`smultiling.sty`: Multilinguality Support for sTeX

Michael Kohlhase, Deyan Ginev
Jacobs University, Bremen
`http://kwarc.info/kohlhase`

November 19, 2015

Abstract

The `smultiling` package is part of the sTeX collection, a version of TeX/LaTeX that allows to markup TeX/LaTeX documents semantically without leaving the document format, essentially turning TeX/LaTeX into a document format for mathematical knowledge management (MKM).

The `smultiling` package adds multilinguality support for sTeX, the idea is that multilingual modules in sTeX consist of a module signature together with multiple language bindings that inherit symbols from it, which also account for cross-language coordination.

Contents

1 Introduction **2**
1.1 sTeX Module Signatures 2

2 The User Interface **2**
2.1 Multilingual Modules 3
2.2 Multilingual Definitions and Crossreferencing Terms 3
2.3 Multilingual Views 4

3 Implementation **6**
3.1 Class Options 6
3.2 Signatures 6
3.3 Language Bindings 7
3.4 Multilingual Statements and Terms 9
3.5 Miscellaneneous 9
3.6 Finale 9

1 Introduction

We have been using sTeX as the encoding for the Semantic Multilingual Glossary of Mathematics (SMGloM; see [**IanJucKoh:sps14**]). The SMGloM data model has been taxing the representational capabilities of sTeX with respect to multilingual support and verbalization definitions; see [**Kohlhase:dmesmgm14**], which we assume as background reading for this note.

1.1 sTeX Module Signatures

(monolingual) sTeX had the intuition that the symbol definitions (`\symdef` and `\symvariant`) are interspersed with the text and we generate sTeX module signatures (SMS `*.sms` files) from the sTeX files. The SMS duplicate "formal" information from the "narrative" sTeX files. In the SMGloM, we extend this idea by making the the SMS primary objects that contain the language-independent part of the formal structure conveyed by the sTeX documents and there may be multiple narrative "language bindings" that are translations of each other – and as we do not want to duplicate the formal parts, those are inherited from the SMS rather than written down in the language binding itself. So instead of the traditional monolingual markup in Figure 1, we we now advocate the divided style in Figure 2.

```
\begin{module}[id=foo]
\symdef{bar}{BAR}
\begin{definition}[for=bar]
  A \defiii{big}{array}{raster} ($\bar$) is a\ldots, it is much bigger
  than a \defiii[sar]{small}{array}{raster}.
\end{definition}
\end{module}
```

Example 1: A module with definition in monolingual sTeX

We retain the old `module` environment as an intermediate stage. It is still useful for monolingual texts. Note that for files with a module, we still have to extract `*.sms` files. It is not completely clear yet, how to adapt the workflows. We clearly need a lmh or editor command that transfers an old-style module into a new-style signature/binding combo to prepare it for multilingual treatment.

2 The User Interface

`langfiles` The `smultiling` package accepts the `langfiles` option that specifies – for a module ⟨*mod*⟩ that the module signature file has the name ⟨*mod*⟩`.tex` and the language bindings of language with the ISO 639 language specifier ⟨*lang*⟩ have the file name ⟨*mod*⟩`.`⟨*lang*⟩`.tex`.[1]

EdN:1

[1] EdNote: implement other schemes, e.g. the onefile scheme.

```
\usepackage[english,ngerman]{multiling}
\begin{modsig}{foo}
\symdef{bar}{BAR}
\symi{sar}
\end{modsig}

\begin{modnl}[creators=miko,primary]{foo}{en}
\begin{definition}
  A \defiii[bar]{big}{array}{raster} ($\bar$) is a\ldots, it is much bigger
  than a \defiii[sar]{small}{array}{raster}.
\end{definition}
\end{modnl}

\begin{modnl}[creators=miko]{foo}{de}
\begin{definition}
  Ein \defiii[bar]{gro"ses}{Feld}{Raster} ($\bar$) ist ein\ldots, es
  ist viel gr"o"ser als ein \defiii[sar]{kleines}{Feld}{Raster}.
\end{definition}
\end{modnl}
```

Example 2: Multilingual STEX for Figure 1.

2.1 Multilingual Modules

`modsig` There the `modsig` environment works exactly like the old `module` environment, only that the `id` attribute has moved into the required argument – anonymous module signatures do not make sense.

