CSDSim: A didactic processor simulation environment based on the client / server architecture

Gustavo R. Rodrigues, Douglas M. Becker, Flávio R. Wagner

Universidade Federal do Rio Grande do Sul, Instituto de Informática
Av. Bento Gonçalves 9500, Porto Alegre - RS, Brasil
{gribeiro, schuster, flavio@inf.ufrgs.br}

Abstract

Computer processors are incredibly complex and many times hard to evaluate and understand. Moreover, processor organization can hardly be studied on a pure theoretical base, and experimenting with real processors does not allow the analysis of the internal processor structure. Therefore, the use of simulation models that describe their behavior can facilitate their comprehension. This work introduces CSDSim, a didactic environment for teaching and evaluating processor organizations using the DLX processor. The environment is based on a client / server architecture and uses concepts of Visual Interactive Simulation and Object Oriented Hardware Modeling.

Keywords: **Processor Organization, DLX, Visual Interactive Simulation, Object Oriented Simulation, SIMOO Environment.**

I. INTRODUCTION

Modern processors are complex entities of difficult understanding and evaluation. Their design and production are expensive and time consuming. Therefore, simulation models of real processors are created to assist hardware engineers to estimate how their decisions will affect the project before it is implemented on an integrated circuit. Although largely used by the industry, simulation also proves to be of great interest as an instrument for helping researchers at universities to develop new processor models, as well as helping students to learn about processor organizations by inspecting the internal processor behavior. Currently, several simulation tools can be found, but most of them require, from the user, a detailed knowledge of computer organization and design.

This work is part of the T&D–Bench (Teaching and Design Workbench) project under development at UFRGS. This project aims at the development of a processor simulation environment exploring the concept of object-oriented modeling to create a very flexible support for investigating and learning computer organization models, giving freedom for users (designers or students) to evaluate different architectural alternatives and their impact on the system performance.

This paper presents CSDSim, a client/server simulator for the DLX processor introduced by Patterson and Hennessy [PAT95]. The main objective of CSDSim is to provide a flexible, didactic simulation environment for the DLX processor, capable of being useful not only by beginners, but also by advanced students. The adoption of the client/server architecture makes CSDSim adequate to a class environment, where instructors and students play different roles.

This paper is organized as follows. Section 2 briefly discusses object-oriented hardware modeling and visual interactive simulation. Section 3 introduces the SIMOO simulation environment, upon which CSDSim is built. The software architecture of CSDSim is detailed in Section 4. Resources for performance evaluation are presented in Section 5, and Section 6 concludes with final remarks and future directions.

II. FUNDAMENTAL CONCEPTS

As part of the T&D-Bench project, CSDSim inherits the SIMOO [COP97] approach for system modeling. The idea is to stress the concepts of object-orientation as a support for a flexible modeling of processor organizations, in order to give the user maximum freedom in exploring various alternatives and analyzing their impact on performance. Although required to implement the core of CSDSim, this approach also proves to be of great use due to inherent benefits of the object-oriented methodology. Some of these benefits are:

- Decreasing of model complexity, since each entity on of the target architecture is mapped to a class in the model.
- Increasing of model reusability. After modeling and validating an entity, it can be included as part of a library of models, thus being usable as a component by other future systems.
- Easier maintainability of models, due to data and code encapsulation and to the fact that the model is broken into several small entities.

Lately, the large use of simulation in different areas of application, in addition to the increase in the complexity of simulated systems, makes the data analysis of simulation results a complex and time-consuming task. As a consequence, new mechanisms to facilitate system modeling and data presentation and to improve user interaction have been investigated and developed.

Visual Interactive Simulation (VIS) is a concept introduced by Hurrion [HUR76] and Bell and OKeefe [BEL87] and proposed as a series of guidelines to smooth the process of conducting simulation experiments, providing intermediate data tracking, user interaction, and model parametrization.

According to Wagner [WAG96], there are some aspects that need to be carefully analyzed before the development of a simulation environment that is meant to be VIS compliant, such as:

- execution mode selection – step-by-step, time slice and continuous execution;
- selective variable tracking;
- modification and query of parameter and attribute values;
- different graphical presentations of results in accordance to the type of the monitored object;
- statistical analysis;
- graphical visualization of the simulated model;
- saving of intermediate results; and
- simulation time advance and backtracking.

CSDSim attempts to incorporate all these features, in order to enhance user interaction and enable total steering of the simulation execution.

III. The SIMOO Environment

SIMOO [COP 97] is a general-purpose, object-oriented platform for multiparadigm, discrete simulation.

Simulation entities in SIMOO are mapped to autonomous elements, which are objects with their own execution threads, so that they may be easily distributed.

Each object in a given system may be modeled by a different and most adequate simulation paradigm – event- or process-oriented behavior, communication by ports or messages, active or passive reaction to messages, client or server perspective, etc.

Visual interaction facilities are implemented as objects from a domain that is completely independent from the model logic domain. These facilities may be associated to entities and their attributes through mappings between the domains, and these mappings may be dynamically changed during the experiments themselves.

The static structure of the model is graphically specified by hierarchical class and instance diagrams that are enriched with specific features for the simulation, while the dynamic behavior of the objects is coded by using calls to a specialized library. From the specified diagrams and entities' behavior, the graphical editor generates an executable model, which already contains default resources for visualization and interaction.

IV. The CSDSim Architecture

CSDSim is intended to be a flexible application where users can interactively participate on the whole design process of a processor's organization.

The first step on designing the CSDSim software architecture was the analysis of the different processor simulation environments that have been developed through the past recent years. Examples of these environments are WinDLX [GRU00], ESCAPE [VER00], SimpleScalar [AUS00] and DLXView [ADA00]. As a result of this study, features that are common to all environments have been identified:

- Single platform – All considered simulators target a specific platform. WinDLX and ESCAPE are designed for Windows, and SimpleScalar and DLXView for Unix.
- User interactivity – Even though all the simulators have some flexibility on choosing the simulation parameters, none of them allow architectural reconfiguration after starting an experiment.
- Construction of the simulation model – All simulators present a fixed model of the target processor that can only be changed through a predefined user interface. If users need to create a new model, they have to modify the source code.
- Single user utilization – All simulators are designed to be used by a single user. They do not provide the ability to have multiple users following the same simulation session.

Based on this analysis, we propose an environment that improves some of these characteristics, but targets a different audience: computer science or computer engineer students enrolled in a computer organization course. The main goal is improve user interaction, allowing the user to modify simulation parameters on the fly and providing a graphical interface for intermediate data analysis. Furthermore, the user shall be able to define its own performance evaluation experiments and to visualize their results in different ways.

