Combining Source, Content, Presentation, Narration, and Relational Representation

Fulya Horozal, Alin Iacob, Constantin Jucovschi,
Michael Kohlhase, Florian Rabe
{f.horozal,a.iacob,c.jucovschi,m.kohlhase,f.rabe}@jacobs-university.de

Computer Science, Jacobs University, Bremen

Abstract. In this paper, we try to bridge the gap between different dimensions/incarnations of mathematical knowledge: MKM representation formats (content), their human-oriented languages (source, presentation), their narrative linearizations (narration), and relational presentations used in the semantic web. The central idea is to transport solutions from software engineering to MKM regarding the parallel interlinked maintenance of the different incarnations. We show how the integration of these incarnations can be utilized to enrich the authoring and viewing processes, and we evaluate our infrastructure on the LATIN Logic Atlas, a modular library of logic formalizations, and a set of computer science lecture notes written in sTeX – a modular, semantic variant of LaTeX.

1 Introduction

Within the Mathematical Knowledge Management (MKM) community, XML-based content representations of mathematical formulae and knowledge have been developed that are optimized for machine processing. They serve as archiving formats, make mathematical software systems and services interoperable and allow to develop structural services like search, documentation, and navigation that are independent of mathematical foundations and logics.

However, these formats are by their nature inappropriate for human processing. Therefore, community uses languages that are less verbose, more mnemonic, and often optimized for a specific domain for authoring. Such human-oriented languages (we call them source languages) are converted — via a complex compilation process — into the content representations for interaction with MKM services, ideally without the user ever seeing them. In addition, we have designed presentation-oriented languages that permit an enriched reading experience compared to the source language.

This situation is similar to software engineering, where programmers write code, run the compiled executables, build HTML-based API documentations, but expect, *e.g.*, the documentation and the results of debugging services in terms of the sources. In software engineering, scalable solutions for this problem have been developed and applied successfully, which we want to transfer to MKM.

The work described here originates from our work on two large collections of mathematical documents: our LATIN Logic Atlas [KMR09] formalized in the logical framework LF; and our General Computer Science lecture notes written in sTEX [Koh08] – a modular, semantic variant of LaTeX. Despite their different flavor, both collections agree in some key aspects: They are large, highly structured, and extensively inter-connected, and both authoring and reading call for machine support.

Moreover, they must be frequently converted between representation dimensions optimized for different purposes: a human-friendly input representation (source), a machine-understandable content markup (content), interactive documents for an added-value reading experience (presentation-content parallel markup), a linearized structure for teaching and publication (narration), and a network of linked data items for indexing and integration with the semantic web (relational).

Therefore, we see a need for a representation and distribution format that specifies these dimension and enables their seamless integration. In this paper, we present one such format and evaluate it within our system and document collections. In Sect. 2, we first give an overview over the document collections focusing on the challenges they present to knowledge management. Then we design our representation format in Sect. 3. In Sect. 4 and 5, we show how we leverage this methodology in the authoring and the viewing process.

2 Structured Document Collections

2.1 The LATIN Logic Atlas

The LATIN Logic Atlas [KMR09] is a library of formalizations of logics and related formal systems as well as translations between them. It is intended as a reference and documentation platform for logics commonly used in mathematics and computer science. It uses a foundationally unconstrained logical framework based on modular LF and its Twelf implementation [HHP93,PS99,RS09] and focuses on modularity and extensibility.

The knowledge in the Logic Atlas is organized as a graph of LF signatures and signature morphisms between them. The latter are split into inheritance translations (inclusions/imports) and representation theorems, which have to be proved. It contains formalizations of type theories, set theories, logics and mathematics. Among the logic formalizations in the Logic Atlas are, for example, the formalizations of propositional (PL), first (FOL) and higher-order logic (HOL), sorted ($SFOL$) and dependent first-order logic ($DFOL$), description logics (DL), modal (ML) and common logic (CL) as illustrated in Fig. 1. Single arrows ($\rightarrow$) in this diagram denote translations between formalizations and hooked arrows ($\hookrightarrow$) denote imports.

All logics are designed modularly formed from orthogonal theories for individual connectives, quantifiers, and axioms. For example, the classical $\wedge$ connective is only declared once in the whole Logic Atlas, and the axiom of excluded middle and its consequences reside in a separate signature.

$$\begin{tikzcd}
 & & PL \arrow[dl, hook] \arrow[dd, hook] \arrow[dr, hook] \arrow[drr, hook] & & \\
 & ML \arrow[dr] & & SFOL \arrow[r] \arrow[dl, bend left] \arrow[dd] & DFOL \arrow[dddl] \\
DL \arrow[rr] \arrow[ddr] & & FOL \arrow[ur, bend left] \arrow[dd] \arrow[dddr, hook] & & \\
 & & & HOL \arrow[dd] & \\
 & OWL \arrow[uur] & CL & & \\
 & & Isabelle \arrow[r] & ZFC \arrow[r] & Mizar
\end{tikzcd}$$

Fig. 1. Logical Languages in the LATIN Logic Atlas

As a running example, we introduce a very simple fragment of the formalization of the syntax of propositional logic.

