



JACOBS  
UNIVERSITY

Michael Kohlhase and Christoph Lange and Christine Müller and Normen Müller and Florian Rabe

## **Adaptation of Notations in Living Mathematical Documents**

KWARC Report No. KWARC2008-2 (version 1)  
April 2008

---

School of Engineering and Science

# Adaptation of Notations in Living Mathematical Documents

Michael Kohlhase and Christoph Lange and Christine Müller and Normen Müller and Florian Rabe

*School of Engineering and Science  
Jacobs University Bremen gGmbH  
Campus Ring 12  
28759 Bremen  
Germany*

## Summary

Notations are central for understanding mathematical discourse. Readers would like to read notations that transport the meaning well and prefer notations that are familiar to them. Therefore, authors optimize the choice of notations with respect to these two criteria, while at the same time trying to remain consistent over the document and their own prior publications. In print media where notations are fixed at publication time, this is an over-constrained problem. In living documents notations can be adapted at reading time, taking reader preferences into account.

We present a representational infrastructure for notations in living mathematical documents. Mathematical notations can be defined declaratively. Author and reader can extensionally define the set of available notation definitions at arbitrary document levels, and they can guide the notation selection function via intensional annotations.

We give an abstract specification of notation definitions and the flexible rendering algorithms and show their coverage on paradigmatic examples. We show how to use this framework to render OPENMATH and Content-MATHML to Presentation-MATHML, but the approach extends to arbitrary content and presentation formats. We discuss prototypical implementations of all aspects of the rendering pipeline.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Use Cases . . . . .	2
1.2	Requirements . . . . .	3
1.3	Related Work . . . . .	6
<b>2</b>	<b>Syntax of Notation Definitions</b>	<b>7</b>
<b>3</b>	<b>Semantics of Notation Definitions</b>	<b>10</b>
<b>4</b>	<b>An XML Encoding for Notation Definitions</b>	<b>12</b>
<b>5</b>	<b>Choosing Notation Definitions Extensionally</b>	<b>13</b>
5.1	Collecting Notation Definitions . . . . .	13
5.2	Discussion of Collection Strategies . . . . .	14
5.2.1	Illustrative Example . . . . .	14
5.2.2	Collection from <i>SD</i> . . . . .	15
5.2.3	Collecting from <i>F</i> . . . . .	16
5.2.4	Collection from <i>CD</i> . . . . .	18
5.2.5	Collection from <i>Doc</i> . . . . .	19
5.2.6	Collection from <i>EC</i> . . . . .	20
5.2.7	Summary/ Findings . . . . .	22
<b>6</b>	<b>Choosing Renderings Intensionally</b>	<b>22</b>
6.1	Collecting Contextual Information . . . . .	23
6.2	Matching the Rendering Context . . . . .	23
6.3	Discussion of Context-Sensitive Selection Strategies . . . . .	25
6.3.1	Collecting from <i>GC</i> . . . . .	25
6.3.2	Collecting from <i>MD</i> . . . . .	26
6.3.3	Collecting from <i>IC</i> . . . . .	27
6.3.4	Collecting from <i>CCF</i> . . . . .	29
6.3.5	Summary/ Findings . . . . .	30
<b>7</b>	<b>Open Questions</b>	<b>30</b>
7.1	Write Protection . . . . .	30
7.2	Notation Management . . . . .	31
7.3	Consistency . . . . .	32
7.4	Flexible Elisions . . . . .	33
7.5	Adaptation of OMDOC Documents . . . . .	34
<b>8</b>	<b>Implementation</b>	<b>36</b>
8.1	mmlkit — Implementing the Presentation Pipeline . . . . .	36
8.2	panta rhei — An Interactive and Collaborative Viewer . . . . .	38
8.3	SWiM – A Semantic Wiki to edit Notation Definitions . . . . .	41
<b>9</b>	<b>Conclusion &amp; Outlook</b>	<b>44</b>

## List of Figures

1	The Grammar for Notation Definitions . . . . .	7
2	Encoding of Notations . . . . .	13
3	OPENMATH Encoding of Patterns . . . . .	14
4	Collecting Notation Definition from the System Defaults . . . . .	16
5	Collecting Notation Definition from an External File . . . . .	17
6	Collecting Notation Definition from Content Dictionaries . . . . .	18
7	Collecting Document-Internal Notation Definition . . . . .	19
8	Collecting Notation Definition via Extensional References . . . . .	21
9	Selection of Renderings based on a Global Context . . . . .	25
10	Selection of Renderings based on metadata . . . . .	27
11	Selection of Renderings based on Intensional References . . . . .	28
12	Managing Notation Definition via <i>notation containers</i> . . . . .	32
13	Flexible Elision of Brackets in XHTML . . . . .	34
14	Collecting Notation Definition from OMDOC Content Dictionaries . . . . .	35
15	<code>mm1kit</code> and its components . . . . .	36
16	The Architecture of <code>panta rhei</code> . . . . .	39
17	The Presentation Workflow in <code>panta rhei</code> . . . . .	39
18	A content dictionary in SWIM . . . . .	42
19	Editing OPENMATH in SWIM . . . . .	43
20	SPARQL query determining the effect of changing a notation definition; see fig. 21 for a graphical representation. . . . .	44
21	Finding pages (depicted as stacks of nodes) affected by changes to a nota- tion definition. Numbers refer to lines of listing 20. Note that both <i>sym</i> and the <i>symDefs</i> are instances of the class <i>SymbolDefinition</i> . . . . .	44

# 1 Introduction

Over the last three millennia, mathematics has developed a complicated two-dimensional format for communicating formulae (see e.g., [Caj93, Wol00] for details). Structural properties of operators often result in special presentations, e.g., the scope of a radical expression is visualized by the length of its bar. Their mathematical properties give rise to placement (e.g., associative arithmetic operators are written infix), and their relative importance is expressed in terms of binding strength conventions for brackets. Changes in notation have been influential in shaping the way we calculate and think about mathematical concepts, and understanding mathematical notations is an essential part of any mathematics education. All of these make it difficult to determine the functional structure of an expression from its presentation.

Content Markup formats for mathematics such as OPENMATH [BCC<sup>+</sup>04] and content MATHML [ABC<sup>+</sup>03] concentrate on the functional structure of mathematical formulae, thus allowing mathematical software systems to exchange mathematical objects. For communication with humans, these formats rely on a “presentation process” (usually based on XSLT style sheets) that transforms the content objects into the usual two-dimensional form used in mathematical books and articles. Many such presentation processes have been proposed, and all have their strengths and weaknesses. In this paper, we conceptualize the presentation of mathematical formulae as consisting of two components: the two-dimensional **composition** of visual sub-presentations to larger ones and the **elision** of formula parts that can be deduced from context.

Most current presentation processes concentrate on the relatively well-understood composition aspect and implement only rather simple bracket elision algorithms. But the visual renderings of formulae in mathematical practice are not simple direct compositions of the concepts involved: mathematicians gloss over parts of the formulae, e.g., leaving out arguments, iff they are non-essential, conventionalized or can be deduced from the context. Indeed this is part of what makes mathematics so hard to read for beginners, but also what makes mathematical language so efficient for the initiates. A common example is the use of  $\log(x)$  or even  $\log x$  for  $\log_{10}(x)$  or similarly  $[[t]]$  for  $[[t]]_{\mathcal{M}}^{\varphi}$ , if there is only one model  $\mathcal{M}$  in the context and  $\varphi$  is the most salient variable assignment.

Another example are the bracket elision rules in arithmetical expressions:  $ax + y$  is actually  $(ax) + y$ , since multiplication “binds stronger” than addition. Note that we would not consider the “invisible times” operation as another elision, but as an alternative presentation.

In this situation we propose to encode the presentational characteristics of symbols (for composition *and* elision) declaratively in **notation definitions**, which are part of the representational infrastructure and consist of “prototypes” (patterns that are matched against content representation trees) and “renderings” (that are used to construct the corresponding presentational trees). Note that since we have reified the notations, we can now devise flexible management process for notations. For example, we can capture the notation preferences of authors, aggregators and readers and adapt documents to these. We propose an elaborated mechanism to collect notations from various sources and specify notation preferences. This brings the separation of *function* from *form* in mathematical objects and assertions in MKM formats to fruition on the document level. This is especially pronounced in the context of dynamic presentation media (e.g., on the screen), we can now realize “*active documents*”, where we can interact with a document directly, e.g., instantiating a formula with concrete values or graphing a function to explore

it or “*living/evolving documents*” which monitor the change of knowledge about a topic and adapt to a user’s notation preferences consistently.

## 1.1 Use Cases

We present four use cases to derive requirements (cf. 1.2) for living mathematical documents with respect to their notations:

**Authoring** Authors want full *flexibility* and *freedom* when writing a document, in particular, when selecting notations for their mathematical expression, no matter how *complex* they might be. They want to be able to use different notations for the same statements throughout different sections, paragraphs, or sentences. Moreover, authors want to be able to *highlight* and *compare* alternative notations for the same mathematical expression and introduce their individual notations, in particular, if they address an *interdisciplinary* or *multicultural* audience. Moreover, authors need to adapt their document to a particular *context*, such as the style template of a conference or the preferences and experiences of their audience. To facilitate the authoring process, authors want to *reuse* and *share* notations among each other.

**Reading** Users want to read documents with as *little effort* as possible. In particular, the initial document needs to be *consistent* in terms of its default notations. Moreover, users want to adapt the notations to their *individual styles* or particular *output requirements* but do not want to *overwrite* any notations that are important for understanding the author’s intentions. The adaptation facilitates the *accessibility*, *usability*, and *readability* of documents since users have to spend less effort and time on trying to *map* and *interpret* the authors notations. Moreover, users want to adapt a document to the style of other users and, consequently, look at documents via the *lenses/ perspective* of others.

**Collaborative Authoring** Each author wants to use his individual notations but still allow others to understand and read his contributions. As a reader he wants to adapt the contributions of others to his style but does not want to overwrite notations that are relevant to understand the contributions of his coauthors. The final document needs to be consistent in terms of its notations, consequently, all authors have to agree on a *common style* or adapt the document to a specific context, e.g. a conference style (cf. [WM07]).

**Generation and Usage of Course Material** A teacher wants to create a script from a *content collection* [Koh06] created by *multiple* authors. The *content management system* [OMB08, Lan08c, HBK03, MAF<sup>+</sup>01] provides an *initial* presentation of the script based on the *author’s notations* and the *system’s default settings*. The teacher adapts the notations of the script for a particular *field of study*, e.g. “mathematics” or “computer science”. She wants to assure a *consistent* use of notations and *lock* specific notations that her students need to learn and, consequently, should not overwrite when reading the document. When reading the course material, e.g. by using a specific *reader/viewer* [pan08c, MAF<sup>+</sup>01, Mat08, Goo09, Ado08] or *online platform* [HBK03, Pla09, Wik07, xma08], the students can adapt all notations but the locked ones.

## 1.2 Requirements

In this section, we derive a list of requirements from the use cases in Section 1.1 and illustrate them with several examples:

### R1: Complex Notations

Users want to choose any notations they consider appropriate and need full flexibility and freedom to introduce their individual styles.

*Example 1* Below you see three application of the *logarithm* function onto a given number  $y$  and a base  $x$ ,  $e$ , or 10. The *applications* (denoted by @) represents the meaning of the expression, i.e. its semantic markup. It is transformed (denoted by  $\rightsquigarrow$ ) to a concrete presentation,  $\log_x y$ ,  $\ln y$ , and  $\lg y$ . The semantic markup can be interpreted as a tree, which is recursively rendered to a concrete presentation.

- (1)  $@(\log, x, y) \rightsquigarrow \log_x y$
- (2)  $@(\log, e, y) \rightsquigarrow \ln y$
- (3)  $@(\log, 10, y) \rightsquigarrow \lg y$

For example, for item (1) the symbol *log* is rendered to  $\log$ , the positions of the parameters are fixed, finally both parameters  $x$  and  $y$  are rendered. Although the applications in item (2) and (3) look quite similar to (1), the rendering differs from the default pattern: For example, in item (2) the symbol *log* is rendered to  $\ln$ , the positions of the parameters are fixed (leaving out the *base*) and finally  $y$  is rendered. Consequently, depending on the *base* parameter, we need to render the application trees differently. We say that item (2) and (3) are *no local tree transformations* (cf. [NW01a]), *non-compositional* and *parameter-sensitive*.

Further example for a parameter-sensitive transformation include the *integral* function, for which we can either specify an interval or set:  $\int_a^b f(x)dx$  versus  $\int_{x \in \{1,2,3,\dots,n\}} f(x)dx$  or the *sum* function  $\sum_{x=1}^n x + y$  versus  $\sum_{x \in M} x + y$ , respectively.

We can also think about the application of the *exponentiation* function  $@(\text{power}, @(\text{sin}, x), y)$  with the two parameters  $y$  (the exponent) and  $@(\text{sin}, x)$  (the base), which is to be presented as  $\sin^y x$  rather than  $(\sin x)^y$ . Alternative, this applies to  $@(\text{plus}, @(\text{power}, x, 2), @(\text{times}, @(\text{times}, @(\text{neg}, 3), y), x), 1)$  which should be rendered to  $x^2 - 3yx + 1$  rather than  $x^2 + (-3y)x + 1$  (cf. [NW01a]).

Further challenges refer to notations with an arbitrary number of elements, e.g. an arbitrary length vector, or notations with *ellipses*: For example, authors might want to render the application  $@(\text{plus}, 1, 2, 3, 4, 5, 6, 7)$  to  $1 + 2 + \dots + 7$  rather than  $1 + 2 + 3 + 4 + 5 + 6 + 7$ .

### R2: Adaptation to Contexts

A user's preferred notation depends on several *context dimensions*, such as his *individual style* and *background*, his *area of application*, *nationality*, *level of expertise* or further restrictions given by the *output format* (PDF, L<sup>A</sup>T<sub>E</sub>X, or HTML) or *output layout* (slide, script, or book).

*Example 2* The adaptation of notations facilitates or prevents users from understanding the definition of a mathematical symbol: For example,  $\mathbb{N}$ ,  $\mathbb{N}_0$ , and  $\mathbb{N}_1$  are used to

denote the set of the *natural numbers*. Further examples are given in the table below (cf. [Mül08a, KMM07, NW01a, SW06b, NW01b, SW06a, MLUM05]):

Dimensions	Alternative Notations
Nationality	$\binom{n}{k}$ ( <i>German</i> ), $C_k^n$ ( <i>Russian</i> ), $C_n^k$ ( <i>French</i> ) $\gcd(a, b)$ ( <i>en</i> ), $kgV(a, b)$ ( <i>de</i> ) 1,000 ( <i>en</i> ) rather than 1.000 ( <i>de</i> ) but 1.2 ( <i>en</i> ) rather than 1,2 ( <i>de</i> )
Individual Styles	$A \subseteq B$ , $A \subseteq\subseteq B$ $A \subsetneq B$ , $A \subsetneqq B$ , $A \subsetneq\subsetneq B$ , $A \subsetneqq\subsetneqq B$
Expertise	$a \div b$ ( <i>elementary school</i> ), $\frac{a}{b}$ ( <i>higher education</i> ) $\log_e y$ rather than $\ln y$
Output Styles	$\sum_{k=1}^n$ ( <i>text</i> ) $\sum_{k=1}^n$ ( <i>display</i> ) $\bigcup_{k=1}^n$ ( <i>text</i> ) $\bigcup_{k=1}^n$ ( <i>display</i> )
Area	$i$ ( <i>mathematics</i> ), $j$ ( <i>electrical engineering</i> ) $\mathbf{N}_0$ ( <i>German, number theory</i> ), $\mathbf{N}$ ( <i>German, set theory</i> ) $\neq$ ( <i>mathematics</i> ), $\neq$ and $\langle \rangle$ ( <i>programming languages</i> )

### R3: Dynamic Notations

Users want to interact with their document and, particularly, adapt notations *dynamically*.

*Example 3* In a learning process it is particularly helpful to extend or reduce the number of brackets or types of an expression:

Expert	...	Beginner
$ax + y$	...	$(ax) + y$
$5 + x * y^{n+3}$	...	$5 + (x * (y^{n+3}))$
$\llbracket t \rrbracket$ (if there is only one model $\mathcal{M}$ in the context and $\varphi$ is the most salient variable assignment)	...	$\llbracket t \rrbracket_{\mathcal{M}}^{\varphi}$

Moreover, it is helpful to allow users to dynamically switch between alternative notations and to provide additional explanations such as how to read the notations or by linking to the definition of the respective symbols.

### R4: Notation Layout

Users want to adapt the layout of notations, e.g. by changing the color, size, format, or background color of the notations in order to emphasize certain important, relevant or interesting properties of the mathematical expression.

*Example 4* Changing the color and font of the notations:

$$\log_e y = \mathbf{ln} \ y \qquad (\mathbf{sin} \ \mathbf{x})^y = \mathbf{sin}^y \ \mathbf{x}$$

## R5: Granularity

Users want to choose different notations for the same expression for *granular document fragments*

*Example 5* An author wants to choose three presentations for the same symbol in one sentence:

We discuss various notations of the binomial coefficient :  $\binom{n}{k}$ ,  $C_k^n$ , and  $C_n^k$ .

The respective content representation of the expression is given below.

We discuss the notations of the binomial coefficient : @ (bc,n,k), @ (bc,n,k), and @ (bc,n,k).

## R6: Write Protection

Users want to *lock* notations and prevent other users from overwriting.

*Example 6* In Example 5, the author intentionally chooses alternative notations to compare them. If we overwrite all of these notations by  $\binom{n}{k}$ , the sentence will become meaningless and confusing:

We will discuss the three notations of the  $\binom{n}{k}$ :  $\binom{n}{k}$ ,  $\binom{n}{k}$ , and  $\binom{n}{k}$ .

## R7: Consistent Merge and Change Management

Users want to *merge* material from a content collection and receive an *initial* as well as *consistent* presentation in terms of the included notations.

*Example 7* A teacher wants to create a course based on material of a mathematics and an electrical engineering course. She needs to decide on whether to use *i* or *j*, two alternative notations for the *imaginary unit*, and use the respective notations consistently throughout the whole course. This even gets more complex, if she wants to use both notations on granular document level.

*Example 8* When adapting notations, the interrelation of alternative notations needs to be considered. For example, if we think of the notations for *subset* and *proper subset*: We can use  $\subset$  and  $\subseteq$ , for denoting a proper subset and subset, respectively. Alternatively, we can use  $\subsetneq$  and  $\subset$ , whereas the latter now denotes a subset rather than a proper subset. When choosing  $\subset$  for the proper subset, we should not use the same notation for a subset.

## R8: Notation Management

Users want to *reuse*, *adapt*, *extend*, and *categorize* their *notation collections* and the collections of other users. Editing and selection of notations should not require well grounded programming skills (e.g. for XSLT programming) or knowledge of a specific document format (e.g. OPENMATH or MATHML).

In particular, we need to provide respective tools: Authoring environments should include accessible and usable editors that facilitate users to create, modify, and select notations without any specific skills [GP06, Lan08c, pla07, HBK03, MAF<sup>+</sup>01]. A reading environments [pan08c, HBK03, MAF<sup>+</sup>01, Ado08] should support the adaptation of documents to a specific context. Moreover, users should be supported to easily share, reuse, and store knowledge in distributed or central notation management system [Zho08, pla07, HBK03, MAF<sup>+</sup>01, Lan08c].

