Appendix A
An Introduction to the Event-B Modelling Method

Thai Son Hoang

Abstract This appendix is a short introduction to the Event-B modelling method for discrete transition systems. Important mechanisms for the step-wise development of formal models, such as *context extension* and *machine refinement*, are discussed. Consistency of the models is presented in terms of proof obligations and illustrated with concrete examples.

A.1 Introduction

Event-B [2] is a modelling method for formalising and developing systems whose components can be modelled as discrete transition systems. An evolution of the (classical) B-method [1], Event-B is now centred around the general notion of *events*, which are also found in other formal methods such as Action Systems [4], TLA [6] and UNITY [5].

Event-B models are organised in terms of two basic constructs: *contexts* and *machines*. Contexts specify the static part of a model, whereas machines specify the dynamic part. The role of the contexts is to isolate the parameters of a formal model and their properties, which are assumed to hold for all instances. A machine encapsulates a transition system with the state specified by a set of variables and transitions modelled by a set of guarded events.

Event-B allows models to be developed gradually via mechanisms such as *context extension* and *machine refinement*. These techniques enable users to develop target systems from their abstract specifications, and subsequently introduce more implementation details. More importantly, properties that are proved at the abstract level are maintained through refinement, and hence are also guaranteed to be satisfied by later refinements. As a result, correctness proofs of systems are broken down and distributed amongst different levels of abstraction, which is easier to manage.

The rest of this appendix is structured as follows. We give a brief overview of the Event-B mathematical language in Sect. A.2. In Sect. A.3, we give an informal description of our running example. Subsequently, we show the basic constructs of

T.S. Hoang (✉)
Institute of Information Security, ETH Zurich, Zurich, Switzerland
e-mail: htson@inf.ethz.ch

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1,

Table A.1 Definitions

Construct	Definition
$r \in S \leftrightarrow T$	$r \in \mathbb{P}(S \times T)$
$f \in S \nrightarrow T$	$f \in S \leftrightarrow T \land (\forall x, y_1, y_2 \cdot x \mapsto y_1 \in f \land x \mapsto y_2 \in f \Rightarrow y_1 = y_2)$
$f \in S \rightarrow T$	$f \in S \nrightarrow T \land (\forall x \cdot x \in S \Rightarrow (\exists y \cdot x \mapsto y \in f))$

Event-B in Sects. A.4 (contexts) and A.5 (machines). We present the mechanisms for context extension in Sect. A.6 and machine refinement in Sect. A.7.

A.2 The Event-B Mathematical Language

The basis for the formal models in Event-B is first-order logic and a typed set theory. We are not going to give full details of the Event-B logic here. For more information, we refer the reader to [2, 8]. We present several main elements of the Event-B mathematical language that are important for understanding the Event-B models of the example below.

The first-order logic of Event-B contains standard operators such as *conjunction* ($\land$), *disjunction* ($\lor$), *implication* ($\Rightarrow$), *negation* ($\neg$), *equivalence* ($\Leftrightarrow$), *universal quantification* ($\forall$), and *existential quantification* ($\exists$). Two constants are defined, namely $\top$ and $\bot$, denoting *truth* and *falsity*, respectively.

A.2.1 Set Theory

An important part of the mathematical language is its set-theoretical notation, with the introduction of the membership predicate $E \in S$, meaning that expression E is a member of set S. A set expression can be a variable (depending on its type). Moreover, a set can be explicitly defined by listing its members (*set extension*), e.g. $\{E_1, \ldots, E_n\}$. Other basic set constructs include Cartesian product, power set, and set comprehension. Given two set expressions S and T, the *Cartesian product* of S and T, denoted $S \times T$, is the set of mappings (ordered pairs) $x \mapsto y$, where $x \in S$ and $y \in T$. The *power set* of S, denoted $\mathbb{P}(S)$, is the set of all subsets of S. Finally, given a list of variables x, a predicate P constraining x and an expression E depending on x, the *set comprehension* $\{x \cdot P \mid E\}$ is the set of elements E where P holds.

A key feature of the Event-B set-theoretical notation is the models of relations as sets of mappings. Different types of relations and functions are also defined as sets of mappings with different additional properties. Given two set expressions S and T, $S \leftrightarrow T$ denotes the set of all *binary relations* from S to T. Similarly, $S \nrightarrow T$ denotes the set of all *partial functions* from S to T, and $S \rightarrow T$ denotes the set of all *total functions* from S to T. Definitions of these relations can be seen in Table A.1, expressed using set memberships.

Table A.2 Calculating well-definedness conditions using $\mathscr{L}$

Formula	Well-definedness condition
x	$\top$
$\neg P$	$\mathscr{L}(P)$
$P \wedge Q$	$\mathscr{L}(P) \wedge (P \Rightarrow \mathscr{L}(Q))$
$\forall x \cdot P$	$\forall x \cdot \mathscr{L}(P)$
$E_1 \div E_2$	$\mathscr{L}(E_1) \wedge \mathscr{L}(E_2) \wedge E_2 \neq 0$
$E_1 \leq E_2$	$\mathscr{L}(E_1) \wedge \mathscr{L}(E_2)$
$\text{card}(E)$	$\mathscr{L}(E) \wedge \text{finite}(E)$
$f(E)$	$\mathscr{L}(E) \wedge f \in S \nrightarrow T \wedge E \in \text{dom}(f)$

Intuitively, a binary relation r from S to T is a set of mappings $x \mapsto y$, where $x \in S$ and $y \in T$. A partial function f from S to T is a binary relation from S to T, where each element x in S has *at most* one mapping to T. A total function f from S to T is a partial function from S to T, where each element x in S has *exactly* one mapping to T.

A.2.2 Types

Variables in Event-B are strongly typed. A type in Event-B can be built-in (e.g., BOOL, $\mathbb{Z}$) or user-defined. Moreover, given types T, T_1 and T_2, the Cartesian product $T_1 \times T_2$ and the power set $\mathbb{P}(T)$ are also types. In contrast with most strongly typed programming languages, the types of variables in Event-B are not presented when they are declared. Instead, they are inferred from constraining properties of variables. Typically, a property of the form $x \in E$, where E is of type $\mathbb{P}(T)$, allows us to infer that x has type T.

A.2.3 Well-Definedness

Event-B requires every formula to be *well defined* [7, 8]. Informally, one has to prove that partial functions (either predefined, e.g. division $\div$, or user-defined) are never evaluated outside of their domain. Ill-defined expressions (such as $x \div 0$) should be avoided. A syntactic operator $\mathscr{L}$ is used to map formulae to their corresponding well-definedness conditions. Table A.2 shows the definition of well-definedness condition using $\mathscr{L}$ for formulae in Event-B. Here x is a variable, P and Q are predicates, E, E_1, E_2 are expressions, and f is a binary relation from S to T. Moreover, $\text{dom}(f)$ denotes the domain of f, i.e. the set of all elements in S that connect to an element in T.

Notice that by using $\mathscr{L}$, we assume a *well-definedness order* from left to right (e.g., for $P \wedge Q$) for formulae (this is similar to evaluating conditional statements, e.g. && or ||, in several programming languages).

A.2.4 Sequents

Event-B defines *proof obligations*, which must be proved to show that formal models fulfil their specified properties. Often these verification conditions are expressed in terms of *sequents*. A sequent $H \vdash G$ means that the *goal* G holds under the assumption of the set of *hypotheses* H. The obligations are discharged using certain inference rules, which we will not describe here. Instead, we will give informal justification of how proof obligations can be discharged. The purpose of presenting proof obligations within this appendix is to illustrate various conditions that need to be proved to maintain the consistency of the example.

A.3 Example. A Course Management System

The running example that we are going to use for illustrating Event-B is a course management system. We describe a requirements document of the system as follows. A club has some fixed *members*; amongst them are *instructors* and *participants*. Note that a member can be both an instructor and a participant.

ASM 1	Instructors are members of the club.

ASM 2	Participants are members of the club.

There are predefined *courses* that can be offered by a club. Each course is associated with exactly one fixed instructor.

ASM 3	There are predefined courses.

ASM 4	Each course is assigned to one fixed instructor.

A course is either *opened* or *closed* and is managed by the system.

REQ 5	A course is either *opened* or *closed*.

REQ 6	The system allows a closed course to be opened.

REQ 7	The system allows an opened course to be closed.

The number of opened courses is limited.

REQ 8	The number of opened courses cannot exceed a given limit.

Only when a course is opened, can participants *register* for the course. An important constraint for registration is that an instructor cannot attend his own courses.

REQ 9	Participants can only register for an opened course.

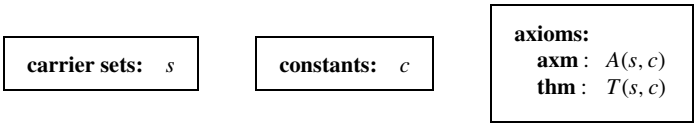
REQ 10	Instructors cannot attend their own courses.

In subsequent sections, we develop a formal model based on the above requirements document. In particular, we will refer to the above requirements in order to justify how they are formalised in the Event-B model.

A.4 Contexts

A context may contain *carrier sets*, *constants*, *axioms*, and *theorems*. Carrier sets are user-defined types. By convention, a carrier set s is *non-empty*, i.e., satisfying $s \neq \varnothing$, and *maximal*, i.e., satisfying $\forall x \cdot x \in s$. Constants c denote *static objects* within the development.[1] Axioms are *presumed properties* of carrier sets and constants. Theorems are *derived properties* of carrier sets and constants. Carrier sets and constants model the parameters of development. Moreover, axioms state parameter properties, assumed to hold for all their possible instances. A context C with carrier sets s, constants c, axioms $A(s, c)$, and theorems $T(s, c)$ can be presented as follows.

[1] When referring to carrier sets s and constants c, we usually allow for multiple carrier sets and constants, i.e., they may be "vectors".



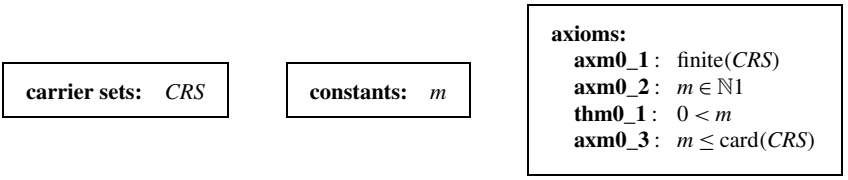
Note that we present axioms and theorems using different *labels*, i.e., **axm** and **thm**. Later on, we also use different labels for other modelling elements, such as invariants (**inv**), guards (**grd**) and actions (**act**).

Proof obligations are generated to ensure that the theorems are derivable from *previously* defined axioms. This is captured by the following proof obligation rule, called THM:

$$A(s, c) \vdash T(s, c). \qquad \text{(THM)}$$

A.4.1 Example. Context coursesCtx

In this initial model, we focus on the opening and closing of courses by the system. As a result, our initial context coursesCtx contains a carrier set *CRS* denoting the set of courses that can be offered by the club (ASM 3). Moreover, coursesCtx includes a constant m denoting the maximum number of courses that the club can have at the same time (with respect to requirement REQ 8). The context coursesCtx is as follows.



Note that we label the axioms and theorems with prefixes denoting the role of the modelling elements, i.e., **axm** and **thm**, with some numbers. For example, **axm0_1** denotes the first (i.e., 1) axiom for the initial model (i.e., 0). We apply this systematic labelling throughout our development.

The assumptions on *CRS* and m are captured by the axioms and theorems as follows. Axiom **axm0_1** states that *CRS* is finite. Axiom **axm0_2** states that m is a member of the set of natural numbers (i.e., m is a natural number). Finally, axiom **axm0_3** states that m cannot exceed the number of possible courses that can be offered by the club, represented as card(*CRS*), the cardinality of *CRS*. A derived property of m is presented as theorem **thm0_1**.

thm0_1/THM A proof obligation is generated for **thm0_1** as follows. Notice that **axm0_3** does not appear in the set of hypotheses for the obligation, since it is declared after **thm0_1**. By convention, each proof obligation is labelled according to the element involved and the name of the proof obligation rule. Here **thm0_1**/THM indicates that it is a THM proof obligation for **thm0_1**.

finite(*CRS*) $m \in \mathbb{N}1$ $\vdash$ $0 < m$	**thm0_1**/THM

The obligations can be trivially discharged since $\mathbb{N}1$ is the set of all positive natural numbers, i.e. $\{1, 2, \ldots\}$.

axm0_3/WD It is required to prove that **axm0_3** is well defined. The corresponding proof obligation is as follows.

finite(*CRS*) $m \in \mathbb{N}$ $0 \leq m$ $\vdash$ finite(*CRS*)	**thm0_1**/WD

Since the goal appears amongst the hypotheses, the proof obligation can be discharged trivially. Note that the order of appearance of the axioms is important. In particular, **axm0_1** needs to be declared before **axm0_3**.

