Reduction and Slicing of Hierarchical State Machines*

Mats P.E. Heimdahl and Michael W. Whalen

University of Minnesota, Institute of Technology
Department of Computer Science, 4-192 EE/CS Bldg.
Minneapolis, MN 55455

Abstract. Formal specification languages are often criticized for being difficult to understand, difficult to use, and unacceptable by software practitioners. Notations based on state machines, such as, Statecharts, Requirements State Machine Language (RSML), and SCR, are suitable for modeling of embedded systems and eliminate many of the main drawbacks of formal specification languages. Although a specification language can help eliminate accidental complexity, the inherent complexity of many of today's systems inevitably leads to large and complex specifications. Thus, there is a need for mechanisms to simplify a formal specification and present information to analysts and reviewers in digestible chunks.

In this paper, we present a two tiered approach to slicing (or simplification) of hierarchical finite state machines. We allow an analyst to simplify a specification based on a *scenario*. The remaining behavior, called an *interpretation* of the specification, can then be sliced to extract the information effecting selected variables and transitions.

To evaluate the effectiveness and utility of slicing in hierarchical state machines, we have implemented a prototype tool and applied our slicing approach to parts of a specification of a large avionics system called TCAS II (Traffic alert and Collision Avoidance System II).

1 Introduction

Formal specification languages are often criticized for being difficult to understand, difficult to use, and unacceptable by software practitioners. Notations based on state machines, such as, Statecharts [2–4], Requirements State Machine Language (RSML) [12, 13], and SCR [8–10], are suitable for modeling of embedded systems and eliminate many of the main drawbacks of formal specification languages. State-based languages are based on familiar concepts, have intuitive syntax and semantics, and help in reducing the perceived complexity of a formal specification. A suitable language syntax and semantics alone, however, cannot overcome the problems caused by inherent system complexity. Although a specification language can help eliminate accidental complexity, the inherent complexity of many of today's systems inevitably leads to large and complex

* This work has been partially supported by NSF grants CCR-9624324 and CCR-9615088, and University of Minnesota Grant in Aid of Research 1003-521-5965.

specifications. Thus, there is a need for mechanisms to simplify a formal specification and present information to analysts and reviewers in digestible chunks.

In this paper, we present a two tiered approach to slicing (or simplification) of hierarchical finite state machines. We allow an analyst to simplify a specification based on a *scenario*. The remaining behavior, called an *interpretation* of the specification, can then be sliced to extract the information effecting selected variables and transitions. The simplified model has the same behavior as the original specification for the reduced input domain defined by the scenario. The objective is for the reduced specification to be significantly smaller than the original specification and to help analysts and domain experts to understand and validate the model.

To evaluate the effectiveness and utility of slicing in hierarchical state machines, we have implemented a prototype tool and applied our slicing approach to a specification of a large avionics system called TCAS II (Traffic alert and Collision Avoidance System II). We used our slicing tool to help clarify a set of questions we have asked ourselves during previous investigations. In this case study, the reduction results were very helpful and slicing of RSML specifications seems to have great potential.

1.1 Background

In a previous investigation, the Irvine Safety Research Group, under the leadership of Dr. Nancy Leveson, developed a requirements specification language called the Requirements State Machine Language (RSML) suitable for the specification of safety-critical embedded control systems [12, 13]. To make RSML suitable as a requirements specification language usable by all stake-holders in a specification effort, the syntax and semantics of RSML were developed with readability, understandability, and ease of use in mind. The usefulness of the language was demonstrated through the successful development of a requirements specification for a large commercial avionics system called TCAS II (Traffic alert and Collision Avoidance System II) [12, 13]. Furthermore, we have developed a collection of automated analysis procedures that check an RSML specification for desirable properties such as completeness, consistency, and determinism [7] and we have explored the possibility of provably correct code generation from RSML specifications [11].

However, even if requirements specifications are readable, understandable, and can be shown to be complete and consistent, the sheer size and complexity of many systems make the specifications difficult to understand and review. For example, a table in an SCR specification typically spans multiple pages and table sizes of 14 pages or more are not uncommon [8]. In RSML, tables are used differently and the table sizes are kept much smaller so that tables always fit on one single page [13]. Nevertheless, a specification for a complex system will, due to inherent system complexity, inevitably grow large and through its size hinder readability. Since manual inspection is an effective way of validating a specification to the customers real needs, readability, understandability, and clarity of a specification document are of utmost importance. Therefore, tools and techniques to help reduce the complexity of a large specification are needed.

To aid in the review and inspection of specifications based on hierarchical state machines we are currently investigating the feasibility of *specification slicing*

as a tool for specification understanding. We have focused on hierarchical state machines since they, in our opinion, are the most viable formalism for the class of systems we are interested in, namely reactive embedded control systems.

1.2 Program Slicing

Weiser introduced the concept of slicing as a means of simplifying programs to aid in debugging and identification of program fragments suitable for parallel execution [17]. A program slice is a projection of a program, which is smaller and potentially more comprehensible than the original program. Traditionally, program slicing is based on variables and statements. A slice consists of the statements that potentially effect the value of a particular variable at a given statement. Today, program slicing is used to reduce the complexity of a program and is successfully used in debugging, program comparison, testing, and maintenance.

Formal specifications provide a concise, mathematically well defined description that details the intended behavior of a system. Yet formal specifications often contain so much information that they overwhelm a reader and make the specifications less useful. Oda and Akari [14], and Chang and Richardson [1] have extended slicing to formal specifications expressed in Z. Their techniques were designed to help alleviate the readability problems in Z specifications. Both techniques are based on a traditional definition of slicing and calculates slices based on the use of a variable in a Z schema post-condition.

