Comparative Analysis of Porting the SIMARC Package to Windows 95 and XWindow

María Teresa Güereña Sosa

Electrical and Computer Engineering

APPROVED:

Dr. Glenn A. Gibson, Chair

Dr. Sergio D. Cabrera

Dr. Ann Q. Gates

Associate Vice President
for Graduate Studies

To my parents and sister,
for all the joy and all the good times.

Comparative Analysis of Porting the SIMARC Package to Windows 95 and XWindow

by

María Teresa Güereña Sosa, B.S.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Electrical and Computer Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

May 1999

Acknowledgements

I want to express my deepest gratitude to Dr. Glenn A. Gibson for the guidance, support and encouragement he has given me over the course of my graduate studies. I also want to thank Dr. Sergio D. Cabrera for his comments and support since my arrival to El Paso. My thanks and appreciation to Dr. Ann Q. Gates, for being not only a good teacher, but also a source of motivation.

Special thanks go to Alejandro E. Brito, for being my "partner in crime" since the very beginning of this journey to the graduate-studies-land, and also to Isabel Vallejo, her good will and great coffee.

Life has blessed me with the friendship of some exquisite people, including my own family. Work is lighter knowing you are out there taking care of me in different ways, and late night music always brings you closer to me.

This work was supported in part by the National Science Foundation under Grant No. CCR-9522810. I also want to acknowledge the work and interaction with the ITESM-MCAP group.

El Paso, Texas
December 13, 1998

Abstract

A migration of the DOS simulation tool SIMARC to two different platforms is presented. The target platforms are Windows 95 and X Window. The basis for the work is the analysis of the original SIMARC, to determine the reusable design and code parts, and then make a design suitable for the new environment characteristics. Each migration is described from the perspective of their requirements, technical, behavioral and data descriptions. Finally, a comparative analysis between each migration construction is presented.

Contents

List of Tables

List of Figures

Chapter 1:

Introduction

1.1 Overview

A migration of the DOS simulation tool SIMARC to two different platforms is presented. Those target platforms are Windows 95 and X Window (running over Solaris).

In this thesis, several issues required for the migration process are described: first, the origin of the SIMARC program and the theory that supports it is presented in this chapter. In Chapter 2, the original program is described from two different points of view: its operation and its design. Also, the drawbacks of the package are discussed. In Chapter 3, the target migration platforms are described also in two parts: first, the environment provided by the platforms themselves, and later, the characteristics of the different programming tools that they provide. Chapter 4 presents the migration from the perspective of their requirements, technical, behavioral and data descriptions. Chapter 5 analyzes the main differences between both migrations due to the tools provided by each environment. Chapter 6 contains the conclusions.

In general, the objective of this Thesis is to present two different migrations of the same system to two different environments, and show the differences that each program presents, resulting in a different type of coding effort. From the comparison, it is possible to determine the best environment for developing a computing intense application like the SIMARC simulator.

1.2 Background

The rapid evolution of computer platforms, besides its obvious benefits such as systems with better performance, new or improved programming environments and in general, new applications, has also its downsize. Organizations are faced with the problem of maintaining aging software systems.

Building from scratch those systems is an expensive idea, both in terms of time and money, so the idea of program reuse has proven through the years to be the most feasible option [1], although it still is a costly activity. In the software engineering community, experts believe that maintenance activities account for roughly 50-70% of software life cycle software costs [2][3].

The migration of a system is considered an adaptive maintenance activity. This type of maintenance is performed to make a computer program usable in a changed environment [2][4]. From the four existing types of software maintenance, -corrective, adaptive, perfective and preventive-, adaptive maintenance is said to account for 25% of the maintenance budget of any organization [5][6].

Migration is not an easy task. Studies have shown that, on average, only 32% of the original code in a system is reused without modifications [7]. The most costly activities in this process are the comprehension activities, (i.e., inspection activities performed before any decision or change in the software is done) range form 40% [8], to 50% [9], and even up to 90% [10].

In this thesis, an example of a successful migration effort is presented, starting with the comprehension of the original system, and the evolution through the new systems.

1.3 Introduction to MCAP

1.3.1 Attached Processors and MCAP

MCAP stands for "Modularly Configurable Attached Processors," a comprehensive class of attached or backend processors based on the definition of a set of hardware components and a set of connections [11].

