`smultiling.sty`: Multilinguality Support for sTeX

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

September 28, 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

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 $\langle mod \rangle$ that the module signature file has the name $\langle mod \rangle$.tex and the language bindings of language with the ISO 639 language specifier $\langle lang \rangle$ have the EdN:1 file name $\langle mod \rangle.\langle lang \rangle$.tex.[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]

\symi   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
\symi*   name. The variant `\symi*{⟨name⟩}` declares ⟨name⟩ to be a primary symbol;
\symii   see [**Kohlhase:dmesmgm14**] for a discussion. sTEX provides variants `\symii`
\symiii   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.

EdN:2

---

[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, 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.

`\mtref*`

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 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`).[3]

`viewsig`

EdN: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

3.1   Class Options

```
 1 ⟨∗sty⟩
 2 \newif\if@langfiles\@langfilesfalse
 3 \DeclareOption{langfiles}{\@langfilestrue}
 4 \ProcessOptions
 5 ⟨/sty⟩
 6 ⟨∗ltxml⟩
 7 # -*- CPERL -*-
 8 package LaTeXML::Package::Pool;
 9 use strict;
10 use LaTeXML::Package;
11 DeclareOption('langfiles',sub {AssignValue('smultiling_langfiles',1,'global');});
12 ProcessOptions();
13 ⟨/ltxml⟩
```

We load the packages referenced here.

```
14 ⟨∗sty⟩
15 \RequirePackage{etoolbox}
16 \RequirePackage{modules}
17 ⟨/sty⟩
18 ⟨∗ltxml⟩
19 RequirePackage('modules');
20 ⟨/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.

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

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

```
26 \newenvironment{viewsig}[4][]{\def\@test{#1}\ifx\@test\@empty%
27 \begin{view}[id=#2,ext=tex]{#3}{#4}\else\begin{view}[id=#2,#1,ext=tex]{#3}{#4}\fi}
28 {\end{view}}
```

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

```
29 \newenvironment{mhviewsig}[4][]{\def\@test{#1}\ifx\@test\@empty%
30 \begin{mhview}[id=#2,ext=tex]{#3}{#4}\else\begin{mhview}[id=#2,#1,ext=tex]{#3}{#4}\fi}
31 {\end{mhview}}
32 ⟨∗sty | ltxml⟩
33 ⟨ltxml⟩');
```

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

```
34 ⟨∗sty⟩
35 \newcommand\symi{\@ifstar\@symi@star\@symi}
36 \newcommand\@symi[1]{\if@importing\else Symbol: \textsf{#1}\fi}
37 \newcommand\@symi@star[1]{\if@importing\else Primary Symbol: \textsf{#1}\fi}
38 \newcommand\symii{\@ifstar\@symii@star\@symii}
39 \newcommand\@symii[2]{\if@importing\else Symbol: \textsf{#1-#2}\fi}
40 \newcommand\@symii@star[2]{\if@importing\else Primary Symbol: \textsf{#1-#2}\fi}
41 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
42 \newcommand\@symiii[3]{\if@importing\else Symbol: \textsf{#1-#2-#3}\fi}
43 \newcommand\@symiii@star[3]{\if@importing\else Primary Symbol: \textsf{#1-#2-#3}\fi}
44 ⟨/sty⟩
45 ⟨∗ltxml⟩
46 DefConstructor('\symi OptionalMatch:* {}',
47     "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2'/>");
48 DefConstructor('\symii OptionalMatch:* {} {}',
49     "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2-#3'/>");
50 DefConstructor('\symiii OptionalMatch:* {} {} {}',
51     "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2-#3-#4'/>");
52 ⟨/ltxml⟩
```

3.3   Language Bindings

modnl:*

```
53 ⟨∗sty⟩
54 \addmetakey{modnl}{load}
55 \addmetakey*{modnl}{title}
56 \addmetakey*{modnl}{creators}
57 \addmetakey*{modnl}{contributors}
58 \addmetakey{modnl}{srccite}
59 \addmetakey{modnl}{primary}[yes]
60 ⟨/sty⟩
61 ⟨∗ltxml⟩
62 DefKeyVal('modnl','title','Semiverbatim');
63 DefKeyVal('modnl','load','Semiverbatim');
64 DefKeyVal('modnl','creators','Semiverbatim');
65 DefKeyVal('modnl','contributors','Semiverbatim');
66 DefKeyVal('modnl','primary','Semiverbatim');
67 ⟨/ltxml⟩
```

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

```
68 ⟨∗sty⟩
69 \newenvironment{modnl}[3][]{\metasetkeys{modnl}{#1}%
70 \def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2.#3]\else\begin{module}[id=#2.#3,#1]\fi%
71 \if@langfiles\importmodule[load=#2,ext=tex]{#2}\else
72 \ifx\modnl@load\@empty\importmodule{#2}\else\importmodule[ext=tex,load=\modnl@load]{#2}\fi%
73 \fi}
74 {\end{module}}
```

7