Slicing high-level specifications is in some aspects quite different from slicing programs. First, in state-based models variables are not the only entity we want to use as a basis for calculating a slice. Meaningful state-based specifications without variables are quite common during the early stages of the specifications effort. For example, if the focus of the specification effort is on the modal aspects of a system, input and output variables may not be defined until later in a project. In this case, the transitions between states are the focus of attention and we should be able to construct a slice containing the parts of the specification that effect a specific transition (or set of transitions).

Second, in the early stages of development, specifications are often incomplete and inconsistent. For example, the events and actions in a state-based specification may not be fully defined, and the specification may be internally inconsistent. Thus, a slicing approach must be able to work with incomplete and inconsistent models.

Sloane and Holdsworth extended the concept of slicing to a generalized marking of a program's abstract syntax tree [15]. This generalization allows (1) slicing of programs without statements and variables, and (2) slicing based on criteria other than the use of a variable at a given statement. Their approach enables slicing based on, for example, call graphs, object structure, and type dependencies. We also base our approach to reduction and slicing of hierarchical state machines on a marking of the abstract syntax tree. Therefore, it is similar to Sloane's and Holdsworth's approach.

1.3 Motivation

When reviewing and or inspecting a specification, we are interested in answering questions about the behavior of the specification. For example, during our work

with TCAS II, we asked questions such as "When do we downgrade a threat that has stopped reporting altitude?" and "How do we treat an intruder that is considered to be on the ground?".

Most questions regarding a specification involve an action, as in, downgrading an intruder that is considered to be a threat, and a specific scenario, such as, when the intruder has stopped reporting altitude. To help answer questions of this type we provide a two tiered approach to specification reduction.

First, we allow the analyst to reduce an RSML specification based on the specific scenario of interest. Our tool accepts a *reduction scenario* that is used to reduce the specification to contain only the behaviors that are possible when the operating conditions defining the reduction scenario are satisfied. We call such a reduced specification the *interpretation* of the specification under this scenario.

Second, we allow the analyst to slice the interpretation based on different entities in the model to highlight, for example, the portions of the specification effecting the value of an output variable or the information effecting whether a specific transition can be taken.

In this paper we report on our first experiences with reduction and slicing of RSML specifications. To evaluate the feasibility of state machine slicing and to get some early feedback on our approach, we have applied our slicing technique to parts of the TCAS II requirements specification. Initial results show that the slicing approach is very useful and would have helped us answer a set of questions we encountered in previous investigations. Naturally, more experimentation is needed, but these initial results are promising and show that slicing of state based specifications provides many benefits for specification readability and understandability.

The next section gives a brief description of the syntax and semantics of RSML. The testbed we have used in this work is introduced in Section 3. Section 4 describes our approach to slicing of RSML specifications and Section 5 discusses our experiences from a case study. Section 6 provides concluding remarks.

2 Requirements State Machine Language (RSML)

RSML was developed as a requirements specification language for embedded systems. The language is based on hierarchical finite state machines and is similar to David Harel's Statecharts[2, 5]. For example, RSML supports parallelism, hierarchies, and guarded transitions which originated in Statecharts (Figure 1).

One of the main design goals of RSML was readability and understandability by non computer professionals such as, in our case, pilots, air frame manufacturers, and FAA representatives. During the TCAS project, we discovered that the guarding conditions required to accurately capture the requirements were often complex. The prepositional logic notation traditionally used to define these conditions did not scale well to complex expressions and quickly became unreadable. To overcome this problem, we decided to use a tabular representation of disjunctive normal form (DNF) that we call AND/OR tables (see Figure 2 for an example from the TCAS II requirements). The far-left column of the AND/OR table lists the logical phrases. Each of the other columns is a conjunction of those phrases and contains the logical values of the expressions. If one of the columns is

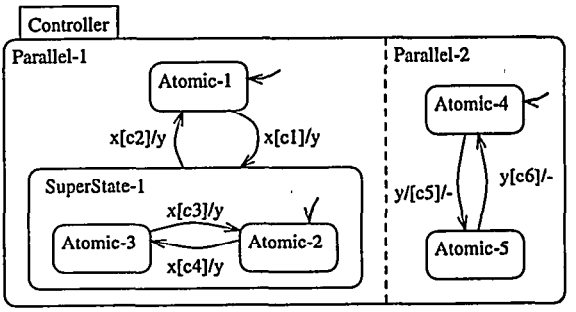

Fig. 1. An example of an hierarchical state machine.

Transition(s): Potential-Threat ⟶ Other-Traffic

Location: Other-Aircraft ▷ Intruder-Status$_{s\text{-}136}$

Trigger Event: Air-Status-Evaluated-Event$_{e\text{-}279}$

Condition:

		OR										
	Alt-Reporting$_{s\text{-}101}$ **in state Lost**	T	T	·	·	T	T	·	·	·	·
	RA-Mode-Cancelled$_{m\text{-}218}$	·	·	T	T	·	·	T	T	·	·
	Alt-Reporting$_{s\text{-}101}$ **in state No**	·	·	T	T	·	·	T	T	·	·
	Other-Bearing-Valid$_{v\text{-}130}$	F	·	F	·	F	·	F	·	·	·
AND	Other-Range-Valid$_{v\text{-}117}$ = True	·	F	·	F	·	F	·	F	·	·
	Potential-Threat-Range-Test$_{m\text{-}214}$	T	T	T	T	F	F	F	F	·	·
	Potential-Threat-Condition$_{m\text{-}213}$	·	·	·	·	·	·	·	·	F	·
	Proximate-Traffic-Condition$_{m\text{-}216}$	·	·	·	·	T	T	T	T	F	·
	Threat-Condition$_{m\text{-}224}$	·	·	·	·	·	·	·	·	F	·
	Other-Air-Status$_{s\text{-}101}$ **in state On-Ground**	·	·	·	·	·	·	·	·	·	T

