Leveraging Event-B Theories for handling domain knowledge in design models

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. Formal system modelling languages lack explicit constructs to model domain knowledge, hindering clear separation of this knowledge from system design models. Indeed, in many cases, this knowledge is hardcoded in the system formal specification or is simply overlooked. Providing explicit domain knowledge constructs and properties would yield a significant improvement in the robustness and confidence of the system design models. Therefore, it speeds up formal verification of safety properties and advances system certification since certification standards and requirements rely on domain knowledge models. The purpose of this paper is to show how formal system design models can benefit from explicit handling of domain knowledge, represented as ontologies. To this end, state-based Event-B modelling language and theories are used to model system models and domain knowledge ontologies, respectively. Our proposition is exemplified by the TCAS (Traffic Collision Avoidance System) system, a critical airborne avionic component. Finally, we provide an assessment highlighting the overall approach.

Keywords: Domain knowledge · Ontologies · System engineering · State-based formal methods · Safety proofs · Invariant preservation · Event-B

1 Introduction

Context. Due to the high level of confidence required, critical systems are subjected to a variety of validation and verification (V&V) activities. Engineering these critical systems requires the use of numerous engineering techniques and methods, standards and certification processes, etc. Formal methods have proved their capability to handle complex verification and validation techniques set up at different development phases. They are grounded on mathematical and logic theories and are equipped with a proof system used to check formalised properties. They advocate the design of a formal model specifying the desired system behaviours and the description of a set of properties expressing requirements, usually safety and security requirements. The underlying proof system allows

to check the properties on the formalised system model. Various formal methods are available and several tools have been developed to support both system modelling and property verification using automatic verification procedures like model checking techniques (e.g. Promela/SPIN, NuSMV, Uppaal) or SMT and SAT solvers (e.g. Z3, CVC4), interactive theorem proving based on higher order logic or type theory (e.g. Isabelle/HOL [36], Coq [11]).

If we consider a system specification S and a set of property requirements R, then property verification consists in proving that requirements can be proved from the specification by establishing $S \vdash R$.

In this case, the designed formal model associated to a specification S must make explicit all the knowledge required to write a specification, in particular the domain knowledge provided by the domain and the context where the system is supposed to evolve i.e. S encapsulates the whole formal system description needed to establish R. The seminal work of [47] and [12,13] suggests to separate so called *Domain Knowledge* from the specification and proposes the well-known triptych $K, S \vdash R$ where K formalises domain knowledge. This separation is motivated by two main arguments: first domain knowledge is usually stable and reusable and second its formalisation is made explicit through K.

Motivation. In general, system engineering approaches, particularly formal methods, do not offer *explicit* constructs allowing the designer to define formal models of domain knowledge, nor mechanisms to import such existing models. However, there exist formal modelling languages and/or meta-models, sometimes standardised [19], that support the formalisation of such domain knowledge. It is often the case that transformations are required to reuse, in the set up formal method, already defined knowledge domain formalisations. As a result, heterogeneous formalisations are obtained, so sharing and reuse are compromised.

Our claim. We believe that ontologies, seen as *an explicit shared specification of a conceptualisation* [26] suit with the requirement of domain knowledge *sharing* and *reuse*. We advocate that domain knowledge must be formalised as an ontology formally modelled as datatypes theories with axioms, theorems and reasoning capabilities, *once and for all*, in the used system development formal method. Moreover, we claim that this formalisation shall not impact system modelling languages nor system models. Last, to avoid semantic heterogeneity, both ontologies, system specifications and requirements shall be formalised in a single *shared* mathematical setting, Event-B [1] with set theory and First order logic in our case.

Objective of this paper. We propose to use the Event-B [1] proof and refinement formal method to express both domain knowledge as ontologies formalised using Event-B theories, and system specification and requirements formalised as Event-B models (machines and invariants). As ontologies constructs are not present as first order concepts in Event-B, we introduce an Event-B meta-theory, based on generic and abstract datatypes and operators, describing an ontology

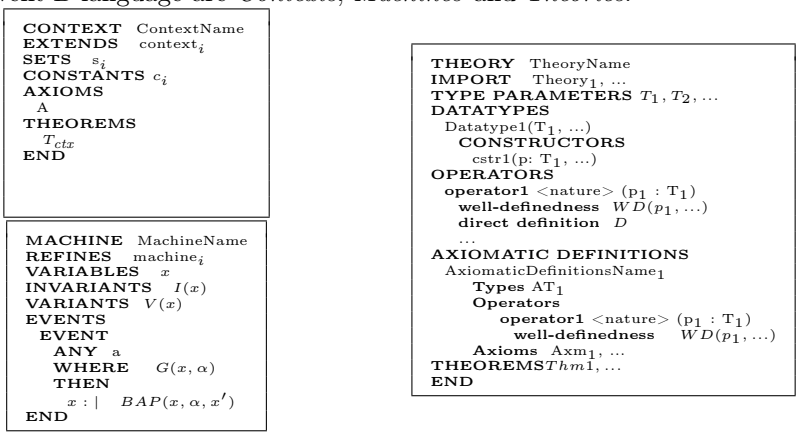
model formalising an ontology modelling language (e.g. OWL [6]), further instantiated to derive specific domain ontologies. These ontologies become *shareable*, *reusable* and *referencable* by any Event-B model using typing, operators and properties guaranteed by proving both ontology instantiated theorems and Well-Definedness (WD) Proof Obligations (POs).

Structure of this paper. Next section presents the Event-B state-based method and theories used to formalise our models. Section 3 is devoted to an overview of the state of the art in handling domain knowledge in system models. Our approach is synthesised in section 4. Then, Section 5 presents a generic theory encoding OWL-like ontologies. In Section 6, we address the problem of domain knowledge handling in system design through the case of Aircraft Critical Interactive Systems design. Section 7 provides an assessment of our approach and Section 8 concludes the paper.

2 Event-B: a Refinement and Proof-Based Formal Method

2.1 Core Event-B

First-order logic (FOL) and set theory underpin the Event-B [1] modelling language. The design process consists of a series of refinements[1] of an abstract model (specification) leading to a final concrete model. The core modeling components of the Event-B language are *Contexts*, *Machines* and *Theories*.



