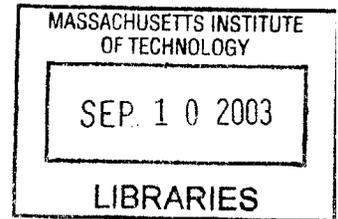


Evaluating Visualization In Formal Requirements Specification: An Experiment With Human Subjects

by

Thomas Viguiet

Ingénieur diplômé de l'Ecole polytechnique
Palaiseau, France, 2000



Submitted in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2003

© 2003 Massachusetts Institute of Technology. All rights reserved.

Second Copy

Signature of Author _____
Department of Aeronautics and Astronautics
June 1, 2003

Certified by _____
Professor Nancy G. Leveson
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by _____
Professor Edward M. Greitzer
H.N. Slater Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

AERO

Evaluating Visualization In Formal Requirements Specification: An Experiment With Human Subjects

by

Thomas Viguier

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Software used onboard aerospace systems is becoming more and more complex as its functionality expands; moreover, much of it is safety-critical. In that context, the requirements specification phase of the development becomes critically important for the success of a project and the safety of the system in operation. Formal methods have the potential to solve many of the problems encountered when specifying large and complex control software systems. However, they need to be reviewable no less than informal methods. Graphical visualization of formal specification represents an attractive way to improve reviewability at little cost. Unfortunately, no visualization design principles exist in the particular field of requirements specification.

This thesis presents a few sample visualizations of the formal specification of the vertical guidance system of a commercial aircraft. It also suggests principles for evaluating such visualizations. Although these principles were drawn from related fields, they still need to be evaluated in the context of formal specification. An experiment using human subjects to evaluate the principles is described.

Thesis Supervisor: Dr. Nancy G. Leveson
Title: Professor of Aeronautics and Astronautics

Acknowledgements

First and foremost, my thanks go to Professor Leveson, my academic advisor and research supervisor. Her support and wisdom over these two years at MIT have been a key element in this work. I am also very grateful to Peggy Storey and Kim Vicente for their valuable advice and their availability.

My deep gratitude goes to Nicolas Dulac, my Quebecois friend and colleague. Nik, you are the perfect guy to work with! I would also like to thank all my SERL labmates for their friendship and help with the experiment. Working and living with such diverse and enthusiastic people was just what I expected from MIT. Thanks Karen, Elwin, Stella, Katie, Victor, Polly, Anna!

Many thanks also to Eric Mibuari and Craig Lebowitz, our two undergrads, and to all the volunteers that agreed to participate in the experiment. Your patience, insight, and intelligence were very appreciated.

Last but not least, my thanks go to my family, whose unconditional support and love over the years are the best motivation-builder I have ever known.

Table of Contents

ABSTRACT	2
ACKNOWLEDGEMENTS	3
TABLE OF CONTENTS	4
TABLE OF FIGURES	6
1. INTRODUCTION	7
1.1. REQUIREMENTS AND FORMAL REQUIREMENTS SPECIFICATION	7
1.2. THE IMPORTANCE OF SPECIFICATION REVIEWABILITY	9
1.3. MEASURING REVIEWABILITY	10
1.4. FACTORS AFFECTING REVIEWABILITY	11
1.5. THE CASE FOR VISUALIZATION	12
1.6. DESIGNING VISUALIZATIONS	13
2. BACKGROUND AND RELATED RESEARCH	14
2.1. STATE MACHINES	14
2.2. ON THE REVIEWABILITY OF FORMAL REQUIREMENTS SPECIFICATION LANGUAGES	15
2.2.1. <i>Readability of formal specification languages</i>	15
2.2.2. <i>SpecTRM</i>	16
2.3. ON VISUALIZATION IN RELATED FIELDS	19
2.3.1. <i>What cognitive science says on visualization</i>	19
2.3.2. <i>Visualization in programming languages</i>	20
2.3.3. <i>Visualization in human-computer interaction</i>	22
3. SAMPLE VISUALIZATIONS OF FORMAL REQUIREMENTS SPECIFICATIONS	23
3.1. STRUCTURAL OVERVIEW (V1)	23
3.2. QUESTIONS-BASED TRANSITION TREE (V2)	26
3.3. FLATTENED STATE MACHINE (V3)	30
3.4. A TAXONOMY OF VISUALIZATIONS	31
3.4.1. <i>Scope</i>	31
3.4.2. <i>Content</i>	32
3.4.3. <i>Selection strategy</i>	32
3.4.4. <i>Annotation support</i>	32
3.4.5. <i>Support for alternative search strategies (flexibility)</i>	32
3.4.6. <i>Static/Dynamic</i>	33
3.4.7. <i>Other dimensions</i>	33
3.5. MAPPING THE VISUALIZATIONS	33
4. PRINCIPLES FOR EVALUATING REQUIREMENTS VISUALIZATIONS	34
4.1. MINIMIZE SEMANTIC DISTANCE	34
4.2. MATCH THE TASK BEING PERFORMED	35

4.3.	SUPPORT THE MOST DIFFICULT MENTAL TASKS	35
4.4.	HIGHLIGHT HIDDEN DEPENDENCIES AND PROVIDE CONTEXT WHEN NEEDED	35
4.5.	SUPPORT TOP DOWN REVIEW	36
4.6.	SUPPORT ALTERNATIVE PROBLEM-SOLVING STRATEGIES	37
4.7.	SHOW ROLES BEING PLAYED	37
4.8.	PROVIDE REDUNDANT ENCODING	38
4.9.	SHOW SIDE EFFECTS OF CHANGES	38
5.	EXPERIMENT DESIGN	39
5.1.	THE NEED FOR AN EXPERIMENT	39
5.2.	OBJECTIVES	39
5.3.	THE MODEL	40
5.4.	THE TOOLS	43
5.4.1.	<i>SpecTRM</i>	43
5.4.2.	<i>The visualization tool</i>	44
5.4.3.	<i>Information redundancy</i>	45
5.5.	EXPERIMENTAL PROCESS	45
5.5.1.	<i>Experiment overview</i>	45
5.5.2.	<i>Tutorial</i>	46
5.5.3.	<i>Questions and tasks</i>	46
5.5.4.	<i>Post-experiment analysis</i>	47
6.	PRELIMINARY EXPERIMENT OBSERVATIONS AND DISCUSSION	48
6.1.	EXPERIMENT LIMITATIONS	48
6.2.	OBSERVATIONS	49
6.2.1.	<i>Validity of the principles</i>	49
6.2.2.	<i>Conclusion: to visualize or not to visualize?</i>	51
	BIBLIOGRAPHY	53
	APPENDIX A. FULL-PAGE FIGURES	55
	APPENDIX B. VISUALIZING STATE TRANSITION CONDITIONS	65
B.1.	ISSUES IN REPRESENTING TRANSITION CONDITIONS	65
B.2.	COMPARATIVE STUDY OF 3 REPRESENTATIONS	67
	APPENDIX C. EXPERIMENT QUESTIONS	71

Table of Figures

Fig. 1:	State machine model for traffic light system.....	14
Fig. 2:	The structure of an intent specification for software systems	17
Fig. 3:	Zoom on a view of V1, showing the 4 state variables that control parts of the display on the Flight Mode Annunciator (FMA) of the MD-11.....	26
Fig. 4:	An example of state machine transition diagram and inverse transition diagram taken from the MD-11 vertical guidance specification. This state variable describes the vertical attitude of the aircraft during cruise.....	31
Fig. 5:	View of the MD-11 cockpit. The display units where VG sends feedback information are highlighted (white squares).....	41
Fig. 6:	Zoom on the main display units in the MD-11 cockpit.....	41
Fig. 7:	Zoom on the Flight Control Panel (FCP) of the MD-11 cockpit.....	42
Fig. 8:	Hierarchical context of the case study. The main input and output devices are also represented.	42

Full-page figures

Figure A:	Three of the 6 AND/OR tables necessary to describe the state variable “Active Add Drag Scenario” in the specification of the MD-11 Vertical Guidance Annunciation Process..	55
Figure B:	Structural overview (V1) of the specification of the MD-11 Vertical Guidance Annunciation Process.....	56
Figure C:	A reorganized view of V1. Some elements that share common properties are grouped together in “cascades”.....	57
Figure D:	Sliced structural overview. The state variable called <i>Origin of Level T_D</i> and all its structural dependencies are emphasized over the rest of the model, which is preserved as context.....	58
Figure E:	An example of Questions-based Decision Tree taken from the MD-11 Vertical Guidance Specification. This state variable (<i>Active Add Drag Scenario</i>) indicates which one of the five possible scenarios for extending airbrakes is active.....	59
Figure F:	An example of Questions-based Decision Tree taken from the MD-11 Vertical Guidance Specification. This state variable (<i>Origin of Econ Airspeed Target</i>) determines how the Vertical Guidance System will compute the <i>Econ Airspeed Target</i>	60
Figure G:	Same as Figure F, except a slicing scenario has been applied. The specification is restricted to those states that are reachable under this scenario. In this case, the scenario specifies that the flightphase is ‘Descent’ and that the current operational procedure is ‘Descent Path’ or ‘Late Descent’	61
Figure H:	Screen capture of the SpecTRM GUI.....	62
Figure I:	Screen capture of the visualization tool.	63
Figure J:	An extract of a possible textual representation of a formal specification. It is adapted from the actual specification of the MD-11 Vertical Guidance System. The state variable is “Active Add Drag Scenario”.....	64

1. Introduction

The current trend in the aerospace industry regarding the use of software is towards more complexity and functionality, which inevitably increases the likelihood of subtle errors. This is a major concern because software is used today to control many of the safety-critical devices in aerospace systems, such as satellite attitude control or aircraft guidance. Examples abound of system failure caused by errors in software: the 1999 Ariane 5 explosion and the loss of Mars Polar Lander are just two famous ones. This trend is likely to continue as software becomes more and more pervasive.

The increasing complexity of software is the result of a strong incentive for more functionality and efficiency at lower cost, which is theoretically made possible by the growing capability of processors and the apparent flexibility of software over hardware. On the Boeing 777 for example, there are more than 4,000,000 lines of code (a 6-time increase from Boeing's previous aircraft program), distributed over 79 systems.

Because this exponential growth is not matched by a parallel improvement of software development methods, it sometimes comes at the expense of reliability and safety. In a study on the role of computers in system safety, Leveson summarizes the current situation with some pessimism: "Few systems today are built without computers to provide control functions, to support design, and sometimes to do both. Computers now control most safety-critical devices, and they often replace traditional hardware safety interlocks and protection systems—sometimes with devastating results." [16], p.22]

Many of the most serious errors in software systems can be traced back to the requirements, either because these requirements were wrong in the beginning, or because they were badly implemented. Both kinds of errors emphasize the importance of the requirements phase of development.

1.1. Requirements and Formal Requirements Specification

Requirements specification is one of the earliest and most decisive stages of system development. The impact of the decisions – or mistakes – made during this phase will resonate throughout the whole lifespan of the system. Some will argue that this is less true of software systems because software is flexible and can be changed or updated anytime at little cost. Unfortunately, this is more myth than reality.

System requirements specification includes several activities, such as creation, modification or review. For the sake of clarity in this thesis, let us give a definition of these activities. *Creation* is the process of writing down the requirements, from scratch. *Modification* is the action of changing already agreed-upon requirements, which occurs inevitably. Finally, *review* is the task of identifying errors in the specification. Requirements specification is not a linear process: these three activities commonly overlap. However, they are typically under the responsibility of different people in order to minimize the risk of errors. Participants in the specification process can include systems engineers, developers, designers, quality insurance people, human factors specialists, regulation experts, final users, customers, etc. The backgrounds of all these participants are usually very diverse.

Like the other development phases, the requirements specification usually follows a well-established process. One could classify this process in either one of these two categories: informal specification vs. formal specification. Informal specification uses plain English language and/or *ad-hoc* diagrams to describe the requirements. The form of such a specification may be constrained by the tool used to produce it or by company policies. Formal specification relies on a mathematical language to describe the requirements. The form of such a specification is always constrained by the nature of this language. Formal specification is just one facet of the so-called formal methods, which Clarke and Wing define as “mathematically based languages, techniques, and tools for verifying and specifying [hardware and software] systems” [7]. Z and Statecharts are two well-known examples of formal methods.

At the core of a formal specification is a mathematical model of the requirements. This formality yields several benefits. Mathematical analysis becomes possible. It can be used to provide proofs of the properties of the model, such as consistency and completeness.

Formal specifications can also help to promote a common understanding of the required functionality of the system. This is key to domain experts finding errors and validating that the specifications describe a system that will be useful and safe in operation. All specifications, by definition, constitute a common language of communication between the parties involved; formal ones have the advantage of being accurate and explicit, providing they can be read and understood by these experts.

Finally a formal specification can come with a set of tools that can be used to relieve the users from the most tedious tasks by automating them; or to make sense of the vast amount of information in the specification in an intelligible way (by running a simulation for instance).

Overall, these benefits can result in safer and more reliable products, a reduction of cost and time-to-market, and an increase in product quality.

However, formal methods are not widely used in industry for software. Clarke and Wing [7], and Gerhart and Craigen [10] have conducted surveys of the state of the practice in industry and academia. Apart from a few successful projects in industry, formal methods remain confined to the academic world. Two reasons for this may be that formal specifications are simply not readable and do not scale well. Readability is arguably one of the most important properties of any specification [17]. As far as scalability is concerned, the scarcity of huge-scale formal specification case studies, even in academia, speaks for itself.

Making formal specification languages readable and applicable to large systems is therefore desirable; this thesis is a contribution to the efforts of the Software Engineering Research Laboratory (SERL), MIT, towards that goal. Although these two drawbacks of formal methods (lack of readability and scalability) affect all specification activities, this thesis will focus on review as a starting point.

1.2. The importance of specification reviewability

Specification review is important because leaving errors in the specification affects all the subsequent phases of the product's life. Specification review typically includes two kinds of tasks:

1. Ensuring a good translation from high-level to low-level requirements. In other words, making sure the system is specified to do what it is intended to do. This kind of review is generally done by domain experts who did not necessarily write the specification.
2. Finding errors in the specification itself. These errors range from typos to local incompleteness to major inconsistencies. This kind of review can be compared to peer review.

Defenders of formal methods argue that mathematical analysis facilitates the review of formal specifications. But this does not remove the need for them to be readable. The reasons are, referring to the two kinds of tasks mentioned above, that:

1. High-level to low-level translation cannot be easily automated, because it requires a lot of human expertise.
2. And previous work at SERL suggests that automated tools will not find all types of errors, especially the most far-reaching ones [27].

An additional issue is that review is necessarily a multidisciplinary problem, more than any other specification activity. In spite of their reliance on an abstract mathematical language, formal specifications need to be easily readable by all, including people with no background in computer science and software engineering.

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. Not only is reviewing these systems a necessarily multi-disciplinary problem, but it is also essential to avoid the kind of semantic errors that can lead the system to unsafe states. For example, a wrong understanding of the requirements of an aircraft guidance system could lead the system to provide the wrong speed target during an Engine-Out at takeoff, in which case the likelihood of accident is high.

1.3. Measuring reviewability

Reviewability is undoubtedly a desirable quality, but how is it measured? Reviewability could be defined as the efficiency with which reviewing tasks are done. But “efficiency” is highly subjective. First, the reviewing tasks will vary a lot according to the domain and particular function of the system. For example, a distributed flight entertainment system onboard a passenger aircraft (which is highly complex but has no influence on the operation of the aircraft), will not be reviewed the same way the landing gear monitoring system (which is relatively simple but safety-critical) will be. The level of expertise of the reviewers will also affect the reviewability of a given specification. For these two reasons, it is hard to measure the intrinsic reviewability of a specification language. Therefore, reviewability is more easily measured for a given task, or category of tasks. These measures can later be aggregated for all common tasks of reviewing a system or a category of systems, if needed.

Nevertheless, one notion can be helpful in any case: the cost structure of information [4], p.14]. The cost structure is a picture of how reachable the information contained in the specification is, in terms of search cost. A good cost structure makes more information available in a given time, and makes the information needed most often available first. The cost structure of an information set can be evaluated simply by measuring the time needed to find relevant pieces of information. Because most, if not all, reviewing tasks involve searching and using information in the specification, a good cost structure is likely to improve reviewability.

1.4. Factors affecting reviewability

It is important, when evaluating the reviewability of a specification language, to bear in mind the various factors that are likely to affect that property. Obviously, the inherent complexity of the reviewing task and the level of expertise of the reviewers with the specification language play an important role. Other factors, not always independent from these two, are equally important.

