Towards Improving Interactive Mathematical Authoring by Ontology-driven Management of Change

Normen Müller[1], **Marc Wagner**[2]

[1]Jacobs University Bremen
D-28759, Bremen, Germany
n.mueller@jacobs-university.de

[2]Universität des Saarlandes
D-66123, Saarbrücken, Germany
wagner@ags.uni-sb.de

Abstract

The interactive use of mathematical assistance systems requires an intensive training in their input and command language. With the integration into scientific WYSIWYG text-editors the author can directly use the natural language and formula notation she is used to. In the new *document-centric* paradigm changes to the document are transformed by a mediator into commands for the mathematical assistance system. This paper describes how ontology-driven management of change can improve the process of interactive mathematical authoring.

1 Introduction

Mathematical proof assistance systems have not yet achieved recognition and relevance in mathematical practice. Significant progress is still required, in particular with respect to the user-friendliness of these systems. Rather than developing a new user interface for the mathematical assistance system ΩMEGA the PLATΩ system [plato, 2007] presents a generic way of integrating proof assistance systems into scientific text-editors by using a flexible and parametric semantic annotation language. PLATΩ allows for the incremental development of mathematical documents in professional type-setting quality by propagation of changes and context sensitive service menu interaction.

The aim of the PLATΩ system is to support the complete authoring process of a mathematical document - from creation through formalization to publication - in a collaborative environment. This paper investigates the added values for the authoring process when integrating the `locutor` system [locutor, 2007; Müller, 2006; 2007] into PLATΩ. Using ontology-driven management of change and hence maintaining semantic dependencies the concrete research questions are: How can we preview the effects of a modification for the author? How can authors be informed about dependency conflicts during collaborative editing? How can we provide suggestions for conflict resolution?

2 The Mediator: PLATΩ

The development of the proof assistance system ΩMEGA is one of the major attempts to build an all-encompassing assistance tool for the working mathematician or for the formal work of a software engineer. It is a representative of systems in the paradigm of *proof planning* and combines interactive and automated proof construction for domains with rich and well-structured mathematical knowledge. The ΩMEGA-system is currently under redevelopment where, among others, it has been augmented by the development graph manager MAYA, and the underlying natural deduction calculus has been replaced with the CORE-calculus [Autexier, 2005].

The MAYA system [Autexier and Hutter, 2005] supports an evolutionary formal development by allowing users to specify and verify developments in a structured manner, it incorporates a uniform mechanism for verification in-the-large to exploit the structure of the specification, and it maintains the verification work already done when changing the specification. Proof assistance systems like ΩMEGA rely on mathematical knowledge formalized in structured theories of definitions, axioms and theorems. The MAYA system is the central component in the new ΩMEGA system that takes care about the management of change of these theories via its OMDOC-interface [Kohlhase, 2006].

The CORE-calculus supports proof development directly at the *assertion level* [Huang, 1996], where proof steps are justified in terms of applications of definitions, axioms, theorems or hypotheses (collectively called *assertions*). It provides the logical basis for the so-called TASK LAYER [Dietrich, 2006], that is the central component for computer-based proof construction in ΩMEGA. The proof construction steps are: (1) the introduction of a proof sketch [Wiedijk, 2004], (2) deep structural rules for weakening and decomposition of subformulas, (3) the application of a lemma that can be postulated on the fly, (4) the substitution of meta-variables, and (5) the application of an inference. Inferences are the basic reasoning steps of the TASK LAYER, and comprise assertion applications, proof planning methods or calls to external theorem provers or computer algebra systems (see [Dietrich, 2006; Autexier and Dietrich, 2006] for more details about the TASK LAYER).

A formal proof requires to break down abstract proof steps to the CORE calculus level by replacing each abstract step by a sequence of calculus steps. This has usually the effect that a formal proof consists of many more steps than a corresponding informal proof of the same conjecture. Consequently, if we manually construct a formal proof many interaction steps are typically necessary. Formal proof sketches [Wiedijk, 2004] in contrast allow the user to perform high-level reasoning steps without having to justify them immediately. The underlying idea is that the user writes down only the interesting parts of the proof and that the gaps between these steps are filled in later, ideally fully automatically (see also [Siekmann *et al.*, 2002]). Proof sketches are thus a highly adequate means to realize the tight integration of a proof assistance system and a scientific text-editor.

Figure 1: Architecture of the integration of the text-editor TEXMACS and the ΩMEGA system via the mediator PLATΩ

The mediator PLATΩ [Wagner *et al.*, 2006] has been designed as a support system to realize the tight integration of a proof assistance system and a text-editor (see Figure 1). PLATΩ is connected with the text-editor by an informal representation language which flexibly supports the usual textual structure of mathematical documents. This semantic annotation language, called *proof language* (PL), allows for underspecification as well as alternative (sub)proof attempts. In order to generate the formal counterpart of a PL representation, PLATΩ separates theory knowledge like definitions, axioms and theorems from proofs. The theories are formalized in the *development graph language* (DL), which is close to the OMDOC theory language supported by the MAYA system, whereas the proofs are transformed into the *tasklayer language* (TL) which are descriptions of TASK LAYER proofs. Hence, PLATΩ is connected with the proof assistance system ΩMEGA by a formal representation close to its internal data structure.

