Specifying Modal Systems using Event-B[*]

Fernando L. Dotti[1,2], Alexei Iliasov[1], and Alexander Romanovsky[1]

[1] Centre for Software Reliability, Newcastle University, UK
{alexei.iliasov, alexander.romanovsky}@newcastle.ac.uk
[2] Faculdade de Informática, PUCRS, Brazil
fernando.dotti@pucrs.br

Abstract. Several safety-critical systems, such as avionic, transportation and space systems, use the notion of operation modes. Operation modes are useful structuring units that facilitate design, specially if used with state-based formal methods. However, modelling abstractions to support the specification, analysis and correct construction of modal systems are still lacking. The contribution of this paper is twofold: (i) modal systems and modal systems refinement are discussed and formalized; (ii) the relation of a modal system specification with an Event-B model is discussed, showing how to demonstrate that the behaviour of an Event-B model can satisfy a modal system.

1 Introduction

A considerable number of systems are described using the notion of 'operation modes', which serves to structure their operation. These are called 'modal systems'. For example, in [1,2] the authors specify and analyse the operation mode logics of space and avionic systems. An extension of an Automated Highway System with degraded operation modes that tolerates several kinds of faults is discussed in [3]. The Steam Boiler Control [4], a classic case study showing the use of formal methods, is based on the notion of operation modes. More recent examples of the extensive use of modes for the specification of transportation and space systems can be found in [5].

As many modal systems are critical, a number of specification languages include modelling abstractions in order to support mode descriptions: Modecharts [6], the Architecture Analysis & Design Language (AADL) [7] and Dependability Requirement Engineering Process (DREP) [8]. Modecharts focus on the specification of real-time properties of mode and mode switching, with less support for specifying the overall system functionality or mode characterization. AADL offers a wide range of concepts for modelling systems, modes being one of them. DREP provides a methodology for developing modal systems using UML diagrams. These efforts notwithstanding, we believe that there is a lack of formally

[*] This work is partially supported by the ICT DEPLOY IP and the EPSRC/UK TrAmS platform grant. Fernando L. Dotti is supported by CNPq/Brazil grant 200806/2008-4.

defined abstractions for describing modal systems as well as approaches to analyzing and rigorously deriving implementations of these systems. In our previous work [9], we introduced modal systems and discussed their use for structuring dependable systems (focusing specifically on the recovery and degradation modes). The aim of this paper is to discuss in detail and to formally define modal systems, and to apply refinement to modal system development. The refinement of modal systems makes it easier to map requirements to models with a step-wise reasoning about system properties. Moreover, the use of modes imposes a structure on the architecture of a model, especially in the case of state-based formal methods. Thus, as the second contribution of this paper, we will demonstrate the use of Event-B for specifying a modal system and discuss how to prove that a given Event-B model satisfies a given modal system specification.

The rest of the paper is structured in 3 major parts: first, modal systems and modal system refinement are formally defined in Sections 2 and 3; then Event-B is briefly introduced and its relation to modal systems defined in Sections 4 and 5; finally, Section 6 presents a case study to illustrate and evaluate the ideas introduced in the paper. Section 7 concludes the paper with a summary and an outline of future extensions of the approach.

2 Modal Systems

While there is no widely accepted formal definition of mode[3] and modal system, several authors use mode to denote the expected system functionality under a distinguished working condition of the system. A modal system would denote the assembly of a set of such modes related by mode transitions. Mode transitions cover the possible changes in the working conditions of a system, originated either by environmental changes or by system evolution. As a brief example, the Steam Boiler Control [4] states that the *normal mode* is characterized by a working water level sensor and the water level in normal range - in such conditions the system works to keep the water level (read from sensor) in normal range. In the event of a detected failure of a water level sensor, the system switches to *rescue mode* where the sensor is not trusted - in such conditions the system operates differently, based on the amount of water pumped into the boiler and amount of steam generated. The case study in section 6 discusses a cruise control system with several modes, each organized in this way and related by transitions. Examples in avionics use modes to denote phases of a flight, and in space systems to denote operational status of on-board instruments, among others.

We are interested in how to specify, analyse and build correct modal systems. In this section we first propose and formalise abstractions to specify and reason about modal systems and then define modal system refinement. The abstractions proposed support the formal specification of requirements of modal systems in terms of (i) functionality and (ii) operating conditions of each mode, and (iii) possible mode switching. A specific modal system specification is an

[3] Mode and operation mode are used as synonyms.

instantiation of such abstractions, formalizing the requirements of a concrete system using modes and mode switching - it defines a class of possible models respecting those requirements. A modal system specification does not replace traditional formal modelling but rather complements it. The construction of a complete model to a modal system specification can be done using any existing formalism, provided it is possible to demonstrate that the model satisfies the modal system specification. This is discussed in Section 5. Due to the nature of modal systems, we follow a state-based approach to propose suitable abstractions. We consider that the state of a model is detailed enough to allow one to distinguish its different operating conditions and also to characterize required mode functionality and possible mode switching in terms of state transitions. Below we introduce the necessary elements (in definitions 1 and 2) to formally define modal systems (definition 3).

Definition 1 (State, Invariant, Assumption, Guarantee). *Given a set of variables Var and a set of values Val, the state of a system is a (total) function $v : Var \rightarrow Val$. We denote as State the set of all states. Invariant and assumption are predicates over state variables. A guarantee is a predicate over $Var \times Var'$, where $Var' = \{x' | x \in Var\}$. It is interpreted over $State \times State$.*

Invariant is a property preserved at each point in a systems life time. Often it is interpreted as a characterisation of safe states of a system. A guarantee is used to express the requirements towards the functionality of a mode, while an assumption expresses the requirements of a mode, to the rest of the system, to assure the functionality required by the guarantee. A pair assume/guarantee can be seen as a contract between the mode and the rest of the system, and is what defines a mode, as follows.