Since CSDSim is an instrument to be used in class, it has been designed as a client/server architecture. This approach was chosen since it permits multiple users to connect to one existing model and interact with it in

different ways at the same time. At the same time, we partially overcome the problem of platform availability introduced by the use of SIMOO on the server side, making CSDSim a cross-platform application on the client side.

The CSDSim architecture has the DLX model on the server side and the user interface on the client side. The communication between server and client is performed through a proper TCP/IP protocol. There are two kinds of possible clients: the master client, which may control the experiment, and the slave client that may only follow it. These three elements can be described as follows.

- Server (CSDSim-Server): it was developed using the SIMOO environment. It contains an object that is responsible for the communication and another that implements the model of the DLX processor.
- Master Client (CSDSim-Client): it is responsible for the control of the simulation. Its functionality includes connection management, model configuration, user interface, and data analysis and presentation.
- Slave Client (CSDSim-Viewer): it is only used on a multi-user mode. Its functionality is a subset of the master client's one, hence being totally dedicated to data analysis and presentation.

The interactions between them are shown in Figure 1.

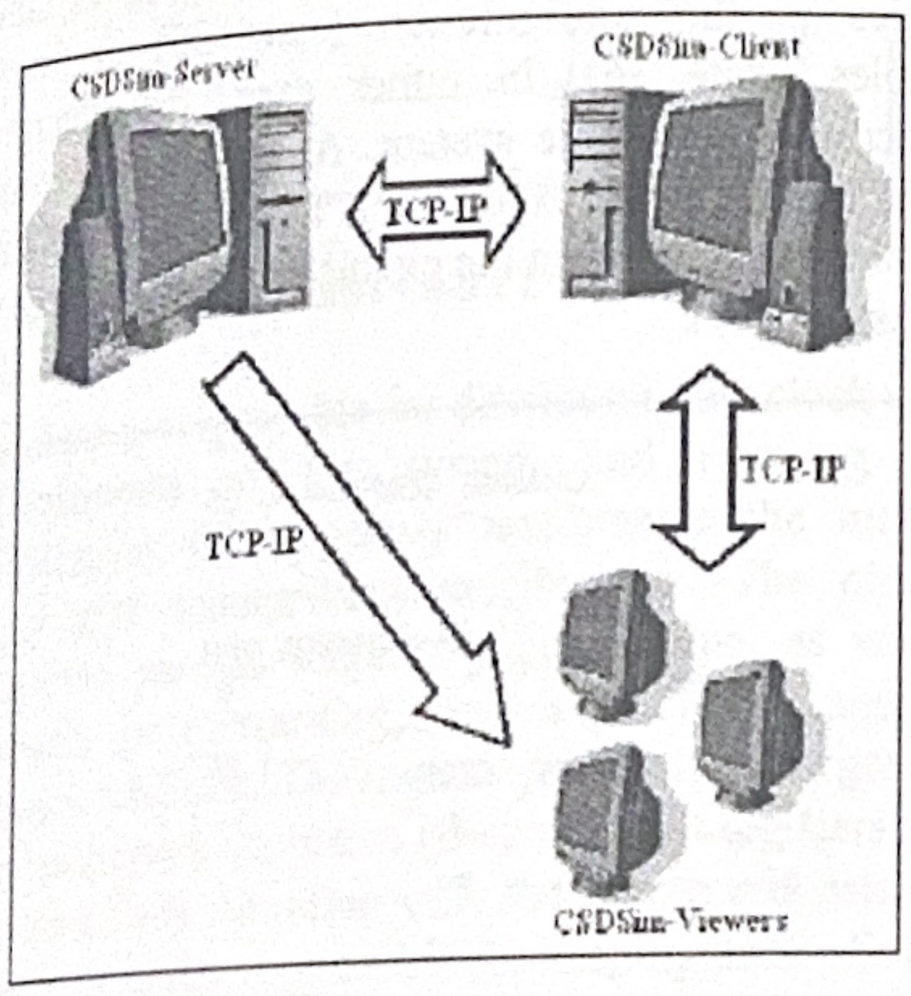

Fig.1 – The CSDSim architecture

The server was developed using the SIMOO framework. In that manner we have the advantages of using an object-oriented model. As a drawback, we restrict the execution of the server to a Windows 95/PC platform.

On the other hand, the client is implemented in Java. Since the performance of the application itself is not a major concern, we take advantage of benefits that the language offers:

- a powerful library for visual interface programming (*swing*);
- elimination of portability issues, since the client can run on any machine that has a Java Virtual Machine installed; and
- code flexibility, since Java is an object-oriented language and it is naturally easier to enhance the client with as many new features as desired.

The implemented architecture evidences several advantages, since:

- it provides natural support for the VIS paradigm, where visualization of simulation results is independent from the simulation process itself, hence requiring from the user only the client software;
- the environment can be easily adapted to other processor architectures;
- the user can make modification on the processor organization without previous knowledge of what has been previously modeled;
- the application can be used in class, since the instructor controls the simulation using CSDSim-Client and students visualize the process using CSDSim-Viewer; and
- the client can be easily adapted to be used within a browser, therefore allowing the user to run the application through the network. This is possible by transforming both CSDSim-Client and CSDSim-Viewer into Java applets that are downloaded and activated when the user accesses a specific URL.

A detailed explanation of the communication protocol, as well as an in-depth description of the three elements of the system, is presented in the remaining of this section.

A. Communication Protocol

The communication protocol is based on a message exchange methodology and implemented upon TCP/IP, as shown in Figure 2. Its messages are divided in two parts: the header, which contains one byte that holds the message identification; and the parameter array, which has zero or more parameters that are particular to that message.