### 1.3 Related Work

Before we present our system, let us review the state of the art. Naylor, Smirnova, and Watt [NW01b, SW06b, SW06a] present an approach based on meta stylesheets that utilizes a MATHML-based markup of arbitrary notations in terms of their content and presentation and, based on the manual selection of users, generates user-specific XSLT style sheets [Kay06] for the adaptation of documents. Naylor and Watt [NW01b] introduce a one-dimensional context annotation of content expressions to intentionally select an appropriate notation specification. The authors claim that users also want to delegate the styling decision to some defaulting mechanism and propose the following hierarchy of default notation specification (from high to low): command line control, input documents defaults, meta stylesheets defaults, and content dictionary defaults.

In [MLUM05], Manzoor et al. emphasize the need for maintaining uniform and appropriate notations in collaborative environments, in which various authors contribute mathematical material. They address the problem by providing authors with respective tools for editing notations as well as by developing a framework for a consistent presentation of symbols. In particular, they extend the approach of Naylor and Watt by an explicit language markup of the content expression. Moreover, the authors propose the following prioritization of different notation styles (from high to low): individual style, group, book, author or collection, and system defaults.

In [KLR07] we have revised and improved the presentation specification of OM-DOC1.2. [Koh06] by allowing a static well-formedness, i.e., the well-formedness of presentation specifications can be verified when writing the presentations rather than when presenting a document. We also addressed the issue of flexible elision. However, the approach does not facilitate to specify notations, which are not local tree transformations of the semantic markup.

In [KMM07] we initiated the redefinition of *documents* towards a more *dynamic* and *living* view. We explicated the narrative and content layer and extended the *document model* by a third dimension, i.e., the *presentation layer*. We proposed the *extensional* markup of the *notation context* of a document, which facilitates users to explicitly select suitable notations for document fragments. These extensional collection of notations can be inherited, extended, reused, and shared among users. For the system presented in this paper, we have re-engineered and extended the latter two proposals.

In Sect. 2, we introduce abstract syntax for notation definitions, which is used for the internal representation of our notation objects. (We use a straightforward XML encoding as concrete syntax.) In Sect. 3, we describe how a given notation definition is used to translate an OPENMATH object into its presentation. After this local view of notation definitions, the remainder of the paper takes a more global perspective by introducing markup that permits users to control which notation definitions are used to present which document fragment. There are two conflicting ways how to define this set of available

notation definitions: extensionally by pointing to a notation container; or intensionally by attaching properties to notation definitions and using them to select between them. These ways are handled in Sect. 5 and 6, respectively.

## 2 Syntax of Notation Definitions

We will now present an abstract version of the presentation starting from the observation that in content markup formalisms for mathematics formulae are represented as “formula trees”. Concretely, we will concentrate on OPENMATH objects, the conceptual data model of OPENMATH representations, since it is sufficiently general, and work is currently under way to re-engineer content MATHML representations based on this model. Furthermore, we observe that the target of the presentation process is also a tree expression: a layout tree made of layout primitives and glyphs, e.g., a presentation MATHML or L<sup>A</sup>T<sub>E</sub>X expression.

To specify notation definitions, we use the one given by the abstract grammar from Fig. 1. Here  $|$ ,  $[-]$ ,  $-^*$ , and  $-^+$  denote alternative, bracketing, and non-empty and possibly empty repetition, respectively. The non-terminal symbol  $\omega$  is used for patterns  $\varphi$  that do not contain jokers. Throughout this paper, we will use the non-terminal symbols of the grammar as meta-variables for objects of the respective syntactic class.

Notation declarations	$ntn$	$::=$	$\varphi^+ \vdash [(\lambda : \rho)^p]^+$
Patterns	$\varphi$	$::=$	
Symbols			$\sigma(n, n, n)$
Variables		$ $	$v(n)$
Applications		$ $	$@(\varphi[, \varphi]^+)$
Binders		$ $	$\beta(\varphi, \Upsilon, \varphi)$
Attributions		$ $	$\alpha(\varphi, \sigma(n, n, n) \mapsto \varphi)$
Symbol/Variable/Object/List jokers		$ $	$\underline{s} \mid \underline{v} \mid \underline{o} \mid \underline{l}(\varphi)$
Variable contexts	$\Upsilon$	$::=$	$\varphi^+$
Match contexts	$M$	$::=$	$[q \mapsto X]^*$
Matches	$X$	$::=$	$\omega^*   S^*   (X)$
Empty match contexts	$\mu$	$::=$	$[q \mapsto H]^*$
Holes	$H$	$::=$	$_  \text{“”}   (H)$
Context annotation	$\lambda$	$::=$	$(S = S)^*$
Renderings	$\rho$	$::=$	
XML elements			$\langle S \rangle \rho^* \langle / \rangle$
XML attributes		$ $	$S = \text{”} \rho^* \text{”}$
Texts		$ $	$S$
Symbol or variable names		$ $	$\underline{q}$
Matched objects		$ $	$\underline{q}^p$
Matched lists		$ $	$\mathbf{for}(q, I, \rho^*)\{\rho^*\}$
Precedences	$p$	$::=$	$-\infty   I   \infty$
Names	$n, s, v, l, o$	$::=$	$C^+$
Integers	$I$	$::=$	integer
Qualified joker names	$q$	$::=$	$l/q s v o l$
Strings	$S$	$::=$	$C^*$
Characters	$C$	$::=$	character except /

Figure 1: The Grammar for Notation Definitions

**Intuitions** The intuitive meaning of a notation definition  $ntr = \varphi_1, \dots, \varphi_r \vdash (\lambda_1 : \rho_1)^{p_1}, \dots, (\lambda_s : \rho_s)^{p_s}$  is the following: If an object matches one of the patterns  $\varphi_i$ , it is rendered by one of the renderings  $\rho_i$ . Which rendering is chosen, depends on the active rendering context, which is matched against the context annotations  $\lambda_i$  (see Sect. 6). Each context annotation is a key-value list designating the intended rendering context. The integer values  $p_i$  give the output precedences of the renderings.

The patterns  $\varphi_i$  are formed from a formal grammar for a subset of OPENMATH objects extended with named jokers. The jokers  $\underline{o}$  and  $\underline{l}(\varphi)$  correspond to  $\backslash(\cdot\backslash)$  and  $\backslash(\varphi\backslash)^+$  in Posix regular expression syntax ([POS88]) – except that our patterns are matched against the list of children of an OPENMATH object instead of against a list of characters. We need two special jokers  $\underline{s}$  and  $\underline{v}$ , which only match OPENMATH symbols and variables, respectively. The renderings  $\rho_i$  are formed by a formal syntax for simplified XML extended with means to refer to the jokers used in the patterns. When referring to object jokers, input precedences are given that are used, together with the output precedences, to determine the placement of brackets.

Match contexts are used to store the result of matching a pattern against an object. Due to list jokers, jokers may be nested; therefore, we use qualified joker names in the match contexts (which are transparent to the user). Empty match contexts are used to store the structure of a match context induced by a pattern: They contain holes that are filled by matching the pattern against an object.

**Example** We will use a multiple integral as an example that shows all aspects of our approach in action.

$$\int_{a_1}^{b_1} \dots \int_{a_n}^{b_n} \color{red}{\sin x_1 + x_2} dx_n \dots dx_1.$$

Let *int*, *iv*, *lam*, *plus*, and *sin* abbreviate symbols for integration, closed real intervals, lambda abstraction, addition, and sine. We intend *int*, *lam*, and *plus* to be flexary symbols, i.e., symbols that take an arbitrary finite number of arguments. Furthermore, we assume symbols *color* and *red* from a content dictionary for style attributions. We want to render into L<sup>A</sup>T<sub>E</sub>X the OPENMATH object

$$\begin{aligned} & @(\textit{int}, @(\textit{iv}, a_1, b_1), \dots, @(\textit{iv}, a_n, b_n), \\ & \beta(\textit{lam}, v(x_1), \dots, v(x_n), \alpha(@(\textit{plus}, @(\textit{sin}, v(x_1)), v(x_2)), \textit{color} \mapsto \textit{red}))) \end{aligned}$$

as `\int_{a_1}^{b_1} \dots \int_{a_n}^{b_n} \color{red}{\sin x_1+x_2} dx_n \dots dx_1`

We can do that with the following notations:

$$\begin{aligned} & @(\textit{int}, \underline{\textit{ranges}}(@(\textit{iv}, \underline{\mathbf{a}}, \underline{\mathbf{b}})), \beta(\textit{lam}, \underline{\textit{vars}}(\underline{\mathbf{x}}, \underline{\mathbf{f}})) \\ & \vdash ((\textit{format} = \textit{latex}) : \\ & \quad \textit{for}(\underline{\textit{ranges}})\{\int_{\underline{\mathbf{a}}^\infty}^{\underline{\mathbf{b}}^\infty}\} \underline{\mathbf{f}}^\infty \textit{for}(\underline{\textit{vars}}, -1)\{d \underline{\mathbf{x}}^\infty\})^{-\infty} \\ & \alpha(\underline{\mathbf{a}}, \textit{color} \mapsto \underline{\textit{col}}) \vdash ((\textit{format} = \textit{latex}) : \{\color{\underline{\textit{col}}}{\underline{\mathbf{a}}^\infty}\})^{-\infty} \\ & @(\textit{plus}, \underline{\textit{args}}(\underline{\textit{arg}})) \vdash ((\textit{format} = \textit{latex}) : \textit{for}(\underline{\textit{args}}, +)\{\underline{\textit{arg}}\})^{10} \\ & @(\textit{sin}, \underline{\textit{arg}}) \vdash ((\textit{format} = \textit{latex}) : \sin \underline{\textit{arg}})^0 \end{aligned}$$

The first notation matches the application of the symbol *int* to a list of ranges and a lambda abstraction binding a list of variables. The rendering iterates first over the ranges rendering them as integral signs with bounds, then recurses into the function

body  $\underline{f}$ , then iterates over the variables rendering them in reverse order prefixed with  $d$ . The second notation is used when  $\underline{f}$  recurses into the presentation of the function body  $\alpha(@(\textit{plus}, @(\textit{sin}, v(x_1)), v(x_2)), \textit{color} \mapsto \textit{red})$ . It matches an attribution of  $\textit{color}$ , which is rendered using the L<sup>A</sup>T<sub>E</sub>X color package. The third notation is used when  $\underline{a}$  recurses into the attributed object  $@(\textit{plus}, @(\textit{sin}, v(x_1)), v(x_2))$ . It matches any application of  $\textit{plus}$ , and the rendering iterates over all arguments placing the separator  $+$  in between. Finally,  $\textit{sin}$  is rendered in a straightforward way. We omit the notation that renders variables by their name.

The output precedence  $-\infty$  of  $\textit{int}$  makes sure that the integral as a whole is never bracketed. And the input precedences  $\infty$  make sure that the arguments of  $\textit{int}$  are never bracketed. Both are reasonable because the integral notation provides its own fencing symbols, namely  $\int$  and  $d$ . The output precedences of  $\textit{plus}$  and  $\textit{sin}$  are 10 and 0, which means that  $\textit{sin}$  binds stronger; therefore, the expression  $\textit{sin } x$  is not bracketed either. However, an inexperienced user may wish to display these brackets: Therefore, our rendering does not suppress them. Rather, we annotate them with an elision level, which is computed as the difference of the two precedences. Dynamic output formats that can change their appearance, such as XHTML with JavaScript, can use the elision level to determine the visibility of symbols based on user-provided elision thresholds: the higher its elision level, the less important a bracket.

**Well-formed Notations** A notation definition  $\varphi_1, \dots, \varphi_r \vdash (\lambda_1 : \rho_1)^{p_1}, \dots, (\lambda_s : \rho_s)^{p_s}$  is well-formed if all  $\varphi_i$  are well-formed patterns that induce the same empty match contexts, and all  $\rho_i$  are well-formed renderings with respect to that empty match context.

Every pattern  $\varphi$  generates an *empty match context*  $\mu(\varphi)$  as follows:

- For an object joker  $\underline{o}$  occurring in  $\varphi$  but not within a list joker,  $\mu(\varphi)$  contains  $o \mapsto \_$ .
- For a symbol or variable with name  $n$  occurring in  $\varphi$  but not within a list joker,  $\mu(\varphi)$  contains  $n \mapsto \text{“”}$ .
- For a list joker  $\underline{l}(\varphi')$  occurring in  $\varphi$ ,  $\mu(\varphi)$  contains
  - $l \mapsto (\_)$ , and
  - $l/n \mapsto (H)$  for every  $n \mapsto H$  in  $\mu(\varphi')$ .

In an empty match context, a hole  $\_$  is a placeholder for an object,  $\text{“”}$  for a string,  $(\_)$  for a list of objects,  $((\_))$  for a list of lists of objects, and so on. Thus, symbol, variable, or object joker in  $\varphi$  produce a single named hole, and every list joker and every joker within a list joker produces a named list of holes ( $H$ ). For example, the empty match context induced by the pattern in the notation for  $\textit{int}$  above is

$$\begin{aligned} \text{ranges} &\mapsto (\_), \text{ ranges/a} \mapsto (\_), \text{ ranges/b} \mapsto (\_), \text{ f} \mapsto \_, \\ \text{vars} &\mapsto (\_), \text{ vars/x} \mapsto (\text{“”}) \end{aligned}$$

A pattern  $\varphi$  is well-formed if it satisfies the following conditions:

- There are no duplicate names in  $\mu(\varphi)$ .
- List jokers may not occur as direct children of binders or attributions.

- At most one list joker may occur as a child of the same application, and it may not be the first child.
- At most one list joker may occur in the same variable context.

These restrictions guarantee that matching an OPENMATH object against a pattern is possible in at most one way. In particular, no backtracking is needed in the matching algorithm.

Assume an empty match context  $\mu$ . We define well-formed renderings with respect to  $\mu$  as follows:

- $\langle S \rangle \rho_1, \dots, \rho_r \langle / \rangle$  is well-formed if all  $\rho_i$  are well-formed.
- $S = \text{"}\rho_1, \dots, \rho_r\text{"}$  is well-formed if all  $\rho_i$  are well-formed and are of the form  $S'$  or  $\underline{n}$ . Furthermore,  $S = \text{"}\rho_1, \dots, \rho_r\text{"}$  may only occur as a child of an XML element rendering.
- $S$  is well-formed.
- $\underline{n}$  is well-formed if  $n \mapsto \text{"}"$  is in  $\mu$ .
- $\underline{o}^p$  is well-formed if  $o \mapsto \_$  is in  $\mu$ .
- $\text{for}(l, I, sep)\{body\}$  is well-formed if  $l \mapsto (\_)$  or  $l \mapsto (\text{"})$  is in  $\mu$ , all renderings in  $sep$  are well-formed with respect to  $\mu$ , and all renderings in  $body$  are well-formed with respect to  $\mu^l$ . The step size  $I$  and the separator  $sep$  are optional, and default to 1 and the empty string, respectively, if omitted.

Here  $\mu^l$  is the empty match context arising from  $\mu$  if every  $l/q \mapsto (H)$  is replaced with  $q \mapsto H$  and every previously existing hole named  $q$  is removed. Replacing  $l/q \mapsto (H)$  means that jokers occurring within the list joker  $l$  are only accessible within a corresponding rendering  $\text{for}(l, I, \rho^*)\{\rho^*\}$ . And removing the previously existing holes means that in  $@(\underline{o}, l(\underline{o}))$ , the inner object joker shadows the outer one.

### 3 Semantics of Notation Definitions

The rendering algorithm takes as input a notation context  $\Pi$  (a list of notation definitions, computed as described in Sect. 5), a rendering context  $\Lambda$  (a list of context annotations, computed as described in Sect. 6), an OPENMATH object  $\omega$ , and an input precedence  $p$ . If the algorithm is invoked from top level (as opposed to a recursive call),  $p$  should be set to  $\infty$  to suppress top level brackets.

It returns as output either text or an XML element. There are two output types for the rendering algorithm: text and sequences of XML elements. We will use  $O + O'$  to denote the concatenation of two outputs  $O$  and  $O'$ . By that, we mean a concatenation of sequences of XML elements or of strings if  $O$  and  $O'$  have the same  $\mu$  type. Otherwise,  $O + O'$  is a sequence of XML elements treating text as an XML text node. This operation is associative if we agree that consecutive text nodes are always merged. The algorithm inserts brackets if necessary. And to give the user full control over the appearance of brackets, we obtain the brackets by the rendering of two symbols for left and right bracket from a special fixed content dictionary. The algorithm consists of the following three steps.

1.  $\omega$  is matched against the patterns in the notation definitions in  $\Pi$  (in the listed order) until a matching pattern  $\varphi$  is found. The notation definition in which  $\varphi$  occurs induces a list  $(\lambda_1 : \rho_1)^{p_1}, \dots, (\lambda_n : \rho_n)^{p_n}$  of context-annotations, renderings, and output precedences.
2. The rendering context  $\Lambda$  is matched against the context annotations  $\lambda_i$  in order. The pair  $(\rho_j, p_j)$  with the best matching context-annotation  $\lambda_j$  is selected (see Section 6.3 for details).
3. The output is  $\rho_j^{M(\varphi, \omega)}$ , the rendering of  $\rho_j$  in context  $M(\varphi, \omega)$  as defined below. Additionally, if  $p_j > p$ , the output is enclosed in brackets.

**Semantics of Patterns** The semantics of patterns is that they are matched against OPENMATH objects. Naturally, every OPENMATH object matches against itself. Symbol, variable, and object jokers match in the obvious way. A list joker  $\underline{l}(\varphi)$  matches against a non-empty list of objects all matching  $\varphi$ .

Let  $\varphi$  be a pattern and  $\omega$  a matching OPENMATH object. We define a match context  $M(\varphi, \omega)$  as follows.

- For a symbol or variable joker with name  $n$  that matched against the sub-object  $\omega'$  of  $\omega$ ,  $M(\varphi, \omega)$  contains  $n \mapsto S$  where  $S$  is the name of  $\omega'$ .
- For an object joker  $\underline{o}$  that matched against the sub-object  $\omega'$  of  $\omega$ ,  $M(\varphi, \omega)$  contains  $o \mapsto \omega$ .
- If a list joker  $\underline{l}(\varphi')$  matched a list  $\omega_1, \dots, \omega_r$ , then  $M(\varphi, \omega)$  contains
  - $l \mapsto (\omega_1, \dots, \omega_r)$ , and
  - for every  $l/q$  in  $\mu(\varphi)$ :  $l/q \mapsto (X_1, \dots, X_r)$  where  $q \mapsto X_i$  in  $M(\varphi', \omega_i)$ .

We omit the precise definition of what it means for a pattern to match against an object. It is, in principle, well-known from regular expressions. Since no backtracking is needed, the computation of  $M(\varphi, \omega)$  is straightforward. We denote by  $M(q)$ , the lookup of the match bound to  $q$  in a match context  $M$ .

**Semantics of Renderings** If  $\varphi$  matches against  $\omega$  and the rendering  $\rho$  is well formed with respect to  $\mu(\varphi)$ , the intuition of  $\rho^{M(\varphi, \omega)}$  is that the joker references in  $\rho$  are replaced according to  $M(\varphi, \omega) =: M$ . Formally,  $\rho^M$  is defined as follows.

