Standard Conformance-by-Construction with Event-B

Ismail Mendil, Yamine Aït-Ameur, Neeraj Singh, Dominique Méry, Philippe Palanque

Standard Conformance-by-Construction with Event-B

I. Mendil[1], Y. Aït-Ameur[1], N. K. Singh[1], D. Méry[2], and P. Palanque[3]

[1]INPT-ENSEEIHT/IRIT, University of Toulouse, France
[2]Telecom Nancy, LORIA, Université de Lorraine, France
[3]IRIT, Université de Toulouse, France
{ismail.mendil,yamine,nsingh}@enseeiht.fr, dominique.mery@loria.fr,
palanque@irit.fr

Abstract. Checking the conformance of a system design to a standard is a central activity in the system engineering life cycle, *a fortiori* when the concerned system is deemed critical. Standard conformance checking entails ensuring that a system or a model of a system faithfully meets the requirements of a specification of a standard improving the robustness and trustworthiness of the system model. In this paper, we present a formal framework based on the correct-by-construction Event-B method and related theories for formally checking the conformance of a formal system model to a formalised standard specification by construction. This framework facilitates the formalization of standard concepts and rules as an ontology, as well as the formalization of an engineering domain, using an Event-B theory consisting of data types and a collection of operators and properties. Conformance checking is accomplished by annotating the system model with typing conditions. We address an industrial case study borrowed from the aircraft cockpit engineering domain to demonstrate the feasibility and strengths of our approach. The ARINC 661 standard is formalised as an Event-B theory. This theory formally models and annotates the safety-critical real-world application of a weather radar system for certification purposes.

Keywords: Standard conformance· Safety properties· Correctness-by-Construction· Event-B and Theories· ARINC 661· Critical Interactive Systems.

1 Introduction

Checking the standard conformance of a system design is a central activity in the system engineering life cycle, *a fortiori* when the concerned system is deemed critical. Standard compliance checking entails ensuring that a system or a model of a system faithfully meets the requirements of a standard, in particular domain and certification standards, improving the robustness and trustworthiness of the system model.

In many cases, conformance of system design models and/or implementation to a standard is achieved by informal or semi-formal processes like argument-based reports produced through model reviews, testing and simulation, experimentation, and so on [28]. Although, these qualification methods have proven to be valuable for system engineering in areas like transportation systems, medical devices, power plants, etc., formal checking of conformance, as advocated by the DO178-C, is more trustworthy and has many advantages, including extensive case coverage and availability of automatic verification capabilities such as model-checking and theorem proving.

Context of the work. As part of the French ANR FORMEDICIS[1] project, we have studied the problem of ARINC 661 [8] standard conformance for CIS (Critical Interactive Systems). ARINC 661 is a standard for the development of flight deck display interfaces. In fact, modern cockpit designs increasingly rely on the ARINC 661 standard series used in several airplane development programs, e.g. Airbus A380, A350, and A400M, as well as the Boeing 787, 737MAX, KC-46A, and B777X[2].
Our claim is that it is possible to check that a formal design model complies with domain standards formalised as a theory with data types, operators, axioms and theorems.
Standard conformance addressed by our approach consists in transferring, to formal design models, theorems proved, once and for all, in the theory formalising a domain standard specification. The conformance is checked by proving the well-definedness proof obligations generated when using the theory operators. Note that we do not address the process of building these theories which requires to move from text-based standard documents to formal theories. Building such theories is out of the scope of this paper. Such processes have been addressed in [12,13,14] to link

text-based standards with formalised theories expressed in Isabelle/HOL.

So, the *goal of this paper* is to demonstrate how to check the compliance of formal design models with domain standards expressed as theories. The overall approach is exemplified on ARINC 661 standard and weather radar system application.
Our contribution. In this paper, we present a formal framework based on the correct-by-construction Event-B method and related theories for formally checking, by construction, the conformance of a formal system model to a formalised standard specification. This framework formalises engineering standard concepts and rules as an ontological Event-B theory. To demonstrate the feasibility and strengths of our approach, we report on our experiments, from the FORMEDICIS project, addressing an interactive system available in aircraft cockpits. Relying on domain ontologies as a ground knowledge model, the ARINC 661 standard is formalised as an Event-B theory which formally *annotates* the model of the real-world weather radar system.
Organisation of this paper. Next section is a brief review related to conformance and certification and Section 3 is devoted to a summary of the Event-B method. Section 4 contains the description of the CIS and the ARINC661 standard. Our framework is presented in Sections 5 and 6 and its application is given in Section 7. Section 8 provides an assessment of the approach. Last, Section 9 concludes this paper.

2 Certification and conformance

According to ISO, a standard is defined as: *Standards are documented agreements containing technical specifications or other precise criteria to be used consistently as rules, guidelines or definitions of characteristics, to ensure that materials, products, process and services are fit for their purpose* [26].

The use of standards has a number of potential advantages. It plays an important role for the development of complex systems, including both product-based and process-

[1] FORmal MEthods for the Development and the engIneering of Critical Interactive Systems (CIS) `https://anr.fr/Projet-ANR-16-CE25-0007`
[2] https://www.aviation-ia.com/activities/cockpit-display-systems-cds-subcommittee

based developments. This process is both time-consuming and difficult. Some work focuses on integrating standards into process development. In [17], the authors propose a model for standards conformance by introducing lightweight mechanisms. In [9], a framework based on Natural Language Semantics techniques is presented. It assists in the processing of legal documents and standards through building a knowledge base that includes logical representations. In [16], the authors propose a step-by-step process for conformance checking that includes process modeling and execution. Similarly, [34] shows how to implement the conformance relation on transition systems. Nair et al. [35] provide a detailed survey how practitioners deal with safety evidence management for critical systems and they also draw the conclusion that there is a limited use of safety evidence in industries based on empirical evaluation.

In recent years, assurance cases have been used in critical domains to establish system safety by presenting appropriate arguments and evidences [30,39]. The chosen evidences are always questionable, regardless of how they are established or how much confidence we have in them. There are several approaches to justifying confidence, such as eliminative induction [19], quantitative estimation [22], provided as claims in the assurance case [20]. Wassyng et al. [43,42] propose an *Assurance Case Template* used in the development of critical systems and their certification within a domain model.

Regarding Event-B [2] and B [1] methods, Fotso et al. [41] present a specification of the hybrid ERTMS/ETCS level 3 standard, in which requirements are specified using SysML/KAOS [32] goal diagrams that are translated into B, and domain-specific properties are specified by ontologies using the SysML/KAOS domain modeling language, which is based on OWL [7] and PLIB [27]. Last, we mention the work of [10,11] which uses the RSL language to model engineering domains.

The interest and motivation of handling domain knowledge has been discussed and argued in [5]. In this paper, we propose to use the capability of Event-B theories to improve the explicitation and integration of domain knowledge in design models. A key advantage of our proposed approach is that the proof of domain properties holding in the design models is explicit since a Well-Definedness (WD) proof obligation (PO) is generated. Such WD POs are generated for each theory defined operator, it states that each parameter belongs to the domain operator. This is particularly relevant for partial operators. Obviously, the approach exempts from explicitly specifying domain properties on the model side. Compared to our approach which relies on an ontology modelling language referenced by formal design models, none of the mentioned work use a shared modelling language.

3 Event-B

Event-B [2] is a *correct-by-construction* method based on set theory and first-order logic. It relies on state-based modelling where a set of events allows for state changes.

