A Taxonomy of Computer Architecture Visualizations

Cecile Yehezkel
Department of Science Teaching
Weizmann Institute of Science
Rehovot 76100 Israel
ntcecile@wisemail.weizmann.ac.il

ABSTRACT
In the domain of software visualizations, taxonomies have been built to classify and evaluate environments for algorithm and program visualization. Taxonomies are valuable too, in the development phase of new environment. The field of computer architecture and assembly language that is at the border of the software and hardware domains has been neglected. In this paper I present a taxonomy focusing on this field with emphasis on the didactic and cognitive aspect of the visualization environment. The ECPU learning environment was developed to teach computer architecture and assembly language at introductory level. The considerations that were taken in account during the ECPU development are presented in the context of the taxonomy.

Categories and Subject Descriptors
I.6.1 [**Simulation and modeling**] Simulation Theory – *Model Classification.*

General Terms
Design

1. INTRODUCTION
Software visualization (SV) environments were developed as programming tools for professionals or as instructional tools for demonstration or interactive study. It is customary to divide SV environments into two groups: algorithm visualization (AV) which is used for studying abstract algorithms, and program visualization (PV) which visualizes source code or data structure [9]. This paper concentrates on PV for teaching computer architecture, building a taxonomy that relates not only on visualizations but on the whole visualization learning environment. Visualizations have didactic and cognitive potential which is effective if exploited by the learning environment to create a fructuous interaction between the student and the visualization. In the context of this taxonomy the ECPU environment, which I designed will, be presented. It was developed for a specific target population: 11th-grade high school students of computer science. The ECPU environment was tightly integrated with theoretical learning material and lab activities [13].

ITiCSE'02, June 24-26, 2002, Aarhus, Denmark.

This paper presents a part of a new taxonomy of program visualizations for computer architecture (PV), based on my experience in designing and teaching with ECPU. During the ECPU development, I was hampered by the lack of guidelines on effective SV design [11], and the lack of an appropriate framework to define the characteristics of the environment. Therefore, I was motivated to build a new taxonomy that could provide a PV classification tool, emphasizing the didactic and cognitive aspects of the visualization environment. These aspects are fundamental to plan activities that are an integrated part of the learning environment. They are crucial for effectiveness. Recently the results of a fructuous meta-study of algorithm visualization effectiveness were published, concluding that the form of learning activity is more important that the form of the visualization itself [4].

2. OVERVIEW OF PV TAXONOMIES
Existing PV taxonomies have been based on different classifications principles and have focused on different characteristics of visualization. Myers [5] taxonomy for PV classifies systems along two axes: program aspect (code, data or algorithm) and display style (static or dynamic). Roman and Cox [10] claim that a PV taxonomy must be based on a theoretical model; their taxonomy is derived by viewing a visualization as a mapping from the program to a graphical representation. Their taxonomy has five categories: scope, abstraction, specification method, interface and presentation (though the last one is not derived from the model). Price, Baecker and Small [8] construct a very broad taxonomy of SV, spread over educational and engineering visualization environments in computer science. They chose a n-ary tree topology to enable future expansion [8]. They categorized visualizations according to general external criteria, but did not focus on didactic and cognitive aspects of visualization. My taxonomy is based on theirs, but I have specialized it to PV environments, and extended it to include more didactic and cognitive aspects.

3. ENVIRONMENTS FOR PV
Cassel et al. [3] have made a representative list of environments used in teaching architecture. In the scope of this paper it is not possible to make a comprehensive review of existing visualizations in this domain However, to illustrate some aspects of the taxonomy, three PV environments will be presented as examples: ECPU, RTLSim and LMC.

The ECPU, RTLSim and LMC environments are described in detail in the paper [12] and have been submitted to Computer Science Teaching Center (http://www.cstc.org).

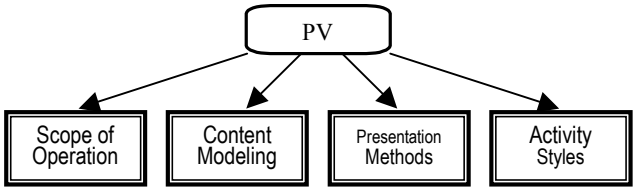
ECPU environment was developed for educational purposes as an integral component of an introductory course on computer architecture and assembly language. It includes a visualization of a simplified model simulating the operation of processor from the Intel 80X86 Family [13].

RTLsim is a program that simulates the data path of a simple non-pipelined MIPs-like processor. When running the simulator, the student acts as the control unit for the data path by selecting the control signals that will be active in each control step [6].