Besides the transformation of complete documents, it is essential to be able to propagate arbitrary changes from an informal PL representation to the formal DL/TL one and the way back. If we always perform a global transformation, we would on the one hand rewrite the whole document in the text-editor which means to lose large parts of the natural language text written by the user. On the other hand we would reset the data structure of the proof assistance system to the abstract level of proof sketches. For example, any already developed expansion towards calculus level or any computation result from external systems would be lost. Therefore, one of the most important aspects of PLATΩ's architecture is the propagation of changes.

The formal representation finally allows the underlying proof assistance system to support the user in various ways. PLATΩ provides the possibility to interact through context-sensitive service menus. If the user selects an object in the document, PLATΩ requests service actions from the proof assistance system regarding the formal counterparts of the selected object. Hence, the mediator needs to maintain the mapping between objects in the informal language PL and the formal languages DL and TL.

In particular, the proof assistance system supports the user by suggesting possible inference applications for a particular proof situation. Since the computation of all possible inferences may take a long time, a multi-level menu with the possibility of lazy evaluation is provided. PLATΩ supports the execution of nested actions inside a service menu which may result in a patch description for this menu.

Furthermore, the PLATΩ system provides an efficient management of user-defined notation [Autexier *et al.*, 2007] that allows the author to define her own notation inside a document in a natural way, and use it to parse the formulas written by the author as well as to render the formulas generated by the proof assistance system.

3 The Document Engineer: `locutor`

The `locutor` system aims at the development of a methodology, techniques, and tools to support a management of change (MOC) for informal but internally structured documents, i.e. to support the evolution, revision and adaptation of collections of technical documents. The system adapts and extends change management techniques from formal methods (cf. development graphs) to the informal setting. Instead of a formal semantics we assume that these documents adopt syntactical and semantic structuring mechanisms formalized in a *document ontology* (cf. Figure 2). This is an ontology that formalizes the document structure rather than the document contents and is also used to classify the type of documents. In particular we assume that it provides a notion of document fragment equivalence (cf. section 5.1). This formalization provides a notion of consistency and invariants that allows one to propagate effects of local changes to entire documents. Conversely, the ontology will provide means to localize effects of changes by introducing a notion for semantic dependencies between document parts.

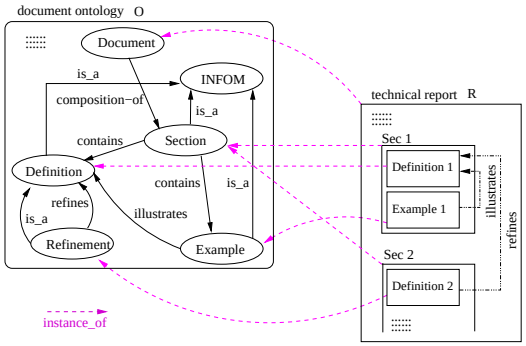

Figure 2: A document ontology $\mathcal{O}$

We regard *documents* as *self-contained structured compositions of information units*. One can pragmatically think of information units as "*tangible/visual text fragments potentially adequate for reuse*" constituting the content of documents. To distinguish the term "information unit" between common speech and the ontological concept, we call the ontological concept INFOM.

Following the OMDOC approach[1] we separate documents into two layers both under version control: A *narrative* and a *content* layer both of which consist of INFOMS and are composed via relations. The presentational order of information units in documents is represented on the narrative layer whereas the information units themselves and the ontological relations between them are placed in the content layer[2]. The connection between the narrative and the content layer is represented via *narrative relations* (analogous to symbolic links in UNIX). The information units and the ontological relations build up the "content commons" [CNX, 2007]. We use the term NARCON for the graph representations of document collections consisting of a narrative layer and a content layer.

[1] The OMDOC group does not claim to have invented this concept, it is part of the XML folklore and can already be found e.g. in [Verbert and Duval, 2004]. But the OMDOC format probably implements this idea in the cleanest way.

[2] Both the structural and the ontological relations are retrieved by the respective document ontology.

Following the initial work in the MMiSS [MMiSS, 2007] project, we also model the concept of *variants*. This expands the application area not only "in-the-breadth" but also "in-the-depth". Thus, by extending the well-known concept of *versions* and *revisions* by the concept of variants, the life-cycle of documents will no longer be only along a horizontal time line but also along a vertical line of variants. On the document level we call the concept of versions, revisions, and variants *document states*.