A set of pre-defined ports is used to facilitate the communication between the elements of the system. They are:

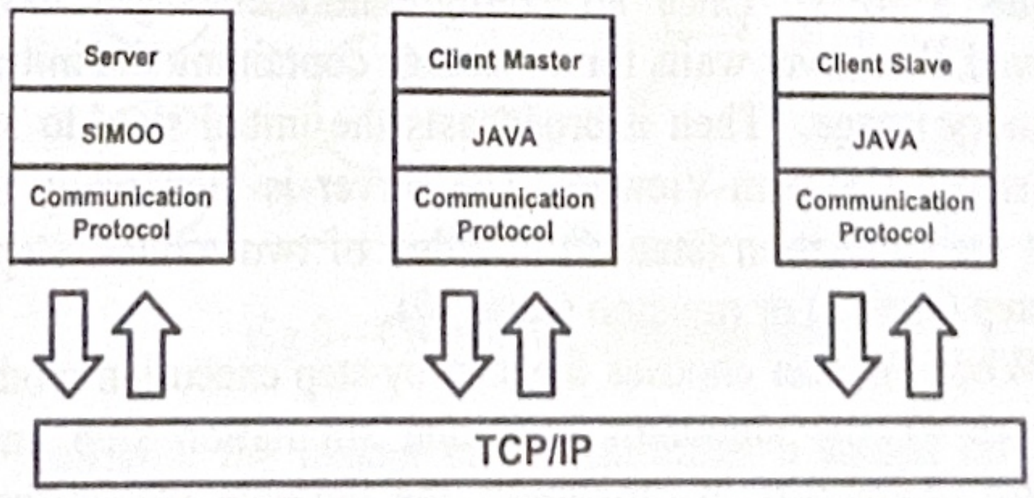

Fig.2 – Communication protocol

- 5000: port used for bi-directional communication between CSDSim-Server and CSDSim-Client.
- 5001-5020: set of ports used for unidirectional communication between CSDSim-Server and CSDSim-Viewers.
- 6000: port used for bi-directional communication between CSDSim-Client and CSDSim-Viewers.

Each part of the system – DLX-Server, DLX-Client, and DLX Viewer – has its behavior described by a Finite State Machine (FSM).

B. CSDSim-Server

CSDSim-Server is the entity responsible for executing the simulation model. It contains the classes that implement the DLX model and the communication protocol on the server side.

B.1. Protocol

The functionality of the server is based on a finite state machine (FSM), which was divided into three sub-FSMs in order to make its understanding more straightforward.

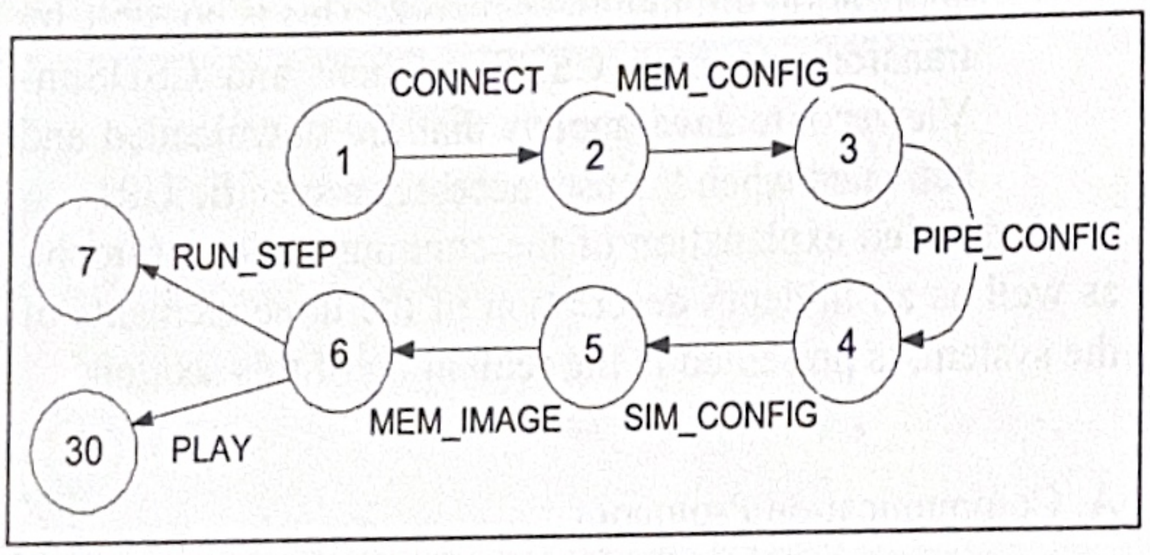

Fig.3 – Server FSM, common part

Figure 3 shows the first part of the FSM. It contains the set of states that is common to two execution modes supported by the simulator (step-by-step and run/stop).

Initially, the server initializes a socket on the port 5000 and waits for connection requests (state 1). When the connection between server and master client is established, the server waits for the arrival of configuration messages (states 2 to 5). Once all configuration messages have arrived, the server waits for a message containing the initial memory image. Then it broadcasts the initial state to all registered CSDSim-Viewers. The server is then ready to start the simulation (state 6) in either of two modes: step-by-step (state 7) or run/stop (state 30).

When the user chooses the step-by-step execution mode (Figure 4), the simulator runs one simulation step and pauses. After that, it broadcasts the variable values that were modified on this step to all clients (state 7). One of four situations can then occur: a) the user changes the configuration of the processor and this change is broadcast to all registered clients (state 9); b) the user changes the value of a variable and the new value is updated in the whole system (state 10); c) the simulation continues and one more step is executed (state 7); and d) the simulation ends (states 8 and 2).

