Semantic Interrelation of Documents via an Ontology*

Bernd Krieg-Brückner, Arne Lindow, Christoph Lüth, Achim Mahnke, George Russell
Bremen Institute of Safe Systems, FB3 Mathematik und Informatik, Universität Bremen

Abstract: This paper describes how to use an ontology for extensive semantic interrelation of documents in order to achieve sustainable development, i.e. continuous long-term usability of the contents. The ontology is structured via packages (corresponding to whole documents). Packages are related by import such that semantic interrelation becomes possible not only within a document but also between different documents. Coherence and consistency are enhanced by change management in a repository, including version control and configuration management. Semantic interrelation is realized by particular LaTeX commands for the declaration and definition of classes, objects and relations, and references to them, such that they can be used in standard LaTeX documents, in particular, with a new LaTeX style for educational material (slides, handouts, annotated courses, assignments, and so on).

1 Introduction

Sharing and reuse is the key to efficient development. Unfortunately, while there has been a large body of research concerning the sharing and reuse of program developments, sharing and reuse of documents has until now been mainly done by little more than cut and paste. However, to ensure *sustainable development*, i.e. continuous long-term usability of the contents, sharing and reuse need to be supported by tools and methods taking into account the *semantic* structure of the document. In developing these methods and tools we can benefit from the experience in formal software development and associated support tools.

In this paper, we address this problem by introducing a methodoloy to specify coherence and consistency of documents by interrelation of semantic terms and structural entities, supported by a tool for fine-grained version control and configuration management including change management. Semantic interrelation explicates the meaning lying behind the textual content, and relates the semantic concepts within and across documents by means of an *ontology*. To allow change management, each document is structured in-the-small. Each document corresponds to a package, and packages may be structured in-the-large using folders and import relations.

The ideas and methods explained in this paper have been developed in the MMiSS project [KBHL+03] which aimed at the construction of a multi-media Internet-based adaptive ed-

*The MMiSS project has been supported by the German Ministry for Research and Education, bmb+f, in its programme "New Media in Education".

ucational system. Its content initially covers a curriculum in the area of Safe and Secure Systems, but the ideas are applicable throughout computer science and beyond. Moreover, the approach for semantic interrelation described in this paper applies to all kinds of (LaTeX) documents. As an application area, we will concentrate on the preparation of lecture material, in particular, on slides for presentation in class.

This paper is strucured as follows: we first introduce semantic interrelation of documents by an ontology (Sect. 2), including the LaTeX commands to annotate semantic structure. We then describe structuring in-the-large (Sect. 3), and show how to manage change in our setting (Sect. 4).

2 Semantic Interrelation and Ontologies

Semantic Terms in an Example Lecture. For a lecture, semantic interrelation is concerned with its topics, i.e. the terminology, and how they are interrelated. As our running example, we will consider an *Introduction to Functional Programming* for undergraduate students, one of more than 20 lectures developed in the MMiSS project. A lecture might start by introducing the difference between imperative and functional programming languages, e.g. HASKELL, and proceed to state that a program consist of type definitions and function definitions. A function definition is given by a (system of) recursive equation(s), and possibly a signature (associating a type to the function). This way, a number of *semantic term*s have been introduced: *function definition*, *signature*, *function*, etc.

There are certain operations on these semantic terms which we implicitly use when writing a text: a semantic term has to be *declared*, is *defined* somewhere in the text, and we can *refer* to it. This constitutes the *semantic interrelation* of a document. A typical lecture will contain a lot of semantic terms; we should impose some structure: an ontology.

Ontologies provide the means for establishing a semantic structure. An ontology is a formal explicit description of concepts in a domain of discourse [UG96]. Ontologies are becoming increasingly important because they also provide the critical semantic foundations for many rapidly expanding technologies such as software agents, e-commerce, the "semantic web", and knowledge management. An *ontology* consists of (a hierarchy) of concepts and relations between these concepts, describing its various features and attributes.

The semantic terms for our introductory functional programming lecture are arranged in an ontology as shown in Fig. 1. Semantic terms for *class*es are ordered by a relation *is a subclass of* (or "*is a*", denoted by a hollow arrow); for example, `FunctionIdH` *is a* `IdH`. Relations between classes may be defined by the user. For example, a `FunctionIdH` *denotes* a `FunctionDefinitionH`, a `FunctionIdH` *calls* a `FunctionIdH`.