Definition 2 (Mode). *Given an invariant I, an assumption A and a guarantee G, a Mode is a pair A/G where:*
- *an assumption characterises a non-empty set of states: $\exists v \cdot A(v)$, assuring that a mode contributes to system functionality;*
- *G is feasible: $\exists v, v' \cdot I(v) \wedge A(v) \Rightarrow G(v, v')$. I.e. a mode should permit a concrete implementation of the required functionality;*
- *G preserves the invariant I and the modes assumption A:*
  $I(v) \wedge A(v) \wedge G(v, v') \Rightarrow I(v') \wedge A(v')$.

Given a mode M_i we denote its assumption by A_i and its guarantee by G_i.

Concerning the last condition, it would not make sense if a guarantee would require the mode to violate the invariant. Also, we postulate that a mode guarantee should neither violate its assumption: we consider that this helps to clearly separate the specification of actions that may cause mode switching from those that preserve current mode, an important feature in modal systems. Therefore, a mode is better understood by looking also at the possible mode transitions where it is involved, according to the next definition.

Definition 3 (Mode Transition, Modal System). *A Modal System is a tuple $MSys = (Var, Val, I, M, T)$ where: Var a set of variables of the system; Val the set of possible values for variables; I is an invariant; and*

1. *M is a finite set of modes $(M_i = A_i/G_i)_{i \leq n, n \in \mathbb{N}} \cup \{\top_M, \bot_M\}$ s.t.:*
   (a) *$I(v) \Leftarrow A_1(v) \oplus \cdots \oplus A_n(v)$, where $\oplus$ is the set partitioning operator. This implies that for each mode a different assumption is declared, that mode assumptions are exclusive, and that assumptions are valid with respect to the invariant.*
2. *$T : M \to M$ is a set of mode transitions, with the following restrictions:*
   (a) *$i \rightsquigarrow \top_M \notin T \;\wedge\; \bot_M \rightsquigarrow j \notin T \;\wedge\; \top_M \rightsquigarrow \bot_M \notin T$*
   (b) *Let T^* be the transitive closure of T, $\forall m \in M - \{\bot_M\} \cdot (\top_M \to m) \in T^*$*
   (c) *If the system terminates, the following restrictions also apply: T is surjective and total; and $\forall m \in M \cdot (m \to \bot_M) \in T^*$;*

A modal system is an assembly of several modes (M) related by mode transitions (T). It is assumed that a system is only in one mode at a time, represented by condition 1a. The meaning and implications of a system being in more than one mode are not trivial and subject of further study. A mode transition is an atomic step switching system from one source i to one destination j mode. Transition from one mode to more than one at the same time is not possible. Modes $\top_M$ and $\bot_M$ are used to denote the mode before system initiation and after termination, respectively. The possible mode transitions of a modal system are defined by T. According to condition 2b, $\top_M$ is present in any modal system specification. A transition $\top_M \rightsquigarrow M_i$ defines that M_i is a possible initial mode of the modal system. Other such transitions may exist defining more than one initial mode. Some systems may be non-terminating, in which case there is no mode transition to $\bot_M$. In case of terminating systems, it is required that $\bot_M$ may be reached - cond. 2c. Condition 2a sates that it is not possible to switch to a state before initiation or from the mode denoting termination to another mode; and during its lifetime a system enters at least one operation mode. The understanding of modes and mode switching is complemented by the definition of modal system behaviour.

Definition 4 (Modal System Behaviour). *The behaviour of a modal system $MSys = (Var, Val, I, M, T)$, given by a transition system $MST = (MState, \to)$ where $MState = \{\langle m, v\rangle \mid m \in \{1..n, \top_M, \bot_M\}$ is a mode index $\wedge v \in \{State \cup Undef\}\}$; and $\to: MState \to MState$ are transitions given by the rules:*

$$\boxed{\textsf{start}}\frac{\top_M \rightsquigarrow k \wedge A_k(v)}{\langle \top_M, Undef\rangle \to \langle k, v\rangle}$$

$$\boxed{\textsf{internal}}\frac{A_m(v) \wedge G_m(v, v') \wedge A_m(v')}{\langle m, v\rangle \to \langle m, v'\rangle}$$

$$\boxed{\textsf{switching}}\frac{m \rightsquigarrow n \wedge A_m(v) \wedge A_n(v')}{\langle m, v\rangle \to \langle n, v'\rangle}$$

The state of a system described using operation modes is a tuple (m, v) where m is the index of a current operation mode and v is the current system state. Mode index helps to clarify how mode switching is done although it may be computed from v alone due to condition 1a in Definition 3. In the following, each of the transition rules is explained.

Initialisation. A system starts executing one of initiating mode transitions $\langle \top_M, Undef \rangle \rightarrow \langle k, v \rangle$. The transition switches the system on, by establishing a possible state defined by $A_k(v)$, and places it into some system mode $M_k = A_k/G_k$. This behaviour is described by rule *start*.

Evolution. A modal system may evolve either performing internal or switching transitions. Rule *internal* states that while the system is in some mode m the state of model variables evolves so that the next state is any state v' satisfying both the corresponding guarantee $G(v, v')$ and the modes assumption $A(v')$. Rule *switching* states that the system may switch modes if there is a defined mode transition originating from the current mode. Both *internal* and *switching* transitions compete with each other: at each step a non-deterministic choice is made between the two.

Termination. A system terminates by executing one of terminating mode transitions $t \leadsto \bot_M$. Not every system has to have this transition: a control system would be typically designed as never aborting. There can be any number of terminating mode transitions. Due to condition 2a, no mode transitions are possible after $\bot_M$.

3 Refinement of Modal Systems

Refinement [10] is a formal technique for step-wise construction of correct systems. At each step a model (said abstract) is refined into a new one (said concrete) by introducing realisation aspects through correctness-preserving transformations: the concrete model preserves properties of the abstract and can be used in place of it. Successive refinement steps would result in an implementation which is correct by construction. Refinement itself is a combination of behavioural refinement, eliminating non-determinism and introducing finer details, data refinement, by changing types and introducing data structures, and superposition refinement when new functionality is added keeping the existing model. Now we discuss behavioural and data refinement of modal systems.

