NISTIR 89-4196

VOILÁ: A SYSTEM FOR LOOKING AT PROCESSES

Sanford Ressler
Stephen Nowland Clark

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Center for Manufacturing Engineering
Factory Automation Systems Division
Gaithersburg, MD 20899

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Raymond G. Kammer, Acting Director

NIST

NISTIR 89-4196

VOILÁ: A SYSTEM FOR LOOKING AT PROCESSES

Sanford Ressler
Stephen Nowland Clark

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Center for Manufacturing Engineering
Factory Automation Systems Division
Gaithersburg, MD 20899

October 1989

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Raymond G. Kammer, Acting Director

Voilá: A System for Looking At Processes

Sanford Ressler
Stephen Nowland Clark
National Institute of Standards and Technology
Gaithersburg MD 20899

Introduction

The ability to visualize data is becoming an increasingly important research field in its own right [18]. How do scientists present, view, and communicate the information contained in data, systems or programs in meaningful ways? This paper presents one approach to the problem of visualization in the context of illustrating and observing a control system. Specifically, the application domain is a real-time control system (RCS) developed at NIST. This system allows a control system to be specified as a hierarchy of finite state machines. Although it was developed primarily for the control of factory floor applications, the RCS has been applied to many other types of systems as well, including multiple autonomous underwater vehicles, the strategic defense initiative, and telerobotic arm control. Integration into the existing control systems development environment was a key goal of the visualization project.

A key concept in *Voilá* is that of varying "levels of detail." Quite often, a system and the data it generates comprise too much information to be easily observed in one "screenful." Mechanisms to filter out and display only the interesting pieces of information must be created. This is nothing new: for many years, systems have been created which filter data (providing abstraction, or a low level of detail about a mass of data) or focus it (providing a high-level of detail about a small portion of the data) in various specific ways. Our approach was to make a flexible levels of detail paradigm an inherent part of the system.

The implementation environment was a strong influence in the design of the system. *Voilá* is implemented in the Smalltalk-80 programming language/environment [8]. We chose Smalltalk because of the inherent extensibility of its environment and the large number of high-level tools which already exist in that environment. We also viewed the object-oriented programming paradigm, of which Smalltalk is the seminal example, as a natural fit for our approach to visualization.

Related Work

Quite a number of systems to visualize algorithms and processes have been implemented by others [7,9,10,15]. The two key differences between our approach and that taken in previous systems are the level of detail concept mentioned above and the goal of integration into an existing environment. Thus, we needed to provide a visualization framework which could be integrated into an existing environment without requiring any modifications to applications in the environment. The proposed visualization methodology takes advantage of the data abstraction mechanisms available in object-oriented programming to ease this integration task.

BALSA [4] is a system to create animations of algorithms. Designed as an educational aid, particularly in the study of algorithms, it has been used to produce many imaginative and interesting animations. However, the requirement that the algorithm to be animated be coded directly into the system becomes a serious drawback if we leave BALSA's intended domain and instead try to visualize existing programs in the real world.

The generation of pictures from program text is analogous to the pretty printing of code, the best example of which is the work of Marcus and Baecker [13]. Their work examines C code from the perspective of a graphic designer, and has resulted in a system which prints C code in a very readable and graphically pleasing way. *Voilá* can generate state graphs in an analogous manner by parsing the state machine code and generating a graph based on it.

The work of Robert Jacob at the Naval Research Labs [11] is another example of a visualization system for state machines. His system approaches the problem from the point of view of visual programming. It allows a user to draw a state graph and associate code with each transition. Executable code for the specified state machine can then be generated. Clearly this is one direction in which *Voilá* can and should be extended. A visual programming front end to a high-level emulation tool is indeed a future goal of the project.

Another class of interesting related work is in the area of the display of data from a debugging point of view. For example, Gdbxtool [12], an extension of a window-based C debugger, displays C data structures graphically. The user defines graphical templates for the structures which are later "filled in" with the actual values of variables. In the Smalltalk world, the work of Cunningham and Beck [5] has gone a step further. They have written a debugger which automatically creates diagrams illustrating the flow of messages from one object to another. The display is animated to illustrate the order of message passing, giving the programmer a feel for the flow of control.

The INCENSE [14] system is very interesting in its approach to dealing with the problem of too much information to be displayed. It is designed to automatically generate displays for a variety of data structures. In addition, INCENSE includes support for managing screen usage, allowing displays to change in response to the amount of space allocated to them. This type of intelligence could be added to *Voilá* by adding functionality to the class Layout, which deals with the physical arrangement of related presentations. Indeed this was one of the original intents for the creation of the Layout class and remains an area for further development.