Classes and relations are used for defining the abstract semantical level; they provide a vocabulary to denote the concrete entities corresponding to these concepts. Once the abstract notions are declared in terms of classes, objects can be used to identify entities of semantic

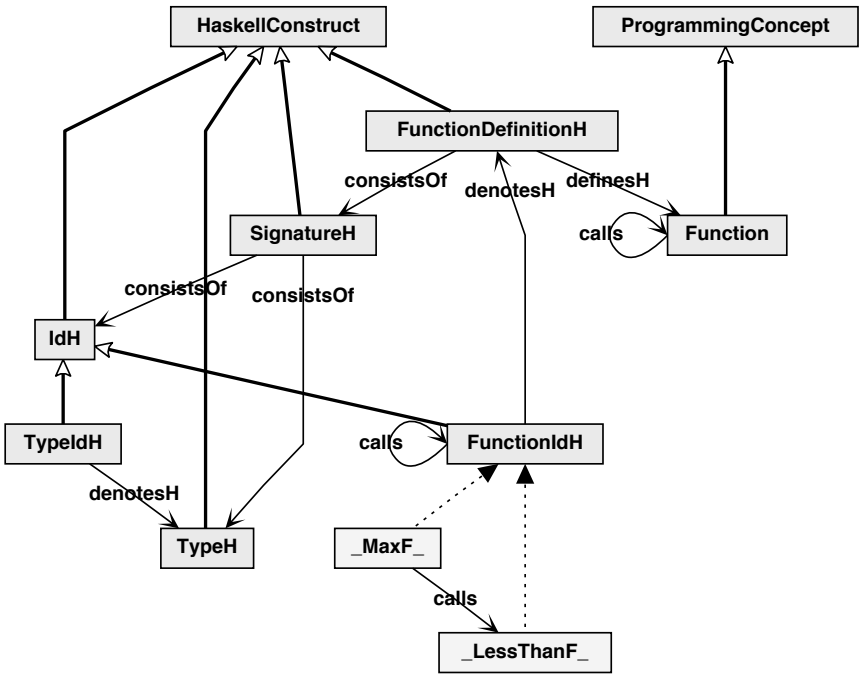

Figure 1: Example Ontology (Extract)

concepts. In our functional programming example, a function *max* could be introduced in the lecture in order to demonstrate different aspects of functional programming concepts. `MaxF` ("max") and `LessThanF` ("<") can be viewed as instances of an identifier and therefore be declared as objects of class `FunctionIdH`[1].

Objects can be semantically related to each other by applying the relations declared in the ontology. Suppose, we have defined a relation *calls*; then "max calls <" on the identifiers can be deduced from the function definitions they denote, and this will induce a corresponding calls relation on the functions they define.

Using the Ontology in the Example Lecture. Fig. 2 shows an example slide taken from the above mentioned lecture about functional programming. We decided to use LaTeX as the authoring format (cf. Sect. 2.1); special LaTeX environments are used to structure the content. Fig. 3 shows the corresponding MMiSSLaTeX source.

The environment `Paragraph` is used to encapsulate a conceptual unit of text that is to be treated as a whole. Each Paragraph is rendered on a separate slide with its title in the headline. The information is grouped into two `Itemize` environments rendered as list

[1] In Fig. 1, objects are depicted by leading and trailing underscores.

