MAKING FORMAL METHODS PRACTICAL

Marc Zimmerman, Mario Rodriguez, Benjamin Ingram,

Masafumi Katahira, Maxime de Villepin, Nancy Leveson, MIT, Cambridge, MA

Abstract

Despite its potential, formal methods have had difficulty gaining acceptance in the industrial sector. Some complaints are based on supposed impracticality: Many consider formal methods to be an approach to system specification and analysis that requires a large learning time. Contributing to this skepticism is the fact that some types of formal methods have not yet been proven to handle systems of realistic complexity. To learn more about how to design formal specification languages that can be used for complex systems and require minimal training, we developed a formal specification of an English language specification of the vertical flight control system similar to that found in the MD-11.[1] This paper describes the lessons learned from this experience. A companion paper at this conference describes how the model can be used in human--computer interaction analysis and pilot task analysis [3].

1 Introduction

Formal specifications and mathematical 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. They have the potential for both increasing safety and decreasing the cost of certifying flight-critical 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 the lack of wide-spread adoption. First, engineers are not trained in discrete mathematics and the notations are not as parsimonious as continuous math. 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. In addition, the tools provided for formal analysis of such specifications are often difficult for engineers to use: They may require in-depth knowledge of discrete mathematics, and may require domain experts to translate the problem as they understand it in their domain of expertise into another domain; for example, they may be required to specify a set of axioms that describe the domain and to restate the problem in terms of theorems and lemmas that they must then prove, perhaps with some assistance from an automated tool.

In addition, the scope and scalability of formal methods remain additional concerns both in industrial and academic communities. There has as yet been only limited success with applying formal methods to complex systems.

We have been experimenting with the design of formal specification languages and analysis tools with usability as a major design criterion. This paper describes our experiences and experimental results in building and using a formal model of the blackbox requirements for a vertical flight control system. The remainder of the paper is organized as follows: Section 2 describes the modeling language we selected for this case study, SpecTRM-RL, and the resulting model. Section 3 describes the lessons learned from the case study and how they might be used to point to further research directions, and compares the language used with others that have been proposed.

[1] The specification, although realistic, is not identical to the real MD-11 software and should not be used to infer anything about that aircraft's software.

2 The SpecTMR-RL Specification Language

To learn more about how to design formal specification languages, we have been experimenting with various language features and using the prototype languages to learn more about what features should be included in such languages. Our latest experimental prototype is called SpecTRM-RL (Specification Tools and Requirements Methodology—Requirements Language).

SpecTRM-RL is part of a larger specification methodology, called Intent Specifications, that includes both formal and informal specifications [1]. The blackbox behavioral requirements specifications (level 3 of an intent specification) are described using SpecTRM-RL, which has an underlying state-machine model. The language itself abstracts away from this formal model and emphasizes readability and reviewability, which distinguishes it from most other formal specification languages.

Reviewability is arguably one of the most important properties of any specification. The specification must not only be reviewable by one group trained in the language, but must be readable by a large variety of people with diverse backgrounds and expertise including system designers and developers, customers, users, certifiers, etc. Readability by a general audience allows all involved parties to discuss and analyze a specification using a single common model. Furthermore, 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. This fact does not mean that automated tools are not useful and important in finding some types of errors, especially those that require rather tedious checks, but humans are required to determine whether a specification conforms with engineering expectations and requirements. In addition, even those design or specifications flaws found by tools will need to be evaluated by human experts. Therefore, readability of system specifications is a requirement not only for human understanding of complex models but also for human processing of the analysis results.

Readability is also desirable as it has associated with it a short familiarization time. Certainly any specification language is going to require some training in order to understand and use. However, particularly with respect to review, this time cannot be too long or it becomes impractical to have the large amount of reviewing that leads to high-quality specifications and software.

Leveson's research group started to look at reviewability and design of formal specification languages while creating such a specification for TCAS II for the FAA. Since that time, we have been creating specifications of real systems and experimenting with specification language features with respect to usability. SpecTRM-RL is the latest manifestation of the lessons we have learned to date.

