

Using Deep Ontologies in Formal Software Engineering

Achim D. Brucker¹[0000-0002-6355-1200], Idir Ait-Sadoune²[0000-0002-6484-8276],
Nicolas Méric³[0000-0002-0756-7072], and Burkhardt Wolff⁴

¹ The University of Exeter, Exeter, UK
a.brucker@exeter.ac.uk

² Université Paris-Saclay, CentraleSupélec, LMF, France
idir.aitsadoune@centralesupelec.fr

³ Université Paris-Saclay, LMF, France
nicolas.meric@universite-paris-saclay.fr

⁴ Université Paris-Saclay, LMF, France
burkhart.wolff@universite-paris-saclay.fr

Abstract. Isabelle/DOF is an ontology framework on top of Isabelle. Isabelle/DOF allows for the formal development of ontologies as well as continuous conformity-checking of integrated documents annotated by ontological data. An integrated document may contain text, code, definitions, proofs and user-programmed constructs supporting a wide range of Formal Methods. Isabelle/DOF is designed to leverage traceability in integrated documents by supporting navigation in Isabelle’s IDE as well as the document generation process.

In this paper we extend Isabelle/DOF with annotations of λ -terms, a pervasive data-structure underlying Isabelle used to syntactically represent expressions and formulas. Rather than introducing an own programming language for meta-data, we use Higher-order Logic (HOL) for expressions, data-constraints, ontological *invariants*, and queries via code-generation and reflection. This allows both for powerful query languages and logical reasoning over ontologies in, for example, ontological mappings. Our application examples cover documents targeting formal certifications such as CENELEC, Common Criteria, etc.

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 modern society. Extracting knowledge from reasonably well-structured informal “raw”-texts is a crucial prerequisite for any form of advanced search, classification, “semantic” validation and “semantic” merge technology. This challenge incites numerous research efforts summarized under the labels “semantic web” or “data mining”. A key role in structuring this

linking are played by document ontologies (also called vocabulary in the semantic networks or semantic web communities), i.e., a machine-readable form of the structure of documents as well as the document discourse. Such ontologies can be used for the scientific discourse underlying scholarly articles, mathematical libraries, and documentations in various engineering domains. In other words, ontologies generate the meta-data necessary to annotate raw text allowing their “deeper analysis”, in particular if mathematical formulas or other forms of formal content occur.

We are in particular interested in a particular application domain of these techniques, namely integrated documentations of software developments targeting certifications (such as CENELEC 50128 [6] or Common Criteria [7]). We consider this domain as a particular rewarding instance of the general problem. Certifications of safety or security critical systems, albeit responding to the fundamental need of the modern society of trustworthy numerical infrastructures, are particularly complex and expensive, since distributed labor as occurring in the industrial practice involving numerous artifacts such as analysis, design, and verification documents including models and code must be kept coherent under permanent changes during the development. Moreover, certification processes impose a strong need of traceability within the global document structure. Last but not least, modifications and updates of a certified product usually result in a complete restart of the certification activity, since the impact of local changes can usually not be mechanically checked and has to be done essentially by manual inspection. Our interest in this domain lead us to the development of Isabelle/DOF, an environment implementing our concept of *deep ontology*.

1.1 A Gentle Introduction into Isabelle/DOF

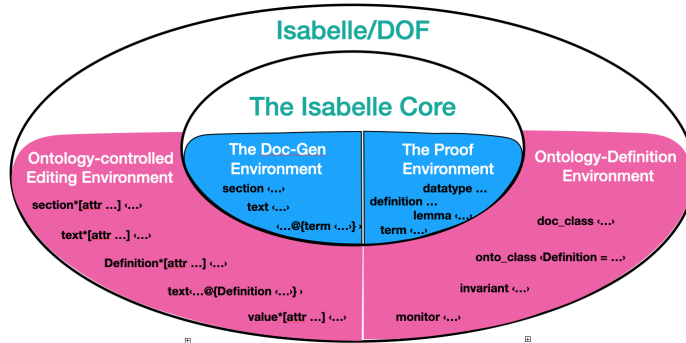


Fig. 1: The Ontology Environment Isabelle/DOF.

