`smultiling.sty`: Multilinguality Support for sTeX

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

May 11, 2014

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

We have been using SᴛEX as the encoding for the Semantic Multilingual Glossary of Mathematics (SMGloM; see [Gin+14]). The SMGloM data model has been taxing the representational capabilities of SᴛEX with respect to multilingual support and verbalization definitions; see [Koh14], which we assume as background reading for this note.

1.1 SᴛEX Module Signatures

(monolingual) SᴛEX had the intuition that the symbol definitions (`\symdef` and `\symvariant`) are interspersed with the text and we generate SᴛEX module signatures (SMS `*.sms` files) from the SᴛEX files. The SMS duplicate "formal" information from the "narrative" SᴛEX 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 SᴛEX 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 SᴛEX

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

The `smultiling` package accepts all options of the `babel.sty` and just passes them on to it. The options specify which languages can be used in the SᴛEX language bindings.

2.1 Multilingual Modules

`modsig` There the `modsig` environment works exactly like the old `module` environment,

```
\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.

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 EdN:1 which serves as the reference in case of translation conflicts).[1]

`\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 [Koh14] 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.

For multilingual term references the situtation is more complex: For single-word verbalizations we could use `\atrefi` for language bindigns. Say we

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

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:2 `fromrepos`, `topath`, and `torepos`).[2]

```
\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:}
    ...
  \end{sproof}
```

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

```
\end{gviewnl}
```

3 Implementation

Technically, the `smultiling` package is essentially a wrapper around the `babel` package but allows specification of languages by their ISO 639 language codes.

3.1 Class Options

To initialize the `smultiling` class, we pass on all options to `babel.cls` and record which languages are loaded by defining `\smul@`⟨*language*⟩`@loaded` macros.[3]

EdN:3

`langfiles` The `langfiles` option specifies that for a module ⟨*mod*⟩, 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`.[4]

EdN:4

```
⟨*sty⟩
\newif\if@langfiles\@langfilesfalse
\DeclareOption{langfiles}{\@langfilestrue}
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{babel}
\@namedef{smul@\CurrentOption @loaded}{yes}}
\ProcessOptions
⟨/sty⟩
⟨*ltxml⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
DeclareOption('langfiles',sub {AssignValue('smultiling_langfiles',1,'global');});
DeclareOption(undef,sub {PassOptions('babel','sty',ToString(Digest(T_CS('\CurrentOption'))));
ProcessOptions();
⟨/ltxml⟩
```

We load `babel.sty`

```
⟨*sty⟩
\RequirePackage{etoolbox}
\RequirePackage{babel}
\RequirePackage{modules}
⟨/sty⟩
⟨*ltxml⟩
RequirePackage('babel');
RequirePackage('modules');
⟨/ltxml⟩
```

3.2 Handling Languages

`\smg@select@language` This macro selects one of the registered languages by its langauage code by setting the internal `\smg@lang` macro to the argument and then runs the actual selection

[3] EdNote: @DG: We also want to do that in LaTeXML

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

code in `\smg@select@lang`. This internal code register is only initialized there, the code is generated by the `\smg@register@language` macro below.

```
⟨ltxml⟩RawTeX('
⟨*sty | ltxml⟩
\newcommand\smg@select@lang{}
\newcommand\smg@select@language[1]{\def\smg@lang{#1}\smg@select@lang}
```

`\smg@register@language` `\smg@register@language{`⟨*lang*⟩`}{`⟨*babel*⟩`}` registers the `babel` language name ⟨*babel*⟩ with its ISO 639 languge code ⟨*lang*⟩ by extending the `\smg@select@language` macro.

```
\newcommand\smg@register@language[2]%
{\@ifundefined{smul@#1@loaded}{}{\appto\smg@select@lang%
{\expandafter\ifstrequal\expandafter\smg@lang{#1}{\selectlanguage{#2}}{}}}}
```

Now we register a couple of languages for which we have `babel` support. Maybe we have to extend this list with others. But then we have to extend the mechanisms.

```
\smg@register@language{af}{afrikaans}
\smg@register@language{de}{ngerman}
\smg@register@language{fr}{french}%
\smg@register@language{he}{hebrew}
\smg@register@language{hu}{hungarian}
\smg@register@language{id}{indonesian}
\smg@register@language{ms}{malay}
\smg@register@language{nn}{nynorsk}
\smg@register@language{pt}{portuguese}
\smg@register@language{ru}{russian}
\smg@register@language{uk}{ukrainian}
\smg@register@language{en}{english}
\smg@register@language{es}{spanish}
\smg@register@language{sq}{albanian}
\smg@register@language{bg}{bulgarian}
\smg@register@language{ca}{catalan}
\smg@register@language{hr}{croatian}
\smg@register@language{cs}{czech}
\smg@register@language{da}{danish}
\smg@register@language{nl}{dutch}
\smg@register@language{eo}{esperanto}
\smg@register@language{et}{estonian}
\smg@register@language{fi}{finnish}
\smg@register@language{ka}{georgian}
\smg@register@language{el}{greek}
\smg@register@language{is}{icelandic}
\smg@register@language{it}{italian}
\smg@register@language{la}{latin}
\smg@register@language{no}{norsk}
\smg@register@language{pl}{polish}
\smg@register@language{sr}{serbian}
\smg@register@language{sk}{slovak}
\smg@register@language{sl}{slovenian}
```

```
\smg@register@language{sv}{swedish}
\smg@register@language{th}{thai}
\smg@register@language{tr}{turkish}
\smg@register@language{vi}{vietnamese}
\smg@register@language{cy}{welsh}
\smg@register@language{hi}{hindi}
```

3.3 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.