`modnl` The `modnl` environment takes two arguments the first is the name of the module signature it provides language bindings for and the second the ISO 639 language specifier of the content language. We add the `primary` key `modnl`, which can specify the primary language binding (the one the others translate from; and which serves as the reference in case of translation conflicts).[2]

EdN:2

`\symi` `\symi*` `\symii` `\symiii` There is another difference in the multilingual encoding: All symbols are introduced in the module signature, either by a `\symdef` or the new `\symi` macro. `\symi{`⟨*name*⟩`}` takes a symbol name ⟨*name*⟩ as an argument and reserves that name. The variant `\symi*{`⟨*name*⟩`}` declares ⟨*name*⟩ to be a primary symbol; see [**Kohlhase:dmesmgm14**] for a discussion. STEX provides variants `\symii` and `\symiii` – and their starred versions – for multi-part names.

2.2 Multilingual Definitions and Crossreferencing Terms

We do not need a new infrastructure for defining mathematical concepts, only the realization that symbols are language-independent. So we can use symbols for the coordination of corresponding verbalizations. As the example in Figure 2 already shows, we can just specify the symbol name in the optional argument of the `\defi` macro to establish that the language bindings provide different verbalizations of the same symbol.

[2] EdNote: @DG: This needs to be implemented in LaTeXML

For multilingual term references the situtation is more complex: For single-word verbalizations we could use `\atrefi` for language bindigns. Say we have introduced a symbol `foo` in English by `\defi{foo}` and in German by `\defi[foo]{Foo}`. Then we can indeed reference it via `\trefi{foo}` and `\atrefi{Foo}{foo}`. But one the one hand this blurs the distinction between translation and "linguistic variants" and on the other hand does not scale to multi-word compounds as `bar` in Figure 2, which we would have to reference as `\atrefiii{gro"ses Feld Raster}{bar}`. To avoid this,

`\mtref*` the `smultiling` package provides the new macros `\mtrefi`, `\mtrefii`, and `\mtrefiii` for multilingual references. Using this, we can reference `bar` as `\mtrefiii[?bar]{gro"ses}{Feld}{Raster}`, where we use the (up to three) mandatory arguments to segment the lexical constituents.

The first argument it syntactially optional to keep the parallelity to `\*def*` `\*tref*` it specifies the symbol via its name ⟨*name*⟩ and module name ⟨*mod*⟩ in a MMT URI ⟨*mod*⟩`?`⟨*name*⟩. Note that MMT URIs can be relative:

1. `foo?bar` denotes the symbol `bar` from module `foo`
2. `foo` the module `foo` (the symbol name is induced from the remaining arguments of `\mtref*`)
3. `?bar` specifies symbol `bar` from the current module

Note that the number suffix `i`/`ii`/`iii` indicates the number of words in the actual language binding, not in the symbol name as in `\atref*`.

2.3 Multilingual Views

Views receive a similar treatment as modules in the `smultiling` package. A

`viewsig` multilingual view consists of a view signature marked up with the `viewsig` environment. This takes three required arguments: a view name, the source module, and the target module. The optional first argument is for metadata (`display`, `title`, `creators`, and `contributors`) and load information (`frompath`,

EdN:3 `fromrepos`, `topath`, and `torepos`).[3]

```
\begin{viewsig}[creators=miko,]{norm-metric}{metric-space}{norm}
  \vassign{base-set}{base-set}
  \vassign{metric}{\funcdot{x,y}{\norm{x-y}}}
\end{viewsig}
```

Views have language bindings just as modules do, in our case, we have

```
\begin{gviewnl}[creators=miko]{norm-metric}{en}{norm}{metric-space}
  \obligation{metric-space}{obl.norm-metric.en}
  \begin{assertion}[type=obligation,id=obl.norm-metric.en]
    $\defeq{d(x,y)}{\norm{x-y}}$ is a \trefii[metric-space]{distance}{function}
  \end{assertion}
  \begin{sproof}[for=obl.norm-metric.en]
    {we prove the three conditions for a distance function:}
```

[3] EdNote: MK: that does not work yet, what we describe here is `mhviewig`; we need to refactor further.

```
    ...
  \end{sproof}
\end{gviewnl}
```

3 Implementation

The general preamble for LaTeXML

```
⟨*ltxml⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
⟨/ltxml⟩
```

