A SOFTWARE ENGINEERING, VISUALIZATION METHODOLOGY FOR PARALLEL PROCESSING SYSTEMS*

James Arthur Kohl: kohl@eng.uiowa.edu and Thomas L. Casavant: tomcw@eng.uiowa.edu

Parallel Processing Laboratory, Department of Electrical and Computer Engineering
University of Iowa, Iowa City, IA 52242

Abstract

This paper focuses on techniques for enhancing the feasibility of using graphic visualization in analyzing the complexities of parallel software. The central drawback to applying such visual techniques is the overhead in developing analysis tools with flexible, customized views. The PARADISE (PARallel Animated Debugging and Simulation Environment) system, which has been in operation since 1989, alleviates some of this design overhead by providing an abstract, object-oriented, visual modeling environment which expedites custom visual tool development. PARADISE is a visual tool which is used to develop other visual tools, or a meta-tool. This paper complements previous work on PARADISE by describing the philosophy behind its design, and how that philosophy leads to a methodology for constructing visual models which characterize parallel systems in general. Emphasis will be on the crucial issues in utilizing visualization for parallel software development, and how PARADISE deals with these issues.

Keywords: software engineering, parallel processing, program visualization, object-oriented design, debugging, performance enhancement, computer-human interfaces.

1. Introduction

Program visualization [32] has become an effective alternative for dealing with the complexities of parallel computing, yet it remains highly experimental in nature and has not proliferated extensively into common use. To date, many techniques have been successfully, though experimentally, explored, but the high development time in using most systems renders them infeasible for practical usage. One apparent fact is that greater visualization flexibility requires a more complicated user interface for specifying and controlling views.

As a result, a majority of visualization environments have emphasized simpler, more automatic view generation. This approach reveals sufficient information about parallel program behavior to be worthwhile and useful. However, there are many advanced visualization capabilities which are ignored because of specification complexity. Sophisticated visual techniques require more flexible, powerful, and user-friendly interfaces to be effectively applied. Dealing with the complexities of the underlying graphics systems and data analysis computation is not the concern of the user, and these details should be hidden via high-level abstractions. The full benefits of intuitive visual representations will not be realized until the user is able to manipulate visual analysis tools with a minimum of effort.

It is the goal of PARADISE (PARallel Animated Debugging and Simulation Environment) [21,22] to provide an efficient, visual framework for analyzing parallel software. PARADISE was constructed to address some of the burden of specifying visual analyses by introducing a formal and abstract approach to visual modeling. Though PARADISE does not solve all of the important problems, it takes a step toward more powerful visualization techniques. PARADISE uses modular, visual models to depict the behavior of user applications or systems. The goal of this model abstraction approach is more efficient organization and specification of custom visual analyses. Visual models are constructed by combining independent objects which represent the behavior, either literally or conceptually, of corresponding portions of the application or system. Behavior is animated in the model using a flow of discrete occurrences, or events, which are input from an external event stream and processed by the objects in the model. The structure of the event flow among the objects is visually specified using interaction port and interaction link abstractions.

The following section overviews current and previous work in visualization for parallel software development. Subsequent sections describe in more detail the philosophy behind the design of PARADISE, along with a methodology for approaching program visualization for analyzing parallel systems. The focus will emphasize the extent to which this particular approach addresses the critical issues in visualization of parallel software. In the last section, some experiences in developing PARADISE, along with limitations of the approach, will be discussed, and the insight gained will be applied to propose directions for future work. Due to space limitations, many details have been omitted. A complete discussion can be found in [20].

