Extending MKM Formats at the Statement Level

Fulya Horozal, Michael Kohlhase and Florian Rabe

Jacobs University Bremen

CICM, 8.-13. July 2012
Bremen, Germany

Designing MKM Formats

A design challenge

“Ease of modeling” vs “ease of implementation”

Designing MKM Formats

A design challenge

"Ease of modeling" vs "ease of implementation"

Mathematics style

<**theorem** name="foo">
$1 + 1 \doteq 2$
</**theorem**>

<**proof** for="foo">
proof-term
</**proof**>

Curry-Howard style

<**constant** name="foo">
<**type**>
$1 + 1 \doteq 2$
</**type**>
<**definition**>
proof-term
</**definition**>
</**constant**>

Designing MKM Formats

A design challenge

"Ease of modeling" vs "ease of implementation"

Mathematics style

```
<theorem name="foo">
  1 + 1 ≐ 2
</theorem>

<proof for="foo">
  proof-term
</proof>
```

Curry-Howard style

```
<constant name="foo">
  <type>
    1 + 1 ≐ 2
  </type>
  <definition>
    proof-term
  </definition>
</constant>
```

Standard solution: Extensibility

Minimal core + extension principles

Classification of Extension Principles

What does the extension principle introduce?

- new object (typically identifier) — e.g., $2 := succ(succ(zero))$
- new statement (typically with keyword) — e.g., locales in Isabelle

Who defines the extension principle?

- the user — e.g., $2 := succ(succ(zero))$
- the programmer — e.g., locales in Isabelle

How is the extension principle interpreted?

- unconstrained — interpretation at runtime
- constrained — well-formedness judgments

Some Extensible MKM Formats

MKM Formats	Extensions: Object Level	Extensions: Statement Level
LaTeX (narrative)	user	user
	unconstrained	unconstrained
Isabelle/HOL (formal math)	user	programmer
	(un)constrained	unconstrained
OMDoc 1.2 (content markup)	user	programmer
	constrained	unconstrained

Some Extensible MKM Formats

MKM Formats	Extensions: Object Level	Extensions: Statement Level
LaTeX (narrative)	user	user
	unconstrained	unconstrained
Isabelle/HOL (formal math)	user	programmer
	(un)constrained	unconstrained
OMDoc 1.2 (content markup)	user	programmer
	constrained	unconstrained

LaTeX

- Object level: e.g., `\newcommand{\mycommand}{...}`
- Statement level: e.g., `\newenvironment{\myenv}{...}{...}`

Some Extensible MKM Formats

MKM Formats	Extensions: Object Level	Extensions: Statement Level
LaTeX (narrative)	user	user
	unconstrained	unconstrained
Isabelle/HOL (formal math)	user	programmer
	(un)constrained	unconstrained
OMDoc 1.2 (content markup)	user	programmer
	constrained	unconstrained

Isabelle/HOL

- Object level: definitions, declarations, etc.
- Statement level: locales, type defns, case-based function defns

Some Extensible MKM Formats

MKM Formats	Extensions: Object Level	Extensions: Statement Level
LaTeX (narrative)	user	user
	unconstrained	unconstrained
Isabelle/HOL (formal math)	user	programmer
	(un)constrained	unconstrained
OMDoc 1.2 (content markup)	user	programmer
	constrained	unconstrained

OMDoc 1.2

- Object level: symbol declarations
- Statement level: theorems, definitions, proofs, etc.

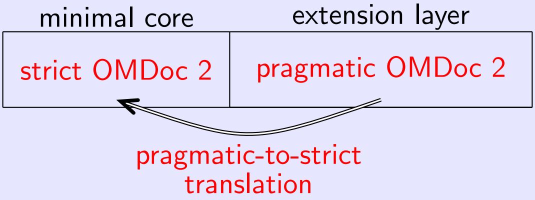
Our Approach: Generic Framework for Extension Principles

MKM Formats	Extensions: Object Level	Extensions: Statement Level
LaTeX (narrative)	user unconstrained	user unconstrained
Isabelle/HOL (formal math)	user (un)constrained	programmer unconstrained
OMDoc 1.2 (content markup)	user constrained	programmer unconstrained
Our approach	user constrained	user constrained

Our Approach: Generic Framework for Extension Principles

	Extensions: Object Level	Extensions: Statement Level
OMDoc 1.2 (content markup)	user	programmer
	constrained	unconstrained
OMDoc 2 (content markup)	user	user
	constrained	constrained