3.1 Contexts and machines (Tables 1.b and 1.c)

The `Context` component describes the static properties of a model. It introduces the definitions, axioms and theorems needed to describe the required concepts using *carrier sets* s, *constants* c, *axioms* A and *theorems* T_{ctx}. `Machine` describes the model be-

haviour as a transition system. A set of guarded events is used to modify a set of states using Before-After Predicates (BAP) to record variable changes. They use *variables* x, *invariants* $I(x)$, *theorems* $T_{mch}(x)$, *variants* $V(x)$ and *events* evt (possibly guarded by G and/or parameterized by α) as core components.

Refinements. Refinement (not used in this paper) decomposes a *machine* into a less abstract one with more design decisions (refined states and events) moving from an abstract level to a less abstract one (simulation relationship). Gluing invariants relating abstract and concrete variables ensure property preservation.

Theory	Context	Machine
THEORY Th **IMPORT** Th1, ... **TYPE PARAMETERS** E, F, ... **DATATYPES** **Type1**(E, ...) **constructors** **cstr1**(p_1: T_1, ...) **OPERATORS** **Op1** <nature> (p_1: T_1, ...) **well−definedness** $WD(p_1, ...)$ **direct definition** D_1 **AXIOMATIC DEFINITIONS** **TYPES** A_1, ... **OPERATORS** **AOp2** <nature> (p_1: T_1, ...): T_r **well−definedness** $WD(p_1, ...)$ **AXIOMS** A_1, ... **THEOREMS** T_1, ... **END**	**CONTEXT** Ctx **SETS** s **CONSTANTS** c **AXIOMS** A **THEOREMS** T_{ctx} **END**	**MACHINE** M **SEES** Ctx **VARIABLES** x **INVARIANTS** $I(x)$ **THEOREMS** $T_{mch}(x)$ **VARIANT** $V(x)$ **EVENTS** **EVENT** evt **ANY** α **WHERE** $G(x, \alpha)$ **THEN** $x :\mid BAP(\alpha, x, x')$ **END** ... **END**
(a)	(b)	(c)

Table 1: Global structure of Event-B Theories, Contexts and Machines

Proof Obligations (PO) and Property Verification. Table 2 provides a set of, automatically generated, POs to guarantee Event-B machines consistency.

(1) Ctx Theorems (ThmCtx)	$A(s,c) \Rightarrow T_{ctx}$ `(For contexts)`
(2) Mch Theorems (ThmMch)	$A(s,c) \wedge I(x) \Rightarrow T_{mch}(x)$ `(For machines)`
(3) Initialisation (Init)	$A(s,c) \wedge G(\alpha) \wedge BAP(\alpha, x') \Rightarrow I(x')$
(4) Invariant preservation (Inv)	$A(s,c) \wedge I(x) \wedge G(x,\alpha) \wedge BAP(x,\alpha,x') \Rightarrow I(x')$
(4) Event feasibility (Fis)	$A(s,c) \wedge I(x) \wedge G(x,\alpha) \Rightarrow \exists x' \cdot BAP(x,\alpha,x')$
(5) Variant progress (Var)	$A(s,c) \wedge I(x) \wedge G^A(x,\alpha) \wedge BAP(x,\alpha,x') \Rightarrow V(x') < V(x)$

Table 2: Relevant Proof Obligations

Core Well-definedness (WD). In addition, WD POs are associated to all built-in operators of the Event-B modelling language. Once proved, these WD conditions are used as hypotheses to prove further proof obligations.

3.2 Event-B extensions with Theories

In order to handle more complex and abstract concepts beyond set theory and first-order logic, an Event-B extension for supporting externally defined mathematical objects has been proposed in [3,15]. This extension offers the capability to introduce new data types by defining new types, operators, theorems and associated rewrite and inference rules, all bundled in so-called *theories*. Close to proof assistants like Isabelle/HOL [36] or PVS [37], they are convenient when modelling *concepts unavailable in core Event-B*.

Theory description (See Table 1.a). Theories define and make available new data types, operators and theorems. Data types (`DATATYPES`) are associated with *constructors*, i.e to build inhabitants of the defined type that may be inductive. A theory defines various

operators further used in Event-B expressions. They may be *FOL predicates* or *expressions* producing actual values (<nature> tag). Operator applications can be used in other Event-B theories, contexts and/or machines. They *enrich the modelling language* as they may occur in axioms, theorems, invariants, guards, assignments, etc.

Operators may be defined either explicitly using an explicit ("direct") equivalent definition, in the `direct definition` clause, (case of a constructive definition), or defined axiomatically in the `AXIOMATIC DEFINITIONS` clause (a set of axioms). Last, a theory defines axioms, completing the definitions, and theorems. Theorems are proved from the definitions and axioms.

Many theories have been defined for sequences, lists, groups, reals, differential equations, etc. Theories can be extended (`Imports`) to define more complex theories and instantiated (in `context`) by providing concrete type parameters.

Well-definedness (WD) in Theories. An important feature provided by Event-B theories is the possibility to define *well-definedness* (WD) conditions. Each defined operator (partially defined) is associated to a condition guaranteeing its correct definition. When it is applied (in an Event-B expression), this WD condition generates a PO requiring to establish that this condition holds, i.e. the use of the operator is correct. The theory developer defines these WD conditions for the partially defined operators. All the WD POs and theorems are proved using the Event-B proof system.

Event-B proof system and its IDE Rodin. Rodin[3] is an open source IDE for modelling in Event-B. It offers resources for model editing, automatic PO generation, project management, refinement and proof, model checking, model animation and code generation. Event-B's theories extension is available under the form of a plug-in. Theories are tightly integrated in the proof process. Depending on their definition (direct or axiomatic), operators definitions are expanded either using their direct definition (if available) or by enriching the set of axioms (hypotheses in proof sequents) using their axiomatic definition. Theorems may be imported as hypotheses and used in proofs. Many provers like predicate provers, SMT solvers, are plugged to Rodin as well. In addition to the known success of the Event-B and B methods in dealing with complex formal system developments, the choice of Event-B as a ground modelling formal method is motivated by the provided abstract modelling level. Indeed, it offers first a built-in mechanism (state and transitions) associated to an inductive proof process for invariants and second an extension mechanism to define theories with operators associated to WD conditions that generate POs when applied. These WD POs are fundamental for our approach to conformance checking. In addition, animators and model checkers like ProB [33] are useful to validating the defined theories over model instances. Finally, other techniques could have been used as long as they could check the correctness of operator applications and they are connected to the Rodin platform.

4 Case study: ARINC 661 + Multi-purpose interactive application

4.1 ARINC 661 standard specification: an extract

ARINC 661 [8] is the Cockpit Display System (CDS) standard for communication protocols between interface objects and aircraft systems. It has been used for the development

[3] Rodin Integrated Development Environment `http://www.event-b.org/index.html`

of interactive applications in, for instance, Airbus A380 and Boeing B787. In ARINC 661 specification standard, an interactive application is called a User Application (UA) that receives input from the the CDS and triggers actions in aircraft systems. Such input are produced by the flying crew manipulating specific input devices such as a KCCU (Keyboard Cursor Control Unit). UAs also receive information flow from aircraft systems that is presented to the flying crew using interactive objects which behaviour and parameters are described in the standards. The current version of the standard (called supplement 7 for part 1) describes in about 800 pages a set of definitions and requirements for the CDS and its graphical objects (called widgets).