2. Background

There have been many attempts to apply the power of visual analysis to assist in developing parallel software. Visualization tools have been developed to provide a wide spectrum of approaches which range from highly-automatic to highly-custom [23]. A brief summary of these approaches is given in Table 1.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Types of Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>Data Structures, Flow Graphs</td>
</tr>
<tr>
<td>Statistical</td>
<td>Charts, Graphs, Diagrams</td>
</tr>
<tr>
<td>Execution Histories</td>
<td>Colored Time-Lines</td>
</tr>
<tr>
<td>Standard Animations</td>
<td>Mobile Icons, Memory Patterns</td>
</tr>
<tr>
<td>Custom Animations</td>
<td>Arbitrary Images, Literal Views</td>
</tr>
</tbody>
</table>

Table 1: Visualization Approaches

Many tools have taken the automatic approach, and have provided visual systems which produce a variety of displays with minimal user specification. These displays are often fixed or restricted, but typically allow simple view adjustments or selections by the user. Tools have been constructed for the automatic display of data structures [9,36] and program dependency- and call-graphs [15,24,30,35], and visual programming tools produce automated displays as well [6,11,19,37].

Several more complicated tools produce a variety of automatic visual graphs, plots, and charts, such as Kiviat diagrams [31], for displaying statistical information from program monitoring data. The ParaGraph tool [17] provides a number of displays for viewing statistics on processor status and communication in a multiprocessor system. Other similar tools are found in [4,16,33].

Another category of visualization tools uses very simple, yet powerful, colored time-lines to visually represent execution histories for multiple parallel processes and processors [2,15,18,24,27,33,35]. These time-lines show sequences of states for each process or processor along a common time axis, and are often referred to as Gantt Charts [14].

With more sophisticated visual views and analyses as those provided by tools in the standard animations category, a trade-off is made between view flexibility and interface complexity. Systems such as TOPSYS [5] and SHMAP [10] produce useful, dynamic, animated views through a fixed set of visual abstractions.

On the other end of the spectrum, some tools are very flexible and produce arbitrary custom animations of algorithm behavior [5,7,25]. These tools utilize many visual representations, but each view can require substantial development of new graphics driving software.

Several visualization projects have attempted to alleviate the higher development costs for use of sophisticated graphics by introducing abstractions and providing more efficient user interfaces. TANGO [39] uses several graphical abstractions to
reduce the complexity in constructing animations. The Q+ tool [13] supports a flexible visual modeling environment based on networks and transactions.

Two other systems have made progress in providing the user with a flexible means for graphically manipulating and displaying data. TRIPLEX [8,33] provides a collection of tools for capturing program execution traces, organizing and filtering the raw trace data, and displaying the data in configurable views. Views are constructed by visually programming a schematic of data filters and display types. HYPERVIEW [28,33] and PABLO [34] also allow configuration of displays through a high-level filter interface.

The PARADISE tool fits among TANGO, TRIPLEX, HYPERVIEW, and PABLO in the more complex end of the visualization spectrum. Where TANGO requires instrumentation of the user application with animation directives, PARADISE utilizes generic program state information obtained through a strictly bounded interface to guide animations via an independent visualization model. However, PARADISE lacks in absolute animation power, such as the free abstract animations provided by TANGO, and is limited to immobile objects with dynamic images. PARADISE provides more sophisticated animations than TRIPLEX, HYPERVIEW, and PABLO, but at a higher cost with respect to user interface complexity. PARADISE cannot completely specify views using pull-down menus or visually arranged filters, as textual specification is necessary for defining the behavior of the objects in the visual model. PARADISE also lacks some of the data organization features of these tools. One advantage of PARADISE is that it applies equally well to all machine architectures, and visual models may be constructed to be either general or architecture specific.

3. Detailed PARADISE Environment Description

PARADISE provides all fundamental visualization tool features and operations intrinsically, freeing the user to focus only on the specification of the details of particular animated views. The following discussion will focus on the internal event processing, the approach to visual modeling, and will further examine the abstract, object-oriented, visual nature of the tool environment. Also included will be a brief discussion of the actual specification of models and their use in analyses.