- $\langle S \rangle \rho_1 \dots \rho_r \langle / \rangle$  is rendered as an XML element with name  $S$ . The attributes are those  $\rho_i^M$  that are rendered as attributes. The children are the concatenation of the remaining  $\rho_i^M$  preserving their order.
- $S = \text{"} \rho_1 \dots \rho_r \text{"}$  is rendered as an attribute with label  $S$  and value  $\rho_1^M + \dots + \rho_n^M$  (which has type text due to the well-formedness).
- $S$  is rendered as the text  $S$ .
- $\underline{s}$  and  $\underline{v}$  are rendered as the text  $M(s)$  or  $M(v)$ , respectively.
- $\underline{o}^p$  is rendered by applying the rendering algorithm recursively to  $M(o)$  and  $p$ .

- $\text{for}(l, I, \rho_1 \dots \rho_r)\{\rho'_1 \dots \rho'_s\}$  is rendered by the following algorithm:
  1. Let  $\text{sep} := \rho_1^M + \dots + \rho_r^M$  and  $t$  be the length of  $M(l)$ .
  2. For  $i = 1, \dots, t$ , let  $R_i := \rho'_1^{M_i^l} + \dots + \rho'_s^{M_i^l}$ .
  3. If  $I = 0$ , return nothing and stop. If  $I$  is negative, reverse the list  $R$ , and invert the sign of  $I$ .
  4. Return  $R_I + \text{sep} + R_{2*I} \dots + \text{sep} + R_T$  where  $T$  is the greatest multiple of  $I$  smaller than or equal to  $t$ .

Here the match context  $M_i^l$  arises from  $M$  as follows

- replace  $l \mapsto (X_1 \dots X_t)$  with  $l \mapsto X_i$ ,
- for every  $l/q \mapsto (X_1 \dots X_t)$  in  $M$ : replace it with  $q \mapsto X_i$ , and remove a possible previously defined match for  $q$ .

**Example** Consider the example introduced in Sect. 2. There we have

$$\omega = @(\text{int}, @(iv, a_1, b_1), \dots, @(iv, a_n, b_n), \\ \beta(\text{lam}, v(x_1), \dots, v(x_n), \alpha(@(\text{plus}, @(sin, v(x_1)), v(x_2)), \text{color} \mapsto \text{red})))$$

And  $\Pi$  is the given list of notation definitions. Let  $\Lambda = (\text{format} = \text{latex})$ . Matching  $\omega$  against the patterns in  $\Pi$  succeeds for the first notation definitions and yields the following match context  $M$ :

$$\begin{aligned} \text{ranges} &\mapsto (@(iv, a_1, b_1), \dots, @(iv, a_n, b_n)), \text{ranges/a} \mapsto (a_1, \dots, a_n), \\ \text{ranges/b} &\mapsto (b_1, \dots, b_n), \text{f} \mapsto \alpha(@(\text{plus}, @(sin, v(x_1)), v(x_2)), \text{color} \mapsto \text{red}), \\ \text{vars} &\mapsto (v(x_1), \dots, v(x_n)), \text{vars/x} \mapsto (x_1, \dots, x_n) \end{aligned}$$

In the second step, a specific rendering is chosen. In our case, there is only one rendering, which matches the required rendering context  $\Lambda$ , namely

$$\rho = \text{for}(\underline{\text{ranges}})\{\backslash\text{int}\_{\{ \underline{a}^\infty \}^{\wedge} \{ \underline{b}^\infty \}} \} \underline{f}^\infty \text{for}(\underline{\text{vars}}, -1)\{\text{d } \underline{x}^\infty\}^{-\infty}$$

To render  $\rho$  in match context  $M$ , we have to render the three components and concatenate the results. Only the iterations are interesting. In both iterations, the separator  $\text{sep}$  is empty; in the second case, the step size  $I$  is  $-1$  to render the variables in reverse order.

## 4 An XML Encoding for Notation Definitions

Now we define an encoding function  $E(-)$  that maps the grammar of Section 2 to XML elements. The encoding of notations and renderings is given in Fig. 2. Fig. 3 gives the encodings of patterns where we use OPENMATH [BCC<sup>+</sup>04] as our specific XML pattern languages. An encoding using content MATHML can be defined similarly.

We assume that the following namespace bindings are in effect:

```
xmlns="http://www.mathweb.org/omdoc"
xmlns:om="http://www.openmath.org/OpenMath"
xmlns:m="http://www.w3.org/1998/Math/MathML"
```

$E(\varphi_1, \dots, \varphi_r \vdash (\lambda_1; \rho_1), \dots, (\lambda_s; \rho_s))$	$\langle \text{notation} \rangle$ $\langle \text{prototype} \rangle E(\varphi_1) \langle / \text{prototype} \rangle$ $\vdots$ $\langle \text{prototype} \rangle E(\varphi_r) \langle / \text{prototype} \rangle$ $\langle \text{rendering } \backslash \text{enc}\{\backslash \text{lambda.1}\} \rangle E(\rho_1) \langle / \text{rendering} \rangle$ $\vdots$ $\langle \text{rendering } \backslash \text{enc}\{\backslash \text{lambda.s}\} \rangle E(\rho_s) \langle / \text{rendering} \rangle$ $\langle / \text{notation} \rangle$
$E(\lambda)$	???
$E(\rho_1, \dots, \rho_r)$	$E(\rho_1) \dots E(\rho_r)$
$E(\langle S \rangle \rho \langle / \rangle)$	$\langle \text{element name} = "S" \rangle$ $E(\rho)$ $\langle / \text{element} \rangle$
$E(S = " \rho ")$	$\langle \text{attribute name} = "S" \rangle$ $E(\rho)$ $\langle / \text{attribute} \rangle$
$E(S)$	$\langle \text{text} \rangle S \langle / \text{text} \rangle$
$E(\underline{p})$	$\langle \text{nameof name} = "p" / \rangle$
$E(\underline{p}^q)$	$\langle \text{render name} = "p" \text{ precedence} = "q" / \rangle$
$E(\text{for}(\underline{p}, I, \rho')\{\rho\})$	$\langle \text{iterate name} = "I" \text{ step} = "I" \rangle$ $\langle \text{separator} \rangle$ $E(\rho')$ $\langle / \text{separator} \rangle$ $E(\rho)$ $\langle / \text{iterate} \rangle$

Figure 2: Encoding of Notations

## 5 Choosing Notation Definitions Extensionally

In the last sections we have seen how collections of notation definitions induce rendering functions. Now we permit users to define the set  $\Pi$  of available notation definitions extensionally. In the following, we discuss the collection of notation definitions from various sources and the construction of  $\Pi_\omega$  for a concrete mathematical object  $\omega$ .

### 5.1 Collecting Notation Definitions

The algorithm for the collection of notation definitions takes as input a tree-structured document, e.g., an XML document, an object  $\omega$  within this document, and a totally ordered set  $\mathcal{S}^N$  of source names. Based on the hierarchy proposed in [NW01a], we use the source names  $EC$ ,  $F$ ,  $Doc$ ,  $CD$ , and  $SD$  explained below. The user can change their priorities by ordering them.

The collection algorithm consists of two steps: The collection of notation definitions and their reorganization. In the first step the notation definitions are collected from the input sources according to the order in  $\mathcal{S}^N$ . The respective input sources are treated as follows:

- $EC$  denotes the **extensional context**, which associates a list of notation definitions or containers of notation definitions to every node of the input document. The effective extensional context is computed according to the position of  $\omega$  in the input document.  $EC$  is used by authors to reference their individual notation context.
- $F$  denotes an **external notation document** from which notation definitions are

$E(\sigma(n, c, b))$	<code>&lt;om:OMS cdbase="b" cd="c" name="n" /&gt;</code>
$E(v(n))$	<code>&lt;om:OMV name="n" /&gt;</code>
$E(@(\varphi_1, \dots, \varphi_n))$	<code>&lt;om:OMA&gt;E(\varphi_1), \dots, E(\varphi_n)&lt;/om:OMA&gt;</code>
$E(\beta(\varphi, \Upsilon, \varphi'))$	<code>&lt;om:OMBIND&gt; E(\varphi) E(\Upsilon) E(\varphi') &lt;/om:OMBIND&gt;</code>
$E(\alpha(\varphi, s \mapsto \varphi'))$	<code>&lt;om:OMATTR&gt; &lt;om:OMATP&gt;E(s) E(\varphi')&lt;/om:OMATP&gt; E(\varphi) &lt;/om:OMATTR&gt;</code>
$E(v_1, \dots, v_n)$	<code>&lt;om:OMBVAR&gt; E(v_1), \dots, E(v_n) &lt;/om:OMBVAR&gt;</code>
$E(\underline{s})$	<code>&lt;expr name="s" type="symbol" /&gt;</code>
$E(\underline{v})$	<code>&lt;expr name="v" type="variable" /&gt;</code>
$E(\underline{o})$	<code>&lt;expr name="o" /&gt;</code>
$E(\underline{l}(\varphi))$	<code>&lt;explist name="l"&gt; E(\varphi) &lt;/explist&gt;</code>

Figure 3: OPENMATH Encoding of Patterns

collected.  $F$  can be used to overwrite the author's extensional context declarations.

- $Doc$  denotes the **input document**. As an alternative to  $EC$ ,  $Doc$  permits authors to embed notation definitions into the input document.
- $CD$  denotes the **content dictionaries** of the symbols occurring in  $\omega$ . These are searched in the order in which the symbols occur in  $\omega$ . Content dictionaries may include or reference default notation definitions for their symbols.
- $SD$  denotes the **system default** notation document, which typically occurs last in  $S^N$  as a fallback if no other notation definitions are given.

In the second step the obtained notation context  $\Pi$  is reorganized: All occurrences of a pattern  $\varphi$  in notation definitions in  $\Pi$  are merged into a single notation definition preserving the order of the  $(\lambda:\rho)^p$ .

## 5.2 Discussion of Collection Strategies

In the following sections, we provide the algorithms for collecting notation definitions from  $EC$ ,  $F$ ,  $Doc$ ,  $CD$  and  $SD$  and illustrate the advantages and drawbacks of basing the rendering on either one of the sources.

### 5.2.1 Illustrative Example

We base our illustration on the input document below, which includes several mathematical concepts, such as *complex number*, *number*, *real number*, *ordered pair*, *imaginary unit*, *electric current* as well as *addition*, *multiplication*, and *exponentiation*. In particular, the document uses alternative notations to present the same mathematical objects: For example, in the second sentence the symbol *imaginary unit* is rendered with  $i$  and  $j$ .

In mathematics, a *complex number* is a *number* which can be formally defined as an *ordered pair* of *real numbers*  $(a, b)$ , often written:  $a + bi$  where  $i^2 = -1$ .

Please note that the usual notation for the *imaginary unit* is  $i$ . However, electrical engineers will rather use  $j$ , not to confuse it with the notation for *electric current*  $I$ .

Below we provide the intended representation of the input document, in which we use the abstract syntax of Section 3 to denote different mathematical (content) object  $\omega$ . For simplicity, we omit the `cdbase` and `cd` attributes of symbols. For example,  $\sigma(\textit{img})$  represents the symbol *imaginary unit* and  $\textcircled{\sigma(\textit{power}), \sigma(\textit{img}), 2}$  is the content markup for  $i^2$ :

In mathematics, a complex number is a number which can be formally defined as an ordered pair of real numbers  $\textcircled{\sigma(\textit{opair}), v(a), v(b)}$ , often written:

$\textcircled{\sigma(\textit{plus}), v(a), \textcircled{\sigma(\textit{times}), v(b), \sigma(\textit{img})}}$  where  
 $\textcircled{\sigma(\textit{equal}), \textcircled{\sigma(\textit{power}), \sigma(\textit{img}), 2}, \textcircled{\sigma(\textit{neg}), 1}}$ .

Please note that the usual notation for the imaginary unit is  $\sigma(\textit{img})$ . However, electrical engineers will rather use  $\sigma(\textit{img})$ , not to confuse it with the notation for electric current  $\sigma(\textit{current})$ .

We exemplify the rendering process by focusing on three mathematical objects of this input document, namely *ordered pair*, *power*, and *imaginary unit*. For simplicity, we denote a matching notation definition  $\textit{ntn}$  for a symbol  $\sigma(\textit{name})$  as well as its application  $\textcircled{\sigma(\textit{name}), \dots}$  by  $\textit{ntn}^{\textit{name}}$ . In the further course, we discuss the rendering of the chosen object based on the collection of notation definitions from *EC*, *F*, *Doc*, *CD*, and *SD*.

### 5.2.2 Collection from *SD*

*SD* denotes the **system defaults**, which are a fallback if no other notation definitions are given. For the course of this paper, the *system defaults* are not filtered for a particular input object, but can be reused for any mathematical object  $\omega$ . Please note that we expect the rendering systems to implement a more intelligent collection from *SD*. System defaults can be stored into system-internal representation formats as well as into the file system (encapsulated into notation documents) or an appropriate database. We leave the concrete storage to the rendering systems.

Figure 4 presents four system default notation definition, according to which all symbols ( $\textit{ntn}^{\sigma(\textit{name})}$ ) and variables ( $\textit{ntn}^{v(\textit{name})}$ ) are rendered with their *name*, while integers ( $\textit{ntn}^{\textit{integer}}$ ) are rendered with their *value*. Moreover, all applications ( $\textit{ntn}^{\textit{appl}(a,b,\dots)}$ ) are rendered in a prefix-syntax  $\underline{a}(b\dots)$ , such as  $\textit{pair}(a, b)$  for the application  $\textcircled{\sigma(\textit{opair}), v(a), v(b)}$ .

We apply the collection algorithm to the system defaults in the figure above and receive  $\Pi_\omega$  in return. For simplicity, we do not display context annotations and precedences. Please note that the presented *SD* is only an simplified extract of the default notation definitions.

---

1. We collect notation definitions from *SD* yielding  $\Pi_\omega$ :

$$\Pi_\omega = (\textit{ntn}^{\sigma(\textit{name})}, \textit{ntn}^{v(\textit{name})}, \textit{ntn}^{\textit{integer}}, \textit{ntn}^{\textit{appl}(a,b,\dots)})$$

$$\Pi_\omega = (\varphi_1 \vdash \textit{name}, \varphi_2 \vdash \textit{name}, \varphi_3 \vdash \textit{value}, \varphi_4 \vdash \underline{a}(b\dots))$$

2. We reorganize  $\Pi_\omega$  yielding  $\Pi'_\omega$ :  $\Pi_\omega = (\varphi_1 \vdash \textit{name}, \varphi_2 \vdash \textit{name}, \varphi_3 \vdash \textit{value}, \varphi_4 \vdash \underline{a}(b\dots))$

---

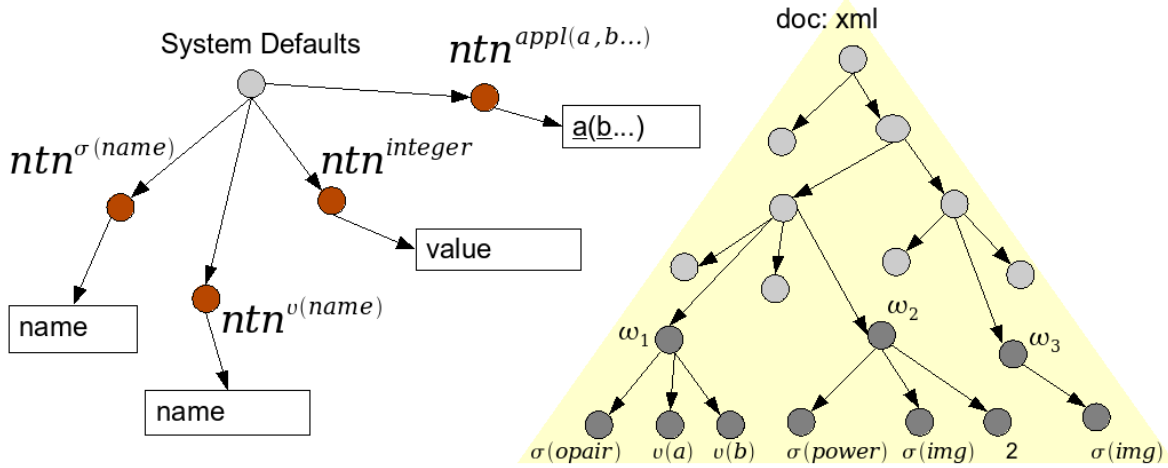


Figure 4: Collecting Notation Definition from the System Defaults

Based on the matching algorithm of Section 3, a matching notation definition is selected for each mathematical object. For simplicity, we omit the rendering context and select the first available rendering  $\rho$  (in this case, only one rendering is available). The concepts in the input document of Section 5.2.1 are presented as follows: For example, the imaginary unit  $\sigma(img)$  is presented by its name  $img$  and the application  $@(\sigma(power), \sigma(img), 2)$  is rendered to a prefix syntax  $(power(img), 2)$ .

In mathematics, a complex number is a number which can be formally defined as an ordered pair of real numbers  $pair(a, b)$ , often written:  $plus(a, times(b, img))$  where  $equal(power(img), 2), neg(1))$ .

Please note that the usual notation for the imaginary unit is  $img$ . However, electrical engineers will rather use  $img$ , not to confuse it with the notation for electric current  $current$ .

When looking at the resulting presentation of the input document, we can clearly see that the rendering strongly diverges from the intended presentation in Section 5.2.1. In particular, it is not possible to use user-specific notations or alternative notations for the same mathematical object. Although, the system defaults allow for the rendering of documents, they are only intended to be fallbacks if no other notation definitions are provided.

### 5.2.3 Collecting from $F$

$F$  denotes an **external notation document** from which we collect all notation definitions. Consequently, the resulting list of collected notations may be reused for any mathematical object  $\omega$ . Please note that we expect the rendering systems to implement a more intelligent collection from an explicitly provided notation document.

We apply the collection algorithm to the notation document  $myntn$  in Figure 5 with  $\mathcal{S}^N = (F, SD)$ , using the system defaults of Section 5.2.2, and receive  $\Pi_\omega$  in return. For simplicity, we do not display context annotations and precedences.

