Investigating the Readability of State-Based Formal Requirements Specification Languages

Marc K. Zimmerman, Kristina Lundqvist, Nancy Leveson
Massachusetts Institute of Technology
Cambridge, MA 02139
+1 617 258 0505
leveson@mit.edu

ABSTRACT

The readability of formal requirements specification languages is hypothesized as a limiting factor in the acceptance of formal methods by the industrial community. An empirical study was conducted to determine how various factors of state-based requirements specification language design affect readability using aerospace applications. Six factors were tested in all, including the representation of the overall state machine structure, the expression of triggering conditions, the use of macros, the use of internal broadcast events, the use of hierarchies, and transition perspective (going-to or coming-from). Subjects included computer scientists as well as aerospace engineers in an effort to determine whether background affects notational preferences. Because so little previous experimentation on this topic exists on which to build hypotheses, the study was designed as a preliminary exploration of what factors are most important with respect to readability. It can serve as a starting point for more thorough and carefully controlled experimentation in specification language readability.

1. INTRODUCTION

Formal requirements specifications and formal analysis theoretically present a way out of the dilemma posed by our inability to test even a small part of the enormous state space involved in most digital systems. The past 30 years have advanced the state of knowledge about formal methods to the point where many important problems can be solved. While formal methods are being applied to hardware in industry, the results of formal methods research for software has only rarely reached beyond the research lab and been used in industrial practice for day-to-day software development.

Several reasons may be hypothesized for this lack of widespread adoption. First, most formal languages are based on discrete mathematics and logic. However, engineers are typically not trained in these fields of mathematics. Furthermore, the notations used in these languages are often not as concise or as parsimonious as their continuous math counterparts. So while a control law can be represented as a differential equation, the discrete mode logic for a flight management system might require hundreds of pages of formal logic to specify. The review of such specifications by domain experts is a daunting task. The scope and scalability of formal methods are additional concerns.

In our experience, one of the biggest stumbling blocks to the use of formal specification languages in industry relates to readability. Readability is arguably one of the most important properties of any specification. Requirements specifications in industrial projects must be readable by a large variety of people with diverse backgrounds and expertise including system designers and developers, customers, users, certifiers, etc. Having a common model that is readable by a general audience will enhance communication among all involved parties, which is widely recognized as the source of the most important outstanding problems in industrial practice.

In addition, our experience in analyzing formal specifications for complex systems suggests that the most significant errors and omissions will be found by human experts rather than automated tools [5]. This observation does not mean that automated tools are not useful and important in finding some types of errors, particularly those involving mathematical properties and those requiring tedious checks. But humans are required to determine whether a specification conforms with engineering expectations and requirements (e.g., whether all necessary conditions have been included under which an aircraft's elevator must be moved to maintain aerodynamic stability). Furthermore, specification flaws found by formal analysis tools will need to be evaluated by human experts. Therefore, readability of requirements specifications is necessary not only for human review of complex models but also for human processing of analysis results. Our experience in working with engineers is that they do not accept and put their confidence in analysis results from a model that they cannot personally validate matches the system they intended to design.

Readability may also lead to reduced learning time. Certainly any specification language is going to require some training in order to understand and use it. However, particularly with respect to review, this time cannot be too long before it becomes impractical to coordinate the considerable amount of reviewing that leads to high-quality specifications and software. It can require as much as 3-6 months training before an engineer can use some formal languages effectively. This amount of required training not only reduces the number of people who can participate in reviewing

the formal specification, but it also puts constraints on the addition of engineers to the specification and design teams[1].

In an effort to increase the practicality of formal methods, this work deals specifically with the problem of readability. Designing a requirements specification language that is readable by a general audience is a difficult problem and little empirical or experimental evidence exists to guide those designing formal specification languages to support readability. This paper reports on preliminary results of such an investigation to help us understand the notational features that are most conducive to readability. We specifically included subjects with both computer science and engineering backgrounds, as these are two fields that can directly benefit from the use of formal methods. Understanding which notations are more readable, and to which audiences, could help us create requirements specification languages that are more expressive and effective, allowing formal methods to become a more attractive alternative for the industrial community.

The next section describes related research and the problems in designing useful experiments on this topic. We then describe the experimental design and the results we obtained.

2. BACKGROUND

Formal requirements specifications all have an underlying mathematical model that can take many different forms. In this work, we will focus specifically on specification languages that use an underlying state-machine model; we have found through 20 years of empirical work that such models are the most easily understood and adopted by engineers working on control systems (our area of interest), and therefore they seem like a reasonable place to start. Our goal is to determine those features of state-based requirements specification languages that can increase the readability and comprehensibility for an appropriate audience.

The experimental design in this paper draws heavily on previous experimentation, particularly with respect to measuring readability. However, while there has been a significant amount of research in experimental methodology within a computer science context (for example, see [2]) and some experimentation on programming language design (summarized in [3]), our specific goal of determining which factors affect the readability of state-based specification languages has not been addressed with one exception. Previous research sought to determine factors that affect the readability of Z specifications, but the number of factors considered (two) is much smaller in scope than those considered here [2]. In addition, previous work with Z focuses on one specification language, rather than a class of specification languages as is considered in this paper.

Because of the lack of previous experimentation on which to build, our goal became to build an experimental design that allowed us to examine a large number of features and to identify which ones appear to play an important enough role in readability. Identification of these features will warrant follow up and more carefully controlled and focused experimentation.

[1] While it also takes time to learn specification languages based on continuous mathematics, engineers do not need to be taught the mathematical foundations and only need to learn the syntax of the language.

Our first problem was in determining which language design features should be investigated. We started by surveying several state-based specification languages that have been used on real aerospace systems to determine the distinguishing features of each. The languages chosen were Statecharts [4], SCR [6], RSML [5], SpecTRM-RL [7], and OpProc Tables [8]. Because each is based on the same underlying state-machine model, what distinguishes each of these languages are the ways each describes the various parts of that model. After selecting these languages, we then established some hypotheses about which of the distinguishing features might affect readability[2].

State machines are an abstract model whose overall structure can be represented using a graphical, textual, or tabular format. Similarly, the conditions that trigger state transitions can be expressed in a variety of ways, e.g., using propositional logic, graphics, tables, etc., as can the internal events that are used to order or synchronize the transitions. Each method of representing the structure and components of a state machine offers different potential benefits. What differentiates one state-based specification language from another is simply the choices for these representation formats.