Communication between the CDS and UA is defined based on the identification of widgets defined in the Widget Library. Different levels widget states are available. 1) *Visibility* level indicating whether the widget is visible or not. 2) *Inner level specific states of a widget* which represents the core of the widget behavior as well as its functional objectives. Examples of inner states for a *CheckButton*, are two stable inner states: *Selected* and *Unselected*. 3) *Interactivity levels* are: *enabled* or *disabled*. An enabled widget is ready to receive input from crew member interaction. Last, 4) *visual* level (visual representation) internal behavior of the widget inside the CDS. Examples are "Normal" and "Focus" denoting different interactions style (e.g. in the "Focus" state a standard interaction such as spacebar keypress would trigger the widget). Usually, implementations of CDS present different graphical appearances for the widgets depending on their state. It is important to note that such rendering is outside the scope of the standard.

4.2 Multi-purpose interactive application & Weather radar system

We demonstrate the relevance of the approach on the formal development of a real-world case study: the multi-purpose interactive application (MPIA) —See Fig. 1, focusing on one of its sub-parts: the weather radar system (WXR). MPIA consists of three pages or tabs: WXR (weather radar system and information), GCAS (Ground Collision Avoidance System) and AIRCOND (setting of AIR CONDitioning). A crew member navigates and switches to a desired page using the corresponding button on the menu bar at the bottom. Each page of the MPIA user interface is made of two distinct parts: an interaction area and the menu bar for selecting one of the three interfaces (bottom of Fig. 1).

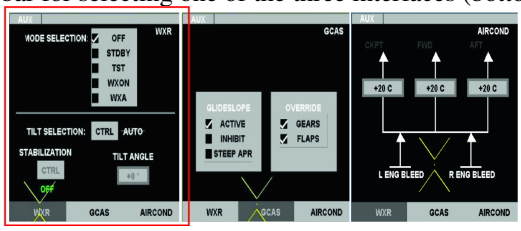

Fig. 1: Tabbed MPIA user interface: WXR, GCAS and AIRCOND

In this paper, we focus on WXR system which is designed to display and modify the mode of the weather radar system (top of the page) and to modify the orientation of the tilt angle in the weather radar system (middle of the page). There are three means for modifying the tilt angle: auto adjustment, auto stabilization, and setting up manually the tilt angle. WXR user interface provides different interactive widgets (PicturePush-

Buttons, RadioButtons, EditBoxNumeric) in order to trigger commands to the weather radar system. The information received from the weather radar (e.g. density of clouds ahead of the aircraft) is not displayed in the WXR page but on another Display Unit (the Navigation Display). The information area displays the current state of the UA, by default the right part is blank but shows errors messages, actions in progress or bad manipulation when necessary. Workspace area controls the corresponding application.

5 Standards formalised as ontologies ((1) on Fig. 2)

Ontologies, as explicit knowledge models [21], have been extensively studied in the literature and applied in several domains spanning semantic web, artificial intelligence, information systems, system engineering etc. Approaches for designing and formalising ontologies for these domains have been proposed. Most of them rely on XML-based formats and pay lot of attention to *web knowledge* which may limit the scope of models.

The challenge of linking domain knowledge and design models is clearly stated in [25]. It includes a mathematical analysis of models and meta models, ontologies, modelling and meta-modelling languages. Design models annotation by domain-specific knowledge has been studied for state-based methods [5] as well. More recently, the textbook [6] reviewed many cases of exploiting explicit models of domain knowledge by system models spanning medical [31,40], e-voting [18], distributed systems etc.

Last, focusing on Event-B, a proposal of simplified ontology description language was put forward and illustrated on case studies in [23,24].

While [5,23,24] and our approach share the same objective and motivation, the two approaches are different. In [5,23,24], Event-B `contexts` are used to formalise domain knowledge in terms of axioms and theorems. However, our approach relies on the theory extension of Event-B providing operators endowed with WD conditions and data types for defining the objects of the knowledge domain. Moreover, they use set-theoretic operators when our approach advocates the exclusive usage of domain-specific operators provided by the theory bearing standard properties together with their WD conditions that need to be discharged when applied in the design model. In addition, the use of data types allowed us to encode an ontology modelling language as an Event-B theory providing a *unified ontological framework* to formalise the various domain knowledge modules. Consequently, WD POs permits a formalisation and integration of domain constraints into design models automatically when used by design models features

In this paper, we rely on engineering domain ontologies in the view of [38,29,4] to model domain knowledge as Event-B theories and on typing to annotate Event-B design models. While [5] use set-theory based contexts where designers explicitly borrow domain standards constraints in the design model, the approach we develop here avoids the developer having to explicitly describe these constraints for each design model.

In the spirit of the OWL [7] ontology modeling language, Listing 1 represents an extract of the `OntologiesTheory` generic Event-B theory parameterised by `C`, `P` and `I` type parameters for classes, properties and Instances, respectively.

```
THEORY OntologiesTheory
TYPE PARAMETERS C,P,I
DATA TYPES Ontology(C,P,I)
CONSTRUCTORS
```

```
consOntology(classes:ℙ(C),properties:ℙ(P),instances:ℙ(I),classProperties:ℙ(C×P),
classInstances:ℙ(C×I),classAssociations:ℙ(C×P×C),instancePropertyValues:ℙ(I×P×I))
OPERATORS
isWDgetInstancePropertyValues <predicate> (o: Ontology(C, P, I))
  well−definedness isWDClassProperites(o) ∧ isWDClassInstances(o) ∧
      isWDClassAssociations(o)
  direct definition
   instancePropertyValues(o) ⊆ { i1 ↦ p ↦ i2 | i1 ∈ I ∧ p ∈ P ∧ i2∈I∧i1↦p↦i2∈
      instances(o)×properties(o)×instances(o) ∧ ( ∃c1, c2 · c1 ∈ C ∧c2∈C∧...)}
getInstancePropertyValues <expression> (o: Ontology(C, P, I))
  well−definedness isWDgetInstancePropertyValues(o)
  direct definition instancePropertyValues(o)
isWDOntology <predicate> (o: Ontology(C, P, I))
  direct definition isWDClassProperties(o) ∧ isWDClassInstances(o) ∧
                    isWDClassAssociations(o) ∧ isWDInstancesAssociations(o)
CheckOfSubsetOntologyInstances <predicate> (o:Ontology(C,P,I),ipvs:ℙ(I×P×I))
      well−definedness isWDOntology(o)
      direct definition
        ipvs ⊆ { i1 ↦ p ↦ i2 | i1 ∈ I ∧ p ∈ P ∧ i2 ∈ I ∧ i1 ↦ p ↦ i2 ∈ instances
            (o) × properties(o) × instances(o) ∧ ...}
isA <predicate> (o: Ontology(C, P, I),c1: C,c2: C) ···
...
THEOREMS
thm1: ∀o,c1,c2,c3·o∈Ontology(C,P,I)∧isWDOntology(o)∧c1 ∈C∧c2∈C∧c3∈C∧
   ontologyContainsClasses(o,{c1,c2,c3})⇒(isA(o,c1,c2)∧isA(o,c2,c3)⇒isA(o,c1,c3))
END
```

Listing 1: Ontology Modelling Language

This theory describes a constructor `consOntology` for ontologies with a set of classes (`classes`), properties (`properties`), instances (`instances`) and associations of properties to classes (`classProperties`), instances to classes (`classInstances`) and classes to classes (`classAssociations`) and property values (`instancePropertyValues`). Expression and predicate operators allowing to manipulate classes, properties and instances are also defined. Predicate operators are used to define WD conditions. For example, the `getInstancePropertyValues` operator retrieving all the properties values is defined under the WD `isWDGetInstancePropertyValues`. The two important operators `isWDOntology` and `CheckOfSubsetOntologyInstances` respectively check that an ontology is well built and a subset of instances is conform to a given ontology. Last, theorems are formalised and proved, e.g. `thm1` for transitivity of `IsA` relationship.