1. We collect all notation definitions yielding  $\Pi_\omega$ 
  - 1.1. We collect notation definitions from  $F$  yielding  $\Pi_\omega$ :  
 $\Pi_\omega = (ntn^{img}, ntn^{opair}) = (\varphi_1 \vdash i, j, \varphi_2 \vdash [\underline{a}, \underline{b}], (\underline{a}, \underline{b}))$

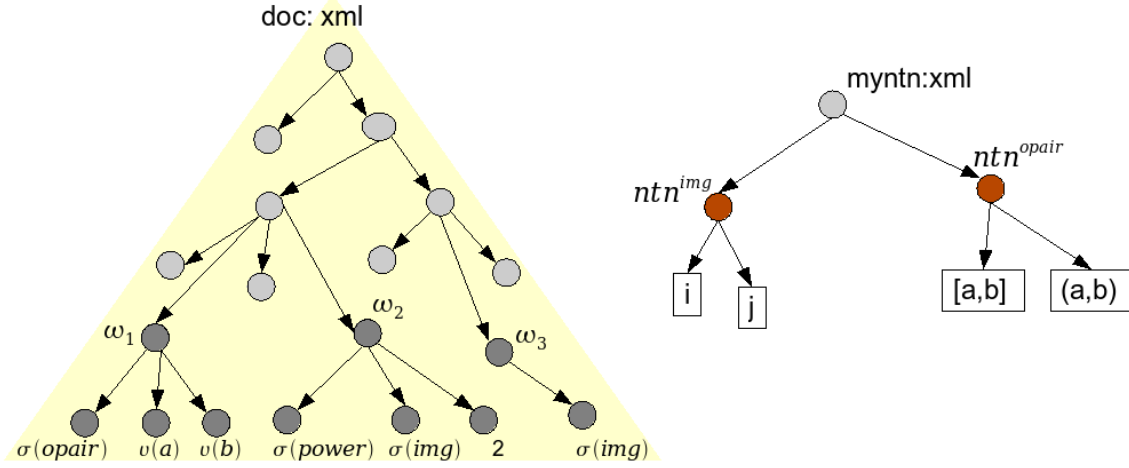


Figure 5: Collecting Notation Definition from an External File

1.2. We collect notation definitions from  $SD$  and append them to  $\Pi_\omega$ :

$$\Pi_\omega = (ntn^{img}, ntn^{opair}, ntn^{\sigma(name)}, ntn^{v(name)}, ntn^{integer}, ntn^{appl(a,b...)} )$$

$$\Pi_\omega = ( \varphi_1 \vdash i, j, \varphi_2 \vdash [a, \underline{b}], (\underline{a}, \underline{b}), \varphi_3 \vdash name, \varphi_4 \vdash name, \varphi_5 \vdash value, \varphi_6 \vdash \underline{a}(\underline{b}...) )$$

2. We reorganize  $\Pi_\omega$  yielding  $\Pi'_\omega$ :

$$\Pi'_\omega = ( \varphi_1 \vdash i, j, \varphi_2 \vdash [a, \underline{b}], (\underline{a}, \underline{b}), \varphi_3 \vdash name, \varphi_4 \vdash name, \varphi_5 \vdash value, \varphi_6 \vdash \underline{a}(\underline{b}...) )$$

Based on the matching algorithm of Section 3, a matching notation definition is selected for each mathematical object. For simplicity, we omit the rendering context and select the first available rendering  $\rho$ . Please note, that the notation context above includes several matching prototypes for a mathematical expression. For example, the symbol  $\sigma(img)$  matches  $\varphi_1$  as well as the more general  $\varphi_3$ . The pattern matcher selects the first matching  $\varphi$  and neglects all others (in our case  $\varphi_2$ ). Consequently, the order of the prototypes in  $\Pi$  influences the rendering result. We recommend to put special prototypes in front of more general ones, using the latter as a fallback for less specific mathematical expression. However, the concrete order of sources is up to the user.

Based on  $\Pi_\omega$  above, the input document of Section 5.2.1 is presented as follows:

In mathematics, a complex number is a number which can be formally defined as an ordered pair of real numbers  $[a, b]$ , often written:  $plus(a, times(b, i))$  where  $equal(power(i, 2), neg(1))$ .

Please note that the usual notation for the imaginary unit is  $i$ . However, electrical engineers will rather use  $\mathbf{i}$ , not to confuse it with the notation for electric current  $current$ .

Based on  $F$ , users can specify their individual notations but are not supported to choose among alternative notations on granular document level. For example, the imaginary unit is presented with  $i$  throughout the whole document. But authors want to use more than one notation for the same symbol, e.g. to introduce new notations or to compare to alternative ones. On the contrary, other users want to adapt documents on a granular level. Moreover, with the current approaches, users have to specify all notation definition for their mathematical objects and cannot reuse or import them from other sources, such as online content dictionaries (cf. Section 5.2.4). For example, the appli-

cation  $@(\sigma(\text{power}), \sigma(\text{img}), 2)$  was rendered to  $\text{power}(\text{img}, 2)$  rather than the commonly accepted notation  $i^2$ .

### 5.2.4 Collection from CD

A **Content Dictionary** [NW01a] is a structured document, which includes a collection of definitions for mathematical concepts. Optionally, it includes default notation definitions for the symbols it defines or is associated with a complementary notation document. The definitions provide the meaning of concepts, while the default notation definitions can be used to present them.

*CD* denotes the **content dictionaries** of the symbols occurring in  $\omega$ . These are searched in the order in which the symbols occur in  $\omega$ . This can be identified by evaluating the symbols `cdbase`, `cd`, and `name` attributes.

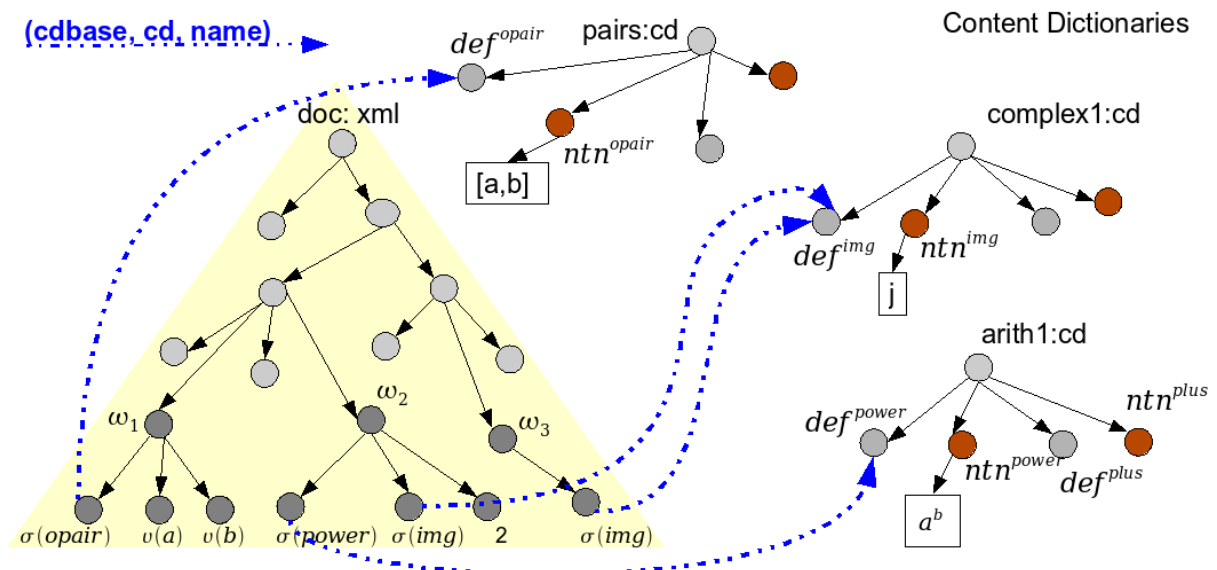


Figure 6: Collecting Notation Definition from Content Dictionaries

Figure 6 associates the symbols  $\sigma(\text{opair})$ ,  $\sigma(\text{power})$ , and  $\sigma(\text{img})$  with their respective definitions in the content dictionaries. The dashed arrows in the figure represent the implicit references from the symbols in the input document to their notation definitions in the content dictionaries by resolving their `cdbase`, `cd`, and `name` attributes.

We apply the collection algorithm with the input object  $\omega_2 = @(\sigma(\text{power}), \sigma(\text{img}), 2)$  and  $\mathcal{S}^N = (F, CD, SD)$ , using the *myntn* file from Section 5.2.3 and the *SD* from Section 5.2.2, and receive  $\Pi_{\omega_2}$  in return. For simplicity, we do not display context annotations and precedences and omit the default specifications of *SD*.

- 
1. We collect all notation definitions yielding  $\Pi_{\omega_2}$ 
    - 1.1. We collect notation definitions from *F* yielding  $\Pi_{\omega_2}$ :  
 $\Pi_{\omega_2} = \Pi_{\omega} = ( ntn^{img}, ntn^{opair} ) = ( \varphi_1 \vdash i, j, \varphi_2 \vdash [\underline{a}, \underline{b}], (\underline{a}, \underline{b}) )$
    - 1.2. We collect notation definitions from *CD* and append them to  $\Pi_{\omega_2}$ :  
 $\Pi_{\omega_2} = ( ntn^{img}, ntn^{opair}, ntn^{power}, ntn^{img} ) = ( \varphi_1 \vdash i, j, \varphi_2 \vdash [\underline{a}, \underline{b}], (\underline{a}, \underline{b}), \varphi_3 \vdash \underline{a}^{\underline{b}}, \varphi_1 \vdash i )$
  2. We reorganize  $\Pi_{\omega_2}$  yielding  $\Pi'_{\omega_2}$ :  $\Pi'_{\omega_2} = ( \varphi_1 \vdash i, j, i, \varphi_2 \vdash [\underline{a}, \underline{b}], (\underline{a}, \underline{b}), \varphi_3 \vdash \underline{a}^{\underline{b}} )$
-

Constructing the  $\Pi_{\omega_i}$  as described above allows to present the input document of Section 5.2.1 as follows:

In mathematics, a complex number is a number which can be formally defined as an ordered pair of real numbers  $[a, b]$ , often written:  $plus(a, times(b, i))$  where  $equal(i^2, neg(1))$ .

Please note that the usual notation for the imaginary unit is  $i$ . However, electrical engineers will rather use  $j$ , not to confuse it with the notation for electric current *current*.

The collection of notation definitions from content dictionaries, relieves users from defining all notations on their own. For example, the *CD* default for the  $\sigma(power)$  symbol was used to render  $i^2$  rather than  $power(i, 2)$ . Users may use *F* to overwrite the *CD*. For example, instead of rendering the ordered pair with the *CD* default  $(a, b)$ , the notation definition in *F* was used to render  $[a, b]$ . However, users are still not able to choose different notations for the same symbol inside the document. Consequently, we analyse whether the collection from the input document facilitates a more granular use and adaptation of notations.

### 5.2.5 Collection from *Doc*

*Doc* denotes the input document. Authors may embed notation definitions into their document to personalize a document with their own notation styles. These are searched according to the position of a respective mathematical object in the document. In particular, we only consider notation definitions to the left of the mathematical object in the document tree. Moreover, local notation definitions, which are *closer* to the object have a higher precedence than more global ones (cf. Section 6.2).

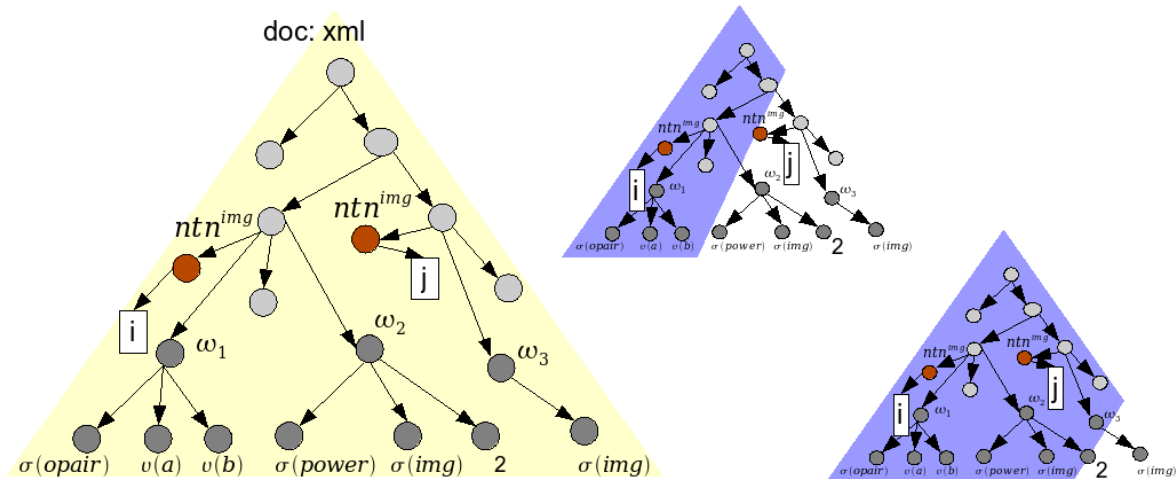


Figure 7: Collecting Document-Internal Notation Definition

Figure 7 above displays the input document and its embedded notation definitions for the symbol  $\sigma(img)$ . When considering the position of the mathematical objects, we collect different sets of notation definitions: For example, we collect one notation definition for  $\omega_2$  (cf. document in the middle), while we can find two notation definitions to the left of  $\omega_3$  (cf. document to the right).

We apply the collection algorithm with the input object  $\omega_3 = \sigma(img)$  and  $\mathcal{S}^N = (Doc, CD)$ , using the  $SD$  from Section 5.2.2, and receive  $\Pi_{\omega_3}$  in return. For simplicity, we do not display context annotations and precedences and omit the default specifications of  $SD$ .

---

1. We collect all notation definitions yielding  $\Pi_{\omega_3}$

1.1. We collect notation definitions from  $Doc$  yielding  $\Pi_{\omega_3}$ :

$$\Pi_{\omega_3} = ( ntn^{img}, ntn^{img} ) = ( \varphi_1 \vdash j, \varphi_1 \vdash i )$$

(The order of the notation definition depends on their distance to the object. More local ones are preferred.)

2. We reorganize  $\Pi_{\omega_3}$  yielding  $\Pi'_{\omega_3} : \Pi'_{\omega_3} = ( \varphi_1 \vdash j, i )$

---

The input document of Section 5.2.1 is presented as follows:

In mathematics, a complex number is a number which can be formally defined as an ordered pair of real numbers  $[a, b]$ , often written: *plus(a, times(b, i))* where *equal(i<sup>2</sup>, neg(1))*.

Please note that the usual notation for the imaginary unit is  $i$ . However, electrical engineers will rather use  $j$ , not to confuse it with the notation for electric current *current*.

The collection of notation definitions from the input document allows user to specify alternative notation on granular document level. However, the granularity depends on the document format, that is the level of granularity on which notation definition can be embedded inside the document. In the given input document, the author is using different notations inside the same paragraph, which may not be support by all document formats.

Moreover, the embedding of notation definitions forces authors to take extreme care, since they might overwrite their actually preferred notations. Moreover, when switching back and forth between notations for the same symbol, authors have to insert several copies of the respective notation definition and, based on the default selection, need to modify the order of the renderings or even filter them. For example, in figure above the  $ntn^{img}$  include only one rendering for the symbol  $\sigma(img)$  to present  $i$  or  $j$ , respectively. This complicates the change and maintenance of notation definitions and causes redundancy. Moreover, when adapting the document based on the user-specific notation definitions in  $F$ , the author's granular notations are overwritten, taking away his intention and the meaning of the text.

### 5.2.6 Collection from $EC$

$EC$  denotes the **extensional context**, which associate a list of notation definitions or containers of notation definitions to every node of the input document. The effective extensional context is computed according to the position of  $\omega$  in the input document.  $EC$  is used by authors to reference their individual notation context.

Figure 8 associates specific sections of the input document with notation definitions or container of notation definition. The dashed arrows in the figure represent extensional references: For example, the `ec` attribute of the document root `doc` references the notation document “`myntn`”, which is interpreted as a container of notation definitions. Please note that extensional references can point to notation documents in the file system as well as respective database entries. We leave the concrete implementation to the rendering system.



specifications with their individual styles in  $F$ , which may reduce the understandability of the document. Consequently, we need a mechanism that facilitates the granular adaptation of documents without modifying the document itself and which prevents the overwriting of meaningful notations.

To implement  $EC$  in arbitrary XML-based document formats, we propose an `ec` attribute in a namespace for notation definitions, which may occur on any element. The value of the `ec` attribute is a whitespace-separated list of URIs of either notation definitions or any other document. The latter is interpreted as a container, from which notation definitions are collected. The `ec` attribute is empty by default. When computing the effective extensional context of an element, the values of the `ec` attributes of itself and all parents are concatenated, starting with the inner-most.

### 5.2.7 Summary/ Findings

In the previous sections, we discussed the specific algorithms for collecting notation definitions from  $EC$ ,  $F$ ,  $Doc$ ,  $CD$  and  $SD$  and illustrated the advantages and drawbacks of basing the rendering on either one of the sources. We conclude with the following findings:

1. Authors can write documents which only include content markup and do not need to provide any notation definitions. The notation definitions are then collected from  $CD$  and  $SD$ .
2. The external document  $F$  permits authors to store their notation definitions centrally, facilitating the maintenance of notational preferences. However, authors may not specify alternative notations for the same symbol on granular document levels.
3. Authors may use the content dictionary defaults or overwrite them by providing  $F$  or  $Doc$ .
4. Authors may embed notation definitions inside their documents. However, this causes redundancy inside the document and complicates the maintenance of notation definitions.
5. Users can overwrite the specification inside the document with  $F$ . However, that can destroy the meaning of the text, since the granular notation contexts of the authors are replaced by only one alternative declaration in  $F$ .
6. Collecting notation definitions from  $F$  or  $Doc$  has benefits and drawbacks. Since users want to easily maintain and change notation definitions but also use alternative notations on granular document levels, we provide  $EC$ . This permits a more controlled and more granular specification of notations.

## 6 Choosing Renderings Intensionally

The extensional notation context declarations do not support authors to select between alternative renderings inside one notation definition. Consequently, if relying only on this mechanism, authors have to take extreme care about which notation definition they reference or embed. Moreover, other users cannot change the granular extensional declarations

in *EC* without modifying the input document. They can only overwrite the author’s granular specifications with their individual styles  $F$ , which may reduce the understandability of the document.

Consequently, we need a more *intelligent, context-sensitive* selection of renderings, which lets users guide the selection of alternative renderings. We use an intensional rendering context  $\Lambda$ , which is matched against the context annotations in the notation definitions. In the following, we discuss the collection of contextual information from various sources and the construction of  $\Lambda_\omega$  for a concrete mathematical object  $\omega$ .

## 6.1 Collecting Contextual Information

We represent contextual information by contextual key-value pairs, denoted by  $(k_i = v_i)$ . The key represents a *context dimension*, such as *language*, *level of expertise*, *area of application*, or *individual preference*. The value represents a *context value* for a specific context dimension. The algorithm for the context-sensitive selection takes as input an object  $\omega$ , a list  $L$  of elements of the form  $(\lambda : \rho)^p$ , and a totally ordered set  $\mathcal{S}^C$  of source names. We allow the names *GC*, *CCF*, *IC*, and *MD*. The algorithm returns a pair  $(\rho, p)$ .

The selection algorithm consists of two steps: The collection of contextual information  $\Lambda_\omega$  and the selection of a rendering. In the first step  $\Lambda_\omega$  is computed by processing the input sources in the order given by  $\mathcal{S}^C$ . The respective input sources are treated as follows:

- *GC* denotes the **global context** which provides contextual information during *rendering time* and overwrites the author’s intensional context declarations. The respective  $(k_i = v_i)$  can be collected from a user model or are explicitly entered. *GC* typically occurs first in  $\mathcal{S}^C$ .
- *CCF* denotes the **cascading context files**, which permit the contextualization analogous to cascading stylesheets[Cas99].
- *IC* denotes the **intensional context**, which associates a list of contextual key-value pairs  $(k_i = v_i)$  to any node of the input document. These express the author’s intensional context. For the implementation in XML formats, we use an *ic* attribute similar to the *ec* attribute above, i.e., the effective intensional context depends on the position of  $\omega$  in the input document.
- *MD* denotes **metadata**, which typically occurs last in  $\mathcal{S}^C$ .

In the second step, the rendering context  $\Lambda_\omega$  is matched against the context annotations in  $L$ . We select the pair  $(\rho, p)$  whose corresponding context annotation satisfies the intensional declaration best.