Among them is the quality of the notation or representation. “Quality” is intentionally used here in a vague meaning, as there will never be a “best” notation, even for a given task. Reviewers all have different experiences, backgrounds and sensibilities, which make them prefer different notations. However, it is a fact that the notation (or representation in the case of visual languages) affects the cost structure of the specification, and therefore the reviewability. In other words, there will never be only one good notation, but there are some universally bad ones.

Another factor affecting reviewability is the availability of operations for acting on the information contained in the specification. These operations range from basic string search to advanced completeness checking; their goal is to automate the tasks that are worth being automated, supposing they can be. It must be noted here that very advanced automated tools may not always improve reviewability as they also impede human expertise.

The complexity of the system obviously decreases the reviewability of the specification. Software systems often tend to show a higher level of complexity than hardware because they are not subject to many constraints that apply to hardware (such as the laws of physics, production constraints, physical separation of functions, etc). The experience at SERL 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 dealing with that complexity.

Size is also a critical factor. 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. Size and complexity are different concepts but size often incurs some form of structural complexity that decreases reviewability by degrading the cost structure of the specification. In a part of the specification of the MD-11 Vertical Guidance System, for

instance, one state variable is repeated 20 times, with only minor differences between the 20 occurrences in terms of semantics, behavior, or even between their names. This set of variables represents 40% of all the variables, meaning that more than a third of the model can be understood just by looking at one of these. Yet the only presence of this set nearly quadruples the amount of structural dependencies that the reviewers have to deal with.

Size, even when it does not imply complexity, reveals the cognitive limitations of humans. Searching among a great number of elements, within a specification that is hundreds of pages long, can become a difficult task. The limits of human short-term memory are soon reached. Other limitations, such as the tendency to lose sight of the big picture when focusing on too many details, also degrade reviewing performance.

These two last factors (size and complexity) are arguments in favor of tools that would help in navigating and understanding large, complex formal specifications. What we need here is in fact a way to “amplify cognition”. In theory, this can be any external tool that minimizes the use of internal working memory (such as paper and pencil!). In the case of formal specification, we believe that information visualization can play this role very effectively.

1.5. The case for visualization

In a compilation of papers on information visualization [4], Card, Mackinlay and Schneiderman define information visualization as “the use of computer-supported, interactive, visual representation of abstract data to amplify cognition”. This definition could arguably be made more flexible to include the much simpler static diagrams drawn on paper; within the scope of formal specification languages, which are themselves computer-based and interactive, we will use it the way Card et al. intended it.

Diagrams are a universal way to amplify cognition; *ad-hoc* diagrams are very often used in informal specifications for instance, although in a non-systematic way. It is obvious that visualization has potential benefits on cognition, less obvious why. Many cognitive scientists and computer scientists have tried to explain the appeal of graphical programming languages for instance, but they do not all agree.

An argument in the case for visualization is that diagrams can be automatically generated and displayed from the formal model itself; they require little edition effort.

But maybe the best way to convince the reader about the potential benefits of visualization of formal specification languages is to give two concrete examples of specification review tasks that could greatly benefit from adequate visualization. During the phase of specification creation, when

long lists of similar requirements are specified, errors often happen by mistake, even if the original requirements are error-free. These errors can be simple typos or more harmful semantic ones. In the specification of the MD-11 Vertical Guidance System already mentioned, the 20 almost-identical variables are actually related to 20 different pseudo-waypoints on the flight plan of the aircraft (a pseudo-waypoint is a moving waypoint that represents an important moment of the flight, such as the top-of-descent). All of them can be displayed on the pilot's Navigation Display; if one is displayed improperly, it will be misleading for the pilot, with foreseeable consequences on safety. Comparing the specified display conditions for these 20 waypoints can be a tedious task; but a visualization that comparatively graphs these conditions can help locate anomalies quickly.

The second example is of a different kind. Specifications usually focus on what a system should do, but sometimes it is important to make sure that a system will *not* transition into a given state. This is the case for instance for the airbrakes of an aircraft during takeoff: they must not be extended during that phase of the flight, when speed is already dangerously low. A visualization that shows all the impossible transitions for the corresponding state variable of the airbrakes control system would be helpful here.

1.6. Designing visualizations

Unfortunately, there are few principles to follow when designing interactive or even non-interactive graphical and symbolic notations for visualizations of formal software requirements specifications. Most research on visualization is in other fields and not directly applicable to our problem. Part of this work is to suggest principles, derived from these other fields, and evaluate them in the particular context of formal requirements specification. The resulting set of design principles can then be used to guide the design of new languages and visualization tools and to assist in critically evaluating them.

The next section provides an overview of the background on the reviewability of formal specification and a survey of related work on visualization. Then specific sample visualizations are presented and a taxonomy of visualizations is introduced. The two following chapters suggest some general design principles and address the need to evaluate them with an experiment with human subjects. The final section presents some of the early conclusions drawn from this experiment and identifies some directions for future work in the area.

2. Background And Related Research

Although there is no research done on the specific topic of visualization of formal requirements specification to date, it is possible to draw on several other fields as well as experience in the reviewability of formal specifications.

2.1. State Machines

The underlying mathematical model in any formal specification can take on several different forms. In this work we will focus on state machine models.

A state machine models system behavior in terms of states and transitions between them. Fig. 1 shows a state machine model for a simple traffic light system, called *Traffic Light*. *Traffic Light* has three different states – Green, Yellow, and Red – which are denoted by circles. The arrows between them denote transitions from one state to another. Transitions have a source and a destination state, and are labeled with triggers. The trigger for a transition is made up of events and/or conditions. An *event* is an occurrence in time, and a *condition* is a statement that can have the value True or False.

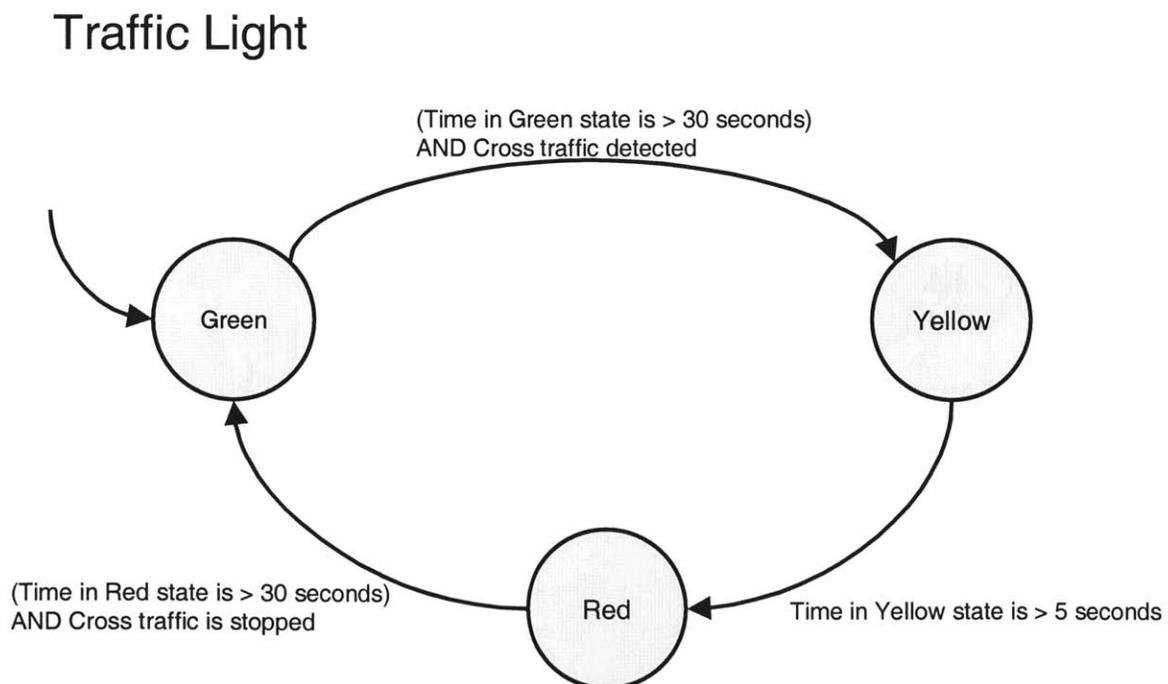


Fig. 1: State machine model for traffic light system.

A transition is taken from its source state to its destination state when the corresponding trigger occurs. For example, the system in Fig. 1 transitions from Green to Yellow when a “Time in Green State is > 30 seconds” event occurs and the condition (i.e. cross traffic has been detected) is true. An arrow with no source state denotes the starting state of the system.

Traffic Light in Fig. 1 is one *state variable* that can take one of three different *state values*. Most state machines are actually made up of several state variables that may depend on one another. In that case, the actual state of the state machine is the cross-product of the states of all the individual state variables.

It is important to note that state machines are an abstract model that can be described in several ways. Fig. 1 describes the traffic light state machine graphically, but it could also be represented textually, or in a table format.

As mentioned before, this work deals specifically with state machine-based specification languages, primarily because they seem more likely to be adopted by industry than other formal methods. The state machine model is well researched, and supported by several proven analysis techniques. It also possesses several features that make it a readable and understandable description of system behavior, arguably more than other formal models. First, state machines do not require knowledge of their formal foundation in order to be used effectively, in contrast to other formal models. In addition, we hypothesize that state machine models are a natural way for engineers to think about control systems. The behaviors of many systems are easily described using modes (states) and transitions between them. This type of description can readily be expressed by a state machine.

2.2. On the reviewability of formal requirements specification languages

One of the research objectives at the Software Engineering Research Laboratory is to determine whether “multiple notations and visualizations generated from a common model can improve the requirements review and understanding process”. This objective is part of a more global research effort on formal specification languages.

2.2.1. Readability of formal specification languages

“One of the biggest drawbacks to using formal languages is that they simply are not readable. Readability is arguably one of the most important properties of any specification” (Zimmerman, [28]). Starting from this observation, Zimmerman conducted a survey of several formal specification languages and an investigation on their readability.

Reviewability and readability are not exactly the same thing, but they are closely related. Both are properties of a specification (or of a specification language, if one is looking at the language itself rather than an instantiation of that language). But reviewability characterizes the *specific* ability to do the task of reviewing, whereas readability characterizes the *general* ability to read and understand the specification. As we have seen, reviewability can be measured, although imperfectly, whereas readability is a more abstract property. Finally, readability is not specific to the review activity: it is also useful for editing. Naturally, readability supports reviewability.

Zimmerman's research includes an experimental study to determine how various factors of state-based specification language design affect readability. These factors can be grouped into 3 categories:

- *Specifying the state machine* – How is the overview of the state machine represented (form and content)? Does it use hierarchies?
- *Specifying events* – Does the state machine description rely on internally broadcast events to order its execution?
- *Specifying transitions* – Are transitions organized by source state or destination state (perspective)? Does the language allow the use of macros? How are transition conditions represented?

Because readability is an evasive property, Zimmerman's conclusions are mixed. The effect on readability of some of the investigated factors is well understood, but in most cases further research is needed. From Zimmerman's experiment, it appears that graphical or tabular representation is preferred over text by most users; that multiple representations may be worthwhile; and that macros and hierarchies are very useful features but may reduce readability. Zimmerman acknowledges that "readability is a complex property, which we do not feel can be predicted theoretically, but rather is best evaluated by human experience", and calls for more experimentation.

Maybe the one lesson to remember from that work is that the most readable notation will vary according to many factors, some of them hard to assess (nature and size of system, background of users, problem being solved, etc.), and that among these factors the scale of the specification is particularly problematic. This lesson will very likely be valid in the case of visualization of formal specification.

2.2.2. SpecTRM

Specification Tools and Requirements Methodology (SpecTRM) is an experimental systems engineering development environment for heterogeneous safety-critical systems that includes

specification and analysis tools integrated with a safety information system. It was developed by Leveson and her students, based on “intent specification” principles [19]. These principles were drawn from ideas in the fields of systems theory and cognitive engineering, and from Leveson’s experience with safety-critical software. An intent specification has three dimensions: a part-whole abstraction (decomposition), a refinement abstraction, and an intent abstraction (see Fig. 2). The intent dimension is a hierarchical decomposition: each one of the five levels provides intent (“why”) to the level below, from system purpose to physical representation.

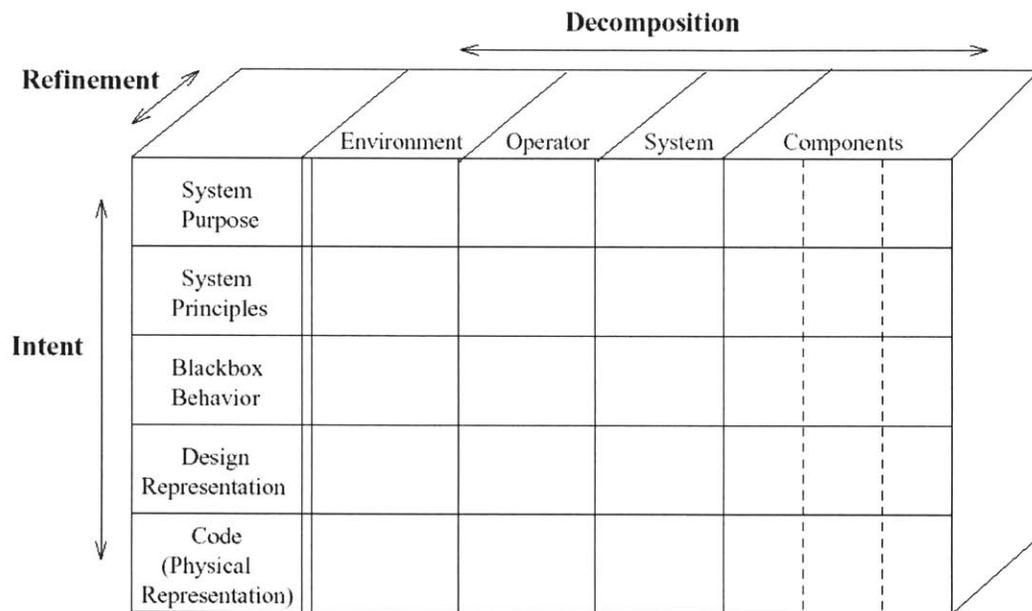


Fig. 2: The structure of an intent specification for software systems

SpecTRM is specifically designed for process-control systems [18], where embedded control software is usually real-time and reactive. The requirements for such systems are particularly difficult to specify and validate. Although SpecTRM was developed in the research laboratory, it has been successfully used in industry. The official specification of TCAS-II, a collision avoidance system for commercial aircraft, was written using RSML, the ancestor of SpecTRM.

SpecTRM-RL (SpecTRM-Requirements language) is the formal modeling language of SpecTRM. It can be used to specify the blackbox functional requirements for the system components (level 3 of an intent specification). The models are blackbox in that only externally visible behavior is specified – no internal (implementation) design is included. Separating the blackbox description from the design description makes the model easier to construct, review, and analyze [18]. SpecTRM-RL uses an underlying state machine model to describe the blackbox behavior. This state machine is represented textually, using a “coming-from” perspective (the focus

is on the destination states, not the source states). Transition conditions are represented in a tabular format.

Reviewability of SpecTRM

One objective of intent specification is to “assist in the tasks of finding information, tracing relationship between info, and understanding system design and rationale” [19], p. 9]. All these tasks are key to the reviewing process.

Indeed, SpecTRM is reviewable. “The most important result of our research is verification that building a formal requirements model for a complex process control system is possible and that such a model can be readable and reviewable by non-computer scientists.” (Leveson, Heimdahl, Reese [18]). Many of the features of SpecTRM make the reviewing tasks easier than with other languages. For example, an intent specification is well structured; SpecTRM not only enforces this structure, but it also provides help in navigating it (with a browser tree and hyperlinks). SpecTRM uses limited and consistent semantics, which reduces considerably the learning time. State transition conditions are represented with a concise tabular notation, which proved to be readable and effective [28], and make no use of internally broadcast events, which were shown to be confusing [18], [28].

SpecTRM features a built-in automatic graphical overview of the mathematical model. During the periodic reviews by domain experts of a formal specification of TCAS-II, the reviewers 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. SpecTRM also features a number of analysis and validation tools that are helpful in day-to-day reviewing, such as consistency and completeness checking. Finally, slicing techniques can be applied to SpecTRM models. Slicing is a powerful technique used to reduce the whole state machine into smaller consistent chunks that are easier to manage individually.

In spite of all this, anecdotal evidence suggests there is still room for improvement. First of all, the cost structure of a SpecTRM specification could certainly be improved. For example, the layout of SpecTRM, although clearly structured, is purely linear. It does not prioritize information, even though a reviewer’s job is actually to sort the relevant information from all the details.

