On the Use of Visualization in Formal Requirements Specification

Nicolas Dulac, Thomas Viguier, Nancy Leveson, Margaret-Anne Storey
Aeronautics and Astronautics Department
Massachusetts Institute of Technology
email: {ndulac, tviguier, leveson, mstorey}@mit.edu

Abstract

A limiting factor in the industrial acceptance of formal specifications is their readability, particularly for large, complex engineering systems. We hypothesize that multiple visualizations generated from a common model will improve the requirements creation, reviewing and understanding process. Visual representations, when effective, provide cognitive support by highlighting the most relevant interactions and aspects of a specification for a particular use. In this paper, we propose a taxonomy and some preliminary principles for designing visual representations of formal specifications. The taxonomy and principles are illustrated by sample visualizations we created while trying to understand a formal specification of the MD-11 Flight Management System.

Keywords
Requirements specification, visualization, formal methods

1. Introduction

Formal specification languages have not been widely used in industry. One limiting factor is their readability, particularly for large, complex software systems. Requirements act as a communication medium between customers, users, and implementers. Promoting a common understanding of the required functionality of the system is key to domain experts finding errors and validating that the specifications describe a system that will be useful and safe in operation.

Reviewability is especially important in real-time software used to control some physical system where the requirements must reflect externally derived properties of the system being controlled. Validating software behavioral requirements in such systems is a necessarily multidisciplinary problem involving a large number of engineering disciplines. While automated analysis tools can find some types of errors, detecting many of the most serious semantic errors (e.g., will the software advance the throttle under unsafe conditions or will the software behavior lead to human errors in controlling the aircraft) requires human expertise.

Size is a critical factor in reviewability. The discrete mode logic for an aircraft flight management system may require hundreds (sometimes thousands) of pages of formal logic to specify in adequate detail. The review of such specifications by domain experts or even by those who are expert in the formal notation itself is a daunting task.

Our experience in trying to build and read very large specifications for systems such as flight management, collision avoidance, and air traffic control has shown that even with a formal notation designed with readability in mind, the complexity of the behavior being described overwhelms the reader. Not only is it difficult to provide notations that can be reviewed by people with different backgrounds and expertise, but for complex systems, most users (even the authors of the specification) need help in navigating and understanding them.

Visualization is often seen as a way to help people gain insight from large and complex data sets. Indeed, people have used external aids for centuries to amplify cognition (e.g. paper, slide rule, diagrams, charts) [CM00]. We believe that the use of computer supported, interactive, visual representations of requirements specifications will help engineers create, review and understand formal specifications. Although graphical notations are often used for presenting formal specifications, we describe how more sophisticated interactive techniques can help users navigate and comprehend the specifications.

Unfortunately, there are few principles to follow when designing interactive or even non-interactive graphical and symbolic notations for formal software requirements specifications or for visualizations of these specifications. This paper provides some foundational ideas upon which further research could be based. The principles were derived by considering principles developed for other types of software visualization (such as visual programming and program comprehension tools) as well as from our own experiences in creating specification languages, in building large complex requirements specifications, and in conducting experiments on specification language design [ZLL02] and on visualization [SWF96]. The resulting set of design principles can be used to guide the design of new languages and visualization tools and to assist in critically evaluating them.

We illustrate the taxonomy and principles using an example specification of the annunciation process of the vertical guidance module of the MD-11 Flight Management System. Vertical guidance is usually the most complex function in a FMS, and our complete formal SpecTRM-RL [Lev00b] specification of the FMS logic is over 500 pages long. The principles and examples presented in this paper derive from the attempts to understand this system by graduate students who had not written it (and even by those who had).

2. Background and related research

Research into the impact of visualizations (diagrams) on cognition was pioneered by Larkin and Simon [LS87]. Their work has become the foundation of most of today's research efforts on this topic. Although there is a large literature on visualization, the majority of it involves visualization of data, usually scientific data, and not visualization of systems or processes.

The way that information is displayed can facilitate or distract from learning and understanding. Effective visualizations will help convey meaning and explain concepts or designs. Visualizations of complex requirements specifications can potentially reduce cognitive load by highlighting the relevant interactions and behavior of the specified system.

There are many ways that representations can affect and alter task performance. They can draw attention to certain aspects of the information that support problem solving. Good representations can also shift the cognitive load – balancing the use of mental resources, shifting attention, and creating perceptual cues. Likewise, poor representations create additional tasks or make the tasks more difficult to perform. Casner and Larkin [CL89] have suggested that good representations reduce the amount of cognitive processing in two ways: (1) they allow users to substitute less demanding *visual* operators for more complex *logical* operators, and (2) they reduce the search time for the information required to perform a task.

There has been a lot of research in visual programming languages and on visualizations to support program comprehension. Fitter and Green [FG79] elucidated five principles for effective visualization design that are potentially applicable to formal specifications. For example, the most usable notations contain both symbolic and perceptual elements. In some cases they are independent and in others they are logically redundant. An example is the use of indenting and other special cues to make programs more legible or the use of layout conventions by mathematicians to make algebra and predicate calculus more readable. These perceptual cues have been called *secondary notation*, i.e., they convey additional meaning above and beyond the "official semantics" of the specification language or they disambiguate syntactic structure in order to assist in interpreting semantics.

Another useful principle from programming language design research is that of using redundant recoding [FG79]. An example is specifying the same information in two different ways, each of which simplifies different cognitive tasks. Redundant recoding may also be used to emphasize certain information. For example, when a piece of information is especially important to a user's task, or if it is highly critical to the overall structure of the information, it is helpful to present a high level view in a perceptual form, while simultaneously presenting the intricate details in symbolic notation.

