OpenMath Language Extensions

Michael Kohlhase

Computer Science, Jacobs University
http://kwarc.info/kohlhase

Abstract

We propose a language extension mechanism for OpenMath that will allow us to introduce new language features (as have been called for by the community) while keeping the underlying core language intact so that we can maintain compatibility with MathML3 and the upcoming ISO standard. We exhibit the mechanism on the example of extending OpenMath with sequences and discuss other language extensions.

1 Introduction

The OpenMath society has decided to task a committee to assess the viability of extending the OpenMath 2 [Bus+04] standard and bring in all the extension proposals out there. There are a variety of proposals that involve extending the set of OpenMath objects, e.g. to introduce sequences in [HKR11]. However, the recent synchronization of OpenMath and content MathML in the MathML 3 recommendation [Aus+10] and its imminent ISO standardization restrict the possible extensions severely.

We propose a way to have our cake (minimize extensions to the OpenMath objects) and eat it (get native syntax for often-used mathematical representational idioms). Instead of extending the OpenMath language with a limited set of primitives we propose to extend OpenMath by a syntax extension mechanism much like OpenMath CDs constitute an extension mechanism for the mathematical vocabularies.

This proposal is similar in spirit to the re-interpretation of the content MathML format into a structurally regular core language ("strict content MathML") which is interpreted as OpenMath objects and a more extensive language ("full content MathML") whose semantics is given by translation into strict content MathML.

In our proposal, OpenMath 2 is retained as a strict core language, but various "pragmatic" language extensions can be introduced via OpenMath **language extension dictionaries** (LED). The OpenMath society adopts and maintains a set of "standard LDs" alongside with the standard CDs. The OpenMath 3 format is then induced by the OpenMath standard LEDs from the strict core given by the OpenMath 2 language. Other entities can standardize other extensions in LED format – e.g. the Math working group at W3C could provide LEDs for full content MathML.

2 Language Dictionaries

Language Dictionaries have three functions: to

1. extend the syntax admissible in the extended language,
2. specify the meaning of the language extensions, and
3. specify when two objects of the extended language are equal.

As the OpenMath and MathML specifications give a RelaxNG schema [CM01] for the syntax, we do the same in the LED. Following the MathML 3 lead we give the semantics of the "full

language" by translation into the strict core. Note that the first two functions are addressed at the user who wants to write meaningful expressions in the extended format. The third function is only addressed to phrasebook implementors of the extended objects.

3 An Example Language Extension: Argument Sequences

We now fortify our intuition with a small LED fragment that introduces two extension elements from [HKR11]. Note that these have been chosen for expository reasons only, in particular, they do not constitute a concrete extension proposal, that would be based on [CICM1414].

3.1 Example A Language Extension Dictionary for Sequences

Listing 1: A LED for sequences

```
<OMLangExt xmlns="http://www.openmath.org/OpenMathCD">
 <OLEName>seq</OLEName>
 <OLEDate>2014−04−22</OLEDate>
 <OLEStatus>experimental</OLEStatus>
 <OLEVersion>1</OLEVersion>
 <OLERevision>1</OLERevision>
  <schemaext>
    OMNATS = element OMNATS {omel}
    OMNTH = element OMNTH {omel,omel}
    omel |= OMNTH | OMNATS
  </schemaext>
  <equality>
      <OMBIND>
        <OMS cd="quant1" name="forall"/>
        <OMBVAR><OMV name="n"/><OMV name="m"/></OMBVAR>
        <OMA>
          <OMS cd="logic1" name="implies"/>
          <OMA><OMS cd="relation1" name="gt"/><OMV name="m"/><OMV name="n"/></OMA>
          <OMA>
            <OMS cd="relation1" name="eq"/>
            <OMNTH>
              <OMI>n</OMI>
              <OMNATS><OMI>m</OMI></OMNATS>
            </OMNTH>
            <OMI>n</OMI>
          </OMA>
        </OMA>
      </OMBIND>
  </equality>
  <translation cd="argseq">
    <rule>
      <OMNTH>
        <expr name="n"/>
        <exprlist name="seq">
```

```
        <expr name="elt"/>
      </exprlist>
    </OMNTH>
    <OMA>
      <OMS cd="seqs" name="nth"/>
      <render name="n"/>
      <iterate name="seq">
        <render name="elt"/>
      </iterate>
    </OMA>
  </rule>
  <rule>
    <OMNATS><expr name="n"/></OMNATS>
    <OMA><OMS cd="seqs" name="nats"/><render name="n"/></OMA>
  </rule>
 </translation>
</OMLangExt>
```