3.1. Discrete-Event Simulator

Because visualization systems typically utilize some form of monitoring or execution tracing, PARADISE is designed specifically to manipulate event streams, and is built with a generic discrete event simulator as its foundation. The use of program annotations as used in the TANGO system [39], though a straightforward approach to animation specification, violates the strict "firewall" between application and visualization, increasing complexity and inadequately separating concerns. The use of relational databases, such as by Snodgrass [38], provides powerful data processing and organization features, but does not directly assist in the generation of views or animation. The modeling capabilities and event manipulation of discrete-event simulation lie between these other approaches.

The high level layered structure of PARADISE is shown in Figure 1. The user definitions for the objects and their associated graphical images build on a generic "simulator skeleton" at the core of PARADISE. The next-state routines which define the behavior of each object are accessible to this simulator skeleton. The simulator applies each object's routine in turn, with the appropriate parameters for the given object instance, then exchanges any events that have been queued for interaction.

An animated simulation in PARADISE proceeds in two distinct phases. In the first phase, each object's local functionality is "computed" independently, and in the second phase, the "results" of this computation are exchanged among the other objects (within the confines allowed by the model structure). This two-phase cycling of functionality and interaction constitutes a "synchronous" simulation of system behavior. This approach is clearly effective for truly synchronous systems, but also applies to asynchronous systems. An asynchronous system will be properly modeled, given an accurate event stream, as long as the order of occurrence of events is correctly maintained during simulation. This can be accomplished by selecting an appropriately small time slice for each simulation cycle to insure that event order is maintained.

PARADISE controls this consistency by dynamically selecting time slices for each simulation cycle. The size of a given time slice depends on the time-stamps of currently pending events and object actions. At any time during a simulation, the next instant in time to simulate corresponds to the smallest pending time-stamp or object service time. (Note that each object must specifically notify PARADISE of the time for its next required service or action, as object state changes may occur without direct influence from events.) This scheme clearly distorts the true flow of time in a simulation, as time slices can become arbitrarily large for vacant periods where no activity occurs. However, this feature will actually be a benefit if the relative timings of events and object actions are not important, but only their order. If, in fact, the relative timing is useful, then the object next-state routines, which control object action timing, can continually request service on each clock tick, thereby forcing a simulation cycle, potentially null, at every time instant.

An issue to consider with this approach involves the ultimate animation performance achieved. For the purposes of visualization, "good" performance relates to the extent to which the computational engine is capable of driving the graphics hardware. The bottleneck given current technology is the graphics bandwidth. It should be noted, however, that the graphics need only proceed as fast as the user can absorb the given visual stimuli. The simulations in PARADISE proceed at a sufficiently fast rate, and are controlled via multiple speed controls to provide the user with a range of visual simulation levels. Given the model size limitations of the current prototype, the graphics technology is indeed the limiting factor in animation speed. Clearly, as the size of visual models increases, the simulation overhead may become a factor, and this issue may require consideration in future prototypes. Other scalability issues are discussed in the last section on future work.

3.2. Model Abstraction

Describing system functionality may often be a complex task, involving many special cases and intricate interactions within the system. This is especially true for concurrent or parallel systems. To simplify this complicated specification of behavior, PARADISE divides the functionality of a system into independent objects. Each object encapsulates some portion of the behavior of the whole system, e.g. a switch box in an interconnection network. These smaller, less complex functional entities
act together to model overall system function.

The behavior of interest in a system can be highly integrated, and will not necessarily divide easily into completely independent. Various portions of the system are often inherently linked together by the data they exchange and the environment they share, therefore objects must interact and exchange information to fully model the original system. The flow of data and control inside the original system is reproduced in the model by allowing objects to transfer and exchange data, or communicate, via events. Note that communication here is referred to in a fundamental sense, implying the simple transfer of information between logical objects. There is no forced implication of physical messages or protocols. For example, this approach applies equally well to both distributed message-passing and shared memory parallel computer systems. The concept of communication is used only as an abstraction to describe the interaction in a system.