Blackwell *et al*. [BWG99] and Petre [Pet95] have proposed some cognitive dimensions for visual language design. Example dimensions are closeness of mapping, consistency, and visibility. Storey *et al*. developed a cognitive framework of design elements to be considered during the design of a software visualization tool [SFM99]. This framework contains two sets of factors to support the variety of comprehension strategies used by programmers during software exploration and to reduce cognitive overhead as they explore and try to understand the software.

Some of the research on human-computer interaction is also applicable to our problem. Designers of interfaces, like those of requirements specifications, need to select the appropriate level of abstraction, determine how to show relationships, provide context for the individual bits of information, and build conceptual spaces using frames of reference [RW95].

In this paper we create a taxonomy for visualizations of formal requirements specifications and propose a preliminary set of principles for creating effective visualizations of formal requirements specifications. The principles are adapted from what has been learned in visual programming language and human-machine interface design, but we have found that effective visualizations to assist in designing a solution to a problem (e.g., programming or manipulating a computer interface to command a desired behavior) are very different than those useful for specifying the problem. In other words, describing "how" differs significantly from describing "what". As a result, we cannot simply apply the same principles and research results. Instead, our first goal is to create an initial framework and some potential hypotheses upon which research in requirements visualization can be based. The hypotheses will then need to be validated using experimentation with human subjects.

The next section presents a set of dimensions upon which requirements visualizations can be evaluated and sample visualizations created using the MD-11 Flight Management System (FMS) specification. Drawing both on principles for visualizations in other fields and on what we learned in our MD-11 case study, Section 4 presents some principles for evaluating visualizations for formal requirements specifications. Section 5 concludes the paper and outlines future work.

3. A taxonomy of visualizations for formal requirements specifications

We start by presenting a taxonomy or set of six dimensions upon which visualizations can be classified:

1. *Scope*
   The visualization may focus on the structure of the model or the goal may be to visualize the behavior of the specified system.
2. *Content*
   The visualization may include the entire model, perhaps using a different notation (e.g., symbolic, tabular, or graphical), or information may be elided. Elision is the ability to temporarily hide parts of the specification that are not of immediate interest. When information is elided, it may still be useful to retain some of the omitted information as context, but it is grayed out or somehow denoted as background rather than foreground. Alternatively, the visualization may not provide context beyond the information provided in the visualization itself.
3. *Selection Strategy*
   The visualization may be created through *slicing* the basic formal model, i.e., a selection based on dependences between the parts of the model or by *filtering*, i.e., eliding parts of the model based on a common property or attribute.
4. *Annotation Support*
   The visualization may include only information provided in the original specification or the user may be able to add extra domain knowledge through annotation.
5. *Support for Alternative Search Strategies (Flexibility)*
   A visualization may be provided that supports a particular search or problem-solving strategy without any options for the user. Alternatively, the user may be able to specify the search strategy to be supported by the visualization. A third option is to provide interactive visualizations where the user can change the search strategy while navigating through the model.
6. *Static/Dynamic*
   A static visualization is a snapshot of the specified behavior of the system at a particular time or a static description of all possible behavior. Dynamic visualizations or animations show the specified behavior of the system as it changes over time.

Any specific visualization can be categorized with respect to each of these dimensions. For example, an animation or dynamic visualization may highlight particular aspects of the behavior, which is a form of elision that retains context (i.e., separates the visualization into foreground/background in order to draw the reviewers attention to particular parts of the behavior) or it may completely omit parts of the specified "machine".

Figures 1 through 6 show some example visualizations created for a formal specification of the MD-11 FMS. Two of us (Dulac and Viguier) inherited this SpecTRM-RL specification written by former students of Leveson. Dulac and Viguier created several visualizations to help in understanding this large and complex system.

A state machine model is used in these examples because we have found this type of formal model easiest for engineers to use for these types of control systems. Different underlying models may be more appropriate for other applications, e.g., models based on set theory may well be the best for representing information systems. The same types of visualization dimensions (and the principles presented in the next section) should apply to these other types of formal specification languages, but the form of appropriate visualizations may differ.

Figure 1 shows a visualization created to help understand the many dependencies between the parts of the specification. In terms of scope, this visualization represents only a small part of the functionality of the FMS (the mode annunciation of the vertical guidance module) but it depicts all the dependencies for that subfunction. In this representation, the arrows represent dependency relationships; an arrow pointing from A to B, for example, means that the value of element B depends directly on the value of element A. As can be seen in Figure 1, there is a high degree of coupling between the elements of the system. Although the details are hard to see (the user would have to zoom in to read them), this picture provides a "gestalt" overview of the size and complexity of the specification structure. The visualization also supports different ways to navigate through the specification by following dependencies and viewing detail if desired.

With respect to the classification presented above, the visualization in Figure 1 shows a static snapshot of the structure (rather than the behavior) of a subfunction. The entire dependency structure is shown, but it represents only a small part of the specification. It has not been annotated with additional information, and the user can search by following dependencies and zoom in to view further details.

Figure 2 shows an example of a further slicing on the dependency relationships that were shown in Figure 1. In this case, the slice is constructed by displaying only the input-to-output paths going through a selected element. The element chosen for Figure 2 is the state variable *FMA Speed Magenta-White discrete*. All the elements that have no effect on the value or that are not affected by the value of the selected element are hidden. The fact that details have been elided is indicated by the light gray input-output-mappings shown on the right hand side of the visualization. This perceptual cue serves to remind the user that details have been hidden from the view and also provides some context for the visible parts.