Modal System behavioural refinement details modes assumption or guarantee or both. A mode can also be detailed in more than one corresponding mode in the concrete level. We postulate that mode assumption cannot be strengthened during refinement. This is based on the understanding that an assumption is a requirement of a mode to its environment. As a system developer cannot assume control over the environment of a modelled system, a stronger requirement to an environment may not be realisable. On the other hand, a weaker requirement to an environment means that a system is more robust as it would remain operational in a wider range of environments. Therefore, weakening assumptions during refinement is desired. Symmetrically, a mode guarantee cannot be weakened as a mode guarantee is understood as a contract of a mode with the rest of a system and the system environment. In other words, weakening a mode guarantee could violate expectations of another system part.

Mode transitions must be consistently refined along with refinement of modes. The general rules for refining mode transitions are: (i) a mode transition present in an abstract model must have at least one corresponding transition in a concrete model. If a mode with a transition is split into two new modes, the transition can be associated with any one of the new modes or both; (ii) no new transitions may appear relating an abstract mode to another mode; (iii) new transitions may be defined on concrete modes. Now we formalize the above discussed notion of behavioural mode refinement.

Definition 5 (Modal System Behavioural Refinement). *Given:*

- *a modal system* $MSys_{abs} = (Var_{abs}, Val_{abs}, I_{abs}, M_{abs}, T_{abs})$*;*
- *a modal system* $MSys_{cnc} = (Var_{cnc}, Val_{cnc}, I_{cnc}, M_{cnc}, T_{cnc})$*;*
- *function source such that:* $\forall\ (t = a \rightarrow b) \in T : M \rightarrow M \cdot src(t) = a$*;*
- *function target such that:* $\forall\ (t = a \rightarrow b) \in T : M \rightarrow M \cdot target(t) = b$*;*

a refinement of $MSys_{abs}$ *into* $MSys_{cnc}$ *is defined by a pair* $ref = (ref^M, refT)$ *of functions* $ref^M : M_{cnc} \rightarrow M_{abs}$ *and* $ref^T : T_{cnc} \rightarrow T_{abs}$ *such that:*

1. ref^M *is total and surjective and* ref^T *is partial and surjective;*
2. *an abstract mode assumption is stronger than the disjunction of assumptions of its concrete modes:* $\forall m \in M_{abs} \cdot \bigvee_{\forall j \cdot j \in M_{cnc} \wedge ref^M(j)=m} A_j \Leftarrow A_m$
3. *an abstract mode guarantee is weaker than the disjunction of guarantees of its concrete modes:* $\forall m \in M_{abs} \cdot \bigvee_{\forall j \cdot j \in M_{cnc} \wedge ref^M(j)=m} G_j \Rightarrow G_m$
4. *concrete transitions not mapped to abstract ones have the same abstract mode as source and target (i.e. it was an internal, or non-observable, transition of an abstract mode):* $\forall t \notin dom(ref^T) \cdot ref^M(src_{ref}(t)) = ref^M(target_{ref}(t))$
5. *for all transition* $t \in dom(ref^T)$*, the squares bellow commute:*

$$\begin{tikzcd}
T_{abs} \arrow[r, mapsto, "src_{abs}"] & M_{abs} \\
dom(T_{cnc}) \arrow[u, "ref^T"] \arrow[r, mapsto, "src_{cnc}"'] \arrow[ur, phantom, "="] & M_{cnc} \arrow[u, "ref^M"']
\end{tikzcd}
\qquad
\begin{tikzcd}
T_{abs} \arrow[r, mapsto, "target_{abs}"] & M_{abs} \\
dom(T_{cnc}) \arrow[u, "ref^T"] \arrow[r, mapsto, "target_{cnc}"'] \arrow[ur, phantom, "="] & M_{cnc} \arrow[u, "ref^M"']
\end{tikzcd}$$

These conditions mean that: (1) all concrete modes have an abstract mode that they refine, and all abstract modes are refined by at least one concrete mode; and all abstract mode transitions are refined into one or more concrete mode transitions; (2) w.r.t. variables in the abstract system, the concrete modes cover the same state space as the abstract one - it is not possible to restrict assumptions by refinement; (3) guarantees given by the concrete system may be stronger (more deterministic) than the abstract one; (4) some transitions may be added in a refinement step, those that, in the abstract modal system, would have a same mode as source and destination; (5) mode and mode transition refinement are consistent with each other.

With data refinement, the set v of model variables is changed to some new set u and model invariant $I(v)$ is replaced with a new invariant $J(v, u)$, often called a *gluing invariant*. The mentioning of old variables v in new invariant J allows a modeller to express a linking relation between the state of concrete and abstract models. Given a gluing invariant $J(v, u)$, data refinement can be added to definition 5 by extending conditions 2 and 3 respectively as:

$\forall m \in M_{abs} \cdot \bigvee_{\forall j \cdot j \in M_{ref} \wedge ref^M(j)=m} J(v, u) \wedge A_j(u) \Leftarrow A_m(v)$

$\forall m \in M_{abs} \cdot \bigvee_{\forall j \cdot j \in M_{ref} \wedge ref^M(j)=m} J(v, u) \wedge J(v', u') \wedge G_j(u, u') \Rightarrow G_m(v, v')$

Proposition 1. *Given:*

- *an abstract modal system* $MSys_{abs} = (Var_{abs}, Val_{abs}, I_{abs}, M_{abs}, T_{abs})$*;*
- *a concrete modal system* $MSys_{cnc} = (Var_{cnc}, Val_{cnc}, I_{cnc}, M_{cnc}, T_{cnc})$*;*
- *a refinement* $ref = (ref^M, ref^T)$ *where* $ref^T : T_{cnc} \rightarrow T_{abs}$*;*

any possible sequence of modes described by the transition system of $MSys_{cnc}$ *can be translated into a possible sequence of modes described by the transition system of* $MSys_{abs}$*.*