## 6.2 Matching the Rendering Context

The matching algorithm takes as input

- a rendering context  $\Lambda$ , i.e. a list of contextual key-value pairs  $(k_i = v_i)$ , and
- a list  $L$ , i.e. a list of elements of the form  $(\lambda : \rho)^p$ .  $\lambda$  denotes a context annotation, i.e. a list of key-value of the form  $(k_i = v_i)$ ,  $\rho$  denotes a rendering element, and  $p$  denotes the precedence.

In statistical terms, the *renderings*  $\rho$  denote *objects* and the context annotations  $\lambda$  denote lists of *characteristics-value* pairs, whereas  $k$  denotes the characteristic and  $v$  denotes a concrete value. Moreover,  $\Lambda$  denotes the *characteristics-value* pairs of an ideal object  $\rho^*$ . These are matched against the  $\lambda_i$  of the given objects  $\rho_i$  to compute their *distance*. The object with the smallest distance to  $\rho^*$  is selected.

- 
1. We build a data matrix based on  $\Lambda$  and  $L$ . We list all objects, their characteristics, and respective values.
  2. We build a distance matrix to compute the distance between the objects.
  3. We select the object  $\rho_i$  with the smallest distance to the ideal object  $\rho^*$ .
- 

In the following example, we select the best matching notation for the binomial coefficient based on a concrete rendering context and a given  $L$ . For simplicity, we omit the precedences. Please note, that the given contextual information is only exemplary and not necessarily realistic.

---


$$L = [ (\lambda_0 = \binom{n}{k}), (\lambda_1 = C_n^k), (\lambda_2 = C_k^n) ]$$

$$\lambda_0 = (\text{lang} = \text{de}, \text{lang} = \text{it}, \text{level} = \text{easy}), \lambda_1 = (\text{lang} = \text{fr}, \text{lang} = \text{gr}, \text{level} = \text{difficult}), \text{ and}$$

$$\lambda_2 = (\text{lang} = \text{rus}, \text{level} = \text{difficult})$$

$$\Lambda = (\text{lang} = \text{de}, \text{lang} = \text{gr}, \text{level} = \text{easy})$$


---

## Constructing a Data Matrix

Characteristic Object	language	level of expertise
?	de, gr	easy
$\binom{n}{k}$	de, it	easy
$C_n^k$	fr, gr	difficult
$C_k^n$	rus	difficult

## Constructing a Distance Matrix

- 
1. We create a distance matrix for each characteristics, whereas multiple occurrence of a characteristics in  $\Lambda$  are considered: Here we build  $D_{lang1}$ ,  $D_{lang2}$ , and  $D_{level}$ 
    - 1.1. We need to create *scales* for the characteristics *language* and *level of expertise*, which are used to associate a *numeric value* to the given values:
      - 1.1.1 We use a *nominal scale* for nominal characteristics such as *language*:  
 if  $v_{ik} \neq v_{jk}$  then  $f(v_{ik}) \neq f(v_{jk})$ .  
 if  $v_{ik} = v_{jk}$  then  $f(v_{ik}) = f(v_{jk})$ .
      - 1.1.2. We use an *ordinal scale* for ordinal characteristics such as *level of expertise*:  
 $f(\text{easy}) = 1$ ,  $f(\text{intermediate}) = 2$ , and  $f(\text{difficult}) = 3$
    - 1.2. We compute the distance index as follows:
      - 1.2.1. The distance index for the nominal scale is computed:  

$$d_k(i, j) = \begin{cases} 0 & \text{for } f(v_{ik}) = f(v_{jk}) \\ c & \text{for } f(v_{ik}) \neq f(v_{jk}) \end{cases}$$
      - 1.2.2. The distance index for the *ordinal scale* is computed:  

$$d_k(i, j) = |f(v_{ik}) - f(v_{jk})|$$
  2. We weight each matrix according to the order in  $\Lambda$ . Here we associate the weights  $w(D_{lang1}) = 3$ ,  $w(D_{lang2}) = 2$ , and  $w(D_{level}) = 1$
  3. We compute an overall distance matrix  $D$
- 

We apply the algorithm above, to select the most preferred rendering  $\rho$ . We assume  $c = 3$ .

$D_{lang1}$	$\binom{n}{k}$	$C_n^k$	$C_k^n$
$\rho^*$	0	3	3
$\binom{n}{k}$	/	3	3
$C_n^k$	/	/	3
$C_k^n$	/	/	/

$D_{level}$	$\binom{n}{k}$	$C_n^k$	$C_k^n$
$\rho^*$	0	2	2
$\binom{n}{k}$	/	2	2
$C_n^k$	/	/	0
$C_k^n$	/	/	/

$D_{lang2}$	$\binom{n}{k}$	$C_n^k$	$C_k^n$
$\rho^*$	3	0	3
$\binom{n}{k}$	/	3	3
$C_n^k$	/	/	3
$C_k^n$	/	/	/

$D$	$\binom{n}{k}$	$C_n^k$	$C_k^n$
$\rho^*$	6	11	17
$\binom{n}{k}$	/	17	17
$C_n^k$	/	/	15
$C_k^n$	/	/	/

Consequently, we choose  $\binom{n}{k}$  since it is the *closest* object to the ideal object  $\rho^*$ . The resulting distance index let us not only choose the most preferred rendering, but also allow to compare notations in terms of their contextual distance. For example,  $C_n^k$  and  $C_k^n$  ( $d = 15$ ) are closer than  $C_n^k$  and  $\binom{n}{k}$  ( $d = 17$ ).

### 6.3 Discussion of Context-Sensitive Selection Strategies

In the following sections, we provide the algorithms for the collection of contextual information from *GC*, *CCF*, *IC* and *MD* in detail and illustrate the advantages and drawbacks of basing the rendering on either one of the sources.

#### 6.3.1 Collecting from *GC*

*GC* denotes the **global context** which provides contextual information during *rendering time* and overwrites the author's intensional context declarations. The respective ( $k_i = v_i$ ) can be collected from a user model or are explicitly entered.

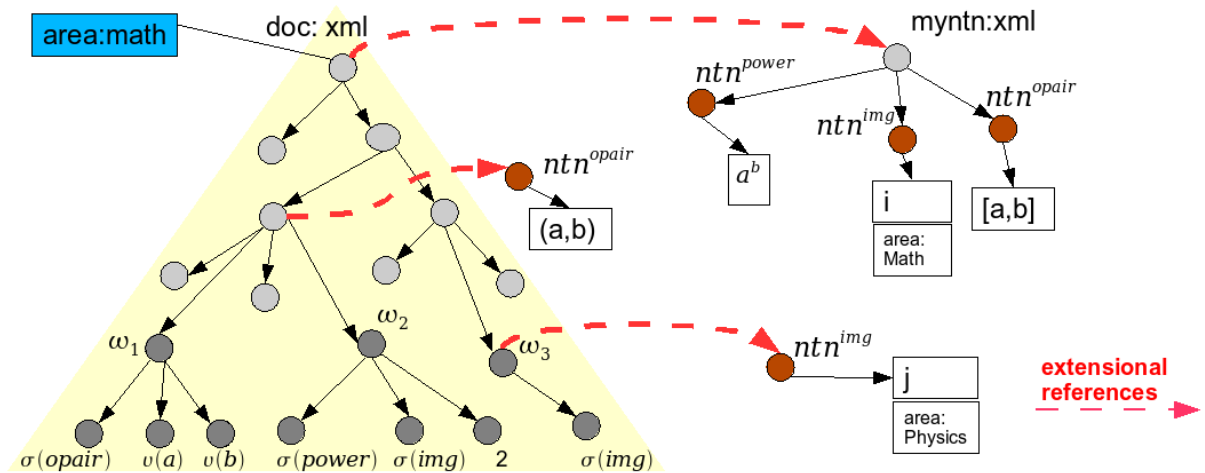


Figure 9: Selection of Renderings based on a Global Context

Figure 9 provides a *global context* for the given input document, which specifies the language and course dimension of the document. We apply the selection algorithm with the input object  $\omega_3$ ,  $\mathcal{S}^C = (GC)$ , and a list of context annotations and rendering pairs based on the formerly created notation context  $\Pi_{\omega_3}$  (cf. Section 5.2.6). For convenience, we do not display the precedences.

---

Given is the notation context  $\Pi_{\omega_3} = (\varphi_1 \vdash j, i, \varphi_2 \vdash \underline{a}^b, \varphi_3 \vdash [a, b])$ . Based on the matching algorithm of Section 3, the matching prototype  $\varphi_1$  is selected. Consequently,  $L = [(\lambda_0 = j), (\lambda_1 = i)]$  is provided.

1. We compute  $\Lambda$  by collecting contextual information from *GC*:  $\Lambda_{\omega_3} = (area = math)$
  2. We match the rendering context against the given context annotations,  $\lambda_0 = (area = physics)$  and  $\lambda_1 = (area = maths)$ , by applying the algorithm of section 6.2 ( $c = 3$ ). We compute the following distance measure:  $d_0 = 3$  and  $d_1 = 1$  for the distance between  $\rho^*$  to  $\rho_0$  and  $\rho_1$ , respectively. Consequently,  $\rho_1 = i$  is the best matching rendering.
- 

The declaration of a *global context* provides a mechanism that allows a more intelligent selection of renderings than the primitive selection of Section 5, in which we omitted the rendering context and simply selected the first available rendering. Instead of changing the extensionally defined set of preferred notation definitions, users can guide the notation selection function via global attributions while reusing the extensional settings. However, the use of a global context is limited in that it does not facilitate users to specify contexts on arbitrary granular level. Moreover, users can overwrite the author’s granular specifications and thereby destroy the meaning of the text. Consequently, we need a more granular approach.

### 6.3.2 Collecting from *MD*

*MD* denotes **metadata**, which is document internal contextual information. Below we propose how the Dublin Core metadata [Dub08] can be interpreted to guide the selection of notations.

metadata	Context Dimension	Interpretation
dc:language	language	Select all German notation definitions.
dc:creator dc:publisher dc:contributor	individual style	Select all notation definitions of the individuals style.
dc:format	media	Select all notation definitions for print media rather than dynamic notations for living documents.
dc:type	type	Select notation definitions for a specific document type.
dc:date	historical period	Select between older or more modern notation definitions: For instance, the older use of lines for grouping, e.g. $3\overline{a + b}$ , versus the modern use of parentheses, e.g. $3(a + b)$ (cf. [SW06b]).
dc:relation dc:source	group preference	Select notation definitions from other documents, e.g. the source document or cited documents.

The effective intensional context based on metadata is computed according to the position of  $\omega$  in the input document. In particular, local metadata have a greater precedence than more global ones. Moreover, the order of the metadata tags defines their priority.

Figure 10 provides the given input document enriched by metadata: The document

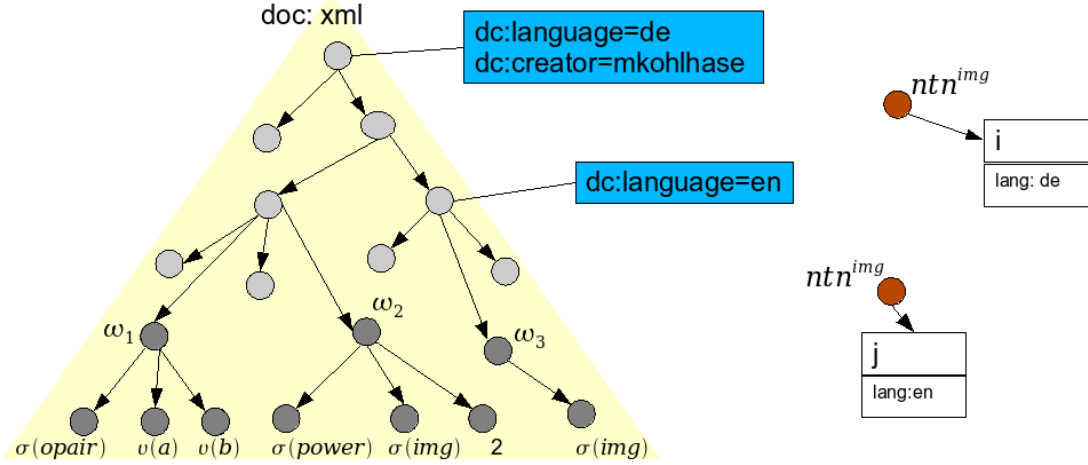


Figure 10: Selection of Renderings based on metadata

is described by the metadata tags *language* and *creator*. One of its subsections has an additional metadata tag *language*. For the sake of this example, we modified the context annotation of  $ntn^{img}$ . We generate the  $\Lambda$  from the metadata and apply the selection algorithm with the input object  $\omega_3$ ,  $\mathcal{S}^C = (MD)$ , and a list of context annotations and rendering pairs based on the formerly created notation context  $\Pi_{\omega_3}$  (cf. Section 5.2.6). For convenience, we do not display the precedences.

---

Given is the notation context  $\Pi_{\omega_3} = (\varphi_1 \vdash j, i, \varphi_2 \vdash \underline{a}^b, \varphi_3 \vdash [\underline{a}, \underline{b}])$ . Based on the matching algorithm of Section 3, the matching prototype  $\varphi_1$  is selected. Consequently,  $L = [(\lambda_0 = j), (\lambda_1 = i)]$  is provided.

1. We compute the intensional rendering context by collecting contextual information from  $MD$ :  $\Lambda_{\omega_3} = (lang = en, lang = de, creator = mkohlhase)$
  2. We match the rendering context against the given context annotations,  $\lambda_0 = (lang = en)$  and  $\lambda_1 = (lang = de)$ , by applying the algorithm of section 6.2 ( $c = 3$ ). We compute the following distance measure:  $d_0 = 9$  and  $d_1 = 12$  for the distance between  $\rho^*$  to  $\rho_0$  and  $\rho_1$ , respectively. Consequently,  $\rho_0 = j$  is the best matching rendering for  $\omega_3$ .
- 

Considering metadata is a more granular approach than the declaration of a global context. However, metadata can not be associated to any node in the XML tree. Moreover, the context dimension are limited based on the specification of the metadata format. The metadata approach does not facilitate other users to select notations, since metadata is document internal and, consequently, only set by the author. Moreover, we need a mechanism to *translate* metadata into our contextual key-value pairs in order to construct  $\Lambda$ .

### 6.3.3 Collecting from $IC$

$IC$  denotes the **intensional context**, which associates a list of contextual key-value pairs ( $k_i = v_i$ ) to any node of the input document. These express the author's intensional context. For the implementation in XML formats, we use an `ic` attribute similar to the `ec` attribute above, i.e., the effective intensional context depends on the position of  $\omega$  in the input document.

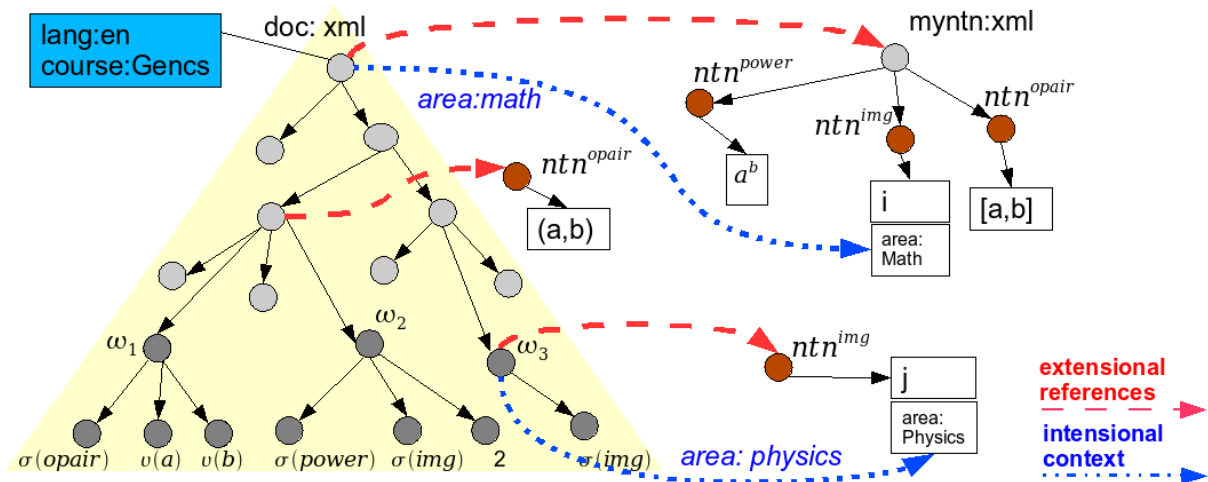


Figure 11: Selection of Renderings based on Intensional References

Figure 11 associates the input document with intensional context declarations. The dashed arrows represent extensional references, the dashed-dotted arrows represent intensional references, i.e., implicit relations between the `ic` attributes of the input document and the context-annotations in the notation document. A global context is declared, which specifies the language and course dimension of the document. We apply the selection algorithm with the input object  $\omega_3$ ,  $\mathcal{S}^C = (GC, IC)$ , and a list of context annotations and rendering pairs based on the formerly created notation context  $\Pi_{\omega_3}$ . For convenience, we do not display precedences.

1. We compute the intensional rendering context

1.1. We collect contextual information from  $GC$

$$\Lambda_{\omega_3} = (lang = en, course = GenCS)$$

1.2. We collect contextual information from  $IC$  and append them to  $\Lambda_{\omega_3}$

$$\Lambda_{\omega_3} = (lang = en, course = GenCS, area = physics, area = maths)$$

2. We match the rendering context against the given context annotations,

$\lambda_0 = (area = physics)$  and  $\lambda_1 = (area = maths)$ , by applying the algorithm of section 6.2 ( $c = 3$ ). We compute the following distance measure:  $d_0 = 24$  and  $d_1 = 30$  for the distance between  $\rho^*$  to  $\rho_0$  and  $\rho_1$ , respectively. Consequently,  $\rho_0 = j$  is the best matching rendering.

The `ic` attribute may be compared to the HTML `style` attribute [W3C02]. However, in contrast to the `style` attribute, we allow multiple values for a specific context dimension. Moreover, we do not overwrite context declarations, but rather append the more global specification to the local ones.

The `ic` attribute provides a mechanism that allows a more intelligent and granular selection of renderings by associating an intensional context any node of the document. Consequently, users do not need to modify or filter the rendering of their notation definitions, but may point to their central notation collection and use the context-attributions to select a specific rendering. However, the approach is limited in that it does not facilitate users to overwrite these document internal attributes on a granular level. For this matter, we propose *Cascading Context Files*, which facilitate to overwrite the author's granular context declaration during rendering time.

### 6.3.4 Collecting from CCF

CCF denotes the **cascading context files**, which permit the contextualization analogously to cascading stylesheets. They can be used by users to contextualize the input document on a granular document level without modifying the document itself.

According to CSS selectors, we introduce CCF selectors for attributing different context specifications:

- *Simple Selectors*: Specify a context for a group of elements.
- *Nested Selectors*: Specify a context for a nested group of elements. Nested selectors have a higher precedence than simple selectors.
- *Class selectors*: Contextualize an element via a `class` attribute. Class selectors have a higher precedence than nested selectors.
- *ID selectors*: Contextualize an element via an `id` attribute. They have the highest precedence.

Analogous to the `style`, we propose a `context` element to include context files into the input document.

Below we provide an XML representation of the input document of Section 5.2.1.