```
\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}}
```

`mhviewsig` The `mhviewsig` environment is just a layer over the `mhview` environment with the keys suitably adapted.

```
\newenvironment{mhviewsig}[4][]{\def\@test{#1}\ifx\@test\@empty%
\begin{mhview}[id=#2,ext=tex]{#3}{#4}\else\begin{mhview}[id=#2,#1,ext=tex]{#3}{#4}\fi}
{\end{mhview}}
⟨*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⟩
```

3.4 Language Bindings

modnl:*

```
⟨*sty⟩
\addmetakey{modnl}{load}
\addmetakey*{modnl}{title}
\addmetakey*{modnl}{creators}
\addmetakey*{modnl}{contributors}
\addmetakey{primary}{contributors}[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}%
\smg@select@language{#3}%
\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));
    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
      $keyval->setValue('ext','tex');
      module_afterDigestBegin(@_);
      importmoduleI(@_); }
    return; },
  afterDigest =>\&module_afterDigest );
⟨/ltxml⟩%$
```

mhmodnl:*

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

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

```
⟨*sty⟩
\newenvironment{mhmodnl}[3][]{\metasetkeys{mhmodnl}{#1}%
\smg@select@language{#3}%
\def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2.#3]\else\begin{module}[id=#2.#3,#1]\fi%
\edef\@repos{\ifx\mhmodnl@repos\@empty\mh@currentrepos\else\mhmodnl@repos}
\if@langfiles\importmhmodule[repos=\@repos,load=#2,ext=tex]{#2}\else
\ifx\mhmodnl@load\@empty\importmodule{#2}\else\importmodule[ext=tex,load=\mhmodnl@load]{#2}\fi%
\fi}
{\end{module}}
⟨/sty⟩
⟨*ltxml⟩
DefEnvironment('{mhmodnl} OptionalKeyVals:mhmodnl {}{}',
         '?#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 $repos = ToString(GetKeyVal($keyval,'torepos'));
    my $current_repos = LookupValue('current_repos');
    if (!$repos) { $repos = $current_repos; }
    my $defpaths = LookupValue('defpath');
    my $load_path = ($$defpaths{MathHub}).$repos.'/source/'.$signature;

    if ($keyval) {
      # If we're not given load, AND the langfiles option is in effect,
      # default to #2
      if ((! $keyval->getValue('path')) && (LookupValue('smultiling_langfiles'))) {
        $keyval->setValue('load',$load_path); }
      # Always load a TeX file
      $keyval->setValue('ext','tex'); }
      module_afterDigestBegin(@_);
      importmoduleI(@_);
    return; },
  afterDigest=>sub {
    module_afterDigest(@_); });
⟨/ltxml⟩%$
```

`viewnl` The `viewnl` environment is just a layer over the `viewsketch` environment with the keys and langauge suitably adapted.[5]

EdN:5

```
⟨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%
\smg@select@language{#3}}
{\end{viewsketch}}
```

`mhviewnl` The `mhviewnl` environment is just a layer over the `mhviewsketch` environment with the keys and langauge suitably adapted.[6]

EdN:6

```
\newenvironment{mhviewnl}[5][]{\def\@test{#1}\ifx\@test\@empty%
\begin{mhviewsketch}[id=#2.#3,ext=tex]{#4}{#5}\else%
\begin{mhviewsketch}[id=#2.#3,#1,ext=tex]{#4}{#5}\fi%
\smg@select@language{#3}}
{\end{mhviewsketch}}
⟨/sty | ltxml⟩
⟨ltxml⟩');
```

[5] 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*⟩!

[6] 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*⟩!

3.5 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 ?.

```
⟨ltxml⟩RawTeX('
⟨*package | ltxml⟩
\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\mtrefii[3][]{\mtrefi[#1]{#2 #3}}
\newcommand\mtrefiii[4][]{\mtrefi[#1]{#2 #3 #4}}
⟨/package | ltxml⟩
⟨ltxml⟩');
```

3.6 Finale

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

```
⟨ltxml⟩1;
```

References

[Gin+14] Deyan Ginev et al. "The SMGLoM Project and System". 2014. URL: http://kwarc.info/kohlhase/submit/cicm14-smglom-system.pdf.

[Koh14] Michael Kohlhase. "A Data Model and Encoding for a Semantic, Multilingual Glossary of Mathematics". In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics. (Coimbra, Portugal, July 7–11, 2014). Ed. by Stephan Watt et al. Lecture Notes in Computer Science. accepted. Springer, 2014. URL: http://kwarc.info/kohlhase/submit/cicm14-smglom-datamdl.pdf. Forthcoming.