

# A Framework for Proving Ontology-Relations and Runtime Testing Ontology Instances

Idir Ait-Sadoune<sup>1</sup>[0000–0002–6484–8276], Nicolas Méric<sup>2</sup>[0000–0002–0756–7072], and Burkhardt Wolff<sup>3</sup>

<sup>1</sup> Université Paris-Saclay, CentraleSupélec, LMF, France

`idir.aitsadoune@centralesupelec.fr`

<sup>2</sup> Université Paris-Saclay, LMF, France

`nicolas.meric@universite-paris-saclay.fr`

<sup>3</sup> Université Paris-Saclay, LMF, France

`burkhart.wolff@universite-paris-saclay.fr`

**Abstract.** Isabelle/DOF is an ontology framework on top of Isabelle [8,4]. Isabelle/DOF allows for the formal development of ontologies as well as continuous checking that a formal document under development conforms to an underlying ontology. Such a document may contain text and code elements as well as formal Isabelle definitions and proofs. Thus, Isabelle/DOF is designed to annotate and trace typed meta-data within formal developments in Isabelle.

In this paper we extend Isabelle/DOF with *invariants* (or: ontological *rules*). Via a reflection mechanism, this allows for efficient run-time checking of abstract properties of formal content under evolution. Additionally, invariants have a formal representation in HOL amenable to formal proofs over mappings between different ontologies. With this feature widely called *ontology mapping* in the literature, our framework paves the way for advanced uses such as “semantic” search and translation. We demonstrate the use of these new features in an extended ontology used for formal developments targeting CENELEC certifications.

**Keywords:** Ontologies, Formal Documents, Formal Development, Isabelle/HOL, Ontology Mapping, Certification

## 1 Introduction

The linking of *formal* and *informal* information is perhaps the most pervasive challenge in the digitization of knowledge and its propagation. Unsurprisingly, this problem reappears in the libraries with formalized mathematics and engineering such as the Isabelle Archive of Formal Proofs [19] (AFP), which passed the impressive numbers of 650 articles, written by 420 authors at the beginning of 2022. Together with the AFP, there is also a growing body on articles concerned with formal software engineering such as standardized language definitions (e.g., [16,7]), data-structures (e.g., [6,23]), hardware- models (e.g., [15]), security-related specifications (e.g., [5,26]), or operating systems (e.g., [27,17]).

Still, while the problem of logical consistency even under system-changes and pervasive theory evolution is technically solved via continuous proof-checking, the problem of knowledge retrieval and of linking semi-formal explanations to definitions and proofs remains largely open. The central role in technologies addressing the *knowledge* problem is played by *document ontologies*, i. e., a machine-readable form of meta-data attached to document-elements as well as their document discourse. In order to make these techniques applicable to *formal theory development*, the following is needed:

- a general mechanism to define and develop *domain-specific* ontologies,
- ... that should be adapted to entities occurring in formal theories, i. e., provide built-in support for types, terms, theorems, proofs, etc.,
- ways to annotate meta-data generated by ontologies to the document elements, as “deep” as possible, together with strong validation checks,
- a smooth integration into the theory document development process, and
- ways to relate ontologies and ontology-conform documents along different ontologies by *ontological mappings* and *data translations* <sup>4</sup>.

Recently, Isabelle/DOF [8,4] <sup>5</sup> has been designed as an Isabelle component that attempts to answer these needs. Isabelle/DOF generates from ontology definitions directly integrated into Isabelle theories typed meta-data, that may be annotated to a number of document elements and that were validated “on-the-fly” during the general continuous type and proof-checking process in an IDE (Isabelle/PIDE). Thus, we extend the document-centric view on code, definitions, proofs, text-elements, etc., prevailing in the Isabelle system framework.

In more detail, Isabelle/DOF introduces a number of “ontology aware” text-elements with analogous syntax to standard ones. The difference is a bracket with meta-data of the form:

```

text*[label::classid, attr1=E1, ... attrn=En]< some semi-formal text >
ML*[label::classid, attr1=E1, ... attrn=En]< some SML code >
...

```

In these Isabelle/DOF elements, a meta-data object is created and associated to it. This meta-data can be referenced via its label and used in further computations in text or code.

Admittedly, Isabelle is not the first system that comes into one’s mind when writing a scientific paper, a book, or a larger technical documentation. However, it has a typesetting system inside which is in the tradition of document generation systems such as mkd, Document! X, Doxygen, Javadoc, etc., and which embed formal content such as formula pretty-prints into semi-formal text or code. The analogous mechanism the Isabelle system provides is a machine-checked macro called *antiquotation* that depends on the logical context of the document element.

<sup>4</sup> We follow throughout this text the terminology established in [12], pp. 39 ff.

<sup>5</sup> The official releases are available at <https://zenodo.org/record/6385695>, the developer version at [https://github.com/logicalhacking/Isabelle\\_DOF](https://github.com/logicalhacking/Isabelle_DOF).

With standard Isabelle antiquotations, for example, the following text element of the integrated source will appear in Isabelle/PIDE as follows:

```
text< According to the reflexivity axiom @{thm refl}, we obtain in  $\Gamma$ 
for @{term fac 5} the result @{value fac 5}.>
```

In the corresponding generated LaTeX or HTML output, this looks like this:

```
According to the reflexivity axiom  $\langle x = x \rangle$ , we obtain in  $\Gamma$ 
for  $\langle \text{fac } 5 \rangle$  the result  $\langle 120 \rangle$ .
```

where the meta-texts `@{thm refl}` (“give the presentation of theorem ‘refl’”), `@{term fac 5}` (“parse and type-check ‘fac 5’ in the previous logical context”) and `@{value fac 5}` (“compile and execute ‘fac 5’ according to its definitions”) are built-in antiquotations in HOL.