A.5 Machines

Machines specify behavioural properties of Event-B models. In order to have access to information on context C, defined in Sect. A.4, machine M must connect with C. When machine M *sees* context C, it has access to C's carrier sets s and constants c, to refer to them when modelling, and C's axioms $A(s, c)$ and theorems $T(s, c)$, to use them as assumptions during proving. For clarity, in the following presentation we will not refer explicitly to the modelling elements of C. Note that in general a machine can see several contexts.

Machines M may contain *variables*, *invariants*, *theorems*, *events*, and a *variant*. Variables v define the *state* of a machine and are *constrained* by invariants $I(v)$. Theorems $R(v)$ are *additional properties* of v derivable from $I(v)$.

variables: v	**invariants:** **inv** : $I(v)$ **thm** : $R(v)$

A proof obligation (also called THM) is generated to prove that the theorem $R(v)$ is derivable from $I(v)$:

$$I(v) \vdash R(v). \qquad \text{(THM)}$$

Possible state changes are described by events (see Sect. A.5.1). The variant is used to prove convergence properties of events (see Sect. A.5.2).

A.5.1 Events

An event e can be represented by the term

$$\mathsf{e} \mathrel{\widehat{=}} \textbf{any } x \textbf{ where } G(x, v) \textbf{ then } Q(x, v) \textbf{ end}, \tag{A.1}$$

where x stands for the event *parameters*,[2] $G(x, v)$ is the *guard* (the conjunction of one or more predicates) and $Q(x, v)$ is the *action*. The guard states the necessary condition under which an event may occur. The event is said to be *enabled* in a state, if there exists some value for its parameter x that makes its guard $G(x, v)$ hold in this state. The action describes how the state variables evolve when the event occurs. We use the short form

$$\mathsf{e} \mathrel{\widehat{=}} \textbf{when } G(v) \textbf{ then } Q(v) \textbf{ end} \tag{A.2}$$

when the event does not have any parameters, and we write

$$\mathsf{e} \mathrel{\widehat{=}} \textbf{begin } Q(v) \textbf{ end} \tag{A.3}$$

when, in addition, the event guard always holds (i.e., equals $\top$). A dedicated event in the form of (A.3) is used for the *initialisation* event (usually represented by init). Note that events may be annotated with their convergence status, which we will look at in Sect. A.5.2.

The action of an event is composed of one or more *assignments* of the form

$$a := E(x, v) \tag{A.4}$$

or

$$a :\in E(x, v) \tag{A.5}$$

or

$$a :| P\big(x, v, a'\big), \tag{A.6}$$

where a is some of the variables contained in v, $E(x, v)$ is an expression, and $P(x, v, a')$ is a predicate. Note that the variables on the left-hand side of the assignments contained in an action must be disjoint. In (A.5), a must be a single variable, whereas it can be a vector of variables in (A.4) and (A.6). In particular, in (A.4), if a is a vector containing $n > 0$ variables, then E must also be a vector of expressions, one for each of the n variables. Assignments of the form (A.4) are *deterministic*, whereas assignments of the other two forms are *nondeterministic*. In (A.5), a is assigned an element of a set $E(x, v)$. (A.6) refers to P, which is a *before-after predicate* relating the values v (before the action) and a' (afterwards). (A.6) is also the

[2] When referring to variables v and parameters x, we usually allow for multiple variables and parameters, i.e., they may be "vectors".

most general form of assignment and nondeterministically selects an after-state a' satisfying P and assigns it to a. Note that the before-after predicates for the other two forms are as expected; namely, $a' = E(x, v)$ and $a' \in E(x, v)$, respectively.

All assignments of an action $Q(x, v)$ occur simultaneously, which is expressed by conjoining their before-after predicates. Hence each event corresponds to a before-after predicate $\boldsymbol{Q}(x, v, v')$ established by conjoining all before-after predicates associated with each assignment and $b = b'$, where b is unchanged variables. Note that the initialisation init therefore corresponds to an *after predicate* $\boldsymbol{K}(v')$, since there are no states before initialisation.

A.5.1.1 Proof Obligations

Below we describe some important proof obligation rules for Event-B machines, namely, invariant establishment and preservation, and action feasibility.

Invariant Establishment and Preservation An essential feature of an Event-B machine M is its invariant $I(v)$. It shows properties that hold in every reachable state of the machine. Obviously, this does not hold a priori for any machines and invariants, and therefore must be proved. A common technique for proving an invariant property is to prove it by induction: (1) to prove that the property is established by the initialisation init (*invariant establishment*), and (2) to prove that the property is maintained whenever variables change their values (*invariant preservation*).

Invariant establishment states that any possible state after initialisation given by the after predicate $\boldsymbol{K}(v')$ must satisfy the invariant I. The proof obligation rule is as follows:

$$\boldsymbol{K}(v') \vdash I(v'). \tag{INV}$$

Invariant preservation makes it necessary to prove that every event occurrence *reestablishes* the invariants I. More precisely, for every event e, assuming the invariants I and e's guard G, we must prove that the invariants still hold in any possible state after the event execution given by the before-after predicate $\boldsymbol{Q}(x, v, v')$. The proof obligation rule is as follows:

$$I(v), G(x, v), \boldsymbol{Q}(x, v, v') \vdash I(v'). \tag{INV}$$

Note that in practice, by the property of conjunctivity, we can prove the establishment and preservation of each invariant separately.

Feasibility *Feasibility* states that the action of an event is always feasible whenever the event is enabled. In other words, there are always possible after values for the variables satisfying the before-after predicate. In practice, we prove feasibility for individual assignment of the event action. For deterministic assignments, feasibility holds trivially. The feasibility proof obligation generated for a nondeterministic assignment of the form $a :| P(x, v, a')$ is as follows:

$$I(v), G(x, v) \vdash \exists a' \cdot P(x, v, a'). \tag{FIS}$$

A.5.2 Event Convergence

A set of events can be proved to collectively converge. In other words, these events cannot take control forever and hence one of the other events eventually occurs. We call these events *convergent*. To prove this, one specifies a variant V which maps a state v to a natural number. One then proves that each convergent event strictly decreases V. More precisely, let e be a convergent event where v is the state before executing e and v' is the state after. Then for each such e, v, and v', one proves that $V(v') < V(v)$, additionally assuming all invariants and the e guard. Since the variant maps a state to a natural number, V induces a well-founded ordering on system states given by the strictly less than order ($<$) of their images under V. The following proof obligation rules apply to every convergent event where VAR ensures the decrement of the variant and NAT ensures that the variant is a natural number when the event is enabled:

$$I(v), G(x, v), \boldsymbol{Q}\big(x, v, v'\big) \vdash V\big(v'\big) < V(v) \qquad \text{(VAR)}$$

$$I(v), G(x, v) \vdash V(v) \in \mathbb{N}. \qquad \text{(NAT)}$$

Note that in some cases the convergence of some events can be shown only in a later refinement, but not immediately. In this case, their convergence is *anticipated* and we must prove that $V(v') \leq V(v)$, that is, these anticipated events do not increase the variant. Anticipated events must obey NAT and the following proof obligation rule, also called VAR:

$$I(v), G(x, v), \boldsymbol{Q}\big(x, v, v'\big) \vdash V\big(v'\big) \leq V(v). \qquad \text{(VAR)}$$

As mentioned above, variant V is a natural number. Alternatively, V can be a *finite* set expression. In this case, for convergent events, one has to prove that it decreases the variant according to the strict subset inclusion $\subset$ ordering. For anticipated events, we ensure that these events do not increase the variant by proving that $V(v') \subseteq V(v)$. The proof obligation rule VAR is adapted accordingly.

For proving that the variant V is a finite set, the following proof obligation rule called FIN applies:

$$I(v) \vdash \text{finite}\big(V(v)\big). \qquad \text{(FIN)}$$

Note that FIN needs to be proved once, i.e., it does not depend on the set of convergent and anticipated events (cf. NAT).

The convergence attribute of an event is denoted by the keyword **status** with three possible values: *convergent*, *anticipated*, and *ordinary* (for events which are neither convergent nor anticipated). Events are *ordinary* by default.

A.5.3 Deadlock-Freeness

A machine M is said to be *deadlocked* in some state if all of its events are disabled in that state. *Deadlock-freeness* for M ensures that there are always some enabled

events during the execution of M. Assume that M contains a set of n events e_i ($i \in 1, \ldots, n$) of the following form:

$$\mathsf{e_i} \mathrel{\widehat{=}} \textbf{any } x_i \textbf{ where } G_i(x_i, v) \textbf{ then } Q_i(x_i, v) \textbf{ . end}$$

The proof obligation rule for deadlock-freeness[3] is as follows:

$$I(v) \vdash \big(\exists x_1 \cdot G_1(x_1, v)\big) \lor \cdots \lor \big(\exists x_n \cdot G_n(x_n, v)\big). \qquad \text{(DLF)}$$

A.5.4 Example. Machine m0

We develop machine m0 of the initial model, focusing on courses opening and closing. This machine sees context coursesCtx as developed in Sect. A.4.1, and as a result has access to the carrier set *CRS* and constant m. We model the set of opened courses by a variable, namely *crs* (REQ 5). Invariant **inv0_1** states that it is a subset of available courses *CRS*. A consequence of this invariant and of axiom **axm0_1** is that *crs* is finite, and this is stated in m0 as theorem **thm0_2**. Requirement REQ 8 is directly captured by invariant **inv0_2**: the number of opened courses, i.e., card(*crs*) is bounded above by m. Initially, all courses are closed; hence *crs* is set to the empty set ($\varnothing$).

variables: crs

invariants:
inv0_1 : $crs \subseteq CRS$
thm0_2 : finite(crs)
inv0_2 : $\text{card}(crs) \leq m$

init
begin
$crs := \varnothing$
end

We model the opening and closing of courses using two events OpenCourses and CloseCourses as follows (REQ 6 and REQ 7).

OpenCourses
status *ordinary*
when
grd0_1 : $\text{card}(crs) \neq m$
thm0_3 : $crs \neq CRS$
then
act0_1 : $crs :| crs \subset crs' \land \text{card}(crs') \leq m$
end

CloseCourses
status *anticipated*
any cs **where**
grd0_1 : $cs \subseteq crs$
grd0_2 : $cs \neq \varnothing$
then
act0_1 : $crs := crs \setminus cs$
end

We deliberately choose to model these events using different features of Event-B. In OpenCourses, we use a nondeterministic action to model the fact that some new courses are opened, i.e. $crs \subset crs'$, as long as the number of opened courses will not exceed its limit, i.e. $\text{card}(crs') \leq m$. The guard of the event states that the current number of opened courses has not yet reached the limit.

[3] Typically, this is encoded as a theorem in the machine after all invariants.

CloseCourses models the set of courses that are going to be closed using parameter *cs*. It is a non-empty set of currently opened courses which is captured by CloseCourses' guard. The action is modelled in a straightforward way by removing *cs* from set *crs*.

We set the convergence status for OpenCourses and CloseCourses to be *ordinary* and *anticipated*, respectively. We postpone the reasoning about the convergence of CloseCourses till later refinements. Our intention is to prove that there can only be a finite number of occurrences of CloseCourses between any two OpenCourses events.

A.5.4.1 Proof Obligations

We present some of the obligations to illustrate what needs to be proved for the consistency of m0. We applied the proof obligation rules as shown earlier in this section. Notice that we can take the axioms and theorems of the seen context coursesCtx as hypotheses in the proof obligations. For clarity, we show only the parts of the hypotheses that are relevant for discharging the proof obligations. Moreover, we also show the proof obligations in their simplified forms, e.g. when event assignments are deterministic.

thm0_2/THM This obligation corresponds to the rule THM, in order to ensure that **thm0_2** is derivable from previously declared invariants.

$\ldots$ $\text{finite}(CRS)$ $crs \subseteq CRS$ $\vdash$ $\text{finite}(crs)$	**thm0_2**/THM

The proof obligation holds trivially since *crs* is a subset of a finite set, i.e., *CRS*.

init/inv0_2/INV This obligation ensures that the initialisation init establishes invariant **inv0_2**.

$\ldots$ $0 \leq m$ $\vdash$ $\text{card}(\varnothing) \leq m$	init/**inv0_2**/INV

Since the cardinality of the empty set $\varnothing$ is 0, the proof obligation holds trivially.

OpenCourses/thm0_3/THM This obligation ensures that **thm0_3** is derivable from the invariants and the previously declared guards of OpenCourses.