Figure 3 provides a different type of overview: all the state variables and the modes in the specification. This overview spans the entire specification, but provides no information about dependencies. In this visualization, all details about transition conditions are elided based on filtering rather than slicing. The filtering here is done by *type* but other kinds of filtering are possible.

Figure 4 is an example of a visualization in which the information in the original specification language is redundantly recoded to assist in answering different types of questions. Like many of the modern specification languages utilizing state machines, SpecTRM-RL uses a metalanguage to describe the states and transitions---writing down the entire state machine would be infeasible for complex systems (our model of an aircraft collision avoidance system has 10^{40} states). Users of such specifications, however, find it very helpful if they can see at least part of the flattened state machine (diagram of states and transitions using the traditional circles and arrows). Figure 4a shows an example for the state variable *Vertical Cruise Sequence*. Clicking on one of the arrows displays the corresponding transition condition (which for the most part are too large to write on the arrows themselves).

The information in this visualization is the same as that in the original notation, but recoded to make it easier for the user to process. Note, however, the difference in scope from Figure 1, which focuses on the *structure* of the model. Figure 4 instead provides information about the *behavior* described by the specification.

Figure 4b shows a recoding of the same information in a form more suitable to answer a different type of question. This visualization is essentially the inverse of 4a, i.e., the impossible transitions. For many real systems, every state is connected to almost every other state, and providing information about the transitions that do not exist is more relevant. In addition, this visualization is useful in checking whether undesired transitions have been correctly omitted from the specified behavior.

The final two visualization examples were created to help us understand the logic behind the most complex transitions in the part of the MD-11 specification being considered. We found that it was easier to answer particular questions if the set of conditions in the transitions (which can be very large) are shown in sequence rather than in parallel. A decision tree for each state variable seemed most natural for us to accomplish this goal. Each node of the decision tree is a question and each branch is a decision (see Figure 5).

From left to right in Figure 5, each column represents one of the decisions that must be made to determine whether the transition will be taken, based on the state or value of the component of the model shown at the top of the column. The final states to which transitions can be made appear at the right end of each path.

Although this basic recoding does not by itself bring much insight compared to the original AND/OR tables from which it was generated, we found that adding annotations or informal questions at the top of each column was very useful. The annotations helped the users of the visualization to understand the reasoning involved in the decisions underlying the contents of the transition conditions. This tree is equivalent to five pages of text and six AND/OR tables. Showing the information together in one concise notation helped us to see similarities and differences (that is, to make comparisons) and to detect omissions.

Another important issue is the ordering of the questions (nodes in the tree). Although algorithms exist to minimize the size of the tree, the resulting ordering may interfere with the cognitive processing of the information in the tree. For example, it may be important to answer one question before another because it is the most important for a certain task or for answering specific questions about the specification of the state transitions. Thus, the user should be able to specify the ordering or to change it dynamically. The latter is an example of dimension 5 above labeled Supporting Alternative Problem Solving Strategies.