The information exchanged in the original system is made available in the model through the events transferred among objects. The creation of each event during simulation is marked with a time-stamp which identifies the moment when data became available in the original system, and the data itself is passed implicitly as the body of the event. The internal state of each object in the model is a function of the events which are processed by that object.

To accurately simulate the patterns of interaction in the original system, the structure and arrangement of the objects in the model must be carefully specified. Not all objects may directly interact in the system, and this fact may very well constitute a substantial portion of the functionality. To allow a precise specification of the interaction in the model, the objects are linked together by explicit connections of interaction links between pairs of interaction ports. Each link between two objects represents the capability of those particular objects to interact through the given ports. This interaction is illustrated by animating the motion of a small dot along the interaction link in the direction of the transfer of information.

It should be noted that, in the current prototype, the time taken for this transfer is contrived, having no correspondence to the actual time taken for the interaction in the original system. All animated event transfers in a given phase are assumed to take place simultaneously and atomically. The rate of transfer in the animation is fabricated based on the relative graphical length of the interaction links, such that all events in a phase begin and end their transfer animation at the same time. Proportional event transfer rates would require specification of begin/end event pairs to identify a time range for each transfer. Alternatively, events could be defined with a duration as well as a time-stamp so that the transfer time is explicitly known.

3.3. Abstraction Level and Object-Oriented Nature

Models in PARADISE serve not only to reproduce the behavior of a system, but also introduce a method for enhanced examination of desired details. By manipulating the level of abstraction in the model, specific characteristics may be emphasized or de-emphasized to lead to a better understanding of the original system. Models can be constructed to view a parallel system throughout its continuum of abstraction levels. Even multiple levels of abstraction may be used together in a single model at the user’s discretion. In the current implementation, the abstraction level is fixed for each object within a given model, but the methodology could support models with an explicit hierarchy of abstraction levels for each object. With these models, the level of abstraction displayed to the user could be interactively manipulated throughout a simulation. This more sophisticated modeling capability might be introduced by constructing model objects in a top-down, modular fashion, such that each individual object would itself be represented hierarchically at multiple abstraction levels, as a composition of lower-level sub-objects.

A natural paradigm for manipulating model objects is the object-oriented approach. PARADISE uses concepts from this approach to construct complex models one object at a time. Each object is designed to possess its own functionality or internal behavior, and no direct consideration is given to the functionality or influence of other objects. Only the external interaction characteristics and protocols require development in a global framework. Specification of these interaction protocols is organized by the interaction port and link mechanisms.

A benefit of using an object-oriented model is the ability to replicate and reuse objects. The user can form a “library” of objects by saving the definition files for previously used objects. New objects can be added to these collections of existing objects, allowing use of combinations of objects together in the same model. Special purpose libraries could be accumulated for particular classes of analyses, providing collections of relevant, predefined objects.

It should be noted that PARADISE obviates the strict boundaries of the object-oriented perspective by allowing use of global information in the specification of object function. The user can define global attributes and state variables for assisting in model specification. The ultimate goal of the model interface is fast and easy development, and this use of global variables can potentially simplify model construction, though this capability violates the true object-oriented criteria and restricts object reusability. Use of global information provides a means for distinguishing like object instances without the need for complex parameterization of object functionality.

3.4. Visual Nature

The true usefulness of the model abstraction approach described above results when it is merged with graphic visualization techniques. Visualization provides a clear, intuitive representation which organizes the model specification and allows a natural interaction by the user. The model abstraction encourages a user to apply creativity and intuition in modeling a system, and the visualization techniques allow a straightforward framework for directly manipulating this model. The constructs and concepts which constitute the model all convert directly to graphical representations. Encapsulated objects with well-defined external interfaces become visual objects which can be completely visually manipulated. Each object has a descriptive graphical icon image, defined with graphical locations corresponding to interaction ports. The interaction links are simple lines drawn to connect the ports. By graphically arranging images for each object and literally drawing the interaction links between port locations, the model structure can be intuitively constructed from the mental picture in the user’s mind. No textual descriptions are necessary for this stage of the design.