LMC is based on the Little Man Computer paradigm. A Little Man performs tasks within a walled mailroom and corresponds to the computer control unit. The intent is to provide students with a visualization of all the components of the architecture of a computer, and the ability to observe the fetch-execute cycle during program execution [14].

4. DESCRIPTION OF THE TAXONOMY

The taxonomy adopted roughly the categories defined in the top-level of [9], as well as its n-ary tree topology. Some branches were pruned and others were redefined in order to focus on the smaller content domain and to extend it to the didactic and cognitive aspects. Here is the top level of the tree I defined:



This paper will focus on the decomposition of the top-level categories *content modeling* and *activity styles* because they contain the main extensions for the didactic and cognitive aspects of visualization. A detailed description of the other top-level categories will be given in a longer version of this paper. During the discussion, each category will be demonstrated by one or more of the environments mentioned in the previous section; italics will be used to distinguish these notes from the explanation of the taxonomy itself.

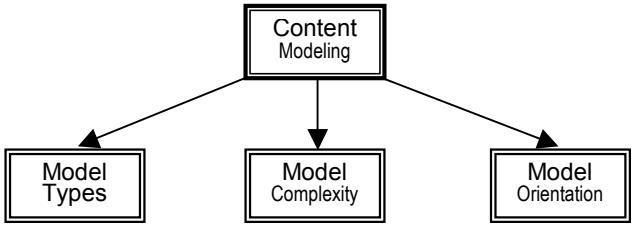
A. Scope of operation

Scope of operation is divided into two sub-categories: (A.1) technical specification of the platform, and (A.2) environment limitations such as the maximal size of a program that can be visualized. This category can significantly affect portability, distribution and accessibility, as well as the complexity of the tasks that the student can perform.

ECPU may not be operated on the Web. The LMC environment was created as a Web environment. It increases accessibility to the environment.

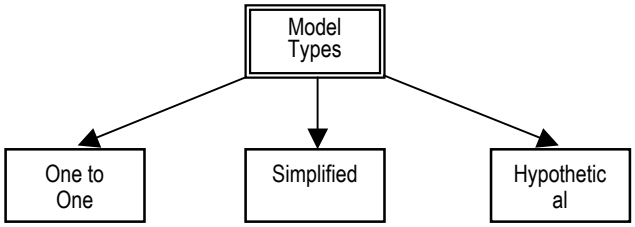
B. Content modeling

This category concerns the choice of a model appropriate to the target population and the didactic approach (cf. [7]). The model will then guide the static and dynamic organization of information within the visualization. The category is sub-divided into three sub-categories, as shown in the following subtree:



B.1. Model type

There are many model types available to the developer of an environment. Price et al. taxonomy contains the category of fidelity and completeness of the content, but they do not relate it to model types. However, the fidelity of the model to a real system is not relevant in the didactic and cognitive context of my taxonomy. Instead, I believe that it is better to distinguish various types of models based upon their didactic and cognitive aspects. I consider three model types as shown in the following subtree:



In a one-to-one model, the goal of the developer is to simulate a real computer to as high a degree of fidelity as possible. A simplified model is close to the real computer, but significantly simpler. A hypothetical model does not purport to simulate a particular computer, but rather to present general principles according to which a computer operates.

B.1.1. One-to-one model

This approach has many limitations. One is the difficulty in preserving fidelity owing to the complexity of computer architecture and to the limitations of the simulation and visualization tools. Even state-of-the-art developers attempting to achieve high fidelity are forced to compromise. Not only are the visualization tools limited, but also there are limitations in the ability of students to perceive and recall the information that is displayed, as well as limitations in their cognitive capabilities to understand the material.

B.1.2. Simplified model

When the developer takes into consideration constraints of the target population such as perception, memory and cognition, he/she may decide to simplify the model according the didactic objectives, while still maintaining fidelity to the basic structure of an architecture. These models are called simplified models.

In developing the ECPU environment, we chose to simulate a particular computer (the Intel 80X86 family processor found in the PC computers that the students use), but we decided that there is no didactic justification to simulate the computer in all its elements, both software and hardware. We adjusted the level of detail and the complexity to the target population. RTLsim

simulates the data-path of a simple non-pipelined MIPs like processor[12].