The computation of structural differences between two document states is based on the insights of XMLDIFF tools and the initial work of [Eberhardt and Kohlhase, 2004; Kohlhase and Anghelache, 2003]. According to this we extended the `diff`–algorithms and unification-based techniques, proposed there, to operate on NARCONs resulting in a $\mathcal{M}$`diff`-algorithm, i.e. a model based `diff`–algorithm comprising an equality theory on NARCONs (w.r.t. the respective document ontology). Therewith `locutor` is able to identify syntactically different INFOMs to be semantically equal and thus to minimize the number of INFOMs affected when changing INFOMs *(Equality Theory)* and to frame the syntactical representation of INFOMs and thus to help to locate changes of INFOMs relative to the internal structure *(Syntactical Structure)*.

In the first step, to compute the *long-range effect of changes* the `locutor` system prompts the author for a *classification* of the computed structural differences. Therefore the system provides a MOC ontology comprising a *taxonomy of change relations*. The idea is to capture the essence of semantically equal INFOMs in the specification of equivalence relations R on INFOMs. Then, dependencies between INFOMs are always relative to equivalence classes, i.e. changing an INFOM I within an equivalence class will not affect INFOMs that depend on I only with respect to R. The connection between a document ontology and MOC ontology is modeled in a so-called *system ontology*. The central intuition behind this approach is that *strong change management* (SCM) techniques can be based on information that can be expressed in system ontologies. We claim that the `locutor` system only needs the system ontology part of a fully formal domain semantics. Thus system ontologies will be the central means for extending the SCM methods to the structured, two-layered and two-dimensional document setting.

In the second step, to propagate changes, the `locutor` system performs a *reasoning on classified structural differences* utilizing inference rules consolidated in a *change relation calculus* based on the respective system ontology.

4 Integrating `locutor` into PLATΩ

We plan to consolidate the two systems as depicted in Figure 3. Therewith we want to gain besides collaborative authoring with version management the following benefits:

4.1 Benefits for the User

Besides version management and collaborative authoring the integration of `locutor` into the PLATΩ system should decrease conflicts and thus time-consuming recomputations.

`locutor` should be able to preview the effects of a modification for the author and to improve consistency on the document level either by adaptation on demand or by automatic adaptation. Figure 4 shows a scenario where the author e.g. modified a variable name inside a formula. Let

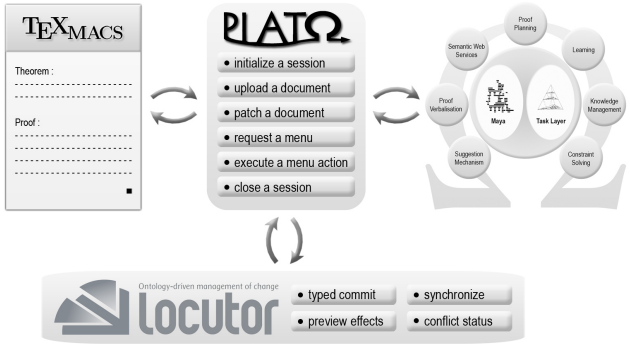

Figure 3: Integrated Architecture of `locutor` and PLATΩ

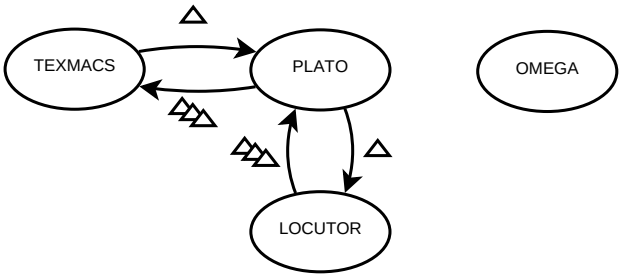

Figure 4: Preview of change effects by `locutor`

$\forall A, B.\ A = B \Leftrightarrow A \subset B \wedge B \subset A$ be the old formula and let $\forall \mathbf{C}, B.\ A = B \Leftrightarrow A \subset B \wedge B \subset A$ be the new one. This single modification is sent to `locutor` which in turn adapts all dependent variable positions in the same formula in order to preview the effects for the author. Thus the formula $\forall \mathbf{C}, B.\ \mathbf{C} = B \Leftrightarrow \mathbf{C} \subset B \wedge B \subset \mathbf{C}$ is shown as preview.

4.2 Benefits for the Proof Assistance System

Regarding PLATΩ's interface to the mathematical assistance system, `locutor` should act like a firewall blocking erroneous. Otherwise the mathematical assistance system would try to verify the erroroneous input and thus wasting the time of the author who waits for feedback. That is for example the scenario in Figure 5, where the author performs a set of modifications which produce conflicts in the document that cannot be resolved automatically. Then the author will be notified of the conflicts and may resolve or force them. As example we consider again the formula $\forall A, B.\ A = B \Leftrightarrow A \subset B \wedge B \subset A$. When the author modifies this formula to $\forall \mathbf{C}, B.\ \mathbf{D} = B \Leftrightarrow A \subset B \wedge B \subset A$ the automatic adaptation fails in the former occurrence of the variable A that has been renamed to D. Therefore the author will be asked whether or not this conflict is intended.

