Modal Systems: Specification, Refinement and Realisation*

Fernando L. Dotti[1,2], Alexei Iliasov[1],
Leila Ribeiro[3], 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
[3] Instituto de Informática, UFRGS, Brazil
leila@inf.ufrgs.br

Abstract. Operation modes are useful structuring units that facilitate design of several safety-critical systems such as such as avionic, transportation and space systems. Although some support to the construction of modal systems can be found in the literature, modelling abstractions for the formal specification, analysis and correct construction of modal systems are still lacking.
This paper discusses existing support for the construction of modal systems and proposes both a formalisation and a refinement notion for modal systems. A modal system, specified using the proposed abstractions, can be realised using different specification languages. Complementing the contribution, we define the requirements for an Event-B model to realise a modal system specification. A case study illustrates the proposed approach.

1 Introduction

Several systems, many of them safety-critical ones, are modal, i.e., they are described using the notion of 'operation modes'. While there is no widely accepted definition of operation mode and modal system, several authors use operation modes to denote the expected system functionality under distinguished working conditions of the system. A modal system denotes the assembly of a set of such modes[4], related by mode transitions that represent the possible changes in the working conditions of a system, originated either by environmental changes or by system evolution.

Given the importance of modal systems, several abstractions for their modelling are provided. Examples of modal systems and a brief survey of modelling approaches are discussed in Section 2. That efforts not withstanding, there is

* 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. Leila Ribeiro is supported by CNPq/Brazil grant 200779/2008-7.

[4] 'Mode' and 'operation mode' are used as synonyms.

a lack of abstraction to allow the formal specification of modal systems as well as approaches to analysing and rigorously deriving implementations from these systems.

In our previous work [1], we introduced modal systems and discussed their use for structuring dependable systems (focusing specifically on the recovery and degradation modes).

As one contribution, this paper presents the formal definitions of the abstractions used to specify modal systems. According to our approach, a modal system is an abstract specification of the modes as well as mode transitions that may occur in a system. It does not specify concretely how the system operates while it is in some specific mode nor how mode transitions occur. It rather imposes requirements on concrete implementations, complementing traditional modelling but not replacing it. The construction of a concrete model that realizes a modal system specification can be done using any existing formalism. Therefore it is important to define when a concrete model realizes a modal system.

Event-B [2] is a state-based formal method closely related to Classical B [3]. It has been successfully used in several applications, having available tool support for both model specification and analysis. Another contribution of the paper are the satisfaction conditions stating when an Event-B model realises a modal system. A series of proof obligations on the Event-B model are derived from the modal system to show its satisfaction. The realisation of a modal system using state-based formal methods is especially interesting since the modal system helps to structure state-based systems.

A final contribution of this paper is the formalization of the notion of modal system refinement. A modal system can be step-wise detailed to model system requirements, allowing an organized way to construct the system and reason about its properties.

The rest of the paper is structured in 4 major parts: first, related work is surveyed in Section 2; second, modal systems and modal system refinement are formally defined in Sections 3 and 4; then Event-B is briefly introduced and its relation to modal systems defined in Sections 5 and 6; finally, Section 7 presents a case study to illustrate and evaluate the ideas introduced in the paper. Section 8 concludes the paper with a summary and an outline of future extensions of the approach.

2 Related Work

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 [4,5] the authors specify and analyse the operation mode logics of space and avionic systems. In both avionics and space systems, modes denote phases of a flight and operational status of on-board instruments, among others. An extension of an Automated Highway System with degraded operation modes that tolerates several kinds of faults is discussed in [6]. The Steam Boiler Control [7], 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 [8].

The use of operation modes is very common in real-time systems. Timing properties are analysed considering operation modes of the system: deadlines to enter or leave modes, or to perform mode changes are investigated. Modecharts [9] focus on the specification of real-time properties of mode and mode switching. The authors propose modes both as partitions of the state space, representing different working conditions of the system, as well as a way to define control information in large state machines, imposing structure on the operation of the system. However, Modecharts lacks adequate support to specifying and reason about functional properties.