In general, an attached processor is a hardware system connected to a host, executing a specific task or a particular set of algorithms. Because of its definite nature, an attached processor is expected to be very fast and efficient. While the host handles all the operating system functions, the attached processor centers on arithmetic processing [12][13].

When designing an attached processor, some issues have to be taken into consideration [14]:

- Although an attached processor has a specific purpose, it is desirable that the hardware architecture be efficiently[1] utilized by a broad set of algorithms so that it is more generally applicable.
- To efficiently match a set of algorithms to a specific hardware configuration, the design phase of the processor must include some kind of modeling technique. Such a technique must provide useful data to the designers in each phase of the process, and therefore, make this procedure cost and time effective.

The MCAP definitions together with the SIMARC package provide a fast prototyping means to create efficient attached processors for executing computationally intense algorithms [11].

In the next section of this chapter, the MCAP definitions are described.

[1] *Efficient* means that the design must have a high sustainable execution rate.

1.3.2 MCAP definition

As mentioned before, an MCAP is entirely constructed from a standard set of connections and component types that can be used and combined in a building block fashion. Each component has specific characteristics and restrictions, which provide general guidelines or rules to create an attached processor [15].

There are ten types of components, as indicated below:

Table 1.1 MCAP components

Clasification:	Type:	Basic characteristics:
	Instruction (I)	• Receives instructions from host. • Executes internal instructions. • Produces external instructions to be sent to other components. • An MCAP contains exactly one.
Processors:	Elementary (E)	• One input – one output.
	Two inputs (T)	• Two inputs – one output.
	Comparator (C)	• Two inputs – one output + special outputs. • Compares two inputs and sets relational flags.
Routers:	Join (J)	• Multiple inputs – one output. • May include different input patterns.
	Fork (F)	• One input – multiple outputs. • May include broadcasting or other input patterns.
	Link (L)	• Multiple inputs - multiple outputs. • May include different input & output patterns, including broadcasting.
Memory subsystems (controllers):	RAM (R)	• One input – one output, without partitions. • Primarly used for temporary storage.
	Single access (S)	• One input – one output, with partitions. • Used for outputting and inputting to memory. • May be connected to more than one banked or interleaved memory module. • Allows use of windows
	Dual access (D)	• Two inputs – two outputs, with partitions. • Basicly, same as (S), except of management of two input streams and two output streams.

Each one of these components provides modes of operation and instructions that allow their particular programmation[2].

The other definition set utilized in MCAP is the one formed by the connections. There are two kinds of connections: instruction connections and data connections. The data connections are specified by the designer between specific components and consist of a data bus and a Req/Ack pair. On the other hand, the instruction connections connect the instruction component to all the components in the architecture with the objective of distributing the instructions produced by the instruction component. An instruction connection consists of an instruction bus, an address bus, and a Req/Ack pair.

All the MCAP connections are unidirectional and asynchronous, i.e., components communicate between themselves asynchronously, and also, each component works independently from the rest of the architecture. Each component performs its operations as the necessary operands are received; this is the reason why MCAPs are similar to dataflow machines [11], i.e., MCAPs do not follow the control-flow mechanism to direct program flow as in Von Neumann machines [13]. In Fig. 1.1 the hardware configuration of an MCAP architecture for performing matrix operations is shown.

1.3.3 Efficient design of MCAPs

Given that the execution of an instruction in an MCAP component depends on data availability, an architecture can be designed exploiting fine-grain, instruction-level parallelism.

[2] For a complete description of components attributes and instructions, consult Appendix A.