The OMLangExt[1] element is the top-level element of language definitions. It contains metadata that is isomorphic to that of OpenMath CDS and three children for the three functions mentioned above.

3.1.1 The Schema Extension

The schemaext contains contains RelaxNG rules in compact form that extend the OpenMath 2 schema. In our example, the language of OpenMath objects is extended by two new constructs:

- the sequence constructor OMNATS that (given a natural number n) represents the sequence of the first n natural numbers starting at zero.
- the sequence selector OMNTH takes an OpenMath object representing a natural number n and a sequence S as arguments and represents the n^{th} element of S (if it exists).

Correspondingly, the schemaExt element contains a rule for both of the elements, an extension of the OpenMath expressions by OMNTH elements and a new syntactic category omseq that contains OMNATS elements. From the LED in Listing 1, we can generate RelaxNG schema for the extended language in Figure 1.

We probably need to have an extension for OMCDs as well. Maybe we can use a dynamic extension mechanism, where an OMOBJ specifies what extensions it uses in a special attribute, which lists the extensions it uses. If we have a list of LEDs supported by OM, then we can probably compile such a RelaxNG schema from them.

```
input "openmath2.rnc"
OMNATS = element OMNATS {omel}
OMNTH = element OMNTH {omel,omseq}
omseq = OMNATS
omel |= OMNTH
```

Figure 1: The RelaxNG Schema omobj+seq.

[1] Note that the names of LED elements have been chosen more or less randomly and should be carefully re-considered before this turns note into a standards extension proposal.

3.1.2 Equality

The equality element contains equality rules that say when two elements in the extended language are equal expressed as OpenMath elements. Here, we have the relation:

$$\forall n.n > m \Rightarrow \mathrm{nth}(\mathrm{nats}(n)) = n \tag{1}$$

3.1.3 Translation to OpenMath 2

And finally, the translation element contains rules that allow to translate objects from the extended language into core OpenMath 2 objects.

We propose the OMDoc presentation rewriting for translation since it is relatively restricted and declarative here. But arguably having XSLT for translation would be more standards-conformant and more powerful – which might or might not be a good thing. In principle any translation mechanism would work, the concrete choice of a mechanism is an open design choice. The XSLT for our example is in Figure 3.

The translation element gives objects in the extended language their meaning by translation into strict OpenMath 2 (with a special content dictionary argseq specified in the cd attribute of the translation element[2]; see Listing 2).

```
<OMNTH>
  <OMA>
    <OMS cd="arith1" name="plus"/>
    <OMI>1</OMI>
    <OMI>2</OMI>
  </OMA>
  <OMNATS><OMI>7</OMI></OMNATS>
</OMNTH>
```

(a) A OpenMath object with Sequences

```
<OMA>
  <OMS cd="argseq" name="nth"/>
  <OMA>
    <OMS cd="arith1" name="plus"/>
    <OMI>1</OMI>
    <OMI>2</OMI>
  </OMA>
  <OMA>
    <OMA>
      <OMS cd="argseq" name="omnats"/>
      <OMI>7</OMI>
    </OMA>
</OMA>
```

(b) Its Translation

Figure 2: Translating Extended OpenMath Expressions

With the LED from Listing 1 we can express OpenMath objects like the one in Figure 2a with the new OpenMath elements licensed by the language definition: The sequence selector OMNTH expects an openmath object (denoting a natural number) as the first child and a sequence as the second. The latter is represented by the OMNATS element that represents the sequence of the first seven natural numbers.

