

Formal Analysis of Domain Models

In Proc. *International Workshop on Requirements for High Assurance Systems (RHAS'02)*. Essen, Germany, September 9, 2002.

Ramesh Bharadwaj
 Center for High Assurance Computer Systems
 Naval Research Laboratory
 Washington DC 20375-5320, USA
 +1-202-767-7210
 ramesh@itd.nrl.navy.mil

ABSTRACT

Recently, there has been a great deal of interest in the application of formal methods, in particular precise formal notations and automatic analysis tools, for the creation and analysis of artifacts of requirements engineering. In this paper we discuss the role of formal methods in requirements engineering (RE), emphasizing that in contrast to their more conventional application in RE for the creation and analysis of *requirements specifications*, formal methods may be applied in a cost-effective way to answer specific questions about the domain by the construction and automated analysis of “domain models”.

Keywords

Formal Methods, Formal Models, Requirements Engineering, Formal Analysis, CASE Tools, Model Checking, Theorem Proving

1 INTRODUCTION

Formal methods, in particular automated techniques for finite-state verification such as model checking, are increasingly being used in industry for the analysis of descriptions of hardware and protocols. However, most practical system descriptions, notably those of software, are not amenable to finite-state verification methods since they have very large or infinite state spaces. To compound the problem, software developers have serious doubts about the scalability and cost-effectiveness of these methods.

Recently, there has been a great deal of interest in the application of formal methods, in particular precise formal notations and automatic analysis tools, for the creation and analysis of *requirements specifications*, i.e., mathematically precise descriptions of the required black-box behavior of a system. Creating a requirements specification, especially in a notation with an efficient operational semantics, offers

a number of advantages: (a) It creates a domain of discourse through which the stakeholders can agree upon the system’s required behavior. There are several *validation* tools that can help in this process, such as simulators, browsers, formatters, and automatic invariant generators. (b) Since a requirements specification is created at a high level of abstraction, and does not include irrelevant details of the implementation, it is more amenable to automated analysis such as model checking and automatic theorem proving using decision procedures. (c) A specification is a “build-to” document. A precise specification of the required behavior of a system not only helps developers when implementing the system, but also serves as the primary reference document for carrying out system integration, system acceptance tests, and system maintenance.

A major barrier to the construction of a precise formal specification of system behavior has so far been that developers find formal methods difficult to understand and apply. One exception is a formal method called SCR which, due to its relatively easy-to-use tabular notation and demonstrated scalability, has achieved a degree of success in industrial applications. Numerous pilot projects have demonstrated the ease-of-use and cost-effectiveness of the SCR approach for requirements specification and analysis.

In this paper we investigate another area where formal methods may be used in requirements engineering, that is, to answer specific questions about the domain. In contrast to the more conventional approach of constructing requirements specifications, we advocate here the creation and analysis of “domain models” just for the purpose of answering specific questions. The effort involved in creating such models is minimal and is comparable to the effort required to peruse a prose specification directly to find answers to those questions (which may often turn out to be incorrect [6]). With our approach, not only

is there the advantage of arriving at the right answer with mathematical certainty, but as an added bonus, the process uncovers anomalies and raises issues about the requirements that the conventional approach does not.

To demonstrate the advantages of this approach, we recently used the SCR method and toolset for the formal modeling and analysis of the “Climb FMS Speed Mode” of a simple Speed Mode indicator (SMI), which operates as part of a flight management system (FMS). We based our model on the description¹ provided in a paper by Zimmerman et al. [6]. In this paper, the authors present an experimental study intended to evaluate the readability of four different notations – textual, graphical, logical, and tabular² – commonly used in formal requirements specifications to express the conditions that trigger state transitions in the description. They do this by posing questions to a number of subjects, to assess their ability to read and understand descriptions of state transitions in the four notations. In the experiment we carried out, we used the SCR Toolset [4] to create a formal model of the domain based on the description provided in [6], and the Toolset’s automated analysis tools [1, 2] to arrive at answers to some of the questions in the original study. Other questions were answered by examining the SCR tables. The development of the domain model and its analysis was carried out in less than one afternoon (about three hours), which is comparable to the time taken by most of the subjects in the original study to directly answer the same questions after studying the four descriptions. During this process, we uncovered a number of issues with the descriptions in [6], a paper presented at a premier conference which has presumably been well scrutinized by referees during its review. It is interesting to note that of the four different notations in the paper, we found the textual description (a structured presentation of informal prose which is reproduced in Appendix A) to be most useful in constructing the SCR domain model.