Output Action: Intruder-Status-Evaluated-Event$_{e\text{-}279}$

Fig. 2. A transition definition from TCAS II with the guarding condition expressed as an AND/OR table.

true, then the table evaluates to true. A column evaluates to true if all of its elements are true. A dot denotes "don't care." To further increase the readability of the specification, we introduced many other syntactic conventions in RSML. For example, we allow expressions used in the predicates to be defined as mathematical functions (e.g., Other-Tracked-Relative-Alt-Rate$_{f\text{-}246}$), and familiar and frequently used conditions to be defined as macros (e.g., Threat-Condition$_{m\text{-}224}$)[1]. A macro is simply a named AND/OR table defined elsewhere in the document. Naturally, the state machine in a real system is never as simple as in Figure 1. As an example of a realistic model, a part of the state machine modeling an intruding aircraft in TCAS II can be seen in Figure 4.

Formally, the behavior of a finite-state machine can be defined using a next-state relation. In RSML, this relation is modeled by the transitions between states and the sequencing of events. Thus, one can view a graphical RSML specification as the definition of the mathematical next-state relation F. In short, an RSML specification is a mapping from a set of states (called the set of all configurations – $Config$) representing the states in the graphical model and a set of variables (V) representing the input and output variables in the model to new states and variables. Thus, the next state relation F is a mapping $C \mapsto C$, where $C \subseteq (Config \times V)$. For a rigorous treatment of formal foundation of RSML the reader is referred to [7]. A detailed description of the graphical notation can be found in [13].

3 Testbed Specification

To evaluate the effectiveness of our approach and to better understand the effect of slicing on a large real world RSML specification, we applied our tool to the TCAS II RSML model. To introduce the reader to our case study, we provide a short overview of TCAS II.

3.1 TCAS II

TCAS is a family of airborne devices that function independently of the ground-based air traffic control (ATC) system to provide collision avoidance protection for commercial aircraft and larger aircraft. TCAS II provides traffic advisories and recommended escape maneuvers (resolution advisories) in a vertical direction to avoid conflicting aircraft.

In this paper, we will use examples from the part of the collision avoidance system (CAS) in TCAS II that classifies intruding aircraft as Other-Traffic, Proximate-Traffic, Potential-Threats, or Threats. In the CAS logic, the states of two main components are modeled: our own aircraft and other aircraft.

Own-Aircraft: Figure 3 shows the expanded Own-Aircraft portion of the CAS model. Effective-SL (sensitivity level) controls the dimensions of the protected airspace around own aircraft.

[1] The subscript is used to indicate the type of an identifier (f for functions, m for macros, and v for variables) and gives the page in the TCAS II requirements document where the identifier is defined.

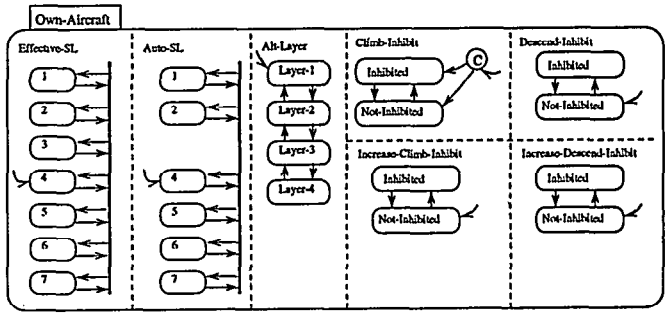

Fig. 3. Model of Own-Aircraft

There are two primary means that CAS uses to determine Effective-SL: ground-based selection and pilot selection. When the pilot selects an automatic sensitivity selection mode, CAS selects sensitivity level based on the current altitude of own aircraft (defined in the Auto-SL state machine).

Alt-Layer effectively divides vertical airspace into layers (e.g., Layer-3 is approximately equal to the range 20,000 feet to 30,000 feet). Alt-Layer and Effective-SL are used in the determination other aircraft threat classification (see Figure 4).

Other-Aircraft: The model of an intruding aircraft can be seen in Figure 4. In short, the top-level state machine reflects whether a particular Other-Aircraft is currently being tracked or not.

The Intruder-Status state within Tracked reflects the current classification of Other-Aircraft (Other-Traffic, Proximate-Traffic, Potential-Threat, and Threat). When an intruder is classified as a threat, a two-step process is used to select a Resolution Advisory (RA). The first step is to select a sense (Climb or Descend). The CAS logic computes the predicted vertical separation for both climb and descend maneuvers, and selects the sense that provides the greater vertical separation.

The second step in selecting an RA is to select the strength of the advisory. The least disruptive vertical rate maneuver that will still achieve safe separation is selected. For a more complete description of TCAS II and how it was modeled using RSML the reader is referred to [13].

4 Specification Slicing

During our work with static analysis [6, 7] we identified questions regarding the behavior of TCAS II where some tool support to aid in answering the questions would have been helpful. The questions were seldom related to output variables. Instead, we wanted to know how the specification behaved when, for example, an intruding aircraft was declared to be on the ground or when an intruding aircraft stopped reporting altitude. Thus, our slicing approach was defined to help an analyst answer questions that we know are common from experience. Typical questions we encountered can be seen in Table 1.