$\ldots$ $m \leq \text{card}(CRS)$ $crs \in \mathbb{P}(CRS)$ $\text{card}(crs) \leq m$ $\text{card}(crs) \neq m$ $\vdash$ $crs \neq CRS$	OpenCourses/**thm0_3**/THM

Informally, we can derive from the hypotheses that $\text{card}(crs) < \text{card}(CRS)$; hence crs must be different from CRS.

OpenCourses/act0_1/FIS This obligation corresponds to rule FIS and ensures that the nondeterministic assignment of OpenCourses is feasible when the event is enabled.

$\ldots$ $crs \neq CRS$ $\text{card}(crs) \leq m$ $\text{card}(crs) \neq m$ $\vdash$ $\exists crs' \cdot crs \subset crs' \wedge \text{card}(crs') \leq m$	OpenCourses/**act0_1**/FIS

The reasoning about the proof obligation is as follows. Since crs is different from CRS, there exists an element c which is closed, i.e., not in crs. By adding c to the set of opened courses, we strictly increase the number of opened courses by 1. Moreover, the number of opened courses after executing the event is still within the limit since originally it is strictly below the limit.

CloseCourses/inv0_2/INV This obligation corresponds to rule INV and is simplified accordingly since the assignment is deterministic. The purpose of the obligation is to prove that CloseCourses maintains invariant **inv0_2**.

$\ldots$ $\text{card}(crs) \leq m$ $\vdash$ $\text{card}(crs \setminus cs) \leq m$	CloseCourses/**inv0_2**/INV

Since removing some courses cs from the set of opened courses crs can only reduce its number, the proof obligation can be trivially discharged.

DLF/THM The deadlock-freeness condition is encoded as theorem **DLF** of machine m0, which results in the following proof obligation.

$\ldots$ $0 < m$ $\vdash$ $(\text{card}(crs) \neq m) \ \vee \ (\exists cs \cdot cs \subseteq crs \wedge cs \neq \varnothing)$	**DLF**/THM

We reason as follows. If card(crs) $\neq m$, the goal trivially holds. Otherwise, if card(crs) $= m$, since $m \neq 0$, $crs \neq \varnothing$. As a result, we can prove that $\exists cs \cdot cs \subseteq crs \wedge cs \neq \varnothing$ by instantiating cs with crs.

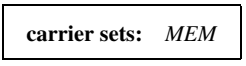
A.6 Context Extension

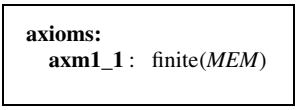
Context extension is a mechanism for introducing more static details into an Event-B development. A context can *extend* one or more contexts. When describing a context D as extending another context C, we call C and D the abstract and concrete context, respectively. By extending C, D "inherits" all the abstract elements of C, i.e., carrier sets, constants, axioms and theorems. This means that (1) a context extending D also implicitly extends C, and (2) a machine seeing D also implicitly sees C. As a result, proof obligation rule THM for D also has additional assumptions in the form of axioms and theorems from the abstract context C.

Subsequently, we present three new contexts that we use in the next refinement of our running example.

A.6.1 Context membersCtx

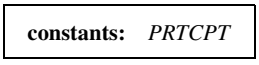
This is an initial context (i.e., it does not extend any other context) containing a carrier set *MEM*. *MEM* represents the set of club members, with an axiom stating that it is finite.

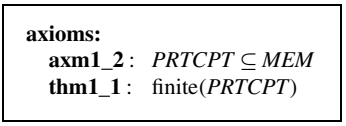




A.6.2 Context participantsCtx

This context extends the previously defined context membersCtx and is as follows.





Constant *PRTCPT* denotes the set of participants that must be members of the club as specified in ASM 2 (**axm1_2**). Theorem **thm1_1** states that there can be only a finite number of participants, which gives rise to the following trivial proof obligation.

finite(*MEM*) *PRTCPT* ⊆ *MEM* ⊢ finite(*PRTCPT*)	**thm1_1**/THM

An important point is that axiom **axm1_1** of the abstract context membersCtx appears as a hypothesis in the proof obligation.

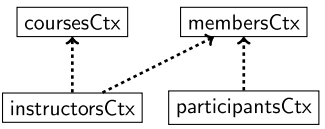
A.6.3 Context instructorsCtx

This context extends two contexts, coursesCtx and membersCtx, and introduces two constants, namely *INSTR* and *instrs*. *INSTR* models the set of instructors that are members of the club as specified by ASM 1 (**axm1_3**). Constant *instrs* models the relationship between courses and instructors and is constrained by **axm1_4**: it is a *total function* from *CRS* to *INSTR*, and hence directly formalises requirement ASM 4. Recall the definition of total function f from set S to set T: f is a relation from S to T where every element in S has exactly one mapping to some element in T.

constants: *INSTR*, *instrs*

axioms:	
axm1_3 :	$INSTR \subseteq MEM$
axm1_4 :	$instrs \in CRS \rightarrow INSTR$

The hierarchy of context extensions for our example is summarised in the following diagram.



A.7 Machine Refinement

Machine refinement is a mechanism for introducing details about the dynamic properties of a model [2]. For more details on the theory of refinement, we refer the reader to the Action System formalism [4], which has inspired the development of Event-B. We present here the proof obligations defined in Event-B, related to refinement. When speaking about machine N refining another machine M, we refer to M as the *abstract* machine and to N as the *concrete* machine.

Despite the fact that the formal definition of Event-B refinement does not distinguish between *superposition refinement* and *data refinement*, we illustrate them in separate sections to show different aspects of the two. In superposition refinement, the abstract variables of M are retained in the concrete machine N, with possibly some additional concrete variables. In data refinement, the abstract variables v are

replaced by concrete variables w and, subsequently, the connections between M and N are represented by the relationship between v and w. In fact, more often, Event-B refinement is a mixture of both superposition and data refinement: some of the abstract variables are retained, while others are replaced by new concrete variables.

A.7.1 Superposition Refinement

As mentioned earlier, in superposition refinement, variables v of the abstract machine M are kept in the refinement, i.e. as part of the state of N. Moreover, N can have some additional variables w. The concrete invariants $J(v, w)$ specify the relationship between the old and new variables. Each abstract event e is refined by a concrete event f (later on we will relax this one-to-one constraint). Assume that the abstract event e and the concrete event f are as follows:

$$\mathsf{e} \mathrel{\widehat{=}} \textbf{any } x \textbf{ where } G(x, v) \textbf{ then } Q(x, v) \textbf{ end}$$

$$\mathsf{f} \mathrel{\widehat{=}} \textbf{any } x \textbf{ where } H(x, v, w) \textbf{ then } R(x, v, w) \textbf{ end}$$

We assume now that e and f have the same parameters x. The more general case, where the parameters are different, is presented in Sect. A.7.2.

Somewhat simplifying, we say that f refines e if the guard of f is stronger than that of e (*guard strengthening*), concrete invariants J are maintained by f, and abstract action Q simulates the concrete action R (*simulation*). These conditions are stated as the following proof obligation rules:

$$I(v), J(v, w), H(x, v, w) \vdash G(x, v) \qquad \text{(GRD)}$$

$$I(v), J(v, w), H(x, v, w), \boldsymbol{R}\big(x, v, w, v', w'\big) \vdash \boldsymbol{Q}\big(x, v, v'\big) \qquad \text{(SIM)}$$

$$I(v), J(v, w), H(x, v, w), \boldsymbol{R}\big(x, v, w, v', w'\big) \vdash J\big(v', w'\big) \qquad \text{(INV)}$$

In particular, if the guard and action of an abstract event are retained in the concrete event, the proof obligations GRD and SIM are trivial; hence we only need to consider INV for proving that the gluing invariants are reestablished.

Proof obligations are generated to ensure that each assignment of concrete event f is feasible. In the case where the action of the abstract event is retained in f, we only need to prove the feasibility of any additional assignment.

In the course of refinement, *new events* are often introduced into a model. New events must be shown to refine the implicit abstract event SKIP, which does nothing, i.e., does not modify abstract variables v.

A.7.1.1 Machine m1

Machine m1 sees contexts instructorsCtx and participantsCtx. As a result, it implicitly sees coursesCtx and membersCtx. Variable *crs* is retained in this refinement.

An additional variable *prtcpts* representing information about course participants is introduced. Invariant **inv1_1** models *prtcpts* as a relation between the set of opened courses *crs* and the set of participants *PRTCPT*. Requirement REQ 10 is directly modelled by invariant **inv1_2**: for every opened course c, the instructor of this course, i.e., $instrs(c)$, is not amongst its participants, represented by $prtcpts[\{c\}]$.

variables: $crs, prtcpts$

invariants:
inv1_1 : $prtcpts \in crs \leftrightarrow PRTCPT$
inv1_2 : $\forall c \cdot c \in crs \Rightarrow instrs(c) \notin prtcpts[\{c\}]$

init
begin
$\ldots$
$prtcpts := \varnothing$
end

Initially, there are no opened courses; hence *prtcpts* is assigned to be $\varnothing$.

The original abstract event OpenCourses stays unchanged in this refinement, while an additional assignment is added to CloseCourses to update *prtcpts* by removing the information about the set of closing courses *cs* from it.

CloseCourses
status *anticipated*
any *cs* **where**
$\ldots$
then
$\ldots$
act1_2 : $prtcpts := cs \lhd prtcpts$
end

A new event is added, namely Register, to model the registration of a participant p for an opened course c. The guard of the event ensures that p is not the instructor of the course (**grd1_3**) and is not yet registered for the course (**grd1_4**). The action of the event updates *prtcpts* accordingly by adding the mapping $c \mapsto p$ to it.

Register
status *convergent*
any p, c **where**
grd1_1 : $p \in PRTCPT$
grd1_2 : $c \in crs$
grd1_3 : $p \neq instrs(c)$
grd1_4 : $c \mapsto p \notin prtcpts$
then
act1_1 : $prtcpts := prtcpts \cup \{c \mapsto p\}$
end

We attempt to prove that Register is convergent and CloseCourses is anticipated using the following variant.

variant: $(crs \times PRTCPT) \setminus prtcpts$

The variant is a set of mappings; each links an opened course to a participant who has *not* registered for the respective course.

We present some of the important proof obligations for m1. Proof obligations GRD and SIM are trivial for events OpenCourses and CloseCourses. Consequently, we only need to consider INV for these old events.

CloseCourses/inv1_2/INV This obligation is to ensure that **inv1_2** is maintained by CloseCourses. The obligation is trivial, in particular, because, given that $c \notin cs$, $(cs \lhd prtcpts)[\{c\}]$ is the same as $prtcpts[\{c\}]$.

$\ldots$ $\forall c \cdot c \in crs \Rightarrow instrs(c) \notin prtcpts[\{c\}]$ $\vdash$ $\forall c \cdot c \in crs \setminus cs \Rightarrow instrs(c) \notin (cs \lhd prtcpts)[\{c\}]$	CloseCourses/**inv1_2**/INV

Register/inv1_1/INV This obligation is to guarantee that **inv1_1** is maintained by the new event Register.

$\ldots$ $prtcpts \in crs \leftrightarrow PRTCPT$ $p \in PRTCPT$ $c \in crs$ $\vdash$ $prtcpts \cup \{c \mapsto p\} \in crs \leftrightarrow PRTCPT$	Register/**inv1_1**/INV

FIN This obligation is to ensure that the declared variant used for proving convergence of events is finite (FIN). This is trivial, since the set of opened courses *crs* and the set of participants *PRTCPT* are both finite.

$\ldots$ finite(crs) finite($PRTCPT$) $\vdash$ finite($(crs \times PRTCPT) \setminus prtcpts$)	FIN

CloseCourses/VAR This proof obligation corresponds to rule VAR, ensuring that anticipated event CloseCourses does not increase the variant.

$\ldots$ $\vdash$ $((crs \setminus cs) \times PRTCPT) \setminus (cs \lhd prtcpts)$ $\subseteq$ $(crs \times PRTCPT) \setminus prtcpts)$	CloseCourses/VAR

Register/VAR This proof obligation corresponds to rule VAR, ensuring that the convergent event Register decreases the variant. This is trivial since a new mapping $c \mapsto p$ is added to *prtcpts*, effectively increasing it, and hence strictly decreasing the variant.

$\ldots$ $c \mapsto p \notin prtcpts$ $\vdash$ $(crs \times PRTCPT) \setminus (prtcpts \cup \{c \mapsto p\}))$ $\subset$ $(crs \times PRTCPT) \setminus prtcpts)$	Register/VAR

A.7.2 Data Refinement

In data refinement, abstract variables v are removed and replaced by concrete variables w. The states of abstract machine M are related to the states of concrete machine N by *gluing invariants* $J(v, w)$. In Event-B, the gluing invariants J are declared as invariants of N and also contain the *local* concrete invariants, i.e., those constraining only concrete variables w.