The resulting visual analysis proceeds in an intuitive manner, animating the behavior of the system using the user-selected images and structure. The state of the overall model is seen in the collective states of the objects, as reflected by their dynamic images. The interaction inside the model is illustrated by graphic animation of the transfer of events among the model objects. Although the current implementation shows this interaction as dots traversing the interaction links, there are other meaningful displays for such interaction (including no display at all). The picture changes dynamically in conjunction with the actions of the actual system, and the desired results of the analysis are readily evident and implicit in the graphic illustration. More importantly, the results are in the intuitive graphical form selected specifically to suit the perspective of the particular user.

3.5. Model Specification and Simulation Control

Though the capabilities of visual analysis using the model abstraction can provide meaningful animations, the feasibility of the approach depends on how easily models can be specified. In PARADISE, the specification of models consists of two distinct stages:

1. Textual Definitions and Next-State Routines, and
2. Graphical Model Configuration and Interconnection

which together produce a complete functional and structural description of a system. These stages utilize a hybrid specification language which combines both textual and graphical

53
The textual definitions currently consist of both static textual declarations and programmed routines. The static declarations are simple statements that describe events, global model and individual object attributes, and other static object information. Events are custom-designed by the user for each analysis, and are selected based on the interaction to be modeled. The semantics of the events are implicit in the user routines which describe object functionality. The only information required for event definitions is the amount of data each event requires. The static textual definitions for the objects describe information about object structure. This information includes the name of the object, the internal states, the type and location of interaction ports, and other user-defined attributes and state variables.

The programmed next-state routines are more complex in nature than the static declarations, and describe the internal functionality or behavior of each object. Based on the flow of input events and the current internal state, these routines determine, if necessary, what state to set each object to at the next instant in simulated time. The next-state routines are written as standard high-level function language in C, with the assistance of C preprocessor macros which perform useful, standard operations. These operations deal with event I/O, state changes, and attribute and state variable access. The next-state routines can potentially be complicated, however a next-state routine need only simulate behavior which is externally visible and of interest to the analyst. This fact can reduce some complexity by allowing removal of details below the "visible" level of abstraction.

Once specified, the textual user definitions are compiled and directly integrated into the tool environment by linking them with the PARADISE system library. This process completes a custom graphical interface for construction of visual models. The user specifies the overall model structure interactively using a graphical editor interface which arranges and connects the objects in the visual model. Each object is selected by its image from a palette of possible objects and graphically placed onto a model canvas. The desired interaction links are graphically drawn as lines connecting the ports between pairs of object images. This stage of the visual modeling is extremely efficient and proceeds in a matter of minutes, even for complicated models. The structure of a model can be modified by graphically adding, moving or removing objects and connections.

Upon completion of the visual arrangement and connection, the result is a custom visual analysis tool which is ready for use. Animated simulations can immediately be run, given an appropriate stream of events as input. PARADISE supports replay from program execution traces and probabilistic or stochastic inputs, as well as pure enclosed simulation without external involvement. The latter is possible because all objects are capable of producing and consuming events, hence a simulation does not always require external driving by traces. In all cases, the user collects or produces any event sequence desired for the particular analysis. It is beyond the scope of this paper to cover the PARADISE preprocessor macros which perform useful, standard operations. These operations deal with event I/O, state changes, and attribute and state variable access. The next-state routines can potentially be complicated, however a next-state routine need only simulate behavior which is externally visible and of interest to the analyst. This fact can reduce some complexity by allowing removal of details below the "visible" level of abstraction.

3.6.2. Animated Simulator Interface

The simulation control interface is identical to the model editor interface, with different functionality assigned to the control panel buttons and canvas interactions. The user can select a model configuration and an event stream using pop-up text windows, and various simulation features can be applied from the control panel buttons. The PARADISE interface is shown during an example animated simulation in Figure 6.