Lst. 1.1: Basic Event-B building blocks: context, machine and theory

Event-B Contexts. Contexts (see Listing 1.1) define the static part of a model. They introduce definitions, axioms and theorems describing the required concepts and their properties. *Carrier sets s* defining algebraically new types (pos-

[1] As refinement is not necessary to understand our contribution, it has been skipped. More details can be found in [1]

sibly constrained in axioms or other extending contexts), *constants* c, *axioms* A and *theorems* T_{ctx} are introduced.

Event-B Machines. A *machine* (see Listing 1.1) describes the dynamic part of a model as a transition system. A set of possibly parameterised and/or guarded events (transitions) modifying a set of state variables (state) represents the core concepts of a machine. *Variables* x, *invariants* $I(x)$, *theorems* $T_{mch}(x)$, *variants* $V(x)$ and *events evt* (possibly guarded by G and/or parameterised by α) are defined in a machine. *Invariants* and *theorems* formalise system safety while *variants* define convergence properties (reachability).
Before-After Predicates (BAP) express state variables changes using prime notation x' to record the new value of a variable x after a change. The *"becomes such that"* :| substitution is used to define the next (transition or event) value of a state variable. We write $x :| BAP(x, x')$ to express that the next value of x (denoted by x') satisfies the predicate $BAP(x, x')$ defined on before and after values of variable x. When a parameter α is involved in a variable the BAP is expressed as $x :| BAP(\alpha, x, x')$.

Proof Obligations (PO) and Property Verification. To establish the correctness of an Event-B model (machine) the POs (automatically generated from the calculus of substitutions) need to be proved.

PO designation	PO formal definition
(1) Ctx Theorems (THM)	$A \Rightarrow T_{ctx}$ `(For contexts)`
(2) Mch Theorems (THM)	$A \wedge I(x) \Rightarrow T_{mch}(x)$ `(For machines)`
(3) Initialisation (INIT)	$A \wedge G(\alpha) \wedge BAP(\alpha, x') \Rightarrow I(x')$
(4) Invariant preservation (INV)	$A \wedge I(x) \wedge G(x, \alpha) \wedge BAP(x, \alpha, x') \Rightarrow I(x')$

Table 1: Relevant Proof Obligations

The main POs, relevant for this paper, are listed in Table 1. They require to demonstrate that both contexts (1) and machines (2) theorems hold, initialisation (3) and each event preserves invariants (induction (4)).

Core Well-definedness (WD). WD POs are associated to all Event-B built-in operators of the Event-B modelling language. For example a WD proof obligation for the division operator $WD(a \div b)$ is $WD(a) \wedge WD(b) \wedge b \neq 0$ and for conjunction $WD(P \wedge Q)$ is $WD(P) \wedge (P \implies WD(Q))$. These proof obligations are defined for every operator of the core Event-B language.

2.2 Event-B Extensions with Theories

To handle additional abstract concepts beyond set theory and first order logic, an Event-B extension supports externally defined mathematical objects, modelled as *theories* [2,18]. Close to other proof assistants (e.g. Isabelle/HOL [36], PVS [37]), this capability is convenient when modelling, using data types, concepts not available in core Event-B.

Theories for abstract data types. They define types (possibly inductive) carried by sets and operators. They introduce axioms and theorems, and use the Event-B sequent calculus as proof system to prove theorems. The concepts defined in theories can be imported by Event-B models and used in further developments.

Well-definedness (WD) in Theories. An important feature provided by Event-B theories relates to the definition of *well-definedness* conditions. Each defined operator may be associated to a condition guaranteeing its correct definition. When the operator is used (either in the theory or in an Event-B machine or context), this well-definedness condition generates a proof obligation requiring to establish that this condition holds, i.e. the use of the operator is correct. The theory developer defines these WD conditions, which are then added to the native Event-B WD POs. We extensively use this feature for defining domain knowledge operators.

Theory description (see Listing 1.1). Theories define and make available new data types, operators and theorems. Data types (`DATATYPES` clause) are associated with *constructors* i.e operators to build inhabitant of the defined type. They may be inductive. A theory may define various *operators* further used in Event-B expressions. They may be *predicates* built using classical first order logic or *expressions* producing actual values (<nature> tag). Operators applications (predicates or expressions) can be used in other Event-B theories, contexts and/or machines. They *enrich the modelling language* as they may occur in the definitions of axioms, theorems, invariants, guards, assignments, etc.
As mentioned above, an operator may be associated with *WD conditions* encoding specific requirements. It defines, as a PO, the condition under which the operator is used. Each time the operator is used, this PO must be proved.
Operators may be defined explicitly using an explicit ("direct") equivalent definition, in the `direct definition` clause, (e.g., in the case of a constructive definition), or defined axiomatically in the `AXIOMATIC DEFINITIONS` clause, (e.g. a set of axioms). At the definition level, operators application mode is characterised: infix or prefix, or if they are commutative and/or associative. Last, a theory defines a set of axioms, completing the definitions, and theorems. Theorems are proved from the definitions and axioms.
We mention that many theories have been defined for sequences, lists, groups, reals, differential equations, etc.

Associated proof system. As mentioned in [18], soundness of theories is achieved through the definition of soundness proof obligations generated following the standard approach of Event-B and their proofs are carried out using the sequent calculus of Event-B encoded in the Rodin provers. They rely on a set-theoretic formalisation of operators. More details can be found in [2,18]. Theories are also 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 (behaving as hypotheses in proof sequents) using their axiomatic definition. Theorems may be imported as

hypotheses and, like other theorems, they may be used in the proof as any other one. They are accessible by the interactive and automatic provers, and SMT solvers of Rodin.

Event-B and its IDE Rodin. Rodin[2] is an open source, Eclipse-based Integrated Development Environment 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, developed for the Rodin platform. Many provers like predicate provers, SMT solvers, are plugged to Rodin.

Application of the B method. Event-B method has been successfully applied to design critical systems for applications, like control system for the Meteor line 14 in Paris or the VAL shuttle for Paris CDG airport [9], medical devices [40], autonomous systems [42], security protocols [10], control-command systems [23] and distributed protocols [34,48]. More information can be found in [17,39].

