Editor, the form designer, the Object Inspector and the main window.

The main window is the control center for the Delphi IDE. It provides all the standard functionality of the main window of any other Windows program. It consist mainly of three parts: the main menu, the Speed Bar, and the Component palette. The Form designer begins as an empty window ready for the user to turn in into a Windows application. On the Form designer the user can place components from the Component palette. Once a Component is placed on the Form the user can control the component appearance either by using the Object Inspector or the Code Editor.

The Object Inspector provides a list of all properties for every component that is placed on the form designer. Changing settings in the list of properties, the user can modify the appearance
and behavior of the component in run time. The code editor is a text editor where the user and the Code Generator write the code that dictates how the program behaves. [7]

**Dialog boxes**

Graphical objects. Usually small windows or forms used to provide feedback or solicit input. Dialog boxes can contain any type of controls, like buttons, labels or list boxes. There are two main types of dialog boxes: non-modal and modal. Non-modal Dialog Boxes merely inform or await for input. A tool bar is an example of a non-modal Dialog Box. Modal Dialog Boxes are used when the application cannot proceed without advice from the user. For example an 'out or paper' message when printing. [8]

**Drag**

See **Drag and drop**.

**Drag and drop**

The action of moving **graphical objects** with the mouse. To drag an object the user clicks on it with the mouse and keeping the mouse button pressed, moves the control on the screen. As soon as the mouse button is released the dragging stops and the control is dropped in its last screen location. Drag and drop can be used to move files in a file manager program, or to move images in an image editor.

**Dynamic Data Exchange, DDE**

In Microsoft Windows, dynamic data exchange (DDE) is a form of inter-process communications that uses shared memory to exchange data between applications. Applications can use DDE for one-time data transfers and for ongoing exchanges and updating of data. [18]

In the DDE protocol a conversation is established between a client and server application. Once the conversation has been established the transfer of data can occur automatically or one application can request data from the other. Despite the roles of Client and Server, the data can flow in any direction.
**Edit Box**

*Graphic object* used to read or write a single line of text. To input text in an edit box the user must select the edit box first and then type-in the text using the keyboard. An edit box can be set to be read only, in which case keyboard input has no effect on it.

**Error Messages**

Messages that are displayed when errors occur. Usually error messages are displayed in Modal *Message Dialog Boxes*. Errors can be caused by user mistakes or by problems in the execution of a program.

**Escape path**

See *Cancel path*.

**Event handler**

Delphi programming term. Events represent user actions (or internal system occurrences) that a program can recognize, such as a mouse click. In Delphi, almost all the code is executed in response to such an event. The code that specifies how a component should respond to an event is called an event handler.

Event handlers are really specialized procedures. Procedures and functions, also known as subroutines or sub-programs, generally constitute the bulk of a program's code in Delphi applications. [14]

**Fast-keys**

Or shortcut keys, are keystrokes that let the user directly access a function even if that function would otherwise require burrowing through several levels of *menus*. [8]

**File search filter**

Is a property of Open, Save and Save As Dialog Boxes. The file filter is a string field that can be used to filter the type of files the Dialog Box displays. For example if in an Open Dialog Box, the user sets the file filter to .BMP then the Dialog Box will only shown files with that extension.
**Floating form or window**

Delphi programming term. A floating form is a small form that is not the main window of an application but shows some information that is useful to keep in view. A floating form can be moved, closed and re-opened.

**Floating pop up menu**

Graphic Component. A floating Pop Up Menu is a menu that can appear at any location on the screen. *Floating Pop Up Menus* are generally displayed when the user clicks the right mouse button of the mouse. Floating Pop Up Menus are convenient to provide a set of options related to one object in particular (the object been clicked) or to provide for an alternative way to access commands of common use.

**Fly-over hints and banners**

In Windows95 fly-over hint are shown when the mouse is placed on top of certain objects. The hint displayed is a rectangular small size banner with a hint text. Hints are shown when the mouse cursor enters a Control, and are displayed for a fixed time, after which they disappear. Fly over hints are particularly useful to identify *speed buttons*.

**Focus**

The component or window to be active in a running application is said to have ‘focus’. Any keyboard input the user enters is directed to that component or window.