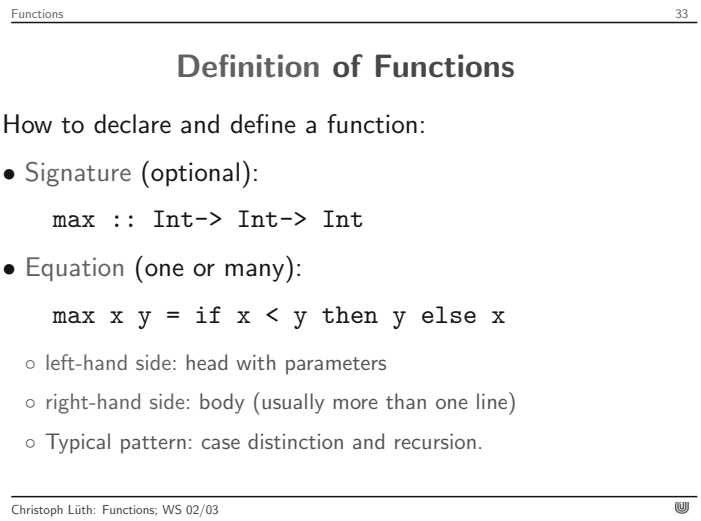

Figure 2: Example Lecture Slide

items with bullet marks. The signature and equation are treated as atoms encapsulated in `Code` environments. Atoms of a particular `FormalismAttribute` such as HASKELL can be analyzed and used by other tools (e.g. for extracting all `Code` fragments belonging to a HASKELL program and feeding them into a compiler).

Each concept introduced on this slide is declared as a semantic term (class) in the lectures ontology. Defining occurrences (`Def`) are highlighted in red in the presentation format, e.g. for `FunctionDefinitionH` (in the title), `SignatureH`, etc. References to one of these concepts will point to the defining occurrence on this slide. Similarly, references to classes defined elsewhere in this lecture or in another imported lecture (e.g. to `Recursion` in line 19 of Fig. 3) are optionally presented as hyperlinks (in blue) and/or with section references for paper output (as in the document you are reading now).

The objects `MaxF` and `LessThanF` are defined in line 10. They are related to each other by the command `\Relate{calls}{MaxF}{LessThanF}` in line 11.

2.1 Ontology Commands in MMiSSLaTeX

MMiSSLaTeX as an Authoring Format for Ontologies. The LaTeX commands described in this section, and other structuring environments for structuring in-the-small, have been developed as a special LaTeX style, called MMiSSLaTeX, to be converted to an XML representation for the Repository (cf. Sect. 4) and as an exchange format such that other tools processing ontologies may be used (cf. Sect. 2.2). The MMiSSLaTeX commands for ontologies and semantic interrelation can be used separately, with any LaTeX

MMiSSLaTeX

```
\begin{Paragraph} [Label=DefFunctions, Formalism=Haskell,
   Title={\Def[Definition of Functions]{FunctionDefinitionH}}]
How to declare and define a \Function{}:
 \begin{Itemize}
  \item \Def[Signature]{SignatureH} (optional):
   \begin{Code}[Label=SigMax]
     max :: Int -> Int -> Int
   \end{Code}
  \item \Def[Equation]{EquationH} (one or many):
   \Def[]{MaxF} \Def[]{LessThanF}
   \Relate{calls}{MaxF}{LessThanF}
   \begin{Code}[Label=EqMax]
     max x y = if x < y then y else x
   \end{Code}
   \begin{Itemize}
     \item \Def{LhsH}: head with parameters
     \item \Def{RhsH}: body (usually more than one line)
     \item Typical pattern: \Reference{CaseDistinction} and
                            \Reference{Recursion}.
   \end{Itemize}
 \end{Itemize}
\end{Paragraph}
```

Figure 3: MMiSSLaTeX Source of the Example Lecture Slide

document. They are used to "declare" the ontology of semantic terms used in a document, in a prelude up front. This "specification" of the document contains at least a rigorous hierarchical structure of the terminology (a taxonomy, the "signature" of the document), and may be seen as an elaborate index structure. Moreover, relations between terms may be defined for more *semantic* interrelation.

The ontology serves a dual purpose – just as the specification of an abstract datatype in program development: it specifies the content to be expected in the body of the document in an abstract, yet precise, manner – the content developers requirement specification; and it specifies the content for reference from the outside – the user's perspective, who may then view the body of the document as a black box. Indeed, foreign documents (not prepared in MMiSSLaTeX) may be encapsulated for the MMiSS Repository by providing an ontology in a MMiSSLaTeX "wrapper".

The content developer will use the `Def` command to specify the defining occurrence of a promised term, as for an index. Using the structuring in-the-large facilities via packages (cf. Sect. 3), the external user may then refer between documents using various kinds of reference commands, as the content developer may within a document.

