sTeXIDE: An Integrated Development Environment for sTeX Collections

Constantin Jucovschi and Michael Kohlhase

Computer Science, Jacobs University Bremen
{c.jucovschi,m.kohlhase}@jacobs-university.de

Abstract. Authoring documents in MKM formats like OMDoc is a very tedious task. After years of working on a semantically annotated corpus of sTeX documents (GenCS), we identified a set of common, time-consuming subtasks, which can be supported in an integrated authoring environment.
We have adapted the modular Eclipse IDE into sTeXIDE, an authoring solution for enhancing productivity in contributing to sTeX based corpora. sTeXIDE supports context-aware command completion, module management, semantic macro retrieval, and theory graph navigation.

1 Introduction

Before we can manage mathematical 'knowledge' — i.e. reuse and restructure it, adapt its presentation to new situations, semi-automatically prove conjectures, search it for theorems applicable to a given problem, or conjecture representation theorems, we have to convert informal knowledge into machine-oriented representations. How to exactly support this formalization process so that it becomes as effortless as possible is one of the main unsolved problems of MKM. Currently most mathematical knowledge is available in the form of LaTeX-encoded documents. To tap this reservoir we have developed the sTeX [Koh08,sTe09] format, a variant of LaTeX that is geared towards marking up the semantic structure underlying a mathematical document.

In the last years, we have used sTeX in two larger case studies. In the first one, the second author has accumulated a large corpus of teaching materials, comprising more than 2,000 slides, about 800 homework problems, and hundreds of pages of course notes, all written in sTeX. The material covers a general first-year introduction to computer science, graduate lectures on logics, and research talks on mathematical knowledge management. The second case study consists of a corpus of semi-formal documents developed in the course of a verification and SIL3-certification of a software module for safety zone computations [KKL10a,KKL10b]. In both cases we took advantage of the fact that sTeX documents can be transformed into the XML-based OMDoc [Koh06] by the LaTeXML system [Mil10], see [KKL10a] and [DKL+10] for a discussion on the MKM services afforded by this.

These case studies have confirmed that writing sTeX is *much* less tedious than writing OMDoc directly. In particular, the possibility of using the sTeX-generated PDF for proofreading the text part of documents. Nevertheless serious usability problems remain. They come from three sources:

P1 installation of the (relatively heavyweight) transformation system (with dependencies on `perl`, `libXML2`, LaTeX, the sTeX packages),

P2 the fact that sTeX supports an object-oriented style of writing mathematics, and

P3 the size of the collections which make it difficult to find reusable components.

The documents in the first (educational) corpus were mainly authored directly in sTeX via a text editor (`emacs` with a simple sTeX mode [Pes07]). This was serviceable for the author, who had a good recollection names of semantic macros he had declared, but presented a very steep learning curve for other authors (e.g. teaching assistance) to join. The software engineering case study was a post-mortem formalization of existing (informal) LaTeX documents. Here, installation problems and refactoring existing LaTeX markup into more semantic sTeX markup presented the main problems.

Similar authoring and source management problems are tackled by Integrated Development Environments (IDEs) like Eclipse [Ecl08], which integrate support for finding reusable functions, refactoring, documentation, build management, and version control into a convenient editing environment. In many ways, sTeX shares more properties with programming languages like Java than with conventional document formats, in particular, with respect to the three problem sources mentioned above

S1 both require a build step (compiling Java and formatting/transforming sTeX into PDF/OMDoc),

S2 both favor an object-oriented organization of materials, which allows to

S3 build up large collections of re-usable components

To take advantage of the solutions found for these problems by software engineering, we have developed the sTeXIDE integrated authoring environment for sTeX-based representations of mathematical knowledge. In the next section we recap the parts of sTeX needed to understand the system. In Section 3 we present the user interface of the sTeXIDE system, and in Section 4 we discuss implementation issues. Section 5 concludes the paper and discusses future work.

2 sTeX: Object-Oriented LaTeX Markup

The main concept in sTeX is that of a "*semantic macro*", i.e. a TeX command sequence $\mathcal{S}$ that represents a meaningful (mathematical) concept or object $\mathcal{O}$: the TeX formatter will expand $\mathcal{S}$ to the presentation of $\mathcal{O}$. For instance, the command sequence `\positiveReals` is a semantic macro that represents a mathematical symbol — the set $\mathbb{R}^+$ of positive real numbers. While the use of semantic macros is generally considered a good markup practice for scientific documents[1],

[1]e.g., because they allow to adapt notation by macro redefinition and thus increase reusability.

regular TeX/LaTeX does not offer any infrastructural support for this. sTeX does just this by adopting a semantic, "object-oriented" approach to semantic macros by grouping them into "modules", which are linked by an "imports" relation. To get a better intuition, consider the example in listing 1.1.