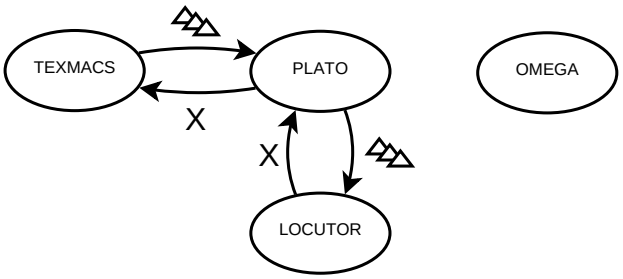

Figure 5: Notification of conflicts by `locutor`

Moreover, by identifying dependent modifications the `locutor` system should be able to return a combined

meta change information. Considering the example with the old formula $\forall A, B.\ A = B \Leftrightarrow A \subset B \wedge B \subset A$ and the new formula $\forall \mathbf{C}, B.\ \mathbf{C} = B \Leftrightarrow \mathbf{C} \subset B \wedge B \subset \mathbf{C}$, `locutor` is able to identify the α-conversion of the variable A to C. Instead of propagating the renaming of each single occurrence of the variable A in that formula to the mathematical assistance system, this set of modifications is sent to `locutor` as shown in Figure 6 who combines them to one meta modification, the *replacement* $\{A \mapsto C\}$, which is finally applied in the mathematical assistance system directly on the whole formula. The mathematical assistance system then takes care about the more complex dependencies between the variable names in formulas occurring in different proofsteps.

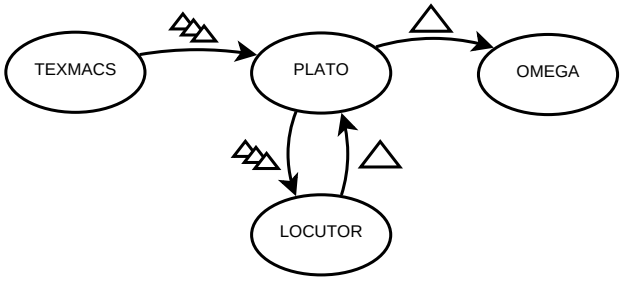

Figure 6: Combining a set of dependent changes to a meta change information

4.3 Ontology-driven Management of Change

First of all, the author uses the document ontology of the PLATΩ system, the *proof language* PL, to semantically annotate the document in the text-editor. This ontology formalizes the document structure and is used by PLATΩ for the communication with the proof assistance system and the efficient propagation of changes. Beside that, the semantics of that document ontology allows to define an ontology inside individual documents: A concept like the predicate $\subset$ can be introduced together with alternative notations using the following annotation format.

```
\begin{definition}{Predicate $\in$}
  The predicate \concept{\in}{elem \times
  set \rightarrow bool} takes an individual
  and a set and tells whether that
  individual belongs to this set.
\end{definition}

\begin{notation}{Predicate $\in$}
  Let \declare{x} be an individual and
  \declare{A} a set, then we write
  \denote{x \in A}, \denote{x is element
  of A} or \denote{A contains x}.
\end{notation}
```

Concepts are implicitely related by their type information. Furthermore, they can be grouped and ordered by precedence using PLATΩ's ontology. All introduced concepts can directly be used in the same document for writing formulas, for example in axioms, theorems and proofsteps. With the integration of `locutor` into PLATΩ we aim at a management of change for documents that is driven by an ontology which is dynamically defined by the documents themselves.

In the following we will discuss the equivalence and change relations of `locutor` and how they can be adapted to the needs of PLATΩ with illustrative examples.

5 Equivalence Relations

A stronger notion of equality leads to more compact, less intrusive edit scripts. For instance, if we know that ordering of elements carries no meaning in a document format (think of BibTeX entries), two documents are considered equal, even if they differ in every single line (w.r.t. to the element order). Consequently, with this notion of equality, the computed edit script (a representation of the document differences) would be empty. This motivates the need to identify syntactically different but semantically equal INFOMs and thus to generate less intrusive edit scripts.