Isabelle/HOL [19] is a well-known semi-automated proof environment and documentation generator. As shown in Fig. 1, Isabelle/DOF extends the open-source Isabelle/HOL core by a number of constructs allowing for the specification

of formal ontologies (left half); additionally, it provides documentation constructs (right half) for text-, definition-, term-, proof-, code-, and user-defined elements that enforce document conformance to a given ontology.

Isabelle/DOF [4]⁵ is a new kind of ontological modelling and document validation tool. In contrast to conventional languages like OWL and development environments such as Protégé [17], it brings forward our concept of *deep ontologies*, i. e. ontologies represented inside a logical language such as HOL rather than a conventional programming language like Java. Deep ontologies generate strongly typed meta-information specified in HOL-theories allowing both for efficient execution **and** logical reasoning about meta-data. They generate a particular form of checked annotations called *antiquotation* to be used inside code and texts. Deeply integrated into the Isabelle ecosystem [5], and thus permitting continuous checking and validation, they also allow ontology-aware navigation inside large documents with both formal and informal content.

In the following, we will detail this by example of annotated text in a document. We will assume a given ontology; an introduction into our ontology definition language ODL is given in Sect. 2.2. The Isabelle’s `text`⟨ ... ⟩-element or `ML`⟨ ... ⟩ code-elements are extended to the corresponding Isabelle/DOF elements:

```
text*[label::cid, attrib-def1,...,attrib-defn]⟨... annotated text ... ⟩
ML*[label::cid, attrib-def1,...,attrib-defn]⟨... annotated code ... ⟩
```

where *cid* is an identifier of an ontological class introduced in an ontology together with attributes belonging to this class defined in ODL. For example, if an ontology provides a concept *Definition*, we can do the following:

```
text*[safe::Definition, name=safety]⟨Our system is safe if the following holds ...⟩
```

The Isabelle/DOF command `text*` creates an instance *safe* of the ontological class *Definition* with the attribute *name* and associates it to the text inside the ⟨...⟩-brackets. We call the content of these brackets the *text-context* (or *ML-context*, respectively). Of particular interest for this paper is the ability to generate a kind of semantic macro, called anti-quotation, which is continuously checked and whose evaluation uses information from the background theory of this text element.

For example, we might refer to the above definition in another text element:

```
text*[...]⟨As stated in @{Definition <safe>}, . . . ⟩
```

Where Isabelle/DOF checks on-the-fly that the reference “safe” is indeed defined in the document and has the right type (it is not an *Example*, for example), generates navigation information (i.e. hyperlinks to *safe* as well as the ontological description of *Definition* in the Isabelle IDE) as well as specific documentation markup in the generated PDF document, e.g.:

⁵ The official releases are available at <https://zenodo.org/record/6385695>, the developer version at https://github.com/logicalhacking/Isabelle_DOF.

As stated in Def. 3.11 (*safety*), ...

where the underline may be blue because the layout description configured for this ontology says so. Moreover, this is used to generate an index containing, for example, all definitions. Similarly, this also works for an ontology providing concepts such as “objectives”, “claims” and “evidences”, and invariants may be stated in an ontological class that finally enforces properties such as that “all claims correspond to evidences in the global document”, and “all evidences must contain at least one proven theorem”, etc. pp. In contrast to a conventional type-setting system, Isabelle can additionally type-check formulas, so for example:

```
text*[...]<The safety distance is defined by @{term distsafety = sqrt(d-a*Δt2)}...>
```

where functions like $dist_{safety}$, $sqrt$, $-*$, etc., have to be defined in the signature and logical context or background theory of this formula. Anti-quotations as such are not a new concept in Isabelle; the system comes with a couple of hand-programmed anti-quotations like $@\{term \dots\}$. In contrast, Isabelle/DOF generates anti-quotations from ontological classes in ODL, together with checks generated from data-constraints (or: class invariants) specified in HOL.

1.2 The novelty: Using HOL-terms for Meta-Data and Invariants

Isabelle uses typed λ -terms as syntactic presentation for expressions, formulas, definition and rules. Rather than using a classical programming language, our concept of deep ontologies led us to use HOL itself and generate the checking-code for anti-quotations via reflection and reification techniques. In particular, this paves the way for a new type context called *term contexts*. As a consequence, we extend Isabelle/DOF framework to use this possibility and will show in this paper how to exploit term contexts to express meta-data-constraints via *invariants*, to formally prove relations between instances and to generate code on-the-fly for advanced queries.