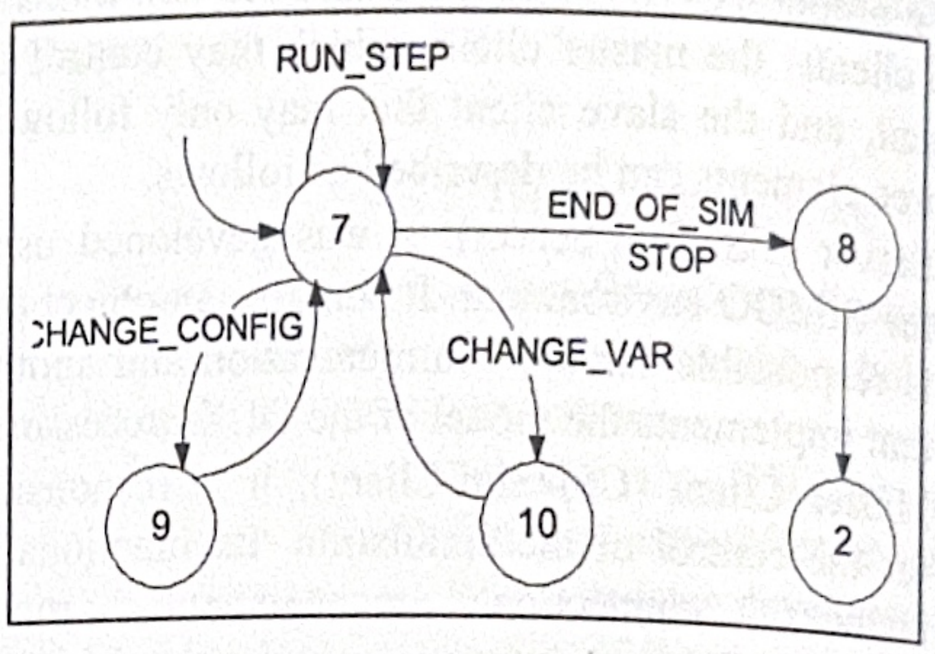

Fig.4 – Server FSM, step-by-step mode

Once the run/stop mode is chosen (Figure 5), the simulator executes until the end of the simulation is reached (state 34), unless it is stopped (state 32) or paused (state 31) by the user. In case of a pause, the user can resume execution (state 30), stop execution (state 32), or introduce changes in the configuration (state 35) or values of variables (state 36). In either case, the changes are broadcast to the whole system. After the user requests a stop or the end of the execution is reached, the values of all variables are updated on the clients (state 33) and the server returns to state 2.

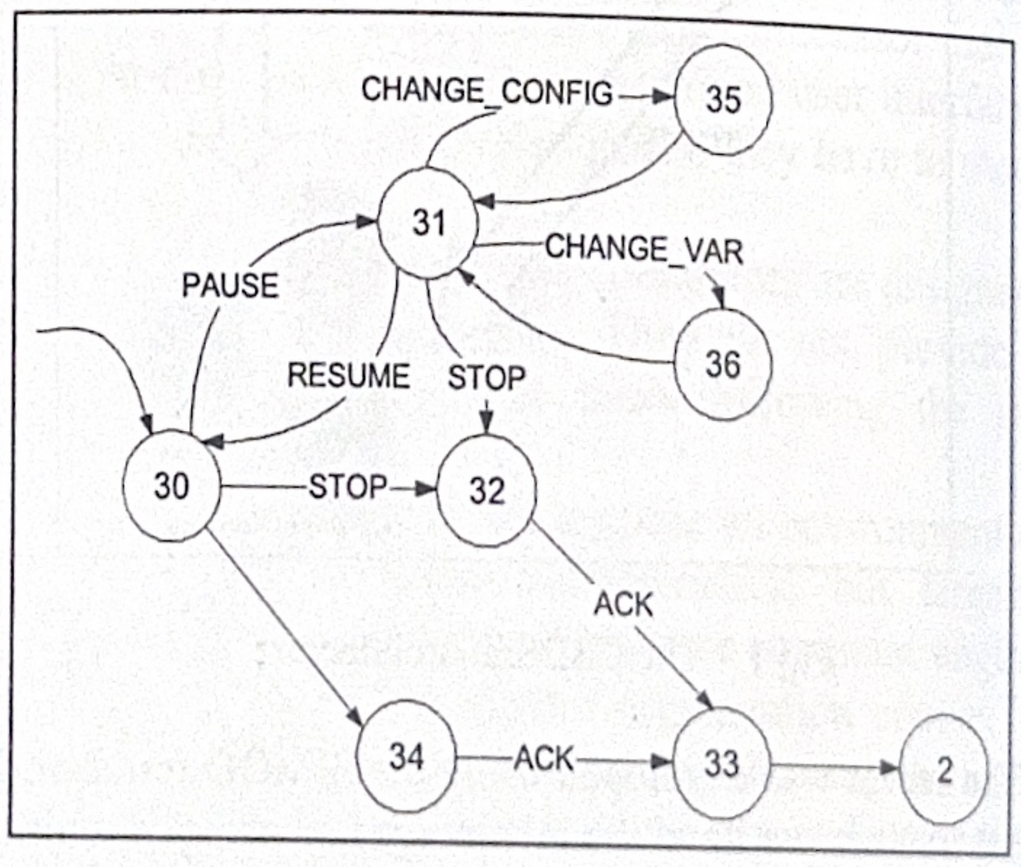

Fig.5 – Server FSM, run/stop mode

B.2. Model

The DLX processor was modeled at 4 different aggregation levels, according to the hierarchical approach enforced by the SIMOO framework.

The first level presents the server as a unique entity in the model. This is the entity that will be executed by the simulator, which will expand its description into its sub-entities.

The second level implements two entities: the communication protocol (the whole protocol is described at this level) and the DLX model, which contains all sub-entities that describe the DLX organization.

The third level introduces the functional blocks of the DLX organization, as shown in Figure 6. Five of these blocks are described here: *clock*, *memory*, *pipeline*, *register file* and *monitor*.