We will now give an overview of the features of the commands, pointing out the main characteristics with a more detailed example for the declaration of a class and summarizing the commands for definitions of and references to classes, objects and relations.

A *class declaration* uses the command `Class`, e.g.

`\Class{FunctionIdH}{identifier}{IdH}`.

The first LaTeX parameter, `FunctionIdH`, denotes the particular semantic term we use in the ontology; the second, `identifier`, the phrase that is to appear in the text by default; the third, `IdH`, the superclass of `FunctionIdH`. The textual effect of a `Class` is nil; however, an entry in the ontology has been made:

- the default phrase, `identifier`, has been associated with the semantic term, `FunctionIdH`, as if a new macro had been defined for `FunctionIdH`, expanding to "identifier";
- a commitment has been made that this semantic term will be defined later on somewhere in the document; and
- the new class, `FunctionIdH`, has been related, as a new subclass, to the existing class `IdH`; it inherits all its properties.

Analogously, we may declare an object of a given class, which populates the ontology's instance level. It is important to point out that we distinguish between the semantic term, which has to be unique within a package, and the textual phrase, which appears within the document when the semantic term is referred to by a `Reference` or in the short form `\FunctionIdH` (not generating a hyperlink). This way, the translation of phrases can help a lot in the first step of translating the document to another language. Moreover, the same textual phrase in the rendered document can be assigned to different concepts, while still preserving the reference to the right semantic terms. Default phrases may be overridden when necessary, e.g. `\Ref[id]{FunctionIdH}`.

The declaration of a relation states the name of the relation itself, a default textual phrase and – possibly – the name of its superrelation. In addition, standard properties (cardinality constraints for domain and range and properties like *partial order* etc.) can be assigned. The domain and range classes are declared with a separate command `RefType`. It may be invoked several times, for different domain and range classes; thus the applicability of a relation can be declared as specifically as desired — and will be checked as specifically as possible.

Being designed specifically for the purpose of semantic interrelation of documents, our MMiSSLaTeX language cannot be directly compared to traditional ontology specification languages like Ontolingua or KIF. Taking the framework of Corcho and Gómez-Pérez ([CGP00]) as reference, our language incorporates primitives for concepts and taxonomies, hierachically structured relations and declaration of individuals (instances and facts). Regardless of being a more "leight-weight" approach to ontology engineering, the declarations can be translated to OWL and CASL-DL (see 2.2) which provides well-established formal semantics and allows for formal specification.

2.2 Benefits of Ontologies for Document Management

The notion of *ontology* in computer science stems from the artificial intelligence research area of knowledge representation. The general idea is to make knowledge explicit by expressing concepts and their relationships formally with the help of mathematical logics. Ontologies play a crucial role in the development of the Semantic Web [BLHL01].

The explication of semantics weaved into the textual contents of documents will enable much more advanced document management facilities. We will explain basic benefits here while postponing more intricate ones concerning change managment to Sect. 4.2.

Resolution of Ambiguities. One simple example illustrating the gains we get from using ontologies is connected with the difficulties we often encounter when we want to reuse or share material: the exact meaning of a concept is unclear, or different terms are used for the same semantic concept, or the same term is used for different semantic concepts. For example, consider the heavily overloaded term "process" in computer science (the process of software development, slightly different notions of parallel process, etc.). While a human user can often discriminate from the context, a tool must have unambiguous information: we would certainly expect a hyperlink to lead to the correct target definition (cf. also Sect. 3). As a more subtle example, consider the example ontology in Fig. 1. In the lecture's ontology, the phrase "identifier" is used for different concepts, distinguished by the terms `FunctionIdH`, `TypeIdH` (and the superclass `IdH`); it is often the case that one prefers light-weight default phrases over heavy phrases such as "function identifier", e.g. `the \FunctionIdH{} \MaxF{}` $\rightsquigarrow$ "the identifier max".

Ontology-Based Search. Semantic mark-up and interrelation also allows more powerful searching in documents, because indexing over the semantic terms is much more accurate than textual search over possibly ambiguous phrases. A user looking for specific material can not only be provided with more adequate terms but is also able to explore properties of concepts in order to understand relations and differences between them. The ontology graph is a powerful information source for navigation, in particular when provided with tools for selective graphical presentation (under development).