According to [10], also focused on real-time systems, a mode is characterized by a group of tasks. The system initiates in a given mode, with a specific set of active tasks. During system evolution or to react to external stimuli, the system may change modes. This involves stopping and starting tasks, keeping or changing parameters of tasks. The approach discusses real-time systems and how to meet deadlines. In such systems, a mode change may result in transient stages where tasks are deleted/created. Mode changes take time but are atomic - not iterrupted by other mode changes. The problem of meeting deadlines during mode changes in real-time systems is also addressed in [11].

Since the notion of operation modes is rather generic and many modal systems are critical, modelling abstractions to support modal system definitions appeared recently, which are not focused on real-time aspects. In the Architecture Analysis & Design Language (AADL) [12] a system is built out of communicating components and each component may have modes, representing alternative operational states. Modes serve to identify configurations of components. A state machine abstraction is used, such that a distinct configuration is a modal state and specific events cause transition among them. A component may have distinct behaviour according to the current mode.

The Dependability Requirement Engineering Process (DREP) [13] provides a methodology for developing modal systems using UML diagrams. A system is considered to offer a set of services and, depending on the mode, different subsets of services may be available. Switching modes means to change the configuration of the system such that the profile of offered services changes. Since DREP is specially focused on the representation of degraded service outcomes, some modes are specially discussed: normal, degraded, emergency and restricted. The use of modes to represent degraded system operation is commonly found in the literature [14].

According to our literature survey, none of the existing approaches discusses refinement of modal systems. Moreover, most of them are not formal, and thus can neither be used as a basis for formal development nor to check whether possible realisations are correct.

3 Modal Systems

As mentioned in the Introduction and observed in Section 2, operation modes are generally used to denote expected system functionality under distinguished working conditions of the system. A modal system consists of the assembly of a set of such modes related by mode transitions. As a brief example, the Steam Boiler Control [7] 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 7 discusses a cruise control system with several modes, each organized in similar way and related by transitions.

We are interested in how to specify, analyse and build correct modal systems. Instead of proposing a specification method, our approach is to provide abstractions that allow to formally specify the requirements of modal systems towards concrete models that realize them. A modal system specification is a complementary view on the system that does not replace traditional formal modelling. The construction of a concrete model that realises 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 further discussed in Section 6.

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 $State \times State$. We assume that there is a special value called $Undef$ in Val, and the undefined state (a state in which all values are mapped to $Undef$) is called $Undef$.*

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, a Mode is a pair A/G where A is an assumption, G is a guarantee and:*

- *the 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 mode's 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: 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.

Definition 3 (Mode Transition, Modal System). *Given a set of modes M, a transition t is a pair (i, j), with $i, j \in M$. A transition is denoted by $i \rightsquigarrow j$, and the source i and target j modes of a transition t are denoted by $src(t)$ and $target(t)$, respectively.*

A Modal System is a tuple $MSys = (Var, Val, I, M, T)$ where:

1. *Var is a set of variables of the system;*
2. *Val is the set of possible values for variables;*
3. *I is an invariant;*
4. *M is a finite set of modes $(M_i = A_i/G_i)_{i \leq n, n \in \mathbb{N}} \cup \{\top_M, \bot_M\}$ such that*
   (a) $I(v) \Rightarrow A_1(v) \oplus \cdots \oplus A_n(v)$, where $\oplus$ is the exclusive or 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.
5. *$T \subseteq M \times M$ 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) $\forall m \in M - \{\bot_M\} \cdot (\top_M, m) \in T^*$, where T^* is the transitive closure of T.

A modal system is an assembly of several modes (M) related by mode transitions (T). Modes $\top_M$ and $\bot_M$ are called start and terminal modes, respectively. It is assumed that a system is only in one mode at a time, represented by condition 4a. The meaning and implications of a system being simultaneously in more than one mode are not trivial and subject of further study. A mode transition is an atomic step switching from one source mode i to one destination mode j. The possible mode transitions of a modal system are defined by T. According to condition 5b, the start mode $\top_M$ 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 will be no mode transition to the terminal mode $\bot_M$. Condition 5a states that it is not possible to switch to a state before initialization or from the terminal mode to another mode; and during its lifetime a system enters at least one operation mode. Now we define the behavior of a modal system.