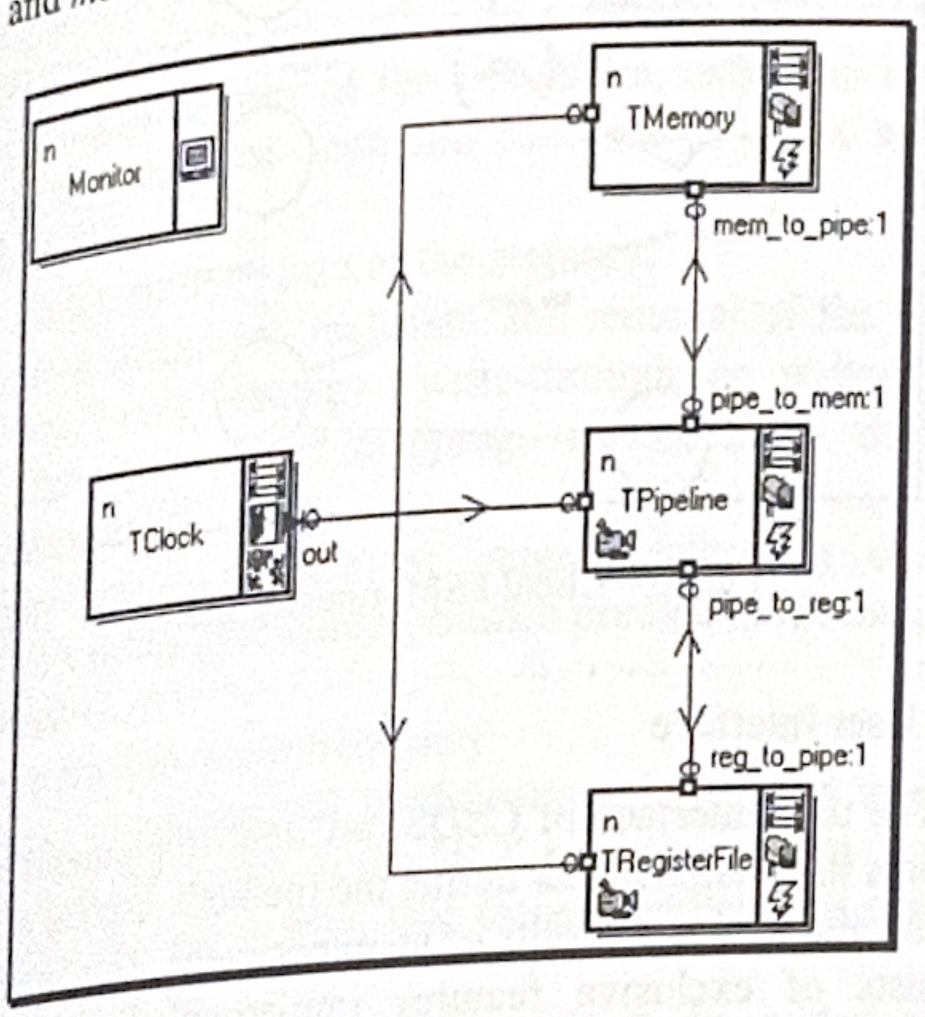

Fig.6 – SIMOO model for the DLX organization

The *clock* is responsible for distributing the clock signal to all elements of the system and ensuring their synchronization. The *memory* implements the memory hierarchy and management mechanisms. The *pipeline* describes the various stages of the pipeline, as well as mechanisms for forwarding, branch prediction, and data dependency detection. The *register file* contains all general-purpose and system registers (floating-point registers were not implemented), handling only read/write requests. The *monitor* is the entity responsible for keeping track of values of variables that are being visualized. It observes all the messages exchanged between the entities of the model and send the values of monitored variables to the clients when these variables are updated. It also contains information on the system configuration.

The lowest level of the model hierarchy presents a more detailed description of the memory and the pipeline. As shown in Figure 7, the *memory* entity contains four sub-entities that model the memory hierarchy: the main memory, the cache, the instruction cache, and the data cache. Here, *cache* is a specialization of *main memory*, and *data cache* and *instruction cache* are specializations of *cache*. The *pipeline* entity contains six classes, the first one describing a generic stage of the pipe, while the remaining ones are specializations of the generic class that correspond to each one of the stages: fetch, decode, execution, memory access, and write back.

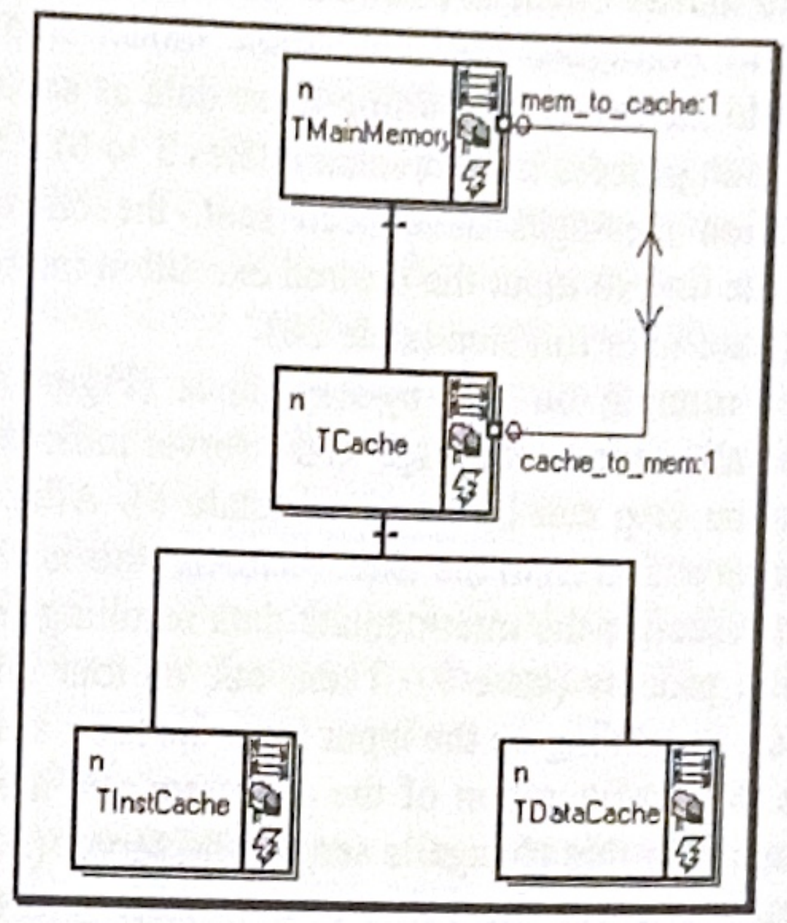

Fig.7 – SIMOO model for the memory

C. CSDSim-Client

CSDSim-Client is the element of the system responsible for controlling the simulation and implementing the user interface for system configuration and data visualization.

C.1. Protocol

As CSDSim-Server, CSDSim-Client also has its functionality based on a finite state machine (FSM). It was also divided into three sub-FSMs in order to make its understanding more straightforward.

Figure 8 shows the set of states that are common to both types of execution supported by the simulator.

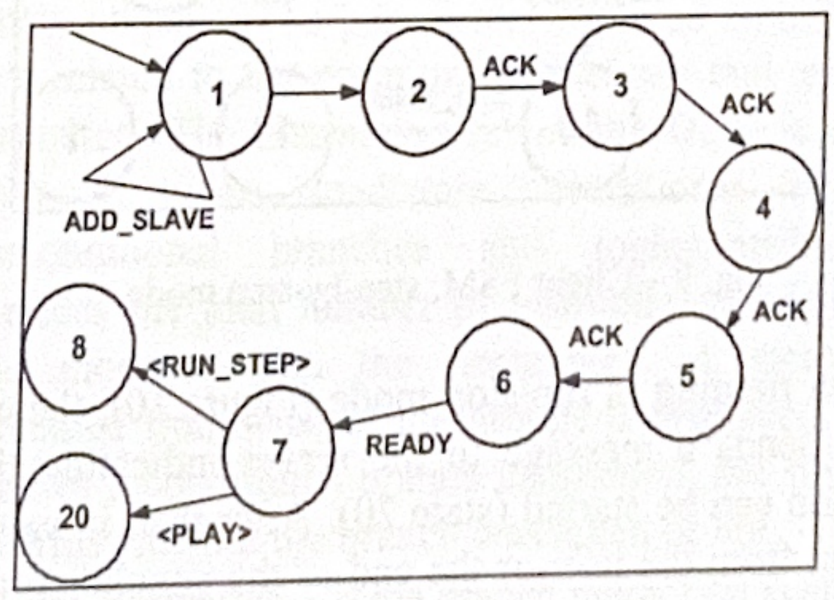

Fig.8 – Client FSM, common part

Initially, the master client initializes a socket on port 6000 to be used for communication with slaves. It waits for requests of connection until a time-out occurs. For each