One distinguishing feature of Isabelle/DOF is that specific antiquotations were generated from an ontology rather than being hard-coded into the Isabelle system infrastructure.

As novel contribution, this work extends prior versions of Isabelle/DOF by

1. support of antiquotations in a new class of contexts, namely *term contexts* (rather than SML code or semi-formal text). Thus, annotations generated from Isabelle/DOF may also occur in  $\lambda$ -terms used to denote meta-data.
2. formal, machine-checked invariants on meta-data, which correspond to the concept of “rules” in OWL [25] or “constraints” in UML, and which can be specified in common HOL  $\lambda$ -term syntax.

For example, the Isabelle/DOF command evaluating the HOL-expression:

```
value*[ass::Assertion, relvce=4::int]
<filter ( $\lambda \sigma. \text{relvce } \sigma > 2$ ) @{Assertion-instances}>
```

where Isabelle/DOF command `value*` type-checks, expands in an own validation phase the *Assertion*-instances-term antiquotation, and evaluates the resulting HOL expression above. Assuming an ontology providing the class *Assertion* having at least the integer attribute *relvce*, the command finally creates an instance of *Assertion* and binds this to label *ass*, while setting its *relvce* to 4.

Beyond the gain of expressivity in Isabelle/DOF ontologies, term-antiquotations pave the way for advanced queries of elements inside an integrated source, and invariants allow for formal proofs over the relations/translations of ontologies and ontology-instances. The latter question raised scientific interest under the label “ontology mapping” for which we therefore present a formal solution. To sum up, we completed Isabelle/DOF to a fairly rich ontology language oriented to interactive theorem proving (ITP) systems, which is a concrete proposal for formal development projects targeting a certification, for technical documentation, for books with a high amount of machine-checked formal content or for mathematical libraries such as the AFP.

## 2 Background

### 2.1 The Isabelle/DOF Framework

Isabelle/DOF [4,8] is a document ontology framework that extends Isabelle/HOL. Isabelle/DOF offers basically two things: a language called Ontology Definition Language (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 *antiquotations* allowing to specify machine-checked links between ODL entities.

The perhaps most attractive aspect of Isabelle/DOF is its deep integration into the IDE of Isabelle (Isabelle/PIDE), which allows a hypertext-like navigation as well as fast user-feedback during development and evolution of the integrated source. 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, e. g., different stake-holders.

### 2.2 A Guided Tour through ODL

Isabelle/DOF provides a strongly typed 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), and
- a special link, the reference to a super-class, establishes an *is-a* relation between classes.

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 [10] is captured in ODL as follows:

```
doc-class requirement = text-element +  
    long-name ::string option  
    is-concerned::role set
```

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 document.

Fig. 1 shows an ontological annotation of a requirement and its referencing via an antiquotation `@{requirement <req1>}`; the latter is generated from the above class definition. Undefined or ill-typed references were rejected, the high-lighting displays the hyperlinking which is activated on a click. The class-definition of *requirement* and its documentation is also revisited via one activation click.

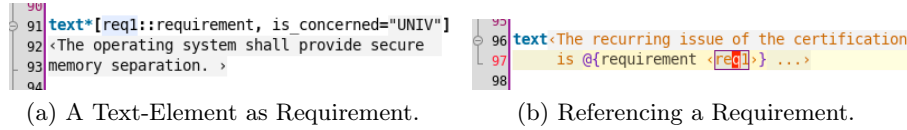


Fig. 1: Referencing a Requirement.

Isabelle/HOL supports records at the level of terms and types. The notation for terms and types is as follows:

- record terms  $\langle x = a, y = b \rangle$  and corresponding record types  $\langle x::A, y::B \rangle$ ,
- the resulting selectors are written  $x\ r, y\ r$ .

In fact, **onto-classes** and the logically equivalent **doc-classes** were represented by *extensible* record types and instances thereof by HOL terms (see [8] for details.). Invariants of an **onto-class** are just predicates over extensible record types and were applied to subclasses.

### 2.3 Term-Evaluations in Isabelle

Besides the powerful, but relatively slow Isabelle rewriting-based proof method, there are two other techniques for term evaluation:

- evaluation via reflection [14] (*eval*), and
- normalization by evaluation [1] (*nbe*).

The former is based on a nearly one-to-one compilation of HOL-level datatype specifications and function definitions into SML datatypes and functions. The latter technique — allowing for free variables in terms — uses a generic data-universe enriched by explicit variables. Both techniques are several orders of magnitude faster than standard rewriting. Isabelle/DOF uses both to generate code that evaluates invariant and data-integrity checks on-the-fly during editing. For all examples in our library, this form of runtime-testing is sufficiently fast to remain unnoticed by the user.

## 3 Term-Context Support, Invariants and Queries in DOF

To offer a smooth integration into the *formal* theory development process, Isabelle/HOL should be able to dynamically interpret the source document. But the specific antiquotations introduced by Isabelle/DOF are not directly recognized by Isabelle/HOL, and the process of term checking and evaluation must be enriched. Previous works [4,8] added a validation step for the SML and semi-formal text contexts. To support Isabelle/DOF antiquotations in the term contexts, the validation step should be improved and a new step, which we call *elaboration* must be added to allow these antiquotations in  $\lambda$ -terms. The resulting process encompasses the following steps:

- Parsing of the term which represents the object in HOL,
- Typeinference/Typechecking of the term,
- Ontological validation of the term: the meta-data of term antiquotations is parsed and checked in the logical context,
- Elaboration of term antiquotations: depending of the antiquotation specific elaboration function, the antiquotations containing references were replaced, for example, by the object they refer to in the logical context,
- Generation of markup information for the Isabelle/PIDE, and
- Code generation:
  - Evaluation of HOL expressions with ontological annotations,
  - Generation of ontological invariants processed simultaneously with the generation of the document (a theory in HOL).