A note on naming: Although the SCR method has explicit naming conventions [5] we chose not to adhere to them in order to maintain compatibility with the naming conventions of [6]. The only deviation was that each embedded blank or whitespace in the names was replaced by the underscore (“_”)

¹Although the authors call it a “requirements specification” we desist from using their terminology because the description is not of the required black box behavior of the flight management system.

²Different from the SCR tabular notation.

character. However, we did discover several anomalies with the naming in [6], which we shall discuss in the sequel.

2 THE SCR MODEL OF SMI

The SCR method is used to document the required system behavior, i.e., the relation between environmental quantities that the system monitors (called the *monitored quantities*) and environmental quantities that the system controls (called the *controlled quantities*). Associated with each SCR variable is a type. We began by determining (guessing) the types of all variables whose names appeared in the textual description of SMI, and classified and documented them in the type dictionary (see Figure 1). To determine the type of a variable, we took the union of all values that the variable was either compared with or assigned to, and, in certain cases (where we believed that the variable could take on other values), included an additional value “other”. In certain cases, if one value set (type) turned out to be a proper subset of another, we discarded the subset and associated the superset with the variable’s type.

Since the SCR model of the simple Speed Mode indicator only describes the behavior of Climb FMS Speed Mode, it has no controlled variables. Figure 2 lists the monitored variables of SMI and their associated types.

The SCR notation includes four constructs – mode classes, terms, conditions, and events. A *mode class* captures historical information that helps make the specifications concise. A mode class may be viewed as a state machine, whose states are called *modes* and whose transitions are triggered by events. The SCR model of SMI has a single mode class – `Climb_FMS_Speed_Mode` – of type `{Default, Max_Climb, Economy, Edit}`.

Issues raised when creating the type dictionary: Our major concern during this phase was that the types of variables had to be inferred by carefully examining the entire description since this information is not available in one place. During this process we detected several anomalies:

- The word “Edit” (by itself) and the word “Edit” followed by the word “Mode” appear in the textual description. We inferred that the word “Mode” was (erroneously) appended to the identifier “Edit” in the “formal” notations, whereas the word “Mode” is presumably a part of the occurring phrase in the textual description. For example, the phrase *Climb FMS Speed Mode previously in Edit Mode* is to be interpreted as “Climb

Type Identifier	Basetype
Climb_Speed_Mode_Requests	{ Economy, Edit, Max_Climb, other }
Engine_Out_Type	{ Not_Engaged, Engaged }
FCC_Engaged_Modes	{ Altitude_Hold_Speed, Altitude_Hold_Idle_Thrust, Altitude_Hold_Maximum_Thrust, other }
FCC_Speed_Mode_Requests	{ Economy, AFS_Speed, Edit_CAS, Edit_Mach, other }
FMS_Modes	{ Lateral_Only, Lateral-Vertical, other }
FMS_Speed_Modes	{ Max_Climb, Edit, other }
Flight_Phase_Type	{ Preflight, Takeoff, Climb, Cruise, Descent, Approach, Done }

Figure 1: Type Dictionary of SMI

Name	Type
Cruise_FMS_Speed_Mode	FMS_Speed_Modes
Descend_FMS_Speed_Mode	FMS_Speed_Modes
Engine_Out	Engine_Out_Type
FCC_Engaged_Mode	FCC_Engaged_Modes
FMS_Mode	FMS_Modes
Flight_Phase	Flight_Phase_Type
Requested_Climb_Speed_Mode	Climb_Speed_Mode_Requests
Requested_FCC_Speed_Mode	FCC_Speed_Mode_Requests

Figure 2: Variable Dictionary of SMI

FMS Speed Mode” previously in *“Edit” Mode* and not as *“Climb FMS Speed Mode”* previously in *“Edit Mode”*. We assumed this is also the case with *“Economy Mode”*.

- We have similar doubts about the inclusion of the word *“Mode”* in the names of variables and in the mode class (which is clearly *not* a *“Mode”*). That is, we were unable to determine whether the name *“Climb FMS Speed Mode”* should or should not include the word *“Mode”* after *“Climb FMS Speed”*.
- Another minor anomaly was the use of redundant symbols in both the graphical and logical descriptions. For example, *“Flight Phase is Takeoff”* and *“Flight Phase = Takeoff”* are both used. Having two alternate denotations for equality makes a gratuitous distinction, which clearly detracts from readability. A syntax checker would have immediately uncovered such anomalies.