Again, we assume a one-to-one correspondence between an abstract event e and a concrete event f. Let e and f be as follows:[4]

$$\mathsf{e} \mathrel{\widehat{=}} \textbf{any } x \textbf{ where } G(x, v) \textbf{ then } Q(x, v) \textbf{ end}$$

$$\mathsf{f} \mathrel{\widehat{=}} \textbf{any } y \textbf{ where } H(y, w) \textbf{ then } R(y, w) \textbf{ end}$$

As with superposition refinement, we can say that f refines e if the guard of f is stronger than the guard of e (*guard strengthening*), and the gluing invariants $J(v, w)$ establish a simulation of f by e (*simulation*). This condition is captured by the following proof obligation rule:

$$\begin{array}{l} I(v) \\ J(v, w) \\ H(y, w) \\ \boldsymbol{R}(y, w, w') \\ \vdash \\ \exists x, v' \cdot G(x, v) \ \wedge \ \boldsymbol{Q}(x, v, v') \ \wedge \ J(v', w') \end{array} \tag{A.7}$$

In order to simplify and split the above proof obligation, Event-B introduces the notion of "witnesses" for the abstract parameters x and the after value of the abstract variables v'. Witnesses are predicates of the form $W_1(x, y, v, w, w')$ (for x)

[4] Concrete events may be annotated with abstract events name and witnesses, which we will show later.

and $W_2(v', y, v, w, w')$ (for v'), which are required to be *feasible*. The corresponding proof obligations are as follows:

$$I(v), J(v, w), H(y, w), \boldsymbol{R}(y, w, w') \vdash \exists x \cdot W_1(x, y, v, w, w') \quad \text{(WFIS)}$$

$$I(v), J(v, w), H(y, w), \boldsymbol{R}(y, w, w') \vdash \exists v' \cdot W_2(v', y, v, w, w') \quad \text{(WFIS)}$$

Typically, witnesses are declared deterministically, i.e. in the form $x = \cdots$ or $v' = \cdots$. In these cases, witnesses are trivially feasible; hence the corresponding proof obligations are omitted.

Given the witnesses, the refinement proof obligation (A.7) is replaced by three different proof obligations as follows:

$$I(v), J(v, w), H(y, w), W_1(x, y, v, w, w') \vdash G(x, v) \quad \text{(GRD)}$$

$$I(v), J(v, w), H(y, w), \boldsymbol{R}(y, w, w'), W_1(x, y, v, w, w'), W_2(v', y, v, w, w') \vdash \boldsymbol{Q}(x, v, v') \quad \text{(SIM)}$$

$$I(v), J(v, w), H(y, w), \boldsymbol{R}(y, w, w'), W_1(x, y, v, w, w'), W_2(v', y, v, w, w') \vdash J(v', w') \quad \text{(INV)}$$

The concrete event f can be denoted by the abstract event e (using keyword **refines**) and the witnesses (using keyword **with**) as follows:

f
  refines e
  any y **where**
    $H(y, w)$
  with
    x : $W_1(x, y, v, w, w')$
    v' : $W_2(v', y, v, w, w')$
  then
    $R(y, w)$
  end

The action of the concrete event is required to be feasible. The corresponding proof obligation FIS is similar to the one presented for the abstract machine, with the exception that both abstract and gluing invariants can be assumed.

For newly introduced events, as with superposition refinement, they must be proved to refine the implicit abstract event SKIP, which is unguarded and does nothing, i.e. does not modify abstract variables v. In this case, GRD is trivial, since the abstract guard is $\top$. For SIM and INV, we omit references to W_1 (since there are no parameters for the abstract SKIP event). Moreover, the witness W_2 for v' is trivial: $v' = v$.

As mentioned earlier, in general, Event-B refinement is a mixture of both superposition and data refinement. Often, some (not all) abstract variables are retained

in the refinement, while the other abstract variables are replaced by new concrete variables. Similarly, some abstract parameters can be present in the concrete event, where other parameters are replaced by some new concrete ones. In general, we only need to give witnesses to disappearing variables and parameters.

The one-to-one correspondence between the abstract and concrete events can be relaxed. When an abstract event e is refined by more than one concrete event f, we say that the abstract event e is *split* and prove that each concrete f is a valid refinement of the abstract event. Conversely, several abstract events e can be refined by one concrete f. We say that these abstract events are *merged* together. A requirement for merging events is that the abstract events must have identical actions. When merging events, we need to prove that the guard of the concrete event is stronger than the disjunction of the guards of the abstract events.

The concrete machine N can be proved to be *relatively deadlock-free* with respect to the abstract machine M. It means that if M can continue in some state, so can N. Assume that M contains a set of n events e_i $(i \in 1, \ldots, n)$ of the following form:

$$\mathsf{e_i} \mathrel{\widehat{=}} \textbf{any } x_i \textbf{ where } G_i(x_i, v) \textbf{ then } Q_i(x_i, v) \textbf{ end}$$

Assume that N contains a set of m events f_j $(j \in 1, \ldots, m)$ of the following form:

$$\mathsf{f_j} \mathrel{\widehat{=}} \textbf{any } y_i \textbf{ where } H_i(y_i, w) \textbf{ then } R_i(y_i, w) \textbf{ end}$$

The proof obligation rule for relative deadlock-freeness[5] is as follows:

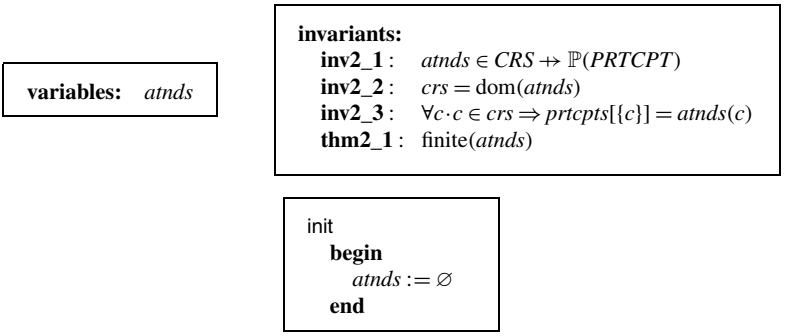
$$\begin{array}{l} I(v) \\ J(v, w) \\ \big(\exists x_1 \cdot G_1(x_1, v)\big) \lor \cdots \lor \big(\exists x_n \cdot G_n(x_n, v)\big) \\ \vdash \\ \big(\exists y_1 \cdot H_1(y_1, w)\big) \lor \cdots \lor \big(\exists y_m \cdot H_m(y_m, w)\big) \end{array} \qquad \text{(REL\_DLF)}$$

A.7.2.1 Machine m2

We perform a data refinement by replacing abstract variables *crs* and *prtcpts* by a new concrete variable *atnds*. This machine does not explicitly model any requirements from Sect. A.3: it implicitly inherits requirements from previous abstract machines. As stated in invariant **inv2_1**, *atnds* is a *partial function* from *CRS* to some set of participants (i.e., member of $\mathbb{P}(PRTCPT)$). Invariants **inv2_2** and **inv2_3** act as gluing invariants, linking abstract variables *crs* and *prtcpts* with concrete variable *atnds*. Invariant **inv2_2** specifies that *crs* is the domain *atnds*. Invariant **inv2_3** states that for every opened course c, the set of partic-

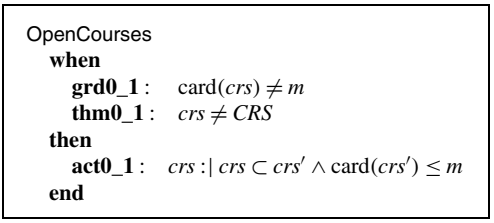
[5] Typically, this is encoded as a theorem in N after declaration of all invariants.

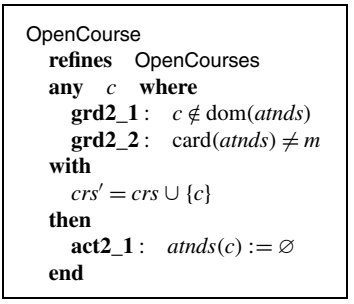
ipants attending that course represented abstractly as $prtcpts[\{c\}]$ is the same as $atnds(c)$.



We illustrate our data refinement by the following example. Assume that the available courses *CRS* are $\{c_1, c_2, c_3\}$, with c_1 and c_2 being opened, i.e., $crs = \{c_1, c_2\}$. Assume that c_1 has no participants, and p_1 and p_2 are attending c_2. Abstract variable *prtcpts* hence contains two mappings as follows: $\{c_2 \mapsto p_1, c_2 \mapsto p_2\}$. The same information can be represented by the concrete variable *atnds* as follows: $\{c_1 \mapsto \varnothing, c_2 \mapsto \{p_1, p_2\}\}$.

We refine the events using data refinement as follows. Event OpenCourses is refined by OpenCourse, where one course (instead of possibly several courses) is opened at a time. The course that is opening is represented by the concrete parameter c.





The concrete guards ensure that c is a closed course and the number of opened courses (card(*atnds*)) has not reached the limit m. The action of OpenCourse sets the initial participants for the newly opened course c to be the empty set. In order to prove the refinement relationship between OpenCourse and OpenCourses, we need to give the witness for the after value of the disappearing variable crs'. In this case, it is specified as $crs' = crs \cup \{c\}$, by adding the newly opened course c to the original set of opened courses *crs*.

Abstract event CloseCourses is refined by concrete event CloseCourse, where one course c (instead of possibly several courses *cs*) is closed at a time. The guard and action of concrete event CloseCourse are as expected.

CloseCourses
status *anticipated*
any cs **where**
grd0_1 : $cs \subseteq crs$
grd0_2 : $cs \neq \varnothing$
then
act0_1 : $crs := crs \setminus cs$
act2 : $prtcpts := cs \lhd prtcpts$
end

CloseCourse
refines CloseCourses
status *convergent*
any c **where**
grd2_1 : $c \in \mathrm{dom}(atnds)$
with
$cs = \{c\}$
then
act2_1 : $atnds := \{c\} \lhd atnds$
end

We need to give the witness for the disappearing abstract parameter cs. It is specified straightforwardly as $cs = \{c\}$. Notice also that we change the convergence status of CloseCourse from *anticipated* to *convergent*. We use the following variant to prove that CloseCourse is convergent.

variant: $\mathrm{card}(atnds)$

The variant represents the number of mappings in *atnds*, and since it is a partial function, it is also the same as the number of elements in its domain, i.e. $\mathrm{card}(atnds) = \mathrm{card}(\mathrm{dom}(atnds))$. As a result, the variant represents the number of opened courses.

Event Register is refined as follows:[6] references to *crs* and *prtcpts* in guard and action are replaced by references to *atnds*.

(abs_)Register
any p, c **where**
grd1_1 : $p \in PRTCPT$
grd1_2 : $c \in crs$
grd1_3 : $p \neq instrs(c)$
grd1_4 : $c \mapsto p \notin prtcpts$
then
act1_1 : $prtcpts := prtcpts \cup \{c \mapsto p\}$
end

(cnc_)Register
refines Register
any p, c **where**
grd2_1 : $p \in PRTCPT$
grd2_2 : $c \in \mathrm{dom}(attendees)$
grd2_3 : $p \neq instrs(c)$
grd2_4 : $p \notin atnds(c)$
then
act2_1 : $atnds(c) := atnds(c) \cup \{p\}$
end

We now show some proof obligations for proving the refinement of m1 by m2.

OpenCourse/act0_1/SIM This proof obligation corresponds to rule SIM, ensuring that the action **act0_1** of abstract event OpenCourses can simulate the action of concrete event OpenCourse. Notice the use of the witness for crs' as a hypothesis in the obligation.

[6] We use prefixes (abs_) and (cnc_) to denote the abstract and concrete versions of the event, respectively.

$\ldots$ $atnds \in CRS \pfun \mathbb{P}(PRTCPT)$ $crs = \text{dom}(attendees)$ $c \notin \text{dom}(attendees)$ $\text{card}(attendees) \neq m$ $crs' = crs \cup \{c\}$ $\vdash$ $crs \subset crs' \land \text{card}(crs') \leq m$	OpenCourse/**act0_1**/SIM

CloseCourse/grd0_1/GRD This proof obligation corresponds to rule GRD, ensuring that the guard of concrete event CloseCourse is stronger than the abstract guard **grd0_1** of abstract event CloseCourses. Note the use of the witness for *cs* as a hypothesis in the obligation.

$\ldots$ $crs = \text{dom}(atnds)$ $c \in \text{dom}(atnds)$ $cs = \{c\}$ $\vdash$ $cs \subseteq crs$	CloseCourse/**grd0_1**/GRD

CloseCourse/NAT This proof obligation corresponds to rule NAT on page 220; it ensures that the variant is a natural number when CloseCourse is enabled.