Isabelle/HOL provides inspection commands to type-check (the command **term**) and to evaluate a term (the command **value**). We provide the equivalent commands, respectively **term\*** and **value\***, which additionally support a validation and elaboration phase. A variant of **value\*** is **assert\***, which additionally checks that the term-evaluation results in *True*. Note that term antiquotations are admitted in all Isabelle/DOF commands, not just **term\***, **value\*** etc.

If we take back the example ontology for mathematical papers of [8] shown in Fig. 2 we can define some class instances for this ontology with the **text\*** command, as in Fig. 3. In the instance *intro1*, the term antiquotation  $\@{\text{myauthor} \langle \text{church} \rangle}$ , or its equivalent notation  $\@{\text{myauthor} \text{ "church"}}$ , denotes the instance *church* of the class *myauthor*, where *church* is a HOL-string. One can now reference a class instance in a **term\*** command. In the command **term\*** $\langle \@{\text{myauthor} \langle \text{church} \rangle} \rangle$  the term  $\@{\text{myauthor} \text{ "church"}}$  is type-checked, i. e., the command **term\*** checks that *church* references a term of type *myauthor* against the global context (see Fig. 4).

The command **value\****email*  $\@{\text{author} \langle \text{church} \rangle}$  validates  $\@{\text{myauthor} \text{ "church"}}$  and returns the attribute-value of *myauthor.email* for the *church* instance, i. e. the HOL string *"church@lambda.org"* (see Fig. 5).

Since term antiquotations are logically uninterpreted constants, it is possible to compare class instances logically. The assertion in the Fig. 6 fails: the class instances *proof1* and *proof2* are not equivalent because their attribute *property* differs. When **assert\*** evaluates the term, the term antiquotations  $\@{\text{thm} \text{ "HOL.refl"}}$  and  $\@{\text{thm} \text{ "HOL.sym"}}$  are checked against the global context such that the strings *"HOL.refl"* and *"HOL.sym"* denote existing theorems.

The mechanism of term annotations is also used for the new concept of invariant constraints which can be specified in common HOL syntax. They were introduced by the keyword **invariant** in a class definition (recall Fig. 2). Following the constraints proposed in [4], one can specify that any instance of a class *myresult* finally has a non-empty property list, if its *kind* is *proof* (see the **invariant** *has-property*), or that the relation between *myclaim* and *myresult* expressed in the attribute *establish* must be defined when an instance of the class *myconclusion* is defined (see the **invariant** *establish-defined*).

```

datatype kind = expert-opinion | argument | proof
doc-class myauthor =
  email :: string <= ""
doc-class mytext-section =
  authored-by :: myauthor set <= {}
  level :: int option <= None
doc-class myintro = mytext-section +
  authored-by :: myauthor set <= UNIV
  uses :: string set
invariant author-set :: authored-by  $\sigma \neq \{\}$ 
and force-level :: the (level  $\sigma$ ) > 1
doc-class myclaim = myintro +
  based-on :: string list
doc-class mytechnical = mytext-section +
  formal-results :: thm list
doc-class myresult = mytechnical +
  evidence :: kind
  property :: thm list <= []
invariant has-property :: evidence  $\sigma = \text{proof} \longleftrightarrow \text{property } \sigma \neq []$ 
doc-class myconclusion = mytext-section +
  establish :: (myclaim  $\times$  myresult) set
invariant establish-defined ::  $\forall x. x \in \text{Domain } (\text{establish } \sigma) \rightarrow (\exists y \in \text{Range } (\text{establish } \sigma). (x, y) \in \text{establish } \sigma)$ 

```

**Isabelle code**

Fig. 2: Excerpt of an Example Ontology for mathematical Papers.

```

text*[church::myauthor, email=<church@lambda.org>]<>
text*[proof1::myresult, evidence=proof, property=@{thm <HOL.refl>}]<>
text*[proof2::myresult, evidence=proof, property=@{thm <HOL.sym>}]<>
text*[intro1::myintro, authored-by=@{myauthor <church>}, level=Some 0]<>
text*[intro2::myintro, authored-by=@{myauthor <church>}, level=Some 2]<>
text*[claimNotion::myclaim, authored-by=@{myauthor <church>}
  , based-on=[<Notion1>,<Notion2>], level=Some 0]<>

```

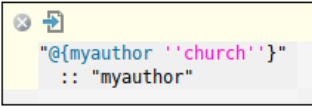
**Isabelle code**

Fig. 3: Some Instances of the Classes of the Ontology of Fig. 2.

```

507 term*<@{myauthor <church>}>
508
509
510
511

```

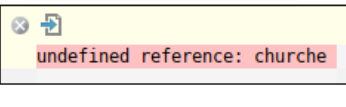


(a) Church is an existing Instance.

```

507 term*<@{myauthor <churche>}>
508
509
510
511

```



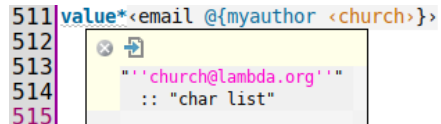
(b) The Churche Instance is not defined.

