Using the Isabelle Ontology Framework
Linking the Formal with the Informal

Achim D. Brucker[1], Idir Ait-Sadoune[2], Paolo Crisafulli[3], and Burkhart Wolff[4]

[1] The University of Sheffield, Sheffield, UK
a.brucker@sheffield.ac.uk
[2] CentraleSupelec, Paris, France
idir.aitsadoune@centralesupelec.fr
[3] IRT-SystemX, Paris, France
paolo.crisafulli@irt-systemx.fr
[4] Université Paris-Sud, Paris, France
wolff@lri.fr

Abstract. While Isabelle is mostly known as part of Isabelle/HOL (an interactive theorem prover), it actually provides a framework for developing a wide spectrum of applications. A particular strength of the Isabelle framework is the combination of text editing, formal verification, and code generation.

Up to now, Isabelle's document preparation system lacks a mechanism for ensuring the structure of different document types (as, e.g., required in certification processes) in general and, in particular, mechanism for linking informal and formal parts of a document.

In this paper, we present Isabelle/DOF, a novel Document Ontology Framework on top of Isabelle. Isabelle/DOF allows for conventional typesetting *as well* as formal development. We show how to model document ontologies inside Isabelle/DOF, how to use the resulting meta-information for enforcing a certain document structure, and discuss ontology-specific IDE support.

Keywords: Ontology · Ontological modeling · Isabelle/DOF

1 Introduction

The linking of the *formal* to the *informal* is perhaps the most pervasive challenge in the digitization of knowledge and its propagation. This challenge incites numerous research efforts summarized under the labels "semantic web", "data mining", or any form of advanced "semantic" text processing. A key role in structuring this linking play *document ontologies* (also called *vocabulary* in the semantic web community [3]), 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 within scholarly articles, mathematical libraries, and in the engineering discourse of standardized software certification documents [9, 10].

Further applications are the domain-specific discourse in juridical texts or medical reports. In general, an ontology is a formal explicit description of *concepts* in a domain of discourse (called *classes*), properties of each concept describing *attributes* of the concept, as well as *links* between them. A particular link between concepts is the *is-a* relation declaring the instances of a subclass to be instances of the super-class.

The main objective of this paper is to present Isabelle/DOF, a novel framework to *model* typed ontologies and to *enforce* them during document evolution. Based on Isabelle, ontologies may refer to types, terms, proven theorems, code, or established assertions. Based on a novel adaption of the Isabelle IDE, a document is checked to be *conform* to a particular ontology—Isabelle/DOF is designed to give fast user-feedback *during the capture of content.* This is particularly valuable in case of document changes, where the *coherence* between the formal and the informal parts of the content can be mechanically checked.

To avoid any misunderstanding: Isabelle/DOF is *not a theory in HOL* on ontologies and operations to track and trace links in texts, it is an *environment to write structured text* which *may contain* Isabelle/HOL definitions and proofs like mathematical articles, tech-reports and scientific papers—as the present one, which is written in Isabelle/DOF itself. Isabelle/DOF is a plugin into the Isabelle/Isar framework in the style of [14].

The plan of the paper is follows: we start by introducing the underlying Isabelle sytem (Sect. 2) followed by presenting the essentials of Isabelle/DOF and its ontology language (Sect. 3). It follows Sect. 4, where we present three application scenarios from the point of view of the ontology modeling. In Sect. 5 we discuss the user-interaction generated from the ontological definitions. Finally, we draw conclusions and discuss related work in Sect. 6.

2 Background: The Isabelle System

While Isabelle is widely perceived as an interactive theorem prover for HOL (Higher-order Logic) [11], we would like to emphasize the view that Isabelle is far more than that: it is the *Eclipse of Formal Methods Tools.* This refers to the "*generic system framework of Isabelle/Isar underlying recent versions of Isabelle. Among other things, Isar provides an infrastructure for Isabelle plug-ins, comprising extensible state components and extensible syntax that can be bound to ML programs. Thus, the Isabelle/Isar architecture may be understood as an extension and refinement of the traditional 'LCF approach', with explicit infrastructure for building derivative* systems." [14]

The current system framework offers moreover the following features:

- a build management grouping components into to pre-compiled sessions,
- a prover IDE (PIDE) framework [12] with various front-ends
- documentation - and code generators,
- an extensible front-end language Isabelle/Isar, and,
- last but not least, an LCF style, generic theorem prover kernel as the most prominent and deeply integrated system component.