$\ldots$ $\text{finite}(atnds)$ $\vdash$ $\text{card}(atnds) \in \mathbb{N}$	CloseCourse/NAT

CloseCourse/VAR This proof obligation corresponds to rule VAR on page 220; it ensures that the variant is strictly decreased by CloseCourse. The obligation is trivial since the variant represents the number of opened courses and CloseCourse closes one of them.

$\ldots$ $atnds \in CRS \pfun PRTCPT$ $c \in \text{dom}(atnds)$ $\vdash$ $\text{card}(\{c\} \domsub atnds) < \text{card}(atnds)$	CloseCourse/VAR

A.8 Summary of the Development

The summary of the hierarchy of the development is illustrated in Fig. A.1.

Table A.3 summarises how assumptions and requirements are taken into account in our formal development. Note that the last refinement, m2, does not explicitly

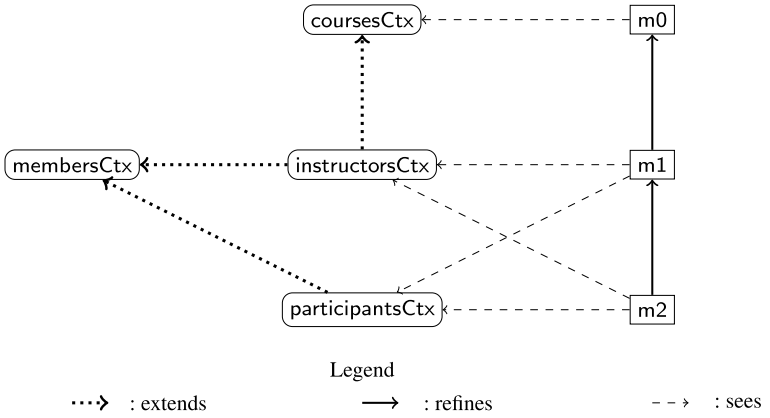


Fig. A.1 Development hierarchy

Table A.3 Requirements Tracing

Requirement	Models	Requirement	Models
ASM 1	instructorsCtx	REQ 6	m0
ASM 2	participantsCtx	REQ 7	m0
ASM 3	coursesCtx	REQ 8	m0
ASM 4	instructorsCtx	REQ 9	m1
REQ 5	m0	REQ 10	m1

take into account any requirements. Indeed, the requirements are implicitly *inherited* from the abstract machines through the refinement relationship.

The development is formalised and proved using the supporting Rodin platform[7] for Event-B [3] and is available online.[8] The summary of the proof statistics for the

Table A.4 Proof statistics

Constructs	Proof obligations	Automatic (%)	Manual (%)
coursesCtx	2	2 (100 %)	0 (0 %)
membersCtx	0	0 (N/A)	0 (N/A)
instructorsCtx	0	0 (N/A)	0 (N/A)
participantsCtx	1	1 (100 %)	0 (0 %)
m0	11	8 (73 %)	3 (27 %)
m1	14	13 (93 %)	1 (7 %)
m2	29	26 (90 %)	3 (10 %)
Total	57	50 (88 %)	7 (12 %)

[7] At the time of writing, we use Rodin version 2.4.0.

[8] http://deploy-eprints.ecs.soton.ac.uk/371/.

development is shown in Table A.4. Around 50 % of the proof obligations appear in m2, where we perform data refinement. Typically, data refinement involves radical changes to developments, since when replacing abstract variables with concrete variables, it is also necessary to adapt the events accordingly.

References

1. Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)
2. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
3. Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: An open toolset for modelling and reasoning in Event-B. Int. J. Softw. Tools Technol. Transf. 12(6), 447–466 (2010)
4. Back, R.-J.: Refinement calculus II: Parallel and reactive programs. In: de Bakker, J.W., de Roever, W.P., Rozenberg, G. (eds.) Stepwise Refinement of Distributed Systems, Mook, The Netherlands, May 1989. Lecture Notes in Computer Science, vol. 430, pp. 67–93. Springer, Berlin (1990)
5. Chandy, K., Misra, J.: Parallel Program Design: A Foundation. Addison-Wesley, Reading (1989)
6. Lamport, L.: The temporal logic of actions. Trans. Program. Lang. Syst. 6(3), 872–923 (1994)
7. Mehta, F.: Proofs for the working engineer. PhD thesis, ETH Zurich (2008)
8. Schmalz, M.: The logic of Event-B. Technical Report 698, Institute of Information Security, ETH Zurich (October 2010). http://www.inf.ethz.ch/research/disstechreps/techreports/show?serial=698

Appendix B
Evidence-Based Assistance for the Adoption of Formal Methods in Industry

Jean-Christophe Deprez, Christophe Ponsard, and Renaud De Landtsheer

Abstract Managing change in industrial development processes is often a challenge. It involves many levels of the organisation: top management, who need to understand the benefits and impacts of change, project managers, who need to understand new processes and adapt their planning and monitoring practices, and engineers, who need to be trained in the use of new methods and tools. This is especially true for the introduction of Formal Methods (FM), which may significantly impact software development lifecycle processes. Industries that seek to adopt formal methods often need assistance in understanding their benefits, what formal methods can be used at what phase of the development lifecycle and what strategy should be used for a successful deployment of formal methods. During the DEPLOY project, an entire segment of work was dedicated to collecting important material to use as evidence, to provide answers to recurring concerns of companies wishing to investigate the use of formal methods. This appendix presents the overall approach used to collect and structure DEPLOY material as an evidence repository, notably, using success stories and Frequently Asked Questions (FAQs). These are fully published on www.fm4industry.org. A sample of success stories and FAQs answers are given, along with explanations on how companies in typical industry situations can navigate the evidence repository to find information relevant to their contexts. Finally, we also discuss how an industry engaged in an adoption process can set up its own internal evidence based on its specific context and how it can share its experience without compromising potentially confidential information.

J.-C. Deprez (✉) · C. Ponsard · R. De Landtsheer
CETIC, Rue des Frères Wright 29/3, Charleroi, Belgium
e-mail: jcd@cetic.be

C. Ponsard
e-mail: cp@cetic.be

R. De Landtsheer
e-mail: rdl@cetic.be

A. Romanovsky, M. Thomas (eds.), *Industrial Deployment of System Engineering Methods*, DOI 10.1007/978-3-642-33170-1,

B.1 Introduction

Industry projects heavily rely on quantitative data when assessing project progress or product quality. However, in the early stage of transferring research results to industry, qualitative information can add very important value. A method based on measurement was proposed during the first year of the DEPLOY project, but it was rejected by Industry partners. Their main reasons for refusing an assessment based on measurements were confidentiality and a belief that numbers would not provide the right type of information to present a convincing argument for promoting formal methods within their organisation. In particular, DEPLOY industry partners recognised that quantitative methods are more appropriate for streamlined development projects, where a development approach is mastered by all project participants. On the other hand, in research transfer, where industry participants must understand new concepts, qualitative information is much more appropriate and convincing. DEPLOY industry partners also stressed the need to present evidence material in a form convenient for industry, such as well-structured Frequently Asked Questions (FAQs).

General surveys, such as that reported in [10] and a later extended one included in this book, confirm the need to provide evidence to support the adoption process by companies considering the use of Formal Methods (FM). **The goal of this appendix is to describe an approach to presenting and exploiting information related to FM adoption by industry**. The starting point was the concept of evidence. However, this needed to be further analysed in order to define how evidence could best provide information about formal methods to industry and in particular, about formal methods explored during the DEPLOY project. While one of the purposes for collecting evidence was to help industry partners of the DEPLOY project in their adoption process, the main target audience is organisations not involved in DEPLOY.

The sections of this appendix are organised as follows. First, a discussion elaborates on what constitutes evidence of formal method use in industry. From this, a general method for collecting and presenting evidence material is proposed. Second, different approaches are presented to searching for evidence material useful in one's context; in particular, different scenarios covering various industry sectors and different staff roles are presented. Then, some concrete exploitation scenarios of the material available online are highlighted. Finally, methods and tools for evidence collection and presentation within an organisation are proposed, including ways to be used by the organisation to share its experience with others.

B.2 A Method for Collecting and Presenting Evidence

Success stories have been shown to be an efficient way of collecting evidence of successful industry adoption. They often take the form of white papers. In some cases, they are published in industry tracks of conferences. Such publications rarely

Table B.1 A specific "success story" template

Name	Short meaningful identification to use as the title of the story.
Keywords	List of keywords helping in the classification of the success story.
Short description	Description (in a few lines) of what the story is about and what claims are supported.
Source	Organisation that contributed to the work of this story and a contextual description of the company's structure and research and innovation process.
Benefits	Positive impacts that can be inferred from the success story. (Note: Themes/questions from the FAQs highlight the important points on which impact are sought; hence one should review them when listing the benefits highlighted in the success story.)
Limitations	Limitations identified (not definitive if the story is ongoing): these are potential show stoppers and barriers to overcome.
Elaboration	Complete description of the work done to document the success story.
Consolidation	Additional arguments reported in the scientific literature corroborating the success story, including citations of scientific publications presenting these arguments.
Further work	Specific work still to be done by DEPLOY partners to make the story a success.
References	List of publications or reports related to the success story.
Related FAQs	List of questions in the FAQs to which this success story provided material.

allow sufficient space. Consequently, success stories often lack details of the very specific contexts in which they took place; for example, little information is usually provided about the overall innovation cycle of the enterprise involved in the success story, or on how researchers and production engineers collaborated during the transfer project.

In conclusion, traditional **success stories** are an interesting base for building an evidence repository of industry adoption, but they must be augmented to provide readers with additional contextual information as well as links to related efforts. Consequently, a specific "success story" template was designed to enhance and systematise the presentation of success stories emerging from the DEPLOY project. This template is presented in Table B.1.

The template above contains information on the context. In particular, the Source section suggests that an organisation should describe its structure and its research and innovation process. There can be large amounts of this context information. Hence it is recommended that the Source field should only provide the organisation's name and a link to another document that describes the full company context. This approach also has the added benefit that an enterprise with several success stories may often be linked to the same context document.

Success stories provide evidence of formal method transfer and usage in industry from a very specific viewpoint. By their very specific nature, success stories

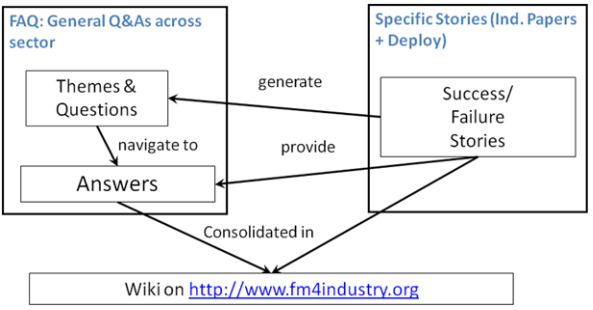


Fig. B.1 Organisation of the Evidence repository

do not, however, address high-level cross-concerns of various industry sectors or, at least, they do not present information in such a way that cross concerns can be easily deduced. This means that success stories should be complemented by another technique for presenting topics of general concern to many readers from various industries. During the DEPLOY project, researchers and industry partners proposed structuring this cross-cutting information in the **Frequently Asked Questions (FAQs) format**. Industry partners are accustomed to this type of format, where generic questions are answered in a fairly limited space. During DEPLOY, the average length targeted for answers was about one page as illustrated in the next section. The resulting structure of our evidence repository is shown in Fig. B.1.

At this point, it is necessary to identify a fairly exhaustive list of common concerns from different industry sectors. For this purpose, many documents provided by the DEPLOY industry partners as well as by the academic researchers and the tool providers were analysed. The high-level topics identified are:

- The impact on an organisation with regards to training scope and resourcing
- The impact on the software/system development process
  - The impact on the quality of product (and its work product) developed using formal methods,
  - The capability to exploit formal models at various stages of the development process,
  - The capability to perform reuse across development projects when formal methods are used, including reuse of formal and proven artefacts,
  - The capability to phase the learning of a formal method in an organisation and eventually limit the number of those who are expected to understand and become an expert in a formal method,
  - The capability to phase the migration to the formal method use incrementally (given the existence of products initially developed without using formal methods)
- Known strengths and weaknesses of tools associated with a formal method as well as the quality of support by tool providers
- External factors advocating take-up (from competition, standards bodies, laws) of formal methods
- General concerns related to formal methods in industry

These various themes validated by the DEPLOY industry and academic partners provide the initial categories for which the FAQs needs to be formulated.