Fig. 4: Type-Checking of Antiquotations in a Term-Context.

```

511 value* <email @ {myauthor <church>} >
512
513
514
515

```

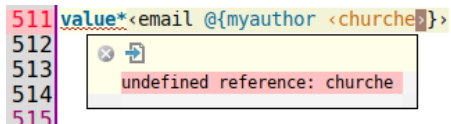


(a) The Evaluation succeeds.

```

511 value* <email @ {myauthor <churche>} >
512
513
514
515

```



(b) The Evaluation fails.

Fig. 5: Evaluation of Antiquotations in a Term-Context.

```

501 assert* <@ {myresult <proof1>} = @ {myresult <proof2>} >
502
503
504

```

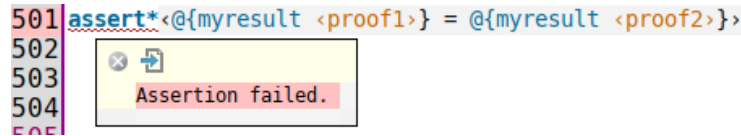


Fig. 6: Evaluation of the Equivalence of two Class Instances.

In Fig. 2, the **invariant** *author-set* of the class *myintro* enforces that a *myintro* instance has at least one author. The  $\sigma$  symbol is reserved and references the future class instance. By relying on the implementation of the Records in Isabelle/HOL [28], one can reference an attribute of an instance using its selector function. For example, *establish*  $\sigma$  denotes the value of the attribute *establish* of the future instance of the class *myconclusion*.

The value of each attribute defined for the instances is checked at run-time against their class invariants. Classes also inherit the invariants from their superclass. As the class *myclaim* is a subclass of the class *myintro*, it inherits the *myintro* invariants. In Fig. 7, we attempt to specify a new instance *claimNotion* of the class *myclaim*. However, the invariant checking triggers an error because the **invariant** *force-level* forces the value of the argument of the attribute *mytext-section.level* to be greater than 1, and we initialize it to *Some 0* in *claimNotion*.

```

517 text [claimNotion::myclaim, authored by = " @ {myauthor <church>} ", based_on = "[ <Notion1>, <Notion2> ", level = "Some 0" ] <
518
519
520
521

```



Fig. 7: Inherited Invariant Violation.

Any class definition generates term antiquotations checking a class instance reference in a particular logical context; these references were elaborated to objects they refer to. This paves the way for a new mechanism to query the “current” instances presented as a HOL *list*. Arbitrarily complex queries can therefore be defined inside the logical language. Thus, to get the list of the properties of the instances of the class *myresult*, or to get the list of the authors of the instances of the *myintro* class, it suffices to treat this meta-data as usual:



```

value*⟨map (myresult.property) @{myresult-instances}⟩
value*⟨map (mytext-section.authored-by) @{myintro-instances}⟩

```

In order to get the list of the instances of the class *myresult* whose *evidence* is a *proof*, one can use the command:

```

value*⟨filter (λσ. myresult.evidence σ = proof) @{myresult-instances}⟩

```

## 4 Proving Morphisms on Ontologies

The Isabelle/DOF framework does not assume that all documents refer to the same ontology. Each document may even build its local ontology without any external reference. It may also be based on several reference ontologies (e.g., from the Isabelle/DOF library). Since ontological instances possess *representations inside the logic*, the relationship between a local ontology and a reference ontology can be formalised using a morphism function also inside the logic. More precisely, the instances of local ontology classes may be described as the image of a transformation applied to one or several other instances of class(es) belonging to another ontology. Thanks to the morphism relationship, the obtained class may either import meta-data (definitions are preserved) or map meta-data (the properties are different but are semantically equivalent) that are defined in the referenced class(es). It may also provide additional properties. This means that morphisms may be injective, surjective, bijective, and thus describe abstract relations between ontologies. This raises the question of invariant preservation.

To illustrate this process, we have defined a simple ontology to classify Hardware objects.

<pre> <b>onto-class</b> Item =   name :: string <b>onto-class</b> Product = Item +   serial-number :: int   provider :: string   mass :: int <b>onto-class</b> Computer-Hardware = Product +   type :: Hardware-Type   composed-of :: Product list <b>invariant</b> c2 :: Product.mass σ = sum(map Product.mass (composed-of σ)) </pre>	Isabelle code
---	---------------

This ontology defines the *Item*, *Product* and *Computer-Hardware* concepts (or classes). Each class contains a set of attributes or properties and some local invariants. In this example, we focus on the *Computer-Hardware* class defined as a list of products characterised by their mass value. This class contains a local **invariant** *c2* to guarantee that its own mass value equals the sum of all the masses of the products composing the object. For the sake of the argument, we use the reference ontology (considered as a standard) described in this listing:

```
definition sum where sum S = (fold (+) S 0)
```

Isabelle code

```
datatype Hardware-Type = Motherboard | Expansion-Card | Storage-Device ...
```

```
onto-class Resource =
```

```
  name :: string
```

```
onto-class Electronic = Resource +
```

```
  provider :: string
```

```
  manufacturer :: string
```

```
onto-class Component = Electronic +
```

```
  mass :: int
```

```
  dimensions :: int list
```

```
onto-class Informatic = Resource +
```

```
  description :: string
```

```
onto-class Hardware = Informatic +
```

```
  type :: Hardware-Type
```

```
  mass :: int
```

```
  composed-of :: Component list
```

```
invariant c1 :: mass  $\sigma$  = sum(map Component.mass (composed-of  $\sigma$ ))
```

This ontology defines the *Resource*, *Electronic*, *Component*, *Informatic* and *Hardware* concepts. In our example, we focus on the *Hardware* class containing a *Component.mass* attribute inherited from the *Component* class and composed of a list of components with a *Component.mass* attribute formalising the mass value of each component. The *Hardware* class also contains a local **invariant** *c1* to define a constraint linking the global mass of a *Hardware* object with the masses of its own components.