6 Our approach

First, standards are formalised as ontology Event-B theories and, second, these theories should provide data types bundled with a collection of operators to be used by Event-B system models. Note that conformance is achieved under the *closure* condition stating that *solely the operators supplied by the theory formalising a standard are used for state variables changes in design models* . The operators WD POs shall be proved. Obviously, all the theorems entailed by every theory operator also hold for all models that use theory operators. So, conformance-by-construction is guaranteed since 1) models type and manipulate state variables using standard data types and operators and 2) theory safety properties and rules formalising a standard are conveyed by all operators.

Conformance is achieved following the three-step methodology depicted in Fig. 2 *Conceptualisation*, *Instantiation*, and *Annotation*. First, standard concepts and operators are formalised in theories **(2)** using `OntologiesTheory` (see Listing 1) **(1)**. Second, the-

ories are instantiated for a particular system to design **(3)**, and last system model is annotated with data types and operators **(4)** to enforce the constraints and rules, expressed as theorems, establishing standard conformance. Note that `OntologiesTheory` **(1)** is formalised once and for all, while standard concepts, rules and properties **(2)** are formalised in stable theories evolving with standard updates. In Fig 2, `Instantiates` and `Imports` links correspond to Event-B built-in constructs (generic type parameters instantiation is automatically achieved by type synthesis), and `Annotation` is implemented by typing model concepts with theories data-types using the `Sees` Event-B construct.

Note.
A key requirement to set up our approach is the exclusive use of data types and operators provided by the Event-B theory formalising the standard specification. In fact, this condition is necessary to ensure that theorems entailed by operators are transferred and then provable in the Event-B model.

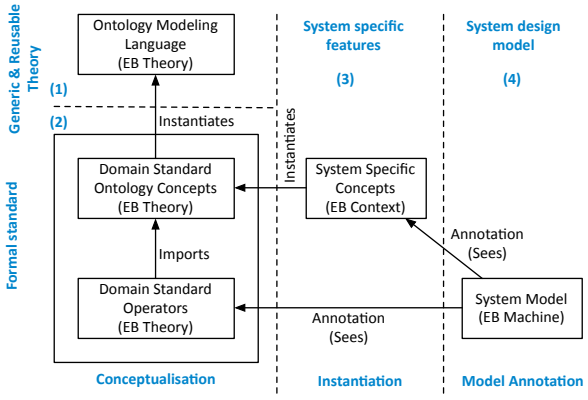

Fig. 2: Standard conformance-by-construction framework

Last, all the developments and Event-B models discussed in this paper are accessible at `https://www.irit.fr/~Ismail.Mendil/recherches/`

6.1 Domain standards as ontology-based theories ((2) on Fig. 2)

The first phase consists in formalising the standard as an ontology using `OntologiesTheory` (see Listing 1). Type parameters `C`, `P` and `I` are instantiated with the standard objects and properties. Furthermore, rules and conformance criteria (i.e. WD condition predicate `isWDOntology`) are formalised as a set of axioms. In a design model, operators allow the modification of the system state variables. A set of theorems, stating that all the defined operators entail standard desired requirements and properties, is also expressed and proved. When these operators are applied in models, these theorems are used to prove model invariants and thus safety properties.

6.2 Standard theory instantiation ((3) on Fig. 2)

At this level, the classes are filled with instances and the associations between instances are specified taking into account the WD conditions required by ontology instantiation, i.e. `isWDgetInstancePropertyValues`. Three components of the ontology are valued by theory instantiation: `instances`, `classInstances` and `instance PropertyValues`. The definitions of these components are *system-dependent* and represent the elements of the system as instances of the standard classes. The `CheckOfSubsetOntologyInstances` operator ensures that system-specific concepts comply with defined standard ontology.

6.3 Model annotation for conformance ((4) on Fig. 2)

Model annotation consists in typing model variable with instance-related ontology components, generally `instancePropertyValues`, to comply with data types originated from the formalised standard. When state changes are done by theory operators, its already proven theorems are transferred to models.

In Event-B, this means that the formalised standard requirements and safety properties expressed as theorems are discharged by deduction as POs of the model. However, this assertion necessitates that the system-specific model state changes to *be realised, exclusively, with the operators provided by the theory describing the domain standard* . Obviously, since the operators are conditional, their WD POs need to be discharged.

7 Standard conformance-by-construction: the case of ARINC 661

In this section, we showcase the approach of section 6 on a part of ARINC 661 and WXR user interface. `ARINC661Theory` is built upon the ontology description theory, which in turn is used to develop the `WXRTheory` theory. Last, the two theories are used to model the WXR user interface as an Event-B machine. Due to space limitation, only an extract of the models covering relevant elements of the WXR case study is presented.

7.1 ARINC 661 standard formalisation ((2) on Fig. 2)

ARINC 661 element	Reference (page)	Event-B formal element
Label	3.3.20 (p114)	Label
RadioBox	3.3.34 (p184)	RadioBox
CheckButton	3.3.5 (p80)	CheckButton
SELECTED, UNSELECTED	3.3.5-1 (p81)	SELECTED, UNSELECTED
CheckButtonState	3.3.5-1 (p81)	hasCheckButtonState
LabelString	3.3.5-1(p81)	hasLabelStringForCheckButton
Textual paragraph	3.3.34 (p185)	isWDRadioBox
...	...	...

Table 3: Correspondence between Event-B formalisation and ARINC 661 standard

ARINCARINC 661 Concepts. After an in-depth analysis of the ARINC 661, many concepts are identified and formalised using `OntologiesTheory`. Table 3 shows some identified correspondences between ARINC 661 concepts and their formal counterparts.

ARINC 661 defines a collection of widgets intended to define the user interfaces. `ARINC661Theory` is described in Listing 2. The formalisation follows the structure of the ARINC 661 widget library and is guided by the ontology description theory. `C`, `P` and `I` of `OntologiesTheory` are instantiated by three abstract types: `ARINC661Classes`, `ARINC661Properties` and `ARINC661Instances`. Constants are defined as well.

```
THEORY ARINC661Theory
IMPORT THEORY PROJECTS OntologiesTheory
AXIOMATIC DEFINITIONS ARINC661Axiomatisation:
TYPES ARINC661Classes, ARINC66Properties, ARINC661Instances
OPERATORS
ARINC661_BOOL <expression> () : ARINC661Classes
A661_TRUE <expression> () : ARINC661Instances
A661_FALSE <expression> () : ARINC661Instances
A661_EDIT_BOX_NUMERIC_ADMISSIBLE_VALUES<expression>():ℙ(ARINC661Instances)
CheckButtonState <expression> () : ARINC661Classes
Label <expression> () : ARINC661Classes
RadioBox <expression> () : ARINC661Classes
```

```
CheckButton <expression> () : ARINC661Classes
hasChildrenForRadioBox <expression> () : ARINC66Properties
hasCheckButtonState <expression> () : ARINC66Properties
SELECTED <expression> () : ARINC661Instances
UNSELECTED <expression> () : ARINC661Instances
isWDRadioBox <predicate> (o: Ontology(ARINC661Classes, ARINC66Properties,
    ARINC661Instances)) :
  well-definedness isWDOntology(o)
isWDARINC661Ontology <predicate> (o: Ontology(ARINC661Classes, ARINC66Properties,
    ARINC661Instances)) :
```