Listing 1.1. An sTeX module for Real Numbers

```
\begin{module}[id=reals]
  \importmodule[../background/sets]{sets}
  \symdef{Reals}{\mathcal{R}}
  \symdef{greater}[2]{#1>#2}
  \symdef{positiveReals}{\Reals^+}
  \begin{definition}[id=posreals.def,title=Positive Real Numbers]
    The set $\positiveReals$ is the set of $\inset{x}\Reals$ such that $\greater{x}0$
  \end{definition}
  ...
\end{module}
```

which would be formatted to

Definition 2.1 (Positive Real Numbers):
The set $\mathbb{R}^+$ is the set of $x \in \mathbb{R}$ such that $x > 0$

Note that the markup in the module `reals` has access to semantic macro `\inset` (element-hood) from the module `sets` that was imported by the document `\importmodule` directive from the `../background/sets.tex`. Furthermore, it has access to the `\defeq` (definitional equality) that was in turn imported by the module `sets`.

From this example we can already see an organizational advantage of sTeX over LaTeX: we can define the (semantic) macros close to where the corresponding concepts are defined, and we can (recursively) import mathematical modules. But the main advantage of markup in sTeX is that it can be transformed to XML via the LaTeXML system [Mil10]: Listing 1.2 shows the OMDoc [Koh06]. representation generated from the sTeX sources in listing 1.1.

Listing 1.2. An XML Version of Listing 1.1

```
<theory xml:id="reals">
 <imports from="../background/sets.omdoc#sets"/>
 <symbol xml:id="Reals"/>
 <notation>
   <prototype><OMS cd="reals" name="Reals"/></prototype>
     <rendering><m:mo>ℝ</m:mo></rendering>
 </notation>
   <symbol xml:id="greater"/><notation>...</notation>
   <symbol xml:id="positiveReals"/><notation>...</notation>
   <definition xml:id="posreals.def" for="positiveReals">
     <meta property="dc:title">Positive Real Numbers</meta>
     The set <OMOBJ><OMS cd="reals" name="postiveReals"/></OMOBJ> is the set ...
   </definition>
   ...
</theory>
```

One thing that jumps out from the XML in this listing is that it incorporates all the information from the sTeX markup that was invisible in the PDF produced by formatting it with TeX.

3 User interface features of STeXIDE

One of the main priorities we set for STeXIDE is to have a relatively gentle learning curve. As the first experience of using a program is running the installation process, we worked hard into making this step as automated and platform independent as possible. We aim at supporting popular operating systems such as Windows and Unix based platforms (Ubuntu, SuSE). Creating a OS independent distribution of Eclipse with our plugin preinstalled was a relatively straightforward task; so was distributing the plugin through an update site. What was challenging was getting the 3rd party software (`pdflatex`, `svn`, `latexml`, `perl`) and hence OS specific ports installed correctly.

After installation we provide a new project wizard for STeX projects which lets the user choose the output format (`.dvi`, `.pdf`, `.ps`, `.omdoc`, `.xhtml`) as well as one of the predefined sequences of programs to be executed for build process. This will control the ECLIPSE-like workflow, where the chosen 'outputs' are rebuilt after every save, and syntactic (as well as semantic) error messages are parsed, cross-referenced, and displayed to the user in a collapsible window. The wizard then creates a stub project, i.e. a file `main.tex` which has the structure of a typical STeX file but also includes `stex` package and imports a sample module defined in `sample\_mod.tex`.

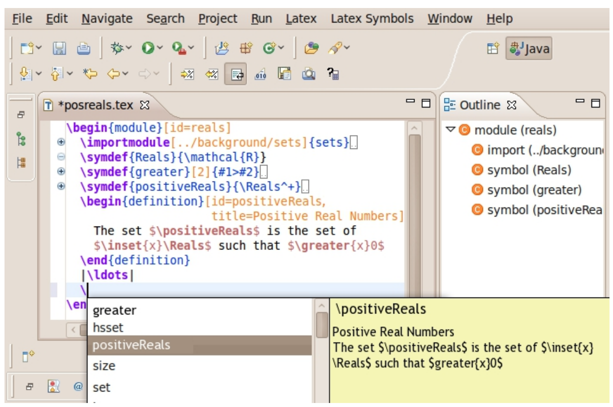
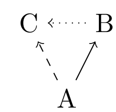