To check the coherence of our local ontology, we define a relationship between the local ontology and the reference ontology using morphism functions (or mapping rules as in ATL framework [11] or EXPRESS-X language [2]). These rules are applied to define the relationship between one class of the local ontology to one or several other class(es) described in the reference ontology. In our case, we have define two morphisms, *Product-to-Component-morphism* and *Computer-Hardware-to-Hardware-morphism*, detailed in the following listing:

```
definition Product-to-Component-morphism ::
```

```
  Product  $\Rightarrow$  Component (- (Component)Product [1000]999)
```

```
where  $\sigma$  (Component)Product = ( $\{$  Resource.tag-attribute = 1::int ,
```

```
  Resource.name = name  $\sigma$  ,
```

```
  Electronic.provider = provider  $\sigma$  ,
```

```
  Electronic.manufacturer = "no manufacturer" ,
```

```
  Component.mass = mass  $\sigma$  ,
```

```
  Component.dimensions = [] )
```

Isabelle code

<b>definition</b> <i>Computer-Hardware-to-Hardware-morphism</i> :: $Computer\text{-}Hardware \Rightarrow Hardware$ $(- \langle Hardware \rangle_{ComputerHardware} [1000]999)$ <b>where</b> $\sigma \langle Hardware \rangle_{ComputerHardware} =$ $\{$ $Resource.tag\text{-}attribute = 2::int ,$ $Resource.name = name \sigma ,$ $Informatique.description = "no description" ,$ $Hardware.type = type \sigma ,$ $Hardware.mass = mass \sigma ,$ $Hardware.composed\text{-}of = map Product\text{-}to\text{-}Component\text{-}morphism$ $(composed\text{-}of \sigma) \}$	Isabelle code
---	---------------

These definitions specify how *Product* or *Computer-Hardware* objects are mapped to *Component* or *Hardware* objects defined in the reference ontology. This mapping shows that the structure of a (user) ontology may be arbitrarily different from the one of a standard ontology it references.

After defining the mapping rules, now we have to deal with the question of invariant preservation. The following example proofs for a simple but typical example of reformatting meta-data into another format along an ontological mapping are nearly trivial:

<b>lemma</b> <i>inv-c2-preserved</i> : $c2\text{-}inv \sigma \Longrightarrow c1\text{-}inv (\sigma \langle Hardware \rangle_{ComputerHardware})$ <b>unfolding</b> $c1\text{-}inv\text{-}def$ $c2\text{-}inv\text{-}def$ $Computer\text{-}Hardware\text{-}to\text{-}Hardware\text{-}morphism\text{-}def$ $Product\text{-}to\text{-}Component\text{-}morphism\text{-}def$ <b>by</b> ( <i>auto simp: comp-def</i> )  <b>lemma</b> <i>Computer-Hardware-to-Hardware-total</i> : $Computer\text{-}Hardware\text{-}to\text{-}Hardware\text{-}morphism \subseteq (\{X. c2\text{-}inv X\})$ $\subseteq (\{X::Hardware. c1\text{-}inv X\})$ <b>using</b> <i>inv-c2-preserved by auto</i>	Isabelle code
---	---------------

After unfolding the invariant and the morphism definitions, the preservation proof is automatic. The advantage of using the Isabelle/DOF framework compared to approaches like ATL or EXPRESS-X is the possibility of formally verifying the *mapping rules*, i. e., proving the preservation of invariants, as we have demonstrated in the previous example.

## 5 Application: CENELEC Ontology

From its beginning, Isabelle/DOF had been used for documents containing formal models targeting certifications. A major case-study from the railways domain based on the CENELEC 50128 standard had been published earlier (cf. [9])<sup>6</sup>.

<sup>6</sup> Our CENELEC ontology in Isabelle/DOF can be found at [https://github.com/logicallhacking/Isabelle\\_DOF/blob/main/src/ontologies/CENELEC\\_50128/CENELEC\\_50128.thy](https://github.com/logicallhacking/Isabelle_DOF/blob/main/src/ontologies/CENELEC_50128/CENELEC_50128.thy).

The CENELEC Standard comprises 18 different “Design and Test Documents”; a fully fledged description of our ontology covering these is therefore out of reach of this paper. Rather, we present how the novel concepts such as invariants and term-antiquotations are used in selected elements in this ontology.

According to CENELEC Table C.1, for example, we specify the category of “Design and Test Documents” as follows:

<pre> <b>doc-class</b> <i>cenelec-document</i> = <i>text-element</i> +   <i>phase</i>      :: <i>phase</i>   <i>written-by</i> :: <i>role</i>    — Annex C Table C.1   <i>fst-check</i>  :: <i>role</i>    — Annex C Table C.1   <i>snd-check</i>  :: <i>role</i>    — Annex C Table C.1   ...   <b>invariant</b> <i>must-be-chapter</i> :: <i>text-element.level</i> <math>\sigma = \text{Some}(0)</math>   <b>invariant</b> <i>two-eyes-prcple</i> :: <i>written-by</i> <math>\sigma \neq \text{fst-check } \sigma</math>                                    <math>\wedge</math> <i>written-by</i> <math>\sigma \neq \text{snd-check } \sigma</math> </pre>	<b>Isabelle code</b>
---	----------------------

This class refers to the “software phases” the standard postulates (like *SPI* for “Software Planning” or *SCDES* for “Software Component Design”) which we model by a corresponding enumeration types (not shown here). Similarly, the standard postulates “roles” that certain specified teams possess in the overall process (like *VER* for verification and *VAL* for validation). We added invariants that specify certain constraints implicit in the standard: for example, a *cenelec-document* must have the textual structure of a chapter (the *level*-attribute is inherited from an underlying ontology library specifying basic text-elements) as well as the two-eyes-principle between authors and checkers of these chapters.