```
75 ⟨/sty⟩
76 ⟨*ltxml⟩
77 DefEnvironment('{modnl} OptionalKeyVals:modnl {}{}',
78         "?#excluded()(<omdoc:theory xml:id='#2.#3'>"
79       . "?&defined(&GetKeyVal(#1,'creators'))(<dc:creator>&GetKeyVal(#1,'creators')</dc:cr
80       . "?&defined(&GetKeyVal(#1,'title'))(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
81       . "?&defined(&GetKeyVal(#1,'contributors'))(<dc:contributor>&GetKeyVal(#1,'contribut
82       . "<omdoc:imports
83         from='?&GetKeyVal(#1,'load')(&canonical_omdoc_path(&GetKeyVal(#1,'load')))()#\e
84         #2'/>"
85       . "#body"
86       . "</omdoc:theory>)",
87    afterDigestBegin=>sub {
88      my ($stomach, $whatsit) = @_;
89      my $keyval = $whatsit->getArg(1);
90      my $signature = ToString($whatsit->getArg(2));
91      my $language = ToString($whatsit->getArg(3));
92      if ($keyval) {
93        # If we're not given load, AND the langfiles option is in effect,
94        # default to #2
95        if ((! $keyval->getValue('load')) && (LookupValue('smultiling_langfiles'))) {
96          $keyval->setValue('load',$signature); }
97        # Always load a TeX file
98        $keyval->setValue('ext','tex');
99        AssignValue('modnl_signature',$signature);
100       $keyval->setValue('id',"$signature.$language"); }
101     module_afterDigestBegin(@_);
102     importmoduleI(@_);
103     AssignValue(multiling => 1);
104     return; },
105   afterDigest =>\&module_afterDigest );
106 ⟨/ltxml⟩%$
```

mhmodnl:*

```
107 ⟨*sty⟩
108 \addmetakey{mhmodnl}{repos}
109 \addmetakey{mhmodnl}{path}
110 \addmetakey*{mhmodnl}{title}
111 \addmetakey*{mhmodnl}{creators}
112 \addmetakey*{mhmodnl}{contributors}
113 \addmetakey{mhmodnl}{srccite}
114 \addmetakey{primary}{mhmodnl}[yes]
115 ⟨/sty⟩
116 ⟨*ltxml⟩
117 DefKeyVal('mhmodnl','title','Semiverbatim');
118 DefKeyVal('mhmodnl','repos','Semiverbatim');
119 DefKeyVal('mhmodnl','path','Semiverbatim');
120 DefKeyVal('mhmodnl','creators','Semiverbatim');
121 DefKeyVal('mhmodnl','contributors','Semiverbatim');
122 DefKeyVal('mhmodnl','primary','Semiverbatim');
```

8

123 ⟨/ltxml⟩

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

```
124 ⟨*sty⟩
125 \newenvironment{mhmodnl}[3][]{\metasetkeys{mhmodnl}{#1}%
126 \def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2.#3]\else\begin{module}[id=#2.#3,#1]\fi%
127 \edef\@repos{\ifx\mhmodnl@repos\@empty\mh@currentrepos\else\mhmodnl@repos}
128 \if@langfiles\importmhmodule[repos=\@repos,load=#2,ext=tex]{#2}\else
129 \ifx\mhmodnl@load\@empty\importmodule{#2}\else\importmodule[ext=tex,load=\mhmodnl@load]{#2}\fi%
130 \fi}
131 {\end{module}}
132 ⟨/sty⟩
133 ⟨*ltxml⟩
134 DefEnvironment('{mhmodnl} OptionalKeyVals:mhmodnl {}{}',
135       "?#excluded()(<omdoc:theory xml:id='#2.#3' >"
136     . "?&defined(&GetKeyVal(#1,'creators'))(<dc:creator>&GetKeyVal(#1,'creators')</dc:cr
137     . "?&defined(&GetKeyVal(#1,'title'))(<dc:title>&GetKeyVal(#1,'title')</dc:title>)()"
138     . "?&defined(&GetKeyVal(#1,'contributors'))(<dc:contributor>&GetKeyVal(#1,'contribut
139     . "<omdoc:imports
140     from='?&GetKeyVal(#1,'load')(&canonical_omdoc_path(&GetKeyVal(#1,'load')))()#\e
141     #2'/>"
142     . "#body"
143     . "</omdoc:theory>)",
144  afterDigestBegin=>sub {
145    my ($stomach, $whatsit) = @_;
146    my $keyval = $whatsit->getArg(1);
147    my $signature = ToString($whatsit->getArg(2));
148    my $language = ToString($whatsit->getArg(3));
149    my $repos = ToString(GetKeyVal($keyval,'torepos'));
150    my $current_repos = LookupValue('current_repos');
151    if (!$repos) { $repos = $current_repos; }
152    my $defpaths = LookupValue('defpath');
153    my $load_path = ($$defpaths{MathHub}).$repos.'/source/'.$signature;
154
155    if ($keyval) {
156      # If we're not given load, AND the langfiles option is in effect,
157      # default to #2
158      if ((! $keyval->getValue('path')) && (LookupValue('smultiling_langfiles'))) {
159        $keyval->setValue('load',$load_path); }
160      # Always load a TeX file
161      $keyval->setValue('ext','tex');
162      $keyval->setValue('id',"$signature.$language"); }
163    module_afterDigestBegin(@_);
164    importmoduleI(@_);
165    return; },
166  afterDigest=>sub {
167    module_afterDigest(@_); });
168 ⟨/ltxml⟩%$
```

9

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

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

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

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

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 `?`.

```
181 ⟨ltxml⟩RawTeX('
182 ⟨*sty | ltxml⟩
183 \def\@instring#1#2{TT\fi\begingroup\edef\x{\endgroup\noexpand\in@{#1}{#2}}\x\ifin@}
184 \newcommand\mtrefi[2][]{\if\@instring{?}{#1}\@mtref #1\relax{#2}\else\termref[cd=#1]{#2}\fi}
185 \def\@mtref#1?#2\relax{\termref[cd=#1,name=#2]}
186 \newcommand\mtrefii[3][]{\mtrefi[#1]{#2 #3}}
187 \newcommand\mtrefiii[4][]{\mtrefi[#1]{#2 #3 #4}}
188 ⟨/sty | ltxml⟩
189 ⟨ltxml⟩');
```

3.5 Finale

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

```
190 ⟨ltxml⟩1;
```

3.6 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.

---

[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⟩!

[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⟩!

`\ttl`

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