Proof. *By definitions 3 and 4 the initial mode of a modal system is* $\top_M$*, and by definition 5* $ref^M(\top_M) = \top_M$*. So* $\top_M$ *is initial in any sequence of modes described by both* $MSys_{abs}$ *and* $MSys_{cnc}$*. Now consider the concrete modal system in any mode* m_{c1}*, corresponding through* ref^M *to an abstract mode* m_a*. By definition 5, conditions 1 and 4, a mode transition in the concrete level is either a new transition or refinement of a transition in the abstract level.*

Consider the first case: by condition 4 a new transition can be added only among modes that refine a same abstract mode. In this case, switching from m_{c1} *to* m_{c2}*, both corresponding through* ref^M *to* m_a*, has no effect at the abstract level -* m_a *is kept.*

Consider the second case: in definition 5, by conditions 1, 4 and 5, any mode transition, which is not new (case above), starting from m_{c1} *refines a transition starting from* m_a *and any mode transition arriving in* m_{c1} *also refines a transition arriving in* m_a*. This means that* m_{c1} *may offer a subset of possibilities of transitions, compared to* m_a*. However, since* ref^T *is surjective (condition 1), all transitions where* m_a *is involved have to be mapped to the concrete level. Thus another mode* m_{c2}*, that have to be refined from* m_a *(due to condition 5), will be associated to transitions that, together with the transitions where* m_{c1} *is involved, are equivalent to the transitions of* m_a*. The transitioning among* m_{c1} *and* m_{c2}*, according to the case above, does not correspond to a mode change in the abstract level because* m_{c1} *and* m_{c2} *refine the same* m_a*. Thus, the transitions of* m_{c1} *and* m_{c2} *equate the transitions where* m_a *is involved.*

Since each refinement does not add new mode switching possibilities, except those that have no observable effect at the abstract level, and since the transitions involving concrete modes of a same abstract mode exactly cover the transitions involving the abstract mode, the observable sequence of modes of a concrete modal system can be translated to an observable sequence of modes of the respective abstract modal system by taking each concrete mode m_{ci} *of the sequence and substituting by the corresponding abstract one* $(ref^M(m_{ri}))$ *while eliminating consecutive switchings to the same resulting abstract mode.*

A terminating modal system (one that defines switching to $\bot_M$) does not introduce deadlock in mode transitions by definition 3, condition 2c.

4 Event-B

Event-B is a state-based formalism closely related to Classical B [11] and Action Systems [12].

Definition 6 (Event-B Model, Event). *An Event-B Model is defined by a tuple* $EBModel = (c, s, P, v, I, R_I, E)$ *where* c *are constants and* s *are sets known in the model;* v *are the model variables*[4]*;* $P(c, s)$ *is a collection of axioms constraining* c *and* s*;* $I(c, s, v)$ *is a model invariant limiting the possible states of* v *s.t.* $\exists c, s, v \cdot P(c, s) \wedge I(c, s, v)$ *- i.e.* P *and* I *characterise a non-empty set of model states;* $R_I(c, s, v')$ *is an initialisation action computing initial values for the model variables; and* E *is a set of model* events.

Given states v, v' *an event is a tuple* $e = (H, S)$ *where* $H(c, s, v)$ *is the guard and* $S(c, s, v, v')$ *is the before-after predicate that defines a relation on current and next states. We also denote an event guard by* $H(v)$*, the before-after predicate by* $S(v, v')$ *and the initialization action by* $R_I(v')$*.*

Model correctness is demonstrated by generating and discharging a collection of proof obligations. Putting it as a requirement that an enabled event produces a new state v' satisfying the model invariant, the model *consistency* condition states that whenever an event on an initialisation action is attempted, there exists a suitable new state v' such that a model invariant is maintained - $I(v')$. This is usually stated as two separate proof obligations: a feasibility $(I(v) \wedge H(v) \Rightarrow \exists v' \cdot S(v, v'))$ and an invariant satisfaction obligation $(I(v) \wedge H(v) \wedge S(v, v') \Rightarrow I(v'))$. The behaviour of an Event-B model is as follows.

Definition 7 (Event-B Model Behaviour). *Given* $EBModel = (c, s, P, v, I, R_I, E)$*, its behaviour is given by a transition system* $BST = (BState, \rightarrow)$ *where:* $BState = \{\langle v\rangle | v \text{ is a state}\} \cup Undef$ *and* $\rightarrow: BState \rightarrow BState$ *are transitions given by the rules:*

$$\boxed{\textsf{start}}\,\frac{R_I(v') \wedge I(v')}{Undef \rightarrow \langle v'\rangle}$$

$$\boxed{\textsf{transition}}\,\frac{\exists (H, S) \in E \cdot I(v) \wedge H(v) \wedge S(v, v') \wedge I(v')}{\langle v\rangle \rightarrow \langle v'\rangle}$$

According to rule *start* the model is first initialized to a state satisfying $R_I \wedge I$ and then, as long as there is an enabled event (rule *transition*), the model may evolve by firing an enabled event and computing the next state according to the event's before-after predicate. Events are atomic. In case there is more than one enabled event at a certain state, the demonic choice semantics is applies. The semantics of an Event-B model is given in the form of proof semantics, based on Dijkstra's work on weakest precondition [13].

To refine model M one constructs a new model M' such that at a certain level of observation new model is at least as good as the old one. Formally, this is demonstrated by constructing a refinement mapping between M' and M that would show that for any valid state of M' there is a corresponding state in M. In Event-B, this is accomplished by discharging a number of refinement proof obligations formulated for each model event.

[4] For convenience, as in [11], no distinction is made between a vector of variables and a state of a system.

An extensive tool support through the Rodin Platform makes Event-B especially attractive [14]. An integrated Eclipse-based development environment is actively developed, and open to third-party extensions in the form of Eclipse plug-ins. The main verification technique is theorem proving supported by a collection of theorem provers. Model checking is also possible.