registered slave, it returns a message confirming the connection request plus the server IP address (state 1). Then, it creates a socket on port 5000 for communicating with the server and sends a message requesting a connection to it (state 2). When the connection between server and master client is established, the client waits for the user to configure the simulation model and sends messages to the server containing these data as soon as the configuration process is completed (states 3 to 6). Once all configuration messages have been sent, the client waits (state 7) the user to input the desired execution mode: step-by-step (state 8) or run/stop (state 20).

When running on step-by-step mode (Figure 9), the client initially sends a message to the server indicating that a simulation step can be executed (state 8). After that, it waits for an action from the user. Actually, this is when the user can visualize the intermediate data resulting from the simulation process (state 9). Then, one of four situations can occur, depending on the input from the user: a) the user changes the configuration of the processor and a message communicating this change is sent to the server (state 13); b) the user changes the value of a variable and the new value is sent by a message to the server (state 14); c) the simulation continues and one more step is executed (state 8); or d) the simulation stops (state 10) or ends (states 11 and 2).

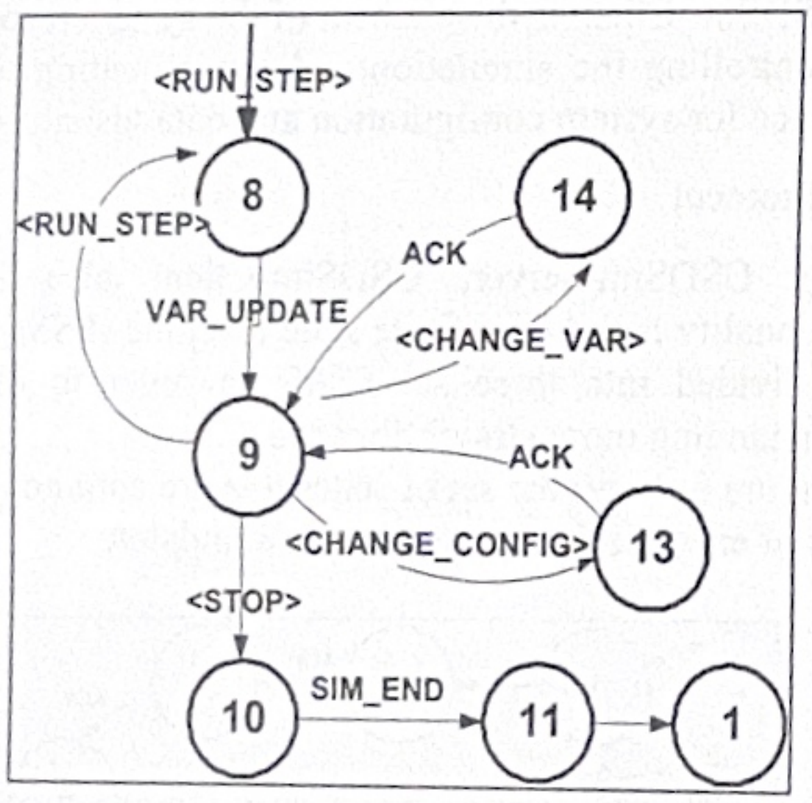

Fig. 9 – Client FSM, step-by-step mode

When running in run/stop mode (Figure 10), the client initially sends a message to the server indicating that a simulation can be started (state 20). After that, it waits for an action from the user or the end of the simulation (state 21). Then, one of three situations can occur: a) the user pauses the execution (state 22) and receives from the server a message with all the updated variables (state 23), returning to state 9; b) the user stops the execution (state 29) and receives a message from the server with the values of all variables on the last executed clock cycle (state 25); or c) the simulation ends (states 25, 28, and 1). The user can introduce any modifications on the processor configuration and variable values at any time during states 21 and 23.