Levels of Detail

What exactly do we mean by "levels of detail"? As indicated above, in different situations a user may wish to view differing amounts of detail on a given piece of data. *Voilá* provides for a hierarchy of display formats, or presentations. Each level in the hierarchy corresponds to a level of detail; thus, as we traverse the hierarchy from top to bottom, we find increasingly detailed presentations. For a particular object, we may choose an iconic presentation or a more fleshed out view which indicates some of the internal structure of the object or its value; this selection can be changed at will. This allows the user to the amount of detail given on a piece of data to change with his level of interest in that data. One can imag-

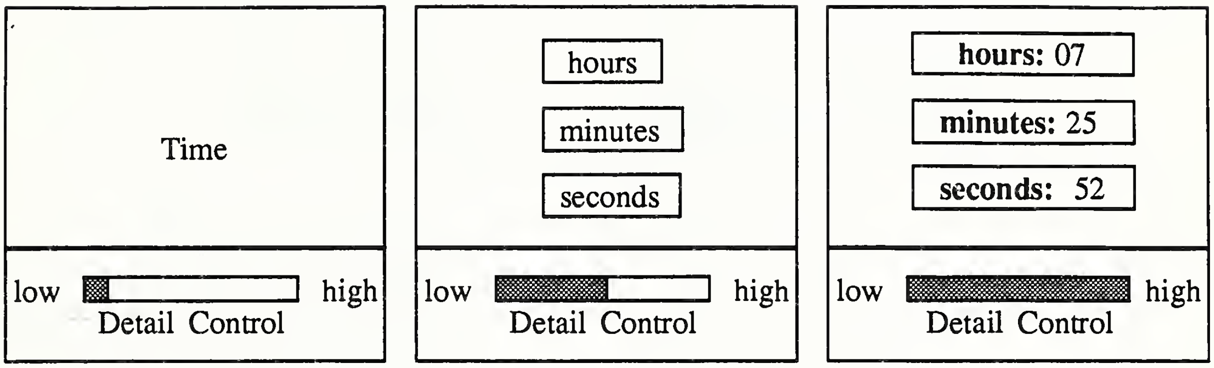

Figure 1: Hypothetical Detail Control Bar

ine a slider type control bar (Figure 1) or dial which the user could use to move between low and high-levels of detail. Although we did not implement such a control bar, one can easily imagine the usefulness of this concept in the context of a highly responsive system.

Visualizing an Application: Real-time Control Systems

The particular domain we were involved in was real-time control systems (RCS)[1]. Before getting into the details of *Voilá* as it relates to RCS, let's briefly review the concepts involved in hierarchical control. An RCS is an architecture for decomposing the functionality of a system into manageable parts. The decomposition is done in a hierarchical way to allow parallelism to be exploited and to create clean interfaces at the various control levels. For example, in the context of a factory control system, a cell process will command a variety of workstations. Each workstation controller will in turn command equipment level processes (robots). Typically, commands flow down the hierarchy and status information flows back up to synchronize the various processes. At NIST one way that we've implemented these control systems is to emulate each process as a finite state machine (FSM). Each control process in the hierarchy is represented as an FSM which communicates with the other processes via a mechanism known as Common Memory. Common Memory is the conceptual medium through which commands and status information flow.

The Hierarchical Control System Emulator

The Hierarchical Control System Emulator (HCSE) [3,6] is a high-level software tool which runs on a VAX under VMS and is used by programmers to create prototype hierarchical RCS's. HCSE code is written using a special-purpose language which is used to specify the behavior of an FSM (with condition-action pairs) and its connections to other state machines (via Common Memory). Arbitrary code can be executed using the Praxis language, which may be embedded in an FSM directly, or other languages, which can be called

```
//output        TOLERANCE_DS_SVCds_request_mbx|Request mailbox
//input         DS_TOLERANCE_RSPds_response_mbx|Response
mailbox


//conditions    curs = "GETTING TOLERANCE DATA";
                TOL_DESCR.TYPE = "EX"
//actions       nexts := "GETTING DRF NAME"
                data_server_request(TOLERANCE_DS_SVC,
                                    DS_last_request,
                                    "DRF_NAME",
                                    DS_DRFName,
                                    PART_NAME,
                                    TOLERANCE_NAME)
```

Figure 2: Sample FSM code

through more round-about mechanisms(Figure 2) .