Listing 2: ARINC 661 theory concept declarations

ARINC 661 theory operators. Axiomatic definitions introduce ontology operators and predicates defining WD conditions. In Listing 4, `consARINC661Ontology` operator completes the construction of the ontology, this operator returns a well-defined ontology provided correct arguments are used. Moreover, `CkeckOfSubsetA661Ontology Instances` enforces ontology rules on machine variables if supplied with a well-defined ontology, e.g. `isWDRadioBox` operator encodes a key safety property. It states that *only one child widget can be selected in a given RadioBox at a time* [4].

```
consARINC661Ontology <expression> (ii: ℙ(ARINC661Instances), cii: ℙ(ARINC661Classes×
ARINC661Instances), ipvs:ℙ(ARINC661Instances×ARINC66Properties×ARINC661Instances)):
            Ontology(ARINC661Classes, ARINC66Properties, ARINC661Instances)
  well-definedness isWDARINC661Ontology(consOntology(ARINC661Classes,
      ARINC66Properties, ii, wellBuiltClassProperties, wellbuiltTypesElements ∪
      cii, wellBuiltClassAssociations, ipvs))
CkeckOfSubsetA661OntologyInstances <predicate> (o: Ontology(ARINC661Classes,
    ARINC66Properties, ARINC661Instances), ui: ℙ(ARINC661Instances ×
    ARINC66Properties × ARINC661Instances)) :
  well-definedness isWDOntology(o)
...
```

Listing 3: ARINC 661 theory operator declarations

ARINC 661 axioms. In Listing 4, `ARINC661ClassesDef` axiom defines all the elements of `ARINC661Classes`. For exemple, `Label` is a widget and `CheckButtonState` corresponds to `SELECTED` and `UNSELECTED` states. Similarly, identified ARINC 661 properties are defined in `ARINC66PropertiesDef` axiom.

```
AXIOMS
ARINC661ClassesDef: partition(ARINC661Classes, {Label},{RadioBox},{CheckButton},{
    CheckButtonState}, ...)
ARINC66PropertiesDef: partition(ARINC66Properties, {hasLabelStringForLabel},
{hasChildrenForRadioBox},{hasCheckButtonState},{hasLabelStringForCheckButton},...)
ARINC661InstancesDef: partition(ARINC661Instances,{A661_TRUE},{A661_FALSE},{SELECTED},
{UNSELECTED},LabelInstances, RadioBoxInstances, CheckButtonInstances, ...)
consARINC661OntologyDef: ∀ii, cii, ipvs · ii ∈ ℙ(ARINC661Instances) ∧
      cii ∈ ℙ(ARINC661Classes × ARINC661Instances) ∧
      ipvs ∈ ℙ(ARINC661Instances × ARINC66Properties × ARINC661Instances) ∧
      wellbuiltTypesElements ∩ cii = ∅ ∧ ii ⊆ WidgetsInstances ⇒
          consARINC661Ontology(ii,cii,ipvs)=consOntology(...)
isWDRadioBoxDef:∀o· o ∈ Ontology(ARINC661Classes, ARINC66Properties,
    ARINC661Instances) ⇒ ( isWDRadioBox(o) ⇔ ( ∀ ... )
isWDARINC661OntologyDef:
  ∀o· o ∈ Ontology(ARINC661Classes, ARINC66Properties, ARINC661Instances) ⇒
  (isWDOntology(o)∧isWDRadioBox(o)∧isWDEditBoxNumeric(o)⇒isWDARINC661Ontology(o))
CheckOfSubsetA661OntologyInstancesDef:∀o,ipvs·o∈Ontology(ARINC661Classes,ARINC66Properties,
ARINC661Instances)∧ipvs∈ℙ(ARINC661Instances×ARINC66Properties×ARINC661Instances)⇒
(isWDARINC661Ontology(consOntology(...))⇒CkeckOfSubsetA661OntologyInstances(...))
  ...
```

Listing 4: ARINC 661 theory definitions

[4] More details are available in Section 3.3.34 page 184 of ARINC 661 standard [8].

ARINC 661 relevant theorems. The correctness of the ontology is ensured by theorems `thm1` and `thm2`. They describe two important properties: classes are related to already defined properties (`thm1`) and class associations relate provided classes and properties (`thm2`). Their proofs are achieved using intermediate abbreviations and proved lemmas.

```
THEOREMS
thm1: ∀ii , cii , ipvs ·
   ii ∈ ℙ(ARINC661Instances) ∧ cii ∈ ℙ(ARINC661Classes × ARINC661Instances) ∧
   ipvs ∈ ℙ(ARINC661Instances × ARINC66Properties × ARINC661Instances) ∧
   wellbuiltTypesElements ∩ cii = ∅ ∧ ii ⊆ WidgetsInstances
            ⇒ isWDClassProperites(consARINC661Ontology(ii , cii , ipvs))
thm2: ∀ii , cii , ipvs ·
   ii ∈ ℙ(ARINC661Instances) ∧ cii ∈ ℙ(ARINC661Classes × ARINC661Instances) ∧
   ipvs ∈ ℙ(ARINC661Instances × ARINC66Properties × ARINC661Instances) ∧
   wellbuiltTypesElements ∩ cii = ∅ ∧ ii ⊆ WidgetsInstances
            ⇒ isWDClassAssociations(consARINC661Ontology(ii , cii , ipvs))
...
END
```

Listing 5: ARINC 661 theory theorems

Ontology building process. The ontology introduced above formalises the concepts of the ARINC 661 standard. This ontology (theory) has been built for the purpose of the FORMEDICIS project and to process the different addressed case studies. The selection of axioms and the formalisation and proofs of theorems have been performed according to the studied case study. In case of a wide and shared usage, as with any standard, the designed theory requires consensus among the stakeholders of the ARINC 661 standard.