We believe that readability and reviewability are enhanced by minimizing semantic distance between the reviewer's mental model of the system being designed and the specification model. Semantic distance can loosely be defined as the amount of effort required to translate from one model to another. We believe that the application expert's ability to find errors in a requirements specification can be enhanced by reducing the semantic distance between their understanding of the required process control behavior and the specification of that behavior. This, in turn, implies that specifications use familiar engineering notations and that they be written in terms of the externally observable behavior of the component being specified. Any information related only to the implementation of that behavior should not be included. That is, the specification should be blackbox. Thus SpecTRM-RL specifications include only the input/output function being computed by the component (the transfer function) and do not include any information about the internal design of the component or how that externally visible behavior is actually realized. In fact, the blackbox behavior might be achieved through the use of hardware or software. In addition, we hypothesize that state-machine models, i.e., a description of a system's behavior in terms of states and transitions between states, is a natural way for engineers to think about control systems. SpecTRM-RL therefore is based on an underlying state machine model.

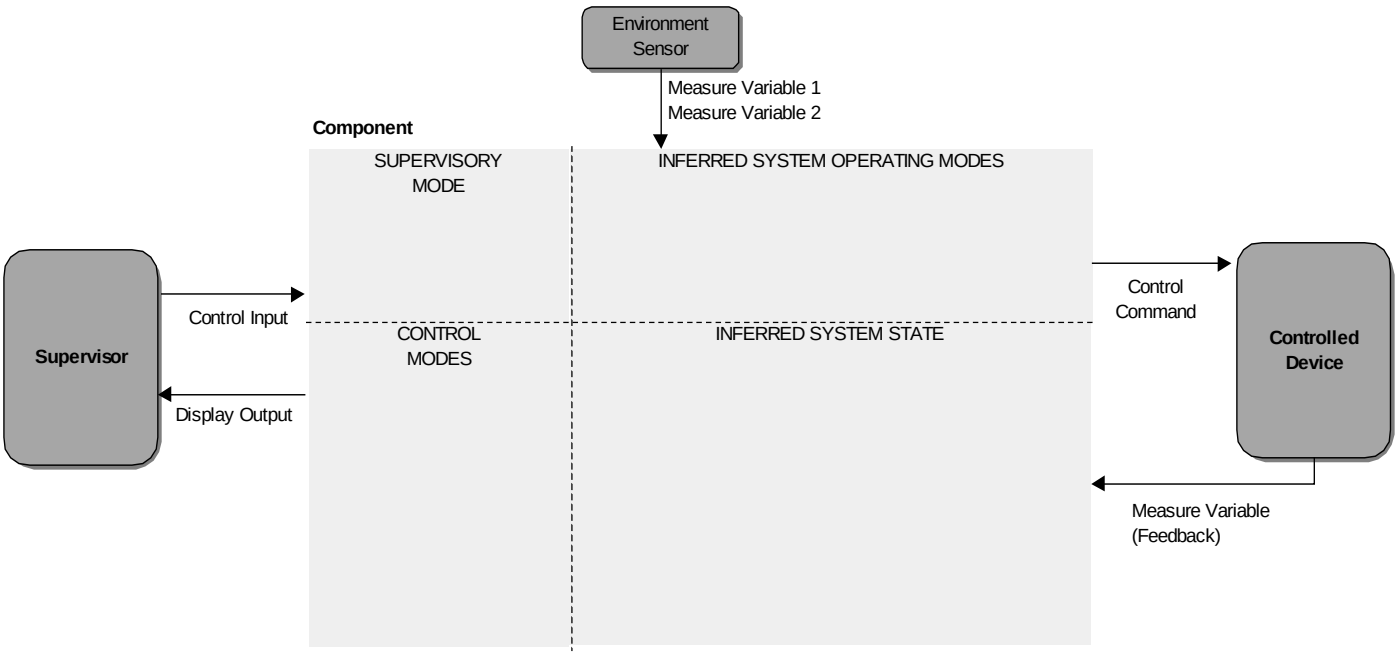

Figure 1. The Form of a SpecTRM-RL Specification

Readability is also enhanced, we believe, by limiting the semantic domain of the language. SpecTRM-RL was designed primarily for process-control systems. The high-levels of the specification look very similar to process-control diagrams. Figure 1 shows the basic format of the specification. There are four main parts: (1) a specification of the supervisory modes of the controller being modeled, (2) a specification of its control modes (3) a model of the controlled process (or *plant* in control theory terminology) that includes the inferred operating modes and system state (these are inferred from the measured inputs), and (4) a specification of the inputs and outputs to the controller. The graphical notation mimics the typical engineering drawing of a control loop.