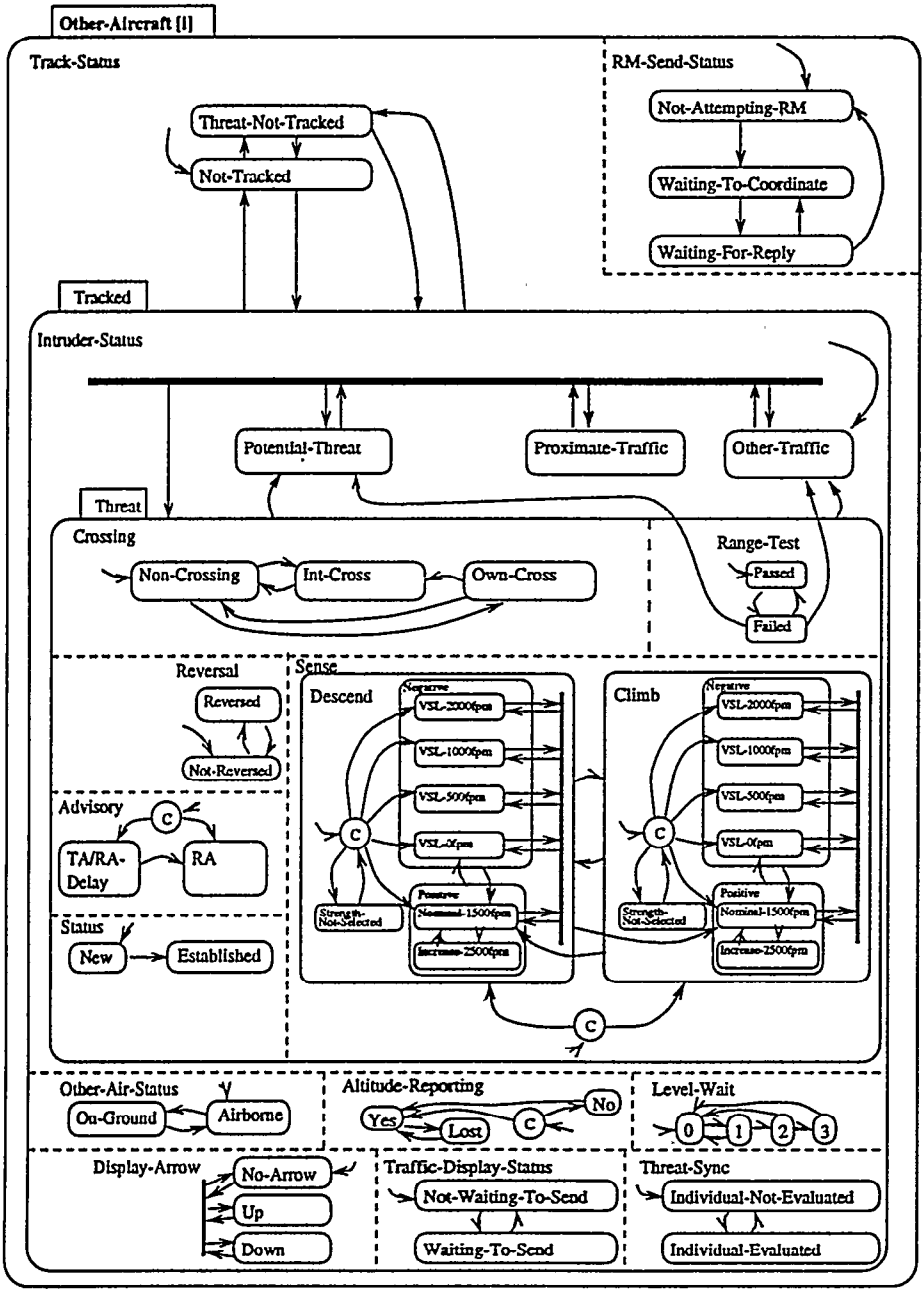

Fig. 4. Model of an intruding aircraft

1. *In Intruder-Status, how does the threat classification logic work for an intruder that reports both valid range and valid bearing?*
2. *How do we classify an intruder that has stopped reporting altitude?*
3. *What happens with a threat that lands and is determined to be on the ground?*

Table 1. Typical questions encountered while reviewing TCAS II.

To aid the analyst in answering questions such as question 1 in Table 1 we need to construct several different slices of the specification. First, we are only interested in the behavior of the system when both bearing and range are valid – we need to eliminate all information that does not pertain to this particular situation. Second, we are only interested in the transitions between the high-level states in Intruder-Status (Figure 4)[2] – we need to eliminate information that cannot effect the truth value of the guarding conditions on the transitions between these states. Finally, we are interested in the sequence of events that can cause these transitions to be taken - we need to eliminate all transitions that can not contribute to this sequence of events.

The following sections illustrate how these slices are constructed and how they are combined.

4.1 The Interpretation Under Scenario *s*

An *interpretation* of an RSML specification under a *scenario* s is a domain restriction (defined by the scenario) of the next state relation. In Section 2 we defined an RSML specification to be a relation F mapping $C \mapsto C$. By restricting the domain to only the states that satisfy the conditions that define a scenario we can produce a simpler mapping that is concerned with only the behaviors that are possible in this scenario. Formally, an interpretation of an RSML specification is defined as the relation R where

$$R \equiv \{c \mid c \in C \ \wedge \ s(c)\} \lhd F$$

Informally, the interpretation R consists of all behaviors that are possible given the restrictions imposed by the scenario s[3].