Definition 4 (Modal System Behaviour). *The behaviour of a modal system $MSys = (Var, Val, I, M, T)$, given by a transition system $MST = (MState, S_0, \rightarrow)$ where $MState = \{\langle m, v\rangle \mid m \in \{1..n, \top_M, \bot_M\}$ is a mode index $\wedge$ v is a state$\}$; the initial state S_0 is $\langle \top_M, Undef\rangle$ and the transition relation $\rightarrow$: $MState \rightarrow MState$ is given by the rules:*

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

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

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

The state of a system described using operation modes is a tuple $\langle m, v\rangle$ where m is the index of a current operation mode and v is the current system state. In the following, each of the transition rules is explained.

Initialisation. A system starts executing one of its initialization 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 mode switching transitions. Rule *internal* states that while the system is in some mode m the state may evolve to a state v' satisfying both the corresponding guarantee $G_m(v, v')$ and the modes assumption $A_m(v')$. Rule *switching* states that the system may switch modes if there is a defined mode transition originating from the current mode. *Internal* and *switching* transitions compete with each other: at each step a non-deterministic choice is made among the enabled transitions.

Termination. A system terminates by executing one of terminating mode transition $t \rightsquigarrow \bot_M$. 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 5a, no mode transitions are possible after $\bot_M$ is reached

4 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 modes at the concrete level. 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 at an abstract model must have at least one corresponding transition at a concrete model. If a source mode of 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* $MSys_{abs} = (Var_{abs}, Val_{abs}, I_{abs}, M_{abs}, T_{abs})$*; and*
- *a modal system* $MSys_{cnc} = (Var_{cnc}, Val_{cnc}, I_{cnc}, M_{cnc}, T_{cnc})$

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

1. ref^M *is total, surjective and preserves the start and terminal modes; 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) considering variables in the abstract system, concrete modes cover the same state space as the abstract one – it is not possible to restrict assumptions by refinement; (3) guarantees of the concrete system may be stronger (more deterministic) than the corresponding abstract ones; (4) if a transitions is added in a refinement step, it must have a non-observable effect on the abstract level (same source and target modes); (5) the transitions that are mapped to the abstract level (those in $dom(T_{cnc})$) must be consistent with the mapping of source and target modes.

Via data refinement, the set v of model variables may change to a new set u and model invariant $I(v)$ is replaced with a new invariant $J(v, u)$, often called a *gluing invariant*. The presence of old variables v in new invariant J allows a modeller to express a linking relation between the states 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 switching from m_{c1} to m_{c2}, according to the case above, does not correspond to a mode change at the abstract level because m_{c1} and m_{c2} refine the same m_a. Thus, the transitions of m_{c1} and m_{c2} correspond 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. $\square$

5 Event-B

Event-B [2] is a state-based formalism closely related to Classical B [3] and Action Systems [15].

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*[5]*; $P(c, s)$ 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 between 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. The model *consistency* condition states that whenever an event on an initialisation action is attempted, there exists a suitable new state v' such that the 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 the transition system defined 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 $BST = (BState, BS_0, \rightarrow)$ where: $BState = \{\langle v \rangle | v \text{ is a state}\} \cup Undef$, $BS_0 = Undef$, and $\rightarrow \subseteq BState \times BState$ is the transition relation given by the rules:*

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

[5] For convenience, as in [3], no distinction is made between a set of variables and a state of a system.

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

According to rule *start* the model is 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 applies. The semantics of an Event-B model is given in the form of proof semantics, based on Dijkstra's work on weakest preconditions [16].

To refine model M one constructs a new model M' that is behaviourally equivalent to the old one. In Event-B, this is achieved by constructing a refinement mapping between M' and M and by discharging a number of refinement proof obligations.

An extensive tool support through the Rodin Platform makes Event-B especially attractiveA development environment for Event-B is supported. 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, but there is also some support for model checking[6].

6 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* $EBModel = (c_E, s_E, P_E, v_E, I_E, R_{I_E}, E_E)$ *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 the modal system state;*
- *a predicate projection function* $fp_{EtoM}(P_E) = P_M$ *that, given a predicate over* v_E *constructs the corresponding predicate over* Var_M*;*

$EBModel$ *satisfies* $MSys$ *iff:*

[6] See Rodin Platform http://www.event-b.org/ (last accessed September 21st 2009). Rodin Development is supported by European Union ICT Projects DEPLOY (2008 to 2012) and RODIN (2004 to 2007).