3.2 The Corresponding CD

The translation in the LED above uses special symbols from the argseq CD, which we give in Listing 2. The CD introduces two symbols nth and nats. Note that a full development of

[2]We probably need to allow multiple CDs here, since we may need combinations. Unfortunately, we do not have inheritance in CDs, so we need more. But on the other hand, the whole attribute is unnecessary, since we can just pick up the necessary CDs from the translation objects themselves.

```
<xsl:template match="om:OMNTH">
  <om:OMA>
    <OMS cd="argseq" name="nth"/>
    <xsl:apply-templates/>
  </om:OMA>
</xsl:template>
<xsl:template match="om:OMNATS">
  <om:OMA>
    <OMS cd="argseq" name="nth"/>
    <xsl:apply-templates/>
  </om:OMA>
</xsl:template>
```

Figure 3: An XSLT alternative to pattern-based translation

argument sequences would contain more symbols and relations.

The CMP and FMP elements state mathematical properties that specify the relations between the symbols, here the strict OpenMath variant of 1[3]

Listing 2: The CD for the LED in 1

```
<CD xmlns="http://www.openmath.org/OpenMathCD">
  <CDComment>
    This CD contains a (part of a) specification of argument sequences
    for an OpenMath language extension.
  </CDComment>
  <CDName>argseq</CDName>
  <CDBase>http://www.openmath.org/cd</CDBase>
  <CDURL>http://www.openmath.org/cd/argseq.ocd </CDURL>
  <CDReviewDate>2014-03-01</CDReviewDate>
  <CDStatus>experimental</CDStatus>
  <CDDate>2013-10-01</CDDate>
  <CDVersion>0</CDVersion>
  <CDRevision>1</CDRevision>

  <Description>
    This CD defines argument sequences for the use in an OpenMaht3
    language extension.
  </Description>
  <CDDefinition>
    <Name>nth</Name>
    <Role>application</Role>
    <Description>
      This symbol represents the sequence selector. The first argument
      is a natural number n and the second an argument sequence S.
      The argument selector returns the n-th element in S, if it exits.
```

[3]Here it would be good to have an id attribute on the FMP, so that we can reference it in the LED as a justification.

```
    </Description>
  </CDDefinition>
  <CDDefinition>
    <Name>nats</Name>
    <Role>application</Role>
    <Description>
      Given a natural number n, this symbol returns the sequence of the
      first n natural numbers (starting at zero).
    </Description>
  <CMP>
    For any natural numbers n and m with m>n, the n−th component of nats(m) is n
  </CMP>
  <FMP>
    <OMOBJ xmlns="http://www.openmath.org/OpenMath">
      <OMBIND>
        <OMS cd="quant1" name="forall"/>
        <OMBVAR><OMV name="n"/><OMV name="m"/></OMBVAR>
        <OMA>
          <OMS cd="logic1" name="implies"/>
            <OMA><OMS cd="relation1" name="gt"/><OMV name="m"/><OMV name="n"/></OMA>
            <OMA>
              <OMS cd="relation1" name="eq"/>
              <OMA>
                <OMS cd="argseq" name="nth"/>
                <OMA><OMS cd="argseq" name="nats"/><OMV name="n"/></OMA>
              </OMA>
            </OMA>
            <OMV name="n"/>
        </OMA>
      </OMBIND>
    </OMOBJ>
  </FMP>
  </CDDefinition>
</CD>
```

4 Conclusion

We have sketched an extension mechanism for OpenMath that allows to interpret pragmatic extensions in terms of a core language using specific CDs. That would allow use to obtain OpenMath 3 as a collection of language extensions while retaining OpenMath 2 as a strict core language – which is important, since it is referenced as a semantic basis of MathML 3 [Aus+10]. The language extension mechanism might be interesting as a tool for formally recast the full MathML 3 language as a collection of extensions to strict content MathML. This would be a nice test case for the extension mechanisms expressivity.