Pieces of evidence are labelled **Industry Roles** to specifically target the relevant types of audience and increase the impact of answers. The organisations involved in DEPLOY were quite different, ranging from the technological SME with highly polyvalent roles to large companies with a specific R&D unit and very well-defined roles. Based on a careful study of the innovation cycle of the involved companies and on our experience in technology transfer, it was, however, possible to identify a set of key roles common to the DEPLOY partners, namely, high-level managers, project managers, engineers and technical analysts, and quality assurance staff. For a transfer project to be successful, many of these people must be involved in and supportive of the proposed transfer of research results. Thus, to obtain clear and concise answers, each FAQ should always explicitly target a single role. This does not necessarily mean that other roles will not be interested in the question, but rather that the named role is the primary target; hence the answer is provided to match this primary role. For instance, the wording of an answer targeted at engineers is different from that targeted at high-level management. Thus, if a high-level manager is interested in reading a FAQ answer primarily targeting engineers, he will understand that the answer is presented from the engineer's viewpoint. Making the targeted audience explicit by stating its roles definitely helps the reader.

In the context of transferring formal method engineering techniques to industry, the following roles were identified:

- High-Level Managers, who make strategic decisions and consider their financial impact
- Project and QA Managers, who supervise active users of FM (in production or R&D), plan projects and perform safety analysis and more traditional QA activities
- Engineers and Analysts, who actively use FM
- QA Practitioners, who must understand documents involving FM notation but do not need to develop the capabilities to produce them.

The FAQs approach suggests identifying questions of interest to each role for each theme. The important subtlety is that a question in the FAQs must be:

- Generic enough to interest enough readers and not be the sole concern of a single sector or, even worse, a single company.
- Specific enough so it is fairly easy to understand what results from an industry pilot should be proposed to what questions from the FAQs.

At the presentation level, the most adequate and commonly used medium is an Internet wiki, which gives wide access to the material. Wiki technology enables feedback from the people accessing the material, from comments to more active contributions, with a degree of control or moderation that can be adjusted. There are many Open Source wiki implementations, and we finally opted for Mediawiki [8], which is made very powerful by numerous extension plug-ins. Figure B.2 shows the homepage, accessible at http://www.fm4industry.org.

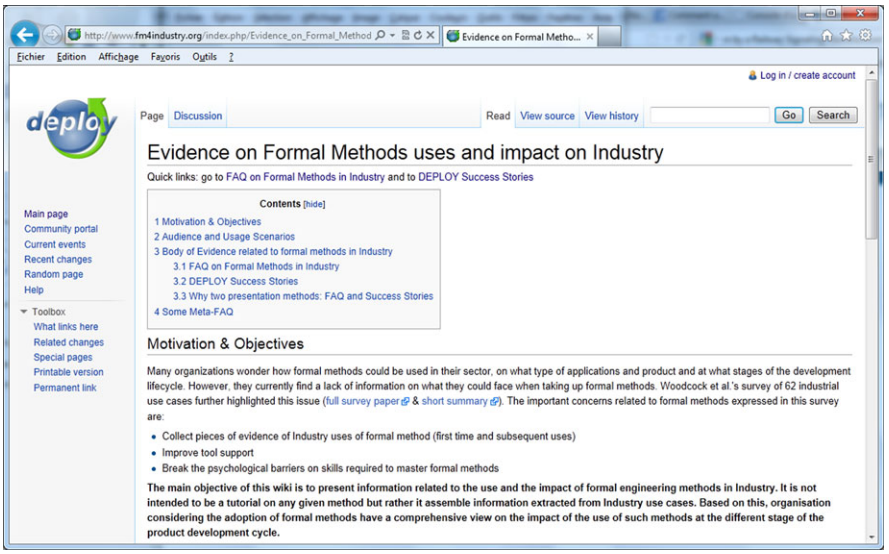


Fig. B.2 Organisation of the Evidence repository

B.3 Selected Highlights of FAQs and Success Stories

In this section we present selected FAQs and success stories to give some concrete and well-selected examples that will help the reader with the industrial background, interested in FM, to figure out what kind of information s/he can find and how it can help him/her. Those examples will serve to illustrate exploitation and production scenarios presented in the next sections, so the appendix is self-contained.

B.3.1 FAQ I: What Is the Position of Standards Regarding Formal Methods in My Industry Segment?

Theme: External Factor Pushing for Formal Method Adoption (ExFac)
Role: High-level Manager

B.3.1.1 Overview of Reference Standards for Safety-Critical Systems

The landscape of standards includes both generic and domain-specific standards. Figure B.3 shows a sectorial classification. A number of sectors have derived their standards from the generic IEC61508 standard while others have their own stand-alone standard.

- IEC 61511:2003 addresses industrial processes
- IEC 61513:2001 addresses the nuclear industry

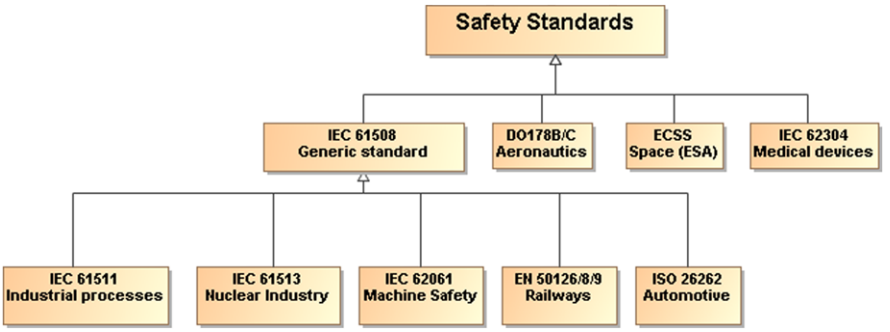


Fig. B.3 Sectorial classification of safety standards

- CE 62061:2005 addresses machine safety
- CENELEC/EN 50126/50128/50129 targets the railway sector
- ISO 26262:2011 addresses the automotive sector
- IEC 62304:2006 addresses the medical sector
- DO-178 addresses the aeronautic sector
- ECSS addresses the space sector (ESA)

In the aeronautic and space sectors, there are standards not directly linked to IEC61508.

- DO-178 (1992) is for the aeronautic sector
- ECSS is for the space sector

Figure B.3 shows a wide variety of standards. In addition to sector-specific constraints, there are also a number of other reasons for such a variety of standards, including historical (uncoordinated work, national level) and market reasons (market protection) [3]. Moreover, standards evolve, hopefully towards better integration, so they are revised (the publication year is often appended to their numerical identifier, as in EN 61508:2002) or made more specialised (as denoted by "B" in DO-178B).

Beyond this variety, standards exhibit common aspects:

- **definition of safety assurance levels:** all standards are based on a risk-oriented approach to their safety functions, classifying them into a number of dependability classes, typically by specifying PFD (Probability of Failure on Demand) and RRF (Risk Reduction Factor) figures. This classification varies across standards; for example, in IEC 61508, the classification ranges from SIL1, which is the least dependable, to SIL4, which is the most dependable. The DO-178B has a letter-based classification ranging from level "E", the least dependable, to level "A", which is the most dependable.
- **prescription level:** standards can be prescriptive (with an obligation to demonstrate compliance) or just give recommendations. However, in practice, deviating from recommendations can be difficult as this may require non-trivial work to justify it and to convince a certification body that is used to the well-established

recommended practices. In this way, the introduction of formal methods depends on whether they are simply mentioned, recommended or highly recommended.
- **scope:** this can address the software, the hardware or the complex system as a whole. It can also focus on specific parts of the development process (such as the development lifecycle, quality management, safety assessment, or system certification).

B.3.1.2 Standards and Formal Methods: An Overview

Generic IEC61508 Based on the SILs defined earlier, IEC 61508 classifies methodologies, techniques and activities as "not recommended", "recommended", "highly recommended" and so on. Among the "highly recommended" techniques for SIL4 are inspection and reviewing, using an independent test team and use of formal methods.

A single "highly recommended" technique is not enough to ensure the required safety, reliability and correctness: the standard requires a carefully chosen combination of appropriate techniques. Achieving this in a project requires a dedicated system engineering process as described in [6] to be defined.

Railway Sector The EN 50128 guidelines, issued by the European Committee for Electrotechnical Standardisation (CENELEC), address the development of "Software for Railway Control and Protection Systems", and constitute the main reference for railway signalling equipment manufacturers in Europe, with their use spreading to other continents and other sectors of the railway industry (as well as other safety-related industries).

In EN 50128, Formal Methods/Proofs are explicitly identified as relevant techniques/measures for software requirements specification, software architecture, software design, implementation, verification and testing and data preparation techniques. More precisely, they are "recommended" for SILs 1 and 2 and "Highly Recommended" for SILs 3 and 4. In particular, the Formal Methods cited include CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM, Z and B. In the ongoing 2011 revision of the standard, additional constraints are put on tools, especially code/data generation tools, with respect to the need for a specification and for evidence that the implementations comply with it.

Despite this, and success stories such as that of the automated metro line 14 in Paris (METEOR) [2], formal methods have not spread across the entire railway signalling industry, where much software is still written and tested in traditional ways (with the testing effort usually adding up to more than 50 % of the development effort). This lack of adoption is due to the investments needed to build up a formal methods culture, and to the high costs of commercial support tools. Moreover, equipment can be made to conform to CENELEC without applying formal methods [1].

However, EN 50128 requires that the bug detection and bug fixing activities include reviews of early phases. This costs more in terms of time and money. Consequently, companies are becoming interested in applying formal methods at the specification and design phases as it is the only solution that transfers the effort to the design team.

Another barrier is that, although mentioned in the standard, the certification bodies might not be familiar with Formal Methods, as most of the systems they certify follow a test-based approach. The first time a certification body is asked to certify a system whose justification relies on such a formal argument, they might require additional information to be convinced. Typically, there needs to be an indication of the level of expertise that an auditor should possess in order to perform certification for a company which bases its development on formal methods. Once certification is acquired, each new system can easily follow the same path. Siemens has gone through this process with the B-method, and it is now accepted by the certification bodies, so that subsequent projects have become easier to certify. In the specific market of metro lines, the B-method has even become the standard required by the market.

Aeronautic Sector DO-178B was published a while ago, in 1992. It does not recommend or propose a specific development process or methodology. The certification approach is to demonstrate compliance of the process and produced artefacts with a set of goals related to a number of lifecycle-related processes (planning, development, V&V, configuration and so on). As mentioned above, DO-178B ranks the software category by five dependability classes: from A (most dependable: catastrophic effect) to E (least dependable: no effect). The scope of the application of formal methods is the verification of software artefacts. In DO-178B, verification can also be achieved though a combination of review, analysis and testing activities.

With regard to Formal Methods, DO-178B categorises them as "alternative methods" because, at the time the document was produced (1992), they were assessed as inadequately mature. However, they can be used "as long as they can be demonstrated to address the goals of the standard, and their usage is adequately planned and described" [9].

A new revision DO-178C was published in January 2012. It is explicitly addressing formal methods to complement dynamic testing; they can be used selectively or as the primary source of evidence.

Automotive Sector The sector-specific standard is ISO 26262 (derived from IEC 61508). It recommends formal methods, but semi-formal methods are highly recommended. In such ISO standards, many development methods can be "recommended", but only one can be "highly recommended" (above all others). At the beginning of the debate on ISO 26262, it was suggested that formal methods should be highly recommended. After intensive lobbying from large automotive companies, which are convinced that formal methods will incur a significant cost overhead, semi-formal methods have been given the "highly recommended" status, while formal methods are just recommended.

B.3.1.3 Conclusion: Standards, Formal Methods and Possible Strategies

Some standards recommend or even highly recommend formal methods, but very few standards describe how to manage formal development. It is therefore difficult to prove that the development process complies with the standards. Certification authorities need to be convinced about this issue.

As formal methods are less widespread and certification authorities are less familiar with them than with the classical development methods, there is more work to be done than with other methods, especially the first time, even though formal methods can provide better arguments. However, the investment can be worth the effort, as shown by the Siemens case for B.

B.3.2 FAQ II: Are the Tools Associated with a Particular Formalism Backed by Responsive, Robust and Enduring Organisations?

Theme: Known Strengths and Weaknesses of Tools and Tool Providers (TOOL)
Role: High-Level Manager

B.3.2.1 Is There Guarantee of Long-Term Tool Availability and Support?

Industry projects may last tens of years from the development to the decommissioning of a system. It is therefore crucial for industry to ensure proper support throughout the complete project lifetime, including its retirement. Tools can be distributed under Open Source or Proprietary Licences. Each model comes with its own risk of disappearance (bankruptcy for proprietary code, community dissolution for Open Source). Given the niche market, securing the support is a nontrivial task (e.g., escrow for proprietary code, direct community involvement or support for Open Source).

B.3.2.2 Is the Tool Reliable?