**Forms**

Delphi programming term. Forms is the name given in Delphi to the object that is used to implement the window frame of applications. Delphi applications are designed putting components on a form. Forms can be used as windows, dialog boxes or simply as forms, such as data-entry forms.

**Free-form exchange format**

General EMTP format aimed to facilitate the exchange of files between different EMTP versions. The free-form exchange format is basically the EMTP data case format without the
constrain of providing data at fixed column positions. It has so far been implemented in Microtran.

**Full-duplex**

A term used to describe a communications channel down which data can travel in both directions at once. The opposite of full-duplex is simplex. [17]

**Graphic Object**

A graphic object is an object that can be seen on screen, e.g., a button or a bitmap. In Delphi programming visual controls are descendants of a common parent class (TControl), however there is a distinction between those controls that can receive user focus and those that cannot. The controls that do not receive user focus, are called Graphic Controls.

**Graphical User Interface, GUI**

The use of pictures rather than just words to represent the input and output of a program. A program with a GUI runs under some windowing system (e.g. The X Window System, Microsoft Windows, Acorn RISC OS, NEXTSTEP). The program displays certain icons, buttons, dialogue boxes etc. in its windows on the screen and the user controls it mainly by moving a pointer on the screen (typically controlled by a mouse) and selecting certain objects by pressing buttons on the mouse while the pointer is pointing at them. The concept of GUI originated in the early 1970s at Xerox's PARC laboratory. [17]

**Group box**

Graphic object. A group box is a rectangular one-pixel frame with a label in its upper-left corner. Group boxes are technically considered controls, but they really do not process any mouse or keyboard input, they simply group together controls that are logically related.

**GUI**

See Graphical User Interface.
**Handles**

Are 16-bit numbers issued by Windows that refer to individual window instances. This means that in the Windows environment, every window has a unique handle. Many Windows API functions require a handle to know on which window to perform the function. [7]

**Help button**

*Button* that, when selected, displays help information. The information can be displayed in a dialog box, or from a Windows Help file.

**Hints**

*See fly-over hints*

**Hotspot**

On-screen areas or objects that are sensitive to *mouse events*. In Delphi programming most objects from Delphi’s Visual Component Library (VCL) provide detection of mouse events in all the area they occupy. The term hotspot is therefore not applied to the VCL, and is rather used for custom implementations of mouse sensitive areas.

**Hour-glass icon**

A type of *cursor* shape that is used to indicate the user to wait while the machine performs a tasks.

**Icons**

A graphic object. A small picture on the screen which represents something. Files and programs have icons, and open when the user clicks twice on the icon. Moving and copying files is done by *dragging* their icons to the desired location. A file is deleted by moving its icon to the picture of a trash can. In Windows icons are stored as bitmaps. [17]
**Image Editor**

The Image editor is Delphi's visual editor. The Image Editor can create and edit binary resources stored in .res and .dcr files, as well as can create and edit *bitmap* (.bmp), *icon* (.ico) and *cursor* (.cur) files.

**Incremental-alphabetic search**

Also called telescoping search. It is used in list boxes to help finding a particular item. In an incremental-alphabetic search, as the user types a word in an *edit box* the partial text is used to locate the item in the list box. As the user types the letters the program scrolls the list box to the item which initial letters are the closest match to the letters the user has typed.

**Instantiate**

To create an instantiation. Producing a more defined version of some object by replacing variables with values (or other variables). In object-oriented programming, producing a particular object from its class template. This involves allocation of a structure with the types specified by the template, and initialization of instance variables with either default values or those provided by the class's constructor function. [17]

**Intel x86**

Name used to identify the family of Intel's processors that includes Intel’s 8086, 8088, 80186, 80286, 80386, 80486, Pentium, *Pentium Pro*, Pentium II. Several processors of the Intel x86 family have been widely used in Personal Computers, and the term x86 family has been informally used to identify Intel processors from other processors in the PC market, like the *Power PC* and *Alpha.*

**InterBase**

A commercial relational *DBMS* from Borland, designed to handle online complex processing. It can be a peer-to-peer or *client-server* system and uses *SQL* plus its own data manipulation language. [15]
Landscape

A way of positioning a page so it is wider than it is high, as in landscape painting. A computer printer usually gives the user a choice between landscape and portrait orientations. [17]