B.1.3. Hypothetical model

A hypothetical model is one that does not simulate a pre-existing computer. The motivation is usually to generalize, in the sense that the model is not faithful to the characteristics of a particular computer, but instead illustrates general principles that reflect a family of architectures. However, the generalization may mask important details of real computers, thus limiting the possible tasks that students can be asked to perform. The developer can design an original model or base it on an existing model as found in the literature on architecture.

The LMC developers decided to present the Little Man Computer paradigm in which an analogy is made between the processor and a postman who sorts the mail and puts it into different boxes.

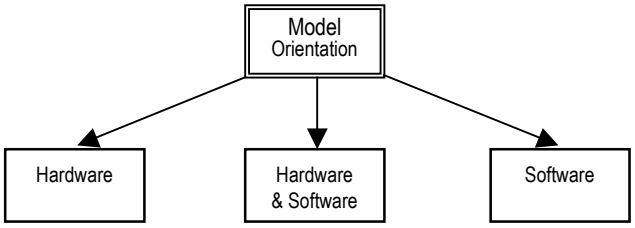
B.2. Model complexity

For the model selected, the developer will adapt the level of model complexity to the cognitive ability of the target population. This category expresses how much detail of the architecture is visualized by the environment.

In ECPU, we hide the arithmetic and logic unit (ALU) in the processor that actually performs arithmetic and logic calculations. Even though the unit is a part of the simulation, it works behind the scenes and need not be visualized. In RTLSim the intern structure of the control unit is hidden.

B.3. Model Orientation

Within a model, it is possible to describe the architecture of the computer and the mechanisms of its operation at different levels of abstraction. Software is considered as a higher level of abstraction than hardware in the hierarchy that starts from the application, through libraries and operating systems, down to microcode and hardware. Model orientation describes the level of abstraction that is visualized by the environment. There are three sub-categories: software, hardware and a combination of the two:



The model orientation of LMC is software, the orientation of ECPU is software-hardware and that of RTLSim is hardware.

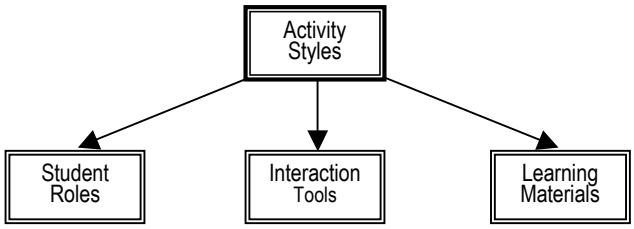
C. Presentation methods

There are various methods that can be used to visualize elements in the model. Sometimes, one method supplies information that extends the information in another, and sometimes, multiple methods of visualization complement each other, enabling the student to form a more complete picture. However, owing to limitations of the environment, there may be a need to hide items or details, and to provide options for focusing on important information. To quote from [7]: "The unifying aim of software visualizations is to make the information apparent - ideally to make selected, elusive information 'visible', even obvious."

In ECPU data can be displayed in binary, hex and decimal (signed/unsigned) format; this can help the student understand that data elements can be interpreted in several ways.

D. Activity Styles

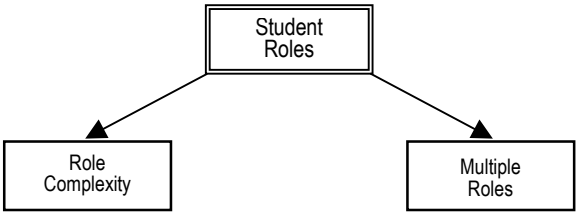
Until now, we have dealt with the definition of what the developer presents to the student and how he/she presents it. This category is concerned with the interaction that the developer creates between the student and the environment. The developer is the one who defines the role that the student will fulfill in the environment; once a role is assigned, the student is given tools for fulfilling that role and in addition tools for interacting with the environment. It is important that didactic considerations guide the development of the interaction tools. By itself, a tool is of little use; it must be integrated with learning materials such as textbooks and lab assignments that require the student to use the environment. The visualization environment, together with the learning material, constitutes a complete study environment. I subdivide the activity styles in three categories as shown in the following subtree:



In the ECPU environment a set of activities constitutes a graduated succession of tasks for self-working, during which the student learns the fundamentals of the structure of the computer and programming at the assembly language level.

D.1. Student roles

The developer can define roles with different levels of complexity (role complexity) as well as wide-ranging roles (multiple roles) for the student. The definition of the roles by the developer influences the depth of the student's understanding. According to Anderson and Naps [1]: "the greater the degree of interaction allowed by the instructional design of the AV system, the greater degree of understanding on the part of the student." There are two sub-categories role complexity and multiple roles.