Every automated controller has at least two interfaces: one with the supervisor(s) that issues instructions to the automated controller (the supervisory interface) and one with each controlled system component (controlled system interface). The supervisory interface is shown to the left of the main controller model while the interface with the controlled component is to shown the right. There may be additional interfaces (shown at the top) with various environmental sensors.

The supervisory interface consists of a model of the operator controls and a model of the displays or other means of communication by which the component relays information to the supervisor. Note that the interface models are simply the logical view that the controller has of the interfaces---the real state of the interface may be inconsistent with the assumed state due to various types of design flaws or failures. By separating the assumed interface from the real interface, we are able to model and analyze the effects of various types of errors and failures (e.g., communication errors or display hardware failures). In addition, separating the physical design of the interface from the logical design (required content) will facilitate changes and allow parallel development of the software and the interface design. During development, mockups of the physical GUI or interface design can be generated and tested using the output of the SpecTRM-RL simulator.

Supervisory modes are used in specifying information about the current supervisor of the controller and are useful when a component may have multiple supervisors at any time. For example, a flight control computer in an aircraft may get inputs from the flight management computer and also directly from the pilot. Required

behavior may differ depending on which supervisory mode is currently in effect. Mode-awareness errors related to confusion in coordination between multiple supervisors can be defined (and the potential for such errors theoretically identified from the models) in terms of these supervisory modes. In systems with complex displays (such as Air Traffic Control systems), it may also be useful to define various *display modes*.

The bottom left quadrant of Figure 1 provides information about the *control modes* for the controller itself. These are not internal states of the controller (which are not included in our specifications) but simply represent externally visible behavior about the controller's modes of operation. *Control Modes* are used in describing the required behavior of the controller. Modern avionics systems may have dozens of modes. Control modes may be used in the interpretation of the component's interfaces or to describe the component's required process-control behavior.

The right half of the controller model represents inferred information about the operating modes and states of the controlled system (the *plant* in control theory terminology). A simple plant model may include only a few relevant state variables. If the controlled process or component is complex, the model of the controlled process may be represented in terms of its operational modes and the states of its subcomponents. *Operational modes* are useful in specifying sets of related behaviors of the controlled-system (plant) model. For example, it may be helpful to define the operational state of an aircraft in terms of it being in takeoff, climb, cruise, descent, or landing mode.

In a hierarchical control system, the controlled process may itself be a controller of another process. For example, the flight management system may be controlled by a pilot and may issue commands to a flight control