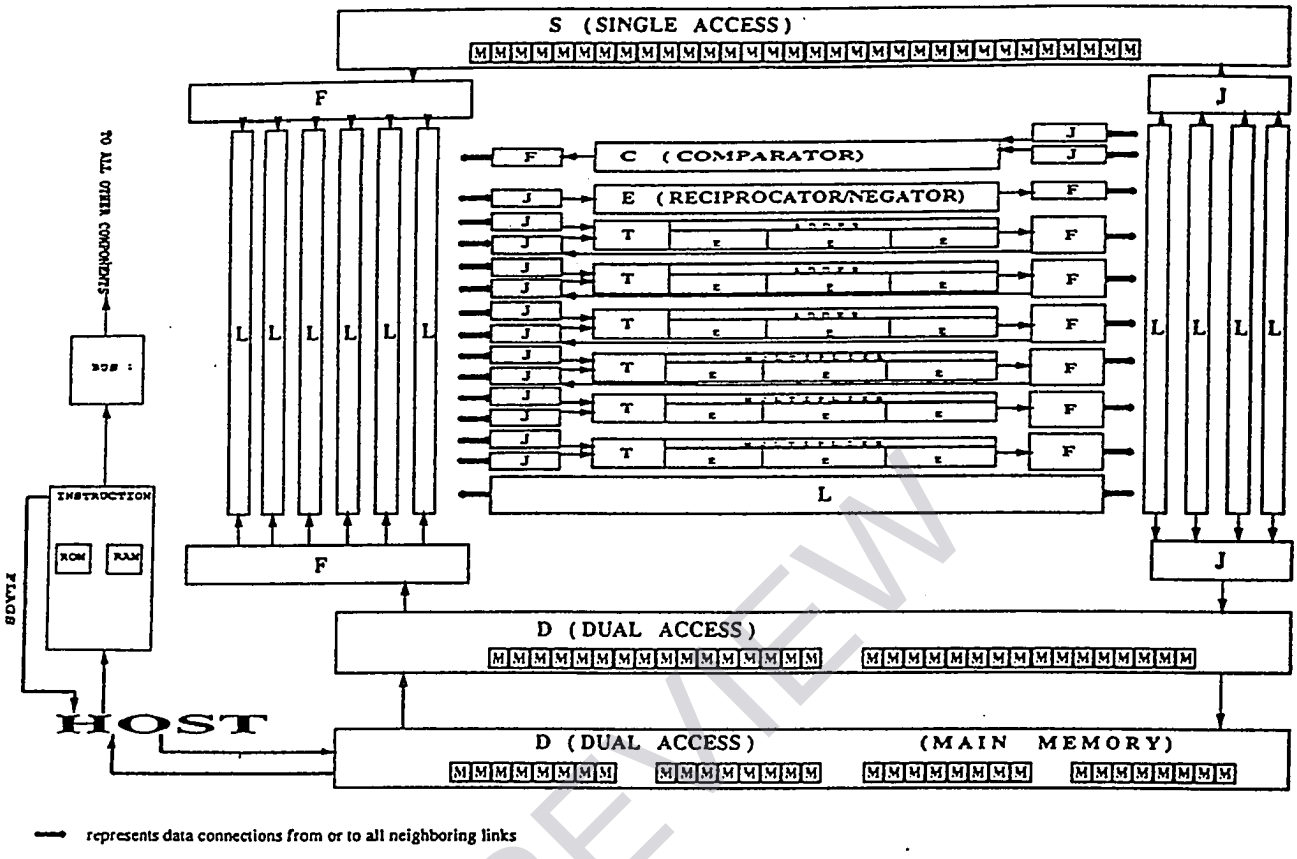

Fig. 1.1 MCAP architecture for performing matrix operations

To exploit to the maximum the MCAP characteristics, a designer should take into consideration the following [16][17]:

- overlap of instruction decoding and distribution with data processing when possible,
- dynamic formation of memory-to-memory pipelines,
- features to reduce the movement of data, like inclusion of memory subsystems in strategic places, to prefetch operands and store results, and
- use of patterns to automatically prefetch data and store results.

Chapter 2:

SIMARC package

2.1 General description: different parts and files generated.

As mentioned in the previous chapter, SIMARC is the simulation package for MCAP architectures. Its original design dates from 1994 and was defined as a menu-driven package written using C++ for PC compatible computers operating under DOS [11].

This system is a set of programs that, as a whole, form a fast prototyping tool for MCAP design. The system is presented to the user as a menu-selection window in which any tool can be selected at any time. The tools available to the user are:

- ***Editor:***
  This program is a graphical tool that allows the creation and modification of an architecture, via a component toolbar [18].

- ***Assembler:***
  When a source program has been created, this program assembles it into a load form that can be used by the simulator [19].

- ***Simulator:***
  Graphically simulates the operation of a program on a particular architecture and also accumulates quantitative results.

- ***Results:***

  This tool displays the results generated on a particular simulation using a combination of graphics and text.

- ***Display:***

  This tool displays an architecture with a previously selected attribute, like type, mnemonic, execution time, etc.

The programs in the SIMARC system communicate through a collection of files, which are described below [20]:

- ***Architectural file (.sar)***

  Contains a description of the components and connections conforming to a certain architecture, (such as font sizes, components positions), and the attributes defining each one of the components (such as queue sizes, execution times). This file is generated by the editor.