Example 1 (Propositional Logic) The formalization of propositional logic syntax consists of the LF signatures illustrated in Fig. 2. We focus on the structural aspects and omit the details of LF. We define four signatures living in two different namespaces. `BASE` declares a symbol for the type of propositions. It is imported into `CONJ` and `IMP` which declare conjunction and implication, respectively, and these are imported to `PROP`.

```
%namespace = "http://cds.omdoc.org/logics"
%sig BASE = {
 o : type. %% Type of propositions
}

%namespace = "http://cds.omdoc.org/logics/propositional"
%sig CONJ = {%include BASE. ...}
%sig IMP  = {%include BASE. ...}
%sig PROP = {%include CONJ. %include  IMP.}
```

Fig. 2. Formalization of Propositional Logic Syntax in Twelf

Overall, the Logic Atlas contains over 750 LF signatures, and their highly modular structure yields a large number of inheritance edges. Additionally, it contains over 500 signature morphisms that connect the nodes. This leads to a highly interlinked non-linear structure of the Logic Atlas. Moreover, it is designed highly collaboratively with strong interdependence between the developers. Therefore, it leads to a number of MKM challenges.

Firstly, the LF modules are distributed over files and these files over directories. This structure is semantically transparent because all references to modules are made by URIs. The URIs are themselves hierarchical grouping the modules into nested namespaces. It is desirable that these namespaces do not have to correspond to source files or directories. Therefore, a mapping between URIs and URLs has to be maintained and be accessible to all systems.

Secondly, LF encodings are usually highly concise, and a complex type reconstruction process is needed to infer additional information. Twelf can generate an OMDoc-based content representation of the sources, but this semantically enriched version is currently not fed back into the editing process.

Thirdly, encodings in LF are often difficult to read for anybody but the author. In a collaborative setting, it is desirable to interact with the Logic Atlas not only through the LF source syntax but also through browsable, cross-referenced XHMTL+MathML. These should be interactive and for example permit looking up the definition of a symbol or displaying the reconstructed type of a variable. While we have presented such an interface in [GLR09] already, the systematic integration into the authoring process, where the state of the art is a text editor, has so far been lacking.

Finally, to understand and navigate the Logic Atlas, it is necessary to visualize its multi-graph structure in an interactive graphical interface.

2.2 Computer Science Lecture Notes in Planetary

The GenCS corpus consists of the course notes and problems of a two-semester introductory course in Computer Science [Gen11] held at Jacobs University by one of the authors in the last eight years. The course notes currently comprise 300 pages with over 500 slides organized in over 800 files; they are accompanied by a database of more than 1000 homework/exam problems. All course materials are authored and maintained in the sTeX format [Koh08], a modular, semantic variant of LaTeX that shares the information model with OMDoc; see Fig. 3 for an example.

```
\begin{module}[id=trees]
  \symdef[name=tdepth]{tdepthFN}{\text{dp}}
  \symdef{tdepth}[1]{\prefix\tdepthFN{#1}}
  \begin{definition}[id=tree-depth.def]
    Let $\defeq{T}{\tup{V,E}}$ be tree, then the {\definiendum [tree-depth]{depth}}
    $\tdepth{v}$ of a node $\inset{v}{V}$ is defined recursively: $\tdepth{r}=0$ for
    the root $r$ of $T$ and $\tdepth(w)=1+\tdepth(w)$ if $\inset{\tup{v,w}}E$.
 \end{definition}
 ...
\end{module}

\begin{module}[id=binary-trees]
  \importmodule[\KWARCslides{graphs-trees/en/trees}]{trees}
  ...
  \begin{definition}[id=binary-tree.def,title=Binary Tree]
    A \definiendum[binary-tree]{binary tree} is a \termref[cd=trees,name=tree]{tree}
    where all \termref[cd=graphs-intro,name=node]{nodes}
    have \termref[cd=graphs-intro,name=out-degree]{out-degree} 2 or 0.
  \end{definition}
  ...
\end{module}
```

Fig. 3. Semiformalization of two course modules

In our nomenclature, sTeX is used as a source language that is transformed into OMDoc via the LaTeXML daemon [GSK11]. For debugging and high-quality print the sTeX sources can also be typeset via `pdflatex`, just as ordinary LaTeX documents. The encoding makes central use of the modularity afforded by the theory graph approach; knowledge units like slides are encoded as "modules" (theories in OMDoc) and are interconnected by theory morphisms (module imports). Modules also introduce concepts via `\definiendum` and semantic macros via `\symdef`, these are inherited via the module import relation.

The challenges discussed for the LATIN Logic Atlas apply to the GenCS corpus and the Planetary system analogously. Moreover: (i) The development of the corpus proceeds along two workflows: drafting of the content via the classical `pdflatex` conversion (in a working copy via `make`) and via the web interface in the Planetary system. (ii) The LaTeXML conversion process needs intermediate files (the sTeX module signatures), all of which have to be kept in sync. (iii) Importing legacy sTeX materials into the Planetary system is nontrivial. For example, the contents of `http://planetmath.org` contain many (semantic) cross-links and metadata. Rather than a set of individualized import scripts at the level of the repositories and databases underlying Planetary, it is desirable to have a file (set) that can be imported uniformly.