5.1 Primitive Equivalence Relations

In order to support such an efficient and less interfering management of change, `locutor` provides *primitive* equivalence relations. These are abstract specifications of equivalence relations, which have to be implemented in the respective document ontology. `locutor` provides a sophisticated plug-in mechanism for them, such that each plug-in provides its own $\mathcal{M}$`diff`-algorithm. These primitives specify and utilize semantic aspects of documents (INFOMs), in particular, they are used to identify syntactically different INFOMs as semantically equal. Changes are only propagated if they change the semantics. Typical examples are parsers of programming languages which will ignore, for instance, the number of white spaces between lexical symbols. A classical way to introduce such a classification on INFOMs is the use of equivalence relations R on INFOMs. Two INFOMs are considered as "semantically" equal w.r.t. an equivalence relations R iff both are in the same equivalence class. Labelling a dependency between two INFOMs with R, a change of one INFOM would be only propagated if that would also change the equivalence class the INFOM belongs to. Currently identified primitives are:

Include Normalization (`INC`)

Many document formats provide some kind of literal inclusion mechanism. For instance, OMDOC provides the `ref` element, TEX provides the `\include` and `\input` macros, and XML provides the construct of parsed entities and XINCLUDE [W3C, 2007]. We call the process of replacing an `include` element by its target in a document *include-reduction*, and the document resulting from the process of systematically and recursively reducing all the `include` elements the *include-normal form* of the source document. As include-normalization may not always be possible, e.g. if the targets do not exist or are inaccessible, we call a document *include-reducible*, iff its `include`-normal form exists, and *include-valid*, iff the `include` normal form exists and is a valid document of the class. Arbitrary *include-valid* documents are considered to be semantically equal to the respective *include-normal form*.

White-Spacing (`WHI`)

In various document formats, multiple consecutive white spaces do not carry any further semantics, but are for readability only. For instance, think of LATEX users using double white spaces, or a double newline to mark off the beginning of a new paragraph. Regarding XML documents, however, the indent level and white space only nodes do matter (cf. `xml:space` attribute). Finally, there is the application/operating system dependent white space and newline character encoding: the carriage return (`\r` or ch(13)), the linefeed (`\n` or ch(10)), the tab (`\t`), and the spacebar (' '). We subsume all these issues under the

term "white-spacing". By this equivalence relation white-spacing within arbitrary documents is ignored, i.e. documents only differing in white-spacing are considered to be equal.

Ordering ($\texttt{ORD}_{\texttt{DocFor}}$)
The order of elements matters in almost all document formats (DocFor), for instance, the raw XML snippet <root><A/><B/></root> is not equal to <root><B/><A/></root>. Additionally the order restriction may change between different elements, i.e. for some elements the order matters but is irrelevant for others in the same document format. Thus this primitive equivalence relation is relative to the grammar of the respective document format, in particular, relative to single elements. For example, key-value pairs in OMDoc do have a strict order, but CMP elements can be freely ordered. We differentiate between *loosely* and *strictly* specified elements. For *loosely* specified elements the logical dependency graph is mandatory to the structural dependency graph. That is documents with the same logical dependency graph but a different structural dependency graph w.r.t. the order of the elements are considered to be equal. For *strictly* specified elements the structural dependency graph implied by the respective grammar is at the front.

URI-Normalization (URI)
This primitive equivalence relation causes `locutor` to consider relative path statements to be equal to normalized path statements, i.e. to the corresponding resolved absolute path. For example, within a document the relative path statement URI(../../lwa07.tex) might normalize to URI(https://www.kwar.info/nmueller/conferences-lwa07/lwa07.tex) and both paths are considered to be $\equiv_{\text{URI}}$ equivalent.

Formulae (FOR)
Content representations of mathematical formulae like OpenMath or MathML come with their own equivalences. We subsume α-conversion, dispensable variables and nested attribution under the term "formulae-equivalence". *α-conversion* Regarding the OpenMath specification [Buswell *et al.*, 2004] binding objects are constructed from an OpenMath object, and from a sequence of zero or more variables followed by another OpenMath object. The first OpenMath object is the "binder" object. Arguments 2 to $n-1$ are always variables to be bound in the "body" which is the n^{th} argument object. We write $\beta(b, x_1, \ldots, x_m, O)$ where β denotes the OpenMath binding operation, $x_1, \ldots, x_m$ the bound variables, and O the body. For example, activating this equivalence relation leads `locutor` to consider the following α-equality: $\beta(\lambda, z, y, x, z(yx)) \equiv_{\text{FOR}} \beta(\lambda, f, g, t, f(gt)) \equiv_{\text{FOR}} \beta(\lambda, x, y, z, x(yz))$.

Dispensable Variables In OpenMath, repeated occurrences of the same variable in a binding operator are allowed. Thus a binding with multiple occurrences of the same variable is considered to be semantically equivalent to the binding in which all but the last occurrence of each variable is replaced by a new variable which does not occur free in the body of the binding. That is $\beta(\lambda, v, v, v \times v) \equiv_{\text{FOR}} \beta(\lambda, v', v, v \times v)$.