Another concern with SpecTRM (and virtually all other specification languages) is its inability to scale well to very large systems. For example, the information search cost in transition tables grows exponentially with the size of these tables. Using macros does reduce the size of the tables,

but it increases the depth of the dependency graph, which also results in degraded cost structure. Finally, the notation adopted by SpecTRM does not match all the tasks being performed in reviewing (although that does not prove that other notations would do better). For instance, the lack of direct representation of the structural dependency graph makes fine structural analysis very tedious. It is also very hard to identify behavioral patterns in the transition conditions, because all the tables look the same. And as the number of lines and columns in these tables grow, it becomes more and more difficult to locate and correct even the simplest errors.

2.3. On visualization in related fields

Visualization, although not researched in the world of formal specification, has been the subject of extensive research in other fields.

2.3.1. What cognitive science says on visualization

Research into the impact of visualizations (diagrams) on cognition was pioneered by Larkin and Simon [15]. 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. 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 [6] 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.

Card, Mackinlay and Schneiderman have edited a compilation of readings on visualization. In their introduction, they recall that “the purpose of visualization is insight, not pictures” [4], p.6]. In other words, visualization is a means to an end, not an end in itself. According to them, a good visualization should be able to amplify cognition, in one or several of six possible ways:

- *Increased resources* – visualization make use of cognitive resources that plain text is unable to use.
- *Reduced search* – visualization improves the cost structure by making information more available.

- *Enhanced recognition of patterns* – visualization simplifies and organizes information.
- *Perceptual inference* – visualization can support perceptual cues that are extremely easy to humans.
- *Perceptual monitoring* – monitoring is easier if events stand out graphically.
- *Manipulable medium* – visualization allows interactive exploration.

On top of this, the natural appeal of visualization over text is an important factor. In a paper on the cognition of graphical programming [21], Petre says: “The importance of sheer likeability should not be underestimated; it can be a compelling motivator. In general, affect may be as important as effectiveness.”

2.3.2. Visualization in programming languages

Visualization in programming languages is widely used and has been extensively studied. It is relevant to this work because the context of programming languages is not so different from our problem. First, programming, just like system specification, belongs to the class of “ill-structured” problems that cannot be solved by a strictly defined procedure [23]. Second, both activities take place within a technically constrained environment, which makes them good candidates for visualization [2]. Finally, many of the cognitive tasks involved in computer programming, as identified by Blackwell et al. in [2], are also involved in system specification. These tasks include “the separate processing of syntactic and semantic information, the collection of expert knowledge into chunks, the structuring of regularly-used information into schemas, and the solution of design problems in terms of previously acquired and frequently modified plans.”

In 1979, Fitter and Green [9] proposed five principles for effective visualization design of diagrammatic computer languages. At that time, “diagrammatic computer languages” included notations such as decision tables, flowcharts, building blocks, etc. The purpose of this pioneer work was to provide sound requirements for a good graphical notation. In Fitter and Green’s own words: “The first principle of a diagrammatic notation is that the information that is encoded perceptually rather than symbolically should be *relevant*. Notations should *restrict* the users to forms that are comprehensible; they should *redundantly recode* important parts of the information; they should *reveal* the underlying process that they represent, preferably in a *responsive* interactive system which permits manipulation of the diagrams; and finally they should be readily *revisable*.” The five principles are explicated below:

- *Relevance*: any choice of visualization will have some information represented perceptually and the rest symbolically; the perceptual information should be relevant to the user's task.
- *Restriction*: diagrams such as flowcharts have the potential to be misused; therefore their use should be restricted to robust, efficient diagrams.
- *Redundant recoding*: the most important type of information should be coded both symbolically and perceptually (example: indentation in programming languages reveals the structure of the code) – redundant recoding is also called secondary notation by other authors.
- *Revelation and responsiveness*: a graphic notation that can reveal the structure inherent in the underlying data or the process by which they are manipulated, will be superior to a linear symbolic language.
- *Revisability*: revision is a common task, which should be supported, or at least not hindered, by the graphical notation.

Although devised 24 years ago for what were at that time very simple diagrammatic programming languages, these five principles seem extremely relevant to today's problem of visual formal specification languages.

More recently, Blackwell, Whitley, Good and Petre identified a set of cognitive dimensions for visual language design. Quoting their 1999 paper [2]: "Cognitive dimensions constitute a small vocabulary of terms describing the cognitively-relevant aspects of structure of an information artifact. (...) Any cognitive artifact can be described in these terms. (...) The so-called 'dimensions' are meant to be coherent with each other, like physical dimensions." The purpose of this work was to provide a framework for the description and analysis of diagrammatic programming languages. This set of dimensions, upon which any visual programming language can be mapped and analyzed, includes notions such as closeness of mapping, error-proneness, hidden dependencies, role-expressiveness, or secondary notation (the reader is referred to [2] for a detailed explanation of these notions). Many of these can be directly used or easily adapted to formal specification languages.

As mentioned before, visualization can be misleading if it does not make proper use of perceptual cues. In 1995, Petre wrote a paper on the use and misuse of secondary notation and perceptual cues in graphical programming. She recalls the potential benefit of visualization in programming languages ("The strength of graphical representations is that they complement

perceptually something also expressed symbolically”). Then she underlines the potential adverse effects (“It is important to recognize that poor use of secondary notation isn’t merely neutral, it can confuse and mislead”). She finally gives a principle for effective visualization design: “In constraint lies the key: ‘good’ secondary notation involves disciplined and appropriate application of constraints to the available freedoms of presentation.”

Storey’s 1999 paper [24] focuses not on visual programming languages but on graphical software exploration tools. These tools present similarities with our problem in that they also aim at visualizing vast amounts of complex abstract data. This paper addresses important issues, in particular the problem of integrating multiple comprehension models (eg: bottom-up and top-down approaches) and the issue of navigating very large data sets. Storey suggests 12 guidelines, or ‘cognitive design elements’, that are also useful for formal specification visualization.

2.3.3. Visualization in human-computer interaction

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 [22].

3. Sample Visualizations Of Formal Requirements Specifications

The need for visualization was made evident to us while trying to translate an existing (textual) specification of the MD-11 Vertical Guidance System into a SpecTRM model. A preliminary step was to understand it. Blindly translating the requirements into a SpecTRM model and hoping that the formal specification would be more understandable than the former one, did not work, partly because of the cognitive of SpecTRM mentioned before. So we started by making informal *ad-hoc* sketches on paper, testing several graphical representations. Then we narrowed down our efforts on 2 or 3 promising ones and took on a more systematic approach, still on paper. It soon appeared that, as users of the specification, we could benefit from having these graphs automatically generated from a formal model and displayed on a computer screen – in short, making *visualizations*.

These visualizations were developed as a complement to the specification language in use at SERL: SpecTRM. Therefore they bring no change in the fundamentals of SpecTRM: a states-based model, the use of macros, the absence of events, the coming-from perspective, and so on. The only change – but it is a fundamental one – is the representation.

3.1. Structural overview (V1)

The structural overview (referred to as V1) is a graphical overview of the structure of the whole model. It shows all the constituents as well as the dependencies between them (Figure B, p.56).

Need

The need for an overview appeared quickly as we struggled to mentally “visualize” the structure of the specification. A textual representation, because of its linearity, shows the details better than the high-level content. Even in SpecTRM, which features a browser tree in addition to the textual representation, the need was felt to shift from a linear overview (mono-dimensional) to a more concise and more natural map representation (bi-dimensional). As far as the dependencies between the different elements are concerned, this kind of representation allows the use of a meaningful perceptual cue (arrows) in lieu of textual links or hyperlinks.

Such a graphical overview was also thought to be a good basis for navigation within the specification.

Description

As can be seen in Figure B, this overview features a rather classic input-to-output layout, from the top to the bottom of the graph. The central white rectangle represents the boundary of the system, while the outer gray rectangles figure the hierarchical context of the system. External devices with which the system exchanges information are represented as boxes all around the central rectangle, possibly within the hierarchical context. The arrows that flow from these devices directly to an internal element of the system are inputs; the ones that flow in the other direction are outputs. The thin rectangles within the boundaries of the system represent internal elements such as state variables or macros. Any link pointing from one element to another (including inputs and outputs) can be considered as dependency links: the TO-element “depends on” the value of the FROM-element in order to determine its own value or state.

As always when trying to visualize a large amount of information, readability issues arise. In the case of V1, a number of perceptual cues and secondary notation are used to improve readability. For example, elements and links are categorized using colors, size, and position. Besides, the layout of V1 is interactive, meaning that users can build their own layout, which is likely to be closer to their own mental picture (see Figure C, p.57, for a reorganized view of V1).

Size is a particular issue in that kind of overview. The size of the model affects both the number of boxes and the number of links. The intricacy of the dependencies, which can be related to the apparent complexity of the model, will only affect the number of links (V1 is particularly useful in quickly assessing the depth of the dependency graph, or level of coupling). Overall large numbers of boxes and links quickly degrade readability. A rational layout alleviates this problem by grouping together elements that share common dependencies, but the fundamental problem remains, so other solutions exist, such as slicing.

Features

V1 is a manipulable medium, in the sense that users can interact with its form and content. An example is structural slicing. Slicing V1 around a given element of the specification (say, a state variable) amounts to selecting only the structural elements of the specification that are connected to this state variable, and putting all the rest in the background. Figure D, p.58, shows a sliced view of V1 around the state variable called ‘Origin of Level T_D’. Everything that is not structurally related to it is grayed out and sent to the background.

Slicing has several possible uses. One of them is just practical: slicing produces coherent subsets of the visualization that are much easier to manage than the entire model. Another possible use for reviewers is analytical: structural slicing shows the impact of one element (for instance, a

failed external device) on the rest of the system. It is important while manipulating a sliced specification to bear in mind the existence of the rest of the specification. Therefore in V1, the subset that is sliced out is not completely removed from the display, but kept in the background as context. This approach implements some “focus+context” ideas found in [4].

Another way to decrease the amount of information is through filtering. Filtering selects subsets of V1 based on properties, rather than structure. For example, one can decide to display only the state variables that depend on macros. Again, the rest of the specification is displayed in the background to remind the user that details have been hidden from the view and also to provide some context for the visible parts.

Anticipated benefits

By mapping the structure of the specification on a 2D space, V1 facilitates the building of a mental picture that we think is useful in reviewing. Because V1 is interactive, users are encouraged to manipulate the information; this way we hope they will become familiar with the system quicker. V1 can also act as a basis for navigation: a starting point in the search for information and details. We hope it will improve the cost structure.

V1 gives a quick snapshot of the level of coupling and the complexity of the model, just by visualizing the amount of links. An adequate layout also reveals structural patterns. Fig. 3 for example shows that 4 of the 36 state variables in this specification share a direct dependency with the Flight Control Computer (FCC). These 4 variables actually control different parts of the same display window in the MD-11 cockpit.

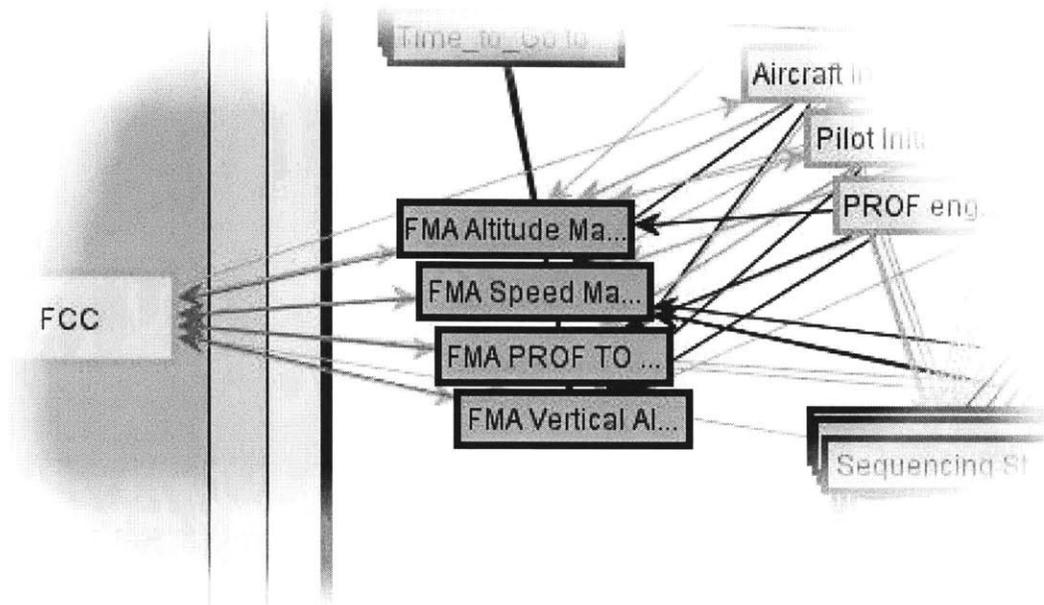


Fig. 3: Zoom on a view of V1, showing the 4 state variables that control parts of the display on the Flight Mode Annunciator (FMA) of the MD-11.

Anticipated drawbacks

Because the goal of V1 is to display all the structural content of the specification, scalability remains an issue. Our experience shows that a formal specification that contains a few dozens state variables and about 200 inputs or outputs remains manageable in V1, but the cognitive limit is not far away. One of the problems with large-scale structural overviews is that they soon become cluttered by the amount of boxes and arrows to display. This is problematic especially if V1 is used as a navigation basis. Another drawback of V1 is that it requires human intervention. Automatic generation will not produce the best layout; some human expertise is needed to make the best of it.

3.2. Questions-based transition tree (V2)

The questions-based transition tree (referred to as V2) is a visualization of state transition conditions that uses the format of a decision tree. Informal questions are added to each level of the tree to assist in understanding the conditions. V2 is the result of a preliminary study whose description can be found in Appendix B.

Need

One of the most difficult tasks of reviewing state-based specifications is to understand and analyze the most complex transition conditions. These complex conditions usually depend on many different variables; besides, there is often much more than one way to transition from state A to state B.

SpecTRM features a tabular representation of transition conditions, which shows all the conditions in parallel; we felt that this representation was inappropriate for the most complex transitions, so we investigated a representation that would treat the conditions in sequence rather than in parallel.

Description

The result is a tree representation such as Figure E, p.59. This graph represents the entire transition conditions for one state variable ('Active Add Drag Scenario' in this case). It should be read from left to right, from the root to the leaves. Each leaf is tagged with one of the possible states. Each node represents a decision that has to be made for the state variable to transition to a new state. This decision is based on the value of the parameter whose name is written just above the tree in the corresponding column; the values that this parameter has to be in in order to make the decision are written directly on top of the corresponding branches (possibly as an equation). These parameters can be inputs, macros, or even other state variables. In Figure E for instance, the first decision (at the root node) is made based on the value of a discrete input called 'Operational Procedure'. If 'Operational Procedure' equals 'Descent Path' for example, then the second of the 3 branches is active and we can move on to the next decision. Once all the decisions are made, we end up in one of the leaves, and we know what the active state is. It follows that each path from root to leaf is a possible way to transition. There may be several ways to transition to a given state; in that case, there will be several leaves that bear the name of that particular state (the case of 'None' in Figure E).

As an example of how V2 works, consider Figure E again. Suppose the following information about the system's environment is known:

- *Operational Procedure* is **Descent Path**;
- *Active Descent_Approach Segment Thrust Type* is **Idle**;
- *Active Descent_Approach Segment Speed Type* is **Mach**;
- *ADC Mach* = **0.82**;

- *Active Descent_Approach Segment Predicted Mach = 0.80;*
- *Flightphase is Descent.*

Then it is possible to follow one path in Figure E, which starts from the root and goes to the leaf labeled as ‘Scenario 3’. In other words, all the information is known for ‘Active Add Drag Scenario’ to be in the state ‘Scenario 3’.

It is important to note that a decision tree is valid regardless of the original (source) state. If a transition depends on the current state of the state variable, then the variable itself is listed as one of the parameters, i.e. one of the decisions (nodes) will be taken based on the current value of the state variable.

So far the reader has been told how to read V2 based on the formal elements of the model: inputs, state variables, and so on. V2 provides an alternative way to consider transition conditions, based on informal questions (hence the name of the visualization). Each decision in the tree can easily be “translated” into an informal question whose answer actually determines the decision. The questions are written at the very top of each column; the answers directly on top of the branches, just below the value of the corresponding parameters. Note that the answers are not explicitly written if they are obvious. Such annotations help the users of the visualization to understand the reasoning involved in the decisions underlying the contents of the transition conditions.