List Boxes

Graphic object. A list box is a control window that contains a list of items from which the user can choose. [18]

Menu

A graphic object. A list from which the user may select an operation to be performed. This is often done with a mouse or other pointing device under Graphical User Interface but may also be controlled from the keyboard. Menus are very convenient for beginners because they show what commands are available and make experimenting with a new program easy, often reducing the need for user documentation. Experienced users however, usually prefer keyboard commands, especially for frequently user operations, because they are faster to use. In situations such as text entry where the keyboard must be used anyway, having to move your hand to the mouse to invoke a menu operation is slow. [17]

Menu object

Delphi programming term. Delphi’s Visual Component Library provides two types of Menu objects to implement menus. The MainMenu component encapsulates a menu bar and its accompanying drop-down menus for a form. The Pop Up Menu component provides a menu that appears independently of any main menu in the form, typically when the user right-clicks.

Message Dialog Boxes

Message Dialog Boxes are Dialog Boxes that are used to display messages to the user. Message dialogs are used for confirmation, warning and information. Message Dialog Boxes are so common that some GUI design environments have them as a predefined, built-in function. Delphi provides five built-in dialog boxes. [8]
**Messaging system**

In Windows, the messaging system is the mechanism in charge of sending messages. A message is a notification of some occurrence sent by Windows to an application. Clicking a mouse button, resizing a window, or pressing a key on the keyboard, for example, cause Windows to send a message to an application notifying the program of what occurred. Delphi encapsulates some Windows messages in its event system, such that the user only has to write event handlers and does not need to deal directly with the detection and identification of the messages. [7]

**Methods**

Term used in Object Pascal programming. Methods are procedures or functions built into a component. Component users use methods to direct a component to perform a specific action or return a certain value not covered by a property. Methods are also useful for updating several related properties with a single call. Because they require execution of code, methods are only available at run time. [11]

**MicroTran**

The University of British Columbia version of the EMTP. It has been developed and maintained by the research and effort of Dr. H.W. Dommel, Dr. J.R. Martí and Dr. I Martí.

**Mouse event**

Term related to Delphi programming. Whenever the user makes use of the mouse he/she generates mouse events. Mouse events can be either: OnMouseDown, when a mouse button is pressed; OnMouseMove, when the mouse is moved; and OnMouseUp when a mouse button is released. These events can be used to trigger an event handler as response to the user's action.

**OBDC, Open DataBase Connectivity**

A standard for accessing different database systems. There are interfaces for Visual Basic, Visual C++, SQL and the ODBC driver pack contains drivers for the Access, Paradox, dBase,
Text, Excel and Btrieve databases. An application can submit statements to ODBC using the ODBC flavor of SQL. ODBC then translates these to whatever flavor the database understands.

ODBC 1.0 was released in September 1992. ODBC is based on Call-Level Interface and was defined by the SQL Access Group. Microsoft was one member of the group and was the first company to release a commercial product based on its work (under Microsoft Windows) but ODBC is not a Microsoft standard (as many people believe). ODBC drivers and development tools are available now for Microsoft Windows, Unix, OS/2, and Macintosh. [17]

**Object Inspector**

Part of *Delphi's IDE*. The Object Inspector is a window that shows all the properties and events for one or more selected components or forms. The properties and events displayed in the Object Inspector reflect whichever *form* or component has *focus* in the *form* designer, unless several components are selected with the mouse, then only the properties and events that are common to all selected components are displayed in the Object Inspector. [5]

**Object-oriented design, ODD**

A design method in which a system is modeled as a collection of cooperating objects and individual objects are treated as instances of a class within a class hierarchy. Four stages can be identified: identify the classes and objects, identify their semantics, identify their relationships and specify class and object interfaces and implementation. object-oriented design is one of the stages of object-oriented programming.


**Object oriented tool**

Programming tool for Object Oriented Programming (OOP). Although the term Object Oriented tool is widely applied to different programming tools that handle objects, there is an important distinction between Object Oriented and Object based tools that is usually not made. An Object Based tool, makes use of an existing set of object by creating instances that the user can manipulate, but does not allow for modifications to those objects, nor to derive new
objects from them. An Object Oriented tool, provides for a library of objects, but also allows the programmer to create new objects and classes. For example, Borland Delphi is an Object Oriented tool, Visual Basic is an Object based tool.