Tools for Ontologies. Tools are available to display (part of) an ontology in a graphic way (cf. Fig. 1). It is planned to extend this tool in order to provide a full graphical interactive user interface for ontology engineering. Moreover, some tools have already been implemented to feed the ontology of a document into ontology-related tools based on the emerging OWL standard for the Semantic Web [BvHH+04]. The connection to ontology tools opens up the possibility for efficient reasoning with ontologies developed in MMiSSLaTeX. The precise formal definition of ontologies (e.g. in a First-Order Logic framework such as CASL [ABKB+02, Mo04, LMKB]) provides considerable potential for the use of formal methods and tools for proving more complex properties, e.g. with Hets [HET].

3 Structuring in-the-Large via Packages

Packages provide a means for modular document development by introducing *name spaces*. When writing a document, authors introduce identifiers as labels for structural entities or as semantic terms in an ontology. If these identifiers, subsumed as *names* in the sequel, are defined more than once, we say there is a *name clash*.

A *package* is the largest structural entity. A package is a document that corresponds to a whole course or book and contains all structural entities pertaining to it. A package encapsulates the name space of a document, such that names defined in a package do not clash with names from other packages. In order to use names from other packages, these have to be imported explicitly (see below). In other words, packages are very much like modules in programming languages such as Modula-2, Haskell, or Java. As a general rule, name clashes are resolved on import by renaming, hiding, restricting or qualified import (or a combination of these), rather than by restricting the export.

A package contains a prelude that contains "global declarations" for it. The prelude consists of the *ontology prelude* and the *import prelude*.

The *ontology prelude* declares elements of an ontology (cf. Sect. 2). It acts like a signature of the package for semantic interrelation, promising these elements to be defined in this package, such that they become available when imported by another.

The *import prelude* specifies other packages to be imported. For each package to be imported, there are one or more *import directives* specifying the modalities of the import. The modalities allow to specify whether the import is *qualified* or *unqualified* (names are prefixed with the package name, or used as they are), whether the import is *local* or *global* (not reexported or reexported automatically), and they allow to *hide* (stating hidden identifiers, all others remain visible), *reveal* (stating only visible identifiers, all others remain hidden) or *rename* identifiers.

4 Change Management in the Repository

4.1 Fine-Grained Version Control

The *repository* provides version control and configuration management for our documents. Version-control is *fine-grained*, i.e. documents are broken down into their constituting structural entities and versioned on that level. Objects in the repository are organised in folders, which allow the grouping of repository objects much like directories in a file system. The objects in the repository form a graph, with structural entities such as sections, units or atoms as nodes, and the relations as edges, in particular the relation comprises. This constitutes the *structure graph* of all objects, which is one possible user's view of the repository. Fig. 4 shows the user interface: the structure graph, here our example slide from Sect. 2, is displayed with the daVinci graph visualization system [daV], which allows the user to navigate in the graph. At each node, users may select operations such as

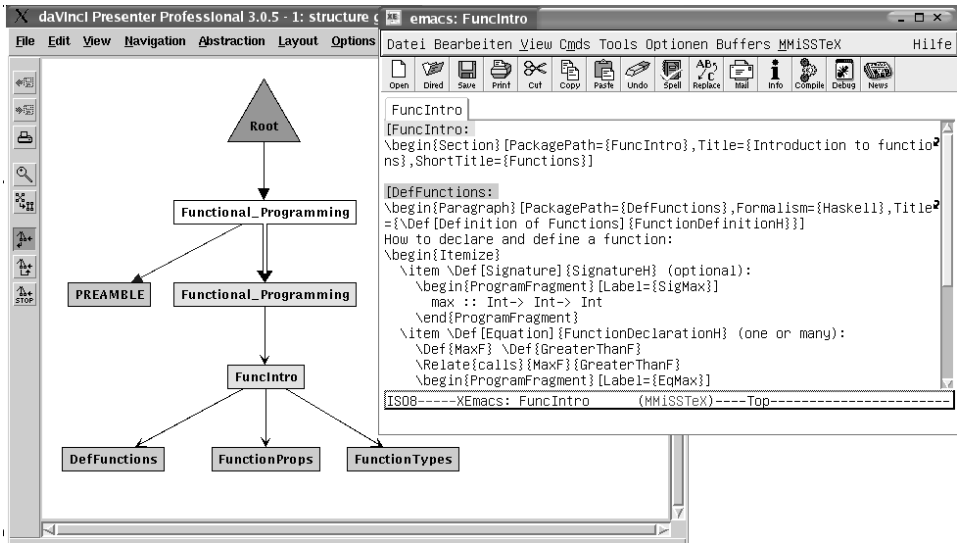

Figure 4: The MMiSS repository at work.