![Example Animated Simulation](image-url)

Figure 6: Example Animated Simulation

There are several parameters which can be interactively adjusted to control different aspects of the animation. The speed of the animation as viewed by the user can be controlled in two different ways. The overall rate at which the simulator steps through each time instant is controlled by the execution speed. The speed at which event propagation is animated is controlled by an Interaction speed. Adjustment of these two speeds allows precise control of the amount of visual stimulation to the user, and the level of detail which is perceived.

There are several special interactive analysis functions which allow more detailed examination of the current state of the model. At any time the user can halt the simulation to determine the values of object attributes and state variables, as well as event data for events in transit between objects. If desired, the model editor can be entered at this point to modify these values, however this procedure should be handled with caution to avoid inconsistent model states. Any global model information can also be displayed with the simulation stopped. When all desired information has been obtained, the simulation can continue from where it stopped. If desired, the pop-up object windows can be left open during the simulation to display the textual object state dynamically.

4. Experiences and Future Work

The past three years have been spent developing and experimenting with the PARADISE prototype, providing a valuable opportunity for close examination of various aspects of program visualization. Many lessons have been learned from this experi-
ment, and the feedback from users, along with our own experiences, has led to a more focused perspective and understanding of the important issues. Though the PARADISE framework for program visualization takes a step towards more efficient and flexible visual analysis tools, there are still many unresolved issues and drawbacks. Many of these relate to implementation issues, but others involve some fundamental, underlying concepts of user interfaces in general.

The following sections describe some of the current drawbacks to the PARADISE prototype, along with potential alternatives and solutions. In addition, some experience and ideas relating to general visualization issues will be discussed, with discussion of the more fundamental interface issues. The focus is on how these experiences and lessons learned should direct future research.

4.1. Limitations of PARADISE
The central problem with the current PARADISE prototype is the overhead in specifying the functionality of model objects. The interactive construction of visual models is highly efficient, but the development of the objects which make up the model is still somewhat involved and requires textual programming. The static specification of object ports, states, and attributes is completely textual, although assisted by high-level macros, but these static declarations could be graphically specified through the use of menus and visual displays. For example, the location of interaction ports for each object would best be specified graphically using the object icon images, with corresponding port types chosen from menus. In addition, the object images used with PARADISE could be more efficiently designed and organized using a special-purpose image editor, rather than standard icon specification tools. A custom image editor would assist in organizing the images for the various object states.

The programming of object next-state routines is the most serious bottleneck in the specification of objects. At this time, there is no approach to specify the necessary object state relations other than use of a high-level textual language. While it may be possible in the future to utilize the hierarchical nature of objects to build complex functionality from simpler sub-objects, this process could still be involved. The best approach could be to formalize the specification of animations and actually construct a visual language for specifying graphical manipulations. Unfortunately, such a language would be difficult to construct. A language which specifies complex behavior completely visually would effectively solve the visual programming problem, but even further would actually stand as a universal language which describes activity in terms of symbolic, visual images. Perhaps within the limited context of visual view specification, such a language will still be possible.

Beyond improving the user interface for PARADISE, there are many capabilities which would enhance the flexibility and generality of the visual analyses presented. Visually oriented charts, graphs, and histograms for the display of various system and application statistics and performance metrics would be beneficial. These types of views provide efficiencies such as qualitative comparisons of quantitative data. Often these types of visual analyses are useful in conjunction with the more sophisticated visualizations, either for initial program analysis or to display information in a more concrete fashion.

Following the meta-tool framework of PARADISE, statistical displays could be integrated in their generic forms and configured as necessary by the user to display desired statistics. These displays could even be combined with the visual models to visualize internal object statistics, including the dynamics of attributes and state variables. The visual specification techniques of TRIPLEX, HYPERVIEW, and PABLO could be combined with the diverse statistical displays of PARAGRAPh and the more generic features of PARADISE to produce a generic framework for the abstract display of statistical quantities.