Although the HCSE provides a number of analytical capabilities which may be valuable sources of data in future visualization efforts, in our current system we are primarily interested in the log file produced by the HCSE (Figure 3). This log file simply records each change in Common Memory, and thus captures the entire history of a particular emulation

```
0:0:4.20    SDCC_CURS  LoopRv
0:0:4.30    SDCC_CURS  PosWait
0:0:4.50    SDCC_CURS  Range
0:0:4.60    SDCC_CURS  AssReg
0:0:4.70    SDCC_CURS  LoopRv
0:0:4.80    SDCC_REG2_CD  COMMAND  GO_REG2
                          CMD_SEQ_NBR  1
0:0:4.80    SDCC_REG1_CD  COMMAND  GO_REG1
                          CMD_SEQ_NBR  1
0:0:4.80    SDCC_CURS  Reg1StsWait
0:0:4.90    SDCC_REG1_STS  STATUS  No Rv to Region 1
                           STS_SEQ_NBR  1
0:0:4.90    REG2_FARM3_CD  COMMAND  GO_REG2FARM3
                           CMD_SEQ_NBR  1
0:0:4.90    REG2_FARM2_CD  COMMAND  GO_REG2FARM2
                           CMD_SEQ_NBR  1
0:0:4.90    REG2_FARM1_CD  COMMAND  GO_REG2FARM1
                           CMD_SEQ_NBR  1
0:0:4.90    REG2_CURS  R2F1StsWait
0:0:5.00    SDCC_CURS  Reg2StsWait
0:0:5.00    R2F1_CURS  DoIt
0:0:5.00    R2F2_CURS  DoIt
```

Figure 3: Portion of HCSE Log file

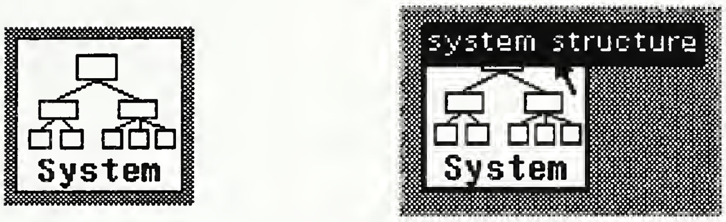

Figure 4

run. Unfortunately (or perhaps fortunately), this history is recorded in excruciating detail, and the typical log file quickly becomes unwieldy. Thus, it seemed ideal to use *Voilá* to manage the viewing of log files.

The log file contains the raw data needed to recreate the course of events in an emulation, but provides no information concerning the structure of the control system or of the individual state machines. We decided to get information about specific state machines directly from the FSM source code. This approach proved to be a useful diagnostic tool in and of itself, pointing out unknown bugs in some of our HCSE code.

Presentations and Layouts

Since the purpose of *Voilá* was to create and manage presentations, it was clear that the concept of a presentation would be central to the system. There is certain functionality which is associated with all types, or classes, of presentations, which is accessible via the middle- and right-button menus. The middle button is used to traverse the presentation hierarchy. Pressing it while the mouse is inside a presentation pops up a menu of all the possible presentations at the next level of detail, filtered according to the type of the data being displayed. When a selection is made, the current presentation is replaced by a new presentation of the indicated type. The right-button menu provides general window-manipulation functions, such as moving, closing, and collapsing to an icon. The left-button menu is reserved for the use of individual presentation types. An example of this is the layout menu of a structure presentation.

The physical relationships between several logically related presentations are captured by a *layout.* The layout concept is not yet very well developed, currently serving primarily as a convenient time-saver. A layout can be used to record, name, and later recreate the physical configuration of a set of related presentations. Future plans include the addition of intelligent layouts capable of responding to high-level instructions such as "lay out in a circle" and perhaps of using context information to help determine a precise physical layout.

Control System Presentations

Let us now turn to the specific topic of illustrating a real-time control system. At first, we may simply represent the control system with an icon. This simple presentation is the root of the presentation hierarchy for any object in *Voilá*. We can open up the icon by selecting a presentation at the next level of detail from the middle mouse button menu (Figure 4) . In this instance there is only one menu item, as there is currently only one possi-

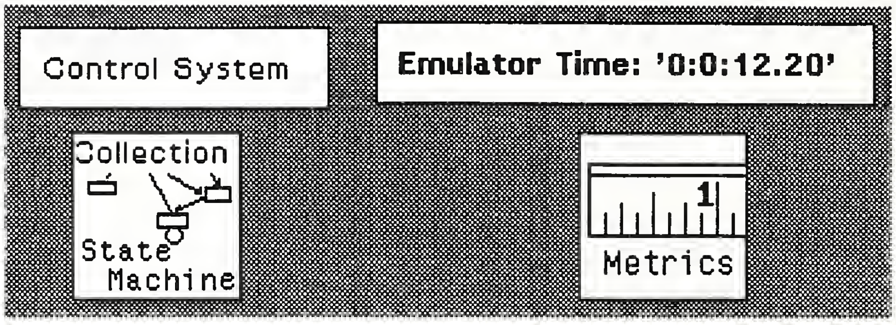

Figure 5

ble presentation for a control system at the next level of detail. In general, however, this menu may contain several options.