Closed source reliability is a matter of trust that can be provided by a certification scheme for example. Concerns have been raised about whether Open Source tools can achieve higher reliability [4]. However, the large number of industrial-strength open source tools available nowadays, e.g. PVS and NuSMV suggests that they can. One reason for this is that they will potentially be peer-reviewed by many; another is that distributed development requires better defined and carefully designed interfaces at the design level. Furthermore, extensive test suites are often available for Open Source tools.

B.3.2.3 Is the Tool Scalable?

The ability to scale up depends on different factors. Tool-induced limitations may be due to the underlying formal technology, implementation problems (e.g., some bottleneck in a processing chain) or simply usability (e.g., limitations in managing large pieces of models). To assess scalability, references, feedback and reviews provide initial information that is useful for completely ruling out tools inadequate for industry. A second step is to challenge the tool on realistic case studies in various industry sectors, as the way models are built can also impact the ability to scale up. Open Source tools may be more likely to be unable to scale up, especially if they are still at the R&D stage. However, there are also highly scalable Open Source tools in the FM area (e.g., SPIN and NuSMV model-checkers, ACL2 and Isabelle theorem provers).

B.3.2.4 Is the Tool Usable?

It is important that tools facilitate various tasks when building or modifying a model, carrying out validation and verification activities, supporting teamwork, and so on. Commercial tools generally have better usability because special attention is devoted to this aspect, whereas Open Source tools tend to focus more on core functionality and efficiency, sometimes with only a command line interface.

B.3.2.5 Does the Tool Integrate Well in Industry Tool Chains?

The ability to integrate into existing industrial tool chains is fundamental. This requires that well-documented data formats are defined, and that APIs and binaries on specific OSs are available. This is an area where Open Source tool developers usually outperforms proprietary ones. Furthermore, Open Source developers often adopt open, standard data formats. By contrast, intense competition frequently pushes proprietary tool developers to keep internal data formats hidden.

B.3.2.6 What Is the Impact of My Tool Regarding Certification?

Using a formal tool in the design flow (i.e., at design time) can have an impact on the certification process, especially if the tool is generating production artefacts such as source code for systems requiring higher integrity levels. Evidence of correctness of the output produced by these tools has to be provided by various means: redundant implementation, extensive test coverage, and specific verification activities. Another supporting success story, the ProB tool used by Siemens and developed by the University of Düsseldorf is undergoing a qualification process for the railways EN-50128 standard.

B.3.3 Success Story I: Productivity Improvement of Data Consistency in Transportation Models

Table B.2 summarises a success story from the transportation sector.

Table B.2 Productivity improvement of data consistency in transportation models

Name	Productivity Improvement of Data Consistency in Transportation Models.
Keywords	B-Methods, assumption validation, model-checking, ProB tool.
Short description	The correct and safe behaviour of Metro models developed by Siemens relies on assumptions made about the topology of each specific site. Those assumptions have to be validated against a number of properties required of the model. Until now this task was achieved through custom rules in Atelier B, which was not very efficient at proving them automatically: it required a lot of manual work, about one person-month for a typical metro project. As an alternative to a proof-based approach, model checking with the ProB model-checker was successfully applied to this task. A dramatic productivity improvement was experienced on a real project (San Juan metro line): the same faults could be discovered completely automatically and within a few minutes of execution time, whereas the previous approach required about one man-month of effort.
Source	Siemens, University of Düsseldorf.
Benefits	• Dramatic productivity improvement for the problem of data validation • Better diagnostics of failed property using output
Limitations	• Specific to transportation domain; extensibility to other sectors to be studied • The ProB Tool must undergo qualification/certification (ongoing)
Elaboration	The success story is fully described in the paper [7] and its qualification report is an internal deliverable of the DEPLOY project. Below we highlight the main arguments and lessons learned: • **Full automation and quick response time.** Checking the properties takes 4.15 seconds, and checking the assertions takes 1,017.7 seconds (i.e., roughly 17 minutes) using ProB 1.3.0-final.4 on a MacBook Pro with a 2.33 GHz Core2 Duo. Manually it takes a month for one person to solve the same problem (San Juan line). • **Easier diagnostics.** Another advantage of model checking over proof is that it provides a counterexample that will be of greater help in understanding the cause than the information found in a failed proof. Furthermore, a specific visualisation tool was developed to make it easier to understand a counterexample. • **Confidence in the results.** An important issue is the qualification of the tool in order to have full confidence in the results produced. With respect to this, the model-checking approach implemented has some form of redundancy: a property is expressed in a positive and a negative form. The produced result should be consistent; otherwise, this indicates a problem. The problem could be located at lower levels; in fact, two bugs were discovered in the Prolog interpreter. The absence of problem is, however, no guarantee of correctness and does not remove the need for tool validation.

Table B.2 (Continued)

Elaboration	• **Qualification process.** In order to be used as a correctness argument in the development process of a qualified product, the tool must be qualified itself. Without such qualification, the tool might be used as a debugger is used by developers, to speed up development, and to conduct manual verification only when the developer is confident that the verification will succeed. The ProB tool has undergone a qualification process. The qualification argument is mainly based on testing various parts of the tool, and on implementing internal sanity checking mechanisms to check that the tool is consistent with itself. Two testing approaches were deployed in parallel: global end-to-end testing of industrial cases and unit testing of specific internal modules. Part of unit testing involved automatically generated test cases. Furthermore, an independent tool was used to double-check the results of some unit tests. An additional lesson for academics is the need to cover the full language used in industry while prototyping. As industry will not normally adapt its models to suit an academic tool, it is necessary to consider extending the tool at the industrialisation phase. The amount of work involved should be evaluated early enough in the R&D process. The ProB tool is already a success as Siemens plans to use it to replace Atelier B for formal data validation. This would require the validation of ProB for SIL4 use, which is being investigated.
Consolidation	Model checking has been tried in the past in the railways domain to validate some interlocking properties. Specific properties (robustness and locality) allowed optimisation of the general CTL algorithm. The case was only validated on a small railway interlocking involving two stations [5]. A CSP approach was also investigated together with the FDR model-checker [Wint02]. Useful counterexamples related to safety requirements could be produced using some examples. However, a conclusion was reached that languages based on process algebra, such as CSP, were not adequate for modelling this domain (especially for the control table). They resulted in models that were difficult for practitioners to understand or validate.
Related FAQs	**of Interest to Project and QA Managers** • **QI-PQAM-2:** How is the productivity of the various stages of the system development cycle affected when formal engineering methods are used? • **ExFac-HM-1:** What is the position of standards regarding formal methods in my industry segment (e.g., are they enforcing them, highly recommending, etc.)? **of Interest to Engineers and Analysts:** • **TOOL-EA-2** Do tools automate all tedious tasks?

B.4 Typical Scenarios Exploiting Shared Evidence Material

A few customary situations faced by professionals with different roles in industrial software-intensive system development are presented, along with the kinds of specific questions they often have about formal methods and their usage in actual in-

dustrial development. Subsequently, a brief explanation shows how they can browse the evidence repository to find elements of answers to their questions.

Scenario I I am a high-level manager who has heard that others in the sector have successfully used formal methods. My staff has no prior experience of them. My question is: "How should I proceed in the evidence repository to better understand if FM could be used in my company, too?"

Scenario II I am an engineer in an SME. Our development team has successfully used advanced static analysis tools during the debugging and testing phases of product development. However, in a recent project, important requirements remained implicit until the customer validation phase. Subsequently, a significant overhaul of the architecture was needed to handle these. Static analysis was clearly no help. My question is: "How should I proceed in the evidence repository to better understand how our development team could start using formal methods earlier in the development lifecycle to increase the chances of identifying all important requirements after the design phase?"

Scenario III I am a QA engineer. I saw a presentation by another company at a conference, which showed QA and safety engineers from other fields using tools to generate test cases for various kinds of coverage. My question is: "Does your repository contain information on test generation formalisms, approaches and tools and their eventual applicability to my sector?"

Scenario IV I am a product (line) manager. I have heard of several success stories on the use of formal methods but my company already has well-tested components that fulfil our quality requirements. Our project mostly consists in configuring these existing components and integrating them. My question is: "Does your evidence repository demonstrate the use of formal methods to ease component integration, even if these components were not developed using formal methods?"

These situations traditionally faced by industry illustrate how one can navigate the evidence repository to find FAQs and success stories containing information relevant to one's context. For each of the scenarios above, a suggested navigation approach will be explained. In general, there are only two ways to start navigating the evidence repository. The top-down approach starts from the FAQs while the bottom-up one starts from success stories. Most readers will come up with a general question and therefore find the top-down approach to be more appropriate. However, in some cases, one may learn about a specific success story at a conference or while reading an article, for example, in Scenario IV. In this case, it may then be more

appropriate to look for this or a similar success story on the evidence repository wiki. Below, the two navigation approaches are briefly discussed and then applied to each of the above scenarios.

I—Top-Down Navigation Approach When one has a general question about formal methods in specific organisational context, it is assumed that this is a rather non-focused exploratory search. In this case, the most optimal way of looking for relevant information is as follows.

- Identifying the themes in the FAQs that are relevant to the question.
- For each theme of interest, reviewing the questions specific to one's role to identify a subset of relevant questions.
- While reading answers, identifying pointers to specific reference papers and success stories; then deepening the search by reading success stories in the evidence repository or the referenced scientific literature.

It is important to note that questions are sometimes worded to highlight broad cross-industry concerns. Such questions can be answered from quite different viewpoints, taking into account different sectors, different types of formal methods and even different phases of the lifecycle. Thus, when first reading the FAQ, readers must understand that the answer may focus not only on their topic of interest. This broad coverage for some questions is provided deliberately, so that the reader will find more general answers than was initially anticipated. Nevertheless, readers will be free concentrate on a particular part of the answer or to explore other information provided in the answer. An answer is normally subdivided into subsections to make this easier: for example, the answer that considers differences in the use of formal methods at different phases of the development lifecycle will be explicitly broken down with one subsection per phase of the lifecycle.

In other words, when initiating a search from the FAQs, the reader must keep an open mind and not search for direct answers to only his/her context. Obtaining tailored answers would require specific studies that can only be achieved through a dedicated subcontract that goes far beyond the evidence repository.

II—Bottom-Up Navigation Approach The reader may start an investigation by considering a success story that closely relates to his/her specific need. Although s/he may be tempted to stop the investigation at that point and to start implementing the approach described in the success story, appropriate caution should be taken. First, his/her specific context is likely not to match the context in which the success story developed. Second, continuing the investigation may highlight new possibilities not initially anticipated by the reader. It is therefore worth spending more time navigating from the success story to its context description. To search for a success story and identify those potentially similar to one read or heard about elsewhere, a review is needed of the complete title list of success stories, which are currently organised by industry sectors. At the moment, the list is still short enough for full browsing to work; it will be subclassified when the number of success stories exceeds 30 or 40. Success stories in the evidence repository include cross-references

to FAQs where they are cited. It is then possible to navigate to an FAQ linked to the success story and identify other potential questions of interest with the same FAQ theme. Finally, success stories are classified by sectors; thus, it also makes sense to review the success stories related to his/her sector.

Scenario I In this scenario, a high-level manager starts with a very broad enquiry. Thus, the manager should use a top-down approach for navigating the evidence repository, starting with the selection of themes. His/her initial concern relates to the lack of current expertise on formal methods. Consequently, the whole theme on training would provide relevant information. Furthermore, the high-level manager should be informed about the different means for including formal methods in the development lifecycle. In particular, it may be possible that the organisation's context makes it possible to restrict the application of formal methods to a small team of experts rather than having to train the entire engineering staff. In addition, early in an enquiry, the high-level manager is likely to be interested in the general topic theme. Thus, in this context, the high-level manager would be very interested in the following FAQ themes and questions:

- Impact on an organisation with regard to training scope and resourcing
  - **TSP-HM-1** What is the cost or effort needed to train engineers/analysts to use a new formalism given their previous experience with formal engineering methods?
- Understanding the impact on the Software/System Development Process The capability of phasing the learning of a formal method in an organisation and eventually of limiting the number of those who must understand and become an expert in a formal method
  - **CIF-HM-1** Is it possible to phase the learning and use of a formal method? (In other words, can the introduction of a formal method be done gradually or must the entire development staff understand (and master) the formal method at once?)
  - **CIF-HM-2** How do organisational procedures used in various system development lifecycle processes need to be adapted when formal methods are introduced?
  - **CIF-PQAM-1** Can the use of a formal method be hidden from the majority of development and management teams, except for a few selected experts who will use it (perhaps even without other team members knowing)?
  - **CIF-PQAM-2** What are the risks of hiding the use of formal methods and what are the strategies for mitigating them?

