`smultiling.sty`: Multilingual Support for sTeX

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

May 7, 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 sTeX 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 sTeX with respect to multilingual support and verbalization definitions; see [Koh14], 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

```
\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}
```

we now advocate the divided style in the listing below.

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

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 sTeX language bindings.

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).[1] EdN:1

`\symbol` `\symbol*` There is another difference in the multilingual encoding: All symbols are introduced in the module signature, either by a `\symdef` or the new `\symbol` macro. `\symbol{`⟨*name*⟩`}` takes a symbol name ⟨*name*⟩ as an argument and reserves that name. The variant `\symbol*{`⟨*name*⟩`}` declares ⟨*name*⟩ to be a primary symbol; see [Koh14] for a discussion.

2.2 Multilingual Views

`viewsig` Views receive a similar treatment as modules in the `smultiling` package. A 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`, `fromrepos`, `topath`, and `torepos`).[2] EdN:2

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

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

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

```
  \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}
\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

EdN:3 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]

`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 {}{}',
         "<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
         .   "#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
    $keyval->setValue('ext','tex'); }
  importmoduleI($stomach,$whatsit)});
⟨/ltxml⟩%$
```

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 {}{}',
        "<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
        .   "#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
      $keyval->setValue('ext','tex'); }
    importmoduleI($stomach,$whatsit)});
⟨/ltxml⟩%$
```

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*⟩!

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.

¡ltxml¿1;