3.1 Class Options

```
⟨*sty⟩
\newif\if@smultiling@mh@\@smultiling@mh@false
\DeclareOption{mh}{\@smultiling@mh@true}
\newif\if@langfiles\@langfilesfalse
\DeclareOption{langfiles}{\@langfilestrue}
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{modules}}
\ProcessOptions
⟨/sty⟩
⟨*ltxml⟩
DeclareOption('mh', sub { AssignValue ('@smultiling' => 1);
PassOptions('modules','sty',ToString(Digest(T_CS('\CurrentOption')))); });
DeclareOption('langfiles',sub {AssignValue('smultiling_langfiles',1,'global');});
DeclareOption(undef,sub {PassOptions('modules','sty',ToString(Digest(T_CS('\CurrentOption'))));
ProcessOptions();
⟨/ltxml⟩
```

We load the packages referenced here.

```
⟨*sty⟩
\if@smultiling@mh@\RequirePackage{smultiling-mh}\fi
\RequirePackage{etoolbox}
\RequirePackage{structview}
⟨/sty⟩
⟨*ltxml⟩
if(LookupValue('@smultiling')) {RequirePackage('smultiling-mh');}
RequirePackage('structview');
⟨/ltxml⟩
```

3.2 Signatures

`modsig` The `modsig` environment is just a layer over the `module` environment. We also redefine macros that may occur in module signatures so that they do not create markup.

```
⟨ltxml⟩RawTeX('
⟨*sty | ltxml⟩
\newenvironment{modsig}[2][]{%
\def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2]\else\begin{module}[id=#2,#1]\fi}
{\end{module}}
```

viewsig The viewsig environment is just a layer over the view environment with the keys suitably adapted.

```
\newenvironment{viewsig}[4][]{\def\@test{#1}\ifx\@test\@empty%
\begin{view}[id=#2,ext=tex]{#3}{#4}\else\begin{view}[id=#2,#1,ext=tex]{#3}{#4}\fi}
{\end{view}}
⟨/sty | ltxml⟩
⟨ltxml⟩');
```

\@sym* has a starred form for primary symbols.

```
⟨*sty⟩
\newcommand\symi{\@ifstar\@symi@star\@symi}
\newcommand\@symi[1]{\if@importing\else Symbol: \textsf{#1}\fi}
\newcommand\@symi@star[1]{\if@importing\else Primary Symbol: \textsf{#1}\fi}
\newcommand\symii{\@ifstar\@symii@star\@symii}
\newcommand\@symii[2]{\if@importing\else Symbol: \textsf{#1-#2}\fi}
\newcommand\@symii@star[2]{\if@importing\else Primary Symbol: \textsf{#1-#2}\fi}
\newcommand\symiii{\@ifstar\@symiii@star\@symiii}
\newcommand\@symiii[3]{\if@importing\else Symbol: \textsf{#1-#2-#3}\fi}
\newcommand\@symiii@star[3]{\if@importing\else Primary Symbol: \textsf{#1-#2-#3}\fi}
⟨/sty⟩
⟨*ltxml⟩
DefConstructor('\symi OptionalMatch:* {}',
     "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2'/>");
DefConstructor('\symii OptionalMatch:* {} {}',
     "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2-#3'/>");
DefConstructor('\symiii OptionalMatch:* {} {} {}',
     "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2-#3-#4'/>");
⟨/ltxml⟩
```

3.3 Language Bindings

modnl:*

```
⟨*sty⟩
\addmetakey{modnl}{load}
\addmetakey*{modnl}{title}
\addmetakey*{modnl}{creators}
\addmetakey*{modnl}{contributors}
\addmetakey{modnl}{srccite}
\addmetakey{modnl}{primary}[yes]
⟨/sty⟩
⟨*ltxml⟩
DefKeyVal('modnl','title','Semiverbatim');
DefKeyVal('modnl','load','Semiverbatim');
DefKeyVal('modnl','creators','Semiverbatim');
DefKeyVal('modnl','contributors','Semiverbatim');
DefKeyVal('modnl','primary','Semiverbatim');
⟨/ltxml⟩
```