This new presentation illustrates the contents of a control system relatively superficially (Figure 5): we see that it consists of the emulated time, a collection of state machines, and some metrics. "The Control System" box is control box which allows us to manipulate these three other presentations as a single unit, for example to close the entire presentation or collapse it back to an icon. A separate control box is necessary in this type of situation (when we have a presentation consisting of several other presentations) because each presentation is an independent entity, giving no clear mechanism for manipulating a group of them together.

The physical arrangement of the four presentations in Figure 5 was predefined as the default layout for a system structure presentation. We can proceed to create new layouts by moving various presentations around on the screen, using the "move" function from the window control (right button) menu, and perhaps by opening out some of the component presentations to greater levels of detail (with the middle button menu). A particular arrangement of the component presentations can be named and saved for later use by selecting the "save layout" option from the left button menu of the control box (Figure 6). Similarly, a presenta-

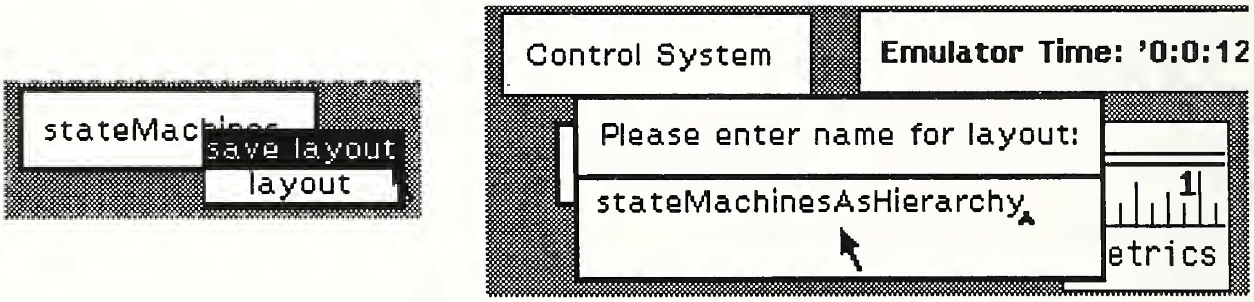

Figure 6

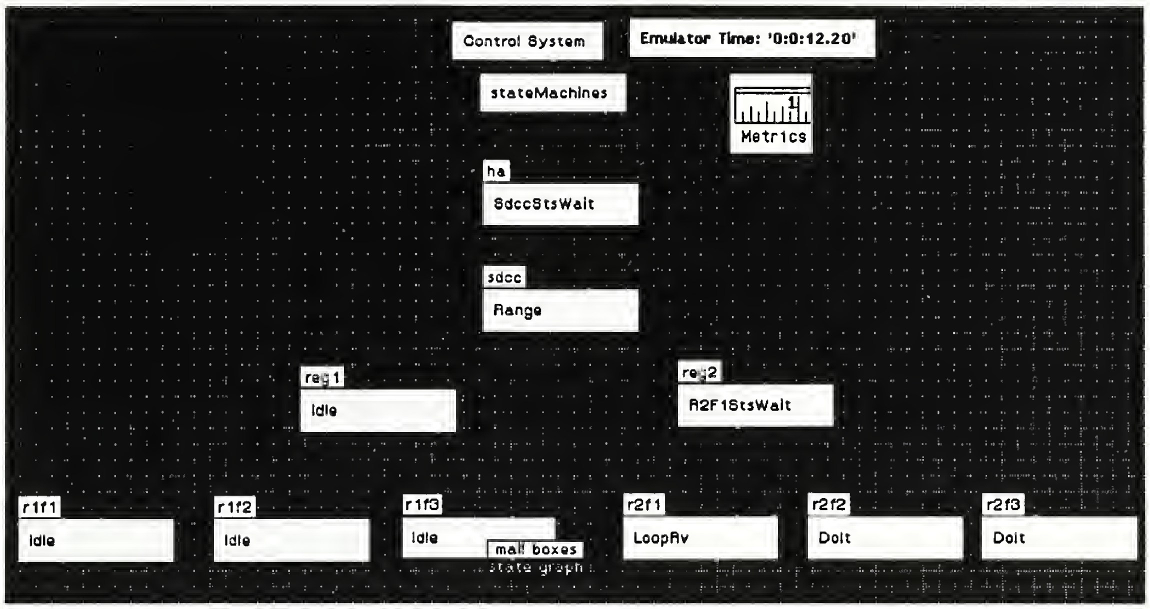

Figure 7

tion can be arranged using a previously saved layout by selecting the "layout" option from this menu.