Fig. 1. Context aware autocompletion feature for semantic macros

STeXIDE supports the user in creating, editing and maintaining STeX documents or corpora. For novice users we provide templates for creating modules, imports and definitions. Later on, user benefits from *context-aware autocompletion*, which assists the user in writing valid LaTeX and STeX macros. Here, by valid macros, we mean macros which were previously defined or imported (both directly or indirectly) from other modules. Consider sample STeX source in listing 1.1. At the end of first line, one would only be able to autocomplete LaTeX macros, whereas at the end of second line one would already have macros like `\inset` from the imported `sets` module (see figure 1). Note that we also make use of the semantic strucuture of the STeX document in listing 1.1 for explanations: the macro `\positiveReals` is linked to the definition via the key for=positiveReals, so we can display the text of the definition as an explanation in the yellow box.

Similarly, *semantic macro retrieval* (triggered by typing '*') will suggest all available macros from all modules of current project. In case auto-completed macro is not valid for current context, STeXIDE will insert the required import statement so that macro becomes valid.

Moreover, STeXIDE supports several typical document/collection maintenance tasks: Supporting *symbol and module names refactoring* is very important as it is extremely error-prone, especially if two different modules define a symbol with the same name and only one of them is to be renamed. The *module splitting* feature makes it easier for users to create smaller but semantically self contained modules which one can easier reuse. This feature takes care that needed imports are copied in the newly created module.



At last, *import minimization* creates warnings for unused or redundant `\importmodule` declarations and suggests to remove them. Consider for instance the situation on the right, where a modules `C` and `B` imports module `A`. Now, if we add a semantic macro in `C` that needs an import from `B`, then we should replace the import of `A` in `C` with one of `B` instead of just adding the latter (i.e. we would replace the dashed by the dotted import).

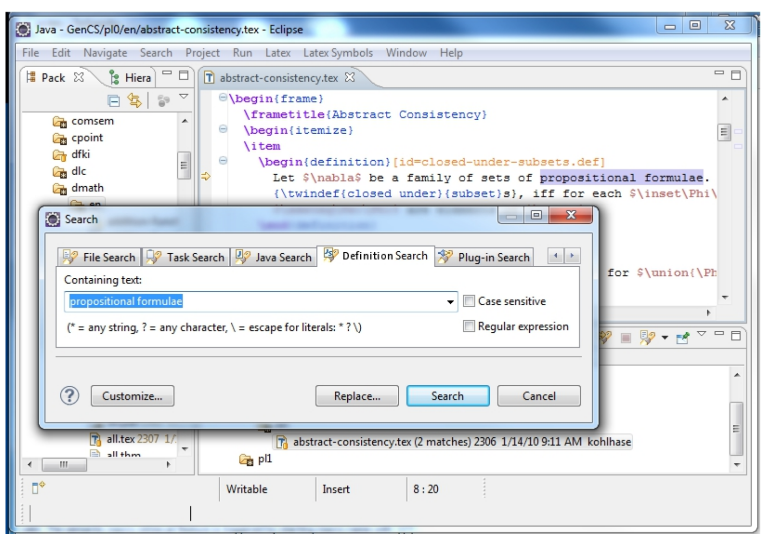


Fig. 2. Macro Retrieval via Mathematical Concepts

Three additional features make navigation and information retrieval in big corpora easier. Outline view of the document (right side of figure 1) displays main semantic document structures. One can use this layout to copy, cut and navigate to areas represented by respective structures. In case of imports one can navigate to imported modules. Theory graph navigation is another feature which creates a graphical representation of how modules are related through imports. This gives user a chance to get a better intuition how concepts and modules are related. And the last feature is the semantic definition search feature. The aim of this feature is to search for semantic macros by their mathematical descriptions, which can be entered into search box in figure 2. This then searches definitions, assumptions, and theorems for the query terms and reports any `\symdef`-defined semantic macros 'near' the hits. This has turned out very convenient in situations, where the macro names are abbreviated (e.g. `\sconcjuxt` for "string concatenation by juxtaposition") or if there are more than one name for a mathematical context (e.g. "concatenation" for `\sconcjuxt`.) and the author wants to re-use semantic macros defined by someone else.