3 Related Works

The contribution presented in this article is at the intersection of three scientific themes studied by the scientific community for decades.

Ontologies and Domain Modelling. Ontologies, as explicit knowledge models [26], have been extensively studied in the literature and applied in several domains spanning semantic web, artificial intelligence, information systems, system engineering etc. Several approaches for describing, designing and formalising ontologies for these application domains have been proposed. Models, browsers like Protégé or PlibEditor, repositories like JENA-SDB, TripleStore, OntoDB or OntoHub query languages like RQL, SPARQL or OntoQL, reasoners like Pellet, RACER or KAON, annotators like CREAM, Terminae or SAWSDL and translators have been proposed to engineer ontologies. Many ontologies have been described for several engineering domains and annotation mechanisms were proposed to establish links to domain objects like texts, images, videos and engineering models. Most of mentioned approaches rely on XML-based formats and are applied to the semantic web. As they are grounded on descriptive logics, they pay lot of attention to the *decidability criterion* for automated reasoning and inference purposes (which may limit the scope of addressed knowledge models and logics). To the best of our knowledge, *formal annotation of formal design models* and their *analysis* have not been set up using the above mentioned approaches or tools.

[2] `http://www.event-b.org/index.html`

Explicit domain models in formal methods. Formalisation of domain knowledge witnessed high interest in the formal methods community where many approaches and frameworks were proposed in Coq [11], Isabelle/HOL [36] with ISADof [16,15] Framework, PVS [37], Event-B with theories [2,18] (e.g. control theory for control-command systems [24,22]) and critical systems [4], DOL - Distributed Ontology Model and Specification Language based on CASL algebraic specification [35] integrated to the OntoHub ontology repository and RSL - RAISE Specification Language [14] for transportation, shipping, and logistic systems. In [21], the authors present a two-layered language ground on higher-order logic of Coq formal system as a lower layer, and ontology language as upper layer for expressing and specifying contexts. Indeed, the higher-order KDTL language [8,20] supports the definition of new contextual categories and facts on the basis of low-order context. The language provides means to support comparability of diverse and non-countable information as well as numeric data. Although the approaches cited above tackle the problem of formalising domain knowledge and provide modular frameworks, they differ from the modelling level where they apply. Two kinds of modelling levels incorporating domain knowledge have been identified. On the one hand, the modelling level that offers domain-specific language constructs, in particular constructs for ontologies allowing the explicit definition of ontology components like classes, properties and instances. In general, such approaches adopt a deep modelling style by explicitly encoding both syntax and semantics of some ontology description language like OWL. On the other hand, the second modelling language level adopts a shallow modelling style. It encodes domain knowledge concepts directly in the hosting formal modelling language using its syntactic and semantic constructs. Knowledge domain concepts are not made explicit in the obtained models.

Domain Knowledge in system design. In [30], the authors clearly state the challenge of linking domain knowledge and design models. A mathematical analysis of models and meta models, ontologies, modelling and meta-modelling languages is included. Design models annotation by domain-specific knowledge has been studied for state-based methods in [4] as well. More recently, the textbook [5] reviewed many cases of exploiting explicit models of domain knowledge by system models spanning medical systems, e-voting systems, distributed systems etc. Indeed, [41] presents a four steps modelling approach based on the methodology described in [4]. It relies on Event-B contexts as domain knowledge models for ontologies for verifying medical protocols. An assessment of the proposed approach is given through a complex case study of a real-life reference protocol (electrocardiogram (ECG) interpretation) covering a wide variety of protocol characteristics related to different heart conditions. [31] showcases how Event-B theories may be used to capture domain-specific abstract data types (ADTs) and build dynamic systems using the developed structures. The authors adopt an incremental approach to model domain knowledge concepts in parallel with the refinement of the model. The approach uses Event-B theories so it benefits from the use of the operators endowed with well-definedness conditions. The case studies used to illustrate the approach also define inference rules for easing the

proving process. The work of [46] describes a meta model for a domain modeling language built from OWL and PLIB which is part of the SysML/KAOS requirements engineering method that includes a goal modeling language. The formal semantics of SysML/KAOS models is specified, verified, and validated using the Event-B method. Goal models provide machines and events of the Event-B specification while domain models provide its structural part (sets and constants with their properties and variables with their invariant). The proposal is exemplified through a case study dealing with a localization component for an autonomous vehicle Last, focusing on Event-B, a proposal of simplified ontology description language was put forward and illustrated on case studies in [28,29]. The approach was based on context extension where the design models need to discharge proof obligation in form of theorems to validate the compliance of the formal design models to the formalised domain knowledge.
The approaches mentioned above illustrate interesting solutions to the formalisation and incorporation of domain knowledge in formal design models. However, several limitations are identified. They constitute part of the challenges we address in our proposal. First, some of these approaches lack a general framework such as ontologies for defining domain knowledge in standardised way. They require to formalise the domain properties and their proofs in Event-B contexts. They do not offer explicit mechanisms enforcing domain knowledge constraints on the design models. The designer has to handle these constraints while formalising systems models.
Last, they strongly focus on full automation for reasoning and inference and target decidable fragments of logic like descriptive logics which do not enable the expression of all properties encountered in engineering domain (e.g. expression of arithmetic properties).
The main purpose of this paper is to define a general proof-based framework addressing the limitations identified above. It relies on engineering ontologies in the view of [3] to model domain knowledge as Event-B theories —a collection of data types and operators with well-definedness conditions — and use typing to annotate system design models formalised in Event-B.