3 A Multi-Dimensional Knowledge Representation

3.1 Dimensions of Knowledge

In order to address the knowledge management challenges outlined above, we devise a methodology that permits the parallel maintenance of the orthogonal dimensions of the knowledge contained in a collection of mathematical documents. It is based on two key concepts: (i) a hierarchic organization of dimensions and knowledge items in a file-system-like manner, and (ii) the use of MMT URIs [RK11] as a standardized way to interlink both between different knowledge items and between the different dimensions of the same knowledge item.

The MMT URI of a toplevel knowledge item is of the form $g?M$ where g is the namespace and M the module name. Namespaces are URIs of the form ⟪`scheme`⟫`://[`⟪`userinfo`⟫`@]`$D_1 \ldots . D_m$`[:`⟪`port`⟫`]/`S_1`/.../`S_n where the D_i are domain labels and the S_i are path segments. Consequently, $g?M$ is a well-formed URI as well. ⟪userinfo⟫, and ⟪port⟫ are optional, and ⟪userinfo⟫, ⟪scheme⟫, and ⟪port⟫ are only permitted so that users can form URIs that double as URLs — MMT URIs differing only in the scheme, userinfo, or port are considered equal.

We arrange a collection of mathematical documents as a folder containing the following subfolders, all of which are optional:

`source` contains the source files of a project. This folder does not have a predefined structure.

`content` contains a semantically marked up representation of the source files in the OMDoc format. Every namespace is stored in one file whose path is determined by its URI. Modules with namespace $D_1. \ldots .D_m/S_1/.../S_n$ reside in an OMDoc file with path `content/`D_m`/.../`D_1`/`S_1`/.../`S_n`.omdoc`. Each module carries an attribute `source="/PATH?colB:lineB-colE:lineE"` giving its physical location as a URL. Here `PATH` is the path to the containing file in the source, and `colB`, `lineB`, `colE`, and `lineE` give the begin/end column/line information.

`presentation` contains the presentation of the source files in the XHTML+MathML format with JOBAD annotations [GLR09]. It has the same file structure as the folder `content`. The files contain XHTML elements whose body has one child for every contained module. Each of these module has the attribute `jobad:href="URI"` giving its MMT URI.

`narration` contains an arbitrary collection of narratively structured documents. These are OMDoc files that contain narrative content such as sectioning and transitions, but no modules. Instead they contain reference elements of the form `<mref target="MMTURI"/>` that refer to MMT modules. It is common but not necessary that these modules are present in the `content` folder.

`relational` contains two files containing an RDF-style relational representation of the content according to the MMT ontology. Both are in XML format with toplevel element `mmtabox` and a number of children. In `individuals.abox`, the children give instances of unary predicates such as `<individual type="IsTheory" uri="MMTURI" source="PATH"/>`. In `relations.abox`, the children give instances of binary predicates such as `<relation subject="MMTURI1" predicate="ImportsFrom" object= "MMTURI2" source="PATH"/>`. Usually, the knowledge items occurring in unary predicates or as the subject of a binary predicate are present in the content. However, the object of a binary predicate is often not present, namely when a theory imports a remote theory. In both cases, we use an attribute `source` to indicate the source that induced the entry; this is important for change management when one of the source files was changed.

Example 2 (Continuing Ex. 1)
The directory structure for the signatures from Ex. 1 is given in Fig. 4 using a root folder named `propositional-syntax`. Here we assume that the subfolder `source` contains the Twelf source files `base.elf` which contains the signature `BASE`, `modules.elf` which contains `CONJ` and `IMP`, and `prop.elf` which contains `PROP`.

Based on the MMT URIs of the signatures in the source files, their content representation is given as follows. The signature `BASE` has the MMT URI `http://cds.omdoc.org/logics?BASE`. The other signatures have MMT URIs such as `http://cds.omdoc.org/logics/propositional/syntax?CONJ`. The content representation of the signature `BASE` is given in the OMDoc file `content/org/omdoc/cds/logics.omdoc`. The other content representations reside in the file `content/org/omdoc/cds/logics/propositional/syntax.omdoc`.

```
propositional-syntax
├ source
│ ├ base.elf
│ ├ modules.elf
│ └ prop.elf
├ content
│ └ org
│   └ omdoc
│     └ cds
│       ├ logics.omdoc
│       └ logics
│         └ propositional
│           └ syntax.omdoc
├ presentation
│ └ org
│   └ omdoc
│     └ cds
│       ├ logics.xhtml
│       └ logics
│         └ propositional
│           └ syntax.xhtml
├ narration
│ ├ base.omdoc
│ ├ modules.omdoc
│ └ prop.omdoc
└ relational
  ├ individual.abox
  └ relations.abox
```

Fig. 4. Folder Structure

The subfolder `presentation` contains the respective XHTML files, `logics.xhtml` and `syntax.xhmtl`. All files in Fig. 4 can be downloaded at `https://svn.kwarc.info/repos/twelf/projects/propositional-syntax`.

Our methodology integrates various powerful conceptual distinctions that have been developed in the past. Firstly, our distinction between the source and the content representation corresponds to the distinction between source and binary in software engineering. Moreover, our directory structure is inspired by software projects, such as in Java programming. In particular, the use of URIs to identify content (binary) items corresponds to identifiers for Java classes. Therefore, existing workflows and implementations from software engineering can be easily adapted, for example in the use of project-based IDEs (see Sect. 4).