5 Modal Systems and Event-B

As already discussed, a modal system defines a class of possible models which may be specified using established formal methods. Therefore, a consistency condition is needed such that we can evaluate if a given model satisfies a modal system. In this section we discuss first such condition and then how to enrich the set of proof obligations on an Event-B model to show that it satisfies a modal system specification.

Definition 8 (Modal System Consistency Conditions for an Event-B Model). *Given:*

- *an Event-B model* $EBModel = (c_E, s_E, P_E, v_E, I_E, R_{IE}, E_E)$ *and*
- *a Modal System* $MSys = (Var_M, Val_M, I_M, M_M, T_M)$ *where* $Var_M \subseteq v_E$*;*
- *a state projection function* $fs_{EtoM}(s_E) = s_M$ *that, given a state* s_E *of the Event-B Model, constructs the corresponding state* s_M *of the modal system by projecting on the modal system state;*
- *a predicate projection function* $fp_{EtoM}(P_E) = P_M$ *that, given a predicate over* v_E *projects to the corresponding predicate over* Var_M*;*

$EBModel$ *satisfies a modal system* $MSys$ *iff:*

1. *both specify the same invariant on* $MSys$*:* $fp_{EtoM}(I_E) = I_M$
2. *the initialisation is compatible, i.e. the initial state* $EBModel$ *is compatible with the assumption of any first mode of* $MSys$*:* $fp_{EtoM}(R_{IE}) \subseteq \bigvee_{\forall t \in T_M \cdot src(t) = \top_M} A_{target(t)}$
3. (a) *every transition* $t_E : s1_E \rightarrow s2_E$ *of the behaviour of* $EBModel$ *has a corresponding transition* $t_M : \langle m1, s1_M \rangle \rightarrow \langle m2, s2_M \rangle$ *in the behaviour of* $MSys$ *such that:* $fs_{EtoM}(s1_E) = s1_M \wedge fs_{EtoM}(s2_E) = s2_M$ *and* t_M *is either:*
      i. *an internal transition of* $MSys$*, when* $m1 = m2$ *or*
      ii. *a switching transition of* $MSys$*, when* $m1 \neq m2$*;*
   (b) *every mode switching transition in* $MSys$ *has at least one corresponding transition in* $EBModel$*;*
4. (a) *every state* s_E *in the behaviour of* $EBModel$ *has a corresponding state* (m, s_M) *in the behaviour of* $MSys$*. The correspondence is such that* $fs_{EtoM}(s_E) = s_M$*;*
   (b) *for each mode* $M_i = A_i/G_i \in M_{MSys}$ *there is at least one state* s_E *of* $EBModel$ *such that* $A_i(fs_{EtoS}(s_E))$*: all modes are reachable according to Modal System definition.*

Now we discuss how the above defined Consistency Conditions can be fulfilled using Event-B proof obligations enriched with information about the Modal System. Conditions 1 and 2 have to be shown and give raise to two proof obligations for an Event-B Model.

Condition 3 relates the transitions of Event-B Model and Modal System. Condition 3a states that any transition in the Event-B Model is a possible transition of the Modal System (3(a)i or 3(a)ii). This can be shown on the structure of events. Each event of the model, whenever enabled in a mode, will either: preserve the modes assumption and guarantee in case of 3(a)i or switch mode according to existing mode switching transition in case of 3(a)ii. The following proof obligation has to be discharged to cover this condition. It considers all events of the Event-B model and all modes of the Modal System.

Definition 9 (PO: EBModel defines valid MSys Transitions 1).

$$\begin{array}{l}\forall\ E_i = (H_i, S_i) \in E_{EBModel}, \forall\ M_j = A_j/G_j \in M_{MSys} \cdot \\ \quad (H_i(v) \wedge A_j(v) \wedge S(v, v')) \Rightarrow \\ \qquad (\ (A_j(v') \wedge G_j(v, v')) \vee \\ \qquad\quad ((\exists M_k = A_k/G_k \in M_{MSys} \cdot A_k(v')) \wedge (j \rightsquigarrow k) \in T_{MSys})\ \ )\end{array}$$

If any pair of event guard and mode assumption may be both enabled (2nd line), the event is possible in that mode. In this case the event either describes an internal transition (3rd line) or a mode transition (4th line). In the first case, both assumption and guarantee of the current mode have to be preserved by the event. In the second case, the modal system specifies the possibility of such transition and the event establishes the new assumption. Since our mode definition allows the invariant to be weaker than the conjunction of assumptions, it is needed to show that any event is enabled only when an assumption is, otherwise the event is specifying some behaviour not matching any mode definition.

Definition 10 (PO: EBModel defines valid MSys Transitions 2).

$$\forall\ E_i = (H_i, S_i) \in E_{EBModel}, H_i(v) \Rightarrow \bigvee_{\forall M_j = A_j/G_j \in M_{MSys}} A_j(v)$$

Condition 3b states that all defined mode switching transitions have a corresponding event in the Event-B model. The following proof obligation is complementary to the one in definition 9:

Definition 11 (PO: Modal System Switching Transitions are Implemented in the Event-B Model Events).

$$\begin{array}{l}\forall\ (i \rightsquigarrow j) \in T_{MSys}, M_i = A_i/G_i, M_j = A_j/G_j \in M_{MSys} \cdot \\ \qquad \exists E_k = (H_k, S_k) \in E_{EBModel} \wedge (H_k(v) \wedge A_i(v) \wedge S_k(v, v') \wedge A_j(v'))\end{array}$$

Condition 4a states that all possible states of the Event-B model are valid states of the Modal System. Assuming that the model is in a valid state by initialization (condition 2), condition 4a is obtained if transitions are sound, according to POs in definitions 9 and 10.