The concrete sub-class of *cenelec-document* is the class *SWIS* (“software interface specification”) as shown below, which provides the role assignment required for this document type:

<pre> <b>doc-class</b> <i>SWIS</i> = <i>cenelec-document</i> + — software interface specification   <i>phase</i>      :: <i>phase</i> &lt;= <i>SCDES</i>   <i>written-by</i> :: <i>role</i> &lt;= <i>DES</i>   <i>fst-check</i>  :: <i>role</i> &lt;= <i>VER</i>     <i>snd-check</i>  :: <i>role</i> &lt;= <i>VAL</i>   <i>components</i>:: <i>SWIS-E list</i> </pre>	<b>Isabelle code</b>
--	----------------------

The structural constraints expressed so far can in principle be covered by any hand-coded validation process and suitable editing support (e. g., Protégé [21]). However, a closer look at the class *SWIS-E* (“software interface specification element”) referenced in the *components*-attribute reveals the unique power of Isabelle/DOF; rather than saying “there must be a pre-condition”, Isabelle/DOF can be far more precise:

<b>doc-class</b> <i>SWIS-E</i> =	<b>Isabelle code</b>
<i>op-name</i> :: <i>string</i>	
<i>op-args-res</i> :: ( <i>string</i> × <i>typ</i> ) <i>list</i> × <i>typ</i> — args and result type	
<i>pre-cond</i> :: ( <i>string</i> × <i>thm</i> ) <i>list</i> — labels and predicates	
<i>post-cond</i> :: ( <i>string</i> × <i>thm</i> ) <i>list</i> — labels and predicates	
<b>invariant</b> <i>well-formed-pre</i> :: $\forall cond \in set(map\ snd\ (pre\ cond\ \sigma)).$	
<i>iswff<sub>pre</sub></i> ( <i>op-args-res</i> $\sigma$ ) ( <i>cond</i> )	
<b>invariant</b> <i>well-formed-post</i> :: ...	

where the constant *iswff<sub>pre</sub>* is bound to a function at the SML-level, that is executed during the evaluation phase of these invariants and that checks:

- Any *cond* is indeed a valid definition in the global logical context (taking HOL-libraries but also the concrete certification target model into account).
- Any such HOL definition has the syntactic form:
 
$$pre\langle op\text{-}name \rangle (a_1::\tau_1) \dots (a_n::\tau_n) \equiv \langle predicate \rangle,$$
 where  $(a_1::\tau_1) \dots (a_n::\tau_n)$  correspond to the argument list.
- The case for the post-condition is treated analogously.

Note that this technique can also be applied to impose specific syntactic constraints on types. For example, via the SI-package available in the Isabelle AFP <sup>7</sup>, it is possible to express that the result of some calculation is of type *32 unsigned [m·s<sup>-2</sup>]*, so a 32-bit natural representing an acceleration in the SI-system. Therefore it is possible to impose that certain values refer to physical dimensions measured in a concrete metrological system.

## 6 Related Work

There are a number of approaches to use ontologies for structuring the link between information and knowledge, and to make it amenable to “semantic” search in or consistency checking of documents. Some are targeting mathematical libraries, like the search engine <http://shinh.org/wfs> which uses clever text-based search methods in a large number of formulas, agnostic of their logical context and of formal proof, or the OAF project [18] which developed a common ontological format, called OMDoc/MMT, and six *export* functions from major ITP systems into it. The more difficult task to develop import functions has not been addressed, not to mention the construction of imported proofs in a native tactic proof format. Rather, the emphasis was put on building a server infrastructure based on conventional, rather heavy-weight database and OWL technology. Our approach targets so far only one ITP system and its libraries, and emphasizes type-safeness, expressive power and “depth” of meta-data, which is adapted to the specific needs of ITP systems and theory developments.

There are also a number of proposals of ontologies targeting mathematics: the OntoMath<sup>PRO</sup> [22] proposes a “taxonomy of the fields of mathematics” (p.

<sup>7</sup> [https://www.isa-afp.org/entries/Physical\\_Quantities.html](https://www.isa-afp.org/entries/Physical_Quantities.html)

6). In total, `OntoMathPRO` contains the daunting number of 3,449 classes, which is in part due to the need to compensate the lack of general datatype definition methods for meta-data. It is nevertheless an interesting starting point for a future development of a mathematics ontology in the Isabelle/DOF framework. Other ontologies worth mentioning are DBpedia [3], which provides with the *SPARQL endpoint* <http://dbpedia.org/sparql> a search engine, and the more general ScienceWISE <sup>8</sup> that allows users to introduce their own category concepts. Both suffer from the lack of deeper meta-data modeling, and the latter is still at the beginning (ScienceWISE marks the Mathematics part as “under construction”).