Interpretations can be used to reduce (or in some sense slice) an RSML specification to only show the behaviors that are possible under the specific conditions we are interested in. For example, the scenario where an intruder has stopped reporting altitude can be formalized (using AND/OR table notation) as in Figure 5 and the scenario where an intruder is providing reliable tracking data, that is, both the bearing and altitude are considered valid, can be seen in Figure 6.

Scenarios can be used to reduce an RSML specification to produce a simplified model. Consider the transition definition in Figure 2. This transition determines under which conditions an intruding aircraft can be downgraded from a

[2] The states involved in threat classification are Other-traffic, Proximate-Traffic, Potential-Threat, and Threat.

[3] The notation $S \lhd R$ is borrowed from Z [16] and defines a relation that relates a to b iff R relates a to b and a is a member of S.

Reduction Scenario: Not-Reporting-Altitude

AND	Alt-Reporting$_{s\text{-}101}$ in state No	T

Fig. 5. An intruder has stopped reporting altitude expressed as an AND/OR table.

Reduction Scenario: Valid-Tracking

AND	Other-Bearing-Valid$_{v\text{-}130}$ = Valid	T
	Other-Range-Valid$_{v\text{-}133}$ = Valid	T

Fig. 6. An intruder reporting reliable tracking data expressed as an AND/OR table.

Transition(s): Potential-Threat ⟶ Other-Traffic

Location: Other-Aircraft ▷ Intruder-Status$_{s\text{-}136}$

Trigger Event: Air-Status-Evaluated-Event$_{e\text{-}279}$

Condition:

				OR						
AND	RA-Mode-Cancelled$_{m\text{-}218}$	T	T	T	T	·	·
	Alt-Reporting$_{s\text{-}101}$ in state No	T	T	T	T	·	·
	Other-Bearing-Valid$_{v\text{-}130}$	F	·	F	·	·	·
	Other-Range-Valid$_{v\text{-}117}$ = True	·	F	·	F	·	·
	Potential-Threat-Range-Test$_{m\text{-}214}$	T	T	F	F	·	·
	Potential-Threat-Condition$_{m\text{-}213}$	·	·	·	·	F	·
	Proximate-Traffic-Condition$_{m\text{-}216}$	·	·	T	T	F	·
	Threat-Condition$_{m\text{-}224}$	·	·	·	·	F	·
	Other-Air-Status$_{s\text{-}101}$ in state On-Ground	·	·	·	·	·	T

Output Action: Intruder-Status-Evaluated-Event$_{e\text{-}279}$

Fig. 7. The transition definition sliced based on the reduction scenario Not-Reporting-Altitude in Figure 5

Transition(s): Potential-Threat ⟶ Other-Traffic

Location: Other-Aircraft ▷ Intruder-Status$_{s\text{-}136}$

Trigger Event: Air-Status-Evaluated-Event$_{e\text{-}279}$

Condition:

		OR	
AND	Potential-Threat-Condition$_{m\text{-}213}$	F	·
	Proximate-Traffic-Condition$_{m\text{-}216}$	F	·
	Threat-Condition$_{m\text{-}224}$	F	·
	Other-Air-Status$_{s\text{-}101}$ **in state** On-Ground	·	T

Output Action: Intruder-Status-Evaluated-Event$_{e\text{-}279}$

Fig. 8. The transition definition sliced based on the scenario Valid-Tracking in Figure 6.

Potential-Threat (indicating that an intruder is close and that a traffic advisory should be issued to the pilot) to Other-Traffic (indicating that an intruder is considered to be irrelevant and no information about the intruder is presented to the pilot). Under normal circumstances, a potential threat is only downgraded to other traffic if it is not considered to be a potential threat, nor a threat or in proximity (captured in column 9 of Figure 2). However, there are many exceptions for abnormal operating conditions, for example, when an intruder stops reporting altitude. These exceptions make the threat detection logic quite complex and obfuscates the specification. If we construct an interpretation of the specification based on the scenario named Not-Reporting-Altitude in Figure 5, we get a simpler transition definition (Figure 7) telling us how TCAS downgrades a non-altitude reporting intruder. The scenario Not-Reporting-Altitude requires the state machine Alt-Reporting to be in state No. Since the state machine Alt-Reporting (Figure 4) by definition can only be in one state at the time, all columns in Figure 2 requiring Alt-Reporting to be in a state other than No can be eliminated (columns 1, 2, 5, and 6 can be eliminated).

If we are only interested in the normal operating condition where we have reliable tracking data from an intruding aircraft, we can construct an interpretation based on the scenario named Valid-Tracking in Figure 6. This interpretation almost eliminates the guarding condition (Figure 8) and we can more clearly see how TCAS operates under normal conditions.

Construction of Interpretations: As mentioned in Section 4, our slicing algorithms are based on a marking of the abstract syntax tree. In a previous investigation we developed an RSML parser as a part of an analysis environment for RSML [7]. This parser has been modified to allow us to mark the abstract syntax tree based on various slicing criteria.

A reduction scenario is used to mark the infeasible columns in each AND/OR table. A column is infeasible if any of the truth values in the column (recall that a column represents a conjunction) contradicts the scenario. We have implemented a collection of decision procedures to determine if the predicates constituting a

column contradict a scenario. The current decision procedures cover predicates expressed over enumerated variables and over the states in the model. The decision procedures do not cover predicates over integer and real variables. These limitations are discussed in more detail below.