7.2 System-specific concepts describing WXR widgets ((3) on Fig. 2)

WXRTheory concepts declaration. `WXRTheory` encompasses constants and operators dealing with instance information (not defined in `ARINC661Theory` as instances are system specific) and allowing to manipulate the user interface. `WXRFeature` gathers the instances used by the WXR design model.

```
THEORY WXRTheory
IMPORT THEORY PROJECTS ARINC661Theory
AXIOMATIC DEFINITIONS WXRUIDescriptoinAxiomatisaiton:
OPERATORS
A661WXROntology<expression>:Ontology(ARINC661Classes , ARINC66Properties ,
    ARINC661Instances)
WXRInstances <expression> : ℙ(ARINC661Instances)
WXRClassInstances <expression> : ℙ(ARINC661Classes × ARINC661Instances)
WXRInstancePropertyValues<expression>:ℙ(ARINC661Instances×ARINC66Properties×
    ARINC66Instances)
MODESELECTIONLabel <expression> : ARINC661Instances
OFFLabel <expression> : ARINC661Instances
OFFCheckButton <expression> : ARINC661Instances
...
WXRFeatures<expression>(o:Ontology(ARINC661Classes ,... , ARINC661Instances)) :
            ℙ(ARINC661Instances × ARINC66Properties × ARINC661Instances)
well−definedness isWDARINC661Ontology(o)
```

Listing 6: WXR theory constant declarations

WXR concepts definitions. In Listing 7, ARINC 661 ontological class instances are used for defining constants of the type ℙ (`ARINC661`). For example, `WXRinstances` is a set of all possible widgets of user interface: `WXRLabels`, `WXRCheckButtons`, etc. The `WXRFeatures` operator restricts ARINC661 ontology to the instances needed to design the WXR user interface i.e. *none of these instances is outside ARINC 661 theory.*

```
AXIOMS
WXRLabelsDef: partition(WXRLabels, {MODESELECTIONLabel}, {OFFLabel}, ... )
WXRcheckButtonsDef: partition(WXRcheckButtons, {OFFCheckButton}, ...)
WXRradioBoxesDef: partition(WXRradioBoxes, {WXRradioBoxModeSelection},...)
WXRInstancesDef: partition(WXRInstances,WXRLabels,WXRcheckButtons,WXRradioBoxes, ...)
WXRClassInstancesDef: WXRClassInstances = ({Label} × WXRLabels) ∪ ({CheckButton} ×
    WXRcheckButtons) ∪ ...
iaCheckBttonsDef: iaCheckBttons=({OFFCheckButton,...}×{hasVisible,hasEnable}×{A661_TRUE
    })∪({OFFCheckButton, ...} × {hasCheckButtonState} × {UNSELECTED}) ∪
        ({OFFCheckButton} × {hasCheckButtonState} × {SELECTED}) ∪
({OFFCheckButton,...}×{hasParentIdent}×{WXRradioBoxModeSelectionWidgetIdent})∪ ...
WXRInstancePropertyValuesDef: WXRInstancePropertyValues=iaCheckBttons∪ioRadioBoxes∪ ...
A661WXROntologyDef: A661WXROntology = consARINC661Ontology(Instances, ClassInstances
    , WXRInstancePropertyValues)
WXRFeaturesDef: ∀o · o ∈ Ontology(ARINC661Classes,ARINC66Properties,ARINC661Instances
    ) ∧ isWDARINC661Ontology(o) ⇒ WXRFeatures(o) = WXRInstancePropertyValues
```

Listing 7: WXR theory constant definitions

WXRTheory Operators. The user interface provides user interactions operators: choosing a mode selection, switching between the two states of the stabilization and tilt section feature and finally input a new tilt angle value. Each interaction is modelled by two operators: a WD predicate and an interactions modelling operators. For example, `isWDChangeModeSelection` and `changeModeSelection` pair of operators deals with mode selection change (see Listing 8) .

```
AXIOMATIC DEFINITIONS EventsAffectingWidgetsAxiomatisation:
OPERATORS
isWDChangeModeSelection <predicate> (o: Ontology(ARINC661Classes, ARINC66Properties,
    ARINC661Instances) ,ui: ℙ(ARINC661Instances × ARINC66Properties ×
    ARINC661Instances),mode: ARINC661Instances) :
changeModeSelection <expression> (o: Ontology(ARINC661Classes, ARINC66Properties,
    ARINC661Instances) ,ui: ℙ(ARINC661Instances × ARINC66Properties ×
    ARINC661Instances),mode: ARINC661Instances) : ℙ(ARINC661Instances ×
    ARINC66Properties × ARINC661Instances)
well−definedness isWDChangeModeSelection(o, ui, mode)
```

Listing 8: WXR theory operator declarations

In the `AXIOMS` clause, several operators are defined (see Listing 9). For example, `changeModeSelection` operator is associated to a WD operator `isWDChangeMode Selection` stating that *crew members may select only specified modes in* `WXRcheck Buttons` and `CkeckOfSubsetA661OntologyInstances` ensures that the `ui` parameter complies with ontology rules and constraints. This principle applies to all operators.

```
AXIOMS
isWDChangeModeSelectionDef:∀o, ui, mode · o ∈ Ontology(ARINC661Classes,
    ARINC66Properties, ARINC661Instances) ∧ ui ∈ ℙ(ARINC661Instances ×
    ARINC66Properties × ARINC661Instances) ∧ mode ∈ ARINC661Instances ⇒
    (isWDChangeModeSelection(o, ui, mode) ⇔ CkeckOfSubsetA661OntologyInstances(o,
        ui) ∧ mode ∈ WXRcheckButtons)
changeModeSelectionDef:∀o, ui, mode · o ∈ Ontology(ARINC661Classes, ARINC66Properties,
    ARINC661Instances) ∧ ui ∈ ℙ(ARINC661Instances × ARINC66Properties ×
    ARINC661Instances) ∧ mode ∈ ARINC661Instances ⇒
(changeModeSelection(o,ui,mode)=(ui\{i↦hasCheckButtonState ↦ UNSELECTED | i ↦
    hasCheckButtonState ↦ SELECTED ∈ ui ∧
i∈(WXRcheckButtons\{mode}) })∪{mode↦hasCheckButtonState↦SELECTED})
...
```

Listing 9: WXR theory operator definitions

WXRTheory theorems. In `WXRTheory`, important safety properties (e.g. theorem `WXR FeaturesSafety`) assert that the selection of the buttons under radio boxes are exclu-

sive ($\Rightarrow b1 = b2$) (Listing 10). All theorems have been proved on the Rodin Platform.

```
THEOREMS
isWDARINC661Ontology: isWDARINC661Ontology(A661WXROntology)
WXRFeaturesSafety:∀o,ipvs·CkeckOfSubsetA661OntologyInstances(o,ipvs)∧(ipvs =
      WXRFeatures(o)) ⇒ (∀rb, b1, b2··· ⇒ b1 = b2) ) ∧ ...
WXRFeaturesCkeckOfSubsetA661OntologyInstances: ∀o,ipvs·isWDARINC661Ontology(o)
∧ipvs∈ℙ(ARINC661Instances × ARINC661Properties × ARINC661Instances) ∧
(ipvs = WXRFeatures(o)) ⇒
      changeModeSelectionCkeckOfSubsetA661OntologyInstances(o, ipvs)
changeModeSelectionSafety: ...
changeModeSelectionCkeckOfSubsetA661OntologyInstances: ...
...
END
```

Listing 10: WXR theory theorems

7.3 Annotated Event-B model of WXR application ((4) on Fig. 2)

The WXR user interface is modelled as an Event-B machine and uses elements defined in `WXRTheory`. In Listing 11, the state of the user interface is modelled by `uiStateVar` variable. The event `changeMode Selection` models the interaction on the mode selection radio box where only one check box shall be selected. The safety properties are entailed by theorems, `WXRFeaturesCkeckOfSubsetA661OntologyInstancesInst` and `SafetyInst`, establishing at the same time the conformance of WXR specification to ARINC 661. However, the approach requires to use the theory operator to update the variable as `uiStateVar` as prescribed by `inv2`.