2.1 Overall State Machine Structure

The structure of the state machine may be depicted in a variety of ways including graphically, in a tabular form, textually, or using a mixture of these. OpProc Tables and SCR represent the structure of the underlying state machine in a table, whereas Statecharts and RSML represent the model graphically, using circles or boxes and arrows. SpecTRM-RL shows the states graphically but the transitions (arrows) are defined separately in order to simplify the graphical view. We hypothesize that the way the state machine structure is represented has an effect on readability.

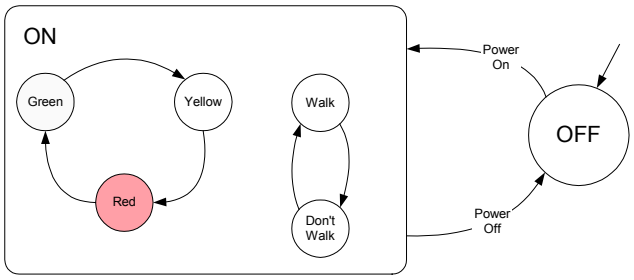

Figure 1. State machine model using a superstate.

A second aspect of representing structure is the use of hierarchies. Due to the inherent complexity of modern software systems, most of the newer state-based requirements specification languages use superstates, or hierarchies, to provide logical modularizations in the model. For example, the states and transitions of a traffic light state machine can be grouped together to form the superstate "On," as shown in Figure 1. In the example, if the event "Power-Off" occurs while in the state On, the state machine will transition

[2] Although two of the languages chosen were designed by members of our research group, we want to stress that the goal was not to show that our languages were better than the others but to assist in establishing future research directions for our group and others with respect to the relationship between readability and improved specification language designs of the future.

to Off and vice versa. This example shows a two-level hierarchy. At the highest level, there is the On-Off state machine. However, within the On state, there is another modular state machine description of the traffic light, which is active only when the system is in the On state. In the specification languages surveyed, all except SCR employ hierarchies.

The use of hierarchies simplifies specifications and removes the need to explicitly specify some transitions. For this reason, it could be argued that allowing hierarchical specifications is essential if formal methods are to be scalable. It might also help the reader to develop a better mental model of the system's behavior than would otherwise be achieved in a flat state machine. However, by not explicitly specifying all the transitions of a state machine, superstates can also lead to confusion regarding execution of the state machine. We wanted to determine if this confusion was indeed significant.

2.2 Internal Events

Statecharts, SCR, and RSML rely on internal broadcast events to order execution of the state machine. However, our experience in using RSML in the specification of TCAS II [8] was that internal broadcast events were the single source of most of the errors found in the specification and, in addition, caused great difficulty for the reviewers in reading the specification, particularly in consistent interpretation of the semantics of state changes among reviewers. As a result of these negative experiences, internal broadcast events were not included in SpecTRM-RL. Instead, ordering of transitions is based on explicitly specified data dependencies. In this experiment, we hoped to provide empirical evidence to test our anecdotal experience about the readability of languages using internal events to order state changes.

2.3 Transitions

The third general issue investigated was the specification of transitions. Several different design features appear to be related to readability. The first of these is perspective. When specifying a state machine, transitions can be organized in one of two ways: (1) by source state, where all the transitions *out of* a certain state are grouped together ("If I am in state X, where can I transition to from here?") or (2) by destination state, where all the transition *to* a certain state are grouped together. The first might be referred to as the going-to perspective while the second as a coming-from perspective. While both express equivalent information, we were interested in determining whether one of these perspectives was a more intuitive way for reviewers to think about state machine behavior. Completely graphical representations like Statecharts provide both perspectives, which may or may not be an ideal property. Others, like SpecTRM-RL and OpProc tables, use a coming from perspective. RSML and SCR do not restrict the designer to either; in fact, there is no enforced organization of the transitions.

The second distinguishing feature of transitions is the use of macros. Macros in a state-based specification language function basically as they do in any programming language. They allow the modularizing of a piece of logic that can then be referred to solely by name in the specification. This modularization allows the user to specify logic in smaller blocks and to use appropriate naming conventions, both of which potentially enhance readability. Previous experimentation using SpecTRM-RL on a very large industrial application prompted us to conclude that macros are a necessity if formal requirements specification languages are ever to scale to realistic systems [5].

However, macros also possess a drawback in that they may require the reader to navigate through several parts of the specification to identify specific logic conditions that may affect the state machine. The use of nested macros also may confuse the reader when trying to understand how the system behaves. This problem is not unique to specification languages, of course, but can arise in programming languages as well. Previous experiments with Z concluded that the effectiveness of macros is related to the number and type of modularizations used [2]. For example, using a few, non-nested macros can aid comprehension, whereas using several nested macros may not. Whether or not these conclusions apply to state-machine specifications is something we decided to investigate in this work. All the state-based specification languages we surveyed employ some form of macros (although they are called terms in SCR), with the exception of Statecharts.

The final feature selected was the format of the triggering conditions associated with a transition. SCR and Statecharts use propositional logic to specify triggers. This representation is relatively concise, but may be difficult to read when used to express complex conditions. Both RSML and SpecTRM-RL use a tabular logic table notation (called AND/OR tables) to express conditions. The designers of these two latter languages claim that engineers found using AND/OR tables to be not only readable, but also easy to learn. OpProc tables also use a tabular logic table notation to represent conditions.

In this experiment, we investigated not only the use of propositional logic and tables to express conditions, but also textual and graphical representations. Textual descriptions are used in most industrial specifications to date, so they served as a good baseline for comparison in this experiment. We also included standard engineering logic gates as a means of graphically specifying logical conditions to determine if these might be easier for engineers to read.

With these features selected, we designed an experiment to determine how each affected a specification's readability.

3. GENERAL EXPERIMENTAL DESIGN

The experiment itself consisted of six (6) parts, one part for each feature tested: state-machine representation, conditions, macros, events, hierarchies, and perspective. Each part of the experiment was run the same. Subjects were presented with two to four equivalent specifications, depending on the feature being tested. They were then asked a series of objective questions about the state machine behavior described by the specifications.