Secondly, the distinction between content and presentation has been well studied in the MKM community and standardized in MathML [ABC+03]. In particular, the cross-references from presentation to content correspond to the interlinking of content and presentation in the parallel markup employed in MathML, which we here extend to the level of document collections.

Thirdly, the distinction between content and narrative structure was already recognized in the OMDoc format. The general intuition there is that narrative structures are "presentations" at the discourse level. But in contrast to the formula level, presentations cannot be specified and managed via notation definitions. Instead we add narrative document structure fragments, i.e. document-structured objects that contain references to the content representations and transition texts as lightweight structures to the content commons; see [Mül10] for details and further references.

Finally, the distinction between tree-structured content representation and the relational representation corresponds to the practice of the semantic web where RDF triples are used to represent knowledge as a network of linked data.

3.2 A Mathematical Archive Format

We will now follow the parallelism to software engineering developed in the previous section: We introduce mathematical archives — `mar` files — that correspond to Java archives, i.e., `jar` files [Ora]. We define a mathematical archive to be a zip file that contains the directory structure developed in Sect. 3.1. By packaging all knowledge dimensions in a single archive, we obtain a convenient and lightweight way of distributing multi-dimensional collections of interlinked documents.

To address into the content of a `mar` archive, we also define the following URL scheme: Given a `mar` whose URL is `file:/A` and which contains the source file `source/S`, then the URL `mar:/A/S` resolves to that source file. We define the URL `mar:/A/S?Pos` accordingly if `Pos` is the position of a module given by its line/column as above.

Similar to the compilation and building process that is used to create `jar` files, we have implemented a building process for `mar` files. It consists of three stages. The first stage (compilation) depends on the source language and produce one OMDoc file for every source file whose internal structure corresponds to the source file. This is implemented in close connection with dedicated tools for the source language. In particular, we have implemented a translation from LF to OMDoc as part of the Twelf implementation of LF [PS99,RS09]. Moreover, we have implemented a translation from sTeX to OMDoc based on the LaTeXML daemon [GSK11].

The second stage (building) is generic and produces the remaining knowledge dimensions from the OMDoc representation. In particular, it decomposes the OMDoc documents into modules and reassembles them according to their namespaces to obtain the content representation. The narrative dimension is obtained from the initial OMDoc representation by replacing all modules with references to the respective content item. We have implemented this as a part of the existing MMT API [RK11]. Finally, the API already includes a rendering engine that we use to produce the presentation and the relational representation.

Then the third stage (packaging) collects all folders in a zip archive. For LF, we integrate all three stages into a flexible command line application.

Example 3 (Continuing Ex. 2) The mathematical archive file for the running example can be obtained at `https://svn.kwarc.info/repos/twelf/projects/propositional-syntax.mar`.

3.3 Catalog Services

The use of URIs as knowledge identifiers (rather than URLs) is crucial in order to permit collaborative authoring and convenient distribution of knowledge. However, it requires a catalog that translates an MMT URIs to the physical location, given by a URL, of a resource. Typical URLs are those in a file system, in a mathematical archive, or a remote or local repository. It is trivial to build the catalog if the knowledge is already present in content form where locations are derived from the URI.

But the catalog is already needed during the compilation process: For example, if a theory imports another theory, it refers to it by its MMT URI. Consequently, the compilation tool must already be aware of the URI-to-URL mapping before the content has been produced. However, the compilation tool is typically a dedicated legacy system that natively operates on URLs already and does not even recognize URIs. This is the case for both Twelf and LaTeX.

Therefore, we have implemented standalone catalog services for these two tools and integrated them with the respective system. In the case of Twelf, the catalog maintains a list of local directories, files, and `mar` archives that it watches. It parses them whenever they change and creates the URI-URL mapping. When Twelf encounters a URI, it asks the catalog via HTTP for the URL. This parser only parses the outer syntax that is necessary to obtain the structure of the source file; it is implemented generically so that it can be easily adapted to other formal declarative languages. In this sense, it is similar to the HTTP Getter service of the HELM library [APSC$^+$03].

An additional strength of this catalog is that it can also handle ill-formed source representations that commonly arise during authoring. Moreover, we also use the catalog to obtain the line/column locations of the modules inside the source files so that the content-to-source references can be added to the content files.

In the case of sTeX, a poor man's catalog services is implemented directly in TeX: the base URIs of the GenCS knowledge collection (see `\KWARCslides` in Fig 3) is specified by a `\defpath` statement in the document preamble and can be used in the `\importmodule` macros. The `module` environments induce internal TeX structures that store information about the imports (`\importmodule`) structure and semantic macros (`\symdef`), therefore these three sTeX primitives have to be read whenever a module is imported. To get around difficulties with selective input in TeX, the sTeX build process excerpts a sTeX signature module `⟪module⟫.sms` from any module `⟪module⟫.tex`. So `\importmodule⟪module⟫.sms` simply reads `⟪module⟫.sms`.

4 The Author's Perspective

The translation from source to a content-like representation has been well-understood. For languages like LF, it takes the form of a parsing and type reconstruction process that transforms external to internal syntax. The translation from internal syntax to an OMDoc-based content representation is conceptually straightforward. However, it is a hard problem to use the content representation to give the author feedback about the document she is currently editing. A more powerful solution is possible if we always produce all knowledge dimensions using the compilation and building process as described in Sect. 3.2. Then generic services can be implemented easily, each of them based on the most suitable dimension, and we give a few examples in Sect. 4.2.