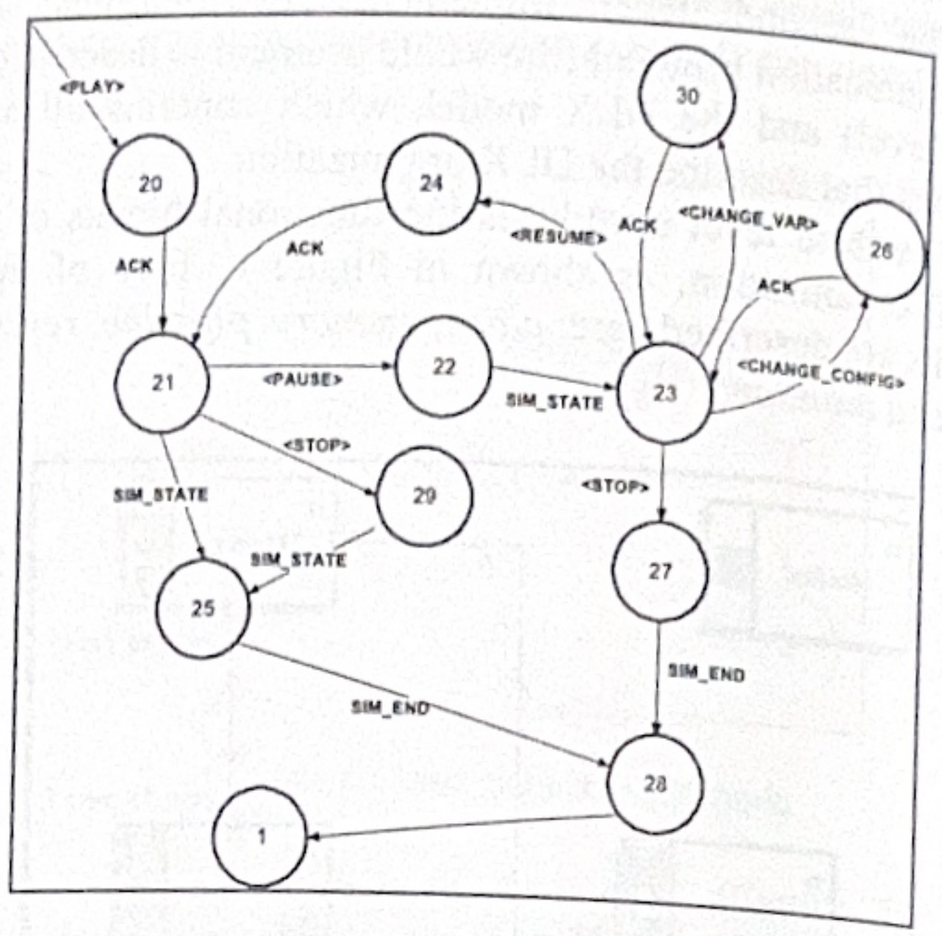

Fig.10 – Client FSM, run/stop mode

C.2. User Interface

The user interface of CSDSim-Client implements a set of tools that help the user define the methods used to create, follow and evaluate results of an experiment. This interface consists of exclusive features implemented to support CSDSim basic functionality (system configuration, performance evaluation and intermediate data visualization), in addition to resources introduced in other tools, like WinDLX, that are used for intermediate data visualization. Figure 11 shows the user interface used for visualization of the instruction flow on a time diagram chart.

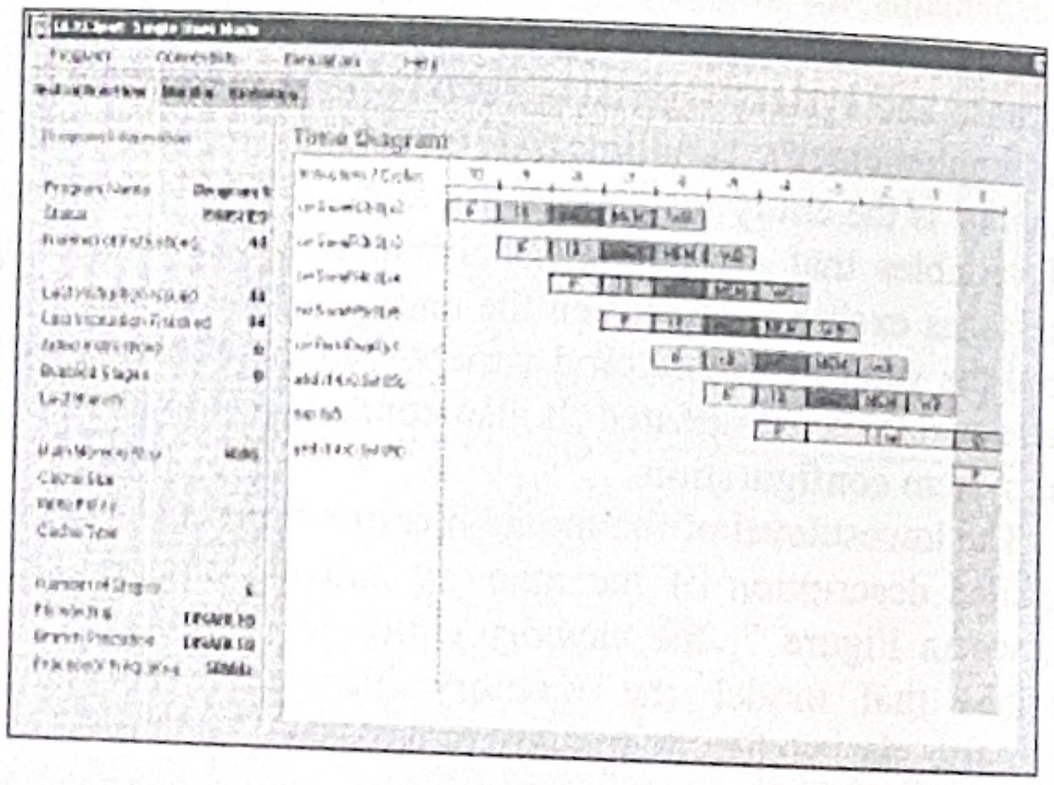

Fig.11 – Displaying the pipeline stages

V. PERFORMANCE EVALUATION

In the process of developing a real processor, several aspects can affect the performance of the processor: instruction set, functional organization, logical design, physical technology, and so on.

Once the requirements of the processor are known, the designer needs to optimize the system organization. It is important to point out that the choice of the metrics used to evaluate performance is extremely important in this design phase.

In order to allow students to experience some of the tasks involved in the real processor design, mechanisms were developed for evaluating the processor performance.

During all phases of the simulation, users can change the configuration of important components of the system, such as:

- main memory – size of the memory;
- cache memory – inclusion and removal of the cache, size, write strategy (write-through or write-back), cache type (direct mapping, set associative, or fully associative);
- pipeline – number of stages, activation of the forwarding mechanism, branch prediction mechanism; and
- system: processor frequency.

These organizational modifications have been already foreseen in the design of the DLX processor model. It was developed in such a way, that these modifications can be made at run time.

An important feature that needs to be added to a performance analysis environment is a mechanism to compare two different systems. Based on that, the simulation model has embedded in its design some variables that quantify statistics about the model execution. These variables are shown in the Statistical Analysis Window (Figure 12).