As we continue asking for more detailed information, we can view the contents of the collection of state machines as a control hierarchy (Figure 7). In this level of detail, each box with a label on it represents an individual state machine. The name of the state machine is on the label and its current state is displayed inside the larger box. If we choose to "play back" an emulation by reading the log file produced on the VAX by the HCSE, we can watch the current states change and the flow of control can be observed. In a typical hierarchical RCS, one state machine will send a command down to its subordinate(s), which will perform some actions and eventually send a status message back. Using *Voilá*, one can literally watch the activity flow down the hierarchy and then see it percolate back up.

We can look inside a state machine by again using the middle mouse button to open up one of the boxes representing an FSM. In this case we can look at the state machine as a state graph or as the set of mailboxes through which it communicates with the other FSMs. If we choose the state graph view, we get a display of the state graph for the particular machine (Figure 8). The graph is generated directly from the HCSE code used to implement the FSM on the VAX.

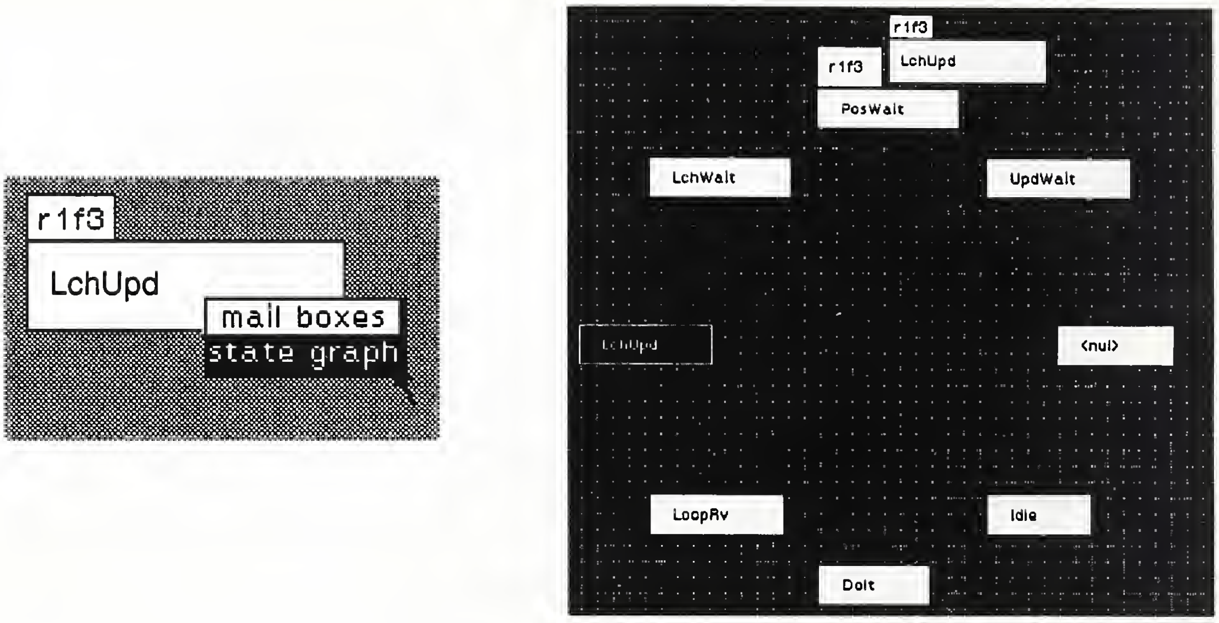

Figure 8

We hope that future developments in layouts can be applied to the state graph presentation, as the current graphs are often difficult to read. In the mean time, being generated directly from the FSM source code, the graphs have already proved to be valuable for quickly checking the code's correctness in implementing an FSM. They provide an easy way to spot certain kinds of problems, such as states with no incoming transitions.

Continuing our procession to greater levels of detail, we can now place the cursor on a transition in the state graph and open it up to display the actions or conditions used to define the transition in the source code (Figure 9). At this point, it is interesting to note that the lines which form the transitions are actually windows. We have implemented, in Smalltalk, a facility to create arbitrarily shaped windows, which are pickable only within an arbitrary area.