Reachability Properties. To complete condition 3b, it has to be shown that each mode transition in the Modal System behaviour is possible in the Event-B model behaviour. This would rule out the possibility of the guard of an event that implements a mode switching never becoming enabled although the respective mode assumption could repeatedly become enabled (a problem with the model). Condition 4b states that any mode in the Modal System has at least one corresponding state in the Event-B Model. Proof obligations to discharge such properties could not be generated in general. The modeller has thus to account for them. This could be done either structuring a model such that these properties can be proven or using additional analysis techniques such as model checking.

6 Cruise Control Case Study

The Cruise Control case study illustrates the proposed technique to the development of a simplified version of one of the DEPLOY case studies [5]. The system assists a driver in reaching and maintaining some predefined speed. In the current modelling we assume an idealised car and idealised driving conditions such that the car always responds to the commands and the actual speed is updated according to the control system commands.

Figure 1 presents the diagrams of the most abstract modal system for the cruise control (A) and the resulting models of three successive refinement steps (B to D). The assumption and guarantee for each mode is given in Figure 2. The diagrams use a visual notation loosely based on Modecharts [6]. A mode is represented by a box with mode name; a mode transition is an arrow connecting two modes. The direction of an arrow indicates the previous and next modes in a transition. Special modes $\top_M$ and $\bot_M$ are omitted so that initiating and terminating transitions appear connected with a single mode. Refinement is expressed by nesting boxes. A refined diagram with an outgoing arrow from an abstract mode is equivalent to outgoing arrows from each of the concrete modes.

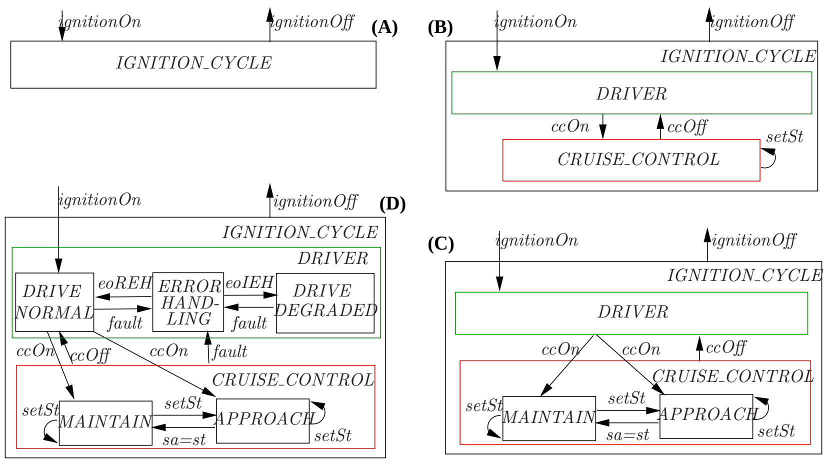

Fig. 1. Cruise control refinement steps (A) to (D).

mode	assumption	guarantee
IGNITION_CYCLE	ignition is on	keep speed under limit and (ac/de)celarate safely
	$ig = true$	$(sa < speedLimit) \wedge (\lvert sa' - sa\rvert < maxSpeedV)$
DRIVER	ignition cycle assumption and cruise control off	ignition cycle guarantee
	$ig = true \wedge cc = false$	$(sa < speedLimit) \ \wedge (\lvert sa' - sa\rvert < maxSpeedV)$
CRUISE_CONTROL	ignition cycle assumption and cruise control on	ignition cycle guarantee and maintain or approach target speed or
	$ig = true \wedge cc = true$	$(sa < speedLimit) \wedge (\lvert sa' - sa\rvert < maxSpeedV) \wedge (\lvert sa' - st'\rvert \leq isp \vee \lvert sa' - st'\rvert < \lvert sa - st\rvert)$
APPROACH	cruise control assumption and speed not close to target	cruise control guarantee and approach target speed
	$ig = true \ \wedge cc = true \ \wedge \lvert sa' - st'\rvert > isp$	$(sa < speedLimit) \wedge (\lvert sa' - sa\rvert < maxSpeedV) \wedge (\lvert sa' - st'\rvert < \lvert sa - st\rvert)$
MAINTAIN	cruise control assumption and speed close to target	cruise control guarantee and maintain target speed
	$ig = true \ \wedge cc = true \ \wedge \lvert sa' - st'\rvert \leq isp$	$(sa < speedLimit) \wedge (\lvert sa' - sa\rvert < maxSpeedV) \wedge (\lvert sa' - st'\rvert \leq isp)$
DRIVE_NORMAL	driver assumption and and no error	driver guarantee (and cruise control available)
	$ig = true \wedge cc = false \wedge error = false$	$(sa < speedLimit) \ \wedge (\ \lvert sa' - sa\rvert < maxSpeedV \ )$
ERROR_HAND-	driver assumption and error and handling not finished	driver guarantee and recovery measures (and cruise control not available)
	$ig = true \wedge cc = false \wedge error = true \wedge eHand = true$	$(sa < speedLimit) \ \wedge (\ \lvert sa' - sa\rvert < maxSpeedV \ )$
DRIVE_DEGRADED	driver assumption and error and handling finished	driver guarantee (and cruise control not available)
	$ig = true \wedge cc = false \wedge error = true \wedge eHand = false$	$(sa < speedLimit) \ \wedge (\ \lvert sa' - sa\rvert < maxSpeedV \ )$

Fig. 2. Modes assumptions and guarantees.

At the most abstract level (Figure 1(A)) we introduce mode *IGNITION_CYCLE* to represent the activity from the instant the ignition is turned on to the instant it is turned off, represented by transitions *ignitionOn* and $ignitionOff$. During an ignition cycle, its guarantee must be respected independently of operation by the driver or by the cruise control. The model includes: the state of ignition (on/off) modelled by a boolean flag *ig*; the current speed of the car (a modelling approximation of an actual car speed), stored in variable *sa*; a safe speed limit *speedLimit* above which the car should not be; and a safe speed variation $maxSpeedV$. No memory is retained about states in the previous ignition cycle.