Initially, subjects were given access to any or all of the specifications when answering the first few questions. They were instructed to indicate which specification(s) they used to answer each. We were interested to see whether subjects had any intuitive biases about which notation or specification they would find to be the most effective or readable.

After this preliminary section, subjects were restricted to a particular specification when answering questions until every specification was tested. For example, in the macros experiment, subjects were asked two questions for which they were given access to both the Macro and Flat specifications, then four questions for which they were restricted to the Macro

specification, then four more questions for which they were restricted to the Flat specification. Following the objective questions, subjects were asked to give a subjective evaluation of each specification used in that particular part of the experiment. We were as interested in subjective evaluations as objective measurements, as discussed below.

Subjects were all graduate students in either aeronautics or computer science. Prior to each experiment, the subject's background and familiarity with both fields were ascertained. As mentioned before, there may perhaps be a correlation between a subject's academic background and notation preferences. Twelve subjects were tested in total, including six computer scientists and six engineers. Statistically, we would like to have had a larger subject base, but as this is a preliminary study, we believe that the number of subjects was adequate. The number was limited by the amount of time it took to give and to complete the survey instrument, which was, in turn affected by the necessarily large scope of this preliminary study. Our primary goal was to use the large scope as a means of identifying the most important factors on which further experiments could be focused.

As far as training, subjects were given an introduction to state machines the week before they participated. The training document familiarized them with basic state-machine terminology such as "states," "triggers," and "transitions." This introduction provided subjects who had no prior experience with state machines enough information to answer the questions. Furthermore, a practitioner was present throughout each experiment to explain the directions for each part of the experiment and to answer questions. From the few questions asked, it appeared the training was adequate. Also, previous experience with state machines was not determined to be a significant factor in the results.

3.1 Question Design

To measure readability, we were looking for indications that the subject was able to read and understand material in the specifications. Several approaches to measuring readability have been proposed in previous work, some relying on specific technical questions, others on general questions about a system's functionality. In this work, we followed the approach taken by Finney et al. [2] and Brooks [1], which is outlined below.

As no single measurement can capture all aspects of readability, we considered both objective and subjective evaluations. Objective questions were designed to test for four different aspects of readability. In order of increasing difficulty, these are:

1. Finding a relevant part of the specification, e.g., "Where in the specification is the trigger specified for a transition from the Cruise to Descent state?"
2. Understanding the notation, e.g., "What does line 6 of the specification say?"
3. Relating the specification to the model, e.g., "What will the output of the system be if the Altitude input is 1000 ft.?"
4. Modifying the specification, e.g., "What changes need to be made to the specification if the transition 'Reorient mode to Spinup Mode when condition C occurs' is added to the state machine?"

The number of questions of each type that were asked was relatively balanced. We did not consider one type of question to be more important than another.

A lot of time was devoted to designing the objective questions for the various parts of the experiment. Our biggest concern was that the questions asked involve realistic tasks, i.e. tasks that would be encountered during the review and modification of real system specifications. For example, answering the question "How many binary state machines are described in the specification?" certainly involves reading and understanding the notation used in the specification, but the task is not a realistic one. We were also concerned that asking too many questions could result in the subject getting tired or bored and impact the results.

A third factor was the potential diversity in our subjects' backgrounds—those that had never seen a state machine before might require more time than estimated. This concern was validated during pretesting, and an agonizing process of eliminating questions was required. In the end, subjects were asked between four and five questions about each specification, which allowed us to test the desired aspects of readability, without requiring an inordinate amount of time for most subjects to complete.

Another issue was consistency throughout the various parts of the experiment. For example, when subjects are given three different specifications testing the readability of a certain feature, we wanted the types of questions asked about the three specifications to be as similar as possible in terms of format and skills involved to answer. Obviously, if more difficult questions were asked about one specification, that would not only affect objective performance in the experiment, but would likely affect subjective evaluation of the specification's readability as well.

No time measurements were made during the experiment. There are several (often overlooked) difficulties and inadequacies involved with using time as a recorded variable. Furthermore, we did not want stress to play any part in a subject's performance. We wanted subjects to be able to work through every question in the experiment, so that their subjective evaluations would be as complete as possible (subjective evaluations will be discussed shortly). If subjects were given a time limit, they may not have been able to attempt every question. In addition, they may have made errors in responding to questions that they would not have otherwise made. Each part of the experiment contained an average of 12 objective questions and was designed to take between 20-25 minutes, bringing the total length of an experiment with 6 parts to roughly 2.5 hours. Subjects were also encouraged (but not forced) to skip questions that required more than 90 seconds to answer. The skipping of difficult questions provides useful information as well. For example, taking longer than 90 seconds to answer a certain type of question using a certain specification is an indicator of limited readability.

After each part of the experiment, subjects were asked a set of subjective questions regarding their experience using the specifications. In some respects, this measure is more important than objective performance. Readability is a complex property, which is difficult if not impossible to measure objectively. Subjects were asked to rank the specifications used for each part of the experiment in terms of readability and then in terms of ease of editing. They were also asked to identify advantages and disadvantages they found in using each specification. Subjective

responses were given verbally, which we feel is important for two reasons. First, subjects tend to be more expressive when communicating orally, rather than in writing. We hoped that this would lead to more insightful responses. Second, asking the experiment practitioner to record subjective responses lessened the workload on the subjects and helped reduce the total duration.

3.2 Materials Used

A specific type of system was carefully selected for each feature to be tested, and we developed several specifications for each system used in the experiment. The specifications for a particular system were entirely equivalent, and differed only with respect to the particular feature being tested. There are several issues that were considered in designing material for each part of the experiment.

First, we wanted the systems specified to be taken from several aerospace applications[3]. Most important was our requirement that the systems selected be real systems: We did not want to use specifications that were contrived, pathological, or unrealistic. Size and scalability are, of course, also important issues: Language features may become much more influential when dealing with large systems. However, we restricted the size of the systems used for a couple of reasons. First, we wanted to minimize the duration of the experiment; as noted, duration may affect performance. Using large system specifications can require a significant amount of familiarization time to be able to read and answer questions about them. We also decided to limit size in order to minimize potential sources of experimental error. Related experiments have found that several factors of a system specification can affect its readability, including the way a specification is modularized, fonts used, and naming conventions. As the size of the system is increased, there is a greater potential for such factors to affect a specification's readability, in addition to the specific feature being tested. By restricting the size of the specifications used, we hoped to minimize any potential effect that external factors may have on the subject's performance.