V2 is a concise notation; Figure E for instance is the equivalent of 5 pages of textual specification and 6 SpecTRM tables. Yet conciseness does not necessarily guarantee readability. The size and position of the many textual elements in V2 is challenging. Perceptual cues such as color, font type and size are used to enhance readability. An algorithm also allocates vertical space to the different branches in order to optimize screen space use.

Size is also an issue, either because the state variable has many different states, or because the transitions rely on many different parameters, or both. The order of the decisions (which parameter should be displayed at the root level, which one next, etc.), although completely arbitrary in theory, determines what the tree will look like. Clearly, some orderings will yield more readable trees than others. It is certainly possible to devise algorithms that determine what ordering would yield the most concise tree, although it is not sure whether this tree would make more sense than one whose ordering was custom-defined by the reviewer.

Figure F, p.60, shows V2 for the biggest (and most complex) state variable in our specification of the MD-11 Vertical Guidance System. It is the equivalent of 20 pages of textual specification and 12 SpecTRM tables. Clearly, this tree is too big to be usable without further simplification,

although it is arguably better than several 21*28 tables. Slicing is again one possible way out of this dilemma.

Features

V1 features *structural* slicing; similarly, V2 features *behavioral* slicing. Behavioral slicing is a powerful way to reduce state machines by applying a scenario to the system. For the vertical guidance system of an aircraft, a possible scenario is: “The flightphase is descent, but the aircraft is still above the normal trajectory – the current operational procedure is ‘late descent’”. Naturally, reviewers are free to apply whatever scenario they deem relevant to their tasks. A scenario typically constrains the inputs of a system, but constraints on internal elements are also possible. The reader is referred to the work of Heimdahl and Wahlen [14] for more details.

Under a given scenario, some transitions become impossible. The state machine is reduced (sliced) to the set of reachable states, which is by definition smaller than the original state-space. V2 visualizes sliced specification by sending to the background all the paths of the tree that are not in the slice. Figure G, p.61, shows the biggest tree of the MD-11 model under the scenario described above. Notice how the tree is much more readable than before. Improving readability by producing coherent sub-trees is actually one possible use of slicing. Another potential use is analytical: it can be used to answer questions such as: “how will the system behave under this set of conditions?”

As with V1, preserving context while dealing with sliced trees can be important, so the discarded paths are not altogether removed. Instead, they are grayed out and compressed, making the rest of the tree stand out in the foreground.

Interactive decision re-ordering was also investigated. Although allowing reviewers to manually reorder the columns of the tree (and thus, the decisions) would add to the flexibility of V2, it appeared that novice users were confused by the fact that even moving one single column radically affects the shape of the tree. This feature was later removed.

Anticipated benefits

We expect V2 to reduce the cognitive load by increasing the use of perceptual cues (e.g. a node with several branches represents a meaningful decision), over textual or tabular notations. The juxtaposition of a formal and an informal interpretation of the same data is expected to help users that are not accustomed to formal notations. We also hope that the cost structure will be improved by the fact that all the transition information is visible on the same screen, and not scattered over several pages and tables.

Finally we think that V2 will help in identifying patterns in transition conditions, which is a common reviewing task. An example is given in Appendix B.

Anticipated drawbacks

As mentioned before, the scalability of V2 is a serious issue. Figure F already goes beyond what is cognitively manageable in a single tree, without slicing. Even more complex transition conditions would require the ability to scroll or zoom in and out on the tree, which reduces its attractiveness. In comparison, textual and tabular representations do not scale well either, but at least they do not assume that all the information is visible on one screen.

A potential adverse effect of a sequential representation of conditions such as V2, as opposed to a parallel representation such as in the SpecTRM tables, is the false impression of sequence in *time*. In the state machine, the decisions are not made in the order in which they are represented in the tree (which is determined by other considerations such as the conciseness of the tree); instead, parameters vary “randomly” as the system’s environment varies. This impression can be confusing to novice users.

A final anticipated drawback is that the automatically generated V2 will often not be readable without some user’s intervention to increase the size of the textual areas or change the width of the different columns. Some of this is already automated but it could be improved.

3.3. Flattened state machine (V3)

The flattened state machine (referred to as V3) is a visualization of part of a state machine, consisting of one or more state variables, using the traditional paradigm of circles (to represent states) and arrows (to represent transitions).

Need

The traditional graphical representation of state machines, such as the traffic light state machine in Fig. 1, is attractive because of its simplicity. For state variables that exhibit complex transition conditions however, this representation becomes impractical because the conditions are too large to be written on the arrows. Still, such a representation remains useful because it makes explicit a piece of information that is hidden in the SpecTRM tables and decision trees: which transitions are possible, and which are impossible, within a given state variable or cross-product of several given state variables.

Description

The result is V3 (Fig. 4), a graphical representation of a state machine, without the details of the transition conditions. Two redundant graphs are shown: the one on the left-hand side makes

visible the *possible* transitions, while its neighbor shows the *impossible* transitions. Although this second form is strictly redundant, it appears to be more suitable in some cases. 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.

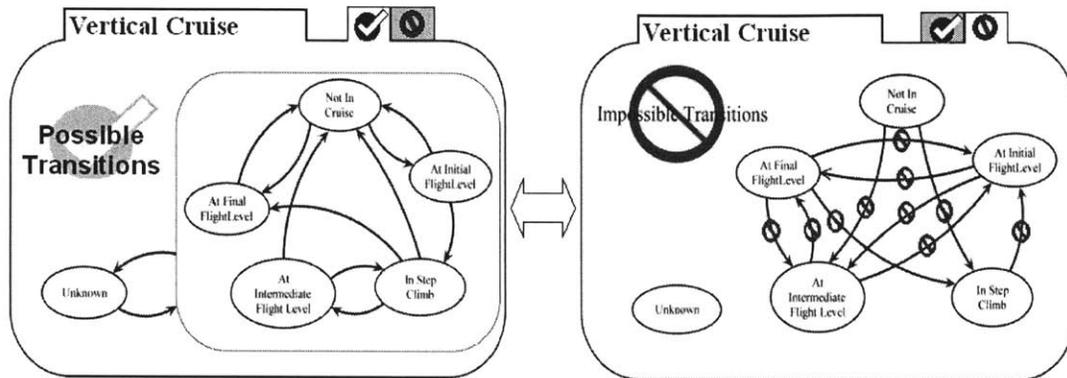


Fig. 4: An example of state machine transition diagram and inverse transition diagram taken from the MD-11 vertical guidance specification. This state variable describes the vertical attitude of the aircraft during cruise.

Only one state variable is visualized in Fig. 4. It is also possible to “flatten” several state variables together in the same visualization. In this case, the number of circles is equal to the product of the number of states of all the selected state variables.

Anticipated benefits

Compared to V1 and V2, V3 is extremely easy to use. It provides highly valuable information (although limited in scope) at very little cost. Besides, it scales well to large state machines, as long as it is not used to display the cross-product of more than 2 or 3 state variables.

3.4. A taxonomy of visualizations

As mentioned before, there is certainly no optimal visualization, even for a given task. When investigating several designs, it is necessary to be able to classify and compare visualizations. A taxonomy of visualizations is introduced below, based on 6 independent dimensions: scope, content, selection strategy, annotation support, flexibility, and static vs. dynamic.

3.4.1. Scope

Information in a formal specification can generally be labeled as structural or behavioral information. Structure refers to the architecture of the model, including all the dependency

relationships between its elements. Behavior refers to how the specified system behaves under a given set of inputs. A visualization's scope can be either structure or behavior.

3.4.2. Content

Except for very simple systems, visualizations that try to display the entire content of the model will not be effective: some details must 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.

The levels and nature of elision are part of the selection strategy (next dimension).

3.4.3. Selection strategy

The selection strategy is how the displayed content is selected. 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.

This dimension can also characterize the available selection strategies for user-derived visualizations (visualizations obtained from the original visualization through further elision of the content).

3.4.4. 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. The objective is to improve reviewability of the model by non-experts.

3.4.5. Support for alternative search strategies (flexibility)

This dimension characterizes the ability for the user to choose a problem-solving strategy, using the visualization. The level of flexibility usually reflects the intended use of the visualization. More flexibility can be valuable but it often means more difficulty in using the visualization. On the other hand, fewer visualizations may be needed for the same number of tasks. The assumed level of expertise of the users is important here.

3.4.6. 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.

3.4.7. Other dimensions

This taxonomy is not a rigid set of dimensions. Others can be added that complement or discard the ones introduced here. These 6 dimensions were useful for us to classify our visualizations, but it remains to be seen whether they can be generalized to any kind of formal specification visualization.

3.5. Mapping the visualizations

The following table maps V1, V2 and V3 onto this taxonomy.

	Structural overview (V1)	Decision tree (V2)	Flattened state machine (V3)
1) Scope	Structure Visualization based on dependency links between the elements of the model.	Behavior Visualization shows transition conditions for a given state variable.	Behavior Visualization shows possible transitions for a given set of state variables.
2) Content	Elided Visualization based on global model but details elided for readability.	Elided Visualization based on a single item of the model.	Elided Visualization based on a single item of the model; Transition conditions hidden.
3) Selection strategy	Filtering <i>(Non-structural information is hidden)</i> – Slicing available to produce derived visualizations.	Slicing <i>(The only information displayed is related to the selected element)</i> – Further slicing available to produce derived visualizations.	Slicing <i>(The only information displayed is related to the selected element)</i>
4) Annotation support	None Visualization based on formal model only.	Yes Informal questions support the understanding process.	None Visualization based on formal model only.
5) Search flexibility	High <i>(Visualization designed for high level reviewing)</i> – Search strategies include top-down review, slicing, filtering, etc.	Moderate <i>(Visualization designed for advanced reviewers)</i> – Search strategies include slicing and condition re-ordering.	Limited <i>(Visualization is limited in scope)</i>
6) Static vs. Dynamic	Static	Static	Static

4. Principles For Evaluating Requirements Visualizations

As mentioned before, not all visualizations are useful – sometimes they may even be misleading. We need criteria for evaluating potential visualizations and for creating effective ones. Research on visualization in the fields of visual programming and human-computer interaction helped us to compile principles that can be used as guidelines. These principles are presented and explained below, along with illustrations of their applications in V1, V2 or V3.

4.1. 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 [13]. 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. Readability and understanding is enhanced by decreasing the semantic distance. By providing visualizations that are close to the standard engineering models of complex systems, we have found that acceptability and usability of formal specification languages is enhanced among people in industry who previously rejected out of hand the use of such specification languages [19].

A visualization, as we intend it, is a true reflection of a specification. The design of the visualization does not affect the content of the specification: it only emphasizes some aspects of it. However, being a graphical interface between the model and the user/reviewer, it directly influences the user's mental model. For example, V1 is a map representation; it tends to impose a "geographical" mental model of the specification on the user. The power of visualization lies in its potential to impose particular perspectives on users, for better or worse. V2 for example, which is a sequential representation of transition conditions, naturally gives the false impression of an inexistent sequence in time.

Because of this, a good visualization has the potential to reduce the semantic distance if two conditions are met: 1) that it gives a fair and useful representation of the specification (the meaning of "fair and useful" will vary a lot with the context); and 2) that it conveys meaning in an attractive manner. This last point can be achieved by providing notations that are close to reviewers' traditional representations of things, but it is not necessary. Zimmerman [29] showed that users may

prefer new notations if they are effective (namely, the tabular representation of transition conditions).

4.2. 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 [11]. 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.

In the case of specification reviewing, a very common task is information searching. Virtually all elaborate reviewing tasks rely on it. In that context, V1 was designed as a basis for navigation, a kind of spatial directory from which all details can be accessed.

4.3. Support the most difficult mental tasks

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. The decision tree for instance, was created only because other notations showed their limits in analyzing the most complex behaviors written in a specification.

On the other hand, if for a given task the cognitive load is low, inserting a layer of visualization between the formal model and the problem-solver may just make the task harder than it really is.

4.4. Highlight hidden dependencies and provide context when needed

A hidden dependency is “a relationship between two components such that one of them is dependent on the other, but that the dependency is not fully visible” [12]. A typical hidden dependency is a one-way pointer in a programming language: it can be followed in one direction but not in the other. Hidden dependencies are often associated with inadequate cost structures. In

the context of formal specification, hidden dependencies are all the relationships that the notation does not make visible. Because formal specifications of software systems are characterized by a high number of dependencies, and because any notation makes some dependencies clear while obscuring others, it is important when designing a visualization to identify which dependencies are important for a particular task.

For example, a frequent reviewing task is to analyze the consequences on the system of a failure of some device external to the system. What matters in that case are the structural dependencies of the formal model. Structural slicing in V1 was implemented with that kind of task in mind (although experience showed that it can be used for a variety of other tasks).

Because a formal model contains a lot of information, any visualization will display some content of the specification while leaving the rest invisible. Whatever the content selection strategy, it is important to address the need for context. It might be useful to remind the reviewer that what he or she is seeing is just one perspective and one part of the entire model. Otherwise we are just hiding more dependencies. In a visualization, this is typically done visually by providing some perceptual cue that part of the specification has been elided. If a slicing strategy is applied to V1 or V2 for instance, the inactive part of the specification remains visible in the background and can be pulled back to the foreground at any time. Providing context often implies trading off between information (show as much information as possible) and conciseness (limit context to the bare minimum so that the focus is on the active part of the specification). Minimizing the search cost for the task at hand is a good guide to balance the two.

4.5. Support top down review

As mentioned before, the favorite feature of SpecTRM among the TCAS-II reviewers was its graphical overview. When dealing with huge amounts of requirements information, reviewers need a high-level entry point that gives them access to the details and where they can come back regularly. This is an example of the *gestalt* effect in cognitive psychology in which providing an overview makes overall structure or relationships visible or clearer [21]. This kind of top-down approach is well supported by graphical overviews.

This idea was at the origin of having structure and behavior visualized separately (V1 and V2), and providing bridges between the two.

4.6. 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" [2]. 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 [21], [25]. 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.

Designing a visualization of abstract data is always a tradeoff between flexibility and such notions as specificity, simplicity, or efficiency. For example: introducing decision re-ordering in V2 adds flexibility at the expense of simplicity. It allows more expert users to shift their strategy along the way by introducing one more degree of freedom. On the other hand, it makes the visualization more difficult to understand and can confuse novice users.

4.7. 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 [20]. Modes are a common way of abstracting and grouping important subsets of behaviors of the overall system behavior in control systems. 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.

4.8. Provide redundant encoding

Any representation makes some tasks easier and others harder [9] (which is one reason why visualizations were developed on top of SpecTRM). 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. Strictly speaking, V2 and the SpecTRM tables are perfectly redundant representations of the transition conditions (setting aside the informal annotation in V2 for a moment). Some tasks are easier using V2, such as finding behavioral patterns or comparing the conditions of two state variables, while other tasks are more efficiently done using the tables (editing tasks in particular).

Although redundant encoding can be desirable, a compromise has to be found between multiple visualization designed specifically for a large number of tasks on the one hand, and the simplicity of a limited number of visualizations on the other hand.

4.9. Show side effects of changes

It is a fact that changes in requirements are difficult to handle, because of the intricacy that characterizes software systems specifications. Visualizations should allow investigating the impact of a change in one part of a specification on other parts (showing the indirect effects of changes).

This list of principles calls for a few comments. First of all, it is by no means an exhaustive set. In addition, it must be emphasized that some of these principles may conflict. For example, enforcing top-down review may decrease the flexibility of a visualization for alternative search strategies. Providing redundant encoding may result in enlarging semantic distance by diluting the mental model. In the end, these principles should be used as guidelines, not rules, by visualization designers.

5. Experiment Design

5.1. The need for an experiment

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. Although the visualizations described in an earlier 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 designed an experiment with human subjects to validate the application of the principles to formal specifications.

5.2. Objectives

The objective of this experiment is two-fold:

1. To evaluate formally the usefulness of 2 visualizations (namely, V1 and V2) in specification reviewing. These visualizations were evaluated as stand-alone tools for one part, and as complements to a formal specification language (SpecTRM) for the other part.
2. To evaluate informally (subjectively) the validity of four of the principles mentioned above in order to be able to refine them in the future. The selected principles are:
 - Highlight hidden dependencies and provide context when needed
 - Support top down review
 - Support alternative problem-solving strategies
 - Provide redundant encoding

They were chosen because they are thought to be fundamental (hidden dependencies, top-down review) or because they are controversial (alternative strategies, redundant encoding).