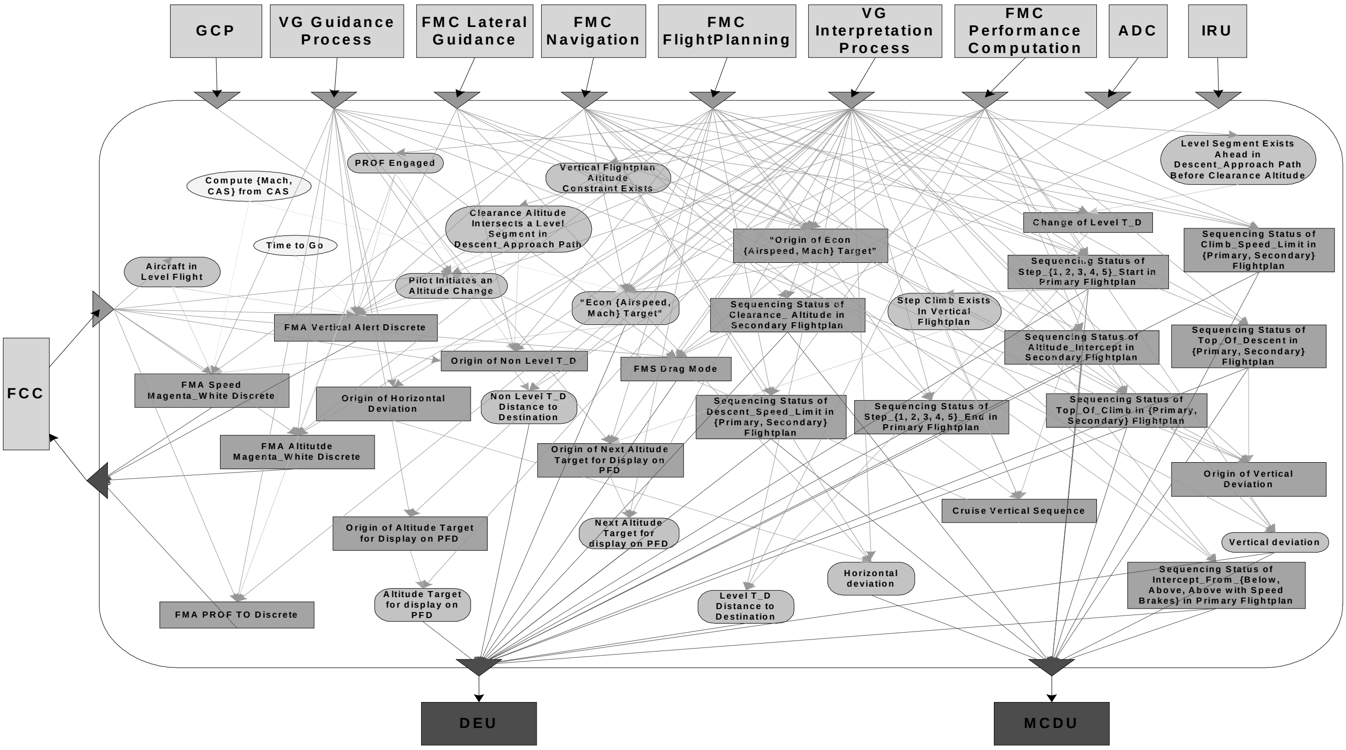

Figure 1. Visualization environment of the MD-11 vertical guidance annunciation process based on dependency relationships between specification elements.

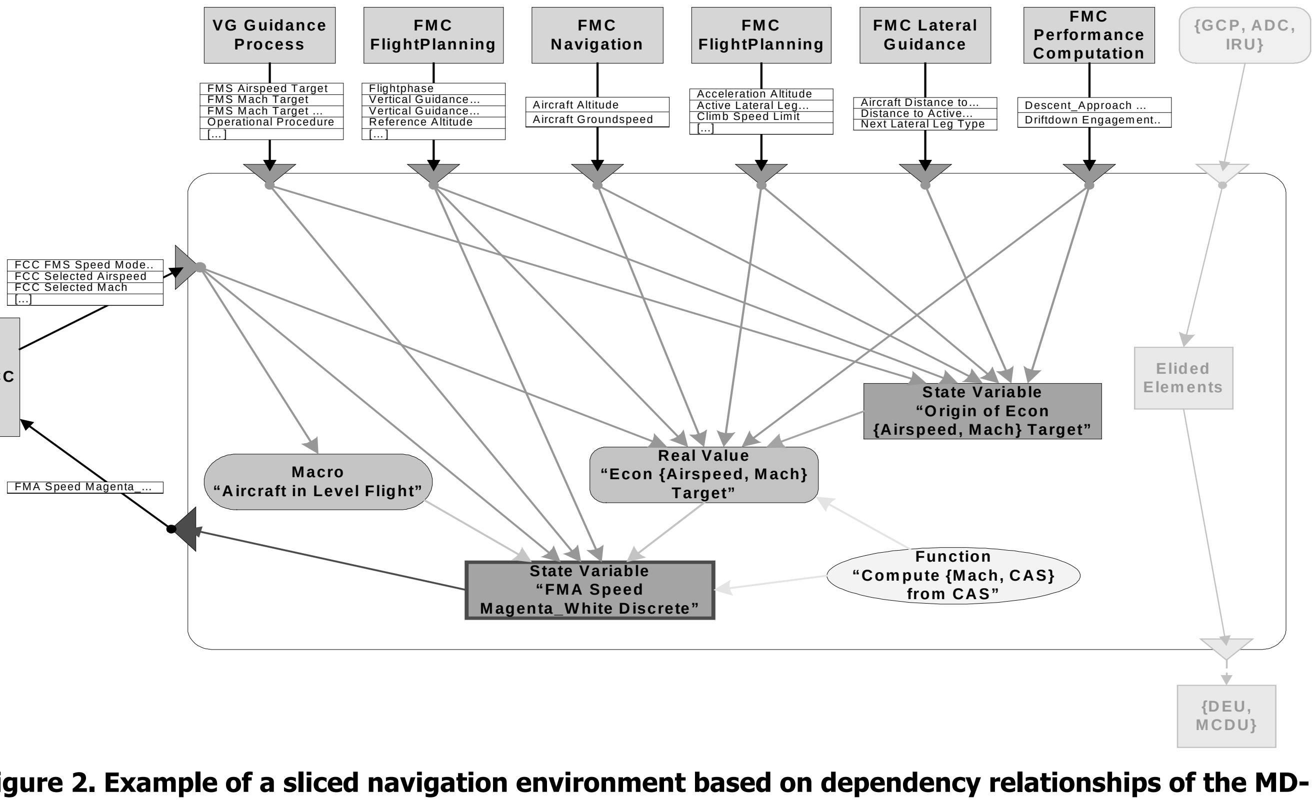

Figure 2. Example of a sliced navigation environment based on dependency relationships of the MD-11 vertical guidance annunciation process specification. The element chosen is the state variable *FMA Speed Magenta-White Discrete.*