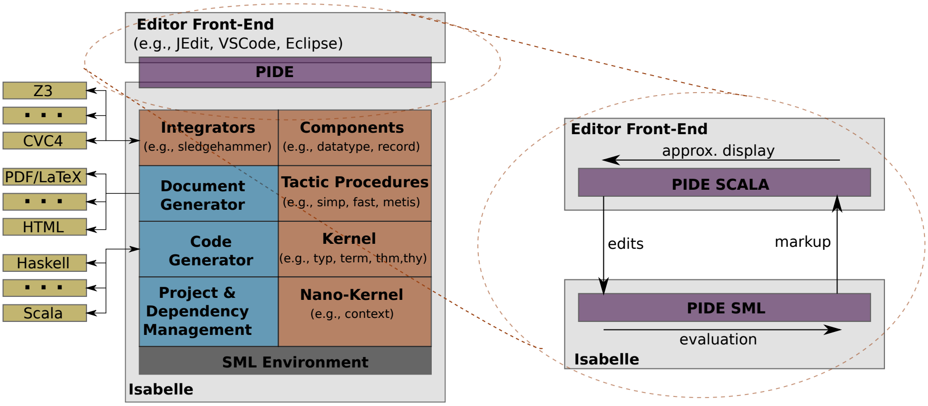


Fig. 1: The system architecture of Isabelle (left-hand side) and the asynchronous communication between the Isabelle system and the IDE (right-hand side).

The Isabelle system architecture shown in Fig. 1 comes with many layers, with Standard ML (SML) at the bottom layer as implementation language. The architecture actually foresees a *Nano-Kernel* (our terminology) which resides in the SML structure `Context`. This structure provides a kind of container called *context* providing an identity, an ancestor-list as well as typed, user-defined state for components (plugins) such as Isabelle/DOF. On top of the latter, the LCF-Kernel, tactics, automated proof procedures as well as specific support for higher specification constructs were built.

We would like to detail the documentation generation of the architecture, which is based on literate specification commands such as **`section`** . . . , **`subsection`** . . . , **`text`** . . . , etc. Thus, a user can add a simple text:

```
text‹This is a description.›
```

These text-commands can be arbitrarily mixed with other commands stating definitions, proofs, code, etc., and will result in the corresponding output in generated LaTeX or HTML documents. Now, *inside* the textual content, it is possible to embed a *text-antiquotation*:

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

which is represented in the generated output by:

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

where `refl` is actually the reference to the axiom of reflexivity in HOL. For the antiquotation `@{value "fac 5"}` we assume the usual definition for `fac` in HOL. Thus, antiquotations can refer to formal content, can be type-checked before being displayed and can be used for calculations before actually being typeset.

When editing, Isabelle's PIDE offers auto-completion and error-messages while typing the above *semi-formal* content.

3 Isabelle/DOF

An Isabelle/DOF document consists of three components:

- the *ontology definition*, which is an Isabelle theory file with definitions for document-classes and all auxiliary datatypes.
- the *core* of the document itself which is an Isabelle theory importing the ontology definition. Isabelle/DOF provides an own family of text-element commands such as **title***, **section***, **text***, etc., which can be annotated with meta-information defined in the underlying ontology definition.
- the *layout definition* for the given ontology exploiting this meta-information.

Isabelle/DOF is a novel Isabelle system component providing specific support for all these three parts. Note that the document core *may*, but *must* not use Isabelle definitions or proofs for checking the formal content—the present paper is actually an example of a document not containing any proof.

The document generation process of Isabelle/DOF is currently restricted to LaTeX, which means that the layout is defined by a set of LaTeX style files. Several layout definitions for one ontology are possible and pave the way that different *views* for the same central document were generated, addressing the needs of different purposes and/or target readers.

While the ontology and the layout definition will have to be developed by an expert with knowledge over Isabelle and Isabelle/DOF and the back end technology depending on the layout definition, the core is intended to require only minimal knowledge of these two. Document core authors *can* use LaTeX commands in their source, but this limits the possibility of using different representation technologies, e.g., HTML, and increases the risk of arcane error-messages in generated LaTeX.

The Isabelle/DOF ontology specification language consists basically on a notation for document classes, where the attributes were typed with HOL-types and can be instantiated by terms HOL-terms, i.e., the actual parsers and type-checkers of the Isabelle system were reused. This has the particular advantage that Isabelle/DOF commands can be arbitrarily mixed with Isabelle/HOL commands providing the machinery for type declarations and term specifications such as enumerations. In particular, document class definitions provide:

- a HOL-type for each document class as well as inheritance,
- support for attributes with HOL-types and optional default values,
- support for overriding of attribute defaults but not overloading, and
- text-elements annotated with document classes; they are mutable instances of document classes.

Attributes referring to other ontological concepts are called *links*. The HOL-types inside the document specification language support built-in types for

Isabelle/HOL `typ`'s, `term`'s, and `thm`'s reflecting internal Isabelle's internal types for these entities; when denoted in HOL-terms to instantiate an attribute, for example, there is a specific syntax (called *inner syntax antiquotations*) that is checked by Isabelle/DOF for consistency.