4 Domain Knowledge in State-Based Formal Methods: the Case of Event-B

Formal methods are equipped with constructs allowing to support formal system developments and reasoning. In general these methods separate the system specification from the properties expressing requirements. They do not offer built-in constructs to axiomatise domain knowledge.
In our approach, we propose *to use the capability of formal methods to describe, import and (re-)use theories in the system design models.*
In the sequel, we use Event-B theories to model domain ontologies as a collection of data types, constructors and operators defined by specific axioms. Each operator is accompanied with WD (Well-Definedness) properties defining the conditions for correct use of each operator. When an operator is used (i.e.

applied), a WD proof obligation, corresponding to this condition, needs to be proved (discharged). Indeed, once the theory formalising an ontology is designed, models import and use its data types and operators and the corresponding WD proof obligations require to be discharged. As a consequence, domain knowledge model is factorised once and for all in a single *reusable* and *shareable* theory and second it does not require, from the designer, to write domain knowledge invariants and properties required to guarantee correctness of the design models. Indeed, the use of operators brings, *for free*, WD conditions as proof obligations to be discharged. Each design model is annotated by domain knowledge through typing. Event-B theories offer services and capabilities to implement the notion of design models conforming to domain knowledge constraints expressed by theories. These theories provide data types and operators for expressiveness while requiring discharging of WD conditions ensuring conformance checking.
At this level, a *formal setting to write theories for domain knowledge modelling is missing. A domain knowledge modelling language shall be used for this purpose.*
Next sections describe an Event-B based development process allowing to handle domain knowledge as formal ontologies and annotate formal models using typing. A generic theory for ontologies is presented and a case study issued from aircraft cockpits engineering showcases the overall approach.

5 Ontologies as Event-B Theories

From section 3, we conclude that ontologies, as descriptive knowledge models for domains, are powerful models for knowledge representation and reasoning, and from section 4 we also conclude that Event-B, and more generally state-based formal methods, are suitable for enforcing design models to reference domain knowledge concepts and express their constraints as well-definedness conditions. To define our ontologies, we rely on defined ontology modelling languages (OML) like OWL [6] or PLIB [38]. In our case, provided that it can be described using data-types based on set theory and first order logic, whatever is the ontology modelling language, it can be described by Event-B theories.
Our approach proposes a formal parameterised theory, acting as a meta-theory associated to the OML and each ontology is described as a theory instance of this meta-theory. More precisely, the ontologies we use are based on a theory inspired from OWL[3] where domain knowledge is formalised as collections of classes, properties and instances..
Listing 1.2 shows an extract of `OntologiesTheory` theory allowing the formalisation of OWL-based ontologies. It is parameterised by `C`, `P`, and `I` which stand for classes, properties and instances. The `Ontology(C,P,I)` data type is built using the `consOntology` constructor based on seven components: `classes`, `properties`, `instances` (i.e. set of classes, properties and instances respectively), `classProperties` for associating classes to properties, `classInstances` for relating instances to classes, `classAssociations` defining a set of property-named binary associations and `instanceAssociations` for representing the as-

[3] https://www.w3.org/TR/owl-features/

sociations between instances. Besides the ontology structure, operators are defined to manipulate, access and update an ontology.
In Listing 1.2, the `getInstanceAssociations`, `instanceHasPropertyValuei`, `addValueOfAnInstanceProperty` and `removeValueOfAnInstanceProperty` operators define access to properties of a class and instances of an association, check if a property is valued and add/remove a property value. The `isA` relationship encoding class subsumption is defined and the theorem `isATransitivityThm`, stating its transitivity, is proven.

```
THEORY OntologiesTheory
TYPE PARAMETERS
  C, P, I
DATATYPES
  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),
                                  instanceAssociations: ℙ(I × P × I) )
OPERATORS
  isWDClassProperites <predicate> (o : Ontology(C, P, I))
          . . .
  getClassProperties <expression> (o : Ontology(C, P, I)
          . . .
  isWDInstancesAssociations <predicate> (o : Ontology(C, P, I))
          . . .
  getInstanceAssociations <expression> (o : Ontology(C, P, I)

  isWDOntology <predicate> (o : Ontology(C, P, I))
    direct definition
      isWDClassInstances(o) ∧ isWDClassProperites(o) ∧
      isWDClassAssociations(o) ∧ isWDInstancesAssociations(o)
. . .
  isWDinstanceHasPropertyValuei <predicate>
          . . .
  instanceHasPropertyValuei <predicate>
   (o : Ontology(C, P, I), ipv: ℙ I × P × I), i: I, p: P, v: I)
    well-definedness
      isWDinstanceHasPropertyValuei(o, ipv, i, p)
    direct definition
      v ∈ ipv[{i ↦ p}]

  getInstancesOfaClass <expression> (o : Ontology(C, P, I), c: C)
    well-definedness
      isWDOntology(o) ∧ ontologyContainsClasses(o, {c})
    direct definition
      getClassInstances(o)[{c}]
  addValueOfAnInstanceProperty <expression>
          . . .
  removeValueOfAnInstanceProperty <expression>
          . . .
  isA <predicate>
   (o : Ontology(C, P, I), c1: C, c2: C)
    well-definedness
      isWDOntology(o)
      ontologyContainsClasses(o, {c1, c2})
    direct definition
      getInstancesOfaClass(o, c1) ⊆ getInstancesOfaClass(o, c2)
THEOREMS
  isATransitivityThm: ∀o, c1, c2, c3 · o ∈ Ontology(C, P, I)∧
    c1 ∈ C ∧ c2 ∈ C ∧ c3 ∈ C∧
          ontologyContainsClasses(o, {c1, c2, c3})
          ⇒ (isA(o, c1, c2) ∧ isA(o, c2, c3) ⇒ isA(o, c1, c3))
```

Lst. 1.2: Excerpt of ontologies theory OML

Thanks to its type system, Event-B theories support the description of other operators e.g. arithmetic or defined-types operators. These operators are associated to WD conditions (logical expressions) to ensure correct use and to preserve a valid ontology structure at instantiation. When an operator is applied, generated

WD proof obligations need to be proved. Hence, depending on the chosen OML, Event-B theories permit the modelling of complex domain knowledge.

Important Note. The choice of the OML is driven by the needs and complexity of the domain knowledge of interest: system engineering. It requires other modelling capabilities like property derivation using arithmetic expressions or context dependent properties and associated proof rules (see [3] [4] for more details). To handle engineering knowledge, we use first-order logic with arithmetic in our OML. This richer expressive power leads to semi-automatic proofs requiring interactive proof effort[4].