- The capability of phasing the migration to the use of a formal method (given the existence of products initially developed without using formal methods)
  - **MF-HM-1** Is it possible to iteratively migrate an existing system to the formal method use?
- General topics of concerns related to formal methods in industry
  - **G-HM-3** What are formal methods?
  - **G-HM-1** What are the main benefits and risks of using formal methods in product development?
  - **G-HM-2** Why have formal methods failed to achieve a breakthrough in the market for such a long time?

Scenario II In this scenario, an engineer states that the development team already has some initial expertise in using advanced static code analysis tools, which usually perform formal verification on the code at the implementation and test phases. These tools help with verification (Am I building the product right?) but not with validation (Am I building the right product?). The engineer therefore wonders how using formal methods at the requirement and design phase could help. Although the engineer's concern seems quite specific, it is still appropriate to perform top-down navigation. From the engineer's perspective, it seems clear that the overall theme of "Understanding the impact on the Software/System Development Process" is of interest. While s/he may then be drawn to other themes, in the list below we identify only themes and questions related to his initial interest.

- The impact on the quality of product (and its work product) developed using formal methods,
  - **QI-PQAM-1** What impact does the use of formal engineering methods have on identifying issues at each phase of the development cycle? (Note: while reading the answer to this FAQ, the engineer will find a link to a success story on how formalising requirements helped improve their quality, through identifying ambiguity in informal requirements among other things)
  - **QI-PQAM-2** How is the productivity of the various stages of the system development cycle affected when formal engineering methods are used (as compared to when non-formal development methods are used)?
  - **QI-EA-1/** How have various formal methods (and modelling patterns in these formal methods) developed to increase various quality attributes

of systems/products such as safety, reliability, guarantees regarding real-time properties, concurrency, availability and maintainability?

- The capability to exploit formal models at various stages of the development process,
  - **EM-EA-3** Is there evidence of formal methods being used to improve the quality of requirements and design documents expressed in less formal notations such as UML?
  - **EM-PQAM-2** Does the use of a formal engineering model have any beneficial effect on requirements and design traceability?
  - **EM-PQAM-3** Is there any guidance on the cost/benefit trade-off in using different validation techniques (testing, model checking, proof) between formal and non-formal methods as well as between formal methods?
  - **EM-EA-2** What formal methods offer modelling languages, tools or theories suited to a particular domain, which may be a vertical (Automotive) or a horizontal domain(System Security)?
  - **EM-QAP-1** Does the use of formal engineering methods help in designing tests?
  - **EM-QAP-2** Does the use of formal engineering methods help provide test oracles?
- The capability to phase the learning of a formal method in an organisation and eventually to limit the number of those who must understand and become an expert in a formal method
  - **CIF-HM-1** Is it possible to phase the learning and use of a formal method? (In other words, can the introduction of a formal method be done gradually or must the entire development staff understand (and master) the formal method at once?)
  - **CIF-HM-2** How do organisational procedures used in various system development lifecycle processes need to be adapted when formal methods are introduced?
  - **CIF-PQAM-1** Can the use of a formal method be hidden from the majority of development and management teams except for a few selected experts who will use it (perhaps even without other team members knowing)?
  - **CIF-PQAM-2** What are the risks of hiding the use of formal methods and what are the strategies for mitigating them?
  - **CIF-EA-1** How far can the use of a formal method be automated?
- The capability to phase the migration to the use of a formal method (given the existence of products initially developed without using formal methods)
  - **MF-PQAM-1** Does an iterative migration to the use of a formal method on an existing system require complete rephrasing of requirements, reengineering of models, rewriting of test plans and so on?

– **MF-EA-1** Is there an efficient staged approach to introducing the use of a formal method iteratively on an existing system (e.g., is there a preferred order of introducing the use of several formal methods?)?

According to the FAQs, a significant proportion of the questions above concern high-level and project managers. However, in this case the questions are highlighted as being "of interest" to an engineer. This should not cause a problem. Indeed, the classification based on roles indicated that the question will usually be of interest to a given role; hence, the answer is adapted to that role. This means that, for example, an answer targeted at a high-level manager will not include technical content. In other words, an engineer reading a question/answer entry for a high-level manager or even a project manager should understand that the answer is provided from the viewpoint of a manager.

Scenario III In this scenario, the QA engineer starts from a particular success story. Hence, this is one of the rare cases where bottom-up navigation is probably preferable. The best case scenario is when the success story is already part of the evidence repository. In this case, the safety engineer can simply read it in the wiki and follow the links to relevant FAQs and their answers. However, if the success story has not appeared in the evidence repository yet, the engineer is first advised to send a email to the fm4industry editorial board to indicate that an interesting success story has been published. Second, the engineer can then browse the current list of success stories, possibly to find a success story similar to the one read or heard about. In this case, after a quick review of various story titles, the QA engineer would discover that there is one which presents the benefits of model-based testing automation. A pointer makes it possible to navigate from the success story to a specific FAQ. Subsequently, if interested, the engineer can continue with the bottom-up browsing of the FAQs.

Success story of Interest:

- In the Business Information Domain—Benefits of Model-Based Test Automation

From this success story, a link to the following FAQs is provided:

- **EM-QAP-1** Is it possible to take advantage of formal models to automate a part of the QA tasks?

Subsequently, the QA engineers may find other FAQs in the same category "Exploiting Models (EM)" interesting:

- **EM-QAP-2** Does the use of formal engineering methods help provide test oracles?
- **EM-PQAM-3** Is there any guidance on the cost/benefit trade-off in using different validation techniques between various formal methods as well as between formal and non-formal methods?

After having read the complete material, the QA engineer should feel more confident about the applicability of formal methods in his/her organisation. S/he may then be interested in reading additional FAQs to better understand how to proceed, in particular, FAQs on training and on controlling the deployment of formal methods in an organisation.

Scenario IV In this scenario, a product line manager indicates that his/her company will not switch to an approach involving a complete reformulation of specifications using formal methods, but the organisation is willing to investigate the use of formal methods for specifying the integration of existing components. Although the manager mentions success stories, they are not specific; hence s/he should probably initiate his/her search using the top-down navigation. Unfortunately, FAQs are not worded so as to make it easy for the manager to directly map the themes to his/her concern with the use of formal methods for integration. However, by analysing the themes a little deeper, the manager would probably become interested in the themes "Control Impact of Formalism (CIF)", "Migration to a Formalism (MF)" and possibly "Exploiting Models (EM)".

Reading the various questions in these 3 themes, the manager could probably identify the following FAQs as potentially pertinent:

- Question ID: MF-PQAM-1. Does an iterative migration to the use of a formal method on an existing system require complete rephrasing of requirements, reengineering of models, rewriting of test plans and so on?
- Question ID: MF-HM-1. Is it possible to migrate an existing system to the use of a formal method iteratively?
- Question ID: CIF-HM-1. Is it possible to phase the learning and the use of a formal method? (In other words, can the introduction of a formal method be done gradually or must the entire development staff understand (and master) the formal method at once?)

Although none of the questions above are directly related to the manager's concern, the fact that they mention iterative migration and gradually increasing learning indicates that formal methods do not need to be applied to the entire system at once. Thus, the manager may wonder if certain answers address the topic of formal specification of system integration as one of the ways of migrating to the use of formal

methods. Unfortunately, none of these questions have yet been answered in the literature or in the course of DEPLOY.

In such a situation, as a last resort, the manager would be advised to browse success stories to identify any similarities with his concerns. As a result, the manager may find that "Adoption Eased by Using the Formal Models Behind Domain Specific Notations" could somehow provide guidance, since the integration language can be viewed as a specific notation for system integration purposes.

The success story does not provide a perfect answer to the manager but seems nonetheless to indicate that a high-level language, in this example, a Business Process Modelling language, can be used and extended to perform formal verification. Although it requires an open mind and a leap in thinking, the manager could draw a parallel between two high-level languages: the one in the success story is used for expressing a Business Process, while in his case, this kind of language is used for system integration. If the manager can make this observation, he should realise that it may be possible to attach a formal method to the traditional system integration language used at the company. Consequently, the success story provides a hint to the manager that a formal method can potentially be used for specifying system integration. Furthermore, it may be possible to hide the use of the formalism from system integrators, who will continue to use the system integration languages they know.

Making such a leap in reasoning may not be possible for the manager. In this case, it is in his/her direct interest to communicate with the fm4industry board, who can guide readers and help them correctly interpret the material in the evidence repository.

B.5 Setting Up Private Evidence Collection and Selective Public Sharing

Setting up an evidence repository for a research project such as DEPLOY is useful for a company, even if this is different from what it may normally aim to do. In particular, for an organisation to make a sound decision on its innovation cycle, it is important to spend time sorting and normalising information collected from trials. As pointed out at the beginning of this appendix, information on innovation should also be qualitative rather than uniquely quantitative. The approach used in the course of DEPLOY to obtain the needed information is based on the FAQs and the enhanced success stories previously described. These two instruments provide appropriate means to gather information for any research transfer project. However, this entails having to identify relevant themes and questions to the target topic. In this section, we assume that an organisation may be willing to roll out an evidence repository as part of its engagement with formal methods. In this case it can directly use the FAQ themes and questions from DEPLOY.

Below are the important aspects of creating an evidence repository:

- Having a duplicate evidence repository, with one of them kept private for storing work and progress and the other one used to publish for the target audience (for example, to make information available to a few departments, a whole enterprise or even the world)
- Nominating a trusted editorial board to vouch for the independence and the overall quality of the information provided. It may even be possible to require a particular turnover of the editorial board over time (e.g., by new nominations or elections)
- Controlling the editing of published evidence material
- Facilitating commenting and obtaining feedback on work in progress as well as on the published evidence material
- Allocating the necessary resources for evidence collection, sorting and transformation

During DEPLOY, two distinct wiki (based on a Redmine forge) were used to address the various aspects above. The first one requires a login and a password for read and write privileges, and was useful for keeping potentially confidential information under control. The second one requires appropriate credentials for write authorisation, but is viewable by the public. Material was moved from one to the other in a controlled way. For an even easier setup, the Mediawiki tool is now used for the final public repository. It provides the appropriate features for controlling read/write access, and it also includes a discussion board with message threading for commenting and providing feedback.

Clearly, organisations will initially prefer to keep most information private at the organisation or at an even more restricted level. However, once the information has matured, it is usual for well-crafted pieces of evidence to be used as advertising. For example, publishing information about training can indirectly show competitors that an organisation is promoting the adoption of formal methods, whereas publishing data on exploiting formal models for various purposes and on reusing formal models will certainly display a growing expertise. This is exactly the type of information sought for publication on the www.fm4industry.org site. However, the caveat is that the information must be validated and approved by the editorial board of fm4industry, which incidentally may evolve over time. The board's basic validation principle is as follows: the presentation of a peer-reviewed article at a workshop or a conference is sufficient guarantee for acceptance. It is nonetheless possible that other information may meet sufficient standards to be accepted; for example, a white paper published by an organisation as part of an EU research project supported by an academic partner may be accepted for publication.

The main goal of fm4industry.org is to promote the use of formal methods in industry and become an almost self-sustainable site to which external parties would make contributions of their own will. However, to guarantee the validity and quality of information, a minimal threshold for acceptance of new publications must be enforced.

References

1. Bacherini, S., Fantechi, A., Tempestini, M., Zingoni, N.: A story about formal methods adoption by a railway signaling manufacturer. In: Proceedings of the 14th International Symposium on Formal Methods, Hamilton, Canada, August 21–27, 2006, pp. 179–189. Springer, Berlin (2006)
2. Behm, P., Benoit, P., Faivre, A., Meynadier, J.M.: Meteor: A successful application of B in a large project. In: Proceedings of Formal Methods 1999. Lecture Notes in Computer Science, vol. 1708, pp. 369–387. Springer, Berlin (1999)
3. Bozzano, M., Villafiorita, A.: Design and Safety Assessment of Critical Systems. CRC Press (Taylor and Francis), an Auerbach Book, Boca Raton (2010)
4. Craigen, D.: Formal methods adoption: What's working, what's not! In: Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking, pp. 77–91. Springer, London (1999)
5. Eisner, C.: Using symbolic CTL model checking to verify the railway stations of Hoorn-Kersenboogerd and Heerhugowaard. Int. J. Softw. Tools Technol. Transf. 4(1), 107–124 (2002)
6. Kars, P.: Formal methods in the design of a storm surge barrier control system. In: Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems, pp. 353–367. Springer, London (1998)
7. Leuschel, M., Falampin, J., Fritz, F., Plagge, D.: Automated property verification for large scale B models. In: Proceedings of the 2nd World Congress on Formal Methods, FM '09, pp. 708–723. Springer, Berlin (2009)
8. Mediawiki. http://www.mediawiki.org
9. RTCA/DO-178B. Software considerations in airborne systems and equipment certification (1992)
10. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.S.: Formal methods: Practice and experience. ACM Comput. Surv. **41**(4), 1–36 (2009)