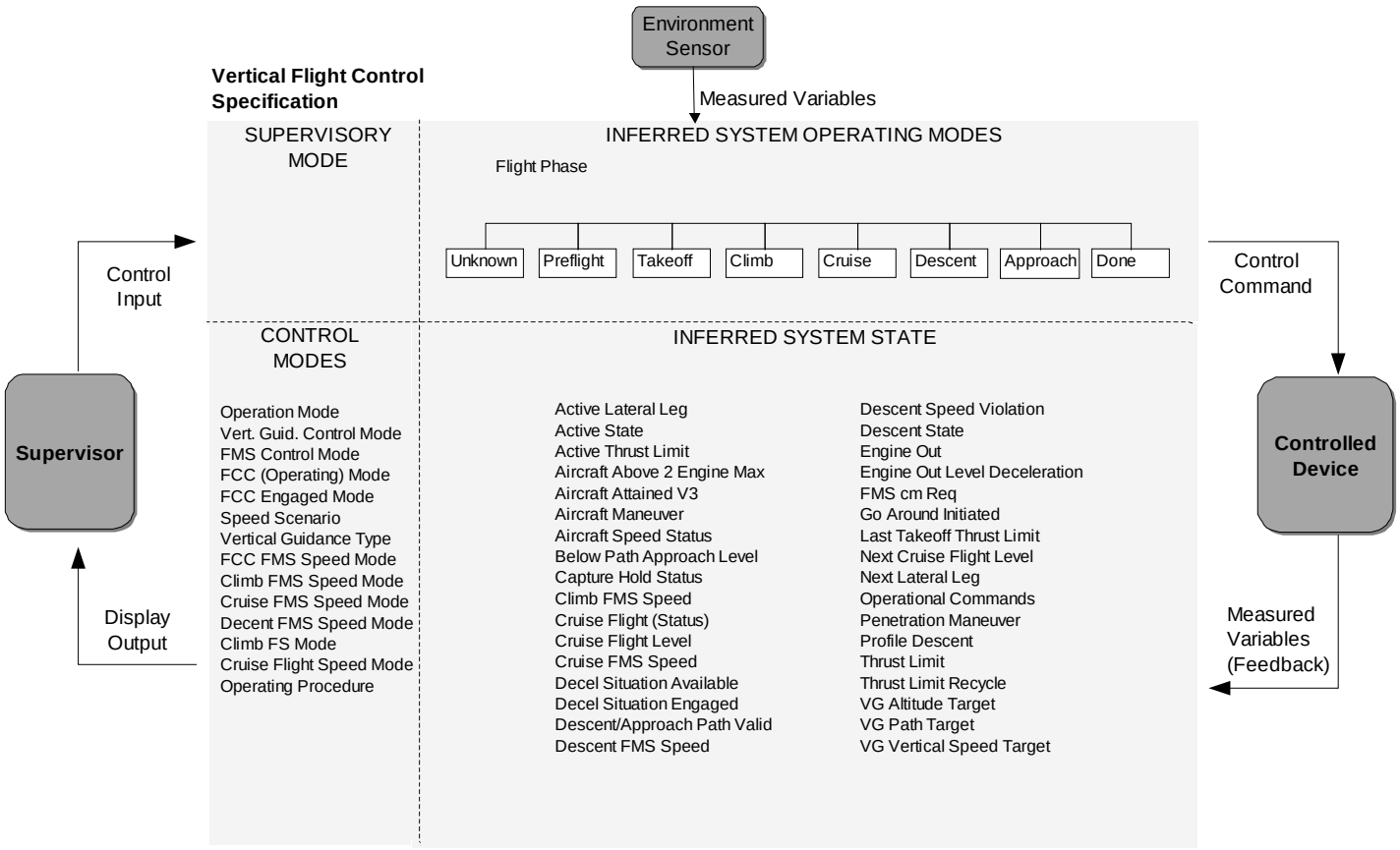

Figure 2. SpecTRM-RL Model for the Vertical Flight Control System.

computer, which issues commands to an engine controller. If, during the design process, components that already exist are used, then those plug-in component models can be inserted into the SpecTRM-RL process model. Additionally, parts of a SpecTRM-RL model can be reused or changed to represent different members of a product family.

The vertical flight control system for a high-tech aircraft like the MD-11 operates as part of the flight management system (FMS) and provides targets and controls necessary to maintain a predetermined vertical profile and provide guidance, control, and annunciation functions. Using the aircraft position relative to its vertical profile and any operational commands from the pilot, the *guidance* function determines appropriate altitude, speed, thrust, and pitch targets as well as the integrated pitch/thrust control mode necessary to maintain the desired trajectory. The *control* function calculates (in real-time) the pitch commands necessary for the aircraft to track the targets computed by the guidance functions. Finally, *annunciation* provides information to the FCC (flight control computer) to be displayed in the cockpit, based on the guidance and control functions. This information includes the flight mode, speed target, and altitude targets of the aircraft.

Figure 2 shows the SpecTRM-RL model of this system. Because the system is too large to show on one page (or one screen), the specification is hierarchically decomposed. Figure 2 also shows the possible values that the *Flight Phase* state variable can assume. In reality, every state variable (including operating modes) listed on the system model is associated with a similar set of values. We listed only the names of the other state variables for space reasons.

Each piece of this model needs to be specified in detail. As an example, Figure 3 shows the specification for the *Target Altitude* output. This variable determines the target altitude for the current leg of the aircraft's path. The conditions under which outputs are assigned values are described using a tabular representation called AND/OR tables. Each AND/OR table is divided into two parts, *Control Modes* and *State Values*. The Control Modes section describes the value of the control modes necessary for the transition, while the State Values section describes the values of and conditions on inputs and state variables. This distinction allows the reader to better understand each mode of the system's behavior, and we found it helpful in detecting specification errors – particularly omissions.

AND/OR tables are concise representations of propositional logic (in disjunctive normal form) that we have found to be easily read and interpreted. The far left column of the table lists logical (boolean) phrases. Each of the other columns is a conjunction of those phrases and contains the logical values of the expressions (a '*' denotes "don't care"). A column evaluates to *true* if all of its elements are true. If one of the columns is *true*, then the table evaluates to *true*. For example, the *Target Altitude* output figure 3 would be *Vertical Guidance Climb Target Altitude* if the *Operating Procedure* is *Airmass Ascent* and the current flight phase is *Takeoff* or *Climb*, OR if the *Operating Procedure* is *Climb InterLev* and the current flight phase is either *Takeoff* or *Climb*.