In the ECPU environment, the student can take several roles: preparing and building the instruction and timing its operation, writing a program, controlling the program running process, writing data in the processor, memory and input. In RTLSim, the

student operates the control unit and synchronizes the data transfer on the data lines by controlling the control lines. In LMC, the student operates the Little Man Computer in the mailroom.

D.1.1. Role complexity

The developer can define the components of roles with a ranking of the level of their complexity to allow the student a graduated process of learning according to his knowledge and skills. The student may be given a succession of roles that are progressively more complex according to his success in fulfilling the previous role or the software may allow him to choose when to move on to a more complicated level.

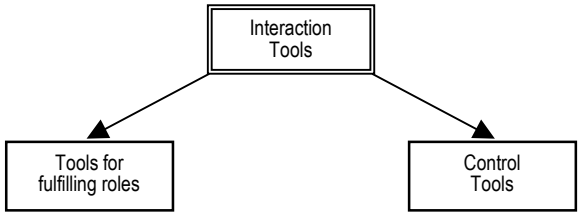
Work in the ECPU environment is divided into two stages: a stage that is intended for beginning students and a stage that is intended for advanced students (basic mode & advanced mode). In the initial stage, the student is given the role of building commands, timing their execution and observing them. In the second stage, the student is given a role as a programmer.

D.1.2. Multiple roles

The developer can give several roles with the purpose of involving the student in several processes and developing several skills. He can ask the student to carry out several functions simultaneously, or to allow him to choose between different functions, or to dictate several functions according to a pre-defined series.

D.2. Interaction tools

The developer gives to the student interaction tools. There are tools designated to enable the student to fulfill the roles assigned to him (tools for fulfilling roles). There are tools designated to control the visualization presentation (control tools). The tools must be appropriated to the roles defined and to the events that the student is supposed to control.



D.2.1. Tools for fulfilling roles

The relation between the role defined by the developer and the tools he provides must be transparent. When experimenting with fulfillment of his role, the student learns the relationship between the actions he performs, using the tools, and the results obtained in various situations. Thus, he will learn the mechanism of operation of the demonstrated process and it will contribute to a deeper understanding of the processes involved.

ECPU gives the student control of output/input as a programmer and as an operator. RTLSim gives the student control over all the internal control signals of the processor and shows the state of the machine as a result of the control. Thus, the student can play the virtual role of the control unit in a MIPS processor, a role that he could only have played thanks to a simulation. In LMC the student can play the role of controlling the program execution as well as the role of programmer.

D.2.2. Control tools

Sometimes, the demonstrated processes are complex and executed too fast for the student to be able to observe. Therefore, the developer gives the student tools to control the timing of the process. Control over timing is significant from the didactic aspect as it may be related to the fulfilling of a role in the demonstrated system.

In ECPU, in the basic mode, the student may control the timing of the transfer of data over the paths, and thus he "advances the processor clock" which in practice is responsible for this operation, and times the cycle of execution of the instruction.

D.3. Integrated learning materials

Learning materials integrated in the environment includes the theoretical material and the written tasks.

The textual material is intended to supply the student with the theoretical background required for the model presented in the visualization environment. The user must be taught how to interpret what he is shown in the visualization [2]. The purpose of the integrated learning material is to instruct and to train the student how to correctly interpret what he is shown in the visualization presentation.

In the activities manual of the ECPU, the student is asked to build a command that calls data from the memory and allows the student to interpret the animation of transferring data that takes place over the lines during the command execution.

5. ECPU IN THE CONTEXT OF THIS TAXONOMY

In this section, we will go into detail on the place of the ECPU environment within the taxonomy just presented. The environment will be described and the relevant category number noted in parentheses.

The objective of the environment from the content aspect is to give insight into the architecture of a computer, the way it works and the execution process of programs. ECPU was developed to support a specific curriculum and target population. The architecture is taught as it appears to a programmer in assembly language (B.3). Studying assembly language is not an objective in itself, rather, it is a means to teach the significance of the elementary operations of a computer, to become familiar with the fundamental components upon which high-level languages are constructed, and to understand basic and compound data types. Other objectives are to understand stages of program development (editing, translation and execution of the computer program) in greater details than can be learned just by programming in high-level languages. These didactic considerations guided the selection of a simplified model of a particular computer (B.1.2). As there was no intention to teach the student the entire instruction set, the environment is limited to a reduced set (B.2).