Strictly speaking a fully Object Oriented implements the following concepts of OOP: 1) Encapsulation. Deals with combining related data fields and hiding the implementation details, this offer advantages in modularity, and code isolation. 2) Inheritance, which is the capability to create new objects that maintain the properties and behavior of ancestor objects. 3) Polymorphism, which allows to make calls to the same method of an object passing different instances in the variable.

**Object Pascal**

Object Pascal is a new and improved, object-oriented version of Borland’s Pascal that was used to implement Borland’s Delphi Development Tool. Object Pascal introduced into Pascal features like exception handling, object oriented programming, and runtime information. The Object Oriented features implement encapsulation, inheritance and polymorphism, but in contrast with C++ it does not support multiple inheritance (deriving an object from two separate objects).

**Objects**

In object-oriented programming, a self-contained module of data and its associated processing. Objects are software building blocks.

**Online help**

On-screen Help that is available immediately. In Windows, Online Help is achieved by the use of Windows Help Files. Online help can be displayed either when the user selects a Help option from a button or menu bar, or by pressing the <F1> key to get context sensitive help related to the last selected control.

**Open Database Connectivity**

A standard for accessing different database systems. There are interfaces for Visual Basic, Visual C++, SQL and the ODBC driver pack contains drivers for the Access, Paradox, dBase,
Text, Excel and Btrieve databases. An application can submit statements to ODBC using the ODBC flavor of SQL. ODBC then translates these to whatever flavor the database understands. ODBC 1.0 was released in September 1992.

ODBC is based on Call-Level Interface and was defined by the SQL Access Group. Microsoft was one member of the group and was the first company to release a commercial product based on its work (under Microsoft Windows) but ODBC is not a Microsoft standard (as many people believe). ODBC drivers and development tools are available now for Microsoft Windows, Unix, OS/2, and Macintosh. [17]

**OS, Operating system**

The low-level software which schedules tasks, allocates storage, handles the interface to peripheral hardware and presents a default interface to the user when no application program is running. The OS may be split into a kernel which is always present and various system programs which use facilities provided by the kernel to perform higher-level house-keeping tasks, often acting as servers in a client-server relationship. Some would include a graphical user interface and window system as part of the OS, others would not.

The facilities an operating system provides and its general design philosophy exert an extremely strong influence on programming style and on the technical cultures that grow up around the machines on which it runs.

Example operating systems include 386BSD, AIX, AOS, Amoeba, Angel, Artemis microkernel, Brazil, COS, CP/M, CTSS, Chorus, DACNOS, DOSEXEC 2, GCOS, GEORGE 3, GEOS, ITS, KAOS, LynxOS, MPV, MS-DOS, MVS, Mach, Macintosh operating system, MINIX, Multics, Multipop-68, Novell NetWare, OS-9, OS/2, Pick, Plan 9, QNX, RISC OS, STING, System V, System/360, TOPS-10, TOPS-20, TRUSIX, TWENEX, TYMCOM-X, Thoth, Unix, VM/CMS, VMS, VRTX, VSTa, VxWorks, WAITS, Windows 95, Windows NT. [17]

**Over-range access**

Error that occurs when a program data pointer exceeds the limits of a variable.
**Palette**

A color palette is the range of colors available to use within a program such as a paint or draw program, photo editing program, or page layout program.

A the tool palette is the collection of tools available to use, such as a paintbrush, charcoal, pencil, paint bucket, airbrush, etc. Palette tools are usually implemented with *Speed Buttons*, and can be placed either on the application window, or a floating window. A convenient approach when a palette has too many *buttons* is to used *tabbed palettes*, where the *buttons* are grouped into several *tabbed pages*. [15]

**Paradox**

Borland's IBM PC relational database. Paradox 7 runs under Windows 95 and Windows NT. Paradox 5 runs on Microsoft Windows and provides a graphical environment, a debugger, a data modeling tool, and many "ObjectPAL" commands. [15]

**Pascal-type-of string**

Strings in Delphi’s Object Pascal, have the following structure: an initial byte represents the number of characters in the string. Following this length byte are from 0 to 255 characters, for a maximum possible size of 256 bytes. This type of string differs from C’s the null-terminated string, which is simply an array of characters that ends with a byte equal to zero. Delphi methods and object fully support the Pascal-type of string, however the Windows *API* calls use null terminated strings. [5]