Note that this is not an efficiency problem: Typically the author only works on a few files that can be compiled constantly. It is even realistic to hold all dimensions in memory. The main problem is an architectural one, which is solved by our multi-dimensional representation. Once this architecture is set up and made available to IDE developers, it is very easy for them to quickly produce powerful generic services.

4.1 Multi-Dimensional Knowledge in an IDE

In previous work, we have already presented an example of a semantic IDE [JK10] based on Eclipse. We can now strengthen it significantly by basing it on our multi-dimensional representation. Inspired by the project metaphor from software engineering, we introduce the notion of a mathematical project in Eclipse.

A mathematical project consists of a folder containing the subfolder from Sect. 3.1. The author works on a set of source files in the `source` directory. Moreover, the project maintains a mathpath (named in analogy to Java's classpath) that provides a set of `mar` archives that the user wishes to include.

The IDE offers the build functionality that runs the compilation and building processes described in Sect. 3.2 to generate the other dimensions from the source dimension. The key requirement here is to gracefully degrade in the presence of errors in the source file. Therefore, we provide an adaptive parser component that consists of three levels:

The **regex level** uses regular expressions to spot important structural properties in the document (e.g. the namespace and signature declarations in the case of LF). This compilation level never fails, and its result is an OMDoc file that contains only the spotted structures and lacks any additional information of the content.

The **CFG parser level** uses a simple context-free grammar to parse the source. It is able to spot more complicated structures such as comments and nested modules and can be implemented very easily within Eclipse. Like the previous level, it produces an approximate OMDoc file, but contrary to the previous level, it may find syntax errors that are then displayed to the user.

The **full parser level** uses the dedicated tool (Twelf or LaTeX). The resulting OMDoc file includes the full content representation. In particular, in the case of Twelf, it contains all reconstructed types and implicit arguments. However, it may fail in the case of ill-typed input.

The adaptive parser component tries all parser in stages and retains the file returned by the last stage that succeeds. This file is then used as the input to produce the remaining dimensions.

4.2 Added-Value Services

In this section we present several services that aim at supporting the authoring process. We analyze each of these services and show that they can can be efficiently implemented by using one or several dimensions of knowledge.

project explorer is a widget giving an integrated view on a project's content by abstracting from the file system location where the sources are are defined. It groups objects by their content location, i.e., their MMT URI. To implement this widget, we populate the non-leaf nodes of the tree from the directory structure of the content dimension. The leaf nodes are generated by running simple XPath queries on the OMDoc files.

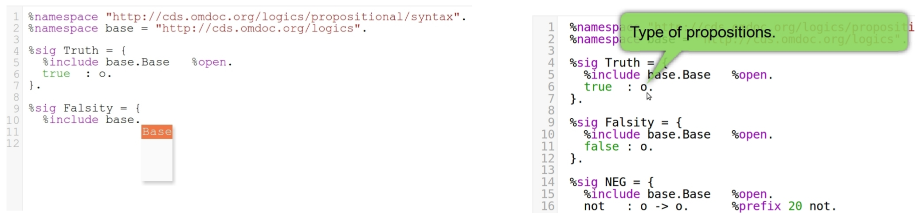
outline view is a source level widget which visualizes the main structural components. For LF, these include definitions of signatures and namespaces as well as constant declarations within signatures. Double-clicking on any such structural components opens the place in the source code where the component is defined. Alternatively, the corresponding presentation can be opened.

autocompletion assists the user with getting location and context specific suggestions, e.g., listing declarations available in a namespace. Fig. 5a) shows an example. Note how the namespace prefix `base` is declared to point to a certain namespace, and the autocompletion suggests only signatures names declared in that namespace. The implementation of this feature requires information about the context where autocompletion is requested, which is obtained from the interlinked source and content dimensions. Moreover, it needs the content dimension to compute all possible completions. In more complicated scenarios, it can also use the relational dimension to compute the possible completions using the relational queries.

hover overlay is a feature that shows in-place meta-data about elements at the position of the mouse cursor such as the full URIs of a symbol, its type or definitions, a comment, or inferred types of variables. Fig. 5b) shows an example. The displayed information is retrieved from the content dimension. It is also possible to display the information using the presentation dimension.

definition/reference search makes it easy for a user to find where a certain item is defined or used. Although the features require different user interfaces the functionality is very similar, namely, finding relations. Just like in the hover overlay feature, one first finds the right item in the content representation and then use the relation dimension to find the requested item(s).

theory-graph display provides a graphical visualization of the relations among knowledge items. To implement this feature we apply a filter on the multigraph from the relations dimension and use 3rd party software to render it.



```
%namespace "http://cds.omdoc.org/logics/propositional/syntax".
%namespace base = "http://cds.omdoc.org/logics".

%sig Truth = {
  %include base.Base   %open.
  true  : o.
}.

%sig Falsity = {
  %include base.
               Base
```

Type of propositions.

```
%namespace "http://cds.omdoc.org/logics/propositi
%namespace base = "http://cds.omdoc.org/logics".

%sig Truth = {
  %include base.Base   %open.
  true  : o.
}.

%sig Falsity = {
  %include base.Base   %open.
  false : o.
}.

%sig NEG = {
  %include base.Base   %open.
  not   : o -> o.      %prefix 20 not.
```

Fig. 5. a) Context aware Auto-Completion b) Metadata information on hover

5 The Reader's Perspective

We have developed the Planetary system [KDG+11] as the reader's complement to our IDE. Planetary is a Web 3.0 system[1] for semantically annotated document collections in Science, Technology, Engineering and Mathematics (STEM). In our approach, *documents published in the Planetary system become flexible, adaptive interfaces to a content commons* of domain objects, context, and their relations.