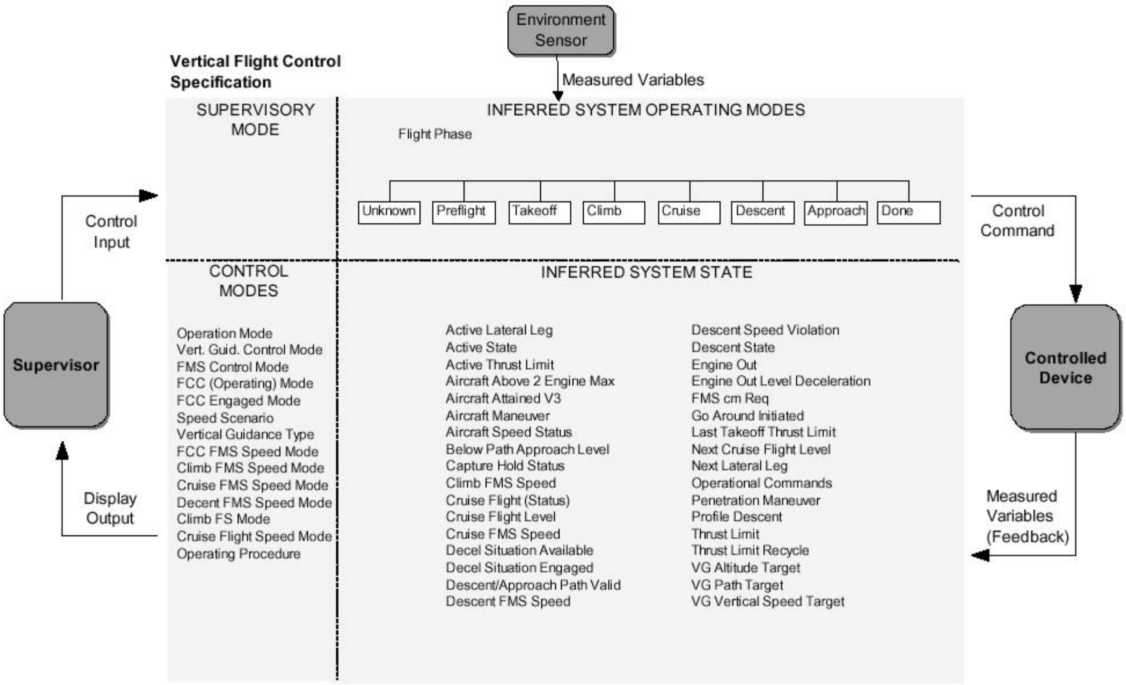

Figure □. □verview visualization of the MD-11 vertical guidance system based on model behavior. The state variables and modes are displayed.

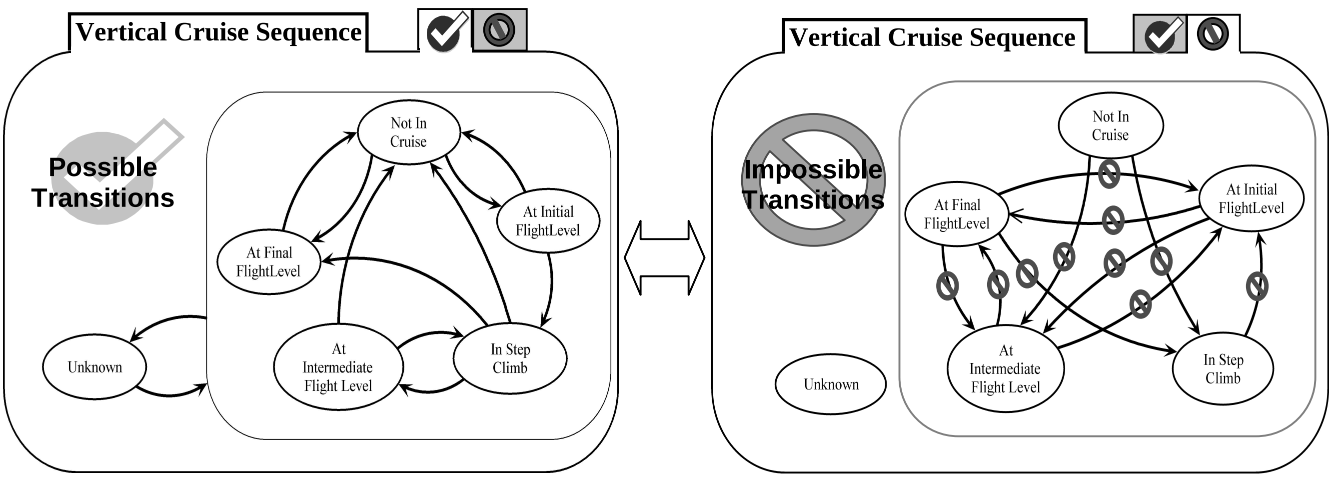

Figure □. An example of state machine a□transition diagram and b□inverse transition diagram ta□en from the MD-11 vertical guidance specification. This state variable describes the vertical attitude of the aircraft during cruise.