- ***Information file (.sif)***

  Contains a textual description of the architecture, i.e., a more readable description. This file is generated by the editor.

- ***Program file (.sas)***

  This file contains the source program generated by the programmer. Originally this file had to be generated by the programmer using a text editor. Because of the difficulty and complexity involved by this kind of programming, a graphical tool was developed to help in this task. This program is the APP (Assembler Pre-Processor), and has been used since 1996 as an extra tool; it is not included in the

original SIMARC package. For more information in the APP program, consult [21].

- ***List file (.slt)***
  This file contains a list of the errors (if any) produced during the assembly of a program file.

- ***Load file (.sld)***
  This file contains a program in the form needed by the simulator. It is produced by the assembler.

- ***Result file (.srt)***
  Contains the results produced by a particular simulation.

- ***Intervals file (.sir)***
  Contains the different times at which results are to be stored for a particular simulation.

The relationships between files and programs can be viewed in the diagram in Fig. 2.1.

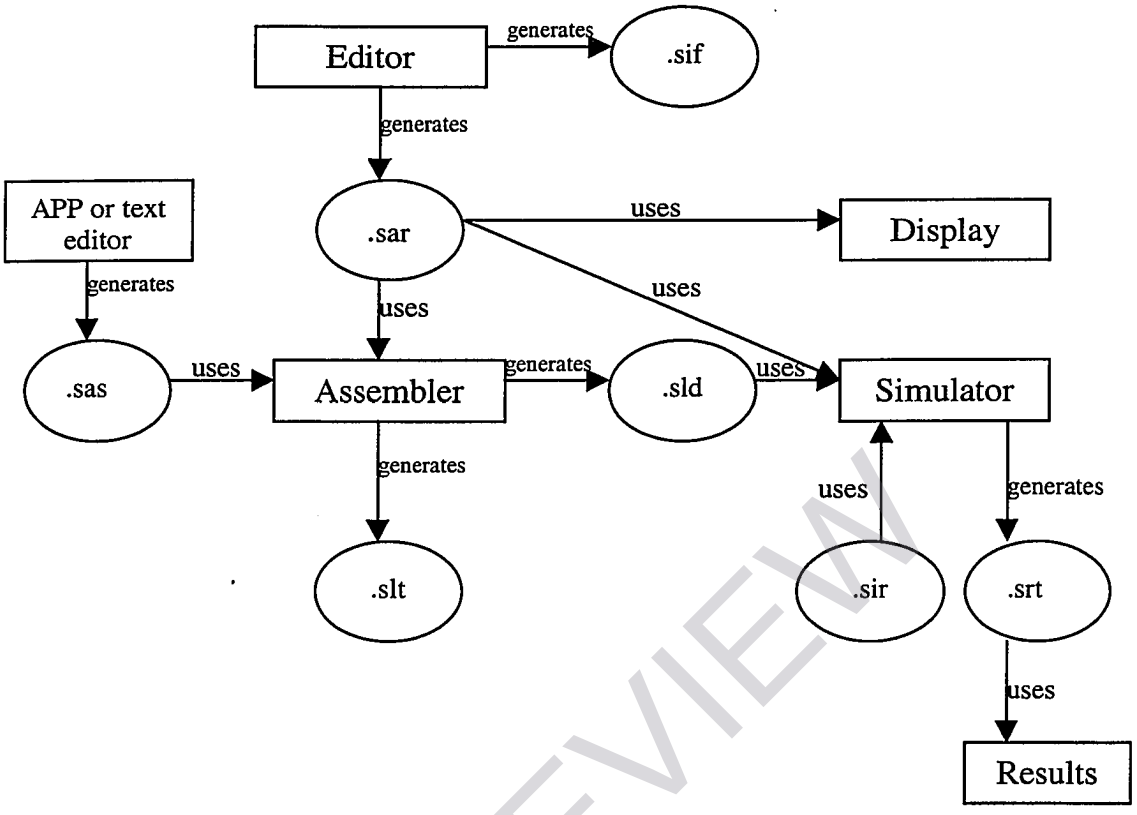

Fig. 2.1 Relationships between files and programs.