It is important to note that a formal (objective) evaluation of the principles is not possible. Doing so would require being able to completely “isolate” the principle and subtract it from the visualizations, in order to compare the performance of reviewers with and without it. In effect, “removing” a principle, even if possible, would profoundly affect the visualizations and most of the other principles underneath it. The result would be a completely different set of visualizations, and it would be impossible to know if the observed difference in performance is actually the result of

the removal of a particular principle, or the consequence of the visualizations being entirely redesigned. As an example, consider a visualization that would play the role of V1 *without* supporting top-down review...

If we still want to evaluate the principles, we must consider a less straightforward strategy (described later) and accept the fact that the evaluation will always be subjective.

5.3. The model

The formal specification we used as a case study for the experiment is derived from a Honeywell experimental specification of the Vertical Guidance System of the MD-11, an airliner produced in the 1990's by McDonnell-Douglas. This specification, originally written by Lance Sherry in 1989, presents some characteristics of formality. A translation of it into a SpecTRM model was made at SERL in 2000, and completed in 2002.

Vertical Guidance (VG) is a function of the Flight Management System (FMS) of the MD-11. Its role is to compute altitude, speed, thrust and pitch targets, so that the aircraft follows a pre-established vertical flightplan. These targets can be sent to the autopilot if it is active, or simply displayed in the cockpit as an advisory to the pilots. Vertical Guidance is separate from Lateral Guidance.

One of the functions of VG is to provide feedback to the pilot, not only in terms of targets as we just mentioned, but also in terms of situation awareness. For example, VG informs the crew about the mode the system is currently in, about whether the aircraft is flying above or below its ideal trajectory, about whether airbrakes should be extended or not, etc. This whole feedback function is known as the 'VG Annunciation Process'. It is this function that we used as a case study for the experiment.

VG Annunciation Process sends outputs to most of the display units in the cockpits, including the Primary Flight Display (PFD), the Navigation Display (ND), and the Multifunction Control/Display Unit (MCDU) (see Fig. 5 and Fig. 6). In addition, it provides data to the flight control computer, also for display purposes.



Fig. 5: View of the MD-11 cockpit. The display units where VG sends feedback information are highlighted (white squares).

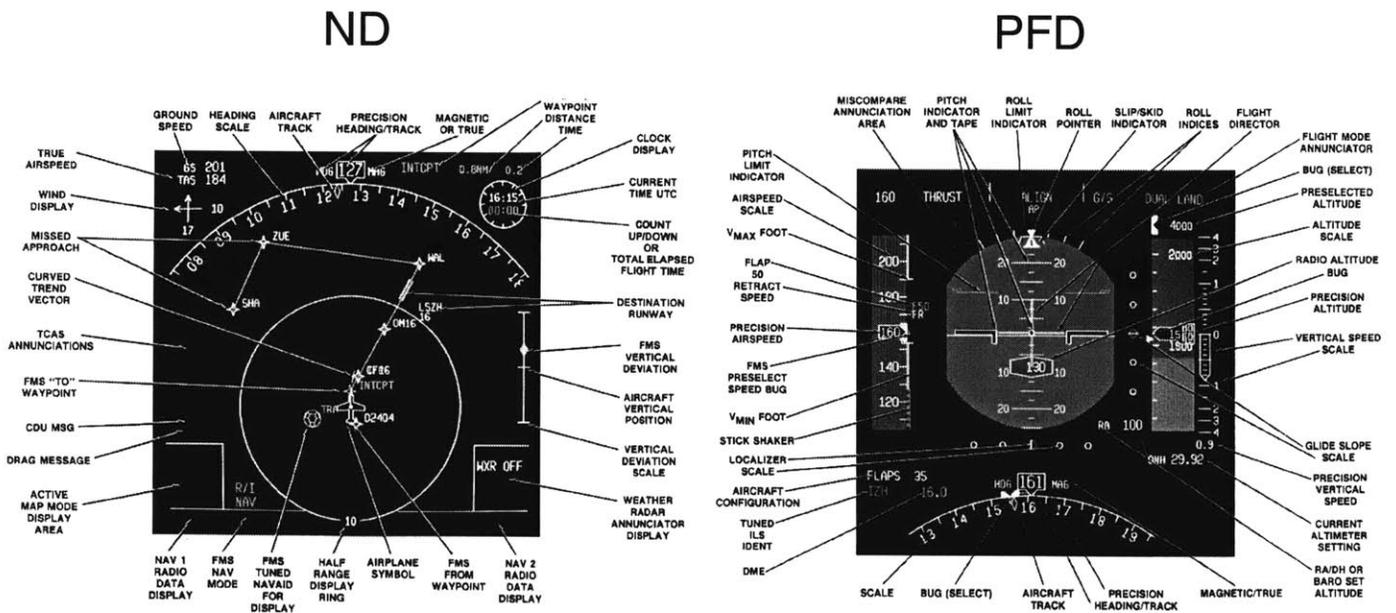


Fig. 6: Zoom on the main display units in the MD-11 cockpit.

VG Annunciation Process receives its inputs from the other processes of VG, from other functions of the FMS (such as Lateral Guidance or Navigation), and from other systems external to the FMS. One of these is the Flight Control Panel (FCP, Fig. 7). The FCP is the main flight control interface of the aircraft. The pilots use it to toggle the autopilot on and off, and to bypass the guidance functions by manually entering targets.

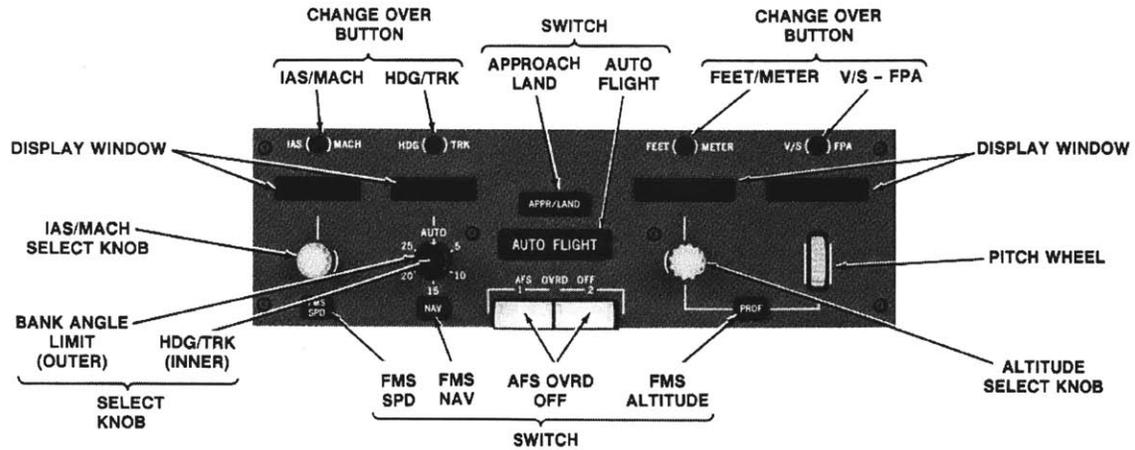


Fig. 7: Zoom on the Flight Control Panel (FCP) of the MD-11 cockpit.

Fig. 8 offers an overview of the case study in its hierarchical context, along with the main IO devices.

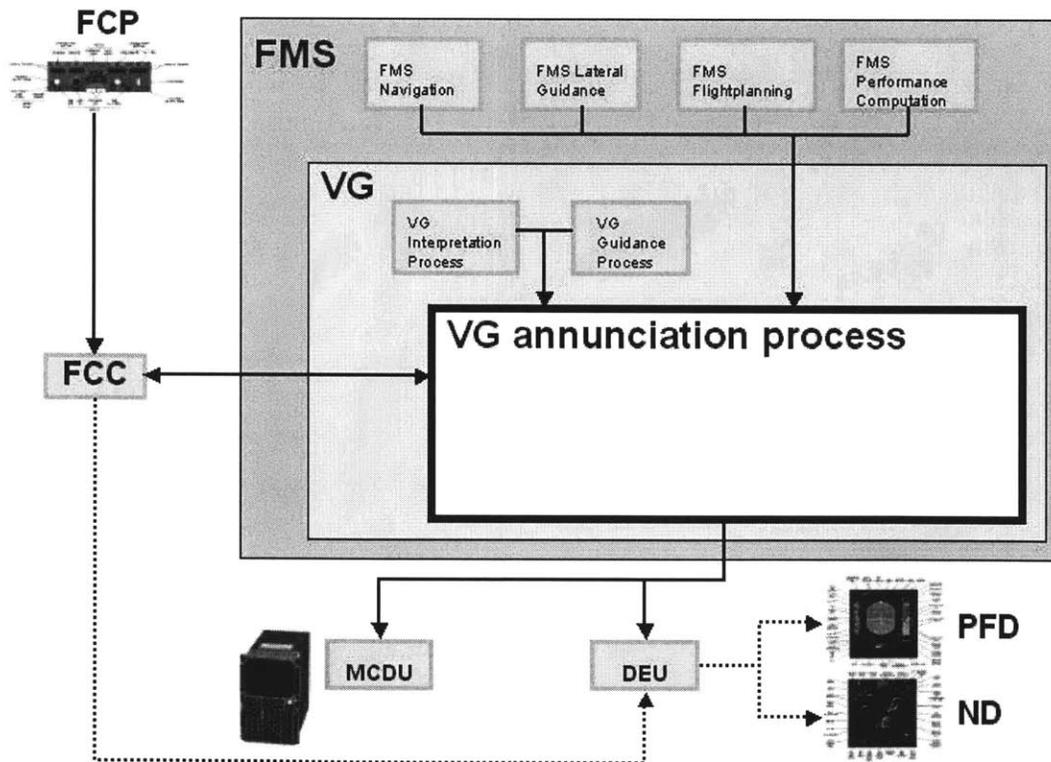


Fig. 8: Hierarchical context of the case study. The main input and output devices are also represented.

The state-based specification of this system is relatively large: it has 36 state variables, which together make a state-space of about 10^{18} states. In addition to that, the model computes 11 continuous variables. The state of the system depends on a total of 120 input variables from 10 different devices. It took a few months for two graduate students to understand the original specification and translate it into this formal model.

The VG Annunciation Process is also a fairly complex system. This complexity is semantic (meaning and behavior of its different components); it is also functional (several interdependent functions coexist); finally it is structural (the level of coupling between the various elements is high).

This model was chosen for several reasons. First of all, it is real. Second reason: it is safety-critical. A wrong feedback to the pilots can lead to hazardous situations. Finally, it was chosen because of its practicality: it has the appropriate size and complexity for our purpose. It is reasonable for two persons to implement it in a few months, yet it is large and complex enough to give the subjects of the experiment the impression that the system is impossible to grasp in the duration of the experiment.

There are some constraints associated with this choice, however. Being real and complex, it requires a certain level of expertise. This implies designing an experiment that does not rely on this expertise (difficult to do), or providing the subjects with a very long tutorial (which was our choice).

5.4. The tools

5.4.1. SpecTRM

The principles of SpecTRM have already been explained. This section presents an overview of the SpecTRM GUI. The interested reader is referred to the SpecTRM user manual for further details.

The SpecTRM GUI is made of 2 distinct panels (Figure H, p.62). The left panel is called 'project browser'. It is a tree similar to the file browser found in Windows Explorer. It is used to navigate within the specification. The right panel, called the 'project editor', can be assimilated to a text editor. The entire model is displayed in this panel and can be edited from there.

The level 3 of SpecTRM (blackbox behavior) is composed of elements, or modules, of one of the following types: state variables, macros, (external) devices, inputs, outputs, and functions. Other types exist but were not used to specify our system. In a SpecTRM specification, the different

elements are specified linearly and grouped by categories. For a given element E, two lists of hyperlinks reflect the structural dependencies of this element. The ‘references’ are the names of all the other elements that directly influence the behavior of E; under the field ‘appears in’ are listed all the names of the elements that are in turn directly influenced by E (see Figure H for an example).

Some categories of elements, such as state variables, need to have their behavior (transition conditions) specified. As mentioned before, SpecTRM uses AND/OR tables for that purpose, such as in Figure A, p.55. Additional information, such as informal comments, can also be entered for each element of the specification.

There are two ways to navigate a SpecTRM specification: using the project browser and following hyperlinks. Depending on the current task, one way will be preferred over the other.

There are constraints associated with the size of the display. In the project editor panel, there is room for about only one element, sometimes less. This makes some tasks (such as comparing two different state variables) rather difficult.

5.4.2. The visualization tool

The visualization tool is a GUI coded in Java that is able to display V1 and V2 in two separate panels and simultaneously interact with them (see Figure I, p. 63). The upper panel (V1) is the structural overview of the specification. The user can interact with it directly (double-clicking a box slices around it; the boxes can be dragged using the mouse) or by using a menu that provides further options for slicing and changing the layout.

The lower panel (V2) can contain as many decision trees as necessary (although only one is visible at a time). Again, slicing functions are easily accessible with a click of the mouse. A third panel, at the bottom left, contains a description and comments about the selected element.

Navigation in the visualization tool is done using the upper panel, which can be seen as a map of the specification. Clicking on a box selects the corresponding variable, macro, or device. If behavioral information is available for that element, the corresponding decision tree appears in the lower panel, and textual description is displayed in the information panel.

As in SpecTRM, there are constraints associated with the size of the display. It is usually not possible to visualize at the same time the whole structural overview and a large tree. All panels are resizable so that the whole screen space can be affected to either V1 or V2, if necessary.

5.4.3. Information redundancy

Because we want to evaluate the visualizations as a stand-alone tool as well as as a complement to SpecTRM, all the tasks of the experiment have to be doable with both tools. Therefore there is an almost perfect redundancy of information between the two tools (SpecTRM and the visualizations). What differentiates the two is really the representation.

However, small differences in the content occur. They are related to specificities of the representation. The informal questions in the decision trees are one example. In no case is this asymmetrical information required to do the tasks of the experiment.

5.5. Experimental process

5.5.1. Experiment overview

This experiment was run on 12 subjects, all of them graduate students in the fields of computer science or aerospace engineering. All of them had no or little experience with SpecTRM and formal methods in general, and none was familiar with vertical guidance systems.

The experiment consisted of a tutorial (about 1 hour) followed by 3 sets of questions and tasks (between 1 hour and 2 hours depending on the subjects). Each subject was asked to answer the questions using the tools he or she had been trained on, in front of the experimenter who will guide him/her and record his/her performance.

All the subjects were asked the same questions in the same order, but using one or both of the tools randomly. In the end, all subjects answered one set of questions using SpecTRM only, one question using the visualizations only, and one question where both tools are available.

The visualizations were formally evaluated against and in complement with SpecTRM, using a “performance measure” that takes into account the quality of the answer and the time required to answer the questions. All questions had one and one only best answer.

In addition, many other parameters were observed that would help informally evaluate the principles and balance the results. The two most straightforward observations were the subjects’ backgrounds and the subjects’ evaluation of the difficulty of each individual question. Less simple but no less important, was the monitoring of the subjects’ method and behavior. What tool was used (if there was a choice)? What navigation or search strategy did the subject employ? What visualization features were used? With what level of success? What confused the subject?

5.5.2. Tutorial

In order to make the measured performances statistically relevant, and in order to be able to compare the observed behaviors, it is important that all subjects start the experiment with the same level of knowledge in formal methods, in state machines, in using the tools we provide, and in Vertical Guidance systems. Although it is not possible to expect from all subjects the exact same expertise in all these fields, it is possible to reduce the differences by recruiting subjects with rather uniform backgrounds and by providing them with a long tutorial.

This tutorial is made of 3 parts:

1. An introduction to vertical guidance systems and a presentation of the model itself.
This part explains the boundaries of the system, its inputs and outputs.
2. An introduction to formal requirements specification and state machines, and a tutorial on the two tools. This part focuses on “know-how” by showing how to do typical tasks using the two tools.
3. A practice session where the subject is asked to answer a question of the same difficulty as the experiment’s questions, with the help of the experimenter.

This tutorial aims at giving each subject all the necessary knowledge and sufficient practice to be able to answer the different questions of the experiment.

5.5.3. Questions and tasks

A copy of the 3 questions can be found in Appendix C. The following table shows the randomization of tool usage.

Subject #	Question 1	Question 2	Question 3
1	S	V	S+V
2	S+V	S	V
3	V	S+V	S
4	S	S+V	V
5	S+V	V	S
6	V	S	S+V
7	S	V	S+V
8	S+V	S	V
9	V	S+V	S
10	S	S+V	V
11	S+V	V	S
12	V	S	S+V

V = visualization
S = SpecTRM

Each question has two parts. Although there is no official time constraint in this experiment, subjects are encouraged to move on if they are stuck on one half-question more than 20 minutes, which is comfortable enough to answer most questions.