6 Application to the Design of Critical Interactive Systems

We highlight the importance of system design models annotation relying on explicit formalised domain knowledge via the development of a critical interactive system (CIS): TCAS - Traffic Collision Avoidance System.
In this section, We describe the development of a critical interactive system (CIS): the TCAS - It is critical to the safe flight of any aircraft, namely Traffic Collision Avoidance System. We show the importance of formalising domain information and its integration to the system design model.
The formal development of this case study relies on domain knowledge formalised as an instantiated theory (`Displayability Theory`) of the ontology model (see Listing 1.2). Then, the definition of the knowledge related to the specific case of *aircraft* objects *displayability* is obtained by instantiating of the latter ontology using an Event-B context (see Listing 1.4), where the seven components of the ontology are defined and used to build the `aircraftOntology` ontology.

6.1 The TCAS Case Study

TCAS is an airborne avionics system that acts as a last resort safety net to mitigate risks of midair collisions. TCAS tracks aircraft in the surrounding airspace exploiting position sent by their transponders to detect collision risks. If an impedent collision is detected, TCAS issues a Resolution Advisory (RA) to the flight crews of concerned aircrafts. These advisories ask them to climb or descend at a given vertical rate to prevent collision [44,25].

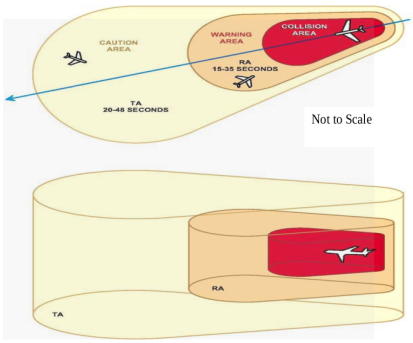


Fig. 1: Protection Volume

[4] *Automatic* reasoners (decidable logics) like Pellet [43] or Racer [27] apply to *less rich* OML than the one offered by Event-B theories.

TCAS computes a virtual protected volume (Fig. 1) which includes the position of the aircrafts nearby. This volume depends on the aircraft speed and trajectory. It is permanently updated. Some of the information related to volume is displayed in a cockpit screen for flight crew usage. An example of such display can be found in [45]. Due to space constraints, we only focus on a single critical safety property: TCAS must display, on a PFD (Primary Flight Display) cockpit screen, the current status of all the aircrafts in the volume. Beyond, a *critical* aircraft (due to its proximity) must be *visible*.

6.2 A Domain Ontology for the Critical Interactive Systems

Two steps are required to build the domain ontology.

THEORY DisplayabilityTheory
IMPORT OntologiesTheory
AXIOMATIC DEFINITIONS
$IOOntology$
TYPES
$IOClasses$, $IOProperties$, $IOInstances$
OPERATORS
isIOOntologyWD $< predicate >$
$(o : Ontology(IOClasses, IOProperties, IOInstances))$
visible $< expression >: IOInstances$
hidden $< expression >: IOInstances$
critical $< expression >: IOInstances$
safe $< expression >: IOInstances$
hasVisibility $< expression >: IOProperties$
hasCriticality $< expression >: IOProperties$
visibility $< expression >: IOClases$
criticality $< expression >: IOClases$
isVisibleWDi $< predicate > \ldots$
isVisiblei $< predicate >$
$(o : Ontology(IOClasses, IOProperties, IOInstances),$
$ipvs : \mathbb{P}(IOInstances \times IOProperties \times IOInstances),$
$i : IOInstances)$
well-definedness
isVisibleWDi(o, ipvs, i)
setCriticaliWD $< predicate > \ldots$
setCriticali $< expression >$
$(o : Ontology(IOClasses, IOProperties, IOInstances),$
$ipv : \mathbb{P}(IOInstances \times IOProperties \times IOInstances),$
$i : IOInstances)$
well-definedness
setCriticaliWD(o, ipvs, i)
AXIOMS
axm1 : $\forall o \cdot o \in \ldots \Rightarrow (isIOOntologyWD(o) \Leftrightarrow isWDOntology(o)$
axm2 : $partition(IOProperties, \{hasVisibility\}, \{hasCriticality\})$
axm3 : $\{visibility, criticality\} \subseteq IOClasses$
axm4 : $\forall o, ipv, i \cdot o \in \ldots \Rightarrow (isVisibleWDi(o, ipv, i) \Leftrightarrow$
$i \in dom(dom(ipv)))$
axm5 : $\forall o, ipv, io \in \ldots \Rightarrow (isVisiblei(o, ipv, i) \Leftrightarrow$
$instanceHasPropertyValuei(o, ipv, i, hasVisibility, visible)$
...
axm18 : $\forall o, ipv, i \cdot o \in \ldots \Rightarrow (setCriticaliWD(o, ipv, i) \Leftrightarrow$
$i \in dom(dom(ipv)) \wedge isVisiblei(o, ipv, i))$
axm19 : $\forall$ **o, ipv1, ipv2, i** $\cdot$ **o** $\in$
Ontology(**IOClasses**, **IOProperties**, **IOInstances**)$\wedge$
ipv1 $\in \mathbb{P}$(**IOInstances** $\times$ **IOProperties** $\times$ **IOInstances**)$\wedge$
ipv2 $\in \mathbb{P}$(**IOInstances** $\times$ **IOProperties** $\times$ **IOInstances**)$\wedge$
i $\in$ **IOInstances** $\Rightarrow$ (**ipv2** = **setCriticali**(**o**, **ipv1**, **i**) $\Leftrightarrow$
ipv2 = (**ipv1** $\setminus$ {**i** $\mapsto$ **hasCriticality** $\mapsto$ **safe**})$\cup$
{**i** $\mapsto$ **hasCriticality** $\mapsto$ **critical**})