In the first refinement step (Figure 1(B)) *IGNITION_CYCLE* is refined in *DRIVER* corresponding to the activity when cruise control is off and *CRUISE_CONTROL* when cruise control is active. *on/off* interface buttons to activate/deactivate the cruise control are mapped to transition events *ccOn* and $ccOff$. This refinement introduces: the state of cruise control (on/off), modelled by boolean flag *cc*; the target speed that a cruise control is to achieve and maintain, represented by variable *st*; an allowance interval *isp* that determines how much actual speed could deviate from a target speed. The next refinement step (Figure 1(C)) introduces different operating strategies: if the difference between current (*sa*) and target (*st*) speeds is within an acceptable error interval (*isp*), the cruise control works to *MAINTAIN* the current speed. Otherwise, it employs different procedures to *APPROACH* the target speed. Switching from *DRIVER* to *CRUISE_CONTROL* may either establish the assumptions of *APPROACH* or *MAINTAIN*, depending on the difference between *st* and *sa*. In either of these two modes the cruise control can be switched off and the control returned to the driver.

At any time failures of the surrounding components (e.g. airbag activated, low energy in battery, etc.) may happen and are signalled to the cruise control system. In the presence of an error, the control is returned to the driver and handling measures are activated. Errors can be reversible or irreversible. After being handled, the first ones allow the cruise control to become available again; the irreversible ones cause the cruise control to become unavailable during the ignition cycle. According to the last refinement step (Figure 1(D)), when an error is detected it is registered in an *error* variable. If an error is signalled in any of the system modes, the system switches to *ERROR_HANDLING*, where control is with the driver. Eventually error handling reestablishes *DRIVE_NORMAL*, with full functionality available, or switches to *DRIVE_DEGRADED* mode where the cruise control is not available. Note that although the guarantees of these three concrete modes from *DRIVER* are the same, they have distinct mode transition possibilities: in modes *DRIVE_DEGRADED* and *ERROR_HANDLING* the cruise control cannot be turned on. After finishing error handling the system continues in either normal or degraded mode.

Once a modal system is sufficiently developed (but not necessarily finalised) one can start building an Event-B model implementing it. The static part of a model, such as variables and invariant is already elaborated to some degree in a modal system specification. These are simply copied into an initial Event-B machine. Next, one has to study a mode diagram to grasp the general architecture of a system: the modes and the mode transitions. It helps to begin such a study with the most abstract diagram as it gives the understanding of the relation between the system modes.

We present excerpts from an Event-B model realising the modal system developed for the case study. For the most detailed modal specification, we have the Event-B declaration of variables and invariant on the right. It is merely a result of mechanically translating definitions from the modal specification into the Event-B syntax. The referenced context *cc_ctx* contains declarations of sets and constants such $SPEED$ and $speedLimit$.

```
system cruisecontrol
sees cc_ctx
variables ig, cc, sa, st, error
invariant
   ig ∈ BOOL
   cc ∈ BOOL
   sa ∈ SPEED
   st ∈ SPEED
   st > 0
   error ∈ BOOL
   eHand ∈ BOOL
```

Initially, the invariant has no interesting statements relating to the safety properties of the system. This is because in a modal system safety properties are put into the guarantees of individual modes. However, once it comes to the verification of an Event-B model against a modal specification (see Definitions 9, 10 and 11), the proof obligations, derived from the condition that an event must satisfy a mode guarantee, would suggest additional invariants. Hence, the process of showing modes/Event-B consistency gradually adds more details into an Event-B model with each additional discharged proof obligation.

In Event-B an initialisation is a special event assigning initial values to all the model variables. While in a modal specification there is no explicit discussion of initialisation in terms of state computations, the conditions (see Definition 11) on all mode transitions originating at $\top_M$ result in a rather detailed characterisation of possible variable initialisations.

For the cruise control case study the initial state should satisfy the invariant and the assumption of the initiating mode 'Drive Normal' and thus the least constrained initialisation event has the form shown on the right.

```
initialisation
$ig := TRUE \parallel cc := FALSE \parallel$
$\mathbf{sa} :\in \mathbf{SPEED} \parallel st :\in \mathbb{N}_1 \parallel$
$error := FALSE \parallel eHand :: BOOL$
```

The non-deterministic initialisation of sa (car speed) should raise concerns as it contradicts our understanding that a car is initially stationary. There is, however, nothing the mode specification that tells this and it is one of those many details we have abstracted away in a mode specification. In this case we choose to strengthen the initialisation event and state that initially sa is zero. Obviously, such initialisation also satisfies the requirements to an event implementing initiating mode transitions. The counterpart of the initialisation event is an event halting the current ignition cycle. This is implemented with an event setting ig to $FALSE$: $ignition\_off$ = when $ig = TRUE$ then $ig := FALSE$ end

Let us now take a look at how a mode is implemented. There is no ready rule for generating events from a mode description. This is the part where a designer has the most freedom within the limits set by the assumption and guarantee of a mode. There is no limitation on the number of events realising a mode. As example, we have found it convenient to have two events for mode *Drive Normal Mode*, each responsible for either decrease or increase in vehicle speed.

```
$speed\_up$ = any $si$ where
        $si \in SPEED$
        $si < maxSpeedV$
        $sa + si < speedLimit$
        $ig = TRUE$
        $cc = FALSE$
    then
        $sa := sa + si$
    end
```

```
$speed\_down$ = any $sd$ where
        $sd \in SPEED$
        $sd < maxSpeedV$
        $sa - sd \in SPEED$
        $ig = TRUE$
        $cc = FALSE$
    then
        $sa := sa - sd$
    end
```

For each of the events we have to demonstrate that it is enabled only when the mode assumption holds (Definition 10). For event *speed_up* we have:

$$\forall si \cdot si \in SPEED \wedge si < maxSpeedV \wedge \tag{1}$$
$$sa + si < speedLimit \wedge ig = TRUE \wedge cc = FALSE \implies \tag{2}$$
$$ig = TRUE \wedge cc = FALSE \tag{3}$$