5.5.4. Post-experiment analysis

As mentioned before, the objectives of this experiment are to 1) objectively and comparatively evaluate tools and 2) informally assess the validity of some principles. These two objectives are met by aggregating and analyzing the data received from each subject's performance. However, the analysis has to be very different for the two objectives, because they are themselves very different in nature.

Evaluating the visualizations can be done objectively by charting the *measured* performance of all subjects on each question according to the tool that was available. Thanks to the randomization of tool usage, the results are statistically significant in spite of the limited number of users. A higher number of subjects would only increase our confidence in the results.

Even if this method gives an objective evaluation of the visualizations for reviewing our model specification (as stand-alone tools or as a complement to SpecTRM), the results have to be analyzed in the light of external factors such as: the background of the subjects, the design and difficulty of the experiment, or the relevance of our choice of specification.

The principles, for reasons we have seen before, can only be evaluated subjectively. The idea is to observe as acutely as possible the subjects' behaviors and actions while doing the tasks, and to analyze and discuss them afterwards. If behavioral patterns emerge that can be linked to the application of some principles to the design of the visualizations, conclusions about these principles can then be drawn.

In order to assist the experimenters in this investigation, the whole experiment is recorded (sound and screen capture). Because the experimenters themselves can introduce a personal bias in this analysis, post-experiment debriefings were conducted in common. While two of the experimenters (including the author of this thesis) were directly involved in proposing the principles, building the formal specification, and designing the experiment, a third one joined the team later.

6. Preliminary Experiment Observations and Discussion

At the time this thesis is being written, the experiment is underway and it is too early to draw final conclusions. A full account of the experiment and its results can be found in a sister thesis by Nicolas Dulac, expected in the summer of 2003. However, preliminary observations can be made.

6.1. Experiment limitations

Running this experiment revealed a number of limitations that were overlooked during the preparation. Three of them in particular have the potential to disturb the initial objectives (although it is too early to say to what extent).

The background of users is the first concern. Hiring subjects in the pool of graduate students of 2 departments at MIT had the advantage of being simple and providing homogeneity, which removes one variable. However, dealing with graduate students is less realistic than dealing with experienced systems engineers (which are the real target of this research), partly because the experimenters themselves share the same background. In addition, the results of the experiment would gain in credibility if the experiment could be run on more subjects.

Another important limitation is the scope and duration of the experiment. Because we wanted it to be realistic, the questions were necessarily challenging. Although this allowed us to really test the subjects' ability to use the tools they were provided with, some of them grew tired (and sometimes discouraged!) rather quickly. This may have affected the answers to the last questions. On the other hand, we were interested in seeing if the subjects were able to change their strategies during the experiment, which requires time. In that respect, we would have liked to run the experiment even longer.

The last limitation has to do with the fact that all users were novices in the field of formal requirements specification. In order to be able to understand the specification and use SpecTRM and the visualizations, the subjects had to go through a one-hour long tutorial before even answering the first question. This in itself made the experiment longer than 2 hours. It appeared however, that this tutorial was not long enough for some subjects to acquire all the skills they needed; this imbalance showed in their performance. One can argue that in the real world, some people also understand faster than others. But the learning process is usually spread on a much longer period of time, with more occasions to practice.

6.2. Observations

6.2.1. *Validity of the principles*

From our observations of the subjects so far, it is already possible to make some interesting comments about the four principles we decided to evaluate.

Highlight hidden dependencies

The specification we used offered many hidden structural dependencies, in one of two ways. The first category is that several state variables depend on a high number of inputs or other elements (up to 20), which themselves affect a great number of other state variables. In these cases, the individual dependencies tend to be obscured by the mass of information, both in SpecTRM and in the unsliced V1. In order to deal with that, structural slicing was very appreciated. Its use in V1 appears to be fairly intuitive, once the concept of slicing itself was understood (one subject misused the visual slicing because she did not understand what it really meant). Subjects were able to isolate structurally coherent subsets, which was useful in several questions. They were also able to transition from sliced to unsliced views with great ease. The fact that context remains visible is thought to be important, although we have no objective evidence to prove that.

However, V1 was designed in a way that structural dependencies of inputs and outputs cannot be sliced. Subjects had to search the “raw” data to isolate these structural patterns: V1 is not able to highlight these dependencies. On the contrary, it tends to obscure them, compared to SpecTRM. Most subjects were confused by this apparently inconsistency within V1, and performed poorly on the questions that involved input influence.

Indirect structural dependencies (those that involve several levels of dependency) are also naturally “hidden” in both SpecTRM and the unsliced V1, because it would be impractical to represent them explicitly for the entire model. These dependencies are an important aspect of any specification, because they are involved in all questions of the kind: “What if that component fails?” or “What happens to my system if I make a change here?” V1 offers a way to visualize these dependencies by applying multiple levels of slicing. Subjects that actually understood the concept of indirect dependencies were able to answer those questions more quickly using V1 than SpecTRM, which does not provide an explicit representation of such dependencies.

For these reasons, this principle appears to be valid. It matters because exploring the dependency structure is a tricky, yet frequent and important review task. The difficulty for the visualization designer is to identify the relevant kinds of dependencies. In the case of structural dependencies, traditional graphs (boxes and arrows) appear to be a quite natural representation.

Support top down review

This principle is one of the less controversial: there is already evidence that top-down review is an effective approach to review [21]. The top-down approach is really built into the visualizations: it is impossible to access the details in V2 without doing a search in the overview (V1) first. The experiment shows that users have absolutely no problem with that approach. No user seemed to be upset by the fact that the details were not available to answer the questions without a preliminary search.

These observations confirm that top-down review is a solid, proven approach when dealing with very large amounts of information.

Support alternative strategies

Paradoxically enough, V1 and V2 are not as flexible as the strict application of this principle would suggest. This is due to the fact that the subjects of the experiment were novices. Implementing the visualizations with numerous features increases their potential, but makes them more difficult to use without extensive training. Because we did not want the tutorial part of the experiment to be longer than one hour, we actually suppressed some of the features, such as the ability to visualize several graphs at the same time, or the ability to reorder columns in the decision trees.

Some flexibility was preserved, and several questions were formulated so that they could be answered using different strategies. For instance, whenever a decision tree had to be used, the subjects had the possibility to get familiar with it by adopting the informal perspective (the questions) or the formal perspective. Interestingly, the vast majority of users was not interested in using the informal perspective, although they could have benefited from it. The reason may be that doing so would appear as a waste of time, since this approach was not at all necessary to answer the question (whereas the formal perspective was). The visualization itself can also be blamed for putting more emphasis on one perspective rather than the other.

Another instance of flexibility is the fact that the V1 “map” does not provide any preferred “entry point”: a search strategy can start with inputs, outputs, or any internal element. More often than not, several approaches are possible to answer a given question. Moreover, the visualization does not provide any absolute hierarchy between elements: in V1 there is only one abstraction level. This proved to be confusing for some users, who did not know where to start. This illustrates the fact that flexibility is more an experts’ characteristic.

Provide redundant encoding

We hoped that by providing both the visualizations and SpecTRM at the same time for one of the 3 sets of questions would enhance users' performance on these questions by giving them the opportunity to choose the most appropriate representation for their tasks. We even thought that a simultaneous and parallel use of both representations might actually be more efficient than any representation alone.

Preliminary results suggest that it is not the case with our novice subjects. The reality is that all users, when provided with both tools, picked one and discarded the other. This choice was often made *before* reading the question, and therefore was based on subjective criteria and not on an analysis of the task. It is true that some users tried to switch to the other representation after a while. But that was almost always because they were unable to answer the question with their initial choice of representation, not because they realized that the other choice would have been better, based on an analysis of the task. As a result, they performed worse than average because they wasted time.

I do not think however, that this principle is bad in itself. Again, getting familiar with any representation takes time (even if it is easier to master a visualization than an abstract mathematical language). It was not realistic to try and validate this principle with a pool of inexperienced subjects.

6.2.2. Conclusion: to visualize or not to visualize?

When this experiment is finished and its results compiled, it will be possible to answer this question with more confidence. However, some strong trends emerge. It appears that most users preferred the visualizations and were slightly more efficient using them – which are two separate things. The first observation – that visualization is more appealing – comforts us in our feeling that visualization can actively participate in making formal methods acceptable to a larger community of engineers and researchers.

It is hard to draw general conclusions from the second observation – that visualization seems to improve performance – because of the specificities of the experiment (homogeneous subject background, only one specification used, limited training, etc.). But it can be said with confidence that visualization of formal requirements specification has the potential to complement more traditional representations, and that the benefit is worth the effort. Research should continue in order to build confidence in the validity of these visualization design principles.

However, this experiment also revealed that visualization has the potential to confuse and mislead, particularly novice users. Because they are graphical by definition, they tend to impose one “vision” (usually the visualization designer’s vision), on users. This is generally good, providing the visualization designer knows the issues of using formal specifications. But it can sometimes be counterproductive if the users’ mental mechanisms are incompatible with the visualizations. This should come as no surprise: researchers have often emphasized this issue in fields where visualization has been investigated before.

Bibliography

- [1] W.R. Ashby, *An Introduction to Cybernetics*, John Wiley, 1956.
- [2] A.F. Blackwell, K.N. Whitley, J. Good, and M. Petre, "Cognitive Factors in Programming with Diagrams", accepted for publication in *Artificial Intelligence Review*.
- [3] 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.
- [4] Card, MacKinlay and Shneiderman (Eds), *Readings in Information visualization, Using Vision to Think*, Morgan Kaufmann, 2000.
- [5] S. Casner, "A Task-Analytic Approach to the Automated Design of Graphic Presentations", *ACM Trans. on Graphics*, 10:111-151, April 1991.
- [6] S. Casner and J.H. Larkin, "Cognitive Efficiency Considerations for Good Graphic Design", 11th Annual Conf. of the Cognitive Science Society, August 1989.
- [7] E. Clarke and J. Wing, "Formal Methods: State of the Art and Future Directions", *ACM Computing Surveys*, vol 28, no. 4, December 1996, pp. 626-43.
- [8] N. Dulac, T. Viguier, N.G. Leveson, and M.-A. Storey, "On the Use of Visualization in Formal Requirements Specification", *Proceedings of the IEEE Joint International Requirements Engineering Conference*, September 2002.
- [9] 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.
- [10] S. L. Gerhart, D. Craigen, and T. Ralston. "Observations on industrial practice using formal methods", *Proceedings 15th International Conference on Software Engineering (ICSE)*, Baltimore, Maryland, USA, May 1993.
- [11] T.R.G Green, "Conditional Programs Statements and their Comprehensibility to Professional Programmers", *Journal of Occupational Psychology*, 50, 93-109, 1977
- [12] T.R.G. Green and A.F. Blackwell, (1998) A tutorial on cognitive dimensions, <http://www.thomas-green.ndtilda.co.uk/workStuff/Papers/>
- [13] Edwin Hutchins, James Hollan, and Donald Norman, Direct Manipulation Interfaces, in Donald Norman and Stephen Draper, *User Centered System Design*, 1986, pp. 87-124
- [14] 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.
- [15] 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.
- [16] N.G. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley Publishing Company, U.S., 1995.
- [17] N.G. Leveson, "SpecTRM: A CAD System for Digital Automation", *Proceedings of the 17th Digital Avionics Systems Conference*, November 1998, pp. B52-1-8.
- [18] N.G. Leveson, M. Heimdahl, and J. Reese, "Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future", SIGSOFT Foundations of Software Engineering '99, Toulouse, France, September 1999.

- [19] N.G. Leveson, "Intent Specifications: An Approach to Building Human-Centered Specifications", *IEEE Trans. on Software Engineering*, 2000.
- [20] N.G. Leveson, "Completeness in Formal Specification Language Design for Process Control Systems", Formal Methods in Software Practice, Portland, 2000.
- [21] M Petre, "Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming", *Comm. of the ACM*, 38(6):33-44, June 1995.
- [22] D.S. Ranson and D.D. Woods, "Making Automation Activity Visible", Technical Report 1995-03, Ohio-State University, December 1995.
- [23] W.R. Reitman, "Heuristic design procedures, open constraints and the structure of ill-defined problems", in M.W. Shelly& G.L. Bryan (Eds), *Human Judgments and Optimality*, New York, Wiley and Sons, 1964.
- [24] 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.
- [25] 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.
- [26] P. Zave and M. Jackson, "Where do Operations Come From? A Multiparadigm Specification Technique", *IEEE Trans. on Software Engineering*, SE-22(7), July 1996.
- [27] M.K Zimmerman, M. Rodriguez, B. Ingram, M. Katahira, M. de Villepin, and N.G. Leveson, "Making Formal Methods Practical", *Proceedings of the 19th Digital Avionics Systems Conference*, October 2000.
- [28] M.K. Zimmerman, "Investigating the Readability of Formal Specification Languages", Massachusetts Institute of Technology, S.M. Thesis, 2001.
- [29] M.K. Zimmerman, K. Lundqvist, N.G. Leveson, "Investigating the Readability of Formal Requirements Specification Languages", *Proceedings of the 24th International Conference on Software Engineering*, May 2002.

Appendix A. Full-Page Figures

= Scenario 3		
<u>Operational Procedure</u> in state <i>Descent Path</i>	T	T
<u>Active Descent Approach Segment Thrust Type</u> in state <i>Idle</i>	T	*
<u>Active Descent Approach Segment Thrust Type</u> in state <i>Speed</i>	*	T
<u>Flightphase</u> in state <i>Descent</i>	T	T
<u>Active Descent Approach Speed Type</u> in state <i>Mach</i>	T	T
<u>ADC Mach</u> > beta + 0.015	T	T
= Scenario 4		
<u>Operational Procedure</u> in state <i>Descent Path</i>	T	
<u>Active Descent Approach Segment Thrust Type</u> in state <i>Decel</i>	T	
<u>Flightphase</u> in state <i>Descent</i>	T	
<u>Descent Approach Path Speed Types[1]</u> in state <i>CAS</i>	T	
<u>ADC CAS</u> > gamma + 5 kts	T	
<u>IRS Flight Path Acceleration</u> > -0.01 g	T	
= Scenario 5		
<u>Operational Procedure</u> in state <i>Descent Path</i>	T	
<u>Active Descent Approach Segment Thrust Type</u> in state <i>Decel</i>	T	
<u>Flightphase</u> in state <i>Descent</i>	T	
<u>Descent Approach Path Speed Types[1]</u> in state <i>Mach</i>	T	
<u>ADC Mach</u> > delta + 0.015	T	
<u>IRS Flight Path Acceleration</u> > -0.01 g	T	

Figure A: Three of the 6 AND/OR tables necessary to describe the state variable “Active Add Drag Scenario” in the specification of the MD-11 Vertical Guidance Annunciation Process..