Another concern in designing the experiment material is that the notations used be generic, so that the effect of prior experience with particular state-based languages was reduced and so that we were not evaluating specific specification languages but generic features. We also felt it important to vary the notations used, so that a subject's preferences would not be affected over the duration of the experiment. For example, if we always described triggers using propositional logic, the subject might become biased against other notations, which could affect performance on other parts of the experiment.

Because of the large scope of the experiment, a detailed description is not possible in the space allowed. Instead, section 3.3 describes the experiment looking at the readability of conditions to provide the reader with a better understanding of the experimental design. More information about the other experiments can be found in [9]. The results of the complete experiment can be found in section 4.

[3] Although the experiment was designed for an aerospace environment, the results should be applicable to other engineering applications.

3.3 Evaluating Readability of Conditions

The system chosen for this part of the experiment is a simple Speed Mode indicator, which operates as part of a flight management system (FMS). The Climb FMS Speed Mode describes the mode to which the FMS should transition depending on the state of the system and its environment. It can be modeled as a single state machine with four states: *Default*, *Economy*, *Max Climb*, and *Edit*. These states and the conditions that trigger transitions between them are described in each specification. The Speed Mode indicator was suitable for this part of the experiment because it has a small number of states and transitions, but the conditions themselves are quite complex.

Four different notations for the expression of conditions were tested, all of which are shown in detail below—textual, graphical, logical, and tabular. The triggering conditions are broken down into a disjunction of conjunctions, so that they can be expressed in a similar format, regardless of the notation used. This approach may mask some benefits obtained by using certain notations, but should also minimize the effect of structure on the readability of the specifications.

3.3.1 Textual notation

The textual expression reads as straightforward English text. The four parts of the textual state machine reads:

The Climb FMS Speed Mode shall be the **Default** if any of the following scenarios are true:

1. the Flight Phase transitions to Done
2. the Flight Phase transitions from Takeoff to Descent
3. the Flight Phase transitions from Climb to Cruise
4. the Flight Phase transitions from Climb to Descent
5. the Climb FMS Speed Mode is Max Climb
   AND
   at least one of the following is true:
   a. FCC Engaged Mode is Altitude Hold Speed
   b. FCC Engaged Mode is Altitude Hold Idle Thrust
   c. FCC Engaged Mode is Altitude Hold Maximum Thrust
6. Engine Out transitions from Not Engaged to Engaged
7. FMS Mode is Lateral Only

The Climb FMS Speed Mode shall be **Economy** if any of the following scenarios are true:

1. Economy is requested for the FCC Speed Mode
   AND
   one of the following is true:
   a. Flight Phase is Preflight
   b. Flight Phase is Takeoff
   c. Flight Phase is Climb
2. AFS Speed is requested for the FCC Speed Mode
   AND
   one of the following is true:
   a. Flight Phase is Preflight
   b. Flight Phase is Takeoff
   c. Flight Phase is Climb

3. Economy is requested for the Climb Speed Mode
   AND
   one of the following is true:
   a. Flight Phase is Takeoff
   b. Flight Phase is Climb

The Climb FMS Speed Mode shall be **Max Climb** if any of the following scenarios is true:

1. Max Climb is requested for the Climb FMS Speed Mode

The Climb FMS Speed Mode shall be **Edit** if any of the following scenarios is true:

1. Edit CAS is requested for the FCC Speed Mode
   AND
   one of the following is true:
   a. Flight Phase is Preflight
   b. Flight Phase is Takeoff
   c. Flight Phase is Climb
2. Edit Mach is requested for the FCC Speed Mode
   AND
   one of the following is true:
   a. Flight Phase is Preflight
   b. Flight Phase is Takeoff
   c. Flight Phase is Climb
3. Edit is requested for Climb Speed Mode
4. Flight Phase transitions from Cruise to Climb
   AND
   Climb FMS Speed Mode previously in Economy Mode
   AND
   Cruise FMS Speed Mode previously in Edit Mode
5. Climb FMS Speed Mode previously in Economy Mode
   AND
   Descent FMS Speed Mode previously in Edit Mode
   AND
   one of the following is true:
   a. Flight Phase transitions from Descent to Takeoff
   b. Flight Phase transitions from Descent to Climb
   c. Flight Phase transitions from Approach to Takeoff
   d. Flight Phase transitions from Approach to Climb

3.3.2 Logical notation

The logical notation for the speed mode specification is shown in Figure 2. The notation uses simple propositional logic to express conditions. One problem common to parenthetical notations is that they are difficult to decompose, i.e., to see how the parentheses line up. For this reason, a standard size font (courier) was used, and the lines of the specification were indented and aligned to make the structure of the logic more readable. The logical specification was nonetheless the most concise of the four tested.

Default =

```
(PREV(Flight Phase≠Done) ∧ (Flight Phase=Done)) ∨
(PREV(Flight Phase=Takeoff) ∧ (Flight Phase=Descent)) ∨
(PREV(Flight Phase=Climb) ∧ (Flight Phase=Cruise)) ∨
(PREV(Flight Phase=Climb) ∧ (Flight Phase=Descent)) ∨
((Climb FMS Speed Mode=Max Climb) ∧
 ((FCC Engaged Mode=Altitude Hold Speed) ∨
  (FCC Engaged Mode=Altitude Hold Idle Thrust) ∨
  (FCC Engaged Mode=Altitude Hold Maximum Thrust))) ∨
(PREV(Engine Out=Not Engaged) ∧ (Engine Out=Engaged)) ∨
(FMS Mode = Lateral Only)
```

Economy =

```
((Requested FCC Speed Mode=Economy) ∧
 ((Flight Phase is Preflight)∨(Flight Phase is Takeoff) ∨
  (Flight Phase is Climb))) ∨
((Requested FCC Speed Mode=AFS Speed) ∧
 ((Flight Phase is Preflight) ∨
  (Flight Phase is Takeoff) ∨ (Flight Phase is Climb))) ∨
((Requested Climb Speed Mode = Economy) ∧
 ((Flight Phase is Takeoff) ∨ (Flight Phase is Climb)))
```