SpecTRM-RL allows the use of macros. Macros are simply named pieces of AND/OR tables that can be referenced from within another table. For example, figure 4 shows a simple macro that was created for this model. It looks and behaves the same as any logic table in the model, but it can be referenced from other logic tables. The *Valid Aircraft Speed* macro will evaluate to *true* if the corresponding AND/OR table evaluates to *true*. While the use of macros is not necessary, we have found it simplifies the specification and thus makes it easier to understand while also enhancing changeability and specification reuse. Macros, for the most part, correspond to typical abstractions used by application experts in describing the requirements and therefore add to the understandability of the specification. In addition, we have found this feature convenient for expressing hierarchical abstraction and enhancing hierarchical review and understanding of the specification.

3 Lessons Learned

Except for Leveson (who acted only as reviewer), the other authors (who actually created the model) had no previous experience with building formal specifications and limited

Output Variable

Target Altitude

Type: INTEGER
Destination: TBD
Initiation Delay: 0 milliseconds
Completion Deadline: TBD
Exception Handling: None specified
References: N/A

Feedback Information:
Variables: UNDEFINED
Values: UNDEFINED
Min time between outputs: 10 MHz
Max time between outputs: UNDEFINED
Exception Handling: None Specified

:= Vertical Guidance Climb Target Altitude **IF**

TRIGGERING CONDITION

Control Modes	Vertical Guidance Operating Procedure IN_STATE Airmass Ascent	T	T	*	*
	Vertical Guidance Operating Procedure IN_STATE Climb InterLev	*	*	T	T
State Values	Flight Phase IN_STATE Takeoff	T	*	T	*
	Flight Phase IN_STATE Climb	*	T	*	T

:= Active Cruise Flight Level **IF**

TRIGGERING CONDITION

Control Modes	Vertical Guidance Operating Procedure IN_STATE Airmass Ascent	*	T	*
	Vertical Guidance Operating Procedure IN_STATE Climb InterLev	*	*	T
State Values	Flight Phase IN_STATE Cruise	*	T	T
	Active Cruise FL Valid ()	T	*	*

:= Vertical Guidance Descent Altitude **IF**

TRIGGERING CONDITION

Control Modes	Vertical Guidance Operating Procedure IN_STATE Cruise	*	T
State Values	Step Climb IN_STATE False	*	T
	Clearance Altitude < Active Cruise FL – 250	*	T
	Vertical Guidance Descet Target Alt != -1000	T	T
	Active Operational Procedure Valid()	T	*

Notes:
The Vertical Guidance Descent Target Altitude is assigned a default of –1000 ft when no other Descent Alt Constraints are entered in the flight plan.

Figure 3. SpecTRM-RL Specification for the Target Altitude Output.

Macro

Aircraft Speed Valid

Parameters: NONE

Condition:

FlightPhase IN_STATE Takeoff	T	*	*	*	*
FlightPhase IN_STATE Climb	*	T	*	*	*
FlightPhase IN_STATE Cruise	*	*	T	*	*
FlightPhase IN_STATE Descent	*	*	*	T	*
ADC CAS < V_{max} + 10	T	T	T	T	*
ADC CAS > V_{min} – 20	T	T	T	T	*
FlightPhase IN_STATE Approach	*	*	*	*	T
ADC CAS < V_{max} + 10	*	*	*	*	T
ADC CAS > V_{min} – 10	*	*	*	*	T

Figure 4. Sample Macro Definition.

knowledge of formal specification languages. Most, in fact, had never written a requirements specification before. Training involved simply reading a paper about SpecTRM-RL and examining a simple example specification. In fact, the specifiers (who were all aerospace engineering undergraduate and graduate students) worked under a handicap as no user documentation or manuals were available so they had to rely on the example specification (which was trivial compared to the specification they were tasked to build).

We estimate the model took approximately 8 person months to create. This result is encouraging considering the lack of experience or knowledge about either SpecTRM-RL or the MD-11 flight management system and the lack of sophisticated tools to assist in the development. We believe this time could be significantly reduced given appropriate tools.

We found that using SpecTRM-RL to build the model was an easier task than our previous specification of TCAS-II using RSML. This result is not surprising as lessons learned while building the TCAS specification were incorporated into the design of SpecTRM-RL. However, despite the improvements of SpecTRM-RL, cognitive manageability remained a concern throughout the specification task.