1. *both specify the same invariant on* Var_M: $fp_{EtoM}(I_E) = I_M$
2. *the initialisation is compatible, i.e. the initial state* $EBModel$ *is compatible with the assumption of any initial mode of* $MSys$:
   $fp_{EtoM}(R_{I_E}) \Rightarrow \bigvee_{\forall t \in T_M \cdot src(t) = \top_M} A_{target(t)}$
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* $MSys$*, where* $fs_{EtoM}(s1_E) = s1_M \wedge fs_{EtoM}(s2_E) = s2_M$
   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 transition* $t_M : \langle m1, s1_M \rangle \rightarrow \langle m2, s2_M \rangle$ *in the behaviour of* $MSys$ *has a corresponding transition* $t_E : s1_E \rightarrow s2_E$ *of the behaviour of* $EBModel$ *with* $fs_{EtoM}(s1_E) = s1_M \wedge fs_{EtoM}(s2_E) = s2_M$.

The following proposition states the compatibility between the computations of the mode system and an Event-B model that realises it, according to Def. 8. Note that this realisation if not just a refinement relation because, besides requiring that the event-B model does not introduce new behavior, it requires also that the event-B model exhibits all possible behaviors defined in the mode system being realised.

Proposition 2. *Given:*

- *a Modal System* $MSys = (Var_M, Val_M, I_M, M_M, T_M)$
  where $Var_M \subseteq v_E$*;*
- *an Event-B model* $EBModel = (c_E, s_E, P_E, v_E, I_E, R_{I_E}, E_E)$ *and*
- *function* fs_{EtoM} *and* fp_{EtoM} *as in Def. 8 ;*

any possible sequence in the transition system of $MSys$ *can be translated into a sequence described by the transition system of* $EBModel$ *and vice versa.*

Proof. Condition 1 of Def. 8 assures that the state space is the same (restricted to the variables of $MSys$).

First, we prove that given a transition sequence of $MSys$, we can generate a corresponding one for $EBModel$. Any sequence of $MSys$ must start with a transition generated by rule *start*, that generates a state in which the assumption of some initial mode A_k is true. Since this state is also possible in the $EBModel$ (because the state spaces are the same) and assumptions of a mode system are disjoint, condition 2 of Def. 8 ensures that there must be a transition generated by rule *startEvent* that leads to this state. From there on, 3b of Def. 8 guarantees that there is a corresponding transition in the transition system of $EBModel$ for each transition of $MSys$.

The proof of the other direction (given a transition sequence of $EBModel$, a corresponding one for $MSsys$ can be found) is analogous, using 3a of Def. 8. □

Based on these consistency conditions, we now define the proof obligations that are necessary to discharge to show that an Event-B model satisfies a mode system. The first two proof obligations correspond to the first two consistency conditions. The other 3 are necessary to ensure condition 3.

Definition 9 (Proof Obligations).

PO1 (Invariant compatibility) *:*

$$fp_{EtoM}(I_E) = I_M$$

PO2 (Initial state compatibility) *:*

$$fp_{EtoM}(R_{I_E}) \Rightarrow \bigvee_{\forall t \in T_M \cdot src(t) = \top_M} A_{target(t)}$$

PO3 (Events/Modes compatibility) *:*

$$\begin{aligned}
&\forall\ E_i = (H_i, S_i) \in E_{EBModel} \cdot \forall\ M_j = A_j/G_j \in M_{MSys} \cdot \\
&\quad (H_i(v) \wedge A_j(v) \wedge S(v, v')) \Rightarrow && (1)\\
&\qquad (\ (A_j(v') \wedge G_j(v, v'))\vee && (2)\\
&\qquad\quad ((\exists M_k = A_k/G_k \in M_{MSys} \cdot A_k(v')) \wedge (j \rightsquigarrow k) \in T_{MSys})\ \ ) && (3)
\end{aligned}$$

PO4 (Event guard/Mode assumption compatibility) *:*

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

PO5 (Events/Transitions compatibility) *:*

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

Condition 3 of Def. 8 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. Proof obligation **PO3** has to be discharged to cover this condition. If an event guard and a mode assumption are true (line 1), the event is possible in that mode. In this case the event either describes an internal transition (line 2) or a mode transition (line 3). 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 that does nor match any mode definition. This is assured by discharging proof obligation **PO4**.