Max Climb =

```
(Requested Climb FMS Speed Mode=Max Climb)
```

Edit =

```
((Requested FCC Speed Mode=Edit CAS) ∧
 ((Flight Phase is Preflight)∨(Flight Phase is Takeoff) ∨
  (Flight Phase is Climb))) ∨
((Requested FCC Speed Mode=Edit Mach) ∧
 ((Flight Phase is Preflight)∨(Flight Phase is Takeoff) ∨
  (Flight Phase is Climb))) ∨
(Requested Climb Speed Mode=Edit) ∨
((PREV(Flight Phase=Cruise) ∧ (Flight Phase=Climb)) ∧
 PREV(Climb FMS Speed Mode=Economy Mode) ∧
 PREV(Cruise FMS Speed Mode=Edit Mode)) ∨
(PREV(Climb FMS Speed Mode=Economy Mode) ∧
 PREV(Descent FMS Speed Mode=Edit Mode) ∧
 ((PREV(Flight Phase=Descent) ∧ (Flight Phase=Takeoff)) ∨
  (PREV(Flight Phase=Descent) ∧ (Flight Phase=Climb)) ∨
  (PREV(Flight Phase=Approach)∧(Flight Phase=Takeoff)) ∨
  (PREV(Flight Phase=Approach) ∧ (Flight Phase=Climb))))
```

Figure 2. Logical specification for trigger conditions.

3.3.3 Graphical notation

The graphical notation makes use of logic gates to express conditions. There are several different ways to express conditions graphically. However, the logic gate is a notation with which most engineers are familiar. Figure 3 shows the graphical speed mode specification. One of the final gates (reading from left to right) in the gate specification evaluates to true depending on whether the speed mode is *Default, Economy, Max Climb* or *Edit*. The graphical specification was the lengthiest, due to the spatial layout required by the logic gate notation.

3.3.4 Tabular notation

The tabular specification uses AND/OR tables (developed for RSML) to express trigger conditions. The tables are used differently than tables in other state-machine specification languages such as SCR, where tables are used to describe the actual transitions between states, rather than the details of the triggering conditions. The AND/OR tables simply represent one predicate logic statement about the conditions on one transition arrow between states. The far-left column of the AND/OR table lists the logical phrases of the predicate. Each of the other columns is a conjunction of those phrases and constrains the logical values of the expressions. If one of the columns evaluates to *true*, then the entire table evaluates to *true*. A column evaluates to *true* if all of its elements match the truth-values of the associated predicates. A dot denotes "don't care." Figures 4 and 5 show the tabular speed mode specification. For example, the state variable "Climb FMS Speed Mode" will be in *Economy* mode if its AND/OR table in Figure 4 evaluates to *true*. This will happen if, for example, both "Requested FCC Speed Mode = Economy" is *true* and "Flight Phase = Preflight" is *true*, OR if both "Requested FCC Speed Mode = Economy" and "Flight Phase = Takeoff" are *true*, OR if any of the other columns in that table evaluate to *true*.

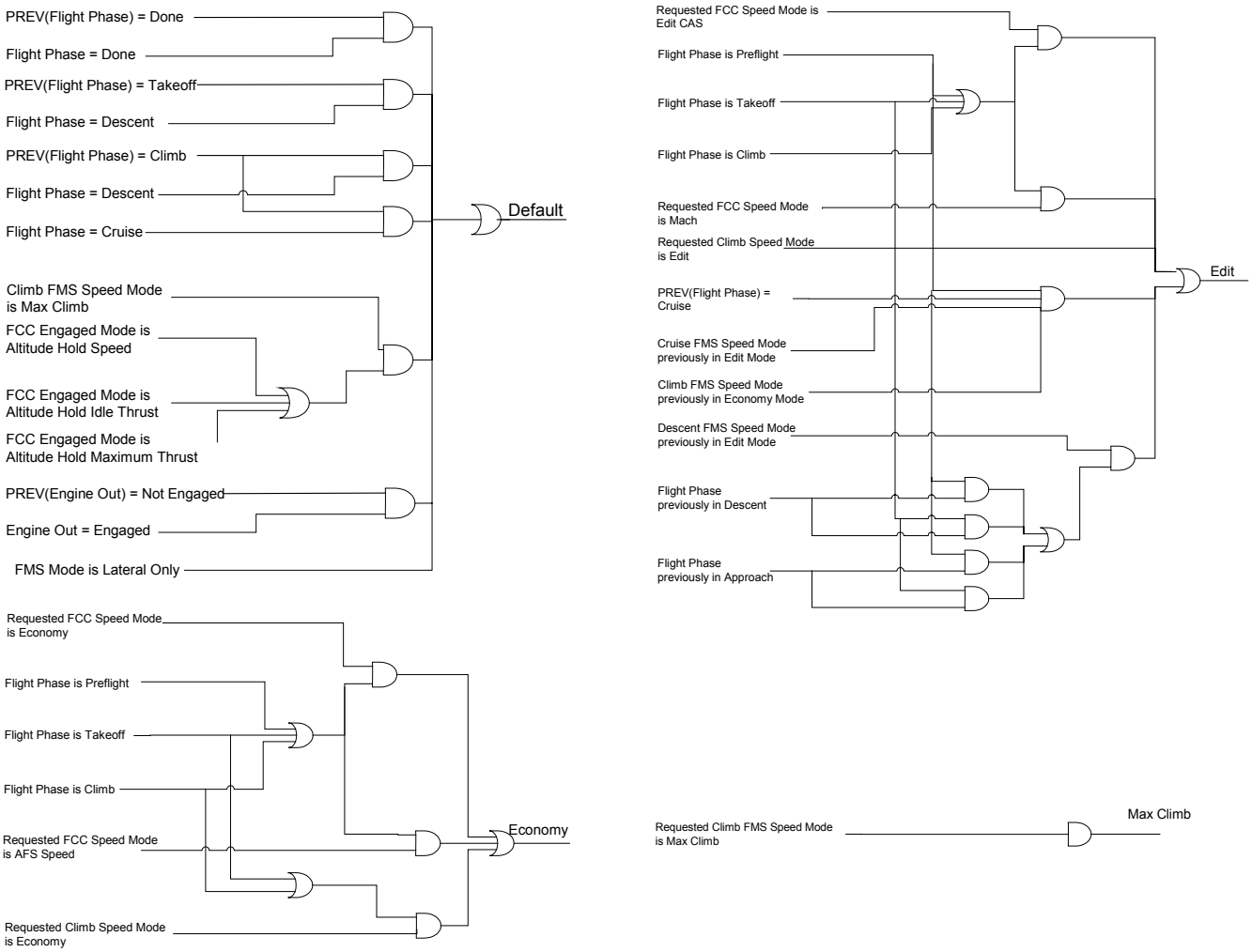

Figure 3. Graphical specification for trigger conditions.