4 Implementation

The implementation of sTeXIDE is based on the TeXlipse [TeX08] plugin for Eclipse. This plugin makes use of Eclipse's modular framework (see figure 3) and provides features like syntax highlighting, code folding, outline generation, autocompletion and templating mechanisms. Unfortunately, the recognizer for a fixed set of LaTeX macros like `\section`, `\input`, etc. is hardwired which made it quite challenging to generalize it to sTeX specific macros. Therefore we had to reimplement parts of TeXlipse so that sTeX macros like `\symdef` and `\importmodule` that extend the set of available macros can be treated specially. We have underlined all the parts of TeXlipse we had to extend or replace in Figure 3.

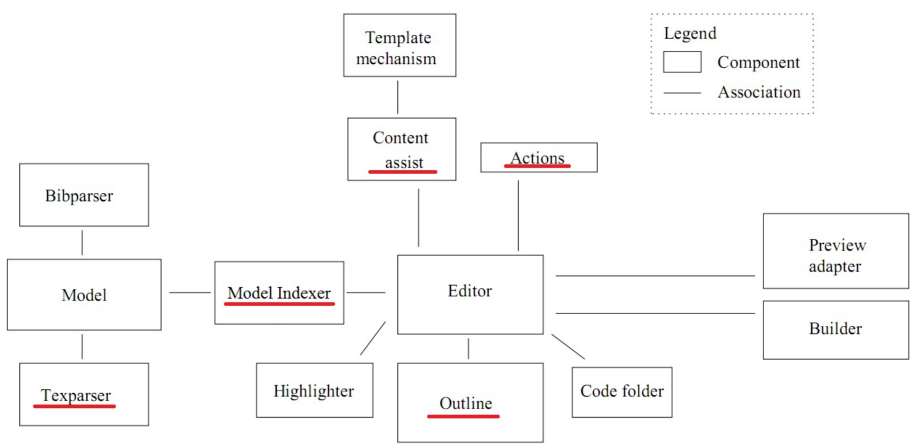


Fig. 3. Component architecture of TeXlipse (adapted from [?])

To support context sensitive autocompletion and refactoring we need to know the exact position in the source code where modules and symbols are defined. Running a full featured LaTeX parser like LaTeXML proved to be too slow (sometimes taking 5-10 sec) and sensitive to errors. For these reasons, we implemented a very fast but naïve LaTeX parser which analyses the source code and identifies commands, their arguments and options. We call this parser naïve because it parses only one file a time (i.e. inclusions, and styles are not processed) and macros are not expanded. We realize the parse tree as an in-memory XML DOM to achieve format independence (see below). Then we run a set of semantic spotters which identify constructs like module and import declarations, inclusions as well as sections/subsections etc, resulting in an index of relevant structural parts of the sTeX source identified by unique URIs and line/column number ranges in the source. For example, a module definition in sTeX begins with `\begin{module}[id=module_id]` and ends in a `\end{module}`, so the structure identifying a module will contain these two ranges.

Note that the LaTeX document model (and thus that of sTeX) is a tree, so two spotted structure domains are either disjoint or one contains the other, so we implement a range tree we use for efficient change management: sTeXIDE implements a class which listens to changes made in documents, checks if they intersect with the important ranges of the spotted structures or if they introduce new commands (i.e. start with '\'). If this not, the range tree is merely updated by calculating new line and column numbers. Otherwise we run the naïve LaTeX parser and the spotters again.

Our parser is entirely generated by a JavaCC grammar, supports error recovery (essential for autocompletion) and does not need to be changed if a new macro needs to be handled: Semantic Spotters can be implemented as XQueries, and our parser architecture provides an API for adding custom semantic spotters. This makes the parser extensible to new sTeX features and allows to work around the limitation of the naïve LaTeX parser of not expanding macros.

We implemented several indexes to support features mentioned in section 3. For theory navigation we have an index called `TheoryIndex` which manages a directed graph of modules and import relationships among them. It allows *a)* retrieving list of modules which import/are imported by module X *b)* checking if module X is directly/indirectly imported by module Y. `SymdefIndex` is another index which stores pairs of module URIs and symbols defined in those modules. It supports fast retrieving of (symbol,module) pairs where symbol name starts with a certain prefix using a trie data structure. As expected this index is used for both context aware autocompletion as well as semantic macro retrieval features. The difference is that context aware autocompletion feature also filters the modules not accessible from current module by using the `TheoryIndex`. Refactoring makes use of an index called `RefIndex`. This index stores (module URI, definition module URI, symbol name) triples for all symbol occurrences (not just definitions as in `SymdefIndex`).