Regarding the use of formal methods to formalise standards, the Event-B method was proposed by Fotso et al. [13] for specifications of the hybrid ERTMS/ETCS level 3 standard, in which requirements are specified using SysML/KAOS goal diagrams. The latter were translated into Event-B, where domain-specific properties were specified by ontologies. In another case, Mendil et al. [20] propose an Event-B framework for formalising standard conformance through formal modelling of standards as ontologies. The proposed approach was exemplified on ARINC 661 standard in the context of a weather radar system application. These works are essentially interested in expressing ontological concepts in a formal method but do not explicitly deal with the formalisation of rules/invariants defined in ontologies. The question of ontology-mappings is not addressed.

## 7 Conclusion and Future Work

We presented Isabelle/DOF, an ontology framework deeply integrating continuous-check/continuous-build functionality into the formal development process in HOL. The novel feature of term-contexts in Isabelle/DOF, which permits term-antiquotations elaborated in the parsing process, paves the way for the abstract specification of meta-data constraints as well the possibility of advanced search in the meta-data of document elements. Thus it profits and extends Isabelle’s document-centric view on formal development.

Many ontological languages such as OWL as well as the meta-modeling technology available for UML/OCL provide concepts for semantic rules and constraints, but leave the validation checking usually to external tools (if implementing them at all). This limits their practical usefulness drastically. Our approach treats invariants as first-class citizens, and turns them into an object of formal study in, for example, ontological mappings. Such a technology exists, to our knowledge, for the first time.

Our experiments with adaptations of existing ontologies from engineering and mathematics show that Isabelle/DOF’s ODL has sufficient expressive power to cover all aspects of languages such as OWL (with perhaps the exception of multiple inheritance on classes). However, these ontologies have been developed specifically *in* OWL and target its specific support, the Protégé editor [21]. We

---

<sup>8</sup> <http://sciencewise.info/ontology/>.

argue that Isabelle/DOF might ask for a re-engineering of these ontologies: less deep hierarchies, rather deeper structure in meta-data and stronger invariants.

We plan to complement Isabelle/DOF with incremental LaTeX generation and a previewing facility that will further increase the usability of our framework for the ontology-conform editing of formal content, be it in the engineering or the mathematics domain (this paper has been edited in Isabelle/DOF, of course).

Another line of future application is to increase the “depth” of built-in term antiquotations such as  $\text{@}\{\text{typ } \langle \tau \rangle\}$ ,  $\text{@}\{\text{term } \langle a + b \rangle\}$  and  $\text{@}\{\text{thm } \langle \text{HOL.refl} \rangle\}$ , which are currently implemented just as validations in the current logical context. In the future, they could optionally be expanded to the types, terms and theorems (with proof objects attached) in a meta-model of the Isabelle Kernel such as the one presented in [24] (also available in the AFP). This will allow for definitions of query-functions in, e. g., proof-objects, and pave the way to annotate them with typed meta-data. Such a technology could be relevant for the interoperability of proofs across different ITP platforms.

## References

1. Aehlig, K., Haftmann, F., Nipkow, T.: A compiled implementation of normalisation by evaluation. *J. Funct. Program.* **22**(1), 9–30 (2012). <https://doi.org/10.1017/S0956796812000019>, <https://doi.org/10.1017/S0956796812000019>
2. Ameur, Y.A., Besnard, F., Girard, P., Pierra, G., Potier, J.: Formal specification and metaprogramming in the EXPRESS language. In: SEKE’95, The 7th International Conference on Software Engineering and Knowledge Engineering, June 22–24, 1995, Rockville, Maryland, USA, Proceedings. pp. 181–188. Knowledge Systems Institute (1995)
3. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A nucleus for a web of open data. In: Aberer, K., Choi, K.S., Noy, N., Allemang, D., Lee, K.I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) *The Semantic Web*. pp. 722–735. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
4. 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). [https://doi.org/10.1007/978-3-319-96812-4\\_3](https://doi.org/10.1007/978-3-319-96812-4_3), <https://www.brucker.ch/bibliography/abstract/brucker.ea-isabelle-ontologies-2018>
5. Brucker, A.D., Brügger, L., Wolff, B.: Formal network models and their application to firewall policies. *Archive of Formal Proofs* (Jan 2017), <http://www.brucker.ch/bibliography/abstract/brucker.ea-upf-firewall-2017>, [http://www.isa-afp.org/entries/UPF\\_Firewall.shtml](http://www.isa-afp.org/entries/UPF_Firewall.shtml), Formal proof development
6. Brucker, A.D., Herzberg, M.: The Core DOM. *Archive of Formal Proofs* (Dec 2018), <http://www.brucker.ch/bibliography/abstract/brucker.ea-afp-core-dom-2018>, [http://www.isa-afp.org/entries/Core\\_DOM.html](http://www.isa-afp.org/entries/Core_DOM.html), Formal proof development
7. Brucker, A.D., Tuong, F., Wolff, B.: Featherweight OCL: A proposal for a machine-checked formal semantics for OCL 2.5. *Archive of Formal Proofs* (Jan 2014), <http://www.brucker.ch/bibliography/abstract/brucker.ea-featherweight-2014>, [http://www.isa-afp.org/entries/Featherweight\\_OCL.shtml](http://www.isa-afp.org/entries/Featherweight_OCL.shtml), Formal proof development