2 Background

2.1 The Isabelle/DOF Framework

As shown in Fig. 1, Isabelle/DOF extends Isabelle/HOL by basically two things: ways to *annotate* an integrated document written in Isabelle/HOL with the specified meta-data and a language called Ontology Definition Language (ODL) allowing to *specify* a formal ontology. Isabelle/DOF generates from an ODL 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 document source. This includes rich editing support, including on-the-fly semantics checks, hinting, or auto-completion. Isabelle/DOF supports

L^AT_EX-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 [6] is captured in ODL as follows:

```

doc-class text-element = . . .
datatype role = developer | verifier | validator
doc-class requirement = text-element +
    long-name ::string option
    is-concerned::role set
Isabelle (Isar)
```

Ontology specifications consist of a sequence of class definitions like these; here, they are intertwined with the standard Isabelle/HOL **datatype** command defining the constructors and the rules for *role*-type. Therefore, it can be referenced in the *requirement* **doc-class**. Note that Isabelle’s session management allows for pre-compiling them before being imported in another document being the instance of this ontology.

```

text*[req1::requirement,
  is_concerned="{developer, validator}"] text<
<The operating system shall provide secure memory separation.>
  is @{requirement <req1>} ...>
```

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

Fig. 2: Referencing a Requirement.

Fig. 2 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 just a click away.

Isabelle/DOF is based on the idea of “deep ontologies”. In this context, this means that a logical representation for the instance *req1* is generated, i.e. a λ -term, which is used to *represent* this meta-data. For this purpose, we use Isabelle/HOL’s record support [22].

For the above example, this means that *req1* is represented by :

- the record term $(\langle \text{long-name} = \text{None}, \text{is-concerned} = \{\text{developer}, \text{validator}\} \rangle)$ and the corresponding record type $\text{requirement} = (\langle \text{long-name}::\text{string option}, \text{is-concerned}::\text{role set} \rangle)$,
- ... while the resulting selectors were written *long-name r*, *is-concerned r*.

In general, **onto-classes** and the logically equivalent **doc-classes** were represented by *extensible* record types and instances thereof by HOL terms (see [5] for details.).

2.3 Term-Evaluations in Isabelle

Besides the powerful, but relatively slow rewriting-based proof method *simp*, there are basically two other techniques for the evaluation of terms:

- evaluation via reflection into SML [12] (*eval*), and
- normalization by evaluation [1] (*nbe*).

The former is based on a nearly one-to-one compilation of datatype specification constructs and recursive function definitions into SML datatypes and functions. The generated code is directly compiled by the underlying SML compiler of the Isabelle platform. This way, pattern-matching becomes natively compiled rather than interpreted as in the matching process of *simp*. Aehlig et al [1] are reporting on scenarios where *eval* is five orders of magnitude faster than *simp*. However, it is restricted to ground terms. *nbe* is not restricted to ground terms, but lies in its efficiency between *simp* and *eval*.

Isabelle/DOF uses a combination of these three techniques in order to evaluate invariants and check data-integrity on the fly during editing. For reasonably designed background theories and ontologies, this form of runtime-testing is sufficiently fast to remain unnoticed by the user.

3 Term-Context Support, Invariants and Queries in DOF

Isabelle/HOL as a system offers a document-centric view to the *formal* theory development process. Over the years, this led to strong documentation generation mechanisms supported by a list of build-in text and code anti-quotations.

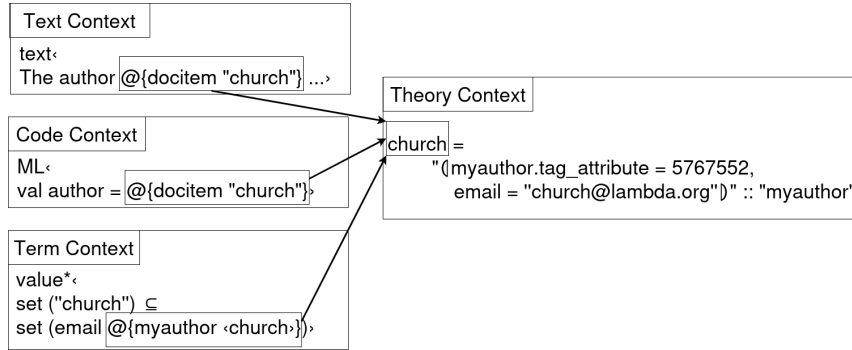


Fig. 3: Contexts in Isabelle/DOF.