Lst. 1.3: Exerpt of Displayability theory

An Ontology of Interactive Objects First, we define a generic domain knowledge model for interactive objects (IOs) by instantiating the ontology theory (see Listing 1.2) to get the `DisplayabilityTheory` Event-B theory (IO ontology - Listing 1.3). It axiomatises a collection of specific operators with WD conditions entailing *displayability* properties of critical IOs. Indeed, `IOClasses`, `IOProperties`, `IOInstances` types and two kinds of operators (predicates and expressions) are defined. Predicates check if a property holds in the system variable or introduce WD conditions required for other operators. Besides, we create constant operators like `visible`, `hidden`, `critical`, `safe` which are instances of `IOInstances` and `hasVisibility`, `hasCriticality` being elements of `IOProperties`.
The `instanceHasPropertyValuei` operator of `OntologiesTheory` is a predicate with five arguments: *ontology*, *system variable*, *instance*, *property* and *value*. The predicate is true when the 3-tuple $instance \mapsto property \mapsto value$ is in *system variable*. For example, the operator `isVisiblei` uses it to state that an IO is visible if and only if its property `hasVisibility` relates the IO to *visible* and complies with the ontology schema (see `instanceHasPropertyValuei` in Listing 1.2). We adopted the same methodology for writing all operators.
For instance, we define the WD condition for `setCriticali` as a predicate `setCriticaliWD` encapsulating the conditions needed to use this operator: $i \in dom(dom(ipv))$ stating that i must be in the model variable and `isVisiblei(o,ipvs,i)` meaning that the IO i must be visible. Last, for domain coverage purposes, the domain knowledge model (theory) is self-contained, i.e. defined concepts and properties are manipulated using theory operators only (no other IO manipulation is allowed). Thus, proved theorems hold for all IOs.
Remark. From system engineering perspective, this assumption means that a designer shall **only** use the types and operators supplied by the theory encoding the domain knowledge ontology.

Instantiation for Aircraft Description IO (Listing 1.4) `Displayability Theory` is instantiated in the context `OntologyInstantiation Context` to define the specific IOs concepts and properties used in the TCAS models. For our development, three classes are introduced: `aircraftClass` (is a thing by `isAthm11`)), `visibility` and `criticality`. The latter two classes are borrowed from the `DisplayabilityTheory` theory. In addition, we introduce the `aircraft Instances` using an extensional axiom `axm2` (Event-B `partition` operator asserts that the first argument is the disjoint union of the others). Afterwards, `aircraftOntology` is built in `axm9`. Last, note that `ConformThm9` theorem is proved from the constituent of the ontology to ensure that this ontology is WD by `isWDIOontology` (Listings 1.3 and 1.2).

6.3 Ontology-Based Annotation of TCAS Design Model

The Event-B machine model `TheoryOperatorsBasedModel` (Listing 1.5) handles the *safety* requirement stating that *a critical aircraft must be visible* thanks to the

CONTEXT InstantiationContext
CONSTANTS
aircraftClass, aircraftInstances, ClassProperties, ClassInstances
ClassAssociations, instanceAssociation, **aircraftOntology**,
thingClass, thingInstances,
AXIOMS
axm1 : $partition(IOClasses, \{thingClass\}, \{aircraftClass\}, \{visibility\}, \{criticality\})$
axm2 : $partition(IOInstances, \{aircraftInstances\}, \{visible\}, \{hidden\}, \{safe\}, \{critical\})$
axm3 : $thingInstances = IOInstances$
axm4 : $ClassProperties = \{aircraftClass\} \times \{hasVisibility, hasCriticality\}$
axm5 : $ClassInstances = (\{aircraftClass\} \times aircraftInstances) \cup (\{visibility\} \times \{visible, hidden\}) \cup (\{criticality\} \times \{critical, safe\}) \cup (thingClass \times thingInstances)$
axm6: $ClassAssociations \in \mathbb{P}(IOClasses \times IOProperties \times IOClasses)$
axm7 : $ClassAssociations = (\{aircraftClass\} \times \{hasVisibility\} \times \{visibility\}) \cup (\{aircraftClass\} \times \{hasCriticality\} \times \{criticality\})$
axm8 : $instanceAssociation = (aircraftInstances \times \{hasVisibility\} \times \{hidden\}) \cup (aircraftInstances \times \{hasCriticality\} \times \{safe\})$
axm9 : $aircraftOntology = consOntology(IOClasses, IOProperties, IOInstances, ClassProperties, ClassInstances, ClassAssociations, instanceAssociation)$
ConformThm10 : **isIOOntologyWD**(**aircraftOntology**)
isAthm11 : $isA(aircraftOntology, aircraftClass, thingClass)$
END

Lst. 1.4: Context of instantiation

annotation of the state variable `system` using `isVariableOfOntology` predicate operator in `inv1` and to the use of `setCriticali` operator, borrowed from the `DisplaybilityOntology`, in the event `CorrectAircraftStatusUpdate`.
Indeed, `isVariableOfOntology` ensures that `system` variable fully complies with `aircraftOntology` rules. From the proof perspective, the guards of the event guarantee correct variable updating and invariant preservation. As a benefit, the two theories exempt the designer from writing domain-related properties, thus focusing only on the system-specific model. Listing 1.5 shows important parts of the TCAS model, noticebly the event `CorrectAircraftStatusUpdate` allows to update the aircraft i status so that it is visible (see its definition of `setCriticali` in Listing 1.3). The guards ensure that the operation is performed in a well-defined fashion through the use of the necessary WD operators.

7 Assessment

The complete Event-B development including all the theories and models may be downloaded from `https://www.irit.fr/~Ismail.Mendil/recherches/`

Previous work. [32] proposed a correct-by-construction Event-B development of TCAS featuring many functionalities. However, domain knowledge formalisation is not explicit and domain-specific rules are hardcoded in the design models. Here, we improved our approach making domain knowledge explicit by annotating models with ontologies, using data types formalised as Event-B theories. Consequently, automatic domain oriented WD proof obligations are generated

```
MACHINE TheoryOperatorsBasedModel
SEES InstantiationContext
VARIABLES system
INVARIANTS
  inv1 : isVariableOfOntology(aircraftOntology, system)
INITIALISATION
  THEN
    act1 : system : |system′ ⊆ instanceAssociation
EVENT CorrectAircraftStatusUpdate
  ANY i
  WHERE
    grd1 : ontologyContainsInstances(aircraftOntology, {i})
    grd2 : isVisibleWDi(aircraftOntology, system, i)
    grd3 : isVisiblei(aircraftOntology, system, i)
    grd4 : isSafeWD(aircraftOntology, system, i)
    grd5 : isSafe(aircraftOntology, system, i)
    grd6 : isWDSetCriticali(aircraftOntology, system, i)
  THEN
   act1 : system := setCriticali(aircraftOntology, system, i)
...
```

Lst. 1.5: Ontology theory based annotated model

and proved in the annotated design models. It is worth noticing that *these theories are built and proved once and for all.*

Explicit Domain Knowledge and Reusability. The proposed ontology modelling language (an Event-B theory) makes it possible to design, systematically, a series of theories, composing and/or extending each other, to model various domain knowledge as instances of the generic theory of Listing 1.2. In addition, domain theories and system models are formalised (integrated) in the single setting of Event-B (Set theory and first order logic) avoiding semantic mismatch that may occur in case of heterogeneous modelling language semantics. Besides, Listing 1.2 generic theory of ontologies supports engineering standards formalisation. Confidence in the consistency of the standard rules is achieved by proving WD and theorems. Last, theorems of the theory are proved once and for all. Like domain knowledge types and operators, these theorems are reused in system models.