Condition 3b of Def. 8 states that all defined mode switching transitions have a corresponding event in the Event-B model. The corresponding proof obligation is **PO5**.

Reachability Properties. To completely assure condition 3b, it has to be shown, additionally to the given proof obligations, that each mode transition in the Modal System behaviour is possible in the Event-B model behaviour. Proof obligations to discharge such properties can not be generated in general, they are specific for each model. They can be assured either by structuring a model such that these properties can be proven or by using additional analysis techniques such as model checking.

7 Cruise Control Case Study

The Case Study is presented in the following parts: first we exemplify modelling with modes; then we discuss aspects of building an Event-B model to realise a modes specification; thirdly we exemplify proof obligations on the case study, and then general comments are made.

7.1 Modelling with Modes

The Cruise Control case study illustrates the proposed technique to the development of a simplified version of one of the DEPLOY case studies [8]. 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 Modechards [9]. 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.

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.

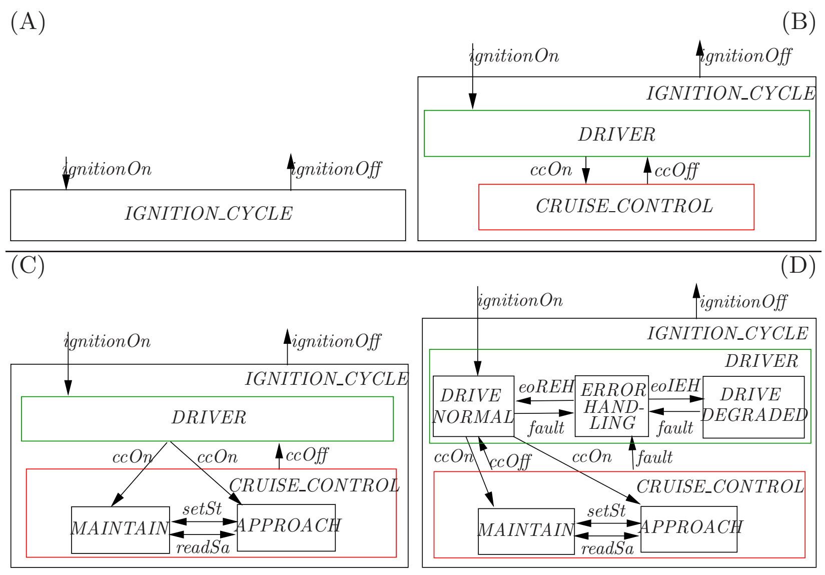

Fig. 1. Cruise control refinement steps (A) to (D).

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

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.

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.