5 Conclusion and Future Work

We have presented the sTeXIDE system, an integrated authoring environment for sTeX collections realized as a plugin to the Eclipse IDE. Even though the implementation is still in a relatively early state, this experiment confirmed the initial expectation that the installation, navigation, and build support features contributed by Eclipse can be adapted to a useful authoring environment for sTeX with relatively little effort. The modularity framework of Eclipse and the TeXlipse plugin for LaTeX editing have been beneficial for our development. However, we were rather surprised to see that a large part of the support infrastructure we would have expected to be realized at the framework were indeed hard-coded into the plugins. This has resulted in un-necessary re-implementation work.

In particular, system- and collection-level features of sTeXIDE like automated installation, PDF/XML build support, and context-sensitive completion of command sequences, import minimziation, navigation, and concept-based search have proven useful, and are not offered by document-oriented editing solutions. Indeed such features are very important for editing and maintaining any MKM representations. Therefore we plan to extend sTeXIDE to a general "MKM IDE", which supports more MKM formats and their human-oriented front-end syntaxes (just like sTeX serves a front-end to OMDoc in sTeXIDE).

The modular structure of Eclipse also allows us to integrate MKM services (e.g. information retrieval from the background collection or integration of external proof engines for formal parts [ALWF06]; see [KRZ10] for others) into this envisioned "MKM IDE", so that it becomes a "rich collection client" to a *a universal digital mathematics library (UDML)*, which *would continuously grow and in time would contain essentially all mathematical knowledge* envisioned as the Grand Challenge for MKM in [Far05].

In the implementation effort we tried to abstract from the sTeX surface syntax, so that we anticipate that we will be able to directly re-use our spotters or adapt them for other surface formats that share the OMDoc data model. The next target in this direction is the modular LF format introduced in [RS09]. This can be converted to OMDoc by the TWELF system, which makes its treatment directly analogous to sTeX, this would provide a way of information sharing among different authoring systems and workflows.

References

[ALWF06] David Aspinall, Christoph Lüth, Daniel Winterstein, and Ahsan Fayyaz. Proof general in eclipse. In *Eclipse Technology eXchange ETX'06*. ACM Press, 2006.

[DKL+10] Catalin David, Michael Kohlhase, Christoph Lange, Florian Rabe, Nikita Zhiltsov, and Vyacheslav Zholudev. Publishing math lecture notes as linked data. In Lora Aroyo, Grigoris Antoniou, and Eero Hyvönen, editors, *ESWC*, Lecture Notes in Computer Science. Springer, June 2010. In press.

[Ecl08] Eclipse: An open development platform, seen May 2008.

[Far05] William M. Farmer. Mathematical Knowledge Management. In David G. Schwartz, editor, *Encyclopedia of Knowledge Management*, pages 599–604. Idea Group Reference, 2005.

[KKL10a] Andrea Kohlhase, Michael Kohlhase, and Christoph Lange. Dimensions of formality: A case study for MKM in software engineering. submitted to MKM (Mathematical Knowledge Management) 2010, 2010.

[KKL10b] Andrea Kohlhase, Michael Kohlhase, and Christoph Lange. sTeX – a system for flexible formalization of linked data. submitted to I-SEMANTICS 2010, 2010.

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, August 2006.

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

[KRZ10] Michael Kohlhase, Florian Rabe, and Vyacheslav Zholudev. Towards mkm in the large: Modular representation and scalable software architecture. submitted to MKM (Mathematical Knowledge Management) 2010, 2010.

[Mil10] Bruce Miller. `LaTeXML`: A LaTeX to XML converter. Web Manual at `http://dlmf.nist.gov/LaTeXML/`, seen March 2010.

[MKM10] *MKM 2010*, 2010. submitted to MKM (Mathematical Knowledge Management) 2010.

[Pes07] Darko Pesikan. Coping with content representations of mathematics in editor environments: nOMDoc mode. Bachelor's thesis, Computer Science, Jacobs University, Bremen, 2007.

[RS09] F. Rabe and C. Schürmann. A Practical Module System for LF. In *Proceedings of the Workshop on Logical Frameworks Meta-Theory and Practice (LFMTP)*, 2009.

[sTe09] Semantic Markup for LaTeX, seen July 2009. available at `http://kwarc.info/projects/stex/`.

[TeX08] Texlipse: Adding latex support to the eclipse ide., seen May 2008.