Document classes can have a **`where`**-clause containing a regular expression over class names. Classes with such a **`where`** are called *monitor classes*. While document classes and their inheritance relation structure meta-data of text-elements, monitor classes enforce structural organization of documents via the language specified by the regular expression enforcing a sequence of text-elements that must belong to the corresponding classes.

To start using Isabelle/DOF, one creates an Isabelle project (with the name `IsaDofApplications`):

```
isabelle DOF_mkroot -o scholarly_paper -t lncs -d IsaDofApplications
```

where the `-o scholarly_paper` specifies the ontology for writing scientific articles and `-t lncs` specifies the use of Springer's LaTeX-configuration for the Lecture Notes in Computer Science series. The project can be formally checked, including the generation of the article in PDF using the following command:

```
isabelle build -d . IsaDofApplications
```

4 Modeling Ontologies in Isabelle/DOF

In this section, we will use the Isabelle/DOF document ontology language for three different application scenarios: for scholarly papers, for mathematical exam sheets as well as standardization documents where the concepts of the standard are captured in the ontology. For space reasons, we will concentrate in all three cases on aspects of the modeling due to space limitations.

4.1 The Scholar Paper Scenario: Eating One's Own Dog Food

The following ontology is a simple ontology modeling scientific papers. In this Isabelle/DOF application scenario, we deliberately refrain from integrating references to (Isabelle) formal content in order demonstrate that Isabelle/DOF is not a framework from Isabelle users to Isabelle users only. Of course, such references can be added easily and represent a particular strength of Isabelle/DOF.

The first part of the ontology `scholarly_paper` (see Fig. 2) contains the document class definitions with the usual text-elements of a scientific paper. The attributes `short_title`, `abbrev` etc are introduced with their types as well as their default values. Our model prescribes an optional `main_author` and a todo-list attached to an arbitrary text section; since instances of this class are mutable (meta)-objects of text-elements, they can be modified arbitrarily through subsequent text and of course globally during text evolution. Since `author` is a HOL-type internally generated by Isabelle/DOF framework and can therefore appear in the `main_author` attribute of the `text_section` class; semantic links between concepts can be modeled this way.

```
doc_class title =
  short_title :: "string option" <= None

doc_class subtitle =
  abbrev :: "string option" <= None

doc_class author =
  affiliation :: "string"

doc_class abstract =
  keyword_list :: "string list" <= None

doc_class text_section =
  main_author :: "author option" <= None
  todo_list :: "string list" <= "[]"
```

Fig. 2: The core of the ontology definition for writing scholarly papers.

The translation of its content to, e.g., Springer's LaTeX setup for the Lecture Notes in Computer Science Series, as required by many scientific conferences, is mostly straight-forward.

Fig. 3 shows the corresponding view in the Isabelle/PIDE of the present paper. Note that the text uses Isabelle/DOF's own text-commands containing the meta-information provided by the underlying ontology. We proceed by a definition of `introduction`'s, which we define as the extension of `text_section` which is intended to capture common infrastructure:

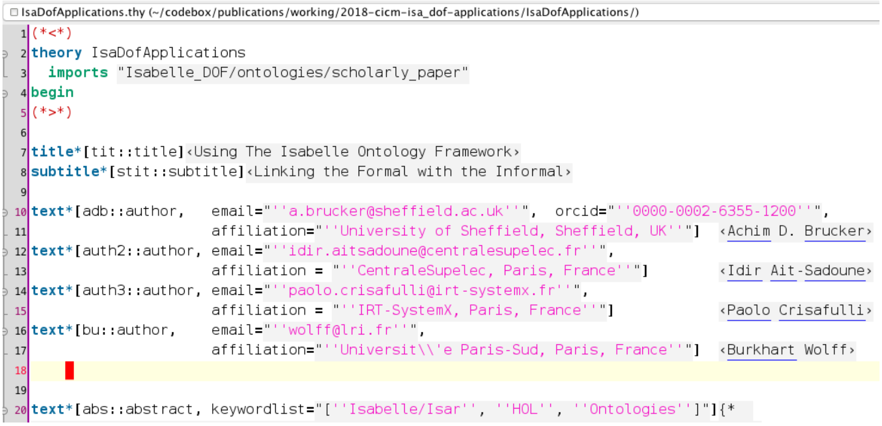
```
doc_class introduction = text_section +
  comment :: string
```

As a consequence of the definition as extension, the `introduction` class inherits the attributes `main_author` and `todo_list` together with the corresponding default values.