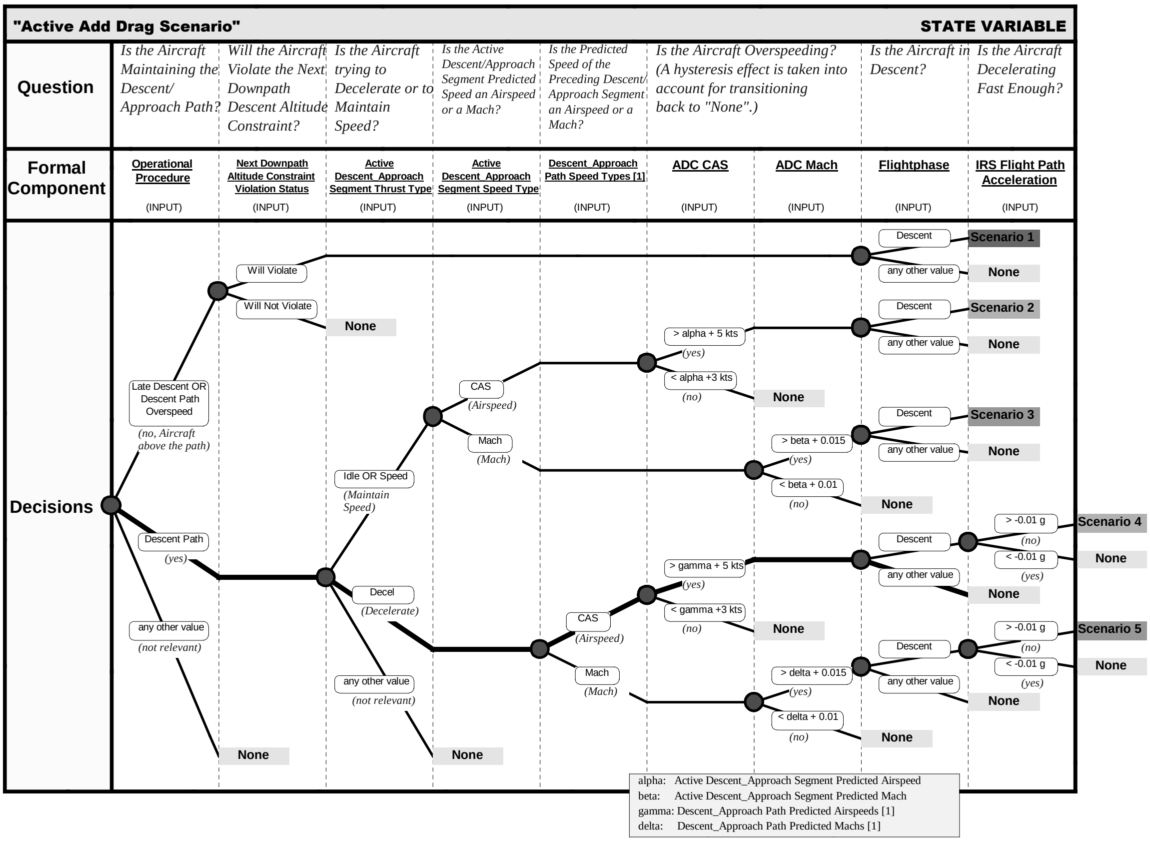

Figure □ An example of Questions-based Decision Tree taken from the MD-11 Vertical Guidance Specification. This state variable indicates which one of the five possible scenarios for extending airbrakes is active.

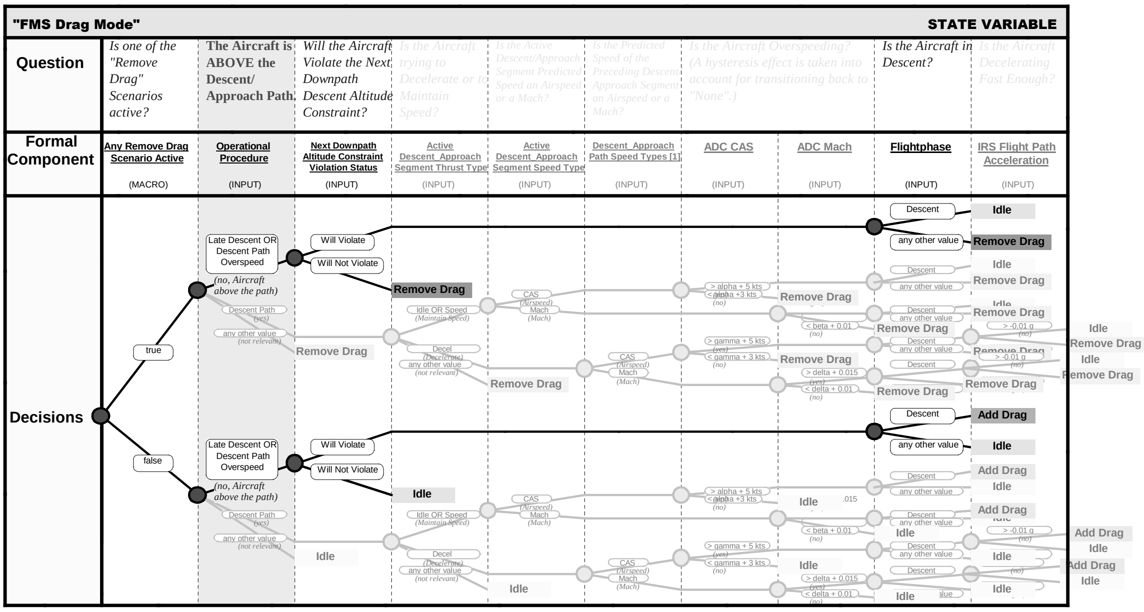

Figure □ An example of sliced Decision Tree taken from the MD-11 Vertical Guidance Specification. This state variable indicates whether the pilot should add or remove drag by extending or retracting airbrakes. The reduction scenario considers only the case when the aircraft is above its nominal descent path (darkened column).

This visualization becomes particularly interesting when combined with Heimdahl and Whalen's behavioral specification slicing [HW97]. In many cases, applying a selection strategy dramatically reduces the size of the tree. Figure 6 shows an example of this recoding into a decision tree using annotation and slicing.

The visualization might also maintain context information by providing perceptual cues where some of the information is foreground while the rest is context. Figure 6 shows a slice on the question of what happens when the aircraft is above its normal trajectory. The column involved is darkened and the many branches of the tree that are now inaccessible are grayed out and compressed. Questions that become irrelevant are also moved to the background shading. Thus the context is preserved while the focus is put on the paths that are still possible under the given scenario.