Fig. 3: WXR annotated with Event-B concepts

Listing 11 shows an extract of WXR model. In particular, `changeModeSelection Evt` event uses `changeModeSelection` operator to select a mode from the mode selection radio box, like *STDBY* (see. Fig. 3). Note that this event is guarded with WD conditions of `WXRTheory`. In Fig. 3, correspondences between WXR widgets and their standard formal counterparts are depicted.

```
MACHINE  WXRModel
VARIABLES  uiStateVar
INVARIANTS
inv1: uiStateVar ∈ ℙ(ARINC661Instances × ARINC66Properties ×
                     ARINC661Instances)
inv2: ∃uiArg · ((uiStateVar = WXRFeatures(A661WXROntology)) ∨
          ∃m · isWDChangeModeSelection(A661WXROntology, uiArg, m) ∧
                    uiStateVar = changeModeSelection(A661WXROntology, uiArg, m)) ∨
          ...
SafetyInst: CkeckOfSubsetA661OntologyInstancesDef(A661WXROntology, uiStateVar)
WXRFeaturesCkeckOfSubsetA661OntologyInstancesInst:
    (∀rb, b1, b2· rb ∈ RadioBoxInstances ∧ b1 ∈ CheckButtonInstances ∧
       b2 ∈ CheckButtonInstances ∧ rb ↦ hasChildrenForRadioBox ↦ b1 ∈ uiStateVar∧
       rb ↦ hasChildrenForRadioBox ↦ b2 ∈ uiStateVar⇒
```

```
                (b1↦hasCheckButtonState↦SELECTED∈ui∧
                 b2↦hasCheckButtonState↦SELECTED∈ uiStateVar ⇒b1=b2)) ∧ ...)
EVENTS
   INITIALISATION
   THEN
      act1: uiStateVar := WXRFeatures(A661WXROntology)
   END
   changeModeSelectionEvt
   ANY   mode
   WHERE
      grd1: mode ∈ WXRcheckButtons
      grd2: isWDChangeModeSelection(A661WXROntology, uiStateVar, mode)
   THEN
      act1: uiStateVar := changeModeSelection(A661WXROntology, uiStateVar, mode)
   END
   ...
END
```

Listing 11: Event-B machine modelling the WXR user interface

8 Assessment

Achieving standard conformance. Provided that the domain knowledge is formalised as a theory and supplied with data types and operators that preserve the safety properties prescribed by the standard specification, the models can be proven to entail desired theorems achieving conformance with the formalised standard.

Enhanced system models. WXR model has been greatly improved as a result of extensive outsourcing of safety properties to the theory level and the use of ontology description theory. The use of a theory validated by experts led to trustworthy models. In addition, this approach enabled domain-specific (standards) models to be validated, once and for all, independently of the systems design models.

Reduction of modelling and proving effort. Although the description of the domain-specific theory, `ARINC661Theory`, requires a significant amount of modelling effort, the specification of the models is simplified as a result of the formalisation of interaction by theory operators. At theory level, the properties (theorems) are proved once and for all. The design models rely on the defined data types and operators *conveying* all desired WD and safety properties expressing the domain constraints encoded in the theory of the standard. Here, the proving process is eased as, on the one hand, the `WD` POs are discharged thanks to WD predicates associated with each operator and, on the other hand, `INV` POs are discharged automatically. Indeed, `inv1` is a typing invariant and `inv2` states that no other operator, except those provided by the theory, is used. Table 4 shows 88 automatically generated POs for the theories and WXRmodel. Theories related POs are discharged using a mix of automatic and interactive proofs, whereas WXRMode POs are discharged by simplifying predicates, instantiating theorems and using proof tactics. System invariants are proved as theorems in one proof step (modus-ponens rule), and the invariants representing our working hypothesis (exclusive use of theory operators) are trivially proved as model events use the operators of `WXRTheory` exclusively.

Deploying the approach in engineering contexts. The work presented in this paper has been conducted in the FORMEDICIS project. As mentioned in section 7.1, the ARINC 661 standard has been formalised following our understanding of the informal descriptions of [8]. However, as the obtained theories play the role of a standard, we believe that this formalisation requires consensus among the stakeholders, engineers

and developers. From the development process point of view, this formalisation and the proofs of theorems are achieved once and for all. When, design models are produced, the conformance consists in discharging POs consisting in instantiating the theorems and using proof tactics. Therefore, we believe that the deployment of the approach, in its current form, is not a heavy task compared to the benefits of the provided proofs.

Standard theories validation. The formalisation of standards relies on axiomatised theories. The quality of these formalisations consist in checking 1) the consistency of the axioms and 2) the validation of these axioms and entailed theorems with respect to the informal descriptions. Fortunately, formal methods such as Event-B, Isabelle/HOL or CoQ come with tools like SMT solvers, animators and model checkers capable to instantiate such axioms with specific values and check axiom consistency or testing instances validity.

Event-B Models and Theories	Proof obligations
OntologiesTheories	21
ARINC661Theory	10
WXRTheory	39
WXRModel	18

Table 4: Proof statistics

Enabling evolution of Standard Last but not least, the approach enables the non-destructive standards evolution. Indeed, the neat separation of the common domain knowledge from system specifics fosters separation of concerns principle and orthogonality of evolution principle. In fact, both domain models and system design models may evolve asynchronously with limited impact on the each other. From a proof perspective, only POs caused by the evolution need to discharged.

9 Conclusion

The approach presented in this paper proposes a generic framework for formalising standard conformance through formal modelling of standards as ontologies. Data types and operators associated to the modelled features become accessible to system design models. We have shown how this approach applies to a real-world case study of aircraft cockpit. This approach is completely formalised using Event-B and relies on three steps: conceptualisation of the domain standard, instantiation to describe the system specific features and finally model annotation through typing of state variables and use of operators for state changes. The approach starts from an already formalised standard. It does not address the process of deriving these theories from text-based standards. It exploits the WD conditions POs that raise when applying theory operators.

The work presented in this paper addressed the issue of standard conformance. It needs to be extended to provide the required safety assurances to meet certification standards, where assurance cases are used in the development of critical systems. The formally proved properties and the generated formal artifacts can be used as evidence in assurance cases, which can aid in the certification process by guiding both the development and regulatory evaluation of CIS. Last, from the standardisation point of view, industry consortium and standardisation bodies shall define formal processes (not studied in this paper) addressing consensual agreement on the definition and consistence of the formal theories modelling domain standards i.e. the process consisting in analysing text-based standards in order to derive domain standard theories and in validating these derived theories. In addition, this work shall be completed by the study of other type of domain standards related to temporal properties, real-time scheduling, common criteria for security etc. and application domains like avionics, transportation systems.