As a variant of the introduction, we could add here an attribute that contains the formal claims of the article—either here, or, for example, in the keyword list of the abstract. As type, one could use either the built-in type `term` (for syntactically correct, but not necessarily proven entity) or `thm` (for formally proven entities). It suffices to add the line:

```
  claims :: "thm list"
```

and to extent the LaTeX-style accordingly to handle the additional field. Note that `term` and `thm` are types reflecting the core-types of the Isabelle kernel. In a corresponding conclusion section, one could model analogously an achievement section; by programming a specific compliance check in SML, the implementation of automated forms of validation check for specific categories of papers is envisageable. Since this requires deeper knowledge in Isabelle programming, however, we consider this out of the scope of this paper.



IsaDofApplications.thy (~/codebox/publications/working/2018-cicm-isa_dof-applications/IsaDofApplications/)

```
(*<*)
theory IsaDofApplications
  imports "Isabelle_DOF/ontologies/scholarly_paper"
begin
(*>*)

title*[tit::title]‹Using The Isabelle Ontology Framework›
subtitle*[stit::subtitle]‹Linking the Formal with the Informal›

text*[adb::author,   email="''a.brucker@sheffield.ac.uk''",  orcid="''0000-0002-6355-1200''",
                     affiliation="''University of Sheffield, Sheffield, UK''"]  ‹Achim D. Brucker›
text*[auth2::author, email="''idir.aitsadoune@centralesupelec.fr''",
                     affiliation = "''CentraleSupelec, Paris, France''"]        ‹Idir Ait-Sadoune›
text*[auth3::author, email="''paolo.crisafulli@irt-systemx.fr''",
                     affiliation = "''IRT-SystemX, Paris, France''"]            ‹Paolo Crisafulli›
text*[bu::author,    email="''wolff@lri.fr''",
                     affiliation="''Universit\\'e Paris-Sud, Paris, France''"]  ‹Burkhart Wolff›


text*[abs::abstract, keywordlist="[''Isabelle/Isar'', ''HOL'', ''Ontologies'']"]{*
```

Fig. 3: Ouroboros I: This paper from inside . . .

We proceed more or less conventionally by the subsequent sections (Fig. 4) and finish with a monitor class definition that enforces a textual ordering in the document core by a regular expression (Fig. 5).

```
doc_class technical = text_section +
  definition_list :: "string list" <= "[]"

doc_class example = text_section +
  comment :: string

doc_class conclusion = text_section +
  main_author :: "author option" <= None

doc_class related_work = conclusion +
  main_author :: "author option" <= None

doc_class bibliography =
  style :: "string option" <= "''LNCS''"
```

Fig. 4: Various types of sections of a scholarly papers.

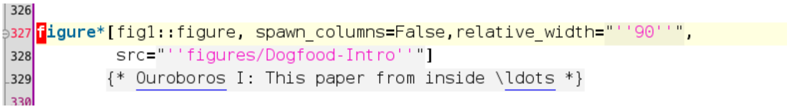
We might wish to add a component into our ontology that models figures to be included into the document. This boils down to the exercise of modeling structured data in the style of a functional programming language in HOL and to reuse the implicit HOL-type inside a suitable document class `figure`:

```
doc_class article =
  trace :: "(title + subtitle + author+ abstract +
            introduction + technical + example +
            conclusion + bibliography) list"
  where "(title  ~~ ⟦subtitle⟧ ~~ ⦃author⦄⁺+ ~~ abstract ~~
          introduction ~~ ⦃technical || example⦄⁺ ~~ conclusion ~~
          bibliography)"
```

Fig. 5: A monitor for the scholarly paper ontology.

```
datatype placement = h | t | b | ht | hb
doc_class figure = text_section +
  relative_width :: "string" (* percent of textwidth *)
  src   :: "string"
  placement :: placement
  spawn_columns :: bool <= True
```

Alternatively, by including the HOL-libraries for rationals, it is possible to use fractions or even mathematical reals. This must be counterbalanced by syntactic and semantic convenience. Choosing the mathematical reals, e.g., would have the drawback that attribute evaluation could be substantially more complicated.



```
figure*[fig1::figure, spawn_columns=False,relative_width="''90''",
         src="''figures/Dogfood-Intro''"]
        {* Ouroboros I: This paper from inside \ldots *}
```

Fig. 6: Ouroboros II: figures . . .

The document class `figure`—supported by the Isabelle/DOF text command **figure***—makes it possible to express the pictures and diagrams in this paper such as Fig. 6.

4.2 The Math-Exam Scenario

The Math-Exam Scenario is an application with mixed formal and semi-formal content. It addresses applications where the author of the exam is not present during the exam and the preparation requires a very rigorous process, as the french *baccaleaureat* and exams at The University of Sheffield.