modnl The `modnl` environment is just a layer over the `module` environment and the `\importmodule` macro with the keys and language suitably adapted.

```
⟨*sty⟩
\newenvironment{modnl}[3][]{\metasetkeys{modnl}{#1}%
\def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2.#3]\else\begin{module}[id=#2.#3,#1]\fi%
\if@langfiles\importmodule[load=#2,ext=tex]{#2}\else
\ifx\modnl@load\@empty\importmodule{#2}\else\importmodule[ext=tex,load=\modnl@load]{#2}\fi%
\fi}
{\end{module}}
⟨/sty⟩
⟨*ltxml⟩
DefEnvironment('{modnl} OptionalKeyVals:modnl {}{}',
         "?#excluded()(<omdoc:theory xml:id='#2.#3'>"
         .   "?&defined(&GetKeyVal(#1,'creators'))(<dc:creator>&GetKeyVal(#1,'creators')</dc:cr
         .   "?&defined(&GetKeyVal(#1,'title'))(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
         .   "?&defined(&GetKeyVal(#1,'contributors'))(<dc:contributor>&GetKeyVal(#1,'contribut
         .   "<omdoc:imports from='?&GetKeyVal(#1,'load')(&canonical_omdoc_path(&GetKeyVal(#1,'
         .   "#body"
         . "</omdoc:theory>)",
  afterDigestBegin=>sub {
    my ($stomach, $whatsit) = @_;
    my $keyval = $whatsit->getArg(1);
    my $signature = ToString($whatsit->getArg(2));
    my $language = ToString($whatsit->getArg(3));
    if ($keyval) {
      # If we're not given load, AND the langfiles option is in effect,
      # default to #2
      if ((! $keyval->getValue('load')) && (LookupValue('smultiling_langfiles'))) {
        $keyval->setValue('load',$signature); }
      # Always load a TeX file
      $keyval->setValue('ext','tex');
      AssignValue('modnl_signature',$signature);
      $keyval->setValue('id',"$signature.$language"); }
    module_afterDigestBegin(@_);
    importmoduleI(@_);
    AssignValue(multiling => 1);
    return; },
  afterDigest =>\&module_afterDigest );
⟨/ltxml⟩%$
```

viewnl The `viewnl` environment is just a layer over the `viewsketch` environment with the keys and langauge suitably adapted.[4]

EdN:4

```
⟨ltxml⟩RawTeX('
⟨*sty | ltxml⟩
\newenvironment{viewnl}[5][]{\def\@test{#1}\ifx\@test\@empty%
\begin{viewsketch}[id=#2.#3,ext=tex]{#4}{#5}\else%
\begin{viewsketch}[id=#2.#3,#1,ext=tex]{#4}{#5}\fi}
```

[4] EdNote: MK: we have to do something about the if@langfiles situation here. But this is non-trivial, since we do not know the current path, to which we could append .⟨*lang*⟩!

```
{\end{viewsketch}}
```

3.4 Multilingual Statements and Terms

\mtref* we first first define an auxiliary conditional `\@instring` that checks of `?` is in the first argument. `\mtrefi` uses it, if there is one, it just calls `\termref`, otherwise it calls `\@mtrefi`, which assembles the `\termref` after splitting at the `?`.

```
\def\@instring#1#2{TT\fi\begingroup\edef\x{\endgroup\noexpand\in@{#1}{#2}}\x\ifin@}
\newcommand\mtrefi[2][]{\if\@instring{?}{#1}\@mtref #1\relax{#2}\else\termref[cd=#1]{#2}\fi}
\def\@mtref#1?#2\relax{\termref[cd=#1,name=#2]}
\newcommand\mtrefis[2][]{\mtrefi[#1]{#2s}}
\newcommand\mtrefii[3][]{\mtrefi[#1]{#2 #3}}
\newcommand\mtrefiis[3][]{\mtrefi[#1]{#2 #3s}}
\newcommand\mtrefiii[4][]{\mtrefi[#1]{#2 #3 #4}}
\newcommand\mtrefiiis[4][]{\mtrefi[#1]{#2 #3 #4s}}
⟨/sty | ltxml⟩
⟨ltxml⟩');
```

3.5 Miscellaneneous

the `\ttl` macro (to-translate) is used to mark untranslated stuff. We need a better LaTeXMLtreatment of this eventually that is integrated with MathHub.info.

\ttl

```
⟨ltxml⟩RawTeX('
⟨*sty | ltxml⟩
\newcommand\ttl[1]{\red{TTL: #1}}
⟨/sty | ltxml⟩
⟨ltxml⟩');
```

3.6 Finale

Finally, we need to terminate the file with a success mark for perl.

```
⟨ltxml⟩1;
```