As mentioned earlier, Isabelle/DOF generates from ODL families of ontology-related anti-quotations used in text and code contexts [4,5]. In this section, we introduce the novel concept of *term contexts*, i. e. annotations to be made inside λ -terms (See Fig. 3). Terms comprising term anti-quotations were treated by a refined process involving the steps:

- *Parsing* and *Typechecking* of the term in HOL theory context,
- Ontological *validation* of the term:
 - the arguments of term anti-quotations are parsed and checked,
 - checks resulting from ontological invariants were applied,
- *Generation of markup information* for the navigation in the IDE,
- *Elaboration* of term anti-quotations: depending of the antiquotation specific elaboration function, the anti-quotations containing references were replaced by the object they refer to, and
- *Evaluation*: HOL expressions were compiled and the result executed.

In order to exemplify this process, we consider the Isabelle/DOF commands **term*** and **value*** (which replace the traditional commands **term** and **value** restricted to parsing and type-checking).

```
term* < @{\thm "HOL.refl"} = @{\thm "HOL.sym"} >
value* < @{\thm "HOL.refl"} = @{\thm "HOL.sym"} >
```

Isabelle (Isar)

Here, **term*** parses and type-checks this λ -term as usual; logically, the $@\{thm\ "HOL.refl"\}$ is predefined by Isabelle/DOF as a constant *ISA-thm*. The validation will check that the string "HOL.refl" is indeed a reference to the theorem in the HOL-library, notably the reflexivity axiom. The type-checking of **term*** will infer *bool* for this expression. Now, **value*** will replace it by a constant representing a symbolic reference to a theorem; code-evaluation will compute *False* for this command. Note that this represents a kind of referential equality, not a "very deep" ontological look into the proof objects (in our standard configuration of Isabelle/DOF). Further note that there is a variant of **value***, called **assert***, which additionally checks that the term-evaluation results in *True*.

In Fig. 4, we present the running example for this section. Note that it is an extract from the ontology of [5], which could be used for writing certification documents.

```

datatype kind = expert-opinion | argument | proof
doc-class Author =
  email :: string <= ''''
doc-class Text-section =
  authored-by :: Author set <= {}
  level :: int option <= None
doc-class Intro = Text-section +
  authored-by :: Author set <= UNIV
  uses :: string set
invariant author-set :: authored-by  $\sigma \neq \{\}$ 
and force-level :: the (level  $\sigma$ ) > 1
doc-class Claim = Intro +
  based-on :: string list
doc-class Result = Text-section +
  formal-results :: thm list
  evidence :: kind
  property :: thm list <= []
invariant has-property :: evidence  $\sigma = \text{proof} \leftrightarrow$  property  $\sigma \neq []$ 
doc-class Conclusion = Text-section +
  establish :: (Claim  $\times$  Result) set
invariant establish-defined ::  $\forall x. x \in \text{Domain} (\text{establish } \sigma)$ 
   $\rightarrow (\exists y \in \text{Range} (\text{establish } \sigma). (x, y) \in \text{establish } \sigma)$ 

```

Fig. 4: Excerpt of an Example Ontology for mathematical Papers.

some class instances can be defined with the **text*** command, as in Fig. 5.

```

text*[church::Author, email=<church@lambda.org>]⟨⟩
text*[proof1::Result, evidence=proof, property=[@{thm <HOL.ref>}]]⟨⟩
text*[proof2::Result, evidence=proof, property=[@{thm <HOL.sym>}]]⟨⟩
text*[intro1::Intro, authored-by={@{Author <church>}}, level=Some 0]⟨⟩
text*[intro2::Intro, authored-by={@{Author <church>}}, level=Some 2]⟨⟩
text*[claimNotion::Claim, authored-by={@{Author <church>}},
  , based-on=[<Notion1>, <Notion2>], level=Some 0]⟨⟩

```

Fig. 5: Some Instances of the Classes of the Ontology of Fig. 4.