**PCI, Peripheral Component Interconnect**

A self-configuring personal computer local bus designed by Intel, released around Autumn 1993 and supported by most major manufacturers including Apple Computer. It is technically far superior to VESA’s local bus and runs at 33 MHz. It is currently mostly used on Pentium/Pentium Pro based computers and on some DEC Alphas and PowerPCs but is processor independent and so can work with other processor architectures as well. [17]
**Pentium-Pro**

Processor known as "P6" during development. It is Intel's successor to the Pentium processor. The P6 has an internal *RISC* architecture with a *CISC-RISC* translator, 3-way super-scalar execution, and out-of-order execution (or "speculative execution", which Intel calls "Dynamic Execution"). It also features branch prediction and register renaming, and is super-pipelined (14 stages).

The P6 is made as a two-chip assembly: the first chip is the CPU and 16 kilobyte first-level cache (5.5 million transistors) and the other is a 256 (or 512) kilobyte second-level cache (15 million transistors). The first version has a clock frequency of 133 MHz and consumes about 20W of power. It is about twice as fast as the 100 MHz Pentium. The original 0.35 micron versions of the Pentium Pro released on 01 Nov 1995 run at 150 and 166 MHz for desktop machines and up to 200 MHz for servers. Heat dissipation is about 20 Watts.

The Pentium Pro is optimized for 32-bit software and runs 16-bit software slower than the original Pentium.

[17]

**Pop down menu**

A pop down menu is the window that appears when the user selects an item from a menu bar.

**Pop Up Menus**

See *Floating pop up menu*.

**PowerBuilder**

A graphical user interface development tool from Powersoft for developing *client-server* database applications. It runs under MS-DOS(?) and Microsoft Windows. There are also versions for Microsoft Windows, Windows NT, Macintosh, and Unix.

Applications can be built by creating windows, controls (such as listboxes and *buttons*), and menus within the PowerBuilder development environment. The language used to program PowerBuilder, PowerScript, is loosely based on BASIC.
PowerBuilder supports programming on many database backends including Sybase and Oracle. It also has added support for ODBC database drivers. PowerBuilder also comes with a built-in database backend (WATCOM SQL 32-bit relational database).

**PowerPC**

A RISC microprocessor designed to meet a standard which was jointly designed by Motorola, IBM, and Apple Computer (the PowerPC Alliance). The PowerPC standard specifies a common instruction set architecture (ISA), allowing anyone to design and fabricate PowerPC processors, which will run the same code. The PowerPC architecture is based on the IBM POWER architecture, used in IBM's RS/6000 workstations. The PowerPC standard specifies both 32-bit and 64-bit data paths. Early implementations will be 32 bit; future higher-performance implementations will be 64 bit (e.g. PowerPC 601). A PowerPC has 32 integer registers (32- or 64 bit) and 32 floating-point (IEEE standard 64 bit) floating-point registers.

**Print Dialog Box**

The Print Dialog Box is a dialog box that permits the user to select which printer to print to, which pages to print, how many copies to print, and if the print job should be collated. If the user chooses the Setup button in the Print dialog box, the Print Setup dialog appears.

**Push Button**

A graphic object. Type of Button that behaves like an electrical push button. The default position of the button is non-pressed, and only when it is selected by the user is changes into the pressed position. As soon as the user releases the button, it returns to the default position. Examples of push buttons are ‘OK’, ‘Cancel’ and ‘Help’ buttons.

**Radio Buttons**

Type of buttons generally used in groups to present different possibilities to the user. Typically each radio button represents a single choice in a set of mutually exclusive choices. For example in any group of radio buttons, the user can only select one at the time. Radio
buttons are buttons represented by circles. If the radio button’s circle is filled, the option associated with that radio button is selected; if empty, the option is deselected. [8]

Rapid Application Development, RAD

An approach to systems development that includes automated design and development tools and joint application development. RAD is associated with a wide range of approaches to software development: from hacking away in a GUI builder with little in the way of analysis and design to complete methodologies expanding on an information engineering framework. Some of the current RAD techniques are: CASE tools, iterative life-cycles, prototyping, workshops, SWAT teams, timebox development, and Re-use of applications, templates and code. [17]

Real-Time Operating System

Any operating system where interrupts are guaranteed to be handled within a certain specified maximum time, thereby making it suitable for control of hardware in embedded systems and other time-critical applications. [17]

Resource Compiler

See Borland’s Resource Compiler.