To assist with this manageability, each of the specifiers independently discovered the usefulness of starting the specification process by creating macros. We have concluded that for very complex models (such as a flight management system), macros are almost a requirement if humans are to be able to handle the complexity involved in constructing the specification. Basically, there needs to be some way of organizing the information into chunks in order to build a formal specification of such a complex system. Most example formal specifications we have seen abstract much of this complexity away, but a complete specification requires that this information be present. We found that other types of standard information organizing tools are also necessary, such as a data dictionary.

Cognitive manageability was also a concern when modifying the specification. It was often very difficult to consider and evaluate the ramifications of a potential change. Verifying the correctness of our model was also a painstaking experience. The model was easy to read and (when looked at in reasonable chunks) understand, but maintaining an accurate mental model of the *entire* system proved to be a challenging, if not impossible, task. Of course this problem is not confined to formal specifications---it is an even worse problem with large English language specifications. In fact, we believe the SpecTRM-RL specification is much easier to review and use than the English specification we used to get the information to create it. But tools are needed to assist humans in dealing with the complexity of any complete specification. This specification effort has provided us with ideas for such tools, which we plan to develop and use in further experimentation.

We did have some simple tools, and we found them very useful. For example, our high-level, graphical view of the specification helped keep the overall structure in mind, and the fact that SpecTRM-RL is executable was useful in error-checking the evolving specification. A consistency and completeness tool identified input conditions that were either not accounted for in our specification or led to an ambiguous system response (i.e., nondeterministic behavior).

Statecharts-like languages, such as UML and RSML, use internal-broadcast events to synchronize and order behavior. We found in our TCAS-II project that nearly all errors found in our final specification were related to internal events and that reviewing event-synchronized specifications was extremely difficult and error-prone. SpecTRM-RL does not use events to denote ordering---instead the ordering of transitions is specified directly---and we found that this greatly simplified the task of creating the FMS specification compared to our TCAS-II specification.

We also found that the hierarchical and modular structure of SpecTRM-RL was conducive to dividing the specification into smaller components and specifying these separately. Any usable formal specification language will need this feature.

Based on this experience, a major area of our future work will be on the visualization of formal specifications. The graphical model conveys a minimal amount of information, but more sophisticated visualizations would have been helpful. We plan to experiment with different visualization tools that will present meaningful information in manageable pieces to help the human user develop a mental model in as much detail as desired. Based on our experiences with specifying the Vertical Flight Control system, we believe that the use of visualization to assist with cognitive manageability is the key to making formal methods practical for large-scale specifications, and thus more attractive to industry. This direction of research is not tied to formal methods alone, but has applications to informal specification development as well. The task of simply understanding a large system is daunting and can be aided by visualization tools, regardless of whether formal analysis of a specification is desired.

4 Conclusions

SpecTRM-RL is our latest experimental specification language to assist us in learning more about making formal specification practical. In the work described here, we created a formal specification of a very complex but real system to learn more about how to create such languages. Some of the features that we had previously introduced proved useful. For example, while some formal specification languages only include symbolic or tabular notations, the ability in SpecTRM-RL to create and hierarchically decompose a graphical model of the system proved to be critical. We cannot imagine how cognitive manageability could be achieved using specification languages that do not provide this feature.

In addition, the substitution of tabular and other formats for propositional logic and other mathematical notations was critical not only in reading the specification but in creating it. We have found that some specification language features are also very helpful in detecting important omissions and other specification errors: These are described in more detail elsewhere [2]. Not using internally broadcast events also seemed to simplify the effort involved in creating and understanding a complex specification.

We are taking what we learned from this effort and developing more experimental tools to evaluate new ideas. A major area of future effort will be on visualization and tools to assist in organizing the information used in developing a specification. We are also working on various types of analysis tools.

References

[1] Leveson, N.G. Intent specifications: An approach to building human-centered specifications. *IEEE Trans. on Software Engineering*, January 2000.

[2] Leveson, N.G. Completeness in Formal Specification Language Design for Process-Control Systems. *Proceedings ACM Formal Methods in Software Practice*, Portland, Oregon August 2000.

[3] Rodriguez, M. et al. Identifying Mode Confusion Potential in Software Design. To appear in the *Proceedings of the 19th Digital Avionics and Systems Conference*. Philadelphia, PA, October 7-13.