We assume that the content has four different types of addressees, which have a different *view* on the integrated document

- the *setter*, i.e., the author of the exam,
- the *checker*, i.e., an internal person that checks the exam for feasibility and non-ambiguity,

- the *external examiner*, i.e., an external person that checks the exam for feasibility and non-ambiguity, and
- the *student*, i.e., the addressee of the exam.

The latter quality assurance mechanism is used in many universities, where for organizational reasons the execution of an exam takes place in facilities where the author of the exam is not expected to be physically present. Furthermore, we assume a simple grade system (thus, some calculation is required).

```
doc_class Author = ...
datatype Subject = algebra | geometry | statistical
datatype Grade = A1 | A2 | A3

doc_class Header = examTitle :: string
               examSubject :: Subject
               date     :: string
               timeAllowed :: int -- minutes

datatype ContentClass = setter
                | checker
                | external_examiner
                | student

doc_class Exam_item =
 concerns :: "ContentClass set"

doc_class Exam_item =
 concerns :: "ContentClass set"

type_synonym SubQuestion = string
```

Fig. 7: The core of the ontology modeling math exams.

The heart of this ontology (see Fig. 7) is an alternation of questions and answers, where the answers can consist of simple yes-no answers (QCM style check-boxes) or lists of formulas. Since we do not assume familiarity of the students with Isabelle (`term` would assume that this is a parse-able and type-checkable entity), we basically model a derivation as a sequence of strings (see Fig. 8).

In many institutions, it makes sense to have a rigorous process of validation for exam subjects: is the initial question correct? Is a proof in the sense of the question possible? We model the possibility that the *examiner* validates a question by a sample proof validated by Isabelle (see Fig. 9). In our scenario this sample proofs are completely *intern*, i.e., not exposed to the students but just additional material for the internal review process of the exam.

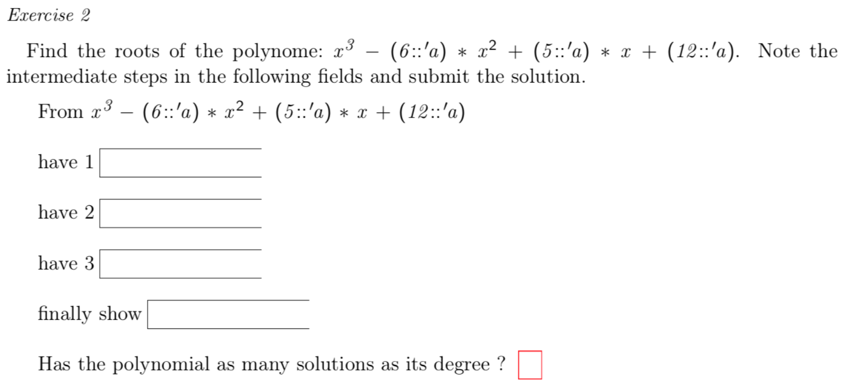
```
doc_class Answer_Formal_Step = Exam_item +
 justification :: string
 "term"    :: "string"

doc_class Answer_YesNo = Exam_item +
 step_label :: string
 yes_no  :: bool -- \isa{for\ checkboxes}

datatype Question_Type =
 formal | informal | mixed

doc_class Task = Exam_item +
 level  :: Level
 type   :: Question_Type
 subitems :: "(SubQuestion *
              (Answer_Formal_Step list + Answer_YesNo) list) list"
 concerns :: "ContentClass set" <= "UNIV"
 mark   :: int
doc_class Exercise = Exam_item +
 type   :: Question_Type
 content :: "(Task) list"
 concerns :: "ContentClass set" <= "UNIV"
 mark   :: int
```

Fig. 8: An exam can contain different types of questions.

```
doc_class Validation =
  tests :: "term list" <="[]"
  proofs :: "thm list" <="[]"

doc_class Solution = Exam_item +
 content :: "Exercise list"
 valids :: "Validation list"
 concerns :: "ContentClass set" <= "{setter,checker,external_examiner}"

doc_class MathExam=
 content :: "(Header + Author + Exercise) list"
 global_grade :: Grade
 where "⦃Author⦄^+ ~~ Header ~~ ⦃Exercise ~~ Solution⦄^+ "
```

Fig. 9: Validating exams.

Using the LaTeX package hyperref, it is possible to conceive an interactive exam-sheets with multiple-choice and/or free-response elements (see Fig. 10). With the help of the latter, it is possible that students write in a browser a formal mathematical derivation—as part of an algebra exercise, for example—which is submitted to the examiners electronically.



Exercise 2

Find the roots of the polynome: $x^3 - (6::'a) * x^2 + (5::'a) * x + (12::'a)$. Note the intermediate steps in the following fields and submit the solution.