Resource File

Resource Files, are files that contain resources. A resource is binary data that a resource compiler or developer adds to an application's executable file, or a .dll file. A resource can be either standard or defined. The data in a standard resource describes an icon, cursor, menu, dialog box, bitmap, enhanced metafile, font, accelerator table, message-table entry, string-table entry, or version. An application-defined resource, also called a custom resource, contains any data required by a specific application. [18]
**RISC, Reduced Instruction Set Computer**

A processor whose design is based on the rapid execution of a sequence of simple instructions rather than on the provision of a large variety of complex instructions (as in a Complex Instruction Set Computer).

Features which are generally found in RISC designs are uniform instruction encoding (e.g. the op-code is always in the same bit positions in each instruction which is always one word long), which allows faster decoding; a homogenous register set, allowing any register to be used in any context and simplifying compiler design; and simple addressing modes with more complex modes replaced by sequences of simple arithmetic instructions.

Examples of (more or less) RISC processors are the Berkeley RISC, HP-PA, Clipper, i960, AMD 29000, MIPS R2000 and DEC Alpha. IBM's first RISC computer was the RT/PC (IBM 801), they now produce the RISC-based RISC System/6000 and SP/2 lines. [17]

**Scroll, scroll bars and scroll box**

A window in an application written for a *windowing environment* can display a data object, such as a document or a *bitmap*, that is larger than the window's client area. When provided with a scroll bar, the user can scroll a data object in the client area to bring into view the portions of the object that extend beyond the borders of the window. In Delphi programming a distinction is made between scroll bars and scroll boxes. Scroll bars are a single sliding scroll control that can be used to relate the scroll position with a variable. Scroll Boxes are graphical containers on which other components can be placed and be panned in run time.

**Scroll event**

Term related to Delphi programming. When the user slides a scroll bar (either from a scroll bar or scroll box object) a scroll event is generated, this event can be used to trigger an event handler which executes some response code to the user's action.

**Smalltalk**

An operating system, and object-oriented programming language that was developed at Xerox Corporation’s Palo Alto Research Center. As an integrated environment, it eliminates the
distinction between programming language and operating system. It also allows the programmer to customize the user interface and behavior of the system. Smalltalk was the first object-oriented programming language. Until recently it was only implemented on Xerox workstations, but a version of Smalltalk-80 (latest version) is now available for Sun workstations, and PC versions are also available. [15]

**Speed Buttons**

Delphi Graphical component. The *Speed* component provides a means to group several buttons so that their shift state at run time is interrelated. For this reason, speed buttons are often used to create tool bars. Compare to Radio buttons and *Check boxes*. Speed buttons can display a glyph, similar to the *Bitmap Button* component, but unlike the *bitmap* button they cannot display a caption. Speed button glyphs must be Windows *bitmap* (.BMP) files. [13]

**SQL, Structured Query Language**

Pronounced "see qwill", a language used to interrogate and process data in a relational database. SQL commands can be used to interactively work with a database or can be embedded within a programming language to interface to a database. [15]

**SQLWindows**

A package used to graphically develop MS-Windows *client-server* applications. Sold by Gupta Corporation. It is used to write Windows applications that access SQL databases in *client-server* environments. [17]

**State Estimator**

Device used to determine the state of a power network (voltages, currents, reactive power etc.) from data read from the actual network.

**Status Bar**

A graphical component. The Status Bar component is a window, typically horizontal and aligned along the bottom edge of an application frame, that displays status information about
the application. Status bars can display either Simple Text information in a single Simple Panel or they can display various types of information in multiple panels each with its own Text. [19]

**Sub-menu**

A menu item that is attached to another Menu item. For example, choosing a Menu option File|Reopen file can open a nested sub-menu with recent file names to chose from.