Nested Attribution An OpenMath attribution decorates an object with a sequence of one or more pairs made up of an OpenMath symbol, the "attribute", and an associated object, the "value". We write $\alpha(O, (k_i \mapsto v_i)_i)$ where α denotes the OpenMath attribution operations, k_i are OpenMath symbols, and v_i and O are OpenMath objects. As the value can be an OpenMath attribution object itself, compositions of attributions are allowed and are considered semantically equivalent to a single attribution. That is, $\alpha(\alpha(O, (k_i \mapsto v_i)_i), (k'_j \mapsto v'_j)_j) \equiv_{\text{FOR}} \alpha(O, (k_i \mapsto v_i, k'_j \mapsto v'_j)_{i,j})$.

Theory-Refactorization (THE)
Top-level OMDoc theories containing nested theories are semantically equal to the so-called *refactored* theories (modulo theory renaming). That is,

```
<theory xml:id="A">
  <symbol name="a"/>
  <theory xml:id="B">
    <symbol name="b"/>
  </theory>
</theory>
```

is equivalent to

```
<theory xml:id="A'">
  <symbol name="a"/>
</theory>
<theory xml:id="B'">
  <imports from="A'"/>
  <symbol name="b"/>
</theory>
```

The top-level theory A containing a nested theory B is equivalent to the refactored, top-level theories A' (w/o B) and B' where B' now imports A'. Note, the refactorization has to be performed from the outermost to the innermost theory.

We are currently investigating further primitive equivalence relations, like in an XML document, attributes having default values and attributes being absent are considered equal. This is important because many applications fill in default values automatically.

5.2 Conjoint Equivalence Relations

Figure 7 in the first column summarizes the previously described primitive equivalence relations. By postulating $\forall \rho, \sigma \in Eq. \rho\sigma = \sigma\rho \Rightarrow \rho\sigma = (\rho \cup \sigma)^*$ where $\rho\sigma$ and $(\rho \cup \sigma)^*$ are again equivalence relations, these primitives may be composed to computable *conjoint* equivalence relations. Default implementations for the document formats XML, OpenMath, and OMDoc are under construction. The predefined conjunctions $\equiv_{\text{XML}}$, $\equiv_{\text{OpenMath}}$ and $\equiv_{\text{OMDoc}}$ are contained in the respective document ontologies. A conjoint equivalence relations is interpreted as the

	$\equiv_{\text{XML}}$	$\equiv_{\text{OpenMath}}$	$\equiv_{\text{OMDoc}}$	$\equiv_{\text{CNXML}}$	$\equiv_{\text{TeX}}$
INC	√	√	√	√	√
WHI		√	√		√
$\text{ORD}_{\text{OMDoc}}$			√		
FOR		√	√		
URI	√	√	√	√	√
THE			√		

Figure 7: Equivalence relation matrix of `locutor`

transitive closure of the union of the implemented primitives, e.g. $\equiv_{\text{XML}} := (\texttt{INC} \cup \texttt{URI})^*$. Note, as for example $\equiv_{\text{XML}} \subseteq \equiv_{\text{OpenMath}} \subseteq \equiv_{\text{OMDoc}}$ holds, the plug-in specification of `locutor` also provides (and encourages) the reuse of implementations of primitives in "larger" conjunctions, i.e. inheritance between document ontologies, in

particular regarding $\mathcal{M}$diff-algorithms. To demonstrate the flexibility of the emerging equivalence relation matrix, we appended the conjunctions $\equiv_{\text{CNXML}}$ and $\equiv_{\text{TeX}}$ to emphasize the potential support of further document formats, e.g. the support of CNXML [Hendricks and Galvan, 2007], XHTML [W3C, 2000], MATHML [W3C, 2003], or even TEX.

6 Change Relations

The following change relations serve as classifications for computed structural differences and as such constitute the input of the change relation calculus. Depending on the classified modifications and the type of the dependency between the elements, `locutor` first reasons on and then propagates the changes w.r.t. the dependency types specified in the system ontology. We propose to annotate each dependency relation by a set of primitive equivalence relations on which they are sensitive to. In addition we propose to annotate each change relation by a set of primitive equivalence relations which they are violating. Thus, if the intersection of two such sets is not empty the change has to be propagated. Subject to the level of resulting consistency the author either retrieves precise (accumulated) locations within the document to manually re-check or the document is automatically adapted to a consistent state (w.r.t the system ontology). In the later case `locutor` returns the meta-diff[3] information to the PLATΩ system which is then able to call a meta-command in the mathematical assistance system instead of calling multiple commands for each single modification. Currently identified change relations are:

6.1 Conservative(φ)

Given a primitive equivalence relation φ, the parametrized change relation *Conservative* denotes a φ-equivalence preserving change. That is, modifying an element A to A' such that $A \equiv_\varphi A'$ still holds, `locutor` generates a *meta-diff* Δ_φ comprising both automatically adapted elements and those elements whose relation to A is violated by this modification. For example, a formula $\forall A, B.\ A = B \Leftrightarrow A \subset B \wedge B \subset A$ referring to $\forall A, B.\ A \subset B \Leftrightarrow \forall x. x \in A \Rightarrow x \in B$ via an operator name is not affected by an α-conversion, like $\forall D, C.\ D \subset C \Leftrightarrow \forall y. y \in D \Rightarrow y \in C$. However, renaming $\subset$ to $\sqsubset$ would violate the "referencing-by-name" relation from $=$ to $\subset$. In this case `locutor` infers a refactorization (cf. section 6.2) and propagates the change along the reverse dependency from $\subset$ to $=$.