Issues raised when creating the variable dictionary:

- A major concern was that there were no descriptions provided of variable names and their meanings. Sometimes called *designations*, these descriptions establish a connection between each

entity in the document and the real world. Fortunately, we drew upon our prior experience with aircraft flight management systems [3] to make sense of the variable names and acronyms such as CAS, FCC, etc.

- We detected an obvious inconsistency (in one of the tables of their tabular description) in the name of variable *“Requested Climb Speed Mode”* which in one instance is written instead as *“Requested Climb FMS Speed Mode”*. (This alternate name also appears in question 1.) Also, we are unsure whether the word *“Requested”* is to be included in the variable name since the textual description reads *“Edit is requested for Climb Speed Mode”*.

A note on the SCR event notation: The SCR notation *“@T(c) WHEN d”* denotes a *conditioned event*, defined as

$$\text{@T}(c) \text{ WHEN } d \stackrel{\text{def}}{=} \neg c \wedge c' \wedge d,$$

where the unprimed conditions c and d are evaluated in the *“old”* state, and the primed condition c' is evaluated in the *“new”* state. Informally, this expression denotes the event *“predicate c becomes true in the new state when predicate d holds in the old state”*. The notation *“@F(c)”* denotes the event $\text{@T}(\text{NOT } c)$ and *“@C(x)”* denotes the event *“term x has changed value”*.

Mode Class = Climb_FMS_Speed_Mode		
Old Mode	Events	New Mode
Edit, Max_Climb, Economy	@T(Flight_Phase = Done)	Default
	@T(Flight_Phase = Descent) when (Flight_Phase = Takeoff)	
	@T(Flight_Phase = Cruise) when (Flight_Phase = Climb)	
	@T(Flight_Phase = Descent) when (Flight_Phase = Climb)	
	@T(Engine_Out = Engaged)	
Max_Climb	@T(FCC_Engaged_Mode = Altitude_Hold_Speed)	Default
	@T(FCC_Engaged_Mode = Altitude_Hold_Idle_Thrust)	
	@T(FCC_Engaged_Mode = Altitude_Hold_Maximum_Thrust)	
Edit, Max_Climb, Default	@T(Requested_FCC_Speed_Mode = Economy) when (Flight_Phase in {Preflight, Takeoff, Climb})	Economy
	@T(Requested_FCC_Speed_Mode = AFS_Speed) when (Flight_Phase in {Preflight, Takeoff, Climb})	
	@T(Requested_Climb_Speed_Mode = Economy) when (Flight_Phase in {Takeoff, Climb})	
Edit, Default, Economy	@T(Requested_Climb_Speed_Mode = Max_Climb)	Max_Climb
Max_Climb, Default, Economy	@T(Requested_FCC_Speed_Mode = Edit_CAS) when (Flight_Phase in {Preflight, Takeoff, Climb})	Edit
	@T(Requested_FCC_Speed_Mode = Edit_Mach) when (Flight_Phase in {Preflight, Takeoff, Climb})	
	@T(Requested_Climb_Speed_Mode = Edit)	
Economy	@T(Flight_Phase = Climb) when (Flight_Phase = Cruise and Cruise_FMS_Speed_Mode = Edit)	Edit
	@T(Flight_Phase = Takeoff) when (Flight_Phase = Descent and Descent_FMS_Speed_Mode = Edit)	
	@T(Flight_Phase = Climb) when (Flight_Phase = Descent and Descent_FMS_Speed_Mode = Edit)	
	@T(Flight_Phase = Takeoff) when (Flight_Phase = Approach and Descent_FMS_Speed_Mode = Edit)	
	@T(Flight_Phase = Climb) when (Flight_Phase = Approach and Descent_FMS_Speed_Mode = Edit)	

Figure 3: Mode Transition Table of SMI

Mode Transition Table The mode transition table of Figure 3 defines the behavior of the mode class; i.e., the table defines all events that change the value of the mode class `Climb_FMS_Speed_Mode`. For example, the first row of the table states: “If `Climb_FMS_Speed_Mode` is `Edit`, `Max_Climb`, or `Economy`, and `Flight_Phase` transitions to `Done`, then `Climb_FMS_Speed_Mode` changes to `Default`.” An assumption is that events omitted from the table do not change the value of the mode class. For example, when `Climb_FMS_Speed_Mode` is `Economy`, the occurrence of the input event `@T(Engine_Out = Not_Engaged)` does not change the value of `Climb_FMS_Speed_Mode`.