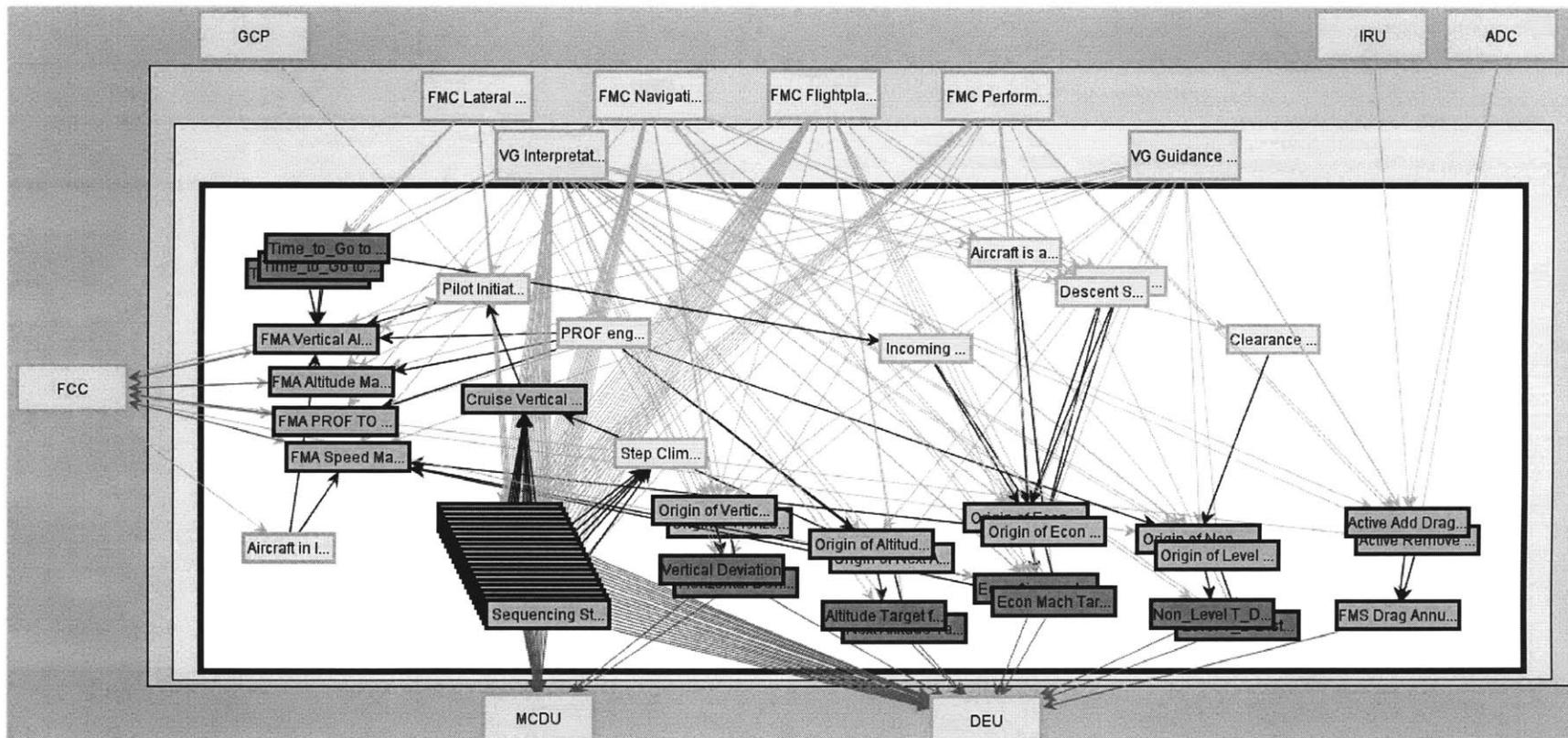


Figure B: Structural overview (V1) of the specification of the MD-11 Vertical Guidance Annunciation Process.

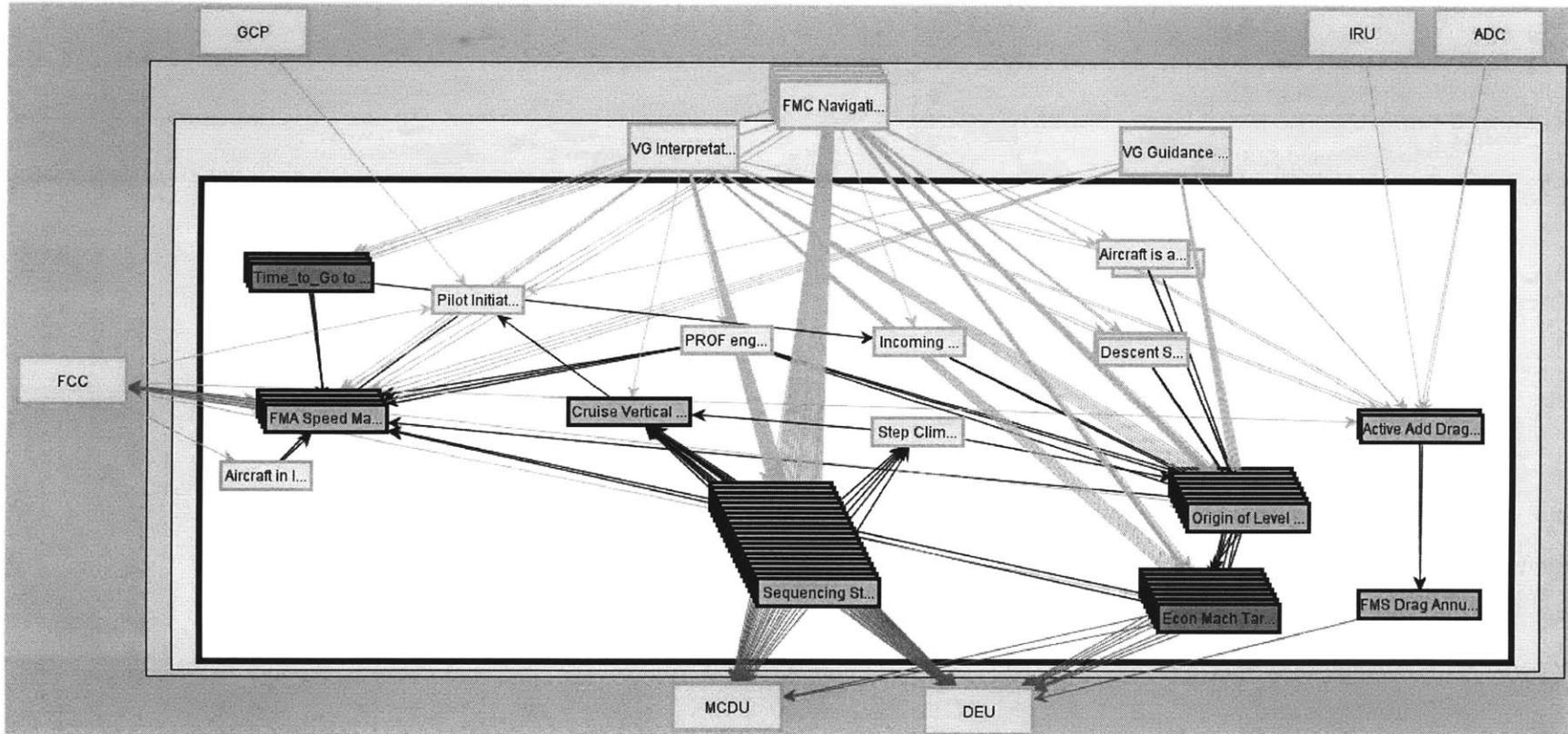


Figure C: A reorganized view of V1. Some elements that share common properties are grouped together in “cascades”.

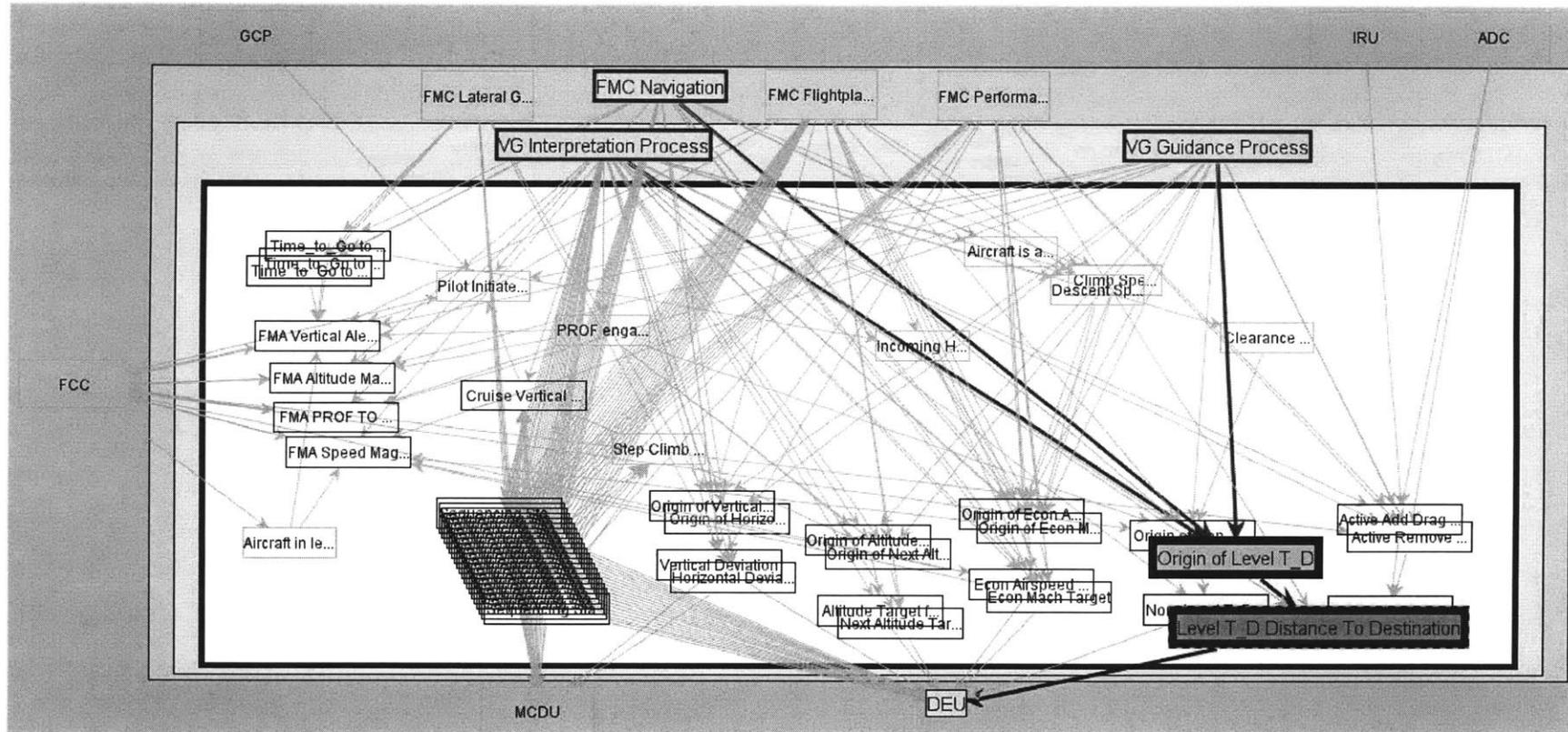


Figure D: Sliced structural overview. The state variable called *Origin of Level T_D* and all its structural dependencies are emphasized over the rest of the model, which is preserved as context.

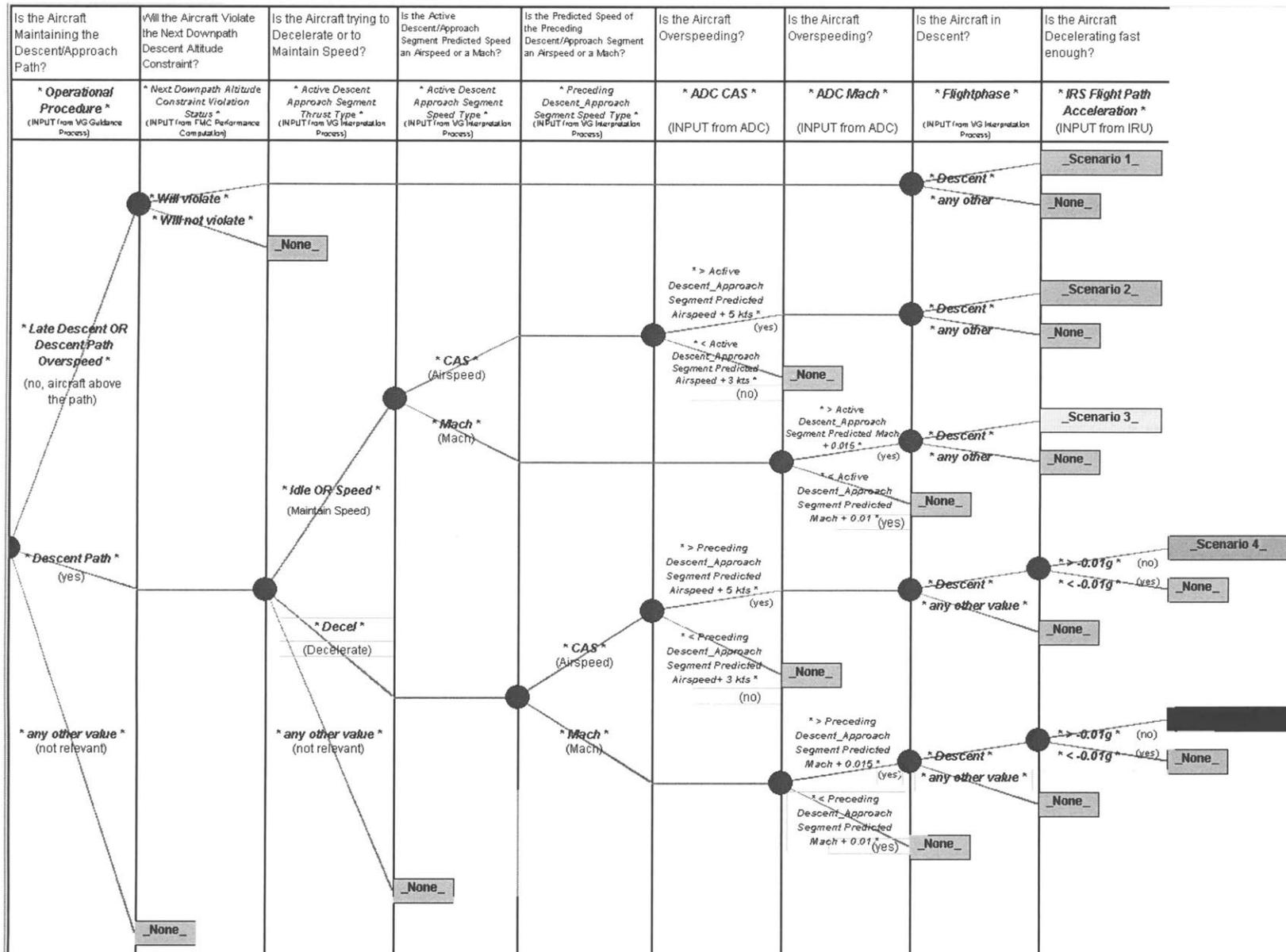


Figure E: An example of Questions-based Decision Tree taken from the MD-11 Vertical Guidance Specification. This state variable (*Active Add Drag Scenario*) indicates which one of the five possible scenarios for extending airbrakes is active.

SpecTRM

File Edit Document Model Dictionary Window Help

AC:Documents and Settings\mguier\Desktop\FMS_experiment_05_march.xpz

New Project

- MD-11 FMS Experiment
 - Level 0: Program Management Information
 - Level 1: System-Level Goals, Requirements, a
 - Level 2: System Design Principles
 - Level 3: Blackbox Behavior
 - Behavioral Assumptions and Models of the
 - Communication
 - Operational Procedures
 - User Model
 - System Blackbox Behavior
 - MD-11 FMS VG Annunciation Process
 - State Variables
 - Macros
 - Devices
 - Inputs
 - Outputs
 - Special Functions used as Real Va
 - Verification and Validation
 - Level 4: Physical and Logical Design Represe
 - Level 5: Physical Implementation
 - Level 6: System Operations
 - Constant Definitions
 - Index
 - Glossary

FMA Vertical Alert Discrete

State Variable

Obsolete use:

Exception Handling:

Related Inputs:

Description: Utilized by the FCC to determine when the VERT ALERT time stage shall flash in the FMA Altitude Window, above the altitude target. VERT ALERT shall flash for automatic altitude changes in PROF, 5 seconds before initiating the maneuver. This includes all scenarios where the aircraft is in level flight due to an altitude constraint or due to the start of a step climb or the top of descent ahead, even though the aircraft has been cleared to a higher or lower altitude. VERT ALERT shall also flash for pilot initiated altitude changes in PROF.

Comments: *VERT ALERT is never displayed if the vertical profile is not engaged.*

References: Aircraft In Level Flight, PROF engaged, Pilot Initiates an Altitude Change In PROF, Time to Go To Step Climb Start Point, Time to Go To Top Of Descent Point, Time to Go To Active Lateral Leg Termination, FMS Altitude Target, Vertical Flightplan Altitude Constraint Existence, FCC Selected Altitude, Pilot Initiates an Altitude Change, Aircraft In Level Flight, PROF engaged, Vertical Flightplan Altitude Constraint Existence, FCC Selected Altitude, FMS Altitude Target

Appears In: FCC Outputs

DEFINITION

= Unknown

Power Up

= Display

PROF engaged	T	T	T	T	T
Aircraft In Level Flight	T	T	T	T	T
FCC Selected Altitude > FMS Altitude Target + 50	T	T	F		
FCC Selected Altitude < FMS Altitude Target - 50			F	T	
Time to Go To Step Climb Start Point() < 5	T	F			
Time to Go To Active Lateral Leg Termination() < 5	T				T
Time to Go To Top Of Descent Point() < 5				T	F
Vertical Flightplan Altitude Constraint Existence = Exists	T				T
Pilot Initiates an Altitude Change In PROF			T		

= Hide

PROF engaged	T	T	T	T	T	F
Aircraft In Level Flight	T	T	T	T	T	F
FCC Selected Altitude > FMS Altitude Target + 50	T	T	F			
FCC Selected Altitude < FMS Altitude Target - 50			F	T		
Time to Go To Step Climb Start Point() > 5	T	T				
Time to Go To Active Lateral Leg Termination() > 5	F	T		F	T	
Time to Go To Top Of Descent Point() > 5				T	T	
Vertical Flightplan Altitude Constraint Existence = Does Not Exist	T			T		
Pilot Initiates an Altitude Change In PROF			F			

Figure H: Screen capture of the SpecTRM GUI.