editing or calling a tool to typeset, compile or check the corresponding part of a document. For editing documents, the Emacs editor has been integrated very closely into the system, allowing the editing of single structural entities, but other editors or LaTeX system may be used to edit the document *in toto*.

Since LaTeX itself has neither name spaces nor a sophisticated module mechanism, the package mechanism is implemented on the level of the repository. When we check out a package from the repository, a MMiSSLaTeX document is generated, and the import prelude is supplemented by MMiSSLaTeX code from imported packages. When a package is commited back to the repository (or on request), the generated prelude is discarded, and the import statements are used to generate a new prelude.

4.2 Change Management

The notion of *change management* is used for the maintenance and preservation of consistency and completeness of a development during its evolution. More precisely, we want to have a *consistent configuration* in which all versions are compatible and there are no cyclic definitions or proofs. At the same time, it should be a *complete configuration*: there are no dangling forward references and links.

In the MMiSS system, consistency is handled on two levels: firstly, by checking that documents confirm to a specific format specified by an XML document type definition (DTD). (MMiSSLaTeX documents are converted into an XML format before being stored in the repository, see [KBHL+03].) This check ensures well-formedness conditions, e.g.

a package contains sections, a section contains another section, a section contains a theory, or a unit is contained in a section. Secondly, additionally implemented checks cover completeness (all forward references and links exist) and structural consistency (no cyclic definitions or proofs).

Such notions are well-known for formal languages. In contrast, natural language used for writing teaching material does not usually possess a well-defined semantics, and the notion of consistency is arguable. Different authors may postulate different requirements on the material in order to regard it as being consistent. The existence of a user-defined ontology helps a great deal to check references. However, we can make even better use of the information contained in the ontology.

The System Ontology. The ultimate aim is to allow change management with regards to consistency and completeness requirements defined by the user in terms of an ontology. In order to unify this approach with the structural consistency and completeness properties introduced above, we express the document structure, originally defined by a document type definition, as an ontology, the so-called *systems ontology*.

The MMiSS systems ontology defines the following relations between structural elements:

comprises An obvious structuring mechanism is nesting of individual parts of a document, leading to the *contains* relation The contains relation is part of a family of *comprises* relations that share common properties.

reliesOn Besides the comprises relation, there is a family of *relies on* relations, reflecting the various dependencies between different parts of a document. For example, a theorem *lives in* a theory, or a proof *proves* a theorem.

pointsTo The family of *points to* relations is very similar, and relates references with the defining occurrence of a semantic term, or the source and target of links.

variantOf Another structuring relation is introduced by variants. Parts of a document may e.g. be written in various languages which gives rise to a *variant of* relation between these document parts and their constituents; it is an equivalence relation.

It is now straightforward to formulate the consistency and completeness rules from above in terms of invariants of these relations. Formulating these invariants as formal rules will enable us to implement a generic and flexible change management that keeps track of the invariants and informs the user about violations when a previously consistent document has been revised.

Properties of Interactions between Structuring Mechanisms. This approach also allows us to lift relations to structuring mechanisms allowing more modular and localized change management. For example, relating the comprises and relies-on relations allows us to formalize invariants regarding the closure of document parts with respect to the relies-on relation: We can require that there is a proof for each theorem in a package. Furthermore,

if two structural entities are related by relies-on, their relation is propagated along the comprises relation towards the root of the hierarchy of nested structural entities, such that (for a theorem T, a proof P, and sections A, B):

B contains P and A contains T and P proves T, then B relies on A.