The ECPU environment supports two modes of operation: a basic mode for beginning students and an advanced mode for advanced

students (D.1). The basic mode presents a visualization of the detailed execution of an individual instruction. Separate windows display the processor, the memory, the basic I/O units and the data path that connects them. The display is a low-fidelity representation of the internal operation of the computer (B.1.2). When a student has mastered the instruction set, he/she is ready to use the advanced mode. The advanced mode display is similar to the basic mode display, except that there is no representation of the data paths (C). Instead, the student is provided with simplified software development tools (D.2.1).

The development of ECPU was accompanied by the development learning materials: a textbook and environment-specific lab activities (D.3).

6. CONCLUSION

The lack of taxonomy for program visualization of computer architecture was the motivation to construct the PV taxonomy partially presented in this paper. The development of a PV environment demands an examination of a wide range of alternatives regarding the scope of operation, the content, the presentation methods and the activity styles. The requirements of the curriculum and target population, the didactic approach, and the limitations of the development and presentation tools must all be taken into account. I hope that future developers will find in this taxonomy some guidelines to design visualization environments.

7. ACKNOWLEDGEMENTS

I would like to thank my advisors Mordechai Ben-Ari and Tommy Dreyfus for their valuable help and support. I would like to note the contribution to this paper of the work done with William Yurcik and Murray Pearson.

8. REFERENCES

[1] Anderson, J. M. Naps, T., L. A context for the assessment of algorithm visualization system as pedagogical tools, *Proceeding of the first Program Visualization Workshop*, University of Joensuu, Finland, 2001, 121-130.

[2] Ben-Ari, M. Program Visualization in Theory and Practice, *Informatik .Informatique* , *2,* 2001, 8-11.

[3] Cassel, L., Kumar, D., Bolding, K., Davies, J., Holliday, M., Impagliazzo, J., Pearson, M., Wolffe, G. and Yurcik., W., Distributed Expertise for Teaching Computer Organization and Architecture *(ITiCSE 2000 Working Group Report)*, *ACM SIGCSE Bulletin 33 (2)*, 2001, 111-126.

[4] Hundhausen, C.D., Douglas, S.A., & Stasko, J.T. (2001). A Meta-Study of Algorithm Visualization Effectiveness. Under review by the *Journal of Visual Languages and Computing*, available at http://lilt.ics.hawaii.edu/%7Ehundhaus/writings.

[5] Myers, B., A., Taxonomy of visual programming and programming visualization. *Journal of Visual Languages and Computing*, *1(1),* 1990, 97-123.

[6] Pearson, M.W., McGregor, A.J. and Holmes, G.. Teaching computer systems to majors: A MIPS based solution. *IEEE Computer Society Computer Architecture Technical Committee Newsletter,* 1999, 22– 24.

[7] Petre, M. Blackwell, A. and Green, T. Cognitive questions in software visualization. In Stasko, J., Domingue, J., Brown, M. and Price, B. (eds.) *Software Visualization*, MIT Press, 1998, 453-480.

[8] Price, B. A. Baecker, R. M. and Small, I. S. A Principled Taxonomy of Software Visualization, *Journal of Visual Languages and Computing*, *4*, 1993, 211-266.

[9] Price, B., A. Baecker, R., M and Small, I. An introduction to software visualization. In Stasko, J., Domingue, J., Brown, M. and Price, B. (eds.) *Software Visualization*, MIT Press, 1998, 3-34.

[10] Roman, G.-C. and Cox, K. A taxonomy of program visualization systems. *Computer* 26 (12), 1993, 11-24.

[11] Stasko, J. and Lawrence, A. (eds.). (1998). Empirically assessment algorithm animation as learning aids. In Stasko, J., Domingue, J., Brown, M. and Price, B. (eds.) *Software Visualization*, MIT Press, 1998, 419-438.

[12] Yehezkel, C. Yurcik, W. and Pearson, M. Teaching computer architecture with a computer-aided learning environment: state-of-the-art simulators, *Proceedings of 2001 International Conference on Simulation and Multimedia in Engineering Education (ICSEE),* Society for Computer Simulation (SCS) Press, 2001.

[13] Yehezkel, C. Yurcik, W., Pearson, M., Armstrong, D. Three Simulator Tools For Teaching Computer Architecture: EasyCPU, Little Man Computer, and RTLSim, *Special Issue on General Computer Architecture Simulators, Journal on Educational Resources in Computing (in press)*.

[14] Yurcik, W. Vila, J. and Brumbaugh, L. A web-based little man computer simulator, *Proceedings of SIGCSE '01*, 2001, 204-208.