After the infeasible columns have been marked they are removed from the table. Furthermore, after the columns have been removed, any rows consisting of all don't care are removed since those rows are now superfluous. If a table is left with no remaining columns, the guarding condition defined by the table cannot be satisfied in this scenario. After all tables have been reduced, the transitions with unsatisfiable guarding conditions are eliminated from the model. The remaining specification constitutes the interpretation of the specification under the scenario.

Limitation: Our current implementation of the reduction algorithm only allows the definition of the scenarios to contain predicates over enumerated variables and over the states in the specification. Since simplification of a table involves determining if two predicates contradict each other, we have limited our approach to predicates where the decision procedures are relatively easy to implement. We are currently extending the decision procedures to handle a wider range of predicates. However, extending the algorithm to handle the full range of predicates in the RSML syntax is not possible since the language allows both linear and non-linear arithmetic in predicate definitions. The interested reader is referred to [7] for a detailed discussion about how the expressive power of RSML effects static analysis.

4.2 Data Flow Slices

To help answering the questions in Table 1, it is not enough to construct the interpretation of the specification under a reduction scenario. In general, few transitions are completely eliminated in an interpretation, most transitions are still satisfiable and cannot be removed from the model. Thus, in addition to constructing an interpretation, we are interested in knowing what parts of the state machine that effect a particular transition or variable we are interested in. We need to construct a slice based on the *data dependency* of the guarding condition on the transition.

To discover the data dependencies for a guarding condition, we perform a graph traversal of an RSML specification. A partial version of the static data dependency graph for the elements of an RSML specification can be seen in Figure 9. The solid lines represent data dependencies and the dotted lines represent a parent child (is-a) relationship between the elements. To discover the data dependencies for a given node in the graph, we traverse all of the nodes below it in the data dependency graph. For example, a transition is data-dependent on its guarding condition, which in its turn is dependent on an AND/OR table. Eventually, all aspects of the system are dependent on the system's current state, constants, and variable values (the gray roundtangles).

To illustrate our approach, consider the set of transitions between the top-level states in Intruder-Status (Figure 4). We are interested in answering the first question in Table 1. Thus, we have first constructed an interpretation based

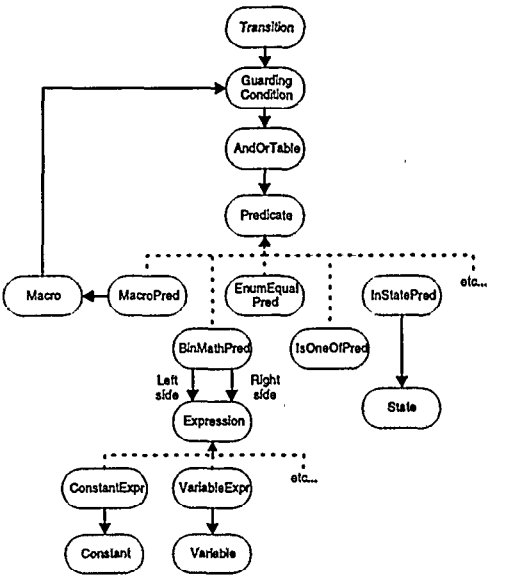

Fig. 9. Data dependencies

on the scenario in Figure 6. The transition between Potential-Threat and Other-Traffic (Figure 2) is in this interpretation reduced to the table in Figure 8. Many other transitions in the model exhibit similar reductions for the same scenario.

To construct the data-flow slice for this transition we simply mark all entities (states, macros, functions, variables, and constants) that can directly or indirectly effect the truth value of the reduced guarding condition. The unmarked entities in the specification cannot effect the behavior of the transition and can safely be removed from the specification.

4.3 Control Flow Slices

The slice constructed based on the data dependencies only shows us what information is needed to determine *if* a transition can be taken. It does not tell us *when* the transition can be taken. To determine this, we need to construct a slice based on the control flow in the specification. In RSML, the order in which state machines are evaluated is based on the events and actions on the transitions (see Section 2). Thus, to determine when a transition can be taken we need to construct a slice that shows all the parts of the specification that are involved in the generation of the trigger event on the transition. For example, the transition in our example is triggered by the Air-Status-Evaluated-Event (Figure 8) so all transitions with this event as an action (all transitions that can generate this event) must be included in our control slice. In this case, the event is generated by the transitions in the state machine Other-Air-Status. This process is now repeated for the transitions just added to the slice. The algorithm is terminated when we reach transitions that are triggered by the receipt of an input from an external source (the event is not generated from within the RSML model).

The construction of the control slices is also based on a simple marking of the abstract syntax tree.

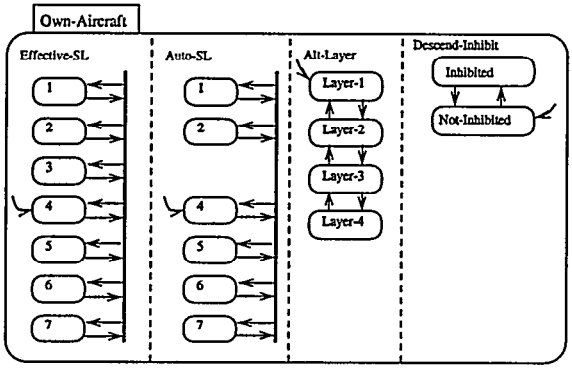

Fig. 10. Model of Own-Aircraft reduced

As mentioned above, a slice is identifies by the set of tagged entities in the abstract syntax tree. By using a different tag for each slice, it is trivial to combine slices using standard set operations (union, intersection, and set complement), For example, the combined slice needed to fully answer question 1 in Table 1 consists of the union of all data flow and control flow slices for all transitions between the high-level states in Intruder-Status (Figure 4). The states and transitions in this slice are shown in Figures 10 and 11.