Figure 10 illustrates the control system from a user's point of view. The user may "open up" the various presentations to greater and greater levels of detail. Nodes in the figure correspond to presentations which the user may request. Arcs represent the paths which may be taken to view the various presentations.

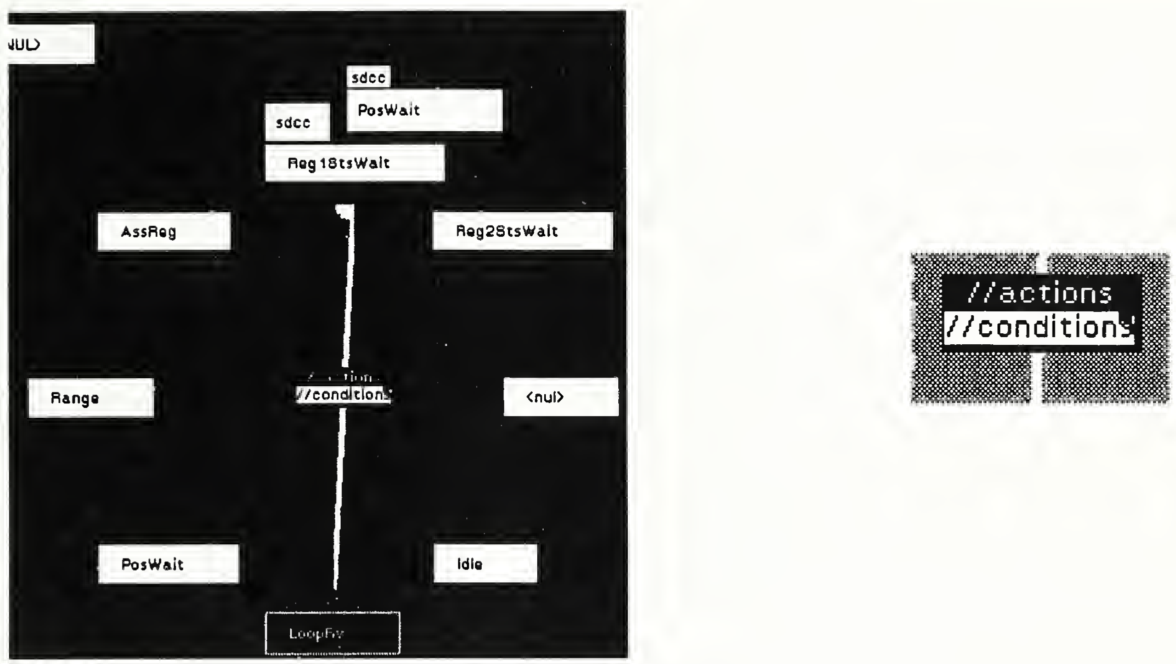

Figure 9

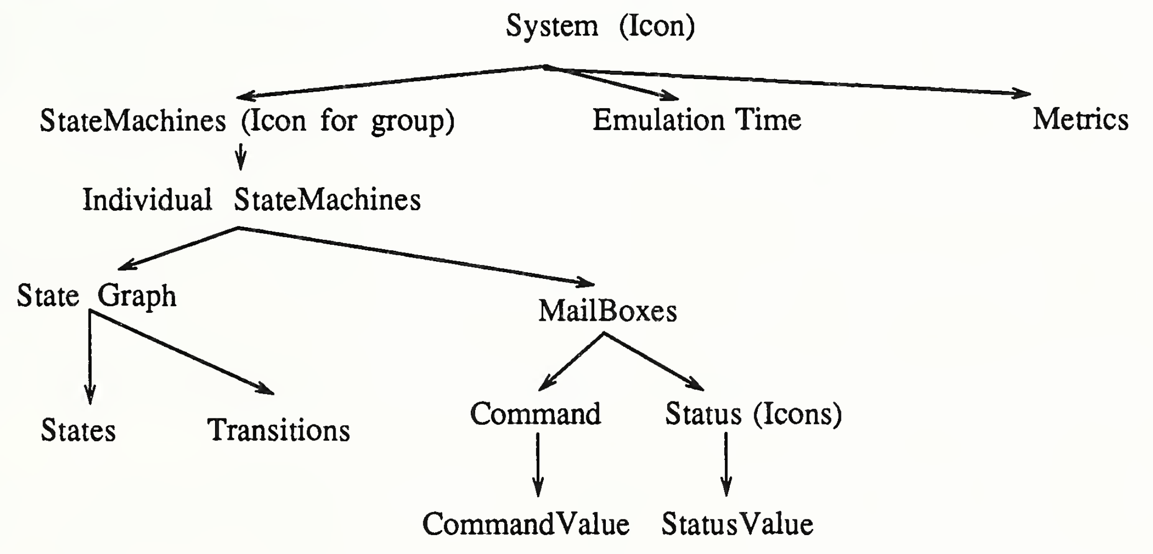

Figure 10: Complete presentation hierarchy for a control system

Activating Processes

So far, we have dealt only with the user interface to a static presentation. We now turn to the problem of animating these displays, binding them to entities in the "real world." We need a mechanism for updating a presentation whenever its model (the object being presented) changes. There are two ways of doing this. One is simply to execute some Smalltalk code which essentially follows a script, updating various presentations along the way. This has the advantage of being the fastest way of animating the presentations, as it has very little overhead, but it also has the limitation of not allowing any user interaction during this code execution. Another, more robust, mechanism is to start up a separate Smalltalk process which continually polls the "real world" entities and issues updates to the Smalltalk objects representing them. This process is managed by an entity called a Watcher.

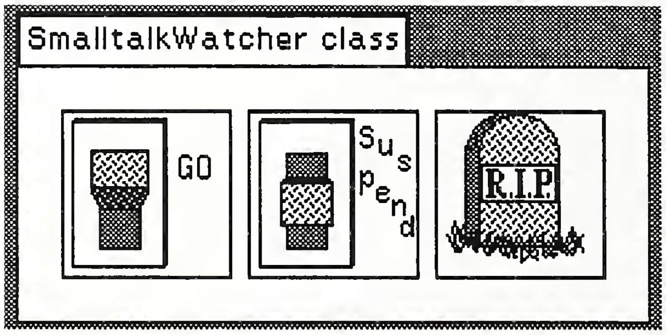

Figure 11

The interface to a Watcher consists of a three button menu (Figure 11). The functions available are: open a new presentation; suspend the watcher process (this vastly improves performance in user interaction, and is particularly useful when many layout changes are needed); and kill the Watcher, together with any presentations it controls. When requesting to start a process the user enters Smalltalk code which is the interface between the external "real world" (e.g., a log file) and its internal representation.

Benefits of the Object-Oriented Approach

The object-oriented approach taken in *Voilá* gives us several benefits. When designing a new presentation, a user can take advantage of inheritance. This allows an existing presentation to be used as a template for the new presentation, thus reusing the code written for the latter. If this approach does not seem appropriate, a new presentation may be created by plugging together several existing presentations, as in the control system presentations described above.