In the instance *intro1*, the term antiquotation $\@{\textit{Author} \langle \textit{church} \rangle}$, or its equivalent notation $\@{\textit{Author} \textit{''church''}}$, denotes the instance *church* of the class *Author*, where *church* is a HOL-string. One can now reference a class instance in a **term*** command. In the command **term*** $\langle \@{\textit{Author} \langle \textit{church} \rangle} \rangle$ the term $\@{\textit{Author} \langle \textit{church} \rangle}$ is type-checked, i. e., the command **term*** checks that *church* references a term of type *Author* against the global context (see Fig. 6).

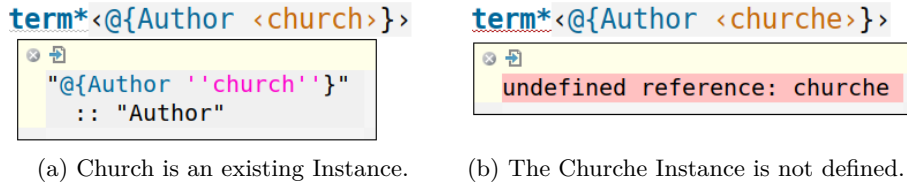


Fig. 6: Type-Checking of Antiquotations in a Term-Context.

The command **value*** $\langle \textit{email} \@{\textit{Author} \langle \textit{church} \rangle} \rangle$ validates $\@{\textit{Author} \langle \textit{church} \rangle}$ and returns the attribute-value of *email* for the *church* instance, i. e. the HOL-string *''church@lambda.org''* (see Fig. 7).

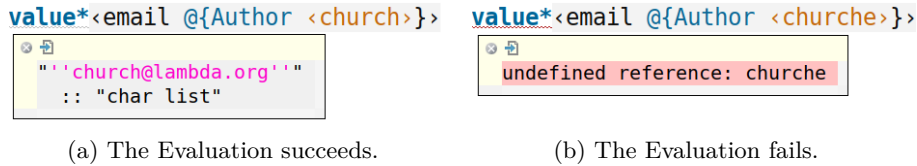


Fig. 7: Evaluation of Antiquotations in a Term-Context.

Since term antiquotations are basically logically uninterpreted constants, it is possible to compare class instances logically. The assertion in the Fig. 8 fails: the class instances *proof1* and *proof2* are not equivalent because their attribute *property* differs. When **assert*** evaluates the term, the term antiquotations $\@{\textit{thm} \langle \textit{HOL.refl} \rangle}$ and $\@{\textit{thm} \langle \textit{HOL.sym} \rangle}$ are checked against the global context such that the strings $\langle \textit{HOL.refl} \rangle$ and $\langle \textit{HOL.sym} \rangle$ 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. 4). Following the constraints proposed in [4], one can specify that any instance of a class *Result* finally has a non-empty property list, if its *kind* is *proof* (see the **invariant has-property**), or that the relation between *Claim* and *Result* expressed in

```
assert*{@Result <proof1>} = {@Result <proof2>}>
```

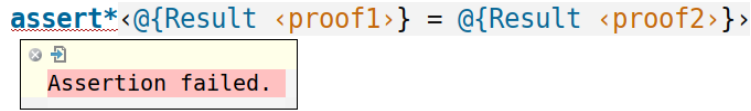


Fig. 8: Evaluation of the Equivalence of two Class Instances.

the attribute *establish* must be defined when an instance of the class *Conclusion* is defined (see the **invariant** *establish-defined*).

In Fig. 4, the **invariant** *author-set* of the class *Intro* enforces that a *Intro* instance has at least one author. The σ symbol is reserved and references the future class instance. By relying on the implementation of extensible records in Isabelle/HOL [22], one can reference an attribute of an instance using its selector function. For example, *establish* σ denotes the value of the attribute *establish* of the future instance of the class *Conclusion*.

The value of each attribute defined for the instances is checked at run-time against their class invariants. Recall that Classes also inherit the invariants from their super-classes. As the class *Claim* is a subclass of the class *Intro*, it inherits the *Intro* invariants. In Fig. 9, we attempt to specify a new instance *claimNotion* of this class. However, the invariant checking triggers an error because the **invariant** *force-level* forces the value of the argument of the attribute *Text-section.level* to be greater than 1, and we initialize it to *Some 0* in *claimNotion*.

```
text*[claimNotion::Claim, authored_by = "@{Author <church>}", level = "Some 0"]<>
```

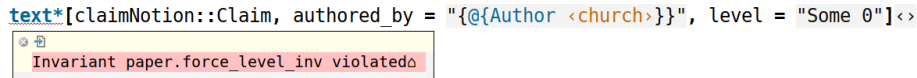


Fig. 9: Inherited Invariant Violation.