= Max Climb **IF**

Requested Climb FMS Speed Mode = Max Climb	T

= Edit **IF**

Requested FCC Speed Mode = Edit CAS	T	T	T	*	*	*	*	*	*	*	*	*
Flight Phase = Preflight	T	*	*	T	*	*	*	*	*	*	*	*
Flight Phase = Takeoff	*	T	*	*	T	*	*	*	T	*	T	*
Flight Phase = Climb	*	*	T	*	*	T	*	T	*	T	*	T
Requested FCC Speed Mode = Edit Mach	*	*	*	T	T	T	*	*	*	*	*	*
Requested Climb FMS Speed Mode = Edit	*	*	*	*	*	*	T	*	*	*	*	*
PREV(Flight Phase) = Cruise	*	*	*	*	*	*	*	T	*	*	*	*
PREV(Climb FMS Speed Mode) = Economy Mode	*	*	*	*	*	*	*	T	T	T	T	T
PREV(Cruise FMS Speed Mode) = Edit Mode	*	*	*	*	*	*	*	T	*	*	*	*
PREV(Descent FMS Speed Mode) = Edit Mode	*	*	*	*	*	*	*	*	T	T	T	T
PREV(Flight Phase) = Descent	*	*	*	*	*	*	*	*	T	T	*	*
PREV(Flight Phase) = Approach	*	*	*	*	*	*	*	*	*	*	T	T

Figure 4. Tabular specification for trigger conditions.

= Default **IF**

PREV(Flight Phase) = Done	F	*	*	*	*	*	*	*	*
Flight Phase = Done	T	*	*	*	*	*	*	*	*
PREV(Flight Phase) = Takeoff	*	T	*	*	*	*	*	*	*
Flight Phase = Descent	*	T	*	T	*	*	*	*	*
PREV(Flight Phase) = Climb	*	*	T	T	*	*	*	*	*
Flight Phase = Cruise	*	*	T	*	*	*	*	*	*
Climb FMS Speed Mode = Max Climb	*	*	*	*	T	T	T	*	*
FCC Engaged Mode = Altitude Hold Speed	*	*	*	*	T	*	*	*	*
FCC Engaged Mode = Altitude Hold Idle Thrust	*	*	*	*	*	T	*	*	*
FCC Engaged Mode = Altitude Hold Maximum Thrust	*	*	*	*	*	*	T	*	*
PREV(Engine Out) = Not Engaged	*	*	*	*	*	*	*	T	*
Engine Out = Engaged	*	*	*	*	*	*	*	T	*
FMS Mode = Lateral Only	*	*	*	*	*	*	*	*	T

= Economy **IF**

Requested FCC Speed Mode = Economy	T	T	T	*	*	*	*	*
Flight Phase = Preflight	T	*	*	T	*	*	*	*
Flight Phase = Takeoff	*	T	*	*	T	*	T	*
Flight Phase = Climb	*	*	T	*	*	T	*	T
Requested FCC Speed Mode = AFS Speed	*	*	*	T	T	T	*	*
Requested Climb Speed Mode	*	*	*	*	*	*	T	T

Figure 5. Tabular specification for trigger conditions

3.3.5 Condition experiment: Questions

The part of the experiment designed to test how the representation of trigger conditions affects readability consisted of 18 objective questions and 4 subjective questions, all shown here below. For the first two objective questions the subject was allowed to use any of the four specifications, thereafter four questions were designated for each specification. Finally, an evaluation was collected with four subjective questions.

(Any specification)

1. Describe two scenarios that would cause the Climb FMS Speed Mode to be Edit.

2. Could the Climb FMS Speed Mode be Max Climb under the following conditions?

Engine Out is Engaged
Requested Climb FMS Speed Mode = Economy
Flight Phase transitions from Takeoff to Climb

(Tabular specification only)

3. Which part of the specification specifies that the Climb FMS Speed Mode will be the Default when the flight phase transitions from Climb to Descent (label rows and columns in table)?

4. If the FCC Engaged Mode is Altitude Hold Speed, what additional conditions are necessary in order for the Climb FMS Speed Mode to be Default?

5. Could the Climb FMS Speed Mode be Default under the following conditions?

FMS Mode is Lateral Only
Engine Out is Not Engaged
Flight Phase is Cruise
Economy is requested for the FCC Speed Mode

6. Suppose that in order for the Climb FMS Speed Mode to be Edit, the FMS Mode must be Lateral-Vertical (this is in addition to the existing requirements). What changes should be made to the specification to reflect this behavior?

(Textual specification only)

7. Which part of the specification specifies that the Climb FMS Speed Mode will be the Max Climb when the Max Climb is requested (label lines in the specification)?

8. If the flight phase is Preflight, under what conditions will the Climb FMS Speed Mode be Economy?

9. Under the following conditions,

FMS Mode is Lateral-Vertical
Engine Out transitions from Not Engaged to Engaged
Flight Phase is Cruise
Economy is requested for the Climb Speed Mode

could the Climb FMS Speed Mode be Default? Economy?

10. Suppose we want to add a new mode for the Climb FMS Speed Mode, called Flex. The Climb FMS Speed Mode will be Flex if the FMS Mode is Lateral-Vertical, and the flight phase is either Takeoff, Climb, or Cruise. What additions should be made to the specification to reflect this behavior?

(AND/OR gate specification only)

11. Which part of the specification specifies that the Climb FMS Speed Mode will be Default when the FMS Mode is Lateral Only (label inputs/gates in the specification)?