Reduction of modelling effort. When models are annotated by references to ontology (Listing 1.5) through typing in Event-B theories, guards are described by WD conditions (`grd3 - grd5`) systematically borrowed from the ontology when an operator is applied. They are mined, in a systematic way, from the well-definedness conditions of the used operator in `act1`. This model provides assistance to the designer as domain knowledge operators applications allow a designer to identify the operator WD conditions for its correct application.

Enhanced safety of system models. We presented an Event-B model (machine) defined for TCAS based on annotation of state variables through typing with domain theory defined types (Listing 1.5). This model based on WD and avoids the designer having to explicitly write invariants. This is a major strength of our approach as it assists the designer by *describing explicitly safety properties in the domain ontology* (Event-B theory of Listings 1.3 and 1.4) as axioms and theorems and by embedding these safety properties in the design model through the WD proof obligation. The designer uses *operators in the models that brings*

their WD proof obligations that ensure safety when discharged (No need to write explicitly the invariants, this work is achieved on the ontology side).

Asynchronous evolution. The neat separation of the general domain knowledge on which the system depends and the specific features of the system under study enforces the separation of concepts principle and promotes formal specification modularisation enabling the orthogonality principle i.e. both domain and system design models may evolve asynchronously with limited impact on previous developments. In case of evolution, solely the proof obligations generated due to this evolution are discharged again.

8 Conclusion and Future Work

The work presented in this paper takes advantage of foundations and methods of knowledge modeling and reasoning on the one hand and formal system engineering on the other hand. This work defines a uniform framework integrating both domain knowledge, system specification and safety requirements in a unique formal modelling setting and proof system offered by Event-B. It advocates the 1) *explicit* modelling of domain knowledge by ontologies as a well-accepted formal modelling framework and the 2) *separation* of domain and system models. The proposition yields three important advantages in formal modelling state-of-the-art. Indeed, it becomes possible to 1) refer to (annotation) domain models concepts (types, operators, etc.), 2) automatically bring, in the system model, checking of well-definedness proof obligations for robustness purposes, and 3) allow asynchronous evolution of both domain and system models thanks to the separation of concerns. However, this evolution does not prevent from checking new occurring proof obligations and/or old ones that may not be preserved.
The overall approach was showcased using a formalised an OWL-based domain modelling language as an Event-B theory where *data types, operators* and *Well-Definedness* play a central role. We used this formal ontology language to describe a domain theory for critical interactive systems (CIS) concepts and safety rules for displaying aircraft in TCAS. Moreover, the system engineering domain was exemplified to shed the light on the gain in robustness when using the *Well-Defined* operators of a domain theory. An assessment is provided to evaluate efficiency of knowledge formalisation and integration in our approach. Finally, the formalized theories developed in this paper were used to annotate design models as part of our approach for standard conformance in [33]. A large part of ARINC 661 [7] standard describing Cockpit Display Systems (CDS) interfaces used in all aircrafts has been formalised. This standard plays important role to minimise costs as well as to meet certification requirements.

Future work. The current study opened a number of new research directions. From the foundational perspective, we intend to formalise knowledge models composition with theory composition operators (importation, extension and instantiation) in order to handle heterogeneity and multi-view problems of complex

systems while maintaining consistency of obtained Event-B theories. Another significant perspective consists in addressing other engineering domains, specifically transportation systems.

References

1. Abrial, J.R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
2. Abrial, J.R., Butler, M., Hallerstede, S., Leuschel, M., Schmalz, M., Voisin, L.: Proposals for mathematical extensions for Event-B. Tech. rep. (2009)
3. 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)
4. Aït Ameur, Y., Méry, D.: Making explicit domain knowledge in formal system development. Science of Computer Programming, Elsevier Journal. **121**, 100–127 (2016)
5. Aït Ameur, Y., Nakajima, S., Méry, D.: Implicit and Explicit Semantics Integration in Proof-Based Developments of Discrete Systems. Springer (2021)
6. Antoniou, G., van Harmelen, F.: Web Ontology Language: OWL, pp. 67–92. Springer Berlin Heidelberg, Berlin, Heidelberg (2004), https://doi.org/10.1007/978-3-540-24750-0_4
7. ARINC: ARINC 661 specification: Cockpit Display System Interfaces To User Systems. By AEEC, Published by SAE, 16701 Melford Blvd., Suite 120, Bowie, Maryland 20715 USA (June 2019)
8. Barlatier, P., Dapoigny, R.: A type-theoretical approach for ontologies: The case of roles. Appl. Ontology **7**, 311–356 (2012)
9. Behm, P., Benoit, P., Faivre, A., Meynadier, J.M.: Météor: A Successful Application of B in a Large Project, pp. 369–387. Springer Berlin (1999)
10. Benaissa, N., Méry, D.: Cryptographic protocols analysis in event b. In: Pnueli, A., Virbitskaite, I., Voronkov, A. (eds.) Perspectives of Systems Informatics. pp. 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
11. Bertot, Y., Castran, P.: Interactive Theorem Proving and Program Development: Coq'Art The Calculus of Inductive Constructions. Springer Publishing Company, Incorporated, 1st edn. (2010)
12. Bjørner, D.: Software Engineering 3 - Domains, Requirements, and Software Design. Texts in Theoretical Computer Science. An EATCS Series, Springer (2006), https://doi.org/10.1007/3-540-33653-2
13. Bjørner, D.: Domain analysis and description principles, techniques, and modelling languages. ACM Trans. Softw. Eng. Methodol. **28**(2), 8:1–8:67 (2019)
14. Bjørner, D.: Domain analysis and description principles, techniques, and modelling languages. ACM Transactions on Software Engineering Methodology **28**(2), 8:1–8:67 (2019)
15. 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)
16. 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)
17. Butler, M.J., Körner, P., Krings, S., Lecomte, T., Leuschel, M., Mejia, L., Voisin, L.: The first twenty-five years of industrial use of the b-method. In: ter Beek, M.H.,

Nickovic, D. (eds.) 25th International Conference, FMICS. LNCS, vol. 12327, pp. 189–209. Springer (2020)
18. Butler, M.J., Maamria, I.: Practical theory extension in Event-B. In: Theories of Prog. and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday. pp. 67–81 (2013)
19. Calegari, D., Mossakowski, T., Szasz, N.: Heterogeneous verification in the context of model driven engineering. Science of Computer Programming, Elsevier Journal. **126**, 3–30 (2016)
20. Dapoigny, R., Barlatier, P.: Modeling ontological structures with type classes in coq. In: ICCS (2013)
21. Dapoigny, R., Barlatier, P.: Formalizing Context for Domain Ontologies in Coq, pp. 437–454 (12 2014)
22. Dupont, G., Aït-Ameur, Y., Pantel, M., Singh, N.K.: Handling refinement of continuous behaviors: A refinement and proof based approach with Event-B. In: 13th International Symposium TASE. pp. 9–16. IEEE Computer Society Press (2019)
23. Dupont, G., Aït-Ameur, Y., Pantel, M., Singh, N.K.: Proof-based approach to hybrid systems development: Dynamic logic and Event-B. In: Butler, M.J., Raschke, A., Hoang, T.S., Reichl, K. (eds.) 6th International Conference, ABZ 2018, Proc. LNCS, vol. 10817, pp. 155–170. Springer (2018)
24. Dupont, G., Aït-Ameur, Y., Pantel, M., Singh, N.K.: Formally Verified Architecture Patterns of Hybrid Systems Using Proof and Refinement with Event-B. In: Rashke, A., Méry, D. (eds.) 7th International Conference, ABZ 2018, Proceedings. LNCS, vol. 12071, pp. 155–170. Springer (2020)
25. EUROCONTROL: Airborne collision avoidance system (acas) guide (December 2017)
26. 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)
27. Haarslev, V., Möller, R.: Description of the racer system and its applications. vol. 2083 (01 2001)
28. 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.) Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA. LNCS, vol. 9952, pp. 340–357 (2016)
29. Hacid, K., Aït Ameur, Y.: Handling Domain Knowledge in Design and Analysis of Engineering Models. Electron. Commun. Eur. Assoc. Softw. Sci. Technol. **74** (2017)
30. Henderson-Sellers, B.: On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages. Springer Briefs in Computer Science, Springer (2012)
31. Hoang, T.S., Voisin, L., Butler, M.: Domain-Specific Developments Using Rodin Theories, pp. 19–37. Springer Singapore, Singapore (2021)
32. Mendil, I., Singh, N.K., Aït Ameur, Y., Méry, D., Palanque, P.A.: An integrated framework for the formal analysis of critical interactive systems. In: 27th Asia-Pacific Software Engineering Conference, APSEC 2020, Singapore, December 1-4, 2020. pp. 139–148. IEEE (2020)
33. Mendil, I., Singh, N.K., Aït Ameur, Y., Méry, D., Palanque, P.A.: Standard Conformance-by-Construction with Event-B. In: Lafuente, A.L., Mavridou, A. (eds.) In Proceedings of 26th FMICS - International Conference, Formal Methods for Industrial Critical Systems (To Appear). LNCS, vol. 12863. Springer (2021)

34. Méry, D., Singh, N.K.: Analysis of DSR Protocol in Event-B. p. 401–415. SSS'11, Springer-Verlag, Berlin, Heidelberg (2011)
35. Mossakowski, T.: The distributed ontology, model and specification language - DOL. In: James, P., Roggenbach, M. (eds.) Recent Trends in Algebraic Development Techniques - 23rd IFIP WG 1.3 International Workshop, WADT, Revised Selected Papers. LNCS, vol. 10644, pp. 5–10. Springer (2016)
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, Berlin, Heidelberg (1992)
38. Pierra, G.: The plib ontology-based approach to data integration. In: Jacquart, R. (ed.) Building the Information Society. pp. 13–18. Springer US, Boston, MA (2004)
39. Romanovsky, A.B., Thomas, M. (eds.): Industrial Deployment of System Engineering Methods. Springer (2013)
40. Singh, N.K.: Using Event-B for Critical Device Software Systems. Springer (2013)
41. Singh, N.K., Ait-Ameur, Y., Méry, D.: Formal Ontological Analysis for Medical Protocols, pp. 83–107. Springer Singapore, Singapore (2021), https://doi.org/10.1007/978-981-15-5054-6_5
42. Singh, N.K., Aït Ameur, Y., Pantel, M., Dieumegard, A., Jenn, E.: Stepwise Formal Modeling and Verification of Self-Adaptive Systems with Event-B. The Automatic Rover Protection Case Study. In: 21st International Conference on Eng. of Complex Computer Systems, ICECCS. pp. 43–52 (2016)
43. Sirin, E., Parsia, B.: Pellet: An OWL DL reasoner. Description Logics pp. 212–213 (2004)
44. ED 143 - Minimum Operational Performance Standards for Traffic Alert and Collision Avoidance System II (TCAS II) (2013)
45. U.S. Department of transportation, F.A.A.: Introduction to TCAS 2, version 7.1 (February 2011)
46. Tueno, S., Laleau, R., Mammar, A., Frappier, M.: Integrating Domain Modeling Within a Formal Requirements Engineering Method, pp. 39–58. Springer Singapore, Singapore (2021)
47. Zave, P., Jackson, M.: Four dark corners of requirements engineering. ACM Transactions on Software Engineering Methodology **6**(1), 1–30 (1997)
48. Zoubeyr, F., Aït Ameur, Y., Ouederni, M., Tari, A.: A correct-by-construction model for asynchronously communicating systems. Int. J. Softw. Tools Technol. Transf. **19**(4), 465–485 (2017)