4. Principles for evaluating requirements visualizations

Not all visualizations are useful – sometimes they may even be misleading. We need criteria for evaluating potential visualizations and for creating effective ones. We adapted principles from visual programming and human-computer interaction research, and added some others based on our experiences and previous experiments. Of course, these principles will have to be evaluated formally, but they can serve as a starting place.

1) *Minimize semantic distance*

Semantic distance is a concept devised for human-computer interface design to describe the distance between the user's model of how the system works and the model of the system presented by the user interface [HHN86]. In the context of visualizing formal specifications, semantic distance is the distance between the model in the system specification and the mental model of the system in the mind of the users of the specification. We hypothesize that readability, reviewability, and writeability will be enhanced if visualizations are provided that minimize semantic distance. Leveson has found informally that reducing the semantic distance between standard engineering models of complex systems and formal specification notations can increase acceptability and usability of formal specification languages among people in industry who previously rejected out of hand the use of such specification languages. As an example, the formal model of TCAS II (a collision avoidance system for commercial aircraft), written with Leveson's modeling language, has become the official specification of this system [Lev00a].

In our experiment on readability of various notational features, we compared the specification of the conditions on state transitions using text, tables, graphical logic gates, and propositional logic. Every computer science student in the experiment commented on the difficulty of using the logic gate notation while the engineering students were mixed [ZLL02]. But similarity to standard notations is not the only relevant criterion, as every participant in the experiment preferred the tables and made the fewest errors in using them. The propositional logic notation ranked at the bottom for both of these criteria. These results confirm our industrial experiences.

2) *Match the task being performed*

Gilmore and Green's basic *match/mismatch hypothesis* states that problem-solving performance depends on whether the structure of a problem is matched by the structure of a notation [Gre77]. Applying this hypothesis to visualization implies that the most effective visualizations of requirements specifications will be those that most closely match the problem being solved or task of the specification user. The goal is to match the task to be performed with a visualization that minimizes the amount of cognitive processing required to perform the task.

3) *Support the most difficult mental tasks [FG79]*

Some tasks that use formal specifications will be more difficult than others in terms of the number and difficulty of the cognitive processing necessary to perform those tasks. The most useful visualizations in this context will obviously support the hardest tasks and not simply those that are easiest to create or are appealing to the visualization tool builder. This principle implies that the first step in creating useful visualizations is to determine who the users will be, to perform a task analysis of their potential uses of the visualization, and to analyze the difficulty of performing the task without a special visualization of the formal model.

4) *Highlight hidden dependencies and provide context when needed [GB98]*

Any notation makes some dependencies clear while obscuring others. These hidden dependencies may or may not be important in performing a particular task. If the dependencies are relevant to a user, then visualizations should be provided that perceptually highlight those dependencies. For example, some formal specification languages based on state machines organize the specification in such a way that it is easy to determine the previous states but not the potential states that follow the current state and vice versa. A good representation will in general point out dependencies or show causal relationships between different automation behaviors. In addition, if only one small part of a specification is being displayed, then context for the rest of the specification may have to be provided (if the context is required for the task at hand).

5) Support top-down review

Graphical overviews of the entire specification can be very powerful. During the periodic reviews by domain experts of a formal specification of a collision avoidance system Leveson provided for the FAA, the reviewers would spend hours reviewing and discussing the graphical overview of the state variables and state values used in the specification without referring to any information about the conditions under which the state values were selected (the conditions on the transitions between the states), which were not visible in the graphical overview. They preferred starting the review using that overview before delving down into the details of the transitions even though all the information in the overview could be deduced from the structure and content of the rest of the specification. This is an example of the *gestalt* effect in cognitive psychology in which providing an overview makes overall structure or relationships visible or clearer [Pet95].

6) *Support alternative problem-solving strategies*

A principle of cognitive psychology is that the reasoning paradigm is distinct from the representation paradigm. "The cost of reasoning about a particular representation may vary, depending on how the programmer's reasoning shifts" [BWG99]. Therefore, the representation may need to change as the user's reasoning process shifts. In addition, different people will employ different problem-solving strategies for the same problem. Experts are more likely than novices to change strategies while problem solving and to exhibit flexibility in their strategies [Pet95, SWF96]. Supporting expert use of formal requirements specifications with visualization will require supporting flexible search strategies and the ability to navigate between abstract and detailed views, as well as within detailed views. This principle emphasizes the fact that there will not be a fixed set of visualizations that are best for all people solving the same problem or performing the same tasks.

7) *Show roles being played*

Visualizations should provide insight into the role being played by a specific part of the specification. As an example, consider the use of modes in control system requirements specifications such as those used in SpecTRM-RL [Lev00b].

Modes are a common way of abstracting and grouping important subsets of behaviors of the overall system behavior in control systems[1]. That is, modes divide the overall system behavior into a set of disjoint behaviors, e.g., the behavior of the flight management system during landing mode or cruise mode. Modes are useful in simplifying (reducing) the amount of specified behavior that must be considered at any time. If there are multiple independent mode classes, the system behavior may be described in terms of the cross product of the individual mode values. Basically modes allow us to divide the behavior of the system into non-overlapping chunks that are easier to process cognitively. Multiple modes allow chunking on different dimensions. Visualizations for control system requirements specifications should allow identifying and highlighting the role of each mode in the overall system behavior being described and the role played by each of the components of the specification in a particular mode. Similar advantages accrue to expressing other important roles in requirements specifications.