From $x^3 - (6::'a) * x^2 + (5::'a) * x + (12::'a)$

have 1

have 2

have 3

finally show

Has the polynomial as many solutions as its degree ? □

Fig. 10: A generated QCM fragment ...

4.3 The Certification Scenario Following CENELEC

Documents to be provided in formal certifications (such as CENELEC 50126/50128, the DO-178B/C, or Common Criteria) can much profit from the control of ontological consistency: a lot of an evaluators work consists in tracing down the links from requirements over assumptions down to elements of evidence, be it in the models, the code, or the tests. In a certification process, traceability becomes a major concern; and providing mechanisms to ensure complete traceability already at the development of the global document will clearly increase speed and reduce risk and cost of a certification process. Making the link-structure machine-checkable, be it between requirements, assumptions, their implementation and their discharge by evidence (be it tests, proofs, or authoritative arguments), is therefore natural and has the potential to decrease the cost of developments targeting certifications. Continuously checking the links between the formal and the semi-formal parts of such documents is particularly valuable during the (usually collaborative) development effort.

As in many other cases, formal certification documents come with an own terminology and pragmatics of what has to be demonstrated and where, and how the trace-ability of requirements through design-models over code to system environment assumptions has to be assured.

In the sequel, we present a simplified version of an ontological model used in a case-study [8]. We start with an introduction of the concept of requirement (see Fig. 11). Such ontologies can be enriched by larger explanations and examples, which may help the team of engineers substantially when developing the central document for a certification, like an explication what is precisely the difference between an *hypothesis* and an *assumption* in the context of the evaluation standard. Since the PIDE makes for each document class its definition available by a simple mouse-click, this kind on meta-knowledge can be made far more accessible during the document evolution.

```
doc_class requirement = long_name :: "string option"

doc_class requirement_analysis = no :: "nat"
  where "requirement_item +"

doc_class hypothesis = requirement +
    hyp_type :: hyp_type <= physical (* default *)

datatype ass_kind = informal | semiformal | formal

doc_class assumption = requirement +
    assumption_kind :: ass_kind <= informal
```

Fig. 11: Modeling requirements.

For example, the term of category *assumption* is used for domain-specific assumptions. It has formal, semi-formal and informal sub-categories. They have to be tracked and discharged by appropriate validation procedures within a certification process, by it by test or proof. It is different from a hypothesis, which is globally assumed and accepted.

In the sequel, the category *exported constraint* (or *ec* for short) is used for formal assumptions, that arise during the analysis, design or implementation and have to be tracked till the final evaluation target, and discharged by appropriate validation procedures within the certification process, by it by test or proof. A particular class of interest is the category *safety related application condition* (or *srac* for short) which is used for *ec*'s that establish safety properties of the evaluation target. Their track-ability throughout the certification is therefore particularly critical. This is naturally modeled as follows:

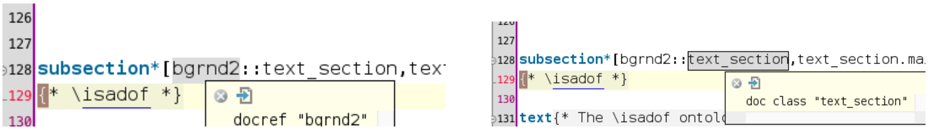
```
doc_class ec = assumption +
    assumption_kind :: ass_kind <= (*default *) formal

doc_class srac = ec +
    assumption_kind :: ass_kind <= (*default *) formal
```

5 Ontology-Based IDE Support

We present a selection of interaction scenarios Sect. 4.1 and Sect. 4.3 with Isabelle/PIDE instrumented by Isabelle/DOF.

5.1 A Scholarly Paper

In Fig. 12a and Fig. 12b we show how hovering over links permits to explore its meta-information. Clicking on a document class identifier permits to hyperlink



```
subsection*[bgrnd2::text_section,tex
{* \isadof *}
docref "bgrnd2"
```

(a) Exploring a Reference of a Text-Element.

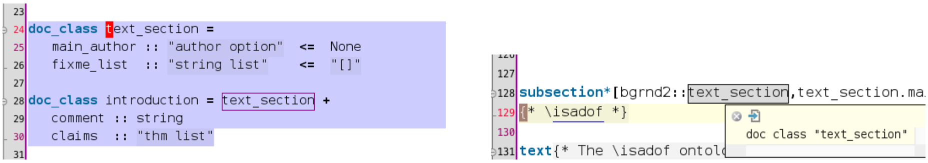
```
subsection*[bgrnd2::text_section,text_section.ma
{* \isadof *}
doc class "text_section"
text{* The \isadof ontol
```

(b) Exploring the class of a text element.

Fig. 12: Exploring text elements.

into the corresponding class definition (Fig. 13a); hovering over an attribute-definition (which is qualified in order to disambiguate; Fig. 13b).