---

```
<document id="gens">
  <definition>
    <p>
      In mathematics, a complex number is a number which can be formally defined as an ordered pair
      of real numbers  $\langle \sigma(\text{opair}) \langle /math \rangle, v(a), v(b) \rangle$ , often written
      :  $\langle \sigma(\text{plus}), v(a), \langle \sigma(\text{times}), v(b), \sigma(\text{img}) \rangle \rangle$  where  $\langle \sigma(\text{equal}), \langle \sigma(\text{power}), \sigma(\text{img}), 2 \rangle, \langle \sigma(\text{neg}), 1 \rangle \rangle$ .
    </p>
    <p>
      Please note that the usual notation for the imaginary unit is  $\sigma(\text{img})$ . However, electrical engineers will rather use  $\sigma(\text{img})$ 
       $\sigma(\text{img})$ , not to confuse it with the notation for electric current  $\sigma(\text{img})$ 
       $\sigma(\text{current})$ .
    </p>
  </definition>
</document>
```

---

The CCF file below specifies the context of the document on a granular level:

---

```
document { /*simple element selector*/
  lang:en;
}
#gens { /*id selector*/
  course:GenCS;
  area:math;
}
.physic { /*class selector*/
  area:physics
}
definition p { /*nested element selector */
}
```

---

We apply the selection algorithm with the input object  $\omega_3$ ,  $\mathcal{S}^C = (CCF)$ , and a list of context annotations and rendering pairs based on the formerly created notation context  $\Pi_{\omega_3}$  (cf. Section 5.2.6). For convenience, we do not display the precedences.

---

Given is the notation context  $\Pi_{\omega_3} = (\varphi_1 \vdash j, i, \varphi_2 \vdash \underline{a}^b, \varphi_3 \vdash [\underline{a}, \underline{b}])$ . Based on the matching algorithm of Section 3, the matching prototype  $\varphi_1$  is selected. Consequently,  $L = [(\lambda_0 = j), (\lambda_1 = i)]$  is provided.

1. We compute  $\Lambda$  by collecting contextual information from *CCF*:  
 $\Lambda_{\omega_3} = (\text{area} = \text{physics}, \text{course} = \text{GenCS}, \text{area} = \text{math}, \text{lang} = \text{en})$
  2. We match the rendering context against the given context annotations,  $\lambda_0 = (\text{area} = \text{physics})$  and  $\lambda_1 = (\text{area} = \text{maths})$ , by applying the algorithm of section 6.2 ( $c = 3$ ). We compute the following distance measure:  $d_0 = 18$  and  $d_1 = 24$  for the distance between  $\rho^*$  to  $\rho_0$  and  $\rho_1$ , respectively. Consequently,  $\rho_0 = j$  is the best matching rendering.
- 

*CCF* facilitate the declaration of intensional contexts on arbitrary granular level. However, in some cases users should be prevented to overwrite the author's declaration. Moreover, when using *CCF* we need to provide a respective parsing and translating of the encoded contextual information into our key-value pairs in order to construct  $\Lambda$ .

### 6.3.5 Summary/ Findings

In the previous sections, we illustrate and evaluate the collection of contextual information from *GC*, *CCF*, *IC* and *MD* in detail. We conclude with the following findings:

1. The declaration of a *global context* provides a more intelligent intensional selection between alternative  $(\lambda : \rho)^p$  triples inside one notation definition: The globally defined  $(k_i = v_i)$  are matched against the context-annotations  $\lambda$  to select an appropriate rendering  $\rho$ . However, the approach does not let users specify intensional contexts on granular levels.
2. Considering *metadata* is a more granular approach than the global context declaration. However, metadata may not be associated to any node in the input document and cannot be overwritten without modifying the input document. Moreover, the available context dimensions and values are limited by the respective metadata format.
3. The *intensional context* supports a granular selection of renderings by associating an intensional context to any node of the input document. However, the intensional references cannot be overwritten on granular document levels.
4. *Cascading Context Files* permit a granular overwriting of contexts.

## 7 Open Questions

### 7.1 Write Protection

In some cases, users should be prevented to overwrite the author's declaration, i.e. the intensional specification of the set of available notation definitions as well as the intensional context declarations. One could argue that *write protection* is rather the responsibility of the authoring environment and viewer. However, we claim that it needs to be encoded into the document itself in order to preserve the access rights of the document during its exchange between users and systems.

We see the following approaches for introducing a write protection onto documents:

1. The `ec` and `ic` attributes are interpreted as a write protection: That is, the notation definitions referenced by the `ec` and the intensional context declaration of the `ic` attribute can not be overwritten by the users via a separate notation document  $F$  or context declaration ( $GB$  or  $CCF$ ). This may be enforced via the order of the sources in  $\mathcal{S}^N$  and  $\mathcal{S}^C$ , respectively. However, these static settings reduce the flexibility and adaptability of a document. Moreover, users are now restricted in influencing the order of sources in  $\mathcal{S}^N$  and  $\mathcal{S}^C$ .
2. We introduce two attributes `lec` and `lic` to explicitly *lock* the extensional and intensional declaration inside the input document. `lec` and `lic` are applied according to `ec` and `ic`, only that they have the highest precedence of all sources, i.e. they are *inserted* at the beginning of  $\mathcal{S}^N$  and  $\mathcal{S}^C$ . Moreover, we introduce a `blec` and `blic` attribute that allows authors to *block* declarations that they want to prohibit.
3. We introduce a `lock` attribute to *lock* the document-internal specifications, namely the `ec`, `ic`, *metadata* and the document itself. By setting the value of `lock` to *true*, the author prevents the overwriting of his settings. If `lock` is not set, it defaults to *false*, meaning the author does not enforce a write protection.

Protecting notations is appropriate in some scenarios but critical in others. We recommend to use it to protect the comparison between notations as well as for the introduction of new notations. However, the more notations the author *locks*, the more of its individual style he enforces on the reader. Consequently, the flexibility and adaptability of his document decreases. Moreover, if only parts of the document are write protected and the user overwrites all other settings, the consistency may no longer be assured. Consequently, the write protection complicates the adaptability of documents and should be handled with care. We see this responsibly in the author or the respective rendering system (cf. Section 8).

## 7.2 Notation Management

We believe that notation definitions are connected to the narrative structure of a document. Consequently, we propose a mechanism to encapsulate and structure notation definitions that can be referenced from inside the narrative structure of a document, i.e. via *EC* references (cf. Section 5.2.6).

We propose to use a *notation container*, which encapsulates one or more notation definitions or references to notation definitions. A notation container may import other notation containers via its import interface. Moreover, context annotations can be associated to the notation container in addition to the context annotation inside the notation definitions.

To implement the notation container in arbitrary XML-based document formats, we propose a `notations` element [W3C07], which encapsulates notation definitions (implemented by a `notation` elements) and references to notation definitions (represented by an `xinclude`[DMOT01] or `ref` [Koh06] element). Moreover, the container may include zero or many `import` elements, which import other notation containers. Please note, that we are using an `import` rather than an `xinclude` element, since, during the `xinclude` contraction, the latter are substituted by the elements they reference. In contrast, the `import` element is substituted by the list of notation definitions, which are encapsulated by the respective notation container referenced.

We use a `context` attribute to associate context annotations to the container as well as the `rendering` elements of the notation definitions. The `context` attribute of the notation container is implicitly inherited by the descendant elements, i.e. by the encapsulated, referenced, and imported notation definitions and may be extended by the `context` attribute inside these notation definitions. When computing the effective extensional context of a `rendering` element, the values of the `context` attributes of itself and the container are concatenated. In contrast to the `ec` and `ic`, the order of the contextual key-value part in the effective context annotation is irrelevant.

In alternative to the `context` attributes, `class` and `id` attributes may be used to outsource the context annotations into a separate cascading context file (CCF) (cf. Section 6.3.4).

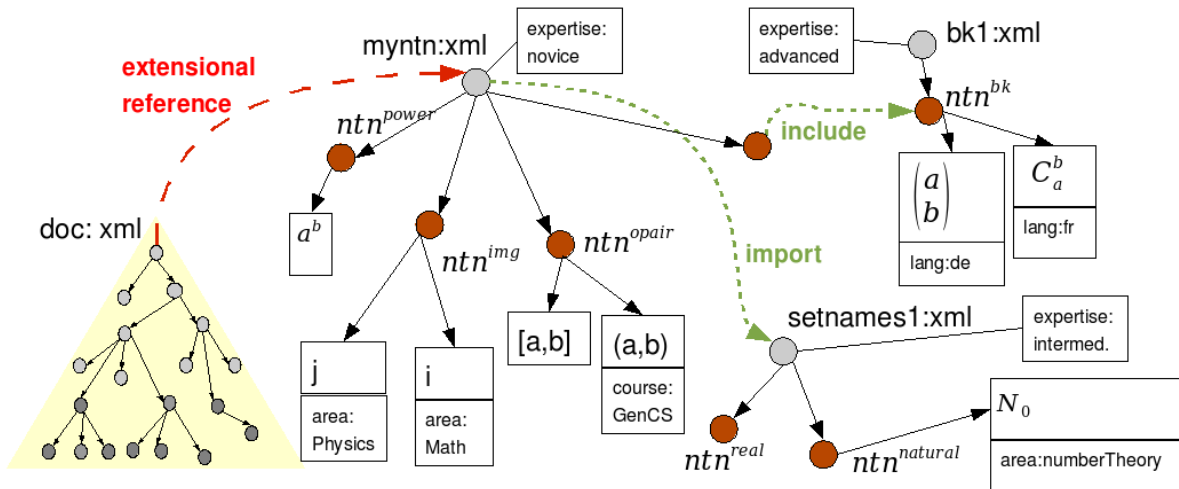


Figure 12: Managing Notation Definition via *notation containers*

Figure 12 associates the input document of Section 5.2.1 with a notation container *myntn* via an `ec` attribute. *myntn* imports the notation container *setnames1* and references the notation definition  $ntn^{bk}$ . The attached context annotations are implicitly inherited by the descendants: For example, the effective context annotation for  $\left(\frac{a}{b}\right)$  is resolved to  $\lambda = (lang = de, expertise = novice)$ . The context annotation of the container *bk1* was not preserved. Instead, the referenced notation definition  $ntn^{bk}$  was placed into the context of the container *myntn*. However, the context annotation of  $ntn^{natural}$  is preserved during the import of the container *setnames1*. Consequently, the effective context annotation of  $N_0$  is  $\lambda = (area = numberTheory, expertise = intermediate, expertise = novice)$ .

So far we proposed an approach to structure and encapsulate notation definitions into different contexts. However, the suggested infrastructure needs to be supported by respective tools that facilitate the reuse, adaptation, extension, and categorization of notation definitions (cf. Section 8).

### 7.3 Consistency

The flexible adaptation of notations can destroy the meaning of documents, in particular, if we use the same notation to denote different mathematical concepts.

To ensure a consistent use of notations when e.g. merging material from a collaborate

content collection, we are applying the recent work on *change management* for semi-structure documents [Mül08b, MW07, loc07].

## 7.4 Flexible Elisions

By flexible elision, we mean the dynamic change of visibility of parts in a rendered documents at viewing time (as opposed to rendering time). We will describe this now in more detail. The following is an adapted version of the flexible elision system we introduced in [KLR07].

There are several situations in which it is desirable to display only some parts of the presentation:

- Brackets that are redundant due to operator precedences can be omitted.
- Arguments that are strictly necessary are omitted to simplify the notation, and the reader is trusted to fill them in from the context.
- Arguments are omitted because they have default values. For example  $\log_{10} x$  is often written as  $\log x$ .
- Arguments whose values can be inferred from the other arguments are usually omitted. For example, matrix multiplication formally takes five arguments, namely the dimensions of the multiplied matrices and the matrices themselves, but usually only the latter two are displayed.
- Type annotations of bound variables are if the types can be inferred from the expression.

Typically, these elisions are confusing for readers who are getting acquainted with a topic, but become more and more helpful as the reader advances. For experienced readers more is elided to focus on relevant material, for beginners representations are more explicit. In the process of writing a mathematical document for traditional (print) media, an author has to decide on the intended audience and design the level of elision (which need not be constant throughout the document though). With electronic media we have new possibilities: We can make elisions flexible. The author still chooses the elision level for the initial presentation, but the reader can adapt it to her level of competence and comfort, making details more or less explicit.

To provide this functionality, we give the rendering items for XML elements, matched objects, and matched lists, two additional arguments: the **elision group** and the **elision level**. For simplicity, we do not add this to the formal syntax of our language. Rather, we only add it in the XML encoding by means of the two attributes **egroup** and **elevel**.

Syntactically, the elision group is any string, and the elision level is any integer including positive and negative infinity (i.e., it has the same type as input and output precedences).

Semantically, if the rendering produces XML, the elision group and level attributes are carried over to the output. Furthermore, we add to the end of Step 3 in the algorithm at the beginning of Sect. 3 the following sentence: If  $p_j \leq p$  and the brackets are rendered as XML, the output is enclosed in brackets as well, but attributes **egroup="bracket"** and **elevel="p - p<sub>j</sub>"** are added. Here, we put  $\infty - \infty = 0$ . If the rendering produces text, all tokens with elidability level above 0 are deleted.

In the case of XML output, the application displaying the document can decide on the visibility of the elidable elements depending on user-provided thresholds or default values. The elision attributes restrict the application's choice of visibility of an element by the following restrictions:

1. If an element has invisible parents, it must be invisible.
2. Otherwise, if an element has no elidability level or an elidability level below or equal 0, it must be visible.
3. Otherwise, elements with positive elidability level may be visible or invisible. If such an element is invisible all elements of the same elidability group and higher or equal elidability level must be invisible, too.

Thus, the application should pick for every elidability group one positive integer or positive infinity as the elision threshold. Then it should make all elements whose elidability levels are above the threshold invisible (including their children elements).

Elision can take various forms in print and digital media. In static media like traditional print on paper or the PostScript format, we have to fix the elision level, and decide at rendering time which elidable tokens will be printed and which will not.

In an output format that is capable of interactively changing its appearance, e.g. dynamic XHTML+MathML (i.e. XHTML with embedded Presentation MATHML formulas, which can be manipulated via JAVASCRIPT in browsers), an application can obtain the elision thresholds dynamically from user interaction.

**Flexible Bracket Elision Demo**

• Powered by [OMDoc](#)  
 • Tested with [Firefox 2.0](#) and [Opera 9.0](#)  
 • Authors: Michael Kohlhase, Christoph Lange, Florian Rabe

$$5 \cdot (x+y)^{n+3} \leq (a \cdot b)! \vee \neg p \wedge \neg (q \leq \pi)$$

$$(5 \cdot (x+y)^{n+3}) \leq (a \cdot b)! \vee (\neg p \wedge \neg (q \leq \pi))$$

$$((5 \cdot (x+y)^{n+3}) \leq ((a \cdot b)!)) \vee ((\neg p) \wedge (\neg (q \leq \pi)))$$

Threshold for showing brackets:  0  200  300  400  500  infinite

Operator	Mixfix declaration	Operator	Mixfix declaration
$x^y$	[199] <sup>[e]</sup> :200	$\neg$	~[600]:600
$!$	[300]:300	$\leq$	[700]≤[700]:700
$\cdot$	[400]{400}:400	$\wedge$	[1000]∧[1000]:1000
$+$	[500]+[500]:500	$\vee$	[1200]∨[1200]:1200

Figure 13: Flexible Elision of Brackets in XHTML

We have implemented a prototypical flexible elision scheme for dynamic XHTML, by generating tokens for all elidable parts, wrapping them in `span` elements and moving the XML attributes `egroupnd` `elevel` to these elements.

## 7.5 Adaptation of OMDoc Documents

Content dictionaries are available in different document formats, e.g. OPENMATH, Content-MATHML, and OMDoc. In contrast to the first two formats, OMDoc content dictionaries are more complex and interrelated: They are represented by mathematical theories, which include *subtheories* and *import* other theories. The *CD* format imposes different requirements onto the input document: For example, when inserting OPENMATH expressions in arbitrary XML documents (any format that supports OPENMATH but OMDoc)

the `base`, `name`, and `cdbase` attributes of the included symbols are used to identify the respective OPENMATH content dictionary and the respective symbol-definition. We propose to collect all notation definitions inside the referenced  $CD$  that apply to these symbols. In contrast, OMDOC input documents need to place mathematical objects inside `theories`, which need to import all required symbols from the OMDOC content dictionaries. The theories inside the OMDOC input documents are the author's private content dictionaries, including his individual notation definition defaults. They can import other theories inside the input document or point to external content dictionaries (either in the author's private collection or any other accessible  $CD$  collection). Consequently, when collecting notation definitions along the *theory graph* [] of OMDOC documents, we do not distinguish default notation definitions from public  $CD$ s or private collection.

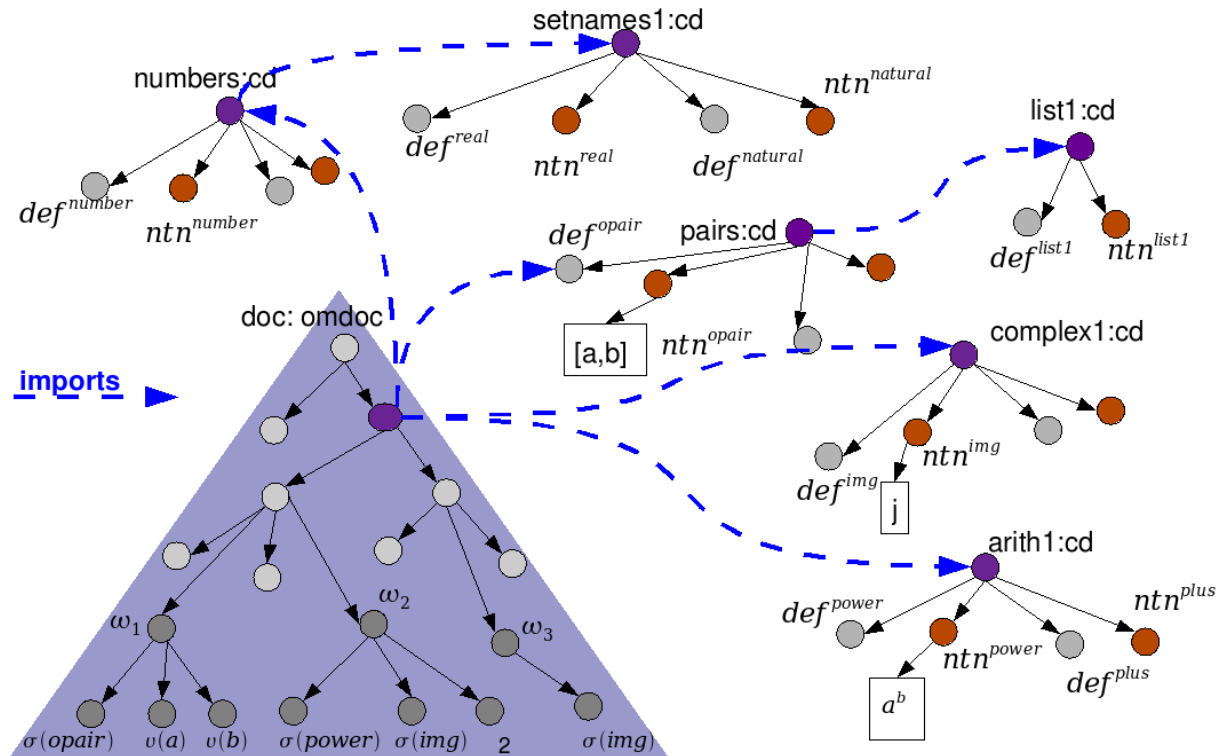


Figure 14: Collecting Notation Definition from OMDOC Content Dictionaries

Figure 6 associates the symbols  $\sigma(opair)$ ,  $\sigma(power)$ , and  $\sigma(img)$  with their respective definitions in the content dictionaries. The dashed arrows in the figure represent the imports of their theory. Please note that we cannot filter the collected notation definitions based on the given symbols since the imports only point to a specific content dictionary and we are not resolving the symbol's `,` and `attributes`. Consequently, the collected notation definitions can apply to any mathematical object  $\omega$  inside the same theory.