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*. Using functions defined in HOL, arbitrarily complex queries can therefore be defined inside the logical language. Thus, to get the property list of the instances of class *Result*, it suffices to process this meta-data via mapping the *property* selector over the *Result* class:

```
value*⟨map (Result.property) @{Result-instances}⟩
```

Isabelle (Isar)

Analogously we can define an arbitrary filter function, for example the HOL *filter* definition on lists:

```

fun filter:: ('a ⇒ bool) ⇒ 'a list ⇒ 'a list
  where filter P [] = []
    | filter P (x # xs) = (if P x then x # filter P xs else filter P xs)

```

Isabelle (Isar)

to get the list of the instances of the class *Result* whose *evidence* is a *proof*:

```

value* filter (λσ. Result.evidence σ = proof) @{Result-instance}

```

Isabelle (Isar)

With Isabelle/DOF comes the concept of monitor classes [5], which are classes that may refer to other classes via a regular expression in an *accepts* clause. Semantically, monitors introduce a behavioral element into ODL and to enforce the structure in a document. Monitors generate traces about a part of a document, recorded in the *trace* attribute of the monitor, and also presented as a *list of string*. For this monitor specification:

```

doc-class doc-monitor =
  ok :: unit
  accepts [Intro] ~~ {Claim}* ~~ [Result]

```

Isabelle (Isar)

... one can define an *is-in* function in HOL to check the trace of a document fragment against a regular expression:

```

definition example-expression
  where example-expression ≡ {["Intro"] || ["Claim"] || ["Result"]}
value* < (map fst @{trace-attribute "monitor1"}) is-in example-expression >

```

Isabelle (Isar)

Here, the term anti-quotation $@\{trace-attribute\ "monitor1"\}$ denotes the instance trace of *monitor1*. It is checked against the regular expression *example-expression*. Actually, *example-expression* is compiled via an implementation of the Functional-Automata of the AFP [18] into a deterministic automaton. On the latter, the above acceptance test is still reasonably fast.

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). Making a relationship between a local ontology and reference ontologies is a way to show that the built document referencing a local ontology is not far away from a domain reference ontology.

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 specified also inside the logic. More precisely, the instances of local ontology classes may be mapped to one or several other instances belonging to another ontology. Thanks to the morphism relationship, the obtained instances may either be an equivalent representations or abstractions

of the original ones. 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 define a simple ontology to classify monitors.

```

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

```

Isabelle (Isar)

```

onto-class Item =
  name :: string
onto-class Product = Item +
  serial-number :: int
  provider :: string
  mass :: int
onto-class Electronic-Component = Product +
  serial-number :: int
onto-class Monitor = Product +
  composed-of :: Electronic-Component list
invariant c2 :: Product.mass  $\sigma$  = sum(map Product.mass (composed-of  $\sigma$ ))

```

This ontology defines the *Item*, *Product* and *Monitor* concepts. Each class contains a set of attributes or properties and some local invariants. In this example, we focus on the *Monitor* 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 equals the sum of all masses of its components. For the sake of the argument, we use the reference ontology described as follows:

```

datatype Hardware-Type = Ouput-Device | Motherboard | Expansion-Card

```

Isabelle (Isar)

```

onto-class Resource =
  name :: string
onto-class Electronic = Resource +
  provider :: string
  manufacturer :: string

onto-class Component = Electronic +
  mass :: int

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 *mass* attribute inherited from the *Component* class and composed of a

list of components with a *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 [9] 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, *Electronic-Component-to-Component-morphism* and *Monitor-to-Hardware-morphism*, detailed in the following listing:

```

definition Electronic-Component-to-Component-morphism Isabelle (Isar)
Electronic-Component  $\Rightarrow$  Component
  (-  $\langle$ Component $\rangle_{ElecCmp}$  [1000]999)
  where  $\sigma$   $\langle$ Component $\rangle_{ElecCmp}$  =
    (| Resource.tag-attribute = 4::int ,
      Resource.name = name  $\sigma$  ,
      Electronic.provider = provider  $\sigma$  ,
      Electronic.manufacturer = "no manufacturer" ,
      Component.mass = mass  $\sigma$  |)

definition Monitor-to-Hardware-morphism :: Monitor  $\Rightarrow$  Hardware
  (-  $\langle$ Hardware $\rangle_{ComputerHardware}$  [1000]999)
  where  $\sigma$   $\langle$ Hardware $\rangle_{ComputerHardware}$  =
    (| Resource.tag-attribute = 5::int ,
      Resource.name = name  $\sigma$  ,
      Informatic.description = "no description" ,
      Hardware.type = Output-Device ,
      Hardware.mass = mass  $\sigma$  ,
      Hardware.composed-of = map Electronic-Component-to-Component-morphism (composed-of  $\sigma$ )
    |)

```

These definitions specify how *Electronic-Component* or *Monitor* 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.

Actually, we implemented a high-level syntax for this:

onto-morphism (*Computer-Hardware*) **to** *Hardware* ..

where the ".." stands for a standard proof attempt consisting of unfolding the invariant predicates and a standard auto proof. With this syntax, we can actually cover more general cases such as :

onto-morphism (A_1, \dots, A_n) **to** X_i **and** (D_1, \dots, D_m) **to** Y_j

were tuples of instances belonging to classes (A_1, \dots, A_n) can be mapped to instances of another ontology.

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:

<p>lemma <i>inv-c2-preserved</i> :</p> <p><i>c2-inv</i> $\sigma \implies c1\text{-inv}(\sigma \langle \text{Hardware} \rangle_{\text{ComputerHardware}})$</p> <p>unfolding <i>c1-inv-def c2-inv-def</i></p> <p style="padding-left: 2em;"><i>Computer-Hardware-to-Hardware-morphism-def</i></p> <p style="padding-left: 2em;"><i>Product-to-Component-morphism-def</i></p> <p>by (<i>auto simp: comp-def</i>)</p>	Isabelle (Isar)
---	------------------------

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 Related Work

In this paper, we already mentioned conventional ontology modeling languages like OWL; these systems possess development environments such as Protégé [17] which allow the documentation generation and ontology-based queries in structured texts. The platform allows also the integration of plug-ins that provide Prolog-like reasoners over class invariants in a description logics or fragments of first-order logic. In contrast to OWL, Isabelle/DOF brings forward our concept of *deep ontologies*, i. e. ontologies represented inside an extensible and expressive language such as HOL. Deep ontologies also allow to use meta-logical entities such as types, terms and theorems, and provide via anti-quotations means to reference *inside* them. The purpose is to establish strong, machine-checkable links between formal and informal content.

Isabelle/DOF's underlying ontology definition language ODL has many similarities with F-Logic [13] and its successors Flora-2 and ObjectLogic⁶. Shared features include object identity, complex objects, inheritance, polymorphic types, query methods, and encapsulation principles. Motivated by the desire for set-theories in modeling, F-Logic possesses syntax for some higher-order constructs but bases itself on first-order logics as foundation; this choice limits the potential for user-defined data-type definitions and proofs over classes significantly. Originally designed for object-oriented databases, F-Logic and its successors became mostly used in the area of the *Semantic Web* (a.k.a. Web 3.0). In contrast, Isabelle/DOF represents an intermediate layer between a logic like HOL and its implementing languages like SML or OCaml (having their roots as meta-language for these systems). This "in-between" allows for both executability and logical reasoning over meta-data generated to annotate formal terms and texts.

⁶ ... with *OntoStudio* as a commercial ObjectLogic implementation.

While F-Logic and its successors have similar design objectives, Isabelle/DOF is tuned towards systems with a document-centric view on code and semi-formal text as is prevailing in proof-assistants. Not limited to, but currently mostly used as *document*-ontology framework, it has similarity with other documentation generation systems such as Javadoc [21,8], Doxygen or ocamldoc [3](chap. 19). These systems are usually external tools run in batch-mode over the sources with a fixed set of structured comments similar to Isabelle/DOF’s antiquotations. In contrast, our approach foresees freely user-definable anti-quotations, which are in the case of references automatically generated. Furthermore, we provide a flexible and highly configurable LaTeX backend.