An ontological reference application in Fig. 14: the ontology-dependant antiquotation `@{example ...}` refers to the corresponding text-elements. Hovering allows for inspection, clicking for jumping to the definition. If the link does not exist or has a non-compatible type, the text is not validated.



```
doc_class text_section =
   main_author :: "author option"  <=  None
   fixme_list  :: "string list"    <=  "[]"

doc_class introduction = text_section +
   comment :: string
   claims  :: "thm list"
```

(a) Hyperlink to Class-Definition.

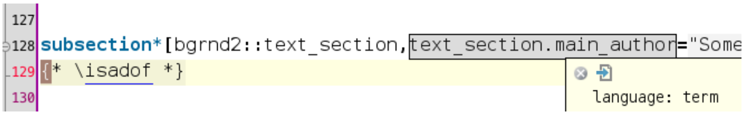
```
subsection*[bgrnd2::text_section,text_section.ma
{* \isadof *}
doc class "text_section"
text{* The \isadof ontol
```

(b) Exploring an attribute.

Fig. 13: Hyperlinks.

5.2 CENELEC

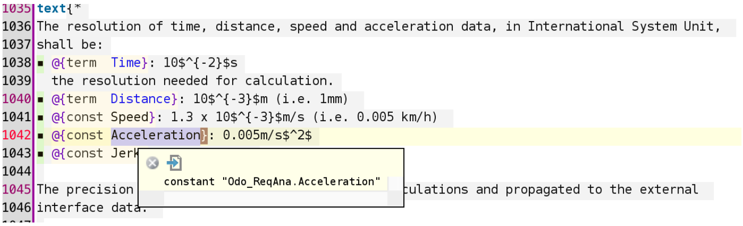
The corresponding view in Fig. 15 shows core part of a document, coherent to the Sect. 4.3. The first sample shows standard Isabelle antiquotations [13] into formal entities of a theory. This way, the informal parts of a document get “formal content” and become more robust under change.



```
subsection*[bgrnd2::text_section,text_section.main_author="Some
{* \isadof *}
language: term
```

Fig. 14: Exploring an attribute (hyperlinked to the class).

The subsequent sample in Fig. 16 shows the definition of an *safety-related application condition*, a side-condition of a theorem which has the consequence that a certain calculation must be executed sufficiently fast on an embedded

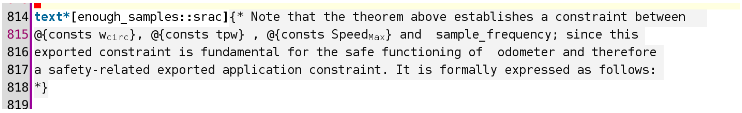


```
text{*
The resolution of time, distance, speed and acceleration data, in International System Unit,
shall be:
▪ @{term  Time}: 10$^{-2}$s
  the resolution needed for calculation.
▪ @{term  Distance}: 10$^{-3}$m (i.e. 1mm)
▪ @{const Speed}: 1.3 x 10$^{-3}$m/s (i.e. 0.005 km/h)
▪ @{const Acceleration}: 0.005m/s$^2$
▪ @{const Jerk

The precision                                              ulations and propagated to the external
interface data.
```

```
constant "Odo_ReqAna.Acceleration"
```

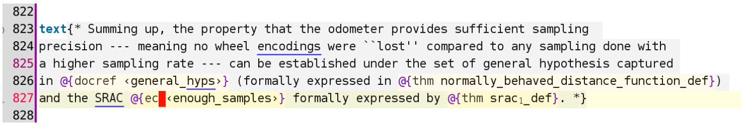
Fig. 15: Standard antiquotations referring to theory elements.

device. This condition can not be established inside the formal theory but has to be checked by system integration tests.



```
text*[enough_samples::srac]{* Note that the theorem above establishes a constraint between
@{consts w_circ}, @{consts tpw} , @{consts Speed_Max} and  sample_frequency; since this
exported constraint is fundamental for the safe functioning of  odometer and therefore
a safety-related exported application constraint. It is formally expressed as follows:
*}
```

Fig. 16: Defining a SRAC reference . . .



```

text{* Summing up, the property that the odometer provides sufficient sampling
precision --- meaning no wheel encodings were ``lost'' compared to any sampling done with
a higher sampling rate --- can be established under the set of general hypothesis captured
in @{docref ‹general_hyps›} (formally expressed in @{thm normally_behaved_distance_function_def})
and the SRAC @{ec ‹enough_samples›} formally expressed by @{thm srac_1_def}. *}

```

Fig. 17: Using a SRAC as EC document reference.

Now we reference in Fig. 17 this safety-related condition; however, this happens in a context where general *exported constraints* are listed. Isabelle/DOF's checks establish that this is legal in the given ontology.