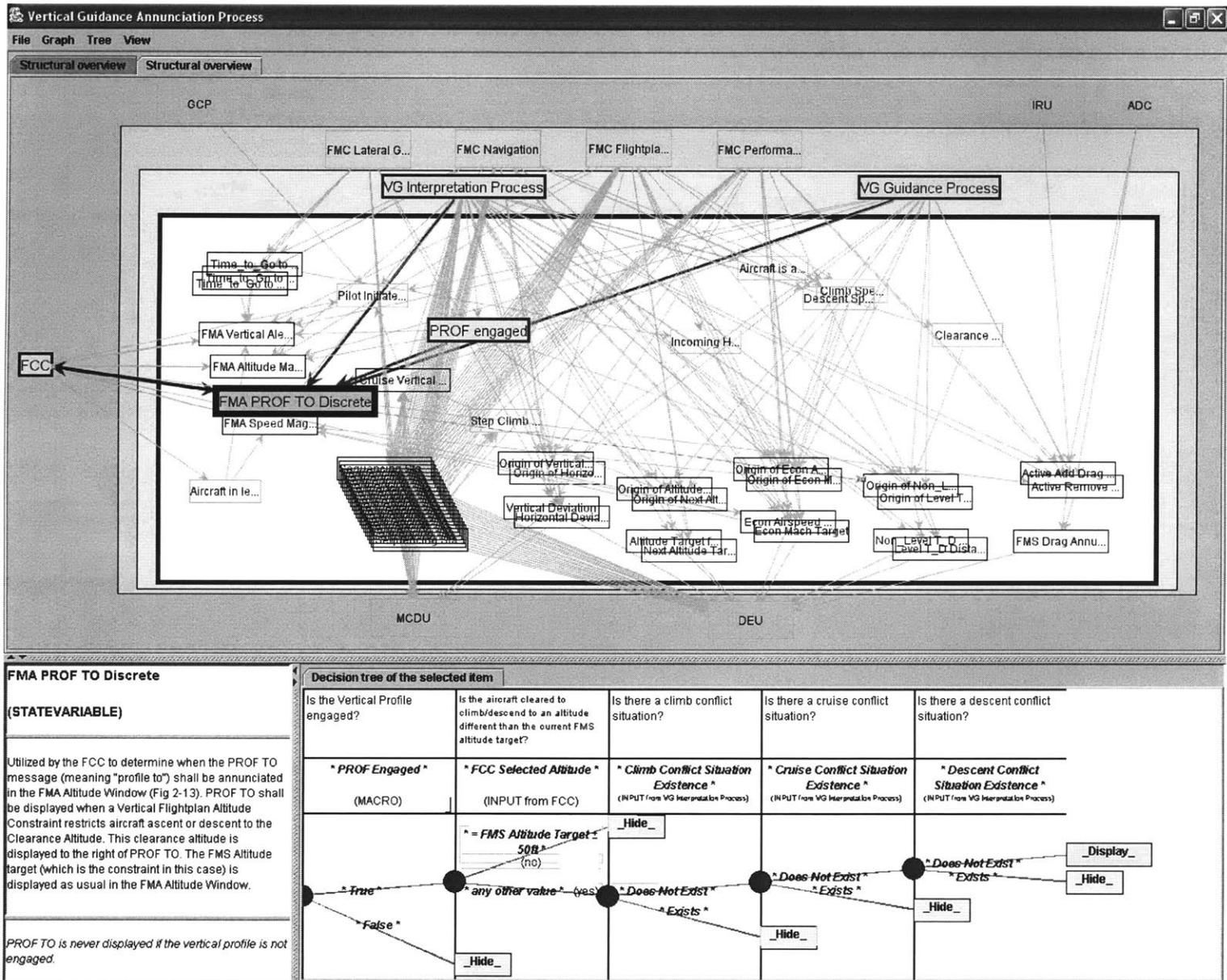


Figure I: Screen capture of the visualization tool.

10.4.5.1.1 FMS Add Drag Mode Scenario 1 Late Descent With Altitude Constraint Violation

FMS Add Drag Mode Scenario 1 shall be activated when all the following conditions are valid:

- a. Flightphase is *Descent*.

COMMENT: This condition inhibits the Add Drag message during Crz Accels, Crz Decels and in Approach.

- b. Vertical Guidance Operational Procedure is *Late Descent* or *Descent Path Overspeed*.

- c. Performance Forward Predictions indicate that the aircraft, with clean configuration, shall violate the next downpath Descent Altitude Constraint.

COMMENT: The Aircraft is above the Descent /Approach Path, and according to Performance Predictions, shall violate the next downpath Descent Altitude Constraint.

FMS Add Drag Mode Scenario 1 shall no longer be active when any of the following conditions is valid:

- a. Flightphase is not *Descent*.

COMMENT: This condition inhibits the Add Drag message during Crz Accels, Crz Decels and in Approach.

- b. Vertical Guidance Operational Procedure is not *Late Descent* or *Descent Path Overspeed*.

- c. Performance Forward Predictions indicate that the aircraft, with clean configuration, shall not violate the next downpath Descent Altitude Constraint.

10.4.5.1.2 FMS Add Drag Mode Scenario 2 Descent Path Speed Control on Constant Speed CAS Segment

FMS Add Drag Mode Scenario 2 shall be activated when all the following conditions are valid:

- a. Flightphase is *Descent*.

COMMENT: This condition inhibits the Add Drag message during Crz Accels, Crz Decels and in Approach.

- b. Vertical Guidance Operational Procedure is *Descent Path*.

- c. Active Distance-Referenced Descent/Approach Path Segment has an associated Speed Type of *CAS*.

- d. Active Distance-Referenced Descent/Approach Path Segment has an associated Thrust Type of *Idle* or *Speed*.

- e. ADC CAS > Active Distance-Referenced Descent/Approach Path Segment Predicted Airspeed + 5 knots.

COMMENT:The aircraft is maintaining the Descent/Approach Path and is overspeeding by 5 knots.

Figure J: An extract of a possible textual representation of a formal specification. It is adapted from the actual specification of the MD-11 Vertical Guidance System. The state variable is "Active Add Drag Scenario".

Appendix B. Visualizing State Transition Conditions

The design of V2 (the transition tree) was the result of a more extensive study than the other two visualizations. This study aimed at finding a good way to represent state transition conditions, which arguably constitute the part of a state-based specification that is hardest to understand and review. This study is summarized in this appendix.

B.1. Issues in representing transition conditions

Whatever the representation, the nature of the problem itself (visualizing transition conditions) raises a number of issues that have to be addressed by the chosen visualization. Here is a list of the main ones:

The use of formal vocabulary

Formal models make an extensive use of formal, precise vocabulary, which abound in the transition conditions. The choice of words is always a trade-off between conciseness, consistency and meaning. Because variable names have to be displayed in whatever space is available on the screen, this choice must be taken into account when designing the visualization. Here are some examples of variable names from our case study: ‘Operational Procedure’, ‘Next Downpath Altitude Constraint Violation Status’, ‘ADC CAS’. These names show that in our model, meaning was preferred over conciseness. This is not without consequences on the design of V2.

Metadata

Representing state transition conditions implies organizing information, hence the use of a meta-language (whether implicit or not) to categorize elements.

Conciseness

Conciseness of the representation is generally important but should not necessarily be regarded as a primordial quality. Some representations may achieve conciseness at the expense of clarity for instance. In transition conditions, relationships between conditions are important, and should not be left invisible just for conciseness sake.

Scaling

The amount of data to be represented varies according to several factors external to particular representations. Within a given model, one should expect a scaling factor of the order of ten. Good visualizations should therefore be adapted to the average amount of displayed

information, and should also be tolerant regarding variations of scale. (Note: the specification writer usually has some latitude regarding the complexity of state variables, which can be used to reduce scale variability – for instance, it is often possible to break down a very complex variable into smaller chunks).

Information priority order

Visualizing information implies giving a priority order to metadata. This order influences the way users are going to solve tasks using the visualization. Here is an example:

In the transition conditions of a state variable, there are two main categories of data (or metadata): the possible state values and the different parameters whose values or states decide of the transition. Giving the priority to the former can reduce the amount of data being looked at (since the user focuses on one state value only); on the other hand, if the latter comes first, cognitive workload is reduced but the data space can be larger. Concrete examples will be given in the following section.

Abstraction management

Using abstractions is a way to deal with scale; it is also a user strategy in the process of understanding complex transition conditions. As an example, let us consider the use of macros in SpecTRM formal models. Macros are internal Boolean variables that depend on the state variables of the model and on external parameters such as inputs. Macros are used within state variables to group several parameters into one, whenever it is logical (and possible) to do so. In that sense macros are a kind of abstraction. A good visualization of transition conditions should be able to represent them in both their abstract and “expanded” form. At the beginning, novice users may want to consider the macros in their abstract form only; when they acquire enough expertise, they may want to use more complex, but more straightforward expanded form.

Slicing ability

Heimdahl and Whalen’s behavioral specification slicing [14] is a way to simplify a given model by applying a scenario, which consists in setting the value or state of the relevant inputs of the model. In the fictitious scenario “Initial climb” for instance, ‘thrust’ would be *100%*, ‘Flightphase’ would be *climb*, ‘Airbrakes’ would be *retracted*, and so forth. The narrower the scenario, the more limited the possible behaviors of the whole model are. Applying a scenario produces a “slice” of the model that is still consistent, but smaller and easier to manage. Slicing has consequences on the structure of the model itself, but the main simplifications occur at the state variables level: transition conditions are made simpler. Slicing is considered to be an important feature in any visualization of transition conditions.

One of the main issues with slicing is to maintain the sense of context while working on the sliced specification. Visualizations can address this by applying focus+context principles, such as a de-emphasis of all the elements that do not belong to the slice.

B.2. Comparative study of 3 representations

The following is a comparative study of three possible representations of state transitions conditions. The first one is a typical formal textual version of the data. The second one (AND/OR tables) is not a new concept either; it is used in particular in SpecTRM. I designed the third representation (a version of decision trees) as an attempt to address the weaknesses of the first two, with all the issues mentioned above in mind.

All representations are supposed to be a kind of visualization in that they are dynamically displayed on a computer screen, as opposed to a static paper-based display. Such visualizations usually imply interactions, from navigation to data manipulation. The three representations presented here are based on the same underlying formal model and therefore are strictly equivalent content-wise.

The following comments are based on a small user study with a handful of graduate students. Subjects were asked to answer a set of questions designed to test their comprehension of the transition conditions of a few state variables, all drawn from a formal specification of the MD-11 Vertical Guidance System. For each question, the subjects were provided with one of the 3 representations.

Textual representation

The most basic representation possible is the textual form. (Note: This should not be mistaken with an informal model, where static text is the natural representation, alongside with supporting graphics). Figure J, p.64, gives an example of what it can look like in the case of a formal model.

PROS: It may not be obvious at first sight, but there actually is a very logical structure in this kind of representation. Text easily supports the use of hierarchy and organization through the use of headings, indentation, fonts, etc. In addition, it is easy (although not space-efficient) to add informal comments wherever necessary to help reviewers and novice users. Finally, one of the main advantages is that no tool other than a word editor is needed.

CONS: First, a purely textual representation mixes formal and informal vocabulary in a dangerous way, all the more so as variable names themselves usually make use of seemingly informal words. Second, this representation is the least concise of all, does not scale well to large

data, and is unable to give any kind of overview as soon as the text is longer than one page (which is almost always the case). Finally, it does not support abstraction or slicing well, since change patterns are not shown (the only noticeable change is the length of the text).

AND/OR tables

AND/OR tables take advantage of the “bi-dimensional” logical structure of transition conditions to reach a very high level of conciseness. The horizontal dimension conveys union (the transition occurs if that given set of conditions is true OR if that other set is true...), whereas the vertical dimension conveys intersection (that given set of condition is defined by this condition AND that one AND...). Examples are given in Figure A, p.55.

PROS: First, tables are concise, essentially because they elide all the text around the formal elements (including the helpful comments); because they get rid of hierarchy or order among the conditions; and because the logical AND/OR structure is perfectly carried by the representation. Second, the fact they are compact make them a good candidate for abstraction and slicing. In the latter case for instance, elided conditions could be greyed out in a way such that the new representation is clearer while the overall picture remains.

CONS: Novice users may not feel comfortable with this representation, because there is no room in the tables for informal comments. In addition, conciseness is impaired if the number of possible states is high (every state requires an independent table; if several states have similar transition conditions, there will be redundancy).

In addition, focus is irrevocably given to state values as opposed to the conditions themselves. This can be good for certain tasks (like understanding the conditions for a particular, isolated transition) but hides important patterns that are only seen when all transitions are seen together.

Decision tree

As opposed to the tables, decision trees focus on the conditions themselves rather than the possible states. Conditions are showed as decisions made sequentially. At a given level of the tree, the number of new branches is equal to the number of possible values or states of the corresponding parameter. See section 3.2 for more details on how decision trees work. One feature in particular is new: the questions at the top of each column in the tree. Answering one of these questions is taking a decision; it is choosing a branch of the tree among all the possible paths available at this level. These questions play the role of an instructor in charge of teaching a

novice the meaning of all the formal vocabulary. As users become familiar with the information in the model, they will refer to the question less and less.

PROS: Firstly, this visualization is quite concise. The use of space is efficient: the whole set of conditions for all states of one given state variable can usually be displayed on one screen while keeping good readability, giving a good understanding of the conditions “as a whole”. In addition, the fact the focus is set on the sequence of conditions (the state value being only the end product, once all decisions are made), gives room for useful informal textual elements intended to explain the role of each condition separately. These additions can take the form of questions as explained above, or simply comments. It is helpful because the most difficult tasks when reviewing transition conditions is not to understand the meaning of the states, but rather the role of the different parameters (that is to say, the meaning of the conditions).

This visualization also reveals patterns that were difficult to see in the text or in the tables. As an example, consider columns 6 and 7 in Figure E, p.59. They play a very similar role, as can be seen visually. After a few seconds of attention you will realize they in fact depend on the decisions taken either in columns 4 or 5, which are also very similar. And the choice between 4 and 5 depends in fact on the decision made in column 3! Now try and find this pattern in Figure J...

Also, this visualization can make a more extensive use of visual hints and helps than others, which can alleviate the cognitive load and extend the manageable amount of data. Examples are the use of font consistency, of spatial location of branches and leaves in the tree, or of a different color for each different state.

Lastly, this representation easily supports abstractions and slicing, using space (compression) and/or brightness (greying out) to emphasize sliced data while keeping context (see Figure G, p.61).

CONS: Since the focus is on conditions, the state values are de-emphasized. As a consequence, trying to understand the transition conditions of a particular state is tricky, especially if there are many paths that lead to this state (i.e., many ways to transition to this state). As an example, consider the state ‘None’ in Figure E. Another weakness is that this representation really gives the impression of an intended order of the conditions, when sometimes there should be none. (Note: this is true also of the other two representations, but the impression is not as strong because the focus is not on the conditions). Note also that order is sometimes actually important – remember columns 3 to 7 in Figure E... Last drawback: scaling remains an

issue. For particularly simple state variables, the tree looks ridiculously small, whereas for the largest ones, zooming or scrolling becomes necessary, at the expense of the feeling of overview.

In summary

The following table summarizes the comparative study. It appears that the decision tree is better at several levels. However, it is impossible to make serious conclusions before the user study. One thing appears to be important, though; depending on the task, the most adequate representation will vary.

	Use of formal vocabulary	Metadata	Conciseness	Scaling	Information ordering	Abstraction management	Slicing ability
Text	★	★★	★	★	★★	★	★
AND/OR tables	★★★	★★	★★★★	★★	★★	★★	★★★
Decision tree	★★★	★★	★★★★	★★	★★★★	★★★★	★★★★

Appendix C. Experiment questions

The following pages are a copy of the forms that were given to the subjects, in their original format.