12. If the flight phase transitions from Approach to Done, what additional conditions are necessary in order for the Climb FMS Speed Mode to be Default?

13. Could the Climb FMS Speed Mode be Edit under the following conditions?

there is no requested Climb FMS Speed mode
Flight Phase is Preflight
Economy is requested for the FCC Speed Mode

14. Suppose that in order for the Climb FMS Speed Mode to be Edit, the FMS Mode must be Lateral-Vertical (this is in addition to the existing requirements). What additions should be made to the specification to reflect this behavior?

(Propositional logic specification only)

15. Which part of the specification specifies that the Climb FMS Speed Mode will be Economy when the Requested FCC Speed Mode is Economy and the Flight Phase is Preflight (label lines in the specification)?

16. If Edit CAS is requested for the FCC Speed Mode, what must the flight phase be in order for the Climb FMS Speed Mode to be Edit?

17. Could the Climb FMS Speed Mode be Economy under the following conditions?

Requested for the FCC Speed Mode is AFS Speed
FCC Engaged Mode is Altitude Hold Economy Thrust
Flight Phase transitions from Descent to Cruise

18. Suppose we want to add a new mode for the Climb FMS Speed Mode, called Endurance. The Climb FMS Speed Mode will be Endurance if the FMS Mode is Lateral-Vertical, there is no requested FCC Speed Mode, and the flight phase is either Takeoff, Climb, or Cruise. What additions should be made to the specification to reflect this behavior?

Conditional Evaluation

1. Rank the textual, tabular, graphical, and logical specification in terms of readability.

2. Do your preferences change with respect to ease of editing?

3. Did you find that certain forms were good for certain tasks, or were there specifications that you consistently found to be easy to use in the experiment?

4. Do you consider it worthwhile to have several representations available? If so, which ones?

The results of the conditions experiment can be found in the next section.

4. RESULTS

There are two parts of the results analysis—objective and subjective. The objective questions for all subjects were graded by the same person to ensure consistency in the analysis. Questions were graded as either correct, partially correct, or incorrect. We did not refine the grading process any more than this, as it did not seem likely that it would affect our conclusions. This grading system is by no means ideal, however. A subject may answer a question incorrectly, but the grader will not be able to distinguish between the subject making a careless error versus not understanding the specification. The only way to clear up this ambiguity would be to talk with each subject about the thought process involved in answering each question, which was not feasible in this experiment

Another problem with this grading system is that although subjects were encouraged to skip questions that required more than ~90 seconds to answer, subjects often worked on a question for several minutes due to the fact that there was no enforced time limit. This situation is not reflected in our grading system. If a subject worked on a problem for 5 minutes, but answered it correctly, it would be recorded just as any correct answer.

The only major unanticipated problem we experienced with the experiment was its duration. As discussed earlier, we took great care to control the duration of the experiment and anticipated, based on pretesting, that each subject would complete it in roughly 2.5 hours. However, in practice subjects took between 2.75 and 4 hours. Subjects were offered short breaks (~10 minutes) after each of the 6 parts was completed to lessen duration side effects.

Though performance time was not officially measured, it was interesting to note that, in general, those with a computer science background performed much faster than those with an aeronautics background, but with comparable accuracy. Computer science students finished in approximately 3 hours, and aeronautics students in almost 4 hours. This observation was contrary to our expectations. We expected that although aerospace engineers were not likely to be familiar with state machines, they would be familiar with the types of aerospace systems used. However, it appears that familiarity with state machines is a much more influential factor than is familiarity with the systems themselves. This observation may help explain the lack of widespread adoption of formal methods among aerospace industries—aerospace engineers are not accustomed to using such notations.

One of the computer scientist subjects later offered another explanation for the difference in performance times. He said that he enjoyed answering the questions because many reminded him of computer science exams, where he was often asked to trace through a system specification or to answer questions about a system's behavior given a specification. Aerospace engineering students are rarely required to answer questions about system specifications.

Representation of State Machines: We tested tabular, graphical, and textual specifications. Subjects consistently performed well on the objective questions. Most of the mistakes made by those with a computer science background occurred when dealing with the tabular specifications, whereas every aerospace student answered these questions correctly. Using a two-tail t-test, we were able to show a statistically significant difference between the two groups in this respect. When given a choice of using any of the specifications for two questions, every subject chose to use the table for the first question and all but two chose to use the table on the second (as an aside, only one subject answered the second question incorrectly and this was one of the two who did *not* use the table).

Subjects evaluated the graphical specification as useful for obtaining a high-level understanding of the system, but said it became cumbersome when asked to answer questions about the

triggers for transitions. Though easy to read, subjects (with two exceptions) found the textual specifications very difficult to use.

According to subjective rankings, eleven of the 12 subjects found either the graphical or the tabular representation to be the most readable (with more preferring the tabular) in terms of usability in answering questions about the specification, with only 1 subject preferring to use the textual specification. This observation is interesting given that most specifications used today are textual.

Conditions: We tested textual, graphical, tabular, and logical expressions of trigger conditions. Subjects generally performed very well on the objective portion of the conditions experiment, with no discernable pattern existing among the types of questions answered incorrectly. However, four subjects answered question 18 incorrectly, which asked them to edit the propositional logic specification. These errors were mainly due to mismatched parentheses and brackets, which of course is a difficulty encountered when expressing complex triggers using propositional logic.

As far as subjective rankings, nine of the 12 subjects found the tabular specification (AND/OR Tables) to be the most readable, followed by the textual, then graphical, and finally logical. Seven of the twelve ranked the propositional logic specification as the least readable. These rankings were consistent among computer science and aerospace students. The rankings were slightly different, however, with respect to ease of editing: Subjects on average found the tabular specification to be the easiest to edit, followed by the graphical, textual, and logical. We found it surprising that every computer science student commented on the difficulty of using the graphical specification, while feelings about it were mixed for the aerospace students.

Based on both objective and subjective evaluations, the tabular representation of transition conditions appears to be the most readable of those tested, while the propositional logic notation was the least readable when used to express complex behavior.