We apply the collection algorithm with the input object  $\omega_2$  and  $\mathcal{S}^N = (CD)$ , using the *myntn* file from Section 5.2.3 and receive  $\Pi_{\omega_2}$  in return. For simplicity, we do not display context annotations and precedences and omit the reorganizing of the notation context.

1. We collect all notation definitions from  $CD$  yielding  $\Pi_{\omega_2}$ :  
 $\Pi_{\omega_2} = ( ntn^{numbers}, ntn^{real}, ntn^{natural}, ntn^{opair}, ntn^{list1}, ntn^{img}, ntn^{power}, ntn^{plus} )$

The algorithm transitively browses all imports. Optionally, the search level could be limited to quit after scanning a certain number of import levels. We apply a *depth first search*.

So far, the OMDOC format does not advance the rendering. However, OMDOC supports the specification of theory morphisms during the theory import. These theory morphism could be resolved during the notation collection [RK08]. Moreover, since notation definitions are collected on theory level rather than for each mathematical object, we can reuse the collected notation definitions for all mathematical objects inside the same theory. We expect that the collection will be more efficient and are going to evaluate in the future. Moreover, we leave it to the user as well as concrete rendering system to choose whether the input document is treated as arbitrary XML document or OMDOC document.

## 8 Implementation

To substantiate our approach, we have developed several prototypical implementations of all aspects of the rendering pipeline. However, these implementations do not yet cover the complete presentation pipeline but rather need to be developed further. This sections introduces the current state as well as the further implementation roadmap of the prototypes.

### 8.1 mmlkit — Implementing the Presentation Pipeline

The proposed presentation pipeline has been implemented in the math markup language toolkit (**mmlkit**) [MMK08], which is a toolkit for adapting XML documents, particularly, with respect to mathematical notations. As displayed in Figure 15, **mmlkit** consists of three Java libraries, namely the **collector**, **rndgrab**, **mmlproc**. Moreover it provides a command-line **client** as well as an Interface for PHP applications.

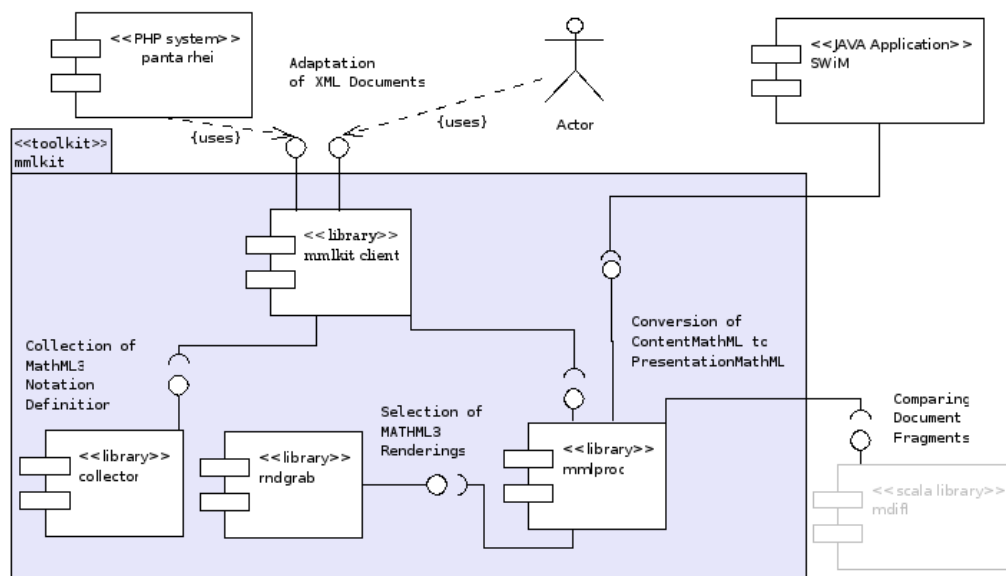


Figure 15: **mmlkit** and its components

`client` [MM08] `mmlkit` provides a commandline `client` as well as an interface for PHP. It provides:

1. The XSLT [Kay06] transformation of XML documents based on explicitly provided XSLT stylesheets.
2. The collection of MATHML3 notation definitions from XML documents (using the `collector` library).
3. The conversion of an XML document's Content-MATHML and OPENMATH expressions into Presentation-MATHML expressions (using the `mmlproc` library).

`collector` The `collector` library implements the collection of notation definition (cf. Section 5.1). It takes as input a mathematical object, arbitrary many external XML files, and an ordered list of source names. It collects all MATHML3 `notation definitions` as given below, and returns the list of collected notation definitions. It provides:

1. The collection of `notation definitions` from an XML document (cf. Section 5.2.3).
2. The collection of `notation definitions` from Content Dictionaries by resolving the symbol's references for a given mathematical object (cf. Section 5.2.4).
3. The collection of `notation definitions` from an XML document in dependence of the position of the mathematical object (cf. Section 5.2.5).
4. The collection of `notation definitions` by resolving the values of the `ec` attributes in the XML document of the mathematical object (cf. Section 5.2.6).

Please note, that the `collector` currently only supports the collection from document that are explicitly provided. Consequently, any implicit references of symbols to content dictionaries as well as any values of `ec` attributes that point to unknown targets are not supported.

`mmlproc` The math markup language processor library undertakes the transformation of Content-MATHML and OPENMATH expressions into Presentation-MATHML expressions. It provides:

1. The reorganization of a given notation context, i.e. the list of collected notation definitions (cf. Section 5).
2. A pattern matcher, which takes as input the reorganized notation context and a mathematical object and returns a list of elements of the form  $(\lambda : \rho)^p$ .
3. A renderer which transforms the mathematical object into Presentation-MATHML. It takes as input the return value of the pattern matcher and calls the `rndgrab` library to select the most appropriate rendering from the list.

`rndgrab` The rendering grabber (`rndgrab`) library implements the intensional selection of renderings (cf. Section 6). It takes as input a mathematical object, a list of elements of the form  $(\lambda : \rho)^p$ , a totally ordered set of source names, and optionally a *global context* or *cascading context file*. It returns the most appropriate rendering. In particular, `rndgrab` provides:

1. The computation of the rendering context based on:
  - (a) A global intensional context (cf. Section 6.3.1).
  - (b) The metadata inside the given XML document (cf. Section 6.3.2).
  - (c) The context declaration via the `ic` attribute (cf. Section 6.3.3).
  - (d) The declaration inside a Cascading Context File (cf. Section 6.3.3).
2. The matching of rendering context and context annotations to identify the most appropriate rendering (cf. Section 6.2).

**mdiff** The *model-based diff* (`mdiff`) [mdp08] is used to compare document fragments, in particular, **notation definitions** based on user-defined equality relations. In particular, when reorganizing notation contexts, the `mmlproc` library currently only applies simple *string comparisons* for prototype identification. However, semantically enriched equality functions facilitate identification of OPENMATH prototypes with their Content-MATHML counterparts, resulting in re-usage of respective renderings.

**Current State** In its current version, `mmlkit` does not provide an intelligent *rendering grabber* and *collector*. The toolkit provides the collection of notation definitions from XML documents, the transformation of content expression into Presentation-MATHML (including substitution as well as parallel rendering), as well as the command line client as describe above. We are currently in the progress of developing the missing components and plan a further release by end of June.

## 8.2 *panta rhei* — An Interactive and Collaborative Viewer

*panta rhei* [pan08c] is an *interactive and collaborative viewer for mathematical content*, which integrates information from arbitrary sources while at the same time facilitating readers to challenge, discuss, and rate the presented information. The system takes the social context of the available information into account in order to make predictions on the relevance of information as well as the users' preferred content, structure, and appearance. *panta rhei* suits as a proof-of-concept prototype to evaluate our theoretic findings in this paper. Consequently, the system integrates `mmlkit` to adapt mathematical documents with respect to mathematical notations.

**Architecture** Figure 16 presents the proposed architecture for the next release of *panta rhei*.

*Panta rhei* uses a *content interface* to combine arbitrary content collections, such as a *problem pool* stored in a MySQL DB, course material from an OMDOC [Koh06] or CNXML [HG07] repository, and available content of the WWW. A *context module* is integrated to adapt the selection, structuring, and layout of the presented content as well as to define different views based on predictions on the user's preferences. The core of *panta rhei* consists of three further modules, namely a *utility module*, a *content module* and a *community* module. The content module includes various containers, such as *slides*, *exams*, *homework*, and *quiz*, which encapsulate fragments of the content collections, such as course snippets and problems. Each container consists of a *model*, which in interplay with the context module selects, structures, and adapts the presented content via the *content interface*, and a *view*, which displays the adapted content. The *community*

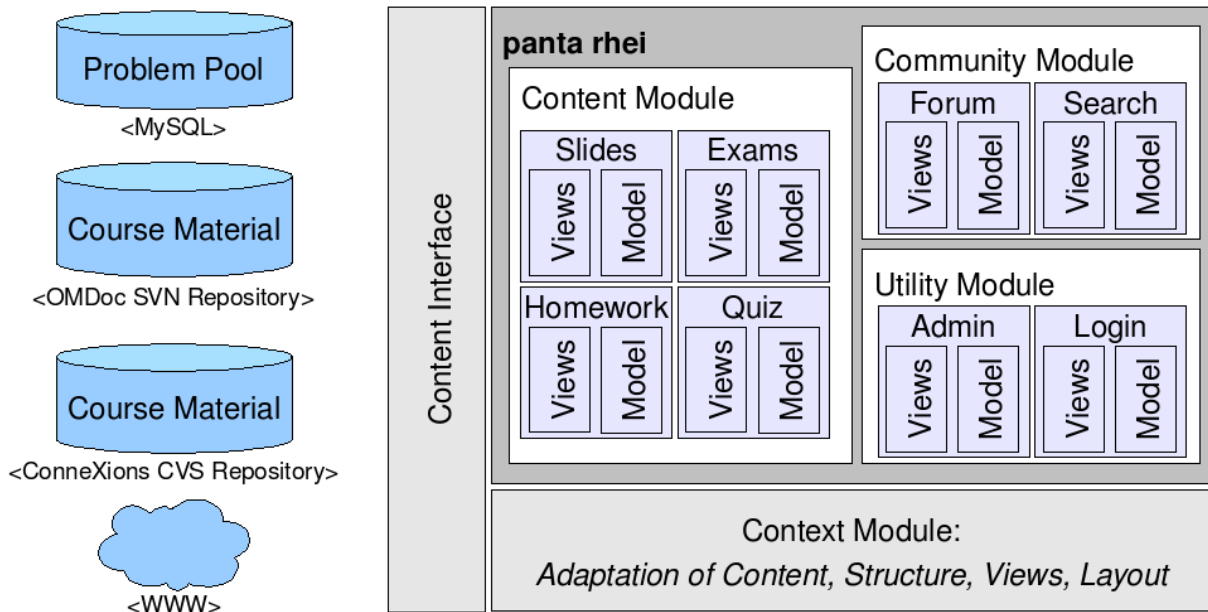


Figure 16: The Architecture of panta rhei

*module* integrates different discussion modules, such as a *forum* or *chat*, which link to the presented content. Moreover, it comprises *community services* such as a *social search*, *social tagging* or *bookmarking*.

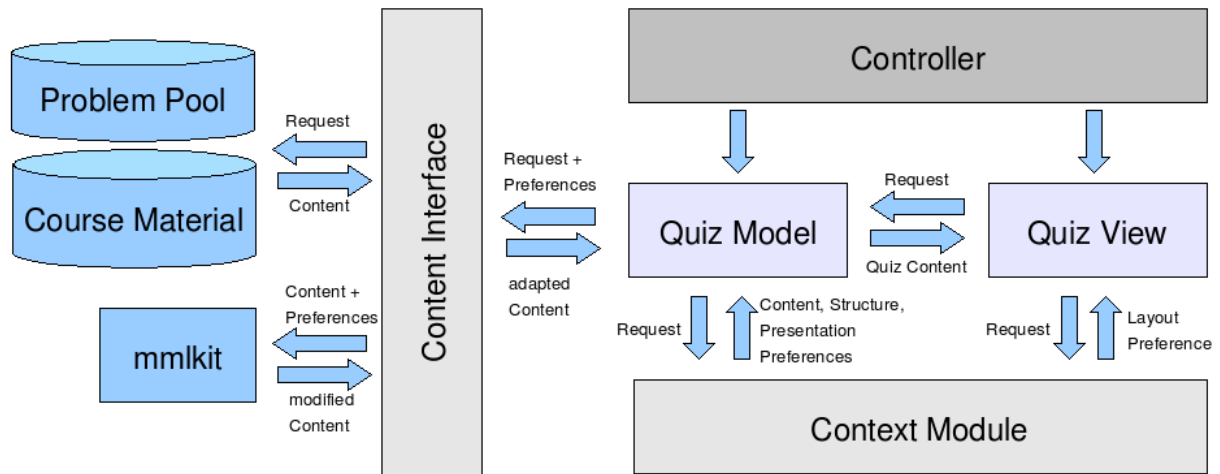


Figure 17: The Presentation Workflow in panta rhei

Figure 17 presents the presentation workflow of panta rhei for adapting a quiz. The *controller* controls the *view* and *model* components. For example, if the user clicks on a link to a specific quiz, the controller prompts the *quiz model* to retrieve the respective content. The *quiz model* prompts the *context module* to provide the user-specific *preference* with respect to *content*, *structure*, and *presentation*. It passes the contextual information to the *content interface*, which provides an abstraction on the actual content collection. The *content interface* first selects the preferred content form a source, e.g. a *problem pool* and structures the retrieved content according to the user's preferences. It then prompts

mmlkit to adapt the content and provides the required preference parameters, such as an *extensional notation document*, *cascading context file*, a *global context*, and an ordered list of *source names* for the collection and selection processes. mmlkit converts the document and returns it to the *content interface*. The quiz document is passed to the *quiz model* and further to the *controller*, which prompts the *quiz view* to display the quiz. Before displaying the content, the *quiz view* prompts the *context module* for the *layout preferences* of the user.

**Case Studies** As of August 2007, *panta rhei* is used for the General Computer Science (GenCS) lecture at Jacobs University [pan08b, Müll]. In this case study, *panta rhei* promotes the learning experience of students by supporting them to challenge and discuss the course material and homework online. The figures below provide screenshots of an assignment of the GenCS course as well as the creation of a new forum post (which is linked to the assignment).

The top screenshot shows the 'panta rhei alpha' web interface. The main content area displays an assignment titled 'Assignment: Combinatorial Circuits' with a rating of 43.33/100. It includes two problems: 'Problem 1 (Length of the inner path in balanced trees) 25 Points' and 'Problem 2 (DNF Circuit with Quine McClusky) 25 Points'. A table of truth values is provided for three functions  $f_1(X)$ ,  $f_2(X)$ , and  $f_3(X)$  based on variables  $X_1$ ,  $X_2$ , and  $X_3$ .

$X_1$	$X_2$	$X_3$	$f_1(X)$	$f_2(X)$	$f_3(X)$
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	1	1	1
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	0	0	0

The bottom screenshot shows the same assignment page, but with a 'Forum Posting Body' dialog box overlaid. The dialog box contains the following text:

Specify title: Basic Circuit Question

Maybe this is a stupid question (but i am not an EE major :-P)  
 Why do we need OR-gates?  
 Can't we just combine the two wires? from my point of view this should not lead to any trouble as the voltage does not increase when connecting circuits in parallel.

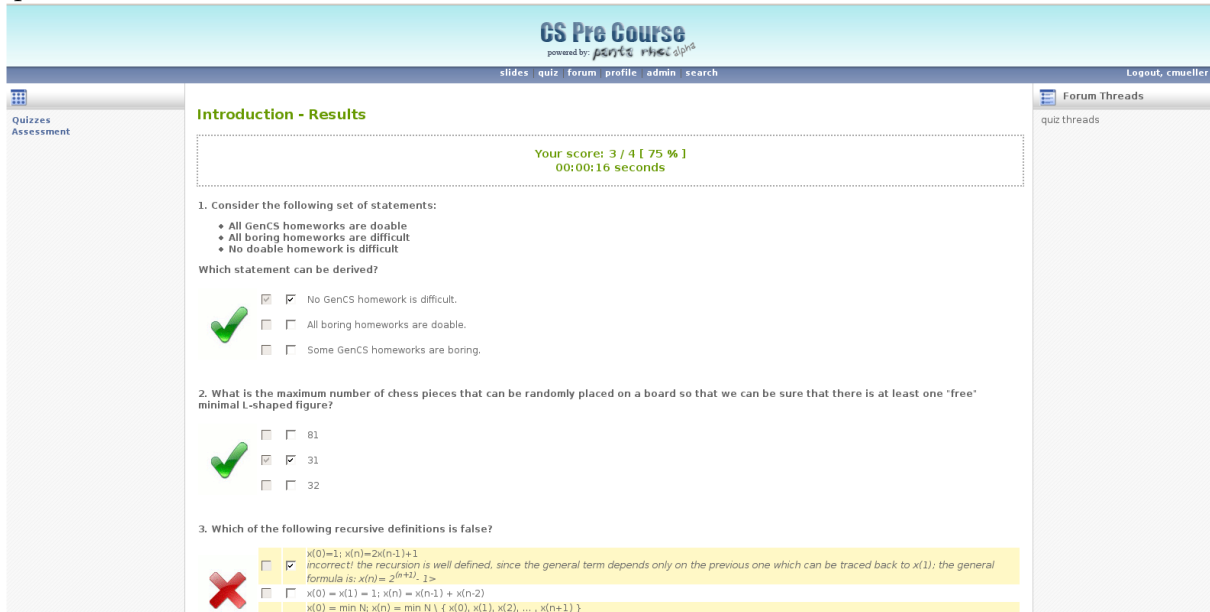
Best,  
 Christine

Information on the annotation:  
 Type:  advice  answer  change  comment  
 example  explanation  question  news  
 Author: cmueller  
 Reference: page:  Post Cancel

Currently, the system enables the browsing of course material, the posting of questions with reference to course material as well as rating and search of forum and course entries.

Based on the users rating and postings, the system explicates the relevance of document fragments or “object of interest”.

We are currently developing a further educational release of *panta rhei*, which is used to set up a CS pre-course at the Jacobs University [pan08a]. It will be based on the new architecture above and integrate *mmlkit*. The figure below presents a screenshot of the quiz module:



Further case studies we are considering are the deployment of *panta rhei* for the further development and discussion of the OMDOC specification as well as the integration with CONNEXIONS [CNX07].