We call this framework the **Active Documents Paradigm** (ADP), since documents can also actively adapt to user preferences and environment rather than only executing services upon user request. Our framework is based on *semantically annotated documents* together with semantic background ontologies (which we call the **content commons**). This information can then be used by user-visible, semantic services like program (fragment) execution, compu-

[1] We adopt the nomenclature where Web 3.0 stands for extension of the Social Web with Semantic Web/Linked Open Data technologies.

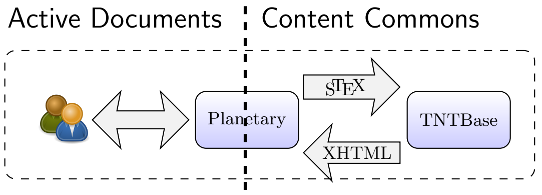


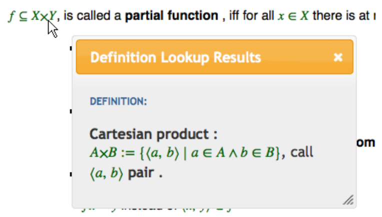
Fig. 6. The Active Documents Architecture

tation, visualization, navigation, information aggregation and information retrieval [GLR09].

The Planetary system directly uses all five incarnations/dimensions of mathematical knowledge specified in Sect 3.1. In the `content` incarnation, Planetary uses OMDoc for representing content modules, but authors create and maintain these using sTeX in the `source` dimension and the readers interact with the active documents encoded as dynamic XHTML+MathML+RDFa (the `source` incarnation of the material). We use the LaTeXML daemon [Mil,GSK11] for the transformation from sTeX to OMDoc, this is run on every change to the sTeX sources. The basic presentation process [KMR08] for the OMDoc content modules is provided by the TNTBase system [ZK09b].

But Planetary also uses the `narrative` dimension: content modules are used not only for representing mathematical theories, but also for document structures: **narrative modules** consist of a mixture of sectional markup, inclusion references, and narrative texts that provide transitions (narrative glue) between the other objects. Graphs of narrative modules whose edges are the inclusion references constitute the content representations of document fragments, documents, and document collections in the Planetary system, wich generates active documents from them. It uses a process of separate compilation and dynamic linking to equip them with document (collection)-level features like content tables, indexes, section numbering and inter-module cross-references; see [DGK+11] for details.

As the active documents use identifiers that are relative to the base URI of the Planetary instance, whereas the content commons uses MMT URIs, the **semantic publishing map** which maintains the correspondence between these is a central, persistent data structure maintained by the Planetary system. This is one instance of the `relational` dimension, another is used in the For instance, the RDFa embedded in the presentation of a formula (and represented in the linking part of the math archive) can be used for definition lookup as shown on the left. Actually the realization of the definition lookup service involves `presentation` (where the service is embedded) and `content` (from which the definition is fetched to be presented by the service) incarnations as well.

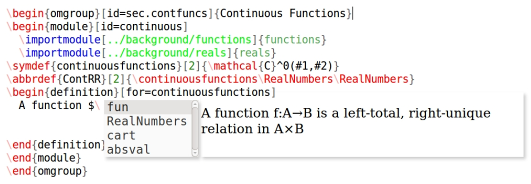

$f \subseteq X \times Y$, is called a **partial function** , iff for all $x \in X$ there is at

Definition Lookup Results

DEFINITION:

Cartesian product :
$A \times B := \{\langle a, b\rangle \mid a \in A \wedge b \in B\}$, call $\langle a, b\rangle$ pair .

In the future we even want to combine this with the `source` dimension by combining it with a `\symdef`-look service that makes editing easier. This can be thought of as a presentation-triggered complement to the editor-based service on the right that looks up `\symdef`s by their definienda.



```
\begin{omgroup}[id=sec.contfuncs]{Continuous Functions}
\begin{module}[id=continuous]
  \importmodule[../background/functions]{functions}
  \importmodule[../background/reals]{reals}
\symdef{continuousfunctions}[2]{\mathcal{C}^0(#1,#2)}
\abbrdef{ContRR}[2]{\continuousfunctions\RealNumbers\RealNumbers}
\begin{definition}[for=continuousfunctions]
  A function $\
\end{definition}
\end{module}
\end{omgroup}
```

A function f:A→B is a left-total, right-unique relation in A×B

6 Discussion

Our approach of folder-based projects is inspired by practices from software engineering. However, while it is appealing to port these successful techniques to mathematics, mathematical knowledge has more dimensions with a more complex inter-dependence than software. In particular, where software IDEs can be optimized for a work flow of generating either binaries or documentation from the sources, the build process for our mathematical projects must employ multiple stages. Moreover, the narrative dimension that is central to mathematics has no direct analog in software engineering.