Regarding the use of formal methods to formalise standards, the Event-B method was proposed by Fotso et al. [11] 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. [16] propose an Event-B framework for formalising standard conformance through formal modelling of standards as ontologies. The proposed approach was exemplified on the ARINC 661 standard. These works are essentially interested in expressing ontological concepts in a formal method but do not explicitly deal with the formalisation of invariants defined in ontologies. The question of ontology-mappings is not addressed.

Another work along the line of certification standard support is Isabelle/SACM [10], which is a plug-in into Isabelle/DOF in order to provide specific support for the OMG Structured Assurance Case Meta-Model. The use of Isabelle/SACM guarantees well-formedness, consistency, and traceability of assurance cases, and allows a tight integration of formal and informal evidence of various provenance.

Obvious future applications for supporting the link between *formal* and *informal* content, i.e. between *information* and *knowledge*, consist in advanced search facilities in mathematical libraries such as the Isabelle Archive of Formal Proofs [15]. The latter passed the impressive numbers of 730 articles, written by 450 authors at the beginning of 2023. Related approaches to this application are a search engine like <http://shinh.org/wfs> which uses clever text-based search methods in a large number of formulas, which is, however, agnostic of their logical context and of formal proof. Related is also the OAF project [14] which developed a common ontological format, called OMDoc/MMT, and six *export* functions from major ITP systems into it. Limited to standard search techniques on this structured format, the approach remains agnostic on logical contexts and an in-depth use of typing information.

6 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 F-Logic 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 or plug-ins. Using a combination of advanced code-generation, symbolic execution and reification techniques existing in the Isabelle ecosystem, we provide the advantages of a smooth integration into the Isabelle IDE. Moreover, our approach leverages the use of 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 [17]. We 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 `@{typ <’ $\tau, @{term < $a + b and @{thm <HOL.refl>}, which are currently implemented just as validations of references into the 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 [20] (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. de Recherche en Informatique et en Automatique, I.N.: The OCaml Manual - Release 5. <https://v2.ocaml.org/manual/ocamldoc.html> (2022), [Online on ar-tima.com; accessed 23-02-2023]

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://www.brucker.ch/bibliography/abstract/brucker.ea-isabelle-ontologies-2018>
5. 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://www.brucker.ch/bibliography/abstract/brucker.ea-isabelledof-2019>
6. 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)
7. Common criteria for information technology security evaluation (version 3.1, release 5) (2017), available at <https://www.commoncriteriaportal.org/cc/>.
8. Corp., O.: The Java API Documentation Generator. <https://docs.oracle.com/javase/1.5.0/docs/tool> (2011), [Online on artima.com; accessed 23-02-2023]
9. Eclipse Foundation: Atl - a model transformation technology, <https://www.eclipse.org/atl/>, Accessed: 2022-03-15
10. Foster, S., Nemouchi, Y., Gleirscher, M., Wei, R., Kelly, T.: Integration of formal proof into unified assurance cases with isabelle/sacm. *Formal Aspects Comput.* **33**(6), 855–884 (2021). <https://doi.org/10.1007/s00165-021-00537-4>, <https://doi.org/10.1007/s00165-021-00537-4>
11. 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
12. 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
13. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. *J. ACM* **42**(4), 741–843 (jul 1995). <https://doi.org/10.1145/210332.210335>, <https://doi.org/10.1145/210332.210335>
14. 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>
15. 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
16. 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

17. 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>
18. Nipkow, T.: Functional automata. *Archive of Formal Proofs* (March 2004), <https://isa-afp.org/entries/Functional-Automata.html>, Formal proof development
19. Nipkow, T., Paulson, L.C., Wenzel, M.: *Isabelle/HOL—A Proof Assistant for Higher-Order Logic*, LNCS, vol. 2283. Springer (2002). <https://doi.org/10.1007/3-540-45949-9>
20. 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)
21. Venners, B., Gosling, J.: Visualizing with JavaDoc. <https://www.artima.com/articles/analyze-this#part3> (2003), [Online on artima.com; accessed 23-02-2023]
22. Wenzel, M.: *The Isabelle/Isar Reference Manual* (2020), part of the Isabelle distribution.