If the user changes section A, the repository will only need to check all sections that A relies on (such as B here) for invariants, and not the whole document. However, in contrast to formal developments as in e.g. the MAYA system [AHMS02], there is no rigorous requirement that a document should obey all the rules. There may be good reasons, for instance, to present first a "light-weight" introduction to all notions introduced in a section before giving the detailed definitions. In this particular case, one would want to introduce forward pointers to the definitions rather than making the definitions rely on the introduction; thus the rules are covered.

In any case, the more structure there is, the better are the chances for preserving consistency and completeness; any investment in introducing more relies-on relations, for example, will pay off eventually. The change management will observe whether revisions by the user will affect these relations and, depending on the user's preferences, emit corresponding warnings.

The eventual aim is to allow users to specify individual notions of consistency by formulating the rules that the relations should obey. This should be possible for the relations between the particular (predefined) structuring mechanisms, but also in general between semantic terms of the user's own ontology.

5 Conclusion

We have presented some contributions to the important issue of *sustainable development* and management of documents, notably *semantic interrelation* via a user-defined ontology. The MMiSSLaTeX style is available, tools for structuring in-the-small and fine-grained version control have been implemented. Structuring in-the-large via a hierarchy of packages also allows import of document-related ontologies and thus supports the *structuring of ontologies* themselves. Similarly, the basic infrastructure for *change management* of documents (and ontologies as a by-product) is available; the ideas about a formal definition of *invariant properties* of relations are presently being implemented. A considerable amount of teaching material has been produced and is actively used in academic teaching; an explicit procedure for evaluation has been set up and is currently being implemented.

MMiSSForum. As an open source model is used, teaching content and tools are freely available to achieve a wider national and international take-up. To assist this, a MMiSS Forum has been set up to evaluate the emerging curriculum and assist its development and distribution; you are welcome to join ([MMi]).

Acknowledgement. We are grateful to the members of the MMiSS project, see also [KBHL+03].

References

[ABKB+02] Astesiano, E., Bidoit, M., Krieg-Brückner, B., Kirchner, H., Mosses, P. D., Sannella, D., und Tarlecki, A.: CASL – the common algebraic specification language. *Theoretical Computer Science*. 286:153–196. 2002. www.cofi.info.

[AHMS02] Autexier, S., Hutter, D., Mossakowski, T., und Schairer, A.: The development graph manager MAYA (system description). In: Kirchner, H. und Reingeissen, C. (Hrsg.), *Algebraic Methodology and Software Technology, 2002*. volume 2422 of *Lecture Notes in Computer Science*. S. 495–502. Springer. 2002.

[BLHL01] Berners-Lee, T., Hendler, J., und Lassila, O.: The Semantic Web. *Scientific American*. May 2001.

[BvHH+04] Bechhofer, S., van Hermelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., und Stein, L. A. W3C: OWL Web Ontology Language – Reference. `http://www.w3.org/TR/owl-ref/`. February 2004.

[CGP00] Corcho, O. und Gómez-Pérez, A.: A roadmap to ontology specification languages. In: *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*. S. 80–96. Springer-Verlag. 2000.

[daV] `http://www.informatik.uni-bremen.de/~davinci/`.

[HET] Heterogeneous tool set (Hets) web site. `http://www.tzi.de/cofi/hets`.

[KBHL+03] Krieg-Brückner, B., Hutter, D., Lindow, A., Lüth, C., Mahnke, A., Melis, E., Meier, P., Poetzsch-Heffter, A., Roggenbach, M., Russell, G., Smaus, J.-G., und Wirsing, M.: Multimedia instruction in safe and secure systems. In: *Recent Trends in Algebraic Development Techniques*. volume 2755 of *Lecture Notes in Computer Science*. S. 82–117. Springer. 2003.

[LMKB] Lüttich, K., Mossakowski, T., und Krieg-Brückner, B.: Ontologies for the Semantic Web in CASL. *17th Int. Workshop on Algebraic Development Techniques*. Submitted for publication.

[MMi] `http://www.mmiss.de`.

[Mo04] Mosses, P. D. (Hrsg.): CASL *Reference Manual*. volume 2960 of *Lecture Notes in Computer Science*. Springer. 2004.

[UG96] Uschold, M. und Grüninger, M.: Ontologies: Principles, methods and applications. *Knowledge Engineering Review*. 11(2):93–155. 1996.