Therefore, we cannot expect to be able to adapt all software engineering practices directly. We have been very successful with authoring-based services, where the IDE can utilize the various dimensions for added-value services. Contrary to software projects, this often requires the integration of information from multiple dimensions. Archive-based packaging and distribution work flows can also be adapted successfully, and we expect that the same holds for dependency management and — along the lines of [DGK$^+$11] — linking.

More difficulties are posed by authoring work flows that depend on the mathematical semantics of the content such as generating content (e.g., theorem proving) or highlighting errors while typing. The necessary integration of the user interface with a deduction system is very difficult, and only few systems realize it. For example, the Agda [Nor05] emacs mode can follow cross-references and show reconstructed types of missing terms. The Isabelle jEdit user interface [Wen10] can follow cross-references and show tooltips derived from the static analysis. In general, we believe that a data model like ours is one ingredient in the design of a comprehensive and scalable solution for this problem.

Various domain-specific systems have adopted ad hoc implementations of multi-dimensional representations. For example, the Mizar workflow [TB85] can be understood as a build process that transforms the source dimension into various content and relational dimensions, which are then used to assemble an enriched content (.abs) and a presentation dimension (.html). Similar to our format, the dimensions are spread over several folders. The Archive of Formal Proofs [KNe04] uses a source and two narrative dimensions (proof outline and document in pdf format) along with an HTML-based presentation dimension. The dimensions are not systematically separated into folders, and projects can be distributed as tar.gz archives.

"Mathematica Applications" distinguish dimensions for mathematical algorithms (kernel), notations (notebooks with palettes front-end), and narration (narrative notebooks in the documentation). They also allow pre-generating XHTML representations of the documentation (without giving it a separate dimension). The documentation notebooks can contain (non-semantic) links. The algorithm aspect — that is at the center of Mathematica — is missing from our math archives as content representations of mathematical computation are largely unstandardized in MKM. Conversely, the separation of the source, content, and presentation dimensions are missing from Mathematica Applications because they are conflated.

We have so far evaluated our infrastructure using our own corpora and systems: two corpora in two source languages with the respective compilation processes (LATIN in Twelf and lecture notes in sTEX), our semantic IDE [JK10], and our semantic publishing system [KDG+11]. Despite the close collaboration of the respective developers and users, we have found that a representation format like ours is a key prerequisite for scalable system integration: We use a mathematical archive as a central knowledge store around which all systems are arranged — with each system producing and/or using some of the knowledge dimensions. For example, the navigation of the LATIN Logic Atlas involves four distinct processes that can only be connected through our format: converting *source* to *content*, generating *relational* from *content*, using *relational* to compute and display the graph, using the cross-references to switch focus in the *source* editor based on the user interaction with the graph.

Our choice of dimensions is not final — we have focused on those that we found most important. Other dimensions can be added to our format easily, for example, discussions and reviews or additional presentation dimensions like pdf. Planetary already allows discussions of individual content items, and sTEX already permits the creation of pdf. A more drastic extension is the use of multiple source dimensions, e.g., for a specification in sTEX, a formalization in a proof assistant, and algorithms implemented in a programming language. Here the MMT URIs will be particularly useful to realize cross-references across dimensions.

Our emphasis on file systems and zip archives is not without alternative. In fact, even our own LATIN Logic Atlas is stored in a custom XML database with SVN interface [ZK09a]; Planetary uses a custom database as well. However, a file system-based infrastructure is the easiest way to specify a multi-dimensional representation format and represents the lowest hurdle for system integration. Moreover, we believe that more sophisticated mathematical databases should always be able to import and export knowledge in a format like ours.

7 Conclusion

We have presented an infrastructure for creating, storing, managing, and distributing mathematical knowledge in various pragmatic forms. We have identified five aspects that have to be taken into account here: (i) human-oriented *source languages* for efficient editing of content, (ii) the modular *content representation* that has been the focus of attention in MKM so far, (iii) *interactive presentations* of the content for viewing, navigation, and interaction, (iv) *narrative structures* that allow binding the content modules into self-contained documents that can be read linearly, and (v) *relational structures* that cross-link all these aspects and permit keeping them in sync.

These aspects are typically handled by very different systems, which makes system integration difficult, often leading to ad-hoc integration solutions. By designing a flexible knowledge representation format featuring multiple interconnected dimensions, we obtained a scalable, well-specified basis for system integration. We have evaluated them from the authoring and reading perspectives using two large structured corpora of mathematical knowledge.

In the future, we want to combine these systems and perspectives more tightly. For example, we could use Planetary to discuss and review logic formalizations in Twelf, or write papers about the formalizations in sTeX. This should not pose any fundamental problems as the surface languages are interoperable by virtue of having the same, very general data model: the OMDoc ontology. By the same token we want to add additional surface languages and presentation targets that allow to include other user groups. High-profile examples include the Mizar Mathematical Language and Isabelle/Isar.

References

[ABC$^+$03] R. Ausbrooks, S. Buswell, D. Carlisle, S. Dalmas, S. Devitt, A. Diaz, M. Froumentin, R. Hunter, P. Ion, M. Kohlhase, R. Miner, N. Poppelier, B. Smith, N. Soiffer, R. Sutor, and S. Watt. Mathematical Markup Language (MathML) Version 2.0 (second edition). Technical report, World Wide Web Consortium, 2003. See `http://www.w3.org/TR/MathML2`.