References

1. Abrial, J.R.: The B-book: assigning programs to meanings. Cambridge Univ. Press (1996)
2. Abrial, J.R.: Modeling in Event-B: system and software engineering. Cambridge University Press (2010)
3. Abrial, J.R., Butler, M., Hallerstede, S., Leuschel, M., Schmalz, M., Voisin, L.: Proposals for mathematical extensions for event-b. Tech. Rep. (2009)
4. Aït Ameur, Y., Baron, M., Bellatreche, L., Jean, S., Sardet, E.: Ontologies in engineering: the ontodb/ontoql platform. Soft Comput. **21**(2), 369–389 (2017)
5. Aït Ameur, Y., Méry, D.: Making explicit domain knowledge in formal system development. Science of Computer Programming, Elsevier Journal. **121**, 100–127 (2016)
6. Aït Ameur, Y., Nakajima, S., Méry, D.: Implicit and Explicit Semantics Integration in Proof-Based Developments of Discrete Systems. Springer (2021)
7. Antoniou, G., van Harmelen, F.: Web Ontology Language: OWL, pp. 67–92. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
8. ARINC: ARINC 661 specification: Cockpit Display System Interfaces to User Systems, Prepared by AEEC, Published by SAE, Melford Blvd., Bowie, Maryland, USA (June 2019)
9. Bartolini, C., Giurgiu, A., Lenzini, G., Robaldo, L.: A framework to reason about the legal compliance of security standards. In: 10th Int. Workshop on Juris-informatics (2016)
10. Bjørner, D.: Manifest domains: analysis and description. Formal Aspects Comput. **29**(2), 175–225 (2017)
11. Bjørner, D.: Domain analysis and description principles, techniques, and modelling languages. ACM Trans. Softw. Eng. Methodol. **28**(2), 8:1–8:67 (2019)
12. Brucker, A.D., Aït-Sadoune, I., Crisafulli, P., Wolff, B.: Using the isabelle ontology framework - linking the formal with the informal. In: Rabe, F., Farmer, W.M., Passmore, G.O., Youssef, A. (eds.) Intelligent Computer Mathematics - 11th International Conference, CICM 2018. LNCS, vol. 11006, pp. 23–38. Springer (2018)
13. Brucker, A.D., Wolff, B.: Isabelle/dof: Design and implementation. In: Ölveczky, P.C., Salaün, G. (eds.) Software Engineering and Formal Methods - SEFM 2019. LNCS, vol. 11724, pp. 275–292. Springer (2019)
14. Brucker, A.D., Wolff, B.: Using ontologies in formal developments targeting certification. In: Ahrendt, W., Tarifa, S.L.T. (eds.) Integrated Formal Methods - 15th International Conference, IFM 2019. LNCS, vol. 11918, pp. 65–82. Springer (2019)
15. Butler, M.J., Maamria, I.: Practical theory extension in Event-B. In: Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He 70th Birthday. pp. 67–81 (2013)
16. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Introduction to Conformance Checking, pp. 3–20. Springer International Publishing, Cham (2018)
17. Emmerich, W., Finkelstein, A., Montangero, C., Stevens, R.: Standards compliant software development. In: in Proc. International Conference on Software Engineering Workshop on Living with Inconsistency, (IEEE CS. pp. 1–8. Press (1997)
18. Gibson, J.P., Raffy, J.L.: Modelling an E-Voting Domain for the Formal Development of a Software Product Line: When the Implicit Should Be Made Explicit, pp. 3–18. Springer Singapore (2021)
19. Goodenough, J., Weinstock, C., Klein, A.: Toward a theory of assurance case confidence. Tech. Rep. CMU/SEI-2012-TR-002, Software Engineering Institute, CMU, Pittsburgh (2012)
20. Grigorova, S., Maibaum, T.S.E.: Argument evaluation in the context of assurance case confidence modeling. In: 25th IEEE ISSRE Workshops. pp. 485–490. IEEE CS (2014)
21. Gruber, T.R.: Towards Principles for the Design of Ontologies Used for knowledge sharing. In: Guarino, N., Poli, R. (eds.) Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publisher's (1993)

22. Guiochet, J., Hoang, Q.A.D., Kaâniche, M.: A model for safety case confidence assessment. In: Koornneef, F., van Gulijk, C. (eds.) Computer Safety, Reliability, and Security - 34th International Conference SAFECOMP. LNCS, vol. 9337, pp. 313–327. Springer (2015)
23. Hacid, K., Aït Ameur, Y.: Strengthening MDE and formal design models by references to domain ontologies. A model annotation based approach. In: Margaria, T., Steffen, B. (eds.) 7th International Symposium, ISoLA. LNCS, vol. 9952, pp. 340–357 (2016)
24. Hacid, K., Aït Ameur, Y.: Handling domain knowledge in design and analysis of engineering models. Electron. Commun. Eur. Assoc. Softw. Sci. Technol. **74** (2017)
25. Henderson-Sellers, B.: On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages. Springer Briefs in Computer Science, Springer (2012)
26. IEC 62304: Medical Device Software – Software Life Cycle Processes (May 2006)
27. ISO: Industrial automation systems and integration - parts library - part 42: Description methodology: Methodology for structuring parts families. ISO ISO13584-42, International Organization for Standardization, Geneva, Switzerland (1998)
28. Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 1: General concepts (1991)
29. Jean, S., Pierra, G., Aït-Ameur, Y.: Domain Ontologies: A Database-Oriented Analysis. In: International Conferences, WEBIST. pp. 238–254. LNBI Processing, Springer (2007)
30. Kelly, T.: Arguing Safety – A Systematic Approach to Managing Safety Cases. Ph.D. thesis, University of York (September 1998)
31. Kumar, S.N., Yamine, A.A., Dominique", M.: Formal Ontological Analysis for Medical Protocols, pp. 83–107. Springer Singapore (2021)
32. van Lamsweerde, A.: Requirements Engineering - From System Goals to UML Models to Software Specifications. Wiley (2009)
33. Leuschel, M., Butler, M.: Prob: A model checker for b. In: International symposium of formal methods europe. pp. 855–874. Springer (2003)
34. Luong, H.V., Lambolais, T., Courbis, A.L.: Implementation of the conformance relation for incremental development of behavioural models. In: Czarnecki, K., Ober, I., Bruel, J.M., Uhl, A., Völter, M. (eds.) MoDELS'08. pp. 356–370. Springer (2008)
35. Nair, S., de la Vara, J.L., Sabetzadeh, M., Falessi, D.: Evidence management for compliance of critical systems with safety standards: A survey on the state of practice. Information and Software Technology **60**, 1–15 (2015)
36. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: A Proof Assistant for Higher-order Logic. Springer-Verlag (2002)
37. Owre, S., Rushby, J.M., Shankar, N.: Pvs: A prototype verification system. In: Kapur, D. (ed.) Automated Deduction—CADE-11. pp. 748–752. Springer Berlin Heidelberg (1992)
38. Pierra, G.: Context representation in domain ontologies and its use for semantic integration of data. Journal on Data Semantics **10**, 174–211 (2008)
39. Rushby, J.: The interpretation and evaluation of assurance cases. Tech. Rep. SRI-CSL-15-01, Computer Science Laboratory, SRI International, Menlo Park, CA (Jul 2015)
40. Singh, N.K., Aït Ameur, Y., Méry, D.: Formal ontology driven model refactoring. In: 23rd International ICECCS. pp. 136–145. IEEE CS (2018)
41. Tueno Fotso, S.J., Frappier, M., Laleau, R., Mammar, A.: Modeling the hybrid ertms/etcs level 3 standard using a formal requirements engineering approach. In: Butler, M., Raschke, A., Hoang, T.S., Reichl, K. (eds.) ABZ'18. pp. 262–276. Springer, Cham (2018)
42. Wassyng, A., Joannou, P., Lawford, M., Maibaum, T.S.E., Singh, N.K.: New standards for trustworthy cyber-physical systems. In: Romanovsky, A., Ishikawa, F. (eds.) Trustworthy Cyber-Physical Systems Engineering, pp. 337–368. Taylor & Francis Group (2016)
43. Wassyng, A., Singh, N.K., Geven, M., Proscia, N., Wang, H., Lawford, M., Maibaum, T.: Can product-specific assurance case templates be used as medical device standards? IEEE Des. Test **32**(5), 45–55 (2015)