6.2 Refactored(ϱ)

Given a sub-type of a refactoring $\varrho \in \{$`Renamed`, `Moved`, `Inlined`, `Deleted`, `Replaced`$\}$, the parameterized change relation denotes a syntatic change of type refactorization with sub-type ϱ. In this case `locutor` fully automatically propagates the changes and adapts the dependency graph. The in addition generated accumulated meta-diff Δ_ϱ comprising all adapted elements w.r.t. to ϱ is returned to the PLATΩ system.

If an element has been `Renamed`, then `locutor` automatically updates all dependent elements by adapting the respective OMDOC `ref` elements. For example, let the document D at revision 512 (denoted by D_{512}) contain $\forall A, B, C.\ A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ and the definition of $=$. If an author syntactically modifies the definiendum $=$, the `locutor` system will automatically propagate the changes along the logical dependency graph to all affected elements and return the respective meta-diffs. Thus PLATΩ is able to trigger the whole renaming in the mathematical assistance system instead of propagating each single modification which in general invalidates previously computed verifications. However, if an author accidentally renames a bound variable, for example, the bound variable A in $=$ to C, `locutor` infers the conservative change relation *Conservative*(FOR) and will automatically adjust the respective definition.

If an element has been `Moved`, for example, from one theory to another, then `locutor` will by this classification automatically update all dependent elements. For example, all values of OMDOC `ref` elements are adapted to the new location.

If an element has been `Inlined` at one certain location, i.e. expanding an element at call side and removing the element itself, then `locutor` automatically propagates this change to all dependent elements and updates the respective `ref` elements. For example, instead of entirely removing $=$ the author may perform an inlining, i.e. replacing each occurrence of $=$ by its definition. In this case `locutor` performs a parallel capture-avoiding substitution along the relations of the dependency graph. That is, we obtain $\forall A, B, C.\ (A \cap (B \cup C) \subset (A \cap B) \cup (A \cap C)) \wedge ((A \cap B) \cup (A \cap C) \subset A \cap (B \cup C))$. Regarding OMDOC, inlining means replacing all references to a `symbol` element (e.g. OPENMATH `OMS` elements) by its corresponding definition.

If an element has been completely `Removed`, then `locutor` accumulates all occurrences and notifies the author. For example, removing the definition of $=$ because of an existing eq funtion, leads `locutor` again to propagate the changes to all affected elements, resulting in $\forall A, B, C.\ \text{eq}(A \cap (B \cup C), (A \cap B) \cup (A \cap C))$.

If an element has been `Replaced` by a new one, then `locutor` updates all elements depending on the removed one by adapting their references. For example, replacing $\wedge$ by an already existent type-conform connective land (e.g. *Refactored*(`Replaced[`$\wedge$`/land]`)), causes `locutor` to substitute $\wedge$ by land in all affected elements, resulting in $\forall A, B.\ A = B \Leftrightarrow \text{land}(A \subset B, B \subset A)$.

6.3 Semantics

If an element is semantically changed, then `locutor` will accumulate all occurrences and notify the author to re-verify the respective dependent elements. For example, if $\subset$ is semantically changed (e.g. semantic modification on the definiens), `locutor` will accumulate all occurrences of $\subset$ and notify the author to re-verify the respective elements (and $\subset$ itself). In addition, if one changes the usage of $\subset$, e.g. by mistake, then `locutor` notifies the author, before PLATΩ has to re-compute the internal data structures.

7 Conclusion and Outlook

We have outlined the integration of the ontology-driven management of change system `locutor` into the elaborated interactive mathematical authoring framework PLATΩ. Automatic classification of changes can be worthwhile comparing to time-consuming computations in the worst case accounted to "slips of the pen".

[3] The concrete specification of a meta-diff is still under investigation.

The integration is at the moment at an early stage of development: The communication between the two systems has been discussed so far. That is, the requirements of the PLATΩ system to the `locutor` system are well understood. The next step is the specification and implementation of the herein described conflation. By accomplishing this task, the authors are confident in both identifying further requirements regarding the communication and continuing improving both systems.

Acknowledgments

The authors would like to thank Michael Kohlhase, Serge Autexier, and Florian Rabe for stimulating discussions in the first phase of this work.