The inheritance mechanism also gives us a very useful defaulting scheme. For example, an icon is used as the default root-level presentation for all types of information. This default mechanism can easily be overridden if a different presentation is desired for a particular type of data.

Visualization as Analysis

Illustrations can be much more than merely pretty pictures and a good demo. If the illustration really conveys some information it can be used as a valuable analytic tool. Educators and imaginative computer graphics people have known this for a long time[17]. The seminal film "Sorting Out Sorting" [2] and the animations by Jim Blinn in the PBS series "The Mechanical Universe" are certainly great examples of conveying complex information via graphics. "Sorting Out Sorting" illustrates the differences between a dozen different sorting algorithms and their effect on data as it is sorted. One can imagine a real-time "sorting out sorting" analysis program which benchmarks performance, illustrates bottle-necks, and allows for graphical programming to correct bugs. A state graph which shows a state with no arrows (transitions) going into it also points out a bug in the state machine. If you can't get to a state than it needn't be there. We would also like to be able to add the missing transition(s) and generate the corresponding state machine code.

The best visualizations do not come from general-purpose tool kits or libraries. Rather they are carefully crafted scenes tailored to the particular application. Good visualizations have a great deal of implicit "knowledge" about the semantics of what is being visualized. This is a tremendous conflict for systems developers. Perhaps the best we can hope for is a robust collection of tools and utilities with a clear methodology for linking these tools to the semantics of an application.

Conclusions and Acknowledgments

The ability to examine objects in a controlled way which connects presentations in a levels of detail oriented manner has proven to be useful. Complex systems may be viewed and interactively browsed to get a "feel" for their structure. There are quite a number of areas for a large amount of improvement. The manner of dealing with collections or structures containing several components lacks coherence. The layouts should have more intelligence, and the use of shape grammars [16] is probably a reasonable approach. The treatment of each presentation as an individual window is a real performance problem, and, more importantly, does not adequately address the problems of larger views which require scrollable displays.