In summary, our approach to slicing of hierarchical state machines allows an analyst to reduce a specification based on a scenario. We call such a reduced specification the interpretation of the specification under the scenario. The interpretation can then iteratively be sliced based on data-flow and control-flow information. The slices can be arbitrarily combined using standard set operations to construct a combined slice containing the information of interests.

5 Case Study

To evaluate the effectiveness of our approach and to better understand the effect on a large real world RSML specification, we applied our tool to the most complex part of the TCAS II RSML model. This section discusses some metrics we used to evaluate the reduction capability and discusses our experiences.

5.1 Evaluation Criteria

In traditional program slicing, the effectiveness of a slicing algorithm is easily evaluated by comparing, for example, the number of statements in the slice to the number of statements in the original program. In hierarchical state machines, however, there is no established way of evaluating a slice. In fact, there are no established metrics to measure the size and complexity of a state machine.

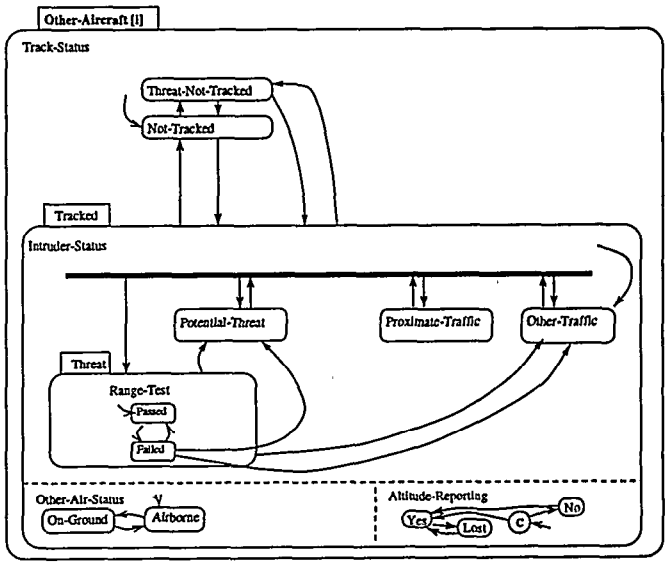

Fig. 11. Model of an intruding aircraft

Measures such as number of reachable states, number of named states, or number of transitions are reasonable, but the correlation between these metrics and the difficulty of understanding a specification is unknown. Nevertheless, in an attempt to make a reasonably objective evaluation of the effectiveness of our approach, we use some metrics to measure the reduction of the specification. We have chosen to measure number of transitions, perceived table size, and effective table size.

Number of transitions: The number of transitions in the model is easily counted and the metric is a reasonable and intuitive measure of the difficulty of understanding a model.

Perceived table size: The perceived size of a table is defined to be the table height (the number of rows in the table) times the table width (the number of columns). This metric indicates the complexity of a single table shown on one page. Naturally, to fully understand a table one may have to trace macros through several layers of indirection and this added complexity is captured in our third metric.

Effective table size: The use of macros reduces the perceived size of a table since much of the complexity of the guarding condition is hidden in the macros. The complexity added through macro indirection is captured by the effective table size. The effective size of a table is defined to be the perceived size of the table with all macro references recursively expanded.

As an absolute measure of the complexity of a state machine these metrics may have little value, there is no evidence that a state machine with 200 transitions is harder to understand than one with 100 transitions. Nevertheless, as measure of the *relative complexity* between the original specification and the reduced model

produced by our reduction tool the metrics make intuitive sense. Intuitively, if a model with 40 transitions is reduced to a model with 20 transitions, it is easier to understand and review.

5.2 Reduction Results

We applied our tool to the transitions defining the behavior of state machine Tracked in Figure 4. We used reduction scenarios based on questions (for example, the questions in Table 1) encountered during previous investigations related to TCAS II. Thus, we believe the reduction scenarios are representative of scenarios that would be used during reviews and inspections of an RSML model. This section summarizes the main observations from this limited case study.

Table size: The reduction scenarios we used provided significant reductions in the tables effected by the criteria. A typical example is the reductions of the transition from Potential-Threat to Other-Traffic discussed in Section 4.

Most of our reduction criteria were related to the classification of an intruding aircraft as a Threat, Potential-Threat, Proximate-Traffic, or Other-Traffic. The perceived size of the conditions guarding these transitions ranged from 1 to 80^4 before reduction and ranged from 0 to 40 after reduction. Subjectively, these reductions helped clarify some issues regarding threat classification (as illustrated by the reductions of the transition in Figure 2).

The effective size of the transitions involved in threat classification ranged from approximately 10^8 to 10^{10} when all macro references were expanded. The effective size after reduction ranged from 0 to 10^8 based on the same set of reduction criteria as in the previous paragraph. These numbers led to two observations. First, we were surprised when we got an indication of the true complexity of the guarding conditions in the threat detection logic. This complexity is effectively hidden by the macro abstractions in RSML. Some guarding conditions involved more that 100 predicates and would be nearly impossible to capture without the use of macros.