[APSC$^+$03] Andrea Asperti, Luca Padovani, Claudio Sacerdoti Coen, Ferruccio Guidi, and Irene Schena. Mathematical knowledge management in HELM. *Annals of Mathematics and Artificial Intelligence, Special Issue on Mathematical Knowledge Management, Kluwer Academic Publishers*, 38(1–3):27–46, May 2003.

[DGK$^+$11] Catalin David, Deyan Ginev, Michael Kohlhase, Bogdan Matican, and Stefan Mirea. A framework for modular semantic publishing with separate compilation and dynamic linking. In Alexander García Castro, Christoph Lange, Evan Sandhaus, and Anita de Waard, editors, *1st Workshop on Semantic Publication (SePublica)*, 2011. accepted.

[Gen11] General Computer Science: GenCS I/II Lecture Notes. `http://gencs.kwarc.info/book/1`, 2011.

[GLR09] J. Gičeva, C. Lange, and F. Rabe. Integrating Web Services into Active Mathematical Documents. In J. Carette and L. Dixon and C. Sacerdoti Coen and S. Watt, editor, *Intelligent Computer Mathematics*, volume 5625 of *Lecture Notes in Computer Science*, pages 279–293. Springer, 2009.

[GSK11] Deyan Ginev, Heinrich Stamerjohanns, and Michael Kohlhase. The LaTeXML daemon: Editable math on the collaborative web. In *Intelligent Computer Mathematics*, 2011. accepted.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[JK10] C. Jucovschi and M. Kohlhase. sTeXIDE: An Integrated Development Environment for sTeX Collections. In S. Autexier, J. Calmet, D. Delahaye, P. Ion, L. Rideau, R. Rioboo, and A. Sexton, editors, *Intelligent Computer Mathematics*, number 6167 in Lecture Notes in Artificial Intelligence. Springer, 2010.

[KDG$^+$11] M. Kohlhase, C. David, D. Ginev, A. Kohlhase, C. Lange, B. Matican, S. Mirea, and V. Zholudev. The Planetary System: Web 3.0 & Active Documents for STEM. 2011. To appear at ICCS 2011 (finalist at the Executable Papers Challenge), see `https://svn.mathweb.org/repos/planetary/doc/epc11/paper.pdf`.

[KMR08] Michael Kohlhase, Christine Müller, and Florian Rabe. Notations for living mathematical documents. In Serge Autexier, John Campbell, Julio Rubio, Volker Sorge, Masakazu Suzuki, and Freek Wiedijk, editors, *Intelligent Computer Mathematics*, number 5144 in LNAI, pages 504–519. Springer Verlag, 2008.

[KMR09] M. Kohlhase, T. Mossakowski, and F. Rabe. The LATIN Project, 2009. See `https://trac.omdoc.org/LATIN/`.

[KNe04] G. Klein, T. Nipkow, and L. Paulson (eds.). Archive of Formal Proofs, 2004. http://afp.sourceforge.net/.

[Koh08] Michael Kohlhase. Using LaTeX as a semantic markup format. *Mathematics in Computer Science*, 2(2):279–304, 2008.

[Mil] Bruce Miller. `LaTeXML`: A LaTeX to XML converter.

[Mül10] Christine Müller. *Adaptation of Mathematical Documents*. PhD thesis, Jacobs University Bremen, 2010.

[Nor05] U. Norell. The Agda WiKi, 2005. `http://wiki.portal.chalmers.se/agda`.

[Ora] Oracle. JDK 6 Java Archive (JAR). `http://download.oracle.com/javase/6/docs/technotes/guides/jar`.

[PS99] F. Pfenning and C. Schürmann. System description: Twelf - a meta-logical framework for deductive systems. *Lecture Notes in Computer Science*, 1632:202–206, 1999.

[RK11] F. Rabe and M. Kohlhase. A Scalable Module System. Under review, see `http://arxiv.org/abs/1105.0548`, 2011.

[RS09] F. Rabe and C. Schürmann. A Practical Module System for LF. In J. Cheney and A. Felty, editors, *Proceedings of the Workshop on Logical Frameworks: Meta-Theory and Practice (LFMTP)*, pages 40–48. ACM Press, 2009.

[TB85] A. Trybulec and H. Blair. Computer Assisted Reasoning with MIZAR. In A. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 26–28, 1985.

[Wen10] Makarius Wenzel. Asynchronous proof processing with Isabelle/Scala and Isabelle/jEdit. In C. Sacerdoti Coen and D. Aspinall, editors, *User Interfaces for Theorem Provers (UITP 2010), FLOC 2010 Satellite Workshop*, ENTCS. Elsevier, July 2010.

[ZK09a] V. Zholudev and M. Kohlhase. TNTBase: a Versioned Storage for XML. In *Proceedings of Balisage: The Markup Conference 2009*, volume 3 of *Balisage Series on Markup Technologies*. Mulberry Technologies, Inc., 2009.

[ZK09b] Vyacheslav Zholudev and Michael Kohlhase. TNTBase: a versioned storage for XML. In *Proceedings of Balisage: The Markup Conference*, volume 3 of *Balisage Series on Markup Technologies*. Mulberry Technologies, Inc., 2009.