Issues raised when creating the mode transition table: We found a type error in the tabular description of [6] where “Requested Climb Speed Mode = Economy” is wrongly entered as “Requested Climb Speed Mode” (which can never by itself be “true”). Also, their description provides no initial values for any variable, including the mode class. However, it is evident from the questions that there is an implied initial mode. For example, question 1: “Describe two scenarios that would cause the Climb FMS Speed Mode to be `Edit`” suggests that the initial mode is *not* “`Edit`”. For the purpose of our analysis, we assumed that the initial mode is “`Default`”. Also, for applying the model checker and simulator we had to assign initial values to all monitored variables.

```

Question2 = @T(Flight_Phase = Climb) when (Engine_Out = Engaged and
      Requested_Climb_Speed_Mode = Economy and
      Flight_Phase = Takeoff) => Climb_FMS_Speed_Mode' != Max_Climb;

```

Variable	Old Value	New Value
=====	=====	=====
Cruise_FMS_Speed_Mode	Max_Climb	Max_Climb
Descent_FMS_Speed_Mode	Max_Climb	Max_Climb
Engine_Out	Engaged	Engaged
FCC_Engaged_Mode	Altitude_Hold_Speed	Altitude_Hold_Speed
FMS_Mode	Lateral_Only	Lateral_Only
Flight_Phase	Takeoff	Climb
Requested_Climb_Speed_Mode	Economy	Economy
Requested_FCC_Speed_Mode	Economy	Economy
Climb_FMS_Speed_Mode	Max_Climb	Max_Climb

Figure 4: Property and Scenario Returned by Salsa

3 APPLYING THE SCR TOOLSET

After creating the domain model, we used the automated consistency checker [4] to check for proper syntax, type correctness, missing cases, nondeterminism, and other application-independent properties. Application of this tool uncovered a missing definition for variable `Cruise_FMS_Speed_Mode` in our model. We noticed that the only value assigned to this variable is `Edit`. However, rather than creating a new type, we decided to assign to this variable the type assigned to variable `Descend_FMS_Speed_Mode` (which additionally contains the value `Max_Climb`).

At this point we were quite confident about the fidelity of our model relative to the descriptions in [6], and therefore proceeded to answer the questions in [6], some of which are reproduced in Appendix B. In the original study, subjects were asked to use a description in any notation to answer the first two questions. Each subsequent set of four questions was designated for a notation of one kind. In our study, answers to all the questions were derived from the mode transition table. To answer some of the questions, we used the automatic invariant checker Salsa [2]. The method used was to propose the negation of each question as an invariant and to use Salsa to find a scenario that falsifies the invariant, thereby answering the question in the affirmative. For example, we reproduce in Figure 4 the predicate corresponding to the negation of question 2: “Could the Climb FMS Speed Mode be Max Climb under the following conditions? (a) Engine Out is Engaged (b) Requested Climb FMS Speed Mode [sic] is Economy and (c) Flight Phase transitions from Takeoff to Climb” and a scenario returned by Salsa (in just over a second)

that falsifies this predicate.

The scenario in Figure 4 shows a state transition where the mode class `Climb_FMS_Speed_Mode` remains `Max_Climb` when `Flight_Phase` transitions from `Takeoff` to `Climb` and the other conditions are met in the pre-state. One thing to keep in mind is that the scenario returned by Salsa may not be a true counterexample since the pre-state (corresponding to “Old Value” of variables) may be unreachable. To establish this as a true counterexample requires validation by domain experts, who are usually able to determine, using their expertise, whether the pre-state is reachable³. In the absence of access to domain experts, we decided to run the model checker Spin after an automatic translation of the SCR model to the language of Spin using the method outlined in [1]. Running Spin immediately yielded a counterexample (a sequence of monitored variable changes starting from the initial state and leading to the transition that falsifies the predicate) of 16 steps. A scenario under which question 2 could be answered in the affirmative was demonstrated by running the simulator of the SCR Toolset on the counterexample returned by Spin. (The pre-state and transition of Spin’s counterexample correspond almost exactly to Salsa’s scenario.)

In addition to question 2, we answered questions 1, 5, 9a, and 9b (see appendix B) along the same lines. We leave it as an exercise to the reader to discover that the answers to the other questions are trivially derived from the mode transition table.