Also, it is required to show that it honors the mode guarantee (Definition 9) and reestablishes the mode assumption.

$$\forall si \cdot si \in SPEED \wedge si < maxSpeedV \wedge \tag{4}$$
$$sa + si < speedLimit \wedge ig = TRUE \wedge cc = FALSE \wedge \tag{5}$$

$$sa' = sa + si \wedge ig' = ig \wedge cc' = cc \wedge \tag{6}$$
$$st' = st \wedge error' = error \wedge eHand' = eHand \implies \tag{7}$$
$$ig' = TRUE \wedge cc' = FALSE \wedge \tag{8}$$
$$(sa' < speedLimit) \wedge |sa' - sa| < maxSpeedV \tag{9}$$

These proof obligations, being formulated as Event-B theorem (extra conditions on Event-B models), are automatically discharged by the Rodin platform theorem prover. This is also true for all of the rest of proof obligations, coming from modes and native to Event-B.

From our experience, the construction of an Event-B from a modal specification is a fairly straightforward process. However, we have also found that, for the few initial development steps, constructing an Event-B model for each step of a modal system refinement makes little impact on understanding the system. This because a mode specification embodies basically the same information (albeit in a structured manner) as an abstract Event-B model.

On the other hand, in absence of a dedicated tool support for checking modal specifications, an Event-B implementation provides a verification platform in the form of the Rodin toolkit. This also defines how we see the application of the approach. A developer would start with translating requirements into a high-level modal specification. More requirements are captured by refining modes and, at some point, an Event-B model is constructed. For several further steps, modal and Event-B developments go hand-in-hand until no further detalisation can be done at the level of a modal specification. This would mark the final transition into an Event-B model. However, even at that point a modal specification is not forgotten. The consistency conditions proved at an earlier refinement are preserved through a refinement chain and thus, even after several refinement steps, an Event-B model still respects all the properties of a modal specification from which it was initially derived.

7 Conclusions

A representative class of critical systems employs the notion of operational modes. While this notion is supported in some languages [6,7], a formal definition for modal systems as well as approaches for their rigorous construction could not be found. Following previous work [9], in this paper we formalize modal systems and modal systems refinement. The use of modes and modal system refinement helps to organize system properties, to trace requirements into model definition and helps to impose control structure in the system. Such advantages are specially welcomed together with a state-based formal method. As a further contribution of this paper we take Event-B and show how to demonstrate that a model in Event-B is according to a modal system, i.e. respecting assumptions, guarantees and mode switchings.

Using modal systems refinement and the notion of modal system consistency for an Event-B model, both defined in this paper, together with the common Event-B refinement notion, it is possible to build a concrete Event-B model

$EBModel_C$ refining an $EBModel_A$ and show that it satisfies an $MSys_C$ which refines $MSys_A$. A natural extension of this work is to formally define restrictions on the refinement starting from $EBModel_A$ leading to $EBModel_C$ which by construction satisfies $MSys_C$. Such restrictions would be based on the refinement from $MSys_A$ to $MSys_C$. Additionally, in future work we intend to investigate the implications of mode concurrency.

References

1. R. W. Butler, "Nasa technical memorandum 110255 an introduction to requirements capture using pvs: Specification of a simple autopilot," 1996.
2. S. P. Miller, "Specifying the mode logic of a flight guidance system in core and scr," in *FMSP '98: Proceedings of the second workshop on Formal methods in software practice.* New York, NY, USA: ACM, 1998, pp. 44–53.
3. J. Lygeros, D. N. Godbole, and M. E. Broucke, "Design of an extended architecture for degraded modes of operation of ivhs," in *In American Control Conference*, 1995, pp. 3592–3596.
4. J.-R. Abrial, E. Börger, and H. Langmaack, Eds., *Formal Methods for Industrial Applications, Specifying and Programming the Steam Boiler Control (the book grow out of a Dagstuhl Seminar, June 1995)*, ser. Lecture Notes in Computer Science, vol. 1165. Springer, 1996.
5. J.-R. Abrial, J. Bryans, M. Butler, J. Falampin, T. S. Hoang, D. Ilic, T. Latvala, C. Rossa, A. Roth, and K. Varpaaniemi, "Report on knowledge transfer - deploy deliverable d5," February 2009.
6. F. Jahanian and A. Mok, "Modechart: A specification language for real-time systems," *IEEE Transactions on Software Engineering*, vol. 20, no. 12, pp. 933–947, 1994.
7. J. J. H. Peter H. Feiler, David P. Gluch, "The architecture analysis & design language (aadl): An introduction," Software Engineering Institute - Carnegie Mellon University, Technical Note CMU/SEI-2006-TN-011, 2006.
8. S. Mustafiz, J. Kienzle, and A. Berlizev, "Addressing degraded service outcomes and exceptional modes of operation in behavioural models," in *SERENE '08: Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems.* New York, NY, USA: ACM, 2008, pp. 19–28.
9. A. Iliasov, F. L. Dotti, and A. Romanovsky, "Structuring specifications with modes," School of Computing Science - Newcastle University, Technical Report CS-TR No 1143, 2009, submitted to LADC 2009.
10. R.-J. J. Back and J. V. Wright, *Refinement Calculus: A Systematic Introduction.* Springer-Verlag New York, Inc., 1998.
11. J. R. Abrial, *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 2005.
12. R.-J. Back and K. Sere, "Stepwise Refinement of Action Systems," in *Proceedings of the International Conference on Mathematics of Program Construction, 375th Anniversary of the Groningen University*, J. L. A. van de Snepscheut, Ed. London, UK: Springer-Verlag, 1989, pp. 115–138.
13. E. Dijkstra, *A Discipline of Programming.* Prentice-Hall International, 1976.
14. "Event-b and the rodin platform," http://www.event-b.org/ (last accessed 8 March 2009). Rodin Development is supported by European Union ICT Projects DEPLOY (2008 to 2012) and RODIN (2004 to 2007).