References

[Autexier and Dietrich, 2006] S. Autexier and D. Dietrich. Synthesizing proof planning methods and Ωants agents from mathematical knowledge. In J. Borwein and B. Farmer, editors, *Proceedings of MKM'06*, volume 4108 of *LNAI*, pages 94–109. Springer, 2006.

[Autexier and Hutter, 2005] S. Autexier and D. Hutter. Formal software development in Maya. In D. Hutter and W. Stephan, editors, *Festschrift in Honor of J. Siekmann*, volume 2605 of *LNAI*. Springer, February 2005.

[Autexier *et al.*, 2007] Serge Autexier, Armin Fiedler, Thomas Neumann, and Marc Wagner. Supporting user-defined notations when integrating scientific text-editors with proof assistance systems. In Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors, *Towards Mechanized Mathematical Assistants*, LNAI. Springer, june 2007.

[Autexier, 2005] S. Autexier. The CoRe calculus. In R. Nieuwenhuis, editor, *Proceedings of CADE-20*, LNAI 3632, Tallinn, Estonia, July 2005. Springer.

[Buswell *et al.*, 2004] 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. `http://www.openmath.org/standard/om20`.

[CNX, 2007] Connexions. Project homepage at `http://www.cnx.org`, seen February 2007.

[Dietrich, 2006] D. Dietrich. The Task Layer of the ΩMEGASystem. Diploma thesis, Saarland University, Saarbrücken, Germany, 2006.

[Eberhardt and Kohlhase, 2004] Frederick Eberhardt and Michael Kohlhase. A Document-Sensitive XML-CVS Client. unpublished KWARC blue notes, 2004.

[Hendricks and Galvan, 2007] Brent Hendricks and Adan Galvan. The Connexions Markup Language (CNXML). `http://cnx.org/aboutus/technology/cnxml/`, 2007. Seen June 2007.

[Huang, 1996] X. Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. Number 112 in DISKI. Infix, Sankt Augustin, Germany, 1996.

[Kohlhase and Anghelache, 2003] Michael Kohlhase and Romeo Anghelache. Towards collaborative content management and version control for structured mathematical knowledge. In Andrea Asperti, Bruno Buchberber, and James Harold Davenport, editors, *Mathematical Knowledge Management, MKM'03*, number 2594 in LNCS, pages 147–161. Springer Verlag, 2003.

[Kohlhase, 2006] M. Kohlhase. *OMDOC - An Open Markup Format for Mathematical Documents [Version 1.2]*, volume 4180 of *LNAI*. Springer, August 2006.

[locutor, 2007] `locutor`: An Ontology-Based Management of Change, seen June 2007. system homepage at `http://www.kwarc.info/projects/locutor/`.

[MMiSS, 2007] MMiSS: Multimedia in Safe and Secure Systems. Web site at `www.mmiss.de`, seen July 2007.

[Müller, 2006] Normen Müller. An Ontology-Driven Management of Change. In *Wissens- und Erfahrungsmanagement LWA (Lernen, Wissensentdeckung und Adaptivität) conference proceedings*, 2006.

[Müller, 2007] Normen Müller. Towards an Ontology-Driven Management of Change. Exposé of PhD research proposal, June 2007.

[plato, 2007] Interactive Mathematical Authoring with PLATO, seen June 2007. System homepage at `http://www.ags.uni-sb.de/plato/bin/view.pl`.

[Siekmann *et al.*, 2002] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Proof development with OMEGA: $\sqrt{2}$ is irrational. In M. Baaz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 9th International Conference, LPAR 2002*, number 2514 in LNAI, pages 367–387. Springer, 2002.

[Verbert and Duval, 2004] Katrien Verbert and Erik Duval. Towards a Global Component Architecture for Learning Objects: A Comparative Analysis of Learning Object Content Models. In *Proceedings of the EDMEDIA 2004 World Conference on Educational Multimedia, Hypermedia and Telecommunications*, pages 202–208, 2004.

[W3C, 2000] W3C. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). `http://www.w3.org/TR/xhtml1/`, 2000. Seen July 2007.

[W3C, 2003] W3C. Mathematical Markup Language (MathML) Version 2.0 (Second Edition). `http://www.w3.org/TR/MathML2/`, 2003. Seen July 2007.

[W3C, 2007] W3C. XML Inclusions (XInclude) Version 1.0 (Second Edition), seen July 2007. http://www.w3.org/TR/xinclude/.

[Wagner *et al.*, 2006] M. Wagner, S. Autexier, and C. Benzmüller. PLATΩ: A mediator between text-editors and proof assistance systems. In C. Benzmüller S. Autexier, editor, *7th Workshop on User Interfaces for Theorem Provers (UITP'06)*, ENTCS. Elsevier, August 2006.

[Wiedijk, 2004] F. Wiedijk. Formal proof sketches. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs: Third International Workshop, TYPES 2003*, LNCS 3085, pages 378–393, Torino, Italy, 2004. Springer.