7.2 Building an Event-B model to realise the modal system

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*.

```
machine 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 the proof obligations (see Def. 9), 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 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 ∥ cc := FALSE ∥
sa :∈ SPEED ∥ st :∈ ℕ₁ ∥
error := FALSE ∥ 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 $FALSE$:

$$\mathit{ignition\_off} = \texttt{when } ig = TRUE \texttt{ then } ig := FALSE \texttt{ end}$$

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

7.3 Examples of proof obligations generated from the modal system

Now we exemplify the application of the proof obligations in Def. 9 to the case study.

PO1 is discharged trivially because the Event-B model has the same variable definitions of the modal system and the same invariant.

PO2 reduces to prove that the initialization implies the assumption of the initial mode which is *Drive Normal Mode*.

According to **PO4**, for each of the events we have to demonstrate that it is enabled only when the mode assumption holds. For instance, for event *speed_up* we have:

$$\begin{aligned}\forall si \cdot si \in SPEED \wedge si < maxSpeedV \wedge \\ sa + si < speedLimit \wedge ig = TRUE \wedge cc = FALSE \implies \\ ig = TRUE \wedge cc = FALSE\end{aligned}$$

According to **PO3**, it is required to show for each event that it either respects the mode guarantee (Def. 9, **PO3**, line 2) or that it switches to another mode according to a possible mode transition (line 3). Below we show the proof to event *speed_up* in respecting guarantee of *Drive Normal Mode*.

$$\begin{aligned}\forall si \cdot si \in SPEED \wedge si < maxSpeedV \wedge \\ sa + si < speedLimit \wedge ig = TRUE \wedge cc = FALSE \wedge \\ sa' = sa + si \wedge ig' = ig \wedge cc' = cc \wedge \\ st' = st \wedge error' = error \wedge eHand' = eHand \implies \\ ig' = TRUE \wedge cc' = FALSE \wedge \\ (sa' < speedLimit) \wedge |sa' - sa| < maxSpeedV\end{aligned}$$

According to **PO5**, for each transition there must be at least one event implementing it. Event $ignition\_off$, for instance, is shown to implement transitions from any mode to $\perp_M$.

The 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 the rest of proof obligations, coming from modes and native to Event-B.

7.4 General comments on the Case Study

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.

8 Conclusions

A representative class of critical systems employs the notion of operation modes. While this notion is supported in some languages [9,12], a formal definition for modal systems as well as approaches for their rigorous construction could not be found. Following previous work [1], 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. Although the satisfaction conditions were shown for Event-B, the same ideas can be generalised to other formal methods.

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.

Acknowledgements

We would like to thank John Fitzgerald for the valuable comments made on earlier versions of this work.

References

1. Iliasov, A., Dotti, F.L., Romanovsky, A.: Structuring specifications with modes. In: Proceedings of the fourth Latin-American Symposium on Dependable Computing, Los Alamitos, CA, USA, IEEE Computer Society (2009) 81–88
2. Abrial, J.R., Métayer, C.: Rodin deliverable 3.2 - event-b language. Technical report, Newcastle University, England (2005) http://rodin.cs.ncl.ac.uk.
3. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (2005)
4. Butler, R.W.: Nasa technical memorandum 110255 an introduction to requirements capture using pvs: Specification of a simple autopilot (1996)
5. Miller, S.P.: 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) 44–53
6. Lygeros, J., Godbole, D.N., Broucke, M.E.: Design of an extended architecture for degraded modes of operation of ivhs. In: In American Control Conference. (1995) 3592–3596
7. Abrial, J.R., Börger, E., Langmaack, H., eds.: Formal Methods for Industrial Applications, Specifying and Programming the Steam Boiler Control (the book grow out of a Dagstuhl Seminar, June 1995). In Abrial, J.R., Börger, E., Langmaack, H., eds.: Formal Methods for Industrial Applications. Volume 1165 of Lecture Notes in Computer Science., Springer (1996)
8. Abrial, J.R., Bryans, J., Butler, M., Falampin, J., Hoang, T.S., Ilic, D., Latvala, T., Rossa, C., Roth, A., Varpaaniemi, K.: Report on knowledge transfer - deploy deliverable d5 (February 2009)
9. Jahanian, F., Mok, A.: Modechart: A specification language for real-time systems. IEEE Transactions on Software Engineering **20**(12) (1994) 933–947
10. Real, J., Crespo, A.: Mode change protocols for real-time systems: A survey and a new proposal. Real-Time Syst. **26**(2) (2004) 161–197
11. Fohler, G.: Realizing changes of operational modes with a pre run-time scheduled hard real-time system. In: In Proceedings of the Second International Workshop on Responsive Computer Systems, Springer Verlag (1992) 287–300

12. Peter H. Feiler, David P. Gluch, J.J.H.: The architecture analysis & design language (aadl): An introduction. Technical Note CMU/SEI-2006-TN-011, Software Engineering Institute - Carnegie Mellon University (2006)
13. Mustafiz, S., Kienzle, J., Berlizev, A.: 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) 19–28
14. Robert, T., Fabre, J.C., Roy, M.: Application of Early Error Detection for Handling Degraded Modes of Operation. In Hélène WAESELYNCK, ed.: Proceedings of the 12th European Workshop on Dependable Computing, EWDC 2009 12th European Workshop on Dependable Computing, EWDC 2009, Toulouse France (05 2009) 3 pages Rapport LAAS n° 09171.
15. Back, R.J., Sere, K.: Stepwise Refinement of Action Systems. In van de Snepscheut, J.L.A., ed.: Proceedings of the International Conference on Mathematics of Program Construction, 375th Anniversary of the Groningen University, London, UK, Springer-Verlag (1989) 115–138
16. Dijkstra, E.: A Discipline of Programming. Prentice-Hall International (1976)