**<Tab> Key**

In Windows, the tab key is used to switch the *focus* from one control to another. As the tab key is pressed the *focus* moves up, down, left or right at the designers discretion. Pressing Shift-Tab moves the *focus* in the reverse order. [8]

**Tabbed pages**

Tabbed pages are a set of pages owned by a tab notebook component. Each page has its own set of controls.

The user selects a page by clicking the page's tab that appears at the top of the control. Tabbed pages are very convenient to implement *Tabbed palettes*, or tabbed access to several files that are opened at the same time.

**Tabbed palette**

Graphical component. A Tabbed palette is a tool bar palette which objects are distributed among several tabbed pages. To select a particular option, the user must first select the corresponding tab, to gain access to all options in that tab, and then select the option. Tabbed palettes are a very convenient way of implementing tool bars that have big numbers of elements, or that could host an increasing number of elements.

**Toggle button**

Graphical component. Type of button that behaves like an electrical two-position toggle button. The two possible positions are pressed and non-pressed. Whenever the user selects the
button it switches from one position to another. Examples of toggle buttons are some Speed Buttons in tool bars, like a pencil or brush selection button in an image editor.

**Visual BASIC**

A popular event-driven visual programming system for Microsoft Windows, in which fragments of BASIC code are invoked when the user performs certain operations on graphical objects on-screen. Widely used for in-house application program development by users and for prototyping.

**Visual C++**

A C and C++ development system for DOS and Windows applications from Microsoft.

**VMEbus**

VersaModule Eurocard bus. A widely accepted backplane interconnection bus system developed by a consortium of companies led by Motorola, now standardized as IEEE 1014. VMEbus is a 32-bit bus, VME64 is an expanded version that provides 64-bit data transfers.

**Warning messages**

Messages that are displayed to warn the user of a situation. Warning messages are less critical than Error Messages.

**WIN32 API**

The 32-bit version of Windows API. For more details refer to API in this glossary.

**Windowing Environment**

An operating system such as Macintosh, Motif, OS/2, RISC OS, Windows, or X Windows that uses windows to display information.

**Windowing objects**

Graphic controls used in a windowing environment, e.g., buttons, icons, windows, cursors, etc.
**Windows or forms**

Windows are the central element of Windowing environments. Forms are the object used in Delphi to implement windows. There are four main types of windows: 1) Application windows, which are the basic visual framework for data and commands in an application; 2) Multiple Document Interface (MDI) windows, this type of window allows the application to handle more than document at the time; 3) Help Windows, these are independent application windows and can be moved outside the primary application window; 4) Dialog Boxes, used by an application to show messages and require input from the user. [8]

**Windows Control Panel**

In Windows95 and Windows NT 4.0, the Control Panel is a set of applications that let the users configure the environment of their computers. The Control Panel applications are available from the Settings item in the Task Bar. One application in particular, Display, gives control over the appearance of the User Interface. Technically speaking, Control Panel applications are special-purpose dynamic-link libraries (DLLs) that export a particular type of function (CPInit), which Windows calls. Control Panel applications are identified by the extension .cpl. [4][17]

**Windows Help File**

Whenever Windows shows help information, it launches a separate program called Winhelp.exe. This program is on every Windows installation. The version of Winhelp that is used by Windows95 is a greatly enhanced 32-bit version of the previous 16-bit Winhelp.exe. Winhelp reads information from Windows .Hlp files, and therefore each application willing to provide for online help must provide for its own .Hlp file.

There are two ways to create a help file, a hard way and a an easy way. The hard way consists on embedding text commands into a Microsoft Word document, converting that document to rich-text-format, and compiling it with Microsoft’s HCW help compiler. The easy way, consists on using third-party tools that are available for developing help files, such as Forehelp, ForeFront, or RoboHelp. [4]
**XOR mode**

Three factors determine how Windows draws pixels or lines to a canvas surface: the pen’s color, the surface or destination color, and the bitwise operation that Windows performs on the two color values. This operation is known as raster operation (ROP). Raster operations involve a logical operation of the colors of the target pixel and the pixel to be drawn. One of such operation is exclusive or (XOR). This is a very useful operation for animation and drawing, because the result of a XOR operation can be reverted by performing the same operation again. For example, if the screen color is black and a XOR operation is performed with a white pixel, the result is a white pixel. If the XOR operation of background color with white is repeated again, the background turns back into black color.