2.2 Design and organization of original SIMARC

2.2.1 Design

Although C++ was used to program SIMARC, the general design structure of the program can be viewed as a structured system. For this reason, the body of the system, i.e. the structure that controls and processes inputs from the user and makes the proper function calls, is described in Section 2.2.1.1 using decomposition. The complete listing of files and functions is included in Appendix B.

On the other hand, based on the definition of components and instructions of MCAP, these sets, their attributes and methods are defined as objects. For this part, Object Modeling Technique (OMT) object diagrams is given in Section 2.2.1.1.

2.2.1.1 Structured design

- **Level 1**

At the very top level of the design of SIMARC, the system can be decomposed as:

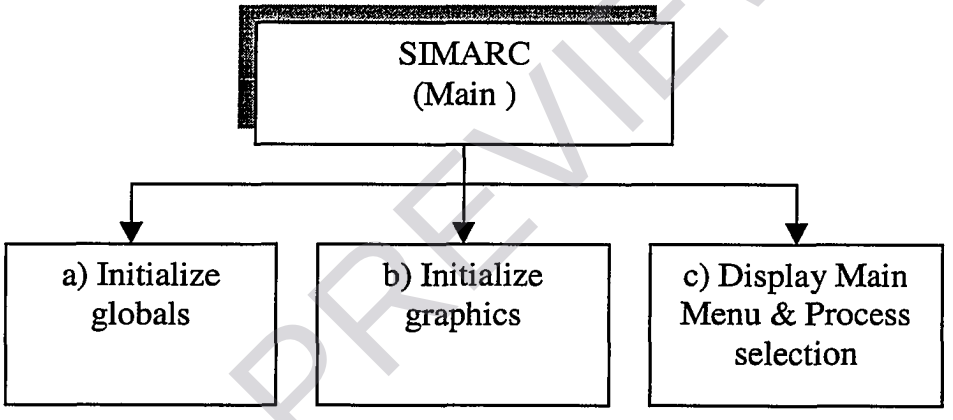

Fig. 2.2 Top – down design level 1

a) Initialization of all the file pointers, variables and object structures needed by the other processes.
b) Initialization of graphics with the proper driver, or aborts the execution of the system
c) Processing and execution of the system, as described in the level 2.

- **Level 2**

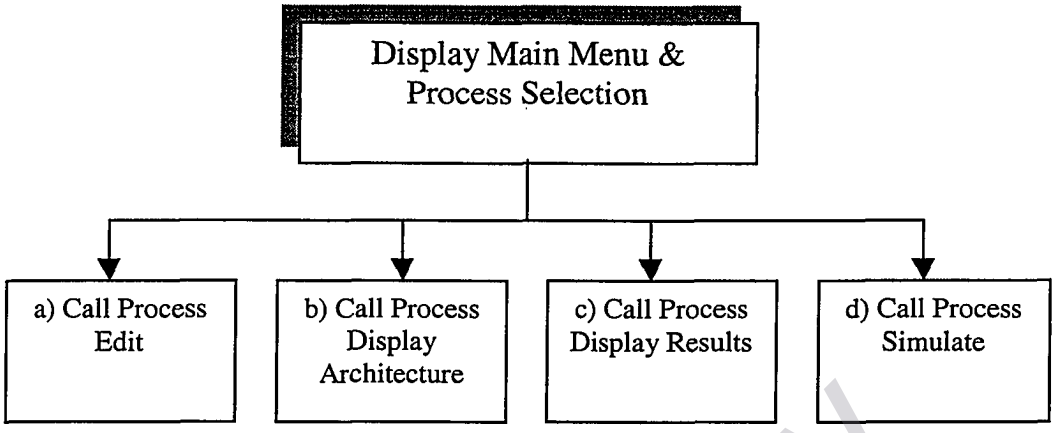

Fig. 2.3 Top – down design level 2

The process "Display Main Menu and Process Selection" can be viewed as a three phase function that:

- graphically displays the main menu on the screen,
- gets the key pressed by the user, and
- if the key is actually a valid option, call the appropriate process.

Each process is described next.

a) Process Edit is a function that, via a menu, obtains the type of file to be edited, and
   - if the file is a text file (i.e., any file except .sar) calls the DOS editor "edit", or
   - if the file is an architecture file (.sar), then calls the SIMARC program editor using a System Call.

b) Process Display Architecture is described at level 3-a.

c) Process Display Results is described at level 3-b.

d) Process Simulate is described at level 3-c.