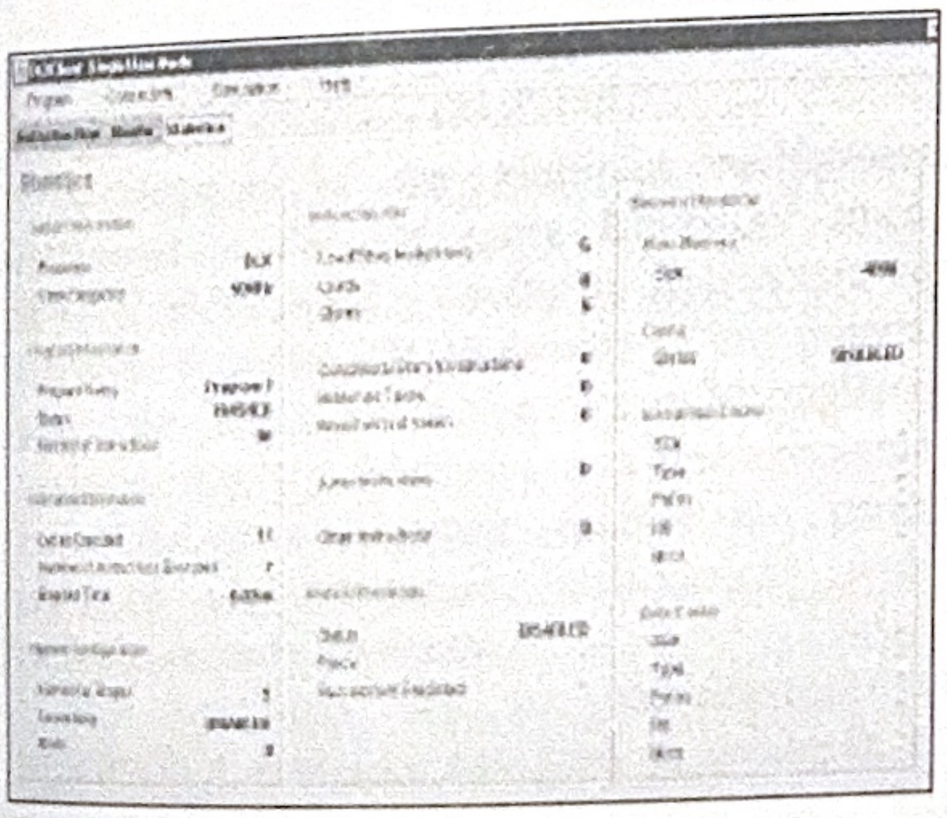

Fig.12 - Statistical Analysis Window

The output data provided by the system are divided into 7 different groups:

- System Information – Here we find information regarding the processor being simulated: processor name and clock frequency.
- Program Information – The data presented here are related to the program that was loaded in the DLX memory. All the information contained in this group is managed by the DLX-Client. The program name is obtained from the user selection from a list of programs available in the configuration menu. The program status can have one of the following values: not initialized, in case the program have not been yet selected; running, if the program is being executed; paused, if the program execution was temporarily interrupted; halted, if the program execution was finished by the user; and finished, if the program execution was finished by the system. The number of instructions in the program is extracted from the assembler.
- Execution Information – In contrast to the previous groups, this information is obtained at execution time, by means of messages sent from the server to the client. Three parameters are contained in this group and regard the program execution up to the current simulation time: number of executed cycles; the number of executed instructions; and total execution time. The execution time (tEX) is calculated by multiplying the number of cycles by the execution time of each cycle.
- Pipeline Configuration – This group is formed by two static parameters that are related to the pipeline configuration: number of pipeline stages and status of the forwarding mechanism. Moreover, there is another parameter that stores the number of stalls that occurred in the pipeline. This last variable is obtained at execution time by means of messages sent by the server.
- Instructions – The parameters supplied in this group are related to the program instructions that are being executed. The instruction set of the DLX is divided into 4 types: load/store, conditional branches, unconditional branches and logical-arithmetical. Besides the total number of instructions executed up the moment, also the percentage of instructions executed from each of the groups is displayed. In the case of a conditional branch instruction, the simulator informs whether the branch was taken or not.
- Branch Prediction – If the branch prediction is enabled, data regarding the chosen algorithm is shown, so that the user can evaluate its effectiveness. At this moment, CSDSim provides the number of correctly predicted

branches and its percentage related to the total number of branch instructions.

- Memory Hierarchy – The configuration of the memory hierarchy is shown. Data to be used for analyzing the performance of data and instruction caches are collected. Memory size, type and hit ratio are some of the values presented in this group.

VI. Conclusions

This paper presented a didactic simulator for the DLX processor, intended for Computer Organization courses. It is based on a client-server architecture that is adequate for a networked laboratory environment, where the model runs on a single server, the instructor may steer the experiments on a master client, and the students follow the experiments on slave clients intended only for visualization purposes (although they may optionally also steer experiments on additional master clients).

The simulator is based on SIMOO, an object-oriented simulation framework that presents several interesting features for high-level modeling and interactive tracking of experiments.

In this initial version, the simulator already offers various configuration items that may be set by the user and displays a rich set of output data collected from the experiments. These resources allow for performance evaluation experiments based on a what-if approach. Future versions of the simulator will extend the user facilities for modifying the processor model as well as the output data gathered for performance evaluation purposes and the corresponding visualization resources. Object-orientation will be a very valuable technique for implementing the desired flexibility.

Acknowledgments

We acknowledge the work of many people who were responsible for the development of the SIMOO framework. Our special thanks go to Bernardo Copstein and João Jornada.

References

[ADA00] ADAMS, George. DLXview - (Preliminary) User's Manual. Available by WWW in http://yara.ecn.purdue.edu/~teamaaa/dlxview

[AUS00] AUSTIN, Todd M.. A User's and Hacker's Guide to the SimpleScalar Architectural Research Tool Set (for tool set release 2.0). Available by WWW in ftp://ftp.cs.wisc.edu/sohi/Code/simplescalar2.0/hack_guide.pdf

[BEL87] BELL, P. C.; OKEEFE, R. M. Visual Interactive Simulation – history, recent developments, and major issues. Simulation, 49(3):109-116, set. 1987.

[COP97] COPSTEIN, Bernardo; PEREIRA, Carlos Eduardo; WAGNER, Flávio R.. The Object Oriented Approach and the Event Simulation Paradigms. In: European Simulation Multiconference, 10., 1996. Proceedings... Budapest: SCS, 1996. P. 56-71.

[GRU00] GRÜNBACHER, Herbert. WinDLX Tutorial - A firstexample. Available by E-mail in maziar@vlsivie.tuwien.ac.at

[HUR76] HURRION, R. D. The design, use, and required facilities of an interactive visual computer simulation language to explore production planning problem. Londres, Univ. of London, 1976. (Ph.D. Thesis)

[PAT95] PATTERSON, D. A.; HENNESSY, J. L. Computer Architecture: A Quantitative Approach. 2nd Edition. Morgan Kauffman Publishers, Inc. 1995.

[PAT97] PATTERSON, D. A.; HENNESSY, J. L. Computer Organization & Design: The Hardware/ Software Interface. 2nd Edition. Morgan Kauffman Publishers, Inc. 1995.

[VER00] VERPLAETSE, Peter. ESCAPE v1.1 Manual. Available by WWW in http://www.elis.rug.ac.be/escape

[WAG96] WAGNER, P.R.; FREITAS, C.M.D.S.; WAGNER, F. R. Um Novo Paradigma para Modelagem e Simulação Interativa Visual. In: Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, IX.Caxambu, Brazil, October 1996. Proceedings, SBC/UFMG, 1996. pp 87-94.

[YOU91] YOURDON, E.; COAD, P. Object-Oriented Analysis. Second Edition. Prentice Hall, 1991.