Macros: Results at first seemed to imply that macros were not conducive to readability. Subjects were given their choice of specifications to answer two questions, and 11 of the 12 subjects chose to use the flat specification. Furthermore, only 5 of the 12 subjects ranked the macros specification as the more readable. However, 11 subjects ranked the macros specification as the easier of the two to edit.

In the subjective feedback, subjects consistently felt that macros were beneficial when specifying complex systems, leading to less cluttered and more compact logical expressions. They also noted that macros assist in reuse and that with proper naming conventions, they could make logical expressions easy to read. However, subjects also commented that macros often made it difficult to navigate complex specifications, requiring lots of page flipping to understand how a particular input affects system behavior. This effort lead to a loss of continuity when reading the specification.

Based on these results, subject interviews, and our experiences with much larger specifications than used in this experiment, it appears that macros are useful when writing a specification, but that they can become difficult to navigate when reading a large specification. Prohibiting nested macros may be a compromise for this situation. However, using macros in an automated environment may alleviate much of the difficulty encountered using macros in this experiment. Navigation could be simplified using automated hyperlinks or even made unnecessary by the automatic expansion of a macro in place.

Internally Broadcast Events: Results regarding the readability of internal events were inconclusive. Seven of the twelve subjects ranked the events specification as more readable. However, difficulty in editing the specifications with events was clear: ten of the twelve found it easier to modify the eventless specification and eleven of the twelve subjects were unable to edit the events specification correctly. This latter result supports our experience with TCAS II and the errors we and our reviewers made. Size and complexity of the specification may be a critical factor here. We note also that although in general the subjects asked few questions during the experiment, a few needed additional explanation regarding the use of internal events, again validating our experiences with TCAS of the difficulty new users may have with internal events.

Hierarchies: All 12 subjects agreed that a hierarchical specification is easier to read than a flat specification and that hierarchical abstractions are absolutely necessary to specify complex systems. However, it is extremely interesting that half of the subjects unknowingly made errors reading the hierarchical specifications, errors that were not made when reading the equivalent flat specification. These results also coincide with our experience and point to the need for careful language design to reduce errors in reading hierarchical specifications.

Perspective: Seven subjects felt that the going-to and coming-from perspectives were interchangeable, and nine subjects responded that it is better to become accustomed to one perspective rather than be provided with both, despite the fact that certain questions were clearly easier to answer using one perspective over the other. However, our results suggest that subjects may actually prefer access to both perspectives in an automated environment.

5. CONCLUSIONS

As an assessment of formal specification readability, these results are preliminary. We would ultimately like to run more extensive experiments focusing on two or three of these features. Based on our results, we will likely focus on conditions, macros and internal events. Complex conditions seem to be more readable when expressed in a table, but we would like to test this with a statistically significant number of subjects before claiming it as a conclusion.

The macros portion of our experiment was inconclusive regarding their effect on readability. Though they seem to be extremely useful when writing and editing specifications, there seems to be difficulty reading specifications with macros, which we would like to address in future work. Specifically, we would like to investigate how various levels of modularization (i.e. nested macros) affect a specification's readability. Macros also are one feature whose utility is dependent on the size of the specification. We would like to investigate how macros can affect the readability of large-scale system specifications.

Future work will also likely address the use of internal events. Again, while most subjects found it easier to edit an eventless specification, feelings were mixed regarding whether it is easier to read an eventless specification. The TCAS experience found that reading large specifications was impeded by internal events.

We would like to empirically evaluate this claim by extensively testing the readability of large specifications with internal events.

We are not planning to investigate the readability of hierarchies or perspective in future work. Our results, as well as previous experience, show that hierarchical abstractions are essential if formal methods are to scale, so we do not feel that any insight would come from further experimentation. We have learned, however, that the semantics of the notations describing these hierarchies need to be carefully defined. Perspective does not appear to be a significant factor affecting readability, and so it does not seem worthwhile to investigate further.

The most important issue that will be addressed in future work is readability of large-scale system specifications. Scalability is one of the largest obstacles facing the adoption of formal methods and was not adequately addressed in this experiment. For example, we tested the readability of a flat specification. While some subjects found a flat specification easy to read when describing a state machine with ten states, it is clearly inappropriate when describing a state machine with 10^{30} states.

If through experimentation we can determine which notations and features are conducive to making large system specifications readable, we should be able to use this information to design more readable and reviewable state-based requirements specification languages.

6. REFERENCES

[1] Brooks, R. Studying Programmer Behavior Experimentally: The Problems of Proper Methodology. Communications of the ACM. vol 23, no. 4, April 1980. pp. 207-14.

[2] Finney, K., Fenton, N., and Fedorec, A. Effects of Structure on the Comprehensibility of Formal Specifications. IEE Proceedings - Software. vol 146, no. 4, August 1999. pp. 193-202.

[3] Fitter, M. and Green, T.R. When Do Diagrams Make Good Computer Languages? International Journal of Man-Machine Studies. vol. 11, 1979. pp. 235-261.

[4] Harel, D, et.al. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming 8. Elsevier Science Publishers B. V., North Holland. 1987. pp. 231-274.

[5] Heimdahl, M., Leveson, N., and Reese, J. Experiences From Specifying the TCASII Requirements Using RSML. Proceedings of the 17th Digital Avionics Systems Conference, November 1998.

[6] Heitmeyer, C., Jeffords, R., and Labaw, B. Automated Consistency Checking of Requirements Specifications. ACM Transactions on Software Engineering and Methodology, vol 5, no 3, July, 1996. pp. 231-261.

[7] Leveson, N. Completeness in Formal Specification Language Design for Process Control Systems. Proceedings of Formal Methods in Software Practice Conference, August 2000.

[8] Sherry, L., Youssefi, D., and Hynes, C. A Formalism for the Specification of Operationally Embedded Reactive Avionic Systems. HSR FD G&C Task 4 - VMS Structure Development. Honeywell, October 1995.

[9] Zimmerman, Marc. Investigation the Readability of Formal Specification Languages. M.Sc. thesis, Massachusetts Institute of Technology, May, 2001.

[10] Zimmerman, M., Rodriguez, M., Ingram, B., Katahira, M., de Villepin, M., and Leveson, N. Making Formal Methods Practical. Proceedings of the 19th Digitial Avionics Systems Conference. October 2000.