³If the pre-state is unreachable, the wording of a (counter-) argument is something like: “But, isn’t it the case that *foo* can never be true unless *bar* is true also?”

Also, when answering question 9 we discovered that variable `FMS_Mode` can take on an additional value `Lateral-Vertical`.

4 CONCLUSIONS

In this paper, we outlined a process for creating a domain model in SCR of a simple Speed Mode indicator (described in [6]), specifically to answer a set of questions. The modeling and analysis took less than one afternoon. The process of constructing the model had the benefit of additionally raising a number of issues in the description provided in the paper, which has been published in a peer reviewed conference. It should be noted that, as opposed to previous reports on using formal methods that have detected problems in *informal* descriptions of software requirements or domain properties, the basis of our study was an informal and *three* formal descriptions of a system. This demonstrates the utility of formal methods (and the power of using the *right* notation) in requirements elicitation, especially for high assurance systems.

In their paper, Zimmerman et al. [6] hypothesize that the readability of formal requirements specifications is the limiting factor in the acceptance of formal methods by the industrial community. Another important factor is whether a formal description is organized to find answers easily to commonly encountered questions. In our opinion, the organization of the mode transition table of SCR facilitates finding precise answers to the questions in [6]. Whereas we do not wish to downplay the importance of readability, we wish to emphasize the importance of automated tool support. For example, all the residual problems in the published descriptions in [6] could have been easily caught by a simple syntax and type checker. Also, in our experience, tools that support a specific conceptual model and method such as SCR are more cost-effective than general-purpose tools. If a formal model lacks a strong underlying method, the benefits of automation are likely to be minimal. Since the SCR method focuses on a class of systems (embedded systems) and standardizes the conceptual model, the notation, and the process, significant automated tool support is possible.

We believe that a promising approach to applying formal methods in requirements engineering is to provide a framework for using the logic-based notation associated with most formal methods by adopting a notation, such as a graphical or tabular notation, that is intuitive and easy to use. Specifications in such “user-friendly” notations are automatically translated to other notations that are more amenable to formal analysis. In addition, with powerful, easy-

to-use tool support it is feasible to answer questions about the domain and to find errors automatically, with the tools providing feedback useful in understanding the domain.

ACKNOWLEDGMENTS I thank Connie Heitmeyer for bringing to my attention the paper by Zimmerman et al., and also for insisting that I read it! Discussions with Jim Kirby helped firm up the idea of a domain model. I thank Eric Tressler and Jim Kirby for providing useful feedback on a previous draft of this paper. Finally, I thank the anonymous referees for their many thought provoking comments.

REFERENCES

1. R. Bharadwaj and C. L. Heitmeyer. Model checking complete requirements specifications using abstraction. *Automated Software Engineering*, 6(1), January 1999.
2. R. Bharadwaj and S. Sims. Salsa: Combining constraint solvers with BDDs for automatic invariant checking. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS '2000)*, Berlin, March 2000.
3. R. Bharadwaj and C. L. Heitmeyer. Applying the SCR requirements method to a simple autopilot. In *Proc. Fourth NASA Langley Formal Methods Workshop (LFM97)*, NASA Langley Research Center, September 1997.
4. C. L. Heitmeyer, J. Kirby, B. G. Labaw, and R. Bharadwaj. SCR*: A toolset for specifying and analyzing software requirements. In *Proc. 10th Computer-Aided Verification*, Vancouver, Canada, 1998.
5. C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. *ACM Trans. on Software Engineering and Methodology*, 5(3):231–261, April–June 1996.
6. M. K. Zimmerman, K. Lundqvist, and N. Leverson. Investigating the readability of state-based formal requirements specification languages. In *Proc. International Conference on Software Engineering (ICSE 2002)*, Orlando, Florida, USA, May 2002. Paper URL: sunnyday.mit.edu/papers/icse-marc-final2.pdf

APPENDIX A: DESCRIPTION OF THE CLIMB FMS SPEED MODE

- 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
 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 Descend to Takeoff
 - b Flight Phase transitions from Descend to Climb
 - c Flight Phase transitions from Approach to Takeoff
 - d Flight Phase transitions from Approach to Climb

APPENDIX B: LIST OF QUESTIONS

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 is Economy

Flight Phase transitions from Takeoff to Climb

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?

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?

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

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 FMS Speed Mode

could the Climb FMS Speed Mode be (a) Default? (b) 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?