8) *Provide redundant encoding*

Any representation makes some questions easier to answer while making others harder [FG79]. For example, a list or table showing classes taught, time, and professor that is ordered by class number will make it easy to answer questions about who is teaching a particular class, but much more difficult to answer a question about which classes a particular professor is teaching. A different ordering will make the latter question easier to answer than the former. By providing redundant but different encoding of the same information about the required behavior of the software, support can be provided for a variety of user tasks.

9) *Show side effects of changes*

Visualizations should allow investigating the impact of a change in one part of a specification on other parts (showing the indirect effects of changes).

5. Discussion and future work

The principles above have been adapted from other fields or introduced on the basis of our specification experience. They should not be considered as a rigid and exhaustive set of rules but as a starting point. They will need to be refined and evaluated against a variety of visualizations.

[1] Some designers of state-based software specification languages have used the term "mode" as a synonym for state; therefore all states are modes. We instead use the term *mode* in the engineering sense and as originally defined by Ashby in systems theory [Ash56]

Although the visualizations described in the previous section were useful in understanding the MD-11 specification, this anecdotal evidence does not prove their usefulness to a broad class of users and specifications. We are designing experiments with human subjects to validate the application of the principles to formal specifications.

Four of the visualizations presented in this paper have already been partially evaluated through a limited pilot user study. Although this evaluation was not specifically focused on the principles of the previous section, the purpose was to assess the hypotheses upon which the design of the visualizations was based. This pilot study will serve as a starting point for a more formal evaluation of the principles.

A longer term research goal is to investigate how the use of visualizations can assist in the synthesis of formal specifications, not just understanding those that have already been created. A third goal involves the use of specifications in training those about a system design (such as training air traffic controllers or pilots to interact with and use advanced automation effectively). In training, the goal is to provide visualizations that help users create accurate and useful mental models of the designed system behavior. Research will be required to effectively achieve these goals.

6. Acknowledgements

This work has been supported by NSF ITR grant CCR-0085829 and by a grant from the NASA Intelligent Systems program (Human-Centered Computing) NCC2-1223.

References

[Ash56] W.R. Ashby, "An Introduction to Cybernetics", John Wiley, 1956.

[BWG99] A.F. Blackwell, K.N. Whitley, J. Good, and M. Petre, "Cognitive Factors in Programming with Diagrams", accepted for publication in *Artificial Intelligence Review*.

[BG92] K. Brade, M. Guzdial, M. Steckel, and E. Soloway, "Whorf: A Visualization Tool for Software Maintenance", presented at 1992 IEEE Workshop on Visual Languages, Seattle, WA, 1992.

[Cas91] S. Casner, "A Task-Analytic Approach to the Automated Design of Graphic Presentations", *ACM Trans. on Graphics*, 10:111-151, April 1991.

[CL89] S. Casner and J.H. Larkin, "Cognitive Efficiency Considerations for Good Graphic Design", 11th Annual Conf. of the Cognitive Science Society, August 1989.

[CM00] Readings in Information visualization, Using Vision to Think, Edited by Card, MacKinlay and Shneiderman, Morgan Kaufmann, 2000.

[FG79] M. Fitter and T.R.G. Green, "When do Diagrams Make Good Programming Languages", *Int. Journal of Man-Machine Studies*, 11(2):235-261, March 1979.

[Gre77] T.R.G Green, "Conditional Programs Statements and their Comprehensibility to Professional Programmers", Journal of Occupational Psychology, 50, 93-109, 1977

[HHN86] Edwin Hutchins, James Hollan, and Donald Norman, Direct Manipulation Interfaces, in Donald Norman and Stephen Draper, *User Centered System Design*, 1986, pp. 87-124

[HW97] M.P.E Heimdahl and M.W. Whalen, "Reduction and Slicing of Hierarchical State Machines", 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering, September 1997.

[LS87] J.H. Larkin and H.A. Simon, "Why a Diagram is (Sometimes) Worth Ten Thousand Words", *Cognitive Science*, 11(1):65-99, Jan-Mar 1987.

[Lev00a] N.G. Leveson, "Intent Specifications: An Approach to Building Human-Centered Specifications", IEEE Trans. on Software Engineering, 2000.

[Lev00b] N.G. Leveson, "Completeness in Formal Specification Language Design for Process Control Systems", Formal Methods in Software Practice, Portland, 2000.

[Pet95] M Petre, "Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming", Comm. of the ACM, 38(6):33-44, June 1995.

[RW95] D.S. Ranson and D.D. Woods, "Making Automation Activity Visible", Technical Report 1995-03, Ohio-State University, December 1995.

[SFM99] M.-A. Storey, F.D. Fracchia, and H.A. Muller, "Cognitive Design Elements to Support the Construction of a Mental Model during Software Exploration", *Journal of Software Systems*, 44:171-185, 1999.

[SWF96] M.-A. Storey, K. Wong, P. Fong, D. Hooper, K. Hopkins, and H. A. Muller, "On Designing an Experiment to Evaluate a Reverse Engineering Tool", Working Conference on Reverse Engineering, Monterey, CA, 1996.

[GB98] Green T.R.G. and Blackwell, A.F., (1998) A tutorial on cognitive dimensions, http://www.cl.cam.ac.uk/users/afb21/publications/CDtutSep98.pdf

[ZJ96] P. Zave and M. Jackson, "Where do Operations Come From? A Multiparadigm Specification Technique", *IEEE Trans. on Software Engineering*,,SE-22(7), July 1996.

[ZLL02] M.K. Zimmerman, K. Lundqvist, N.G. Leveson, "Investigating the Readability of Formal Requirements Specification Languages", International Conference on Software Engineering, May 2002.