Program visualization would benefit much from the use of more sophisticated, existing visualization techniques. The use of color and color intensity has proven highly effective in revealing contrasts and patterns. The addition of color in visual modeling would enhance flexibility and allow increased user creativity in the representation of system behavior. PARADISE would also gain flexibility if more sophisticated graphical animation techniques were utilized towards more efficient and flexible visualization. For example, moving objects would allow animation of behavior without static structural constraints. This feature would be especially beneficial for analyses of more abstract or structureless systems and characteristics.

The introduction of existing three-dimensional graphics techniques would expand model structure possibilities by venturing beyond the two-dimensional plane in model layout and organization. There are also other visualization dimensions which are more hypothetical and experimental, but which might prove useful. Simple uses of audio and audio intensity (volume) can be used to present information audibly in the form of tones at different pitches or qualities [12,34].

4.2. Other Visualization Issues
In many existing visualization approaches there are several unresolved issues which require attention. In most visual analysis systems there is a scalability limitation introduced by the inherent bounds of the graphics technology. Due to the potentially large amount of detail for massively parallel systems, it becomes impossible to fit an entire display in full detail on even the largest, high-resolution graphics monitors. Solutions to this problem could lie in more powerful views which allow adjustable perspectives. By providing a zooming capability, the user could zoom in on a display for examination of precise detail. Alternatively, the general status of the entire model could be examined by zooming out and visually smoothing over individual object detail. Along with graphical zooming, other capabilities such as abstraction zooming would allow useful traversals of the levels of abstraction in a display. At the highest level, only highly abstract information would be visible. When zoomed in to lower levels, the more detailed, low-level information would be displayed. This technique, combined with the graphical zooming, could reduce the effects of scaling.

Another issue in scalability lies in the ability of the user to construct displays with large numbers of graphical images. Especially in massively parallel systems, many objects and connections are replicated in highly regular patterns, suggesting the need for visual macros which could automatically produce large models given small scale templates for structure and interconnections. This is conceptually similar to the construction of hierarchical objects from collections of sub-objects.

Other than scalability issues, various other capabilities are needed relating to the focusing of large, complicated views. The ability to adjust animation speed is important for this purpose. Likewise, the direction of the animation, forward or backward in time, is necessary for flexible control of analyses. In addition, visual analyses would benefit from summarization techniques which would compress information by organizing system components into behavior. This summarization would allow revealing comparisons and contrasts.

4.3. General Visual Interface Experiences
Aside from the above specific issues, there are other important general issues to be considered in constructing any interface tool. The experience gained in implementing the PARADISE prototype has led to an understanding of many of the parameters involved in designing visual tools to assist the user. There are two distinct perspectives which both need to be considered in constructing a tool, that of the user and the tool designer.

The user's perspective on any given tool relates to what features and functions are desired, independent of their feasibility or the implementation details. The typical user has certain requirements as to the necessary flexibility, power, and efficiency of a tool, as well as the extent to which a tool can be customized to suit the user's personal preferences. A user likely wants the best of both worlds, such that all functions will be performed automatically, but the option is always available to completely customize them.

On the other side of tool development, it is the responsibility of the tool designer, especially a meta-tool designer, to find the means for providing the user with the desired flexibility and
functionality. The designer must organize the functionality of a tool to allow straightforward access by the user, in as intuitive a manner as possible, while still providing generally useful functions. The emphasis of the designer is on evaluating the possible alternatives and choices a user will require, and the ultimate goal is to identify those common classes of options which will encompass the full spectrum of analysis functions. The tool designer must parameterize the very analysis itself into its fundamental components.

The difficulty in designing any tool lies in combining these two perspectives successfully to arrive at a feasible tool which will meet the user's requirements easily and actually be used. Only a handful of visualization tools to date have gained wide acceptance or are used as a standard analysis alternative. More consideration is needed of the concerns and perspectives of the user in the development of visualization tools.

References:


