Using Ontologies in Formal Developments Targeting Certification

Achim D. Brucker[1][0000−0002−6355−1200] and Burkhart Wolff[2]

[1] University of Exeter, Exeter, UK
a.brucker@exeter.ac.uk
[2] Université Paris-Saclay, LRI, Paris, France
wolff@lri.fr

Abstract. A common problem in the certification of highly safety or security critical systems is the consistency of the certification documentation in general and, in particular, the linking between semi-formal and formal content of the certification documentation.
We address this problem by using an existing framework, Isabelle/DOF, that allows writing certification documents with consistency guarantees, in both, the semi-formal and formal parts. Isabelle/DOF supports the *modeling* of document ontologies using a strongly typed ontology definition language. An ontology is then *enforced* inside documents including formal parts, e. g., system models, verification proofs, code, tests and validations of corner-cases. The entire set of documents is checked within Isabelle/HOL, which includes the definition of ontologies and the editing of integrated documents based on them. This process is supported by an IDE that provides continuous checking of the document consistency.
In this paper, we present how a specific software-engineering certification standard, namely CENELEC 50128, can be modeled inside Isabelle/DOF. Based on an ontology covering a substantial part of this standard, we present how Isabelle/DOF can be applied to a certification case-study in the railway domain.

Keywords: Certification of Safety-Critical Systems · CENELEC 50128 · Formal Document Development · Isabelle/DOF · Isabelle/HOL

1 Introduction

The initial motivation of this work lies in a failure: the second author was part of the theorem proving team of the EUROMILS project which attempted to certify the commercial operating system PikeOS 3.4 according to CC EAL5+ [12]. When the project came to an end, it became clear that the project would not achieve this goal.[3] The evaluator informed us about an embarrassing number of

[3] The company SYSGO/Thales behind this initiative finally abandoned the approach and restarted a certification on a later version.

inconsistencies in the provided documents (written in Word, Excel, LaTeX, and documents generated from 3 000 lines of proof code written in Isabelle/HOL). While impressed by the proof work done, he pointed out that the overall inconsistency in the documents provided made an evaluation impossible. For example, he highlighted a number of informal definitions of the security target that did not fit to what was modeled, or that the implementation model did not fit to what was tested. He also made accurate comments on inconsistent references and terminologies.

This failure led the authors of this paper to the following insight: For a successful formal certification process, it is *by far not enough* to have abstract models and corresponding refinement proofs to some implementation model (or even, as is the case in [13] or the seL4 initiative [20], to realistic C code). Certification processes targeting higher-levels of assurance such as CENELEC 50128/SIL 4 [11] or CC EAL7 [12] are all requiring the use of formal methods. Therefore, they are a rewarding target for research in this domain. Their core concern, however, is the traceability of requirements, assumptions, application constraints of various nature, and the demonstration of evidence for their satisfaction. Proofs, as a means to connect models, are part of the solution; however the underlying notion of *evidence* in certifications comprises also tests, informal arguments or just references to an expert opinion.

In a wider perspective, it turns out to be a substantial problem in large, distributed development processes, to keep track of the evolution of project specific knowledge and to control the overall documentation effort.

In this paper, we present a methodology together with a set of mechanisms and techniques for an automated impact analysis of document changes in order to *achieve* coherence as well as to *maintain* it during evolution. For this purpose, we present

1. the formal development in Isabelle/HOL of an industrial case-study: the odometric function measuring position, speed, and acceleration of a train,
2. an ontology formalizing parts of the CENELEC 50128 standard in ODL, for which we use Isabelle/DOF [8, 10], and
3. the methodology to annotate an Isabelle formal development with the concepts of this CENELEC in order to enforce coherence.

With respect to the formal development, we restrict ourselves to core aspects of this development, ranging from modeling the physics of a train over the physics of a measuring device down to a C implementation running on a Sabre-Light Card using seL4.

2 Background

In this section, we provide a guided tour through the underlying technologies of this paper: 1. Isabelle and Isabelle/HOL, 2. Isabelle/DOF and its Ontology Definition Language (ODL).

2.1 Isabelle and HOL

While still widely perceived as an interactive theorem proving environment, Isabelle [22] has become a generic system framework providing an infrastructure for plug-ins. This comprises extensible state components, extensible syntax, code-generation, and advanced documentation support. The plugin Isabelle/HOL offers a modeling language similar to functional programming languages extended by a logic and automated proof and animation techniques.

2.2 The Isabelle/DOF Framework

Isabelle/DOF [8–10] is a document ontology framework that extends Isabelle/HOL. We understand by a *document ontology* structured meta-data attached to an integrated document allowing classifying text-elements, connect them to typed meta-data, and establishing typed links between text- and formal elements (such as definitions, proofs, code, test-results, etc).

Isabelle/DOF offers basically two things: a language called ODL to *specify* a formal ontology, and ways to *annotate* an integrated document written in Isabelle/HOL with the specified meta-data. Additionally, Isabelle/DOF generates from an ontology a family of semantic macros—called *antiquotations* that may appear in text or code—allowing establishing machine-checked links between classified entities. Not unlike the UML/OCL meta-model, ODL offers class invariants as well as means to express structural constraints in documents. Unlike UML, however, Isabelle/DOF allows for integrated documents with informal and formal elements including the necessary management of logical contexts.

The perhaps most attractive aspect of Isabelle/DOF is its deep integration into the IDE of Isabelle (PIDE), which allows hypertext-like navigation as well as fast user-feedback during development and evolution of the integrated document. This includes rich editing support, including on-the-fly semantics checks, hinting, or auto-completion. Isabelle/DOF supports LaTeX-based document generation as well as ontology-aware "views" on the integrated document, i. e., specific versions of generated PDF addressing, for example, different stake-holders.

2.3 A Guided Tour Through ODL

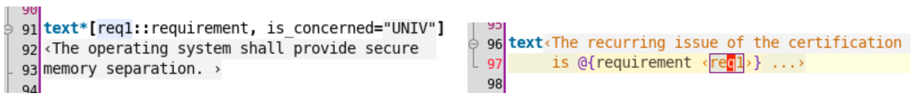
Isabelle/DOF provides a strongly typed Ontology Definition Language (ODL) that provides the usual concepts of ontologies such as

– *document class* (using the `doc_class` keyword) that describes a concept,
– *attributes* specific to document classes (attributes might be initialized with default values),
– a special link, the reference to a super-class, establishes an *is-a* relation between classes;
– classes may refer to other classes via a regular expression in an optional *where* clause (a class with a where clause is called *monitor*);

The types of attributes are HOL-types. Thus, ODL can refer to any predefined type from the HOL library, e. g., `string`, `int` as well as parameterized types, e. g., `option`, `list`. As a consequence of the Isabelle document model, ODL definitions may be arbitrarily mixed with standard HOL type definitions. Document class definitions are HOL-types, allowing for formal *links* to and between ontological concepts. For example, the basic concept of requirements from CENELEC 50128 [11] is captured in ODL as follows:

```
doc_class requirement = text_element + (* derived from text_element    *)
          long_name    ::string option  (* an optional string attribute *)
          is_concerned::role set        (* roles working with this req.  *)
```

This ODL class definition maybe part of one or more Isabelle theory–files capturing the entire ontology definition. Isabelle's session management allows for pre-compiling them before being imported in the actual target documentation required to be compliant to this ontology.



```
text*[req1::requirement, is_concerned="UNIV"]
‹The operating system shall provide secure
memory separation. ›
```

```
text‹The recurring issue of the certification
    is @{requirement ‹req1›} ...›
```

(a) A Text-Element as Requirement. (b) Referencing a requirement.

Fig. 1: Referencing a Requirement.

Fig. 1 shows an ontological annotation of a requirement and the referencing via an antiquotation `@{requirement ‹req1›}` generated by Isabelle/DOF from the above class definition. Undefined or ill-typed references were rejected, the high-lighting displays the hyperlinking which is activated on a click. Revising the actual definition of *requirement*, it suffices to click on its keyword: the IDE will display the class-definition and its surrounding documentation in the ontology.

Isabelle/DOF's generated antiquotations are part of a general mechanism of Isabelle's standard antiquotations heavily used in various papers and technical reports. For example, in the following informal text, the antiquotation `@{thm refl}` refers to the reflexivity axiom from HOL:

```
text‹According to the reflexivity axiom @{thm refl}, we obtain in Γ
      for @{term ‹fac 5›} the result @{value ‹fac 5›}.›
```

In the PDF output, this is represented as follows:

According to the reflexivity axiom $x = x$, we obtain in Γ for fac 5 the result 120.

The antiquotation `@{value ‹fac 5›}` refers to a function that is defined in the preceding logical context (and parsed as inner syntax) to compute the value of 5!, i. e., 120. Summing up, antiquotations can refer to formal content, can be type-checked before being displayed and can be used for calculations before actually being typeset. All these features can be used for the calculation of attribute values (as in Fig. 1, observe the value `UNIV` used to set the attribute `is_concerned` is a HOL-constant denoting the universal set).

Finally, for each ontological concept, a custom representation, using LaTeX-notation, for the generated PDF document can be defined. The latter includes, e. g., the possibility to automatically generated glossaries or lists of concepts.

3 The Underlying Methodology

We assume that all documentation, models, proofs, test-execution scripts, and code are managed in an *integrated document*; Isabelle/DOF supports this approach by admitting an acyclic graph of sub-documents consisting of different type of files. We are well aware that this precondition will raise the question of scalability; however, the Isabelle system is based on a document model allowing for efficient, parallelized evaluation and checking of its document content (cf. [5, 24, 25] for the fairly innovative technologies underlying the Isabelle architecture). These technologies allow for scaling up to fairly large documents: we have seen documents with 150 files be loaded (excluding proof-checking) in about 4 min, and individual files—like the x86 model generated from Antony Fox's L3 specs—can have 80 kLOC and were loaded in about the same time.

Only *inside* an integrated document Isabelle/DOF can manage and check the mutual dependencies and give automated and fast feedback to the validity of ontological dependencies; document boundaries imply a drastically reduced information flow and the need for complex round-engineering techniques.

Methodologically, the integrated document is central; subsequent versions evolve from the informal to the formal, from unstructured to structured text.

We will use the odometry case-study as a show-case of our methodology, which consists of four (not necessarily sequential) phases:

1. *Textual Elicitation* of informal pre-documents into an integrated source,
2. *Formal Enrichment* of the integrated source with definitions in HOL, capturing the lexicon of concepts and notions concerning system environment, architecture, and required performances,
3. *Verification* of the theory resulting from these definitions; in our case, this comprises formal proofs of safety properties or refinements from high-level system models to design and from design to the code-level, and
4. *Ontological Embedding* of text-, model-, and evidence-elements in the integrated source into a concrete target ontology of a certification standard.

In the following, we will present selected snapshots of the document evolution covering the phases 1 to 3, while phase 4 is presented in Sect. 6.

4 A Case-Study: An Odometer-Subsystem

In our case study, we will follow the phases of analysis, design, and implementation of the odometry function of a train. This software processes data from an odometer to compute the position, speed, and acceleration of a train. This system provides the basis for many safety critical decisions, e. g., the opening of the doors. Due to its relatively small size, it is a manageable, albeit realistic target for a comprehensive formal development: it covers a physical model of the environment, the physical and architectural model of the odometer including the problem of numerical sampling, and the boundaries of efficient computations. The interplay between environment and measuring-device as well as the implementation problems on a platform with limited resources makes the odometer a fairly typical safety critical embedded system.

We start with our phase called *textual elicitation* of a number of informal documents available at the beginning of the development; since our approach assumes an integrated document for the entire project—only inside these documents, the checking if coherence is possible—all initial texts must be brought into this format. We selected a few text snippets from original documents and their treatment during this phase.

4.1 System Requirements Specification as an *Integrated Source*

Textual Elicitation of "Basic Principles of Motion and Motion Measurement." The motion of a train and the method for measuring the motion is textually described as follows: "A rotary encoder measures the motion of a train. To achieve this, the encoder's shaft is fixed to the trains wheels axle. When the train moves, the encoder produces a signal pattern directly related to the trains progress. By measuring the fractional rotation of the encoders shaft and considering the wheels effective ratio, relative movement of the train can be calculated."

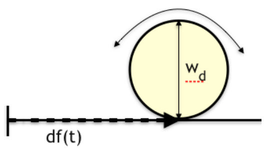


Fig. 2: Motion sensing via an odometer.

Fig. 2 shows that we model a train, seen from a pure kinematics standpoint, as physical system characterized by a one-dimensional continuous distance function, which represents the observable of the physical system. Concepts like speed and acceleration were derived concepts defined as their (gradient) derivatives. We assume the use of the meter, kilogram, and second (MKS) system.

This model is already based on several fundamental assumptions relevant for the correct functioning of the system and for its integration into the system as a whole. In particular, we need to make the following assumptions explicit:

– that *a perfectly circular wheel profile* is assumed, with constant radius,
– that the *slip between the trains wheel and the track* is *negligible*,
– the distance between all teeth of a wheel is the same and constant, and
– the sampling rate of positions is a given constant.

These assumptions have to be traced throughout the certification process as *derived requirements* (or, in CENELEC terminology, as *exported constraints*), which is also reflected by their tracing throughout the body of certification documents. This may result in operational regulations, e. g., regular checks for tolerable wheel defects. As for the *no slip*-assumption, this leads to the modeling of constraints under which physical slip can be neglected: the device can only produce reliable results under certain physical constraints (speed and acceleration limits). Moreover, the *no slip*-assumption motivates architectural arrangements for situations where this assumption cannot be assured (as is the case, for example, of an emergency breaking) together with error-detection and error-recovery.

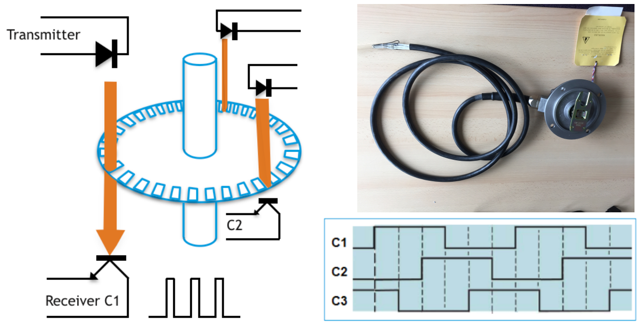


Fig. 3: An odometer with three sensors `C1`, `C2`, and `C3`.

Textual Elicitation of "System Architecture." The requirements analysis also contains a sub-document *interface data* which can be subsumed into the CENELEC notion *system architecture description*. It contains technical drawing of the odometer, a timing diagram (see Fig. 3), and tables describing the encoding of the position for the possible signal transitions of the sensors `C1`, `C2`, and $C3$.

Textual Elicitation of "System Interfaces." The initial document contains a section *Interface data* which is subsumed under the CENELEC notion *functions and interfaces* required as part of the requirements analysis. It describes the technical format of the output of the odometry function, e. g., specifies the output *speed* as given by a `int_32` to be the "Estimation of the speed (in mm/sec) evaluated over the latest N_{avg} samples" where the speed refers to the physical speed of the train and N_{avg} a parameter of the sub-system configuration.

Textual Elicitation of "Required Performances." The analysis documents were relatively implicit on the expected precision of the measurements; however, certain interface parameters like `Odometric_Position_TimeStamp` (a counter on the number of samplings) and `Relative_Position` are defined by an unsigned 32 bit integer. The textual elicitation phase even revealed minor errors in the consistent spelling of parameter names and a more severe ontological confusion between the *physical time* (i. e., *time* in the sense of the physical environment model) and the time measured by the device, which is not identical under all circumstances. These parameter definitions imply *exported constraints* (CENELEC notion) concerning the acceptable time of service as well the maximum distance before a necessary reboot of the subsystem. For our case-study, we assume maximum deviation of the `Relative_Position` to the theoretical distance.

The requirement analysis document describes the physical environment, the architecture of the measuring device, and the required format and precision of the measurements of the odometry function as represented (see Fig. 4).

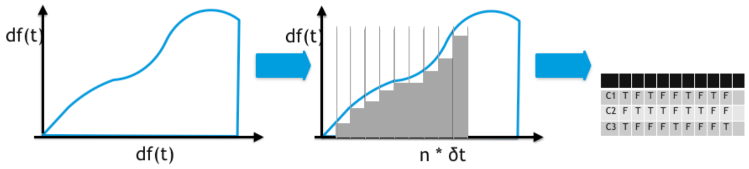


Fig. 4: Real distance vs. discrete distance vs. shaft-encoder sequence

Textual Elicitation of the "Software Design Spec" (Resume). The design provides a function that manages an internal first-in-first-out buffer of shaft-encodings and corresponding positions. Central for the design is a step-function analyzing new incoming shaft encodings, checking them and propagating two kinds of error-states (one allowing recovery, another one, fatal, signaling, e. g., a defect of the receiver hardware), calculating the relative position, speed and acceleration.

Textual Elicitation of the "Software Implementation" (Resume). While the design is executable on a Linux system, it turns out that the generated code from an Isabelle model is neither executable on resource-constraint target platform, an ARM-based Sabre-light card, nor certifiable, since the compilation chain via ML to C implies the inclusion of a run-time system and quite complex libraries. We adopted therefore a similar approach as used in the seL4 project [21]: we use a hand-written implementation in C and verify it via AutoCorres [17] against the design model. The hand-written C-source is integrated into the Isabelle/HOL technically by registering it in the build-configuration and logically by a trusted C-to-HOL compiler included in AutoCorres.

4.2 Formal Enrichment of the Software Requirements Specification

After the *eliciation*-phase, we turn now to *formal enrichment* phase. For example, the assumptions in the system architecture were formalized by the Isabelle/HOL definitions, which were added as close as possible to the informal text:

```
definition teeth_per_wheelturn::nat  (tpw) where tpw ≡ SOME x. x > 0
definition wheel_diameter::real (w_d) where w_d ≡ SOME x. x > 0
definition wheel_circumference::real (w_circ) where w_circ ≡ pi * w_d
definition δs_res::real where δs_res ≡ w_circ / (2 * 3 * tpw)
```

Here, `real` refers to the real numbers as defined in the HOL-Analysis library, which provides concepts such as Cauchy Sequences, limits, differentiability, and a very substantial part of classical Calculus. SOME is the Hilbert choice operator from HOL; the definitions of the model parameters admit all possible positive values as uninterpreted constants. Our perfect-wheel assumption is translated into a calculation of the circumference of the wheel, while δs_{res}, the resolution of the odometer, can be calculated from the these parameters. HOL-Analysis permits to formalize the fundamental physical observables:

```
type_synonym distance_function = real⇒real
definition Speed::distance_function⇒real⇒real where Speed f ≡ deriv f
definition Accel::distance_function⇒real⇒real
                                            where Accel f ≡ deriv (deriv f)
```

which permits to constrain the central observable distance_function in a way that they describe the space of "normal behavior" where we expect the odometer to produce reliable measurements over a distance_function df.

The essence of the physics of the train is covered by the following definition:

```
definition normally_behaved_distance_function :: (real ⇒ real) ⇒ bool
    where  normally_behaved_distance_function df =
            ( ∀ t. df(t) ∈ ℝ≥0  ∧  (∀ t ∈ ℝ≤0. df(t) = 0)
             ∧ df differentiable on_R ∧ (Speed df)differentiable on_R
             ∧ (Accel df)differentiable on_R
             ∧ (∀ t. (Speed df) t ∈ {-Speed_Max .. Speed_Max})
             ∧ (∀ t. (Accel df) t ∈ {-|Accel_Max| .. |Accel_Max|}))
```

which constrains the distance functions in the bounds described of the informal descriptions and states them as three-fold differentiable function in certain bounds concerning speed and acceleration. Violations, in particular of the constraints on speed and acceleration, *do* occur in practice. In such cases, the global system adapts recovery strategies that are out of the scope of our model. Concepts like shaft_encoder_state (a triple with the sensor values C1, C2, C3) were formalized as types, while tables were defined as recursive functions:

```
fun phase₀ :: nat ⇒ shaft_encoder_state   where
   phase₀ (0) =  (| C1 = False, C2 = False, C3 = True  |)
  |phase₀ (1) =  (| C1 = True,  C2 = False, C3 = True  |)
  |phase₀ (2) =  (| C1 = True,  C2 = False, C3 = False |)
  |phase₀ (3) =  (| C1 = True,  C2 = True,  C3 = False |)
  |phase₀ (4) =  (| C1 = False, C2 = True,  C3 = False |)
  |phase₀ (5) =  (| C1 = False, C2 = True,  C3 = True  |)
  |phase₀ x   =  phase₀(x - 6)
definition Phase ::nat⇒shaft_encoder_state where Phase(x) = phase₀(x-1)
```

We express the *shaft encoder sequences* as a translations of distance functions:

```
definition encoding::distance_function⇒nat⇒real⇒shaft_encoder_state
  where encoding df init_pos ≡ λx. Phase(nat⌊df(x) / δs_res⌋ + init_pos)
```

where $\texttt{init}_{pos}$ is the initial position of the wheel. sampling's were constructed from encoding sequences over discretized time points:

```
definition sampling::distance_function⇒nat⇒real⇒nat⇒shaft_encoder_state
  where sampling df init_pos δt ≡ λn::nat. encoding df init_pos (n * δt)
```

The sampling interval δt (the inverse of the sampling frequency) is a critical parameter of the configuration of a system.

Finally, we can formally define the required performances. From the interface description and the global model parameters such as wheel diameter, the number of teeth per wheel, the sampling frequency etc., we can infer the maximal time of service as well the maximum distance the device can measure. As an example configuration, choosing 1 m for w_d, 100 for `tpw`, 80 km/h $\texttt{Speed}_{\texttt{Max}}$, and 14400 Hz for the sampling frequency, results in an odometer resolution of 2.3 mm, a maximum distance of 9878 km, and a maximal system up-time of 123.4 h. The required precision of an odometer can be defined by a constant describing the maximally allowed difference between `df(n*δt)` and `sampling df init`$_{\texttt{pos}}$ `δt n` for all `init`$_{\texttt{pos}}$ `∈{0..5}`.

4.3 Verification of the Software Requirements Specification

The original documents contained already various statements that motivate certain safety properties of the device. For example, the `Phase`-table excludes situations in which all sensors `C1`, `C2`, and `C3` are all "off" or situations in which sensors are "on," reflecting a physical or electrical error in the odometer. It can be shown by a very small Isabelle case-distinction proof that this safety requirement follows indeed from the above definitions:

```
lemma Encoder_Property_1:(C1(Phase x) ∧ C2(Phase x) ∧ C3(Phase x))=False
  proof (cases x)
    case 0 then show ?thesis  by (simp add: Phase_def)
  next
    case (Suc n) then show ?thesis
      by(simp add: Phase_def,rule_tac n = n in cycle_case_split,simp_all)
  qed
```

for all positions `x`. Similarly, it is proved that the table is indeed cyclic: $\texttt{phase}_0$ `x =` $\texttt{phase}_0$`(x mod 6)` and locally injective: `∀x<6. ∀y<6.` $\texttt{phase}_0$ `x =` $\texttt{phase}_0$ `y → x = y`. These lemmas, building the "theory of an odometer," culminate in a theorem that we would like to present in more detail.

```
theorem minimal_sampling :
  assumes * : normally_behaved_distance_function df
    and  ** : δt * Speed_Max < δs_res
  shows ∀ δX≤δt. 0<δX →
              ∃f. retracting (f::nat⇒nat) ∧
                  sampling df init_pos δX = (sampling df init_pos δt) o f
```

This theorem states for `normally_behaved_distance_functions` that there is a minimal sampling frequency assuring the safety of the measurements; samplings on some `df` gained from this minimal sampling frequency can be "pumped up" to samplings of these higher sampling frequencies; they do not contain more information. Of particular interest is the second assumption, labelled "`**`," which establishes a lower bound from $\texttt{w}_{\texttt{circ}}$, `tpw`, $\texttt{Speed}_{\texttt{Max}}$ for the sampling frequency.

Methodologically, this represents an exported constraint that can not be represented *inside* the design model: it means that the computations have to be fast enough on the computing platform in order to assure that the calculations are valid. It was in particular this exported constraint that forced us to give up the original plan to generate the code from the design model and to execute this directly on the target platform.

For our example configuration (1m diameter, 100 teeth per wheel, 80 km/h max), this theorem justifies that 14,4 kHz is indeed enough to assure valid samplings. Such properties are called "internal consistency of the software requirements specification" in the CENELEC standard [11], 7.2.4.22 and are usually addressed in an own report.

5 The CENELEC Ontology

Modeling an ontology from a semi-formal text such as [11] is, like any other modeling activity, not a simple one-to-one translation of some concepts to some formalism. Rather, implicit and self-understood principles have to be made explicit, abstractions have to be made, and decisions about the kind of desirable user-interaction may have an influence similarly to design decisions influenced by strengths or weaknesses of a programming language.

5.1 Tracking Concepts and Definitions

Isabelle/DOF is designed to annotate text elements with structured meta-information and to reference these text elements throughout the integrated source. A classical application of this capability is the annotation of concepts and terms definitions—be them informal, semi-formal or formal—and their consistent referencing. In the context of our CENELEC ontology, e. g., we can translate the third chapter of [11] "Terms, Definitions and Abbreviations" directly into our Ontology Definition Language (ODL). Picking one example out of 49, consider the definition of the concept "traceability" in paragraphs 3.1.46 (a notion referenced 31 times in the standard), which we translated directly into:

```
Definition*[traceability::concept]⟨ degree to which relationship
  can be established between two or more products of a development
  process, especially those having a predecessor/successor or
  master/subordinate relationship to one another. ⟩
```

In the integrated source of the odometry study, we can reference in a text element to this concept as follows:

```
text*[...]⟨  ... to assure @{concept traceability} for
  @{requirement bitwiseAND}, we prove  ... ⟩
```

The presentation of this document element inside Isabelle/DOF is immediately hyperlinked against the `Definition*` element shown above; this serves as documentation of the standard for the development team working on the integrated source. The PDF presentation of such links depends on the actual configurations for the document generation; We will explain this later. CENELEC foresees also a number of roles, phases, safety integration levels, etc., which were directly translated into HOL enumeration types usable in ontological concepts of ODL.

```
datatype role =
   PM  (* Program Manager *) |  RQM (* Requirements Manager *)
 | DES (* Designer *)        |  IMP (* Implementer *)        |
 | VER (* Verifier *)        |  VAL (* Validator *)          |  ...
datatype phase =
   SYSDEV_ext (* System Development *) | SPl (* Software Planning     *)
 | SR    (* Software Requirement    *) | SA  (* Software Architecture *)
 | SDES  (* Software Design         *) | ...
```

Similarly, we can formalize the Table A.5: Verification and Testing of [11]: a classification of *verification and testing techniques*:

```
datatype vnt_technique =
            formal_proof thm list    | stat_analysis
          | dyn_analysis dyn_ana_kind  | ...
```

In contrast to the standard, we can parameterize `formal_proof` with a list of theorems, an entity known in the Isabelle kernel. Here, Isabelle/DOF assures for text elements annotated with theorem names, that they refer indeed to established theorems in the Isabelle environment. Additional checks could be added to make sure that these theorems have a particular form, for example.

While we claim that this possibility to link to theorems (and test-results) is unique in the world of systems attempting to assure traceability, referencing a particular (proven) theorem is definitively not sufficient to satisfy the claimed requirement. Human evaluators will always have to check that the provided theorem *adequately* represents the claim; we do not in the slightest suggest that their work is superfluous. Our framework allows to statically check that tests or proofs have been provided, at places where the ontology requires them to be, and both assessors and developers can rely on this check and navigate through related information easily. It does not guarantee that intended concepts for, e. g., safety or security have been adequately modeled.

5.2 Major Ontological Entities: Requirements and Evidence

We introduce central concept of a *requirement* as an ODL `doc_class` based on some generic basic library `text_element` providing basic layout attributes.

```
doc_class requirement = text_element +
   long_name    :: string option
   is_concerned :: role set
```

where the `roles` are exactly the ones defined in the previous section and represent the groups of stakeholders in the CENELEC process. Therefore, the `is_concerned`-attribute allows expressing who “owns” this text-element. Isabelle/DOF supports a role-based presentation, e. g., different presentation styles of the integrated source may decide to highlight, to omit, to defer into an annex, text entities according to the role-set.

Since ODL supports single inheritance, we can express sub-requirements and therefore a style of requirement decomposition as advocated in GSN [19]:

```
doc_class sub_requirement =
   decomposes :: requirement
   relates_to :: requirement set
```

5.3 Tracking Claims, Derived Requirements and Evidence

As an example for making explicit implicit principles, consider the following statement [11], pp. 25.:

> The objective of software verification is to examine and arrive at a judgment based on evidence that output items (process, documentation, software or application) of a specific development phase fulfill the requirements and plans with respect to completeness, correctness and consistency.

The terms *judgment* and *evidence* are used as a kind of leitmotif throughout the CENELEC standard, but they are neither explained nor even listed in the general glossary. However, the standard is fairly explicit on the *phase*s and the organizational roles that different stakeholders should have in the process. We express this key concept "judgment", by the following class:

```
doc_class judgement =
   refers_to       :: requirement
   evidence        :: vnt_technique list
   status          :: status
   is_concerned    :: role set <= {VER,ASR,VAL}
```

As one can see, the role set is per default set to the verification team, the assessors and the validation team.

There are different views possible here: an alternative would be to define `evidence` as ontological concept with `vnt_technique`’s (rather than an attribute of judgement) and consider the basis of judgments as a relation between requirements and relation:

```
doc_class judgement =
   based_on        :: (requirement × evidence) set
   status          :: status
   is_concerned    :: role set <= {VER,ASR,VAL}
```

More experimentation will be needed to find out what kind of ontological modeling is most adequate for developers in the context of Isabelle/DOF.

6 Ontological Embedding and Compliance

From the variety of different possibilities for adding CENELEC annotations to the integrated source, we will, in the following, point out three scenarios from the phase *ontological embedding* (cf. Sect. 3).

Internal Verification of Claims in the Requirements Specification. In our case, the SR-team early on detected a property necessary for error-detection of the device (c.f. Sect. 4.3):

```
text*[encoder_props::requirement]⟨ The requirement specification team ...
    C1 & C2 & C3  = 0   (bitwise logical AND operation)
    C1 | C2 | C3  = 1   (bitwise logical OR operation) ⟩
```

After the Isabelle proofs shown in Sect. 4.3, we can register the theorems directly in an evidence statement:

```
text*[J1::judgement, refers_to=@{docitem ⟨encoder_props⟩},
      evidence=[formal_proof[@{thm ⟨Encoder_Property_1⟩},
                              @{thm ⟨Encoder_Property_2⟩}]]]
  ⟨The required encoder properties are in fact verified to be consistent
   with the formalization of @{term phase₀}.⟩
```

The references `@{ ... }`, called antiquotation, allow us not only to reference to formal concepts, they are checked for consistency and there are also antiquotations that print the formally checked content (e. g., the statement of a theorem).

Exporting Claims of the Requirements Specification. By definition, the main purpose of the requirement specification is the identification of the safety requirements. As an example, we state the required precision of an odometric function: for any normally behaved distance function `df`, and any representable and valid sampling sequence that can be constructed for `df`, we require that the difference between the physical distance and distance calculable from the *Odometric-Position-Count* is bound by the minimal resolution of the odometer.

```
text*[R5::safety_requirement]⟨We can now state ... ⟩
definition
Odometric_Position_Count_precise::(shaft_encoder_state list⇒output)⇒bool
where Odometric_Position_Count_precise odofunction ≡
      (∀ df. ∀S.  normally_behaved_distance_function df
             ⟶ representable S
             ⟶ valid_sampling S df
             ⟶ (let pos = uint(Odometric_Position_Count(odofunction S))
                in |df((length S - 1)*δt_odo) - (δs_res * pos)| ≤ δs_res))

update_instance*[R5::safety_requirement,
       formal_definition:=[@{thm ⟨Odometric_Position_Count_precise_def⟩}]]
```

By `update_instance*`, we book the property `Position_Count_precise_def` as `safety_requirement`, a specific sub-class of `requirements` requesting a formal definition in Isabelle.

Exporting Derived Requirements. Finally, we discuss the situation where the verification team discovered a critical side-condition for a major theorem necessary for the safety requirements; this was in our development the case for the condition labelled "`**`" in Sect. 4.3. The current CENELEC standard clearly separates "requirement specifications" from "verification reports," which is probably motivated by the overall concern of organizational separation and of document consistency. While this document organization is possible in Isabelle/DOF, it is in our experience often counter-productive in practice: organizations tend to defend their documents because the impact of changes is more and more difficult to oversee. This effect results in a dramatic development slow-down and an increase of costs. Furthermore, these barriers exclude situations where developers perfectly know, for example, invariants, but can not communicate them to the verification team because the precise formalization is not known in time. Rather than advocating document separation, we tend to integrate these documents, keep proof as close as possible to definitions, and plead for consequent version control of the integrated source, together with the proposed methods to strengthen the links between the informal and formal parts by anti-quotations and continuous ontological checking. Instead of separation of the documents, we would rather emphasize the *separation of the views* of the different document representations. Such views were systematically generated out of the integrated source in different PDF versions and for each version, document specific consistency guarantees can be automatically enforced.

In our case study, we define this condition as predicate, declare an explanation of it as `SRAC` (CENELEC notion for: safety-related application condition; ontologically, this is a derived class from `requirement`.) and add the definition of the predicate into the document instance as described in the previous section.

7 Generating Document Variants

Often in certification processes, traditional documents are required. Reasons for this include that traditional documents can ensure long-term archivability (which is much harder to ensure for a interactive document that requires Isabelle/HOL). Moreover, the requirements for reading and checking traditional documents are much smaller and no Isabelle expertise is required. To address these needs, Isabelle/DOF can generate a static certification document in the PDF/A format (i. e., the variant of PDF for archiving as defined in the ISO standard 19005)—reading these documents only requires a PDF reader, which one can expect to be available even 50 years from now.

In the context of the CENELEC 50128, we generate one document containing all sub-documents (including all formal proofs) and one document for reach role. The latter only contain the aspects relevant for this particular role (based on the `is_concerned` attribute). For each of the document variants, both semantical and syntactical consistency is checked and the PDF generation fails if these checks (e. g., due to dangling references in a sub-document) are not successful. In addition, we also generate role-specific (hyper-linked) glossaries and tables of relevant concepts (e. g., a table of all `SRAC`s). The latter helps validators that either prefer to work with the final PDF instead of working with the interactive Isabelle/DOF system directly.

8 Related Work

Already in 1993, the need for an integrated and ontological under-pinned document model that are able to integrate formal verification aspects has been motivated by Rushby [23]. More recent reports on the industrial practice of high-assurance safety certifications, e. g., [14, 16, 18], show that not much has happened since the report from Rushby: certification processes still rely on a plethora of tools managing different aspects of the same system and/or model, and ensuring the consistency between the different tools and documents is a risk (and a significant cost factor in a certification).

The support for modeling ontologies in Isabelle/DOF shares similarities with existing ontology editors such as Protégé [4], Fluent Editor [1], NeOn [2], or OWLGrEd [3]. These editors allow for defining ontologies and also provide certain editing features such as auto-completion. In contrast, Isabelle/DOF does not only allow for defining ontologies, directly after defining an ontological concept, they can also be instantiated and their correct use is checked immediately.

Existing works on using ontologies as part of safety-critical (e. g., [7, 26]) or security-critical (e. g., [15]) focus on using ontologies for structuring queries on the set of specifications documents. While not discussed in this paper, Isabelle/DOF supports this time of knowledge management as well: the Isabelle/DOF editor allows for interactively querying for instances of concepts defined in the underlying ontologies as well as for the formal artifacts (formal definitions, proofs, etc.). To our knowledge, none of the existing works provides a deep integration of formal and semi-formal content of certification documents.

9 Conclusion

We presented a methodology for developing certification documents including semi-formal and formal content by using Isabelle/DOF, a framework for ontological modeling and document enforcement. Isabelle/DOF is deeply integrated into Isabelle/PIDE, which allows for a particularly fluid development, immediate ontological feedback and a strong and efficient impact-analysis (already known from developments in interactive theorem proving).

We demonstrate the methodology by a) modeling a non-trivial part of the CENELEC certification standard [11] in Isabelle/DOF ODL, and apply this b) to a non-trivial formal development of a safety-critical component in railways systems. The study consists of five theories with about 4 000 LOC, consisting of documentation, definitions, proofs and test-executions of the model. Various PDF presentations of the integrated source were generated, depending on the different roles assumed in the process. The target C-code is about 300 LoC's, and has been integrated in an experimental continuous build-continuous integration environment on top of seL4 [6].

The approach offers mechanical checking of the links between formal and informal parts, technically assured traceability and, last but not least, fast impact analysis on changes, which is usually in the order of a few seconds. Experiments with the entire seL4-stack (100 theories, 200 000 LOC and 10 000 LOC of C code show that our approach can scale up to the size of integrated, medium-size critical software-subsystems [6].

Availability. The Isabelle/DOF framework [9], the discussed ontology definitions, and examples are available at https://git.logicalhacking.com/Isabelle_DOF/Isabelle_DOF/. Isabelle/DOF is licensed under a 2-clause BSD license (SPDX-License-Identifier: BSD-2-Clause).

Acknowledgments. This work has been partially supported by IRT SystemX, Paris-Saclay, France, and therefore granted with public funds of the Program "Investissements d'Avenir."

References

[1] Fluent editor (2018). http://www.cognitum.eu/Semantics/FluentEditor/
[2] The neon toolkit (2018). http://neon-toolkit.org
[3] Owlgred (2018). http://owlgred.lumii.lv/
[4] Protégé (2018). https://protege.stanford.edu
[5] Barras, B., González-Huesca, L.D.C., Herbelin, H., Régis-Gianas, Y., Tassi, E., Wenzel, M., Wolff, B.: Pervasive parallelism in highly-trustable interactive theorem proving systems. In: Intelligent Computer Mathematics - MKM, Calculemus, DML, and Systems and Projects, pp. 359–363 (2013). doi:10.1007/978-3-642-39320-4_29
[6] Bezzecchi, S., Crisafulli, P., Pichot, C., Wolff, B.: Making agile development processes fit for V-style certification procedures. In: ERTS Conference Proceedings (2018)
[7] Bicchierai, I., Bucci, G., Nocentini, C., Vicario, E.: Using ontologies in the integration of structural, functional, and process perspectives in the development of safety critical systems. In: Keller, H.B., Plödereder, E., Dencker, P., Klenk, H. (eds.) Reliable Software Technologies – Ada-Europe 2013, pp. 95–108. Springer (2013)
[8] Brucker, A.D., Ait-Sadoune, I., Crisafulli, P., Wolff, B.: Using the Isabelle ontology framework: Linking the formal with the informal. In: Conference on Intelligent Computer Mathematics (CICM), no. 11006 in Lecture Notes in Computer Science. Springer-Verlag, Heidelberg (2018). doi:10.1007/978-3-319-96812-4_3.
[9] Brucker, A.D., Wolff, B.: Isabelle/DOF (2019). doi:10.5281/zenodo.3370483.
[10] Brucker, A.D., Wolff, B.: Isabelle/DOF: design and implementation. In: Ölveczky, P.C., Salaün, G. (eds.) Software Engineering and Formal Methods (SEFM), no. 11724 in Lecture Notes in Computer Science. Springer-Verlag, Heidelberg (2019). doi:10.1007/978-3-030-30446-1_15.
[11] BS EN 50128:2011: Bs en 50128:2011: Railway applications – communication, signalling and processing systems – software for railway control and protecting systems. Standard, Britisch Standards Institute (BSI) (2014)

[12] Common Criteria: Common criteria for information technology security evaluation (version 3.1), Part 3: Security assurance components (2006). Available as document CCMB-2006-09-003

[13] Daum, M., Dörrenbächer, J., Wolff, B.: Proving fairness and implementation correctness of a microkernel scheduler. Journal of Automated Reasoning **42**(2), 349–388 (2009). doi:10.1007/s10817-009-9119-8.

[14] Denney, E., Pai, G.: Evidence arguments for using formal methods in software certification. In: IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 375–380 (2013). doi:10.1109/ISSREW.2013.6688924

[15] Ekelhart, A., Fenz, S., Goluch, G., Weippl, E.: Ontological mapping of common criteria's security assurance requirements. In: Venter, H., Eloff, M., Labuschagne, L., Eloff, J., von Solms, R. (eds.) New Approaches for Security, Privacy and Trust in Complex Environments, pp. 85–95. Springer (2007)

[16] Gleirscher, M., Ratiu, D., Schatz, B.: Incremental integration of heterogeneous systems views. In: 2007 International Conference on Systems Engineering and Modeling, pp. 50–59 (2007). doi:10.1109/ICSEM.2007.373334

[17] Greenaway, D., Andronick, J., Klein, G.: Bridging the gap: Automatic verified abstraction of C. In: Beringer, L., Felty, A. (eds.) Interactive Theorem Proving, pp. 99–115. Springer (2012)

[18] Kaluvuri, S.P., Bezzi, M., Roudier, Y.: A quantitative analysis of common criteria certification practice. In: Eckert, C., Katsikas, S.K., Pernul, G. (eds.) Trust, Privacy, and Security in Digital Business, pp. 132–143. Springer (2014)

[19] Kelly, T., Weaver, R.: The goal structuring notation – a safety argument notation. In: Dependable Systems and Networks (2004)

[20] Klein, G.: Operating system verification — an overview. Sādhanā **34**(1), 27–69 (2009)

[21] Klein, G., Andronick, J., Elphinstone, K., Murray, T.C., Sewell, T., Kolanski, R., Heiser, G.: Comprehensive formal verification of an OS microkernel. ACM Trans. Comput. Syst. **32**(1), 2:1–2:70 (2014). doi:10.1145/2560537.

[22] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL—A Proof Assistant for Higher-Order Logic, *LNCS*, vol. 2283. Springer (2002). doi:10.1007/3-540-45949-9

[23] Rushby, J.: Formal methods and the certification of critical systems. Tech. Rep. SRI-CSL-93-7, Computer Science Laboratory, SRI International, Menlo Park, CA (1993). Also issued under the title *Formal Methods and Digital Systems Validation for Airborne Systems* as NASA Contractor Report 4551, December 1993

[24] Wenzel, M.: Asynchronous user interaction and tool integration in Isabelle/PIDE. In: Interactive Theorem Proving (ITP), pp. 515–530 (2014). doi:10.1007/978-3-319-08970-6_33

[25] Wenzel, M.: System description: Isabelle/jEdit in 2014. In: Proceedings Eleventh Workshop on User Interfaces for Theorem Provers, UITP 2014, Vienna, Austria, 17th July 2014., pp. 84–94 (2014). doi:10.4204/EPTCS.167.10

[26] Zhao, Y., Sanán, D., Zhang, F., Liu, Y.: Formal specification and analysis of partitioning operating systems by integrating ontology and refinement. IEEE Trans. Industrial Informatics **12**(4), 1321–1331 (2016)