The authors would like to thank Ted Hopp for his initial implementation of the control system and state machine structures and his general Smalltalk enlightenment. Tina Lee implemented the HCSE code and modified the log file for our usage. And finally, Anne Litcher was brave enough to actually try using this stuff.

References

[1] Albus, J.S., et. al., "A Control System for an Automated Manufacturing Research Facility", *Proceedings Robots 8 Conference and Exposition*, Detroit, MI, June 1984.

[2] Baeker, R., "Sorting Out Sorting", film The Dynamic Graphics Project at the University of Toronto, 1981

[3] Bloom, H.M., Furlani, C.M., and Barbera, A.J, "Emulation as a Design Tool in the Development of Real-Time Control Systems", *Winter Simulation Conference*, Dallas, Texas Nov. 1984.

[4] Brown, Marc H., Sedgewick, Robert, "A System for Algorithm Animation" *Computer Graphics* Vol 18, Number3 July 1984, Siggraph Proceedings

[5] Cunningham, Ward and Beck, Kent "A Diagram for Object-Oriented Programs" *OOP-SLA 86*, SIGPLAN Vol 21, Number 11 Nov. 1986 pp 361-367

[6] Furlani, Cita M. (Editor), "Hierarchical Control System Emulation User's Manual" *National Bureau of Standards NBS-IR-85-3156*

[7] Furness, G. "Generalized fisheye views. " *CHI '86 Conference on Human Factors in Computing Systems* (Boston Mass, Apr. 14-18) ACM/SIGCHI, New York, 1986 pp. 16-23

[8] Goldberg, A.J., and Robson, D. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley, Reading, MA, 1983

[9] Halasz, Moran, Trigg, "NoteCards in a Nutshell" *CHI '87 Conference on Human Factors in Computing Systems* (Toronto, Canada, Apr. 5-9) ACM/SIGCHI, New York, 1987 pp. 45-52

[10] Henderson, D. and Card, Stuart K., "Rooms: The Use of Multiple Virtual Workspaces to Reduce Space ontention in a Window-Based Graphical User Interface" *ACM Transactions on Graphics* Vol 5, Number3, July 1986

[11] Jacob, Robert J.K. "A State Transition Diagram Language for Visual Programming" *IEEE Computer*, 18(8) 51-59

[12] Potrebic, Peter and Goldman, Phil "A Debugger-based System for Graphical Display and Editing of Data Structures" *Summer 1987 USENIX Conference Proceedings* 147-158

[13] Marcus, Aaron and Baecker, Ronald "On the Graphic Design of Program Text", in *Graphics Interface 82 Conference Proceedings* (Toronto, Canada May 17-21) pp 303-311

[14] Meyers, Brad A., "INCENSE: A System For Displaying Data Structures" *Computer Graphics* Vol 17, Number3, July 1983, Siggraph Proceedings

[15] Smith, R.B. "The Alternate Reality Kit: An Animated Environment for Creating Interactive Simulations." *Proceedings of the 1986 IEEE Computer Society on Visual Languages*. (Dallas, June 1986) pp 99-106.

[16] Stiny, George, "Introduction to Shape and Shape Grammars", *Environment and Planning* B, 7:(1980) p. 343.

[17] Tufte, Edward R. *The Visual Display of Quantitative Information*, Cheshire, Conn.: Graphics Press 1983

[18] Visualization in Scientific Computing , *National Science Foundation report in Siggraph's Computer Graphics*, Volume 21, Number 6 Nov. 1987

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** *(See instructions)*	1. PUBLICATION OR REPORT NO. NISTIR 89-4196	2. Performing Organ. Report No.	3. Publication Date OCTOBER 1989

4. TITLE AND SUBTITLE

Voilá: A System for Looking at Processes

5. AUTHOR(S)

Sanford P. Ressler and Stephen Nowland Clark

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

The ability to visualize data is becoming an increasingly important research field in its own right. This paper presents one approach to the problems of visualization in the context of illustrating and observing a control system. Specifically, the application domain is a real-time control system (RCS) developed at NBS primarily for the control of factory floor applications. The user interacts with the system via multiple levels of detail selecting only those portions of the system the user wishes to view. The implementation environment for the system was primarily Smalltalk-80 which provided a high degree of extensibility and flexibility. Integration of existing control system development tools into a visualization environment was a key goal of the project.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

computer graphics, programming environments, Smalltalk, state machines, user interface, visualization

13. AVAILABILITY

[✓] Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

[✓] Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

15

15. Price

A02

USCOMM-DC 6043-P80