Second, the effective size metric does not tell us much about the readability of a real-world specification—all the metric indicates is that this model is very large. Focusing on the size of individual tables seems to be a better gauge of readability, small tables seem to be easy to understand regardless of how many macros and levels of indirection are used. Nevertheless, to accurately determine the effect of slicing on the readability of RSML specifications we need to conduct controlled readability studies. Such studies are in the planning stages.

Transitions: When constructing an interpretation, the number of transitions in the model was not effected to the extent we had hoped. Some interpretations produced useful elimination of transitions, for example, all transitions upgrading an intruding aircraft to a threat are unsatisfiable if the intruder is declared to be on the ground, but as a whole most transitions were still satisfiable under the reduction scenarios we applied. On the other hand, after an analyst has selected a subset of the transitions for closer study, the data flow slices and control flow slices eliminated large unrelated parts of the specification. The simplification is

[4] 8 columns by 10 rows.

clearly illustrated in the difference between Figure 4 and Figure 11. Nevertheless, all the questions we had compiled were related to the threat detection encapsulated in Intruder-Status. Thus, all data flow and control flow slices produced similar results. We are currently in the process of slicing TCAS II based on a multitude of data-flow and control-flow criteria, for example, slices based on the data dependencies for all output variables and all transitions. We are collecting data and we hope to be able to present an empirical study of the reduction capabilities in a large real world application shortly.

6 Conclusion

In this paper we described an approach to reducing an RSML specification based on a reduction scenario in conjunction with traditional program slicing techniques. Our approach is two tiered. First, we allow an analyst to simplify a specification based on a *scenario*. As scenario is some operating condition the under which we are interested in inspecting or reviewing a specification. Second, the remaining specification, called an *interpretation* of the specification, is sliced based on data-flow and control-flow information to extract the parts of the specification effecting selected variables and transitions.

To evaluate the effectiveness of the approach, we implemented a prototype tool and applied it to a specification for a large avionics system called TCAS II. We used the tool to slice the specification to help us answer a set of questions we have asked ourselves during previous investigations. In this case study, the reduction results were very helpful and the approach seems to have great potential. The reduction scenarios provided significant reductions in the tables relevant to answer the question and the data-flow and control-flow slices eliminated large parts of, for these questions, irrelevant information.

Although the slices helped clarify and simplify the model, we have not yet collected sufficient empirical data regarding the reduction capabilities in hierarchical state machines. The metrics we used in the case study are inadequate. The perceived size of a table is a very simplistic metric and the effective size of a table (the size of a table with all macro references expanded) only told us that guarding conditions are very complex, but did not aid much in the evaluation of the reduction capability of the slicing approach. New metrics and controlled readability studies are needed to objectively evaluate the effect of slicing in hierarchical state machines.

Finally, the approach to specification slicing outlined in this paper is not limited to RSML specifications. The approach is general enough to apply to all languages based on state machines using guarded transitions. From our experiences in this investigation, we are convinced that specification slicing holds great potential in areas such as requirements development, requirements inspections, and visual requirements verification. Specification languages such as SCR [8] would benefit from the same simplification mechanisms and we strongly encourage tool developers to include such a mechanism in future versions of their tools.

References

1. J. Chang and D.J. Richardson. Static and dynamic specification slicing. In *Proceedings of the Fourth Irvine Software Symposium*, April 1994.

2. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, .8:231–274, 1987.
3. D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.
4. D. Harel and A. Naamad. The STATEMATE semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, vol-5(4):293–333, October 1996.
5. D. Harel and A. Pnueli. On the development of reactive systems. In K.R. Apt, editor, *Logics and Models of Concurrent Systems*, pages 477–498. Springer-Verlag, 1985.
6. M. P.E. Heimdahl and N.G. Leveson. Completeness and Consistency Analysis of State-Based Requirements. In *Proceedings of the 17th International Conference on Software Engineering*, pages 3–14, April 1995.
7. M. P.E. Heimdahl and N.G. Leveson. Completeness and Consistency Analysis of State-Based Requirements. *IEEE Transactions on Software Engineering*, TSE-22(6):363–377, June 1996.
8. C. L. Heitmeyer, , R.D. Jeffords, and B. L. Labaw. Consistency checking of SCR-style requirements specifications. *ACM Transactions on Software Engineering and Methodology*, vol-5(3):231–261, July 1996.
9. C. L. Heitmeyer, B. L. Labaw, and D. Kiskis. Consistency checking of SCR-style requirements specifications. In *Proceedings of the International Symposium on Requirements Engineering*, March 1995.
10. K. L. Heninger. Specifying software for complex systems: New techniques and their application. *IEEE Transactions on Software Engineering*, 6(1):2–13, January 1980.
11. D.J. Keenan and M.P.E. Heimdahl. Code generation from hierarchicl state machines. In *Proceedings of the International Symposium on Requirements Engineering*, 1997.
12. N. G. Leveson, M. P.E. Heimdahl, H. Hildreth, J. Reese, and R. Ortega. Experiences using Statecharts for a system requirements specification. In *Proceedings of the Sixth International Workshop on Software Specification and Design*, pages 31–41, 1991.
13. N. G. Leveson, M. P.E. Heimdahl, H. Hildreth, and J. D. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, 20(9):694–707, September 1994.
14. T. Oda and K. Araki. Specification slicing in formal methods of software engineering. In *Proceedings of the Seventeenth International Computer Software and Applications Conference*, November 1993.
15. A.M. Sloane and J. Holdsworth. Beyond traditional program slicing. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 180–186, January 1996.
16. J.M. Spivy. *The Z Notation: A Reference Manual.* Prentice-Hall, 1992.
17. M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, SE-10(4):352–357, July 1984.