### 8.3 SWiM – A Semantic Wiki to edit Notation Definitions

SWiM [Lan08b] is a semantic wiki for collaboratively building, editing and browsing a mathematical knowledge base. It is based on the general-purpose semantic wiki IkeWiki [Sch06] and enhances it by support for OMDOC, OPENMATH, and MATHML. Our long-term objective is to develop a software that facilitates the creation of a shared, public collection of mathematical knowledge and serves work groups of mathematicians as a tool for collaborative development of new mathematical theories.

**Semantic Wikis** A wiki [LC01] is a web server application that allows users to browse, create, and edit hyperlinked pages in a web browser. A *semantic* wiki uses semantic web technologies for knowledge representation [VSD06]. Most commonly that means that pages and links are typed with terms from ontologies, but other approaches to adding semantics to wikis exist as well [BGE+08]. From these type annotations, an RDF [LS99] graph is extracted where every nodes corresponds to one wiki page. In combination with the background knowledge in the ontologies additional knowledge can be inferred from the RDF graph of explicit knowledge. This additional knowledge can then be supplied to external services, or it can be used inside the wiki to improve navigation, search, or authoring.

**Knowledge Representation** One page in SWiM represents one mathematical theory or statement (such as a symbol declaration, definition, axiom, theorem, proof, or example)

Figure 18: A content dictionary in SWiM

in the case of OMDOC, and a content dictionary, a symbol declaration, a formal or an informal property of a symbol, or an example in the case of OPENMATH – or it represents one notation definition [Lan08b, Lan08a]. Small pages like this are advisable in the collaborative setting of a wiki, as they reduce the potential of editing conflicts. With small pages, page-level annotation as SWiM uses it is at its most impressive in terms of easy-to-use authoring, navigation, and search from a user’s point of view, and easy maintenance within the system.

All symbols to be used in formulae on a SWiM site, as well as their notation definitions, are assumed to be defined in the wiki itself. SWiM allows for importing and exporting OMDOC and OPENMATH documents from and to the local file system or a Subversion repository. Usually such a file contains a whole theory or content dictionary. Therefore, SWiM splits imported documents into subparts as listed above, every part being stored on its own wiki page. The containment relations are preserved as XLinks [DMOT01], which are traversed on export again. The XLinks are also resolved when a document is rendered, so that a whole content dictionary or theory can conveniently be viewed at once.

**A Document Ontology** An ontology had to be developed to allow for making RDF statements about resources in content dictionaries or theories, and an automated extraction of RDF statements from the mathematical markup had to be developed. The OPENMATH document ontology models classes and properties for all structural entities found in OPENMATH’s content dictionary groups, content dictionaries, type signatures, and notation definitions in OWL-DL [MvH04] in accordance with the specification of OPENMATH [BCC<sup>+</sup>04, chap. 4]. For the theory and statement level of OMDOC this was done similarly. Properties from common ontologies like Dublin Core were reused as appropriate. The inner structures of formulae (also known as “OPENMATH objects”) were not modeled [Lan08a]. There is, however, a property that states that a formal definition or an example uses a symbol – which is contained in some OPENMATH object inside the formal definition or example in the OPENMATH case – and points to the definition of that symbol in some content dictionary. This is not only useful for determining dependencies among content dictionaries (What other content dictionaries do I need to load in order to get a self-contained collection?), but also for rendering formulae according to notation definitions (see sect. ??). This has first been investigated for OPENMATH documents in SWiM but will later also be supported for OMDOC documents.

Whenever a wiki page is stored in SWiM, i. e. whenever it is saved in the editor or imported, an RDF representation in terms of the document ontology is extracted from the markup. Consider the following content dictionary:

```
<CD xml:id="sample" >
  <CDName>sample</CDName>
  <CDDate>2008-03-05</CDDate>
  <CDStatus>private</CDStatus>
  <Description>A sample CD</Description>
  <CDDefinition xlink:type="simple"
    xlink:href="url/of/cddefinition" xlink:show="embed" /></CD>
```

From this, the following RDF would be extracted (in Turtle syntax [Bec07]), where `omo` is the prefix of the OPENMATH document ontology namespace:

```
<#sample>
```

Figure 19: Editing OPENMATH in SWiM

```
rdf:type omo:ContentDictionary ;
dc:identifier "sample" ;
dc:date 2008-03-05 ;
omo:status "private" ;
dc:description "A sample CD" ;
omo:containsSymbolDefinition <url/of/cddefinition> .
```

IkeWiki currently utilizes the RDF graph in order to generate a list of incoming and outgoing links for the current page, grouped by property (shown on the right side in fig. 18), to feed a graphical RDF browser, to preselect properties of pages and links an author would probably want to annotate, and it supports embedding arbitrary inline queries in the SPARQL query language [PS08] into pages.

**Authoring** SWiM supports editing semantic markup, including notation definitions, in a semi-WYSIWYG way. Plain text can be edited and styled visually (but styles are lost on export), for OPENMATH objects there is a simple linear ASCII syntax, and other XML structures are made accessible as special HTML tables (see fig. 19).

**Use Case** In the course of opening up new mathematical areas, definitions of new symbols and their axiomatization are not fixed initially but subject to continuous evolution and refactoring. Similarly the *notation* of a symbol can evolve in the course of time, or additional notation definitions could be provided. The current version 0.2 of SWiM supports one notation definition per symbol, or takes the first one it can find if there are multiple ones, but it supports changes to this notation definition. Changes are instantly reflected in the places where the respective notation definition is used to present a symbol. That means that whenever the notation of a symbol  $\sigma$  has changed, all the presentation markup generated from formulae containing  $\sigma$  has to be invalidated and re-rendered upon the next request. In a single-author environment this frees the author from recompiling all the affected presentation markup and gives instant visual feedback about whether the new notation works, and in a collaborative environment it relieves *other* authors from worrying whether they are looking at an up-to-date presentation of a document.

**Defining Notations and (Re-)Rendering Formulae** SWiM employs the mmlproc rendering library of `mmlkit` (cf. section [?]) for rendering OPENMATH objects to Presentation-MATHML, which can be viewed in recent versions of the Firefox or Opera browsers. Whenever a wiki page containing notation definitions is saved or imported, the notation definitions are put into a cache read by mmlproc. To symbols without a notation definition, mmlproc applies a default rendering (cf. section [?]).

If a notation definition has been added, deleted, or changed, the affected documents have to be re-rendered. In order to do this properly, SWiM has to

1. identify changes to notation definitions
2. identify documents affected by a change

(1) is done by computing an XML diff between the cached and the newly inserted version of a notation definition. (2) is done by querying the RDF graph for all formal properties and examples using the symbol rendered by the respective notation definition, as shown

in fig. 20 and 21. Not only the wiki pages holding these formal properties and examples have to be re-rendered, but also those pages (symbol definitions and content dictionaries) that directly or indirectly *include* these fragments.

```

1 SELECT DISTINCT ?page WHERE {
2   <changed-ntn-def> omo:rendersSymbol ?sym .
3   { ?page omo:usesSymbol ?sym } UNION
4   { ?exOrFMP omo:usesSymbol ?sym .
5     { ?page omo:contains ?exOrFMP } UNION
6     { ?page omo:contains ?symDef .
7       ?symDef omo:contains ?exOrFMP } } }

```

Figure 20: SPARQL query determining the effect of changing a notation definition; see fig. 21 for a graphical representation.

**Future Work** So far, SWiM assumes that there is at most one notation definition per symbol. The `mmlproc` renderer supports callbacks to a custom rendering grabber that selects the most appropriate out of a set of multiple possible renderings for a symbol (cf. section 8.1). A default implementation of a rendering grabber that considers the intensional context of a formula (cf. section 6.3.3) is provided with the `rndgrab` library and could easily be enabled in SWiM. In future, it is planned to provide a user interface inside SWiM that lets the user select his preferred rendering for every symbol and then implement a custom rendering grabber that takes these user preferences into account.

A visual editor for formul will be provided as well. Available editors will be evaluated w. r. t. their extensibility by new symbols and notation definitions. Ideally, the tool palette of a visual formula editor would be supplied dynamically with all known instances of the *SymbolDefinition* class of the OPENMATH document ontology, i. e. all symbols known to the system.

## 9 Conclusion & Outlook

We introduced a representational infrastructure for notations in living mathematical documents. We provided a flexible declarative specification language for notation definitions and gave a rendering algorithm. We described how authors and users can extensionally extend the set of available notation definitions on granular document levels, and how they

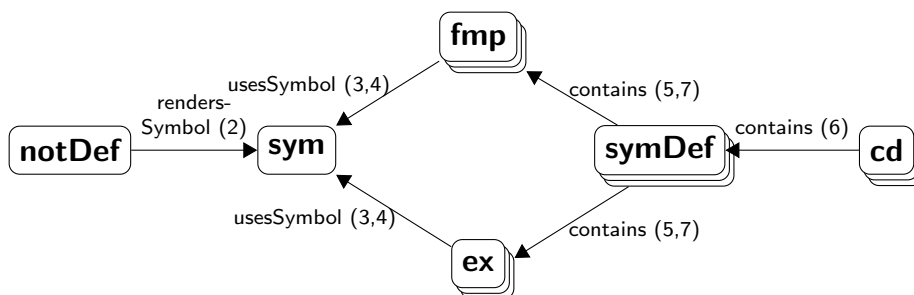


Figure 21: Finding pages (depicted as stacks of nodes) affected by changes to a notation definition. Numbers refer to lines of listing 20. Note that both *sym* and the *symDefs* are instances of the class *SymbolDefinition*.

can guide the notation selection via intensional context declarations. Moreover, we discussed different approaches for collecting notation definitions and contextual information. To substantiate our approach, we have developed several prototypical implementations of all aspects of the rendering pipeline.

We will invest further work into our implementations as well as the evaluation of our approach.

## Acknowledgment

Our special thanks go to Dimitar Misev for the further development to `mmlkit` as well as Andrei Aiordachioaie and Alen Stojanov for their contribution to `panta rhei`. Moreover, we want to thank Alberto Gonzáles Palomo and Paul Libbrecht for the discussions on their work. We would also like to thank the other members of the KWARC group at Jacobs University for the fruitful discussions and continuous feedback on our work.

## References

- [ABC<sup>+</sup>03] Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 2.0 (second edition). W3C recommendation, World Wide Web Consortium, 2003.
- [Ado08] Adobe acrobat reader. web page at <http://www.adobe.com/>, seen April 2008.
- [BCC<sup>+</sup>04] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004.
- [Bec07] Dave Beckett. Turtle – terse RDF triple language, November 2007.
- [BGE<sup>+</sup>08] Michel Buffa, Fabien Gandon, Guillaume Ereteo, Peter Sander, and Catherine Faron. SweetWiki: A semantic wiki. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2008.
- [Caj93] Florian Cajori. *A History of Mathematical Notations*. Courier Dover Publications, 1993. Originally published in 1929.
- [Cas99] Cascading Style Sheets. <http://www.w3.org/Style/CSS/>, 1999.
- [CNX07] CONNEXIONS. Project homepage at <http://www.cnx.org>, seen Dec. 2007.
- [DMOT01] Steve DeRose, Eve Maler, David Orchard, and Ben Trafford. XML linking language (XLink version 1.0). W3C recommendation, W3C, 2001.
- [Dub08] Dublin Core, seen March 2008. web page at <http://www.dublincore.org>.
- [Goo09] Google reader. web page at <https://www.google.com/reader>, seen March 2009.
- [GP06] Alberto González Palomo. Sentido: an authoring environment for OMDoc. In *OMDOC – An open markup format for mathematical documents [Version 1.2]* [Koh06], chapter 26.3.
- [HBK03] Geneva Henry, Richard G. Baraniuk, and Christopher Kelty. The connexions project: Promoting open sharing of knowledge for education. In *Syllabus, Technology for Higher Education*. 2003.
- [HG07] Brent Hendricks and Adan Galvan. The Connexions Markup Language (CNXML). <http://cnx.org/aboutus/technology/cnxml/>, 2007. Seen June 2007.
- [Kay06] Michael Kay. XSL Transformations (XSLT) Version 2.0. W3C Candidate Recommendation, World Wide Web Consortium (W3C), June 2006.
- [KLR07] Michael Kohlhase, Christoph Lange, and Florian Rabe. Presenting mathematical content with flexible elisions. In Olga Caprotti, Michael Kohlhase, and Paul Libbrecht, editors, *OPENMATH/ JEM Workshop 2007*, 2007.

- [KMM07] Michael Kohlhase, Christine Müller, and Normen Müller. Documents with flexible notation contexts as interfaces to mathematical knowledge. In Paul Libbrecht, editor, *Mathematical User Interfaces Workshop 2007*, 2007.
- [Koh06] Michael Kohlhase. OMDOC – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, 2006.
- [Lan08a] Christoph Lange. Mathematical Semantic Markup in a Wiki: The Roles of Symbols and Notations. In Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *Proceedings of the 3<sup>rd</sup> Workshop on Semantic Wikis, European Semantic Web Conference 2008*, volume 360 of *CEUR Workshop Proceedings*, Costa Adeje, Tenerife, Spain, June 2008.
- [Lan08b] Christoph Lange. SWiM – a semantic wiki for mathematical knowledge management. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, pages 832–837. Springer, 2008.
- [Lan08c] Christoph Lange. SWiM: A semantic wiki for mathematical knowledge management. web page at <http://kwarc.info/projects/swim/>, seen October 2008.
- [LC01] Bo Leuf and Ward Cunningham. *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley Professional, 2001.
- [loc07] *locutor*: An ontology-driven management of change, seen June 2007. system homepage at <https://locutor.kwarc.info>.
- [LS99] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C recommendation, World Wide Web Consortium (W3C), 1999.
- [MAF<sup>+</sup>01] E. Melis, E. Andres, A. Franke, G. Goguadse, P. Libbrecht, M. Pollet, and C. Ullrich. ACTIVE MATH System Description. In Johanna D. Moore, Carol Luckhard Redfield, and W. Lewis Johnson, editors, *Artificial Intelligence in Education*, volume 68 of *Frontiers in Artificial Intelligence and Applications*, pages 580–582. IOS Press, 2001.
- [Mat08] Wolfram mathematica player: The free player with a mathematica engine. web page at <http://www.wolfram.com/products/player/>, seen April 2008.
- [mdp08] *mdpm*: A Collection of Model-based DIFF, PATCH, MERGE Algorithms, seen March 2008. system homepage at <http://www.kwarc.info/projects/mdpm/>.
- [MLUM05] Shahid Manzoor, Paul Libbrecht, Carsten Ullrich, and Erica Melis. Authoring Presentation for OPENMATH. In Michael Kohlhase, editor, *Mathematical Knowledge Management, MKM'05*, number 3863 in LNAI, pages 33–48. Springer Verlag, 2005.
- [MM08] Normen Müller and Christine Müller. client - The mmlkit command-line client and mmlkit interface for JAVA and PHP Applications. maintained until Jan 08, 2008.

- [MMK08] Normen Müller, Christine Müller, and Michael Kohlhase. The math markup language toolkit (mmlkit). seen June 2008.
- [Mül] Christine Müller. Case study with panta rhei. <http://kwarc.info/blog/2007/12/15/panta-rhei-a-review-of-a-successful-case-study-at-kwarc/>. Seen March 2008.
- [Mül08a] Christine Müller. A Survey on Mathematical Notations. KWARC report, Jacobs University Bremen, 2008.
- [Mül08b] Normen Müller. MDPM – A Collection of Model-based DIFF, PATCH, MERGE Algorithms. Technical report, Jacobs University Bremen, 2008. in progress.
- [MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL web ontology language overview. W3C recommendation, W3C, February 2004.
- [MW07] Normen Müller and Marc Wagner. Towards Improving Interactive Mathematical Authoring by Ontology-driven Management of Change. In Alexander Hinneburg, editor, *Wissens- und Erfahrungsmanagement LWA (Lernen, Wissensentdeckung und Adaptivität) conference proceedings*, pages 289–295, 2007.
- [NW01a] Bill Naylor and Stephen M. Watt. Meta-Stylesheets for the Conversion of Mathematical Documents into Multiple Forms. In *Proceedings of the International Workshop on Mathematical Knowledge Management* [NW01b].
- [NW01b] Bill Naylor and Stephen M. Watt. Meta-Stylesheets for the Conversion of Mathematical Documents into Multiple Forms. In *Proceedings of the International Workshop on Mathematical Knowledge Management*, 2001.
- [OMB08] OMBase Project. <http://kwarc.info/projects/ombase/>, seen Dec. 2008.
- [pan08a] The CS precourse project. <http://cs-precourse.kwarc.info/>, 2008.
- [pan08b] The GenCS project. <http://panta-rhei.kwarc.info/>, 2008.
- [pan08c] The panta rhei Project. <http://trac.kwarc.info/panta-rhei>, 2008.
- [pla07] PLATO: Interactive Mathematical Authoring, seen August 2007. system homepage at <http://www.ags.uni-sb.de/plato/bin/view.pl>.
- [Pla09] Planet Math – math for the people, by the people. <http://www.planetmath.org>, seen March2009.
- [POS88] IEEE POSIX, 1988. ISO/IEC 9945.
- [PS08] Eric Prud’hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C Recommendation, World Wide Web Consortium, January 2008.
- [RK08] Florian Rabe and Michael Kohlhase. A web-scalable module system for mathematical theories. Manuscript, to be submitted to the Journal of Symbolic Computation, 2008.
- [Sch06] Sebastian Schaffert. IkeWiki: A semantic wiki for collaborative knowledge management. In *1st International Workshop on Semantic Technologies in Collaborative Applications STICA 06, Manchester, UK*, June 2006.

- [SW06a] Elena Smirnova and Stephen M. Watt. Generating TeX from Mathematical Content with Respect to Notational Settings. In *Proceedings International Conference on Digital Typography and Electronic Publishing: Localization and Internationalization (TUG 2006)*, pages 96–105, Marrakech, Morocco, 2006.
- [SW06b] Elena Smirnova and Stephen M. Watt. Notation Selection in Mathematical Computing Environments. In *Proceedings Transgressive Computing 2006: A conference in honor of Jean Della Dora (TC 2006)*, pages 339–355, Granada, Spain, 2006.
- [VSD06] Max Völkel, Sebastian Schaffert, and Stefan Decker, editors. *1st Workshop on Semantic Wikis*, volume 206 of *CEUR Workshop Proceedings*, Budva, Montenegro, June 2006.
- [W3C02] W3C. Syntax of CSS rules in HTML’s ”style” attribute. <http://www.w3.org/TR/css-style-attr>, 2002. Seen March 2008.
- [W3C07] W3C. Mathematical Markup Language (MathML) Version 3.0 (Third Edition). <http://www.w3.org/TR/MathML3/>, 2007. Seen November 2007.
- [Wik07] Wikipedia, the free encyclopedia. <http://www.wikipedia.org>, 2001–2007.
- [WM07] Marc Wagner and Christine Müller. Towards Community of Practice Support for Interactive Mathematical Authoring. In Christine Müller, editor, *Proceedings of the 1st SCooP Workshop*, 2007.
- [Wol00] Stephen Wolfram. Mathematical notation: Past and future. In *MathML and Math on the Web: MathML International Conference*, Urbana Champaign, USA, October 2000. <http://www.stephenwolfram.com/publications/talks/mathml/>.
- [xma08] Formelsammlungen: Mathematik, physik, technik, und finanzmathematik. web page at <http://www.seeit.de/xedu/>, seen April 2008.
- [Zho08] Vyacheslav Zholudev. Towards distributed mathematical knowledge management. <http://kwarc.info/pubs/proposal.pdf>, 2008.