This example shows that ontological modeling is indeed adequate for large technical, collaboratively developed documentations, where modifications can lead easily to incoherence. The current checks help to systematically avoid this type of incoherence between formal and informal parts.

6 Conclusion and Related Work

We have demonstrated the use of Isabelle/DOF, a novel ontology modeling and enforcement IDE deeply integrated into the Isabelle/Isar Framework. The two most distinguishing features are

– Isabelle/DOF and its ontology language are a strongly typed language that allows for referring (albeit not reasoning) to entities of Isabelle/HOL, most notably types, terms, and (formally proven) theorems, and
– Isabelle/DOF is supported by the Isabelle/PIDE framework; thus, the advantages of an IDE for text-exploration (which is the type of this link? To which text element does this link refer? Which are the syntactic alternatives here?) were available during editing instead of a post-hoc validation process.

Of course, a conventional batch-process also exists which can be used for the validation of large document bases in a conventional continuous build process. This combination of formal and semi-informal elements, as well as a systematic enforcement of the coherence to a document ontology of the latter, is, as we believe, novel and offers a unique potential for the semantic treatment of scientific texts and technical documentations.

To our knowledge, this is the first ontology-driven framework for editing mathematical and technical documents that focuses particularly on documents mixing formal and informal content—a type of documents that is very common in technical certification processes. We see mainly one area of related works: IDEs and text editors that support editing and checking of documents based on an ontology. There is a large group of ontology editors (e.g., Protégé [5], Fluent Editor [1], NeOn [2], or OWLGrEd [4]). With them, we share the support for defining ontologies as well as auto-completion when editing documents based on an ontology. While our ontology definitions are currently based on a textual definition, widely used ontology editors (e.g., OWLGrEd [4]) also support graphical notations. This could be added to Isabelle/DOF in the future. A unique feature of Isabelle/DOF is the deep integration of formal and informal text parts. The only other work in this area wea are aware of is rOntorium [6], a plugin for Protégé that integrates R [7] into an ontology environment. Here, the main motivation behind this integration is to allow for statistically analyze ontological documents. Thus, this is complementary to our work.

Isabelle/DOF in its present form has a number of technical short-comings as well as potentials not yet explored. On the long list of the short-comings is the fact that strings inside HOL-terms do not support, for example, Unicode. For the moment, Isabelle/DOF is conceived as an add-on for Isabelle/HOL; a much deeper integration of Isabelle/DOF into Isabelle could increase both performance and uniformity. Finally, different target presentation (such as HTML) would be highly desirable in particular for the math exam scenarios. And last but not least, it would be desirable that PIDE itself is "ontology-aware" and can, for example, use meta-information to control read- and write accesses of *parts* of documents.

Availability. The implementation of the framework, the discussed ontology definitions, and examples are available at https://git.logicalhacking.com/HOL-OCL/Isabelle_DOF/.

Acknowledgement. This work was partly supported by the framework of IRT SystemX, Paris-Saclay, France, and therefore granted with public funds within the scope of the Program "Investissements d'Avenir".

References

[1] Fluent editor (2018). http://www.cognitum.eu/Semantics/FluentEditor/

[2] The neon toolkit (2018). http://neon-toolkit.org

[3] Ontologies (2018). https://www.w3.org/standards/semanticweb/ontology

[4] Owlgred (2018). http://owlgred.lumii.lv/

[5] Protégé (2018). https://protege.stanford.edu

[6] R language package for fluent editor (rontorion) (2018). http://www.cognitum.eu/semantics/FluentEditor/rOntorionFE.aspx

[7] Adler, J.: R in a nutshell. O'Reilly, Sebastopol, CA (2010)

[8] Bezzecchi, S., Crisafulli, P., Pichot, C., Wolff, B.: Making agile development processes fit for v-style certification procedures. In: ERTS'18, ERTS Conference Proceedings (2018)

[9] Boulanger, J.L.: CENELEC 50128 and IEC 62279 Standards. Wiley-ISTE, Boston (2015). The reference on the standard.

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

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

[12] Wenzel, M.: Asynchronous user interaction and tool integration in isabelle/PIDE. In: Klein, G., Gamboa, R. (eds.) Interactive Theorem Proving (ITP), *LNCS*, vol. 8558, pp. 515–530. Springer (2014).

[13] Wenzel, M.: The Isabelle/Isar Reference Manual (2017). Part of the Isabelle distribution.

[14] Wenzel, M., Wolff, B.: Building formal method tools in the Isabelle/Isar framework. In: Schneider, K., Brandt, J. (eds.) TPHOLs 2007, no. 4732 in LNCS, pp. 352–367. Springer (2007).