References

[Aus+10] Ron Ausbrooks et al. *Mathematical Markup Language (MathML) Version 3.0*. W3C Recommendation. World Wide Web Consortium (W3C), 2010. URL: http://www.w3.org/TR/MathML3.

[Bus+04] Stephen Buswell et al. *The Open Math Standard, Version 2.0*. Tech. rep. The OpenMath Society, 2004. URL: http://www.openmath.org/standard/om20.

[CICM1414] Fulya Horozal, Michael Kohlhase, and Florian Rabe. "Flexary Operators for Formalized Mathematics". In: *Intelligent Computer Mathematics 2014*. Ed. by Stephan Watt et al. Lecture Notes in Computer Science. accepted. Springer, 2014. URL: https://svn.kwarc.info/repos/fhorozal/papers/submitted/cicm14_flex-op.pdf. Forthcoming.

[CM01] James Clark and Makoto Murata. *RELAX NG Specification*. Tech. rep. OASIS, Dec. 3, 2001. URL: http://www.relaxng.org/spec-20011203.html.

[HKR11] Fulya Horozal, Michael Kohlhase, and Florian Rabe. "Extending OpenMath with Sequences". In: ed. by A. Asperti et al. Vol. UBLCS-2011-04. Technical Reports of University of Bologna. University of Bologna, 2011, pp. 58–72. URL: http://kwarc.eecs.iu-bremen.de/frabe/Research/HKR_sequences_11.pdf.

A A RelaxNG Schema for LEDs

Here we have a RelaxNG schema for LEDs, note that OLEs are self-referential in the sense that they already contain the language extensions they introduce, so we have to include a subschema ext.rnc that is the contents oft he schemaext element. But with this little trick, we can validate quite nicely.

Listing 3: The CD for the LED in 1

```
# ******************************************
#
# Relax NG Schema for OpenMath Language Extensions
#
# ******************************************

default namespace = "http://www.openmath.org/OpenMathCD"

## we include an encapsulated version of the OpenMath Schema
omelorig = grammar {include "openmath2.rnc" {start=omel}}
## and one we extend with the generated extension (extract from the LED in question)
omelext = grammar{include "openmath2.rnc" {start=omel} include "ext.rnc"}

## metadata analogous to the one of CDs.
OLEComment = element OLEComment { text }
OLEName = element OLEName { xsd:NCName }
OLEUses = element OLEUses { OLEName* }
OLEURL = element OLEURL { xsd:anyURI }
OLEBase = element OLEBase { xsd:anyURI }
OLEReviewDate = element OLEReviewDate { xsd:date }
```

```
OLEDate = element OLEDate { xsd:date }
OLEVersion = element OLEVersion { xsd:nonNegativeInteger }
OLERevision = element OLERevision { xsd:nonNegativeInteger }
OLEStatus = element OLEStatus { "official" | "experimental" | "private" | "obsolete"}

Description = element Description { text }

## the top−level element
start = OMLangExt
 OMLangExt =
  element OMLangExt {
    (OLEComment* & Description? &
     OLEName & OLEURL? & OLEBase? &
     OLEReviewDate? & OLEDate & OLEStatus &
     OLEUses? &
     OLEVersion & OLERevision),
     schemaext, equality, translation}

name.attrib = attribute name {xsd:NCName}?
exprlist.attribs = name.attrib
exprlist.model = headexp*
exprlist = element exprlist {exprlist.attribs & exprlist.model}

expr.attribs = name.attrib
expr.model = empty
expr = element expr {expr.attribs & expr.model}

head.class = exprlist | expr
headexp = grammar {include "openmath2.rnc" {start = omel}
                                    include "ext.rnc"
                                    omel |= parent head.class
                                    omvar |= parent head.class}

## and now the LED−specific elements
schemaext = element schemaext {text}
equality = element equality {omelext+}
translation = element translation {attribute cd {text},rule+}
rule = element rule {headexp,omelorig}
```