8. 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). [https://doi.org/10.1007/978-3-030-30446-1\\_15](https://doi.org/10.1007/978-3-030-30446-1_15), <https://www.brucker.ch/bibliography/abstract/brucker.ea-isabelledof-2019>
9. 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, Bergen, Norway, December 2-6, 2019, Proceedings*. *Lecture Notes in Computer Science*, vol. 11918, pp. 65–82. Springer (2019). [https://doi.org/10.1007/978-3-030-34968-4\\_4](https://doi.org/10.1007/978-3-030-34968-4_4), [https://doi.org/10.1007/978-3-030-34968-4\\_4](https://doi.org/10.1007/978-3-030-34968-4_4)
10. Bs en 50128:2011: Railway applications – communication, signalling and processing systems – software for railway control and protecting systems. Standard, British Standards Institute (BSI) (Apr 2014)
11. Eclipse Foundation: Atl - a model transformation technology, <https://www.eclipse.org/atl/>, Accessed: 2022-03-15
12. Euzenat, J., Shvaiko, P.: *Ontology Matching*, Second Edition. Springer (2013). <https://doi.org/10.1007/978-3-642-38721-0>
13. Fotso, S.J.T., Frappier, M., Laleau, R., Mammar, A.: Modeling the hybrid ERTM-S/ETCS level 3 standard using a formal requirements engineering approach. In: *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 6th International Conference, ABZ, Southampton, UK. LNCS*, vol. 10817, pp. 262–276. Springer (2018). [https://doi.org/10.1007/978-3-319-91271-4\\_18](https://doi.org/10.1007/978-3-319-91271-4_18)
14. Haftmann, F., Nipkow, T.: Code generation via higher-order rewrite systems. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) *Functional and Logic Programming, 10th International Symposium, FLOPS 2010, Sendai, Japan, April 19-21, 2010. Proceedings*. *Lecture Notes in Computer Science*, vol. 6009, pp. 103–117. Springer (2010). [https://doi.org/10.1007/978-3-642-12251-4\\_9](https://doi.org/10.1007/978-3-642-12251-4_9), [https://doi.org/10.1007/978-3-642-12251-4\\_9](https://doi.org/10.1007/978-3-642-12251-4_9)
15. Hou, Z., Sanan, D., Tiu, A., Liu, Y.: A formal model for the sparcv8 isa and a proof of non-interference for the leon3 processor. *Archive of Formal Proofs* (Oct 2016), <http://isa-afp.org/entries/SPARCV8.html>, Formal proof development
16. Hupel, L., Zhang, Y.: Cakeml. *Archive of Formal Proofs* (Mar 2018), <http://isa-afp.org/entries/CakeML.html>, Formal proof development
17. 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). <https://doi.org/10.1145/2560537>
18. Kohlhase, M., Rabe, F.: Experiences from exporting major proof assistant libraries. *J. Autom. Reason.* **65**(8), 1265–1298 (2021). <https://doi.org/10.1007/s10817-021-09604-0>, <https://doi.org/10.1007/s10817-021-09604-0>
19. M.Eberl and G. Klein and A. Lochbihler and T. Nipkow and L. Paulson and R. Thiemann (eds): *Archive of Formal Proofs*. <https://afp-isa.org> (2022), Accessed: 2022-03-15
20. Mendil, I., Ait-Ameur, Y., Singh, N.K., Méry, D., Palanque, P.A.: Standard conformance-by-construction with event-b. In: *Formal Methods for Industrial Critical Systems - 26th International Conference, FMICS, Paris, France. LNCS*, vol. 12863, pp. 126–146. Springer (2021). [https://doi.org/10.1007/978-3-030-85248-1\\_8](https://doi.org/10.1007/978-3-030-85248-1_8)
21. Musen, M.A.: The protégé project: A look back and a look forward. *AI Matters* **1**(4), 4–12 (jun 2015). <https://doi.org/10.1145/2757001.2757003>, <https://doi.org/10.1145/2757001.2757003>



22. Nevzorova, O., Zhiltsov, N., Kirillovich, A., Lipachev, E.K.: OntoMath<sup>PRO</sup> ontology: A linked data hub for mathematics. ArXiv **abs/1407.4833** (2014). [https://doi.org/10.1007/978-3-319-11716-4\\_9](https://doi.org/10.1007/978-3-319-11716-4_9)
23. Nipkow, T.: Splay tree. Archive of Formal Proofs (Aug 2014), [http://isa-afp.org/entries/Splay\\_Tree.html](http://isa-afp.org/entries/Splay_Tree.html), Formal proof development
24. Nipkow, T., Roßkopf, S.: Isabelle’s metalogic: Formalization and proof checker. In: Platzer, A., Sutcliffe, G. (eds.) Automated Deduction – CADE 28. pp. 93–110. Springer International Publishing, Cham (2021)
25. Sengupta, K., Hitzler, P.: Web ontology language (OWL). In: Encyclopedia of Social Network Analysis and Mining, pp. 2374–2378 (2014). [https://doi.org/10.1007/978-1-4614-6170-8\\_113](https://doi.org/10.1007/978-1-4614-6170-8_113)
26. Sprenger, C., Somaini, I.: Developing security protocols by refinement. Archive of Formal Proofs (May 2017), [http://isa-afp.org/entries/Security\\_Protocol\\_Refinement.html](http://isa-afp.org/entries/Security_Protocol_Refinement.html), Formal proof development
27. Verbeek, F., Tverdyshev, S., Havle, O., Blasum, H., Langenstein, B., Stephan, W., Nemouchi, Y., Feliachi, A., Wolff, B., Schmaltz, J.: Formal specification of a generic separation kernel. Archive of Formal Proofs (Jul 2014), <http://isa-afp.org/entries/CISC-Kernel.html>, Formal proof development
28. Wenzel, M.: The Isabelle/Isar Reference Manual (2020), part of the Isabelle distribution.