Redesign the OMDoc format



Motivating Example

pragmatic surface syntax

theorem	$1+1 \doteq 2$	(foo)
proof	*proof-term*	

↓ notation parser

pragmatic OMDoc 2 abstract syntax

$$foo \quad : \quad theorem \; 1+1 \doteq 2 \; (proof\text{-}term)$$

↓ pragmatic-to-strict

strict OMDoc 2 abstract syntax

$$foo \quad : \quad 1+1 \doteq 2 \quad = \quad (proof\text{-}term)$$

Our Core Language (strict OMDoc 2 = MMT)

- A module system for mathematical theories (MMT)
- Foundation-independent
- Logics and logical frameworks represented as theories
- Generic declarative language: theories, declarations, expressions

Our Core Language (strict OMDoc 2 = MMT)

- A module system for mathematical theories (MMT)
- Foundation-independent
- Logics and logical frameworks represented as theories
- Generic declarative language: theories, declarations, expressions

Syntax

Modules	M	$::=$	$(\texttt{theory}\ T = \{\Sigma\})^*$
Theories	Σ	$::=$	$(c\,[:E]\,[=E] \mid \texttt{include}\ T \mid \texttt{meta}\ T)^*$
Expressions	E		OpenMath expressions

MMT Theories

Examples

$$
\begin{array}{l}
\texttt{theory}\ \mathit{Propositions} = \{ \\
\quad \texttt{type} \\
\quad \rightarrow \\
\quad \mathit{lam} \\
\quad \mathit{prop} \quad : \quad \texttt{type} \\
\quad \mathit{proof} \quad : \quad \mathit{prop} \rightarrow \texttt{type} \\
\}
\end{array}
$$

$$
\begin{array}{l}
\texttt{theory}\ \mathit{FOL} = \{ \\
\quad \texttt{include}\ \mathit{Propositions} \\
\quad \mathit{term} \quad : \quad \texttt{type} \\
\quad \wedge \quad : \quad \mathit{prop} \rightarrow \mathit{prop} \rightarrow \mathit{prop} \\
\quad \vdots \\
\quad \exists \quad : \quad (\mathit{term} \rightarrow \mathit{prop}) \rightarrow \mathit{prop} \\
\}
\end{array}
$$

Our Extension Layer (pragmatic OMDoc 2)

- Built on top of MMT
- Specify extension principles declaratively
- Two new primitives
  - extension declarations to introduce extension principles
  - pragmatic declarations to use extension principles

Syntax

$$\begin{array}{llll} \text{Theories} & \Sigma & ::= & (\ldots \\ & & \mid & \texttt{extension}\; e = \lambda x_1 : E_1 . \ldots \lambda x_n : E_n . \{\Sigma\} \\ & & \mid & \texttt{pragmatic}\; c : e\; E_1 \ldots E_n)^* \end{array}$$

Examples

An extension principle

$\texttt{extension}\ \mathit{theorem} = \lambda F : \mathit{prop}.\ \lambda D : \mathit{proof}\ F.\ \{$

$\quad c \quad : \quad \mathit{proof}\ F \ = \ D$

$\}$

A pragmatic declaration

$\texttt{pragmatic}\ \mathit{foo}\ :\ \mathit{theorem}\ (1 + 1 \doteq 2)\ (\mathit{proof\text{-}term})$

Modularity

An extension principle

$\texttt{theory}\ \mathit{Theorems} = \{$

$\quad\texttt{meta}\ \mathit{Propositions}$

$\quad\texttt{extension}\ \mathit{theorem} = \lambda F : \mathit{prop}.\ \lambda D : \mathit{proof}\ F.\ \{$

$\qquad c \quad : \quad \mathit{proof}\ F \ = \ D$

$\quad\}$

$\}$

A pragmatic declaration

$\texttt{theory}\ \mathit{MyTheorem} = \{$

$\quad\texttt{meta}\ \mathit{Theorems}$

$\quad\texttt{include}\ \mathit{Nats}$

$\quad\texttt{pragmatic}\ \mathit{foo}\ :\ \mathit{theorem}\ (1 + 1 \doteq 2)\ (\mathit{proof\text{-}term})$

$\}$

Pragmatic-to-Strict Translation

Semantics of pragmatic declarations:
Elaborate pragmatic declarations into strict (core) declarations.

$$
\begin{array}{l}
\texttt{extension}\ e = \lambda x_1 : E_1. \ldots \lambda x_n : E_n. \{ \\
\quad c_1 \ : \ \tau_1 \ = \ D_1 \\
\quad \vdots \\
\quad c_n \ : \ \tau_n \ = \ D_n \\
\} \\
\texttt{pragmatic}\ p : e\ A_1 \ldots A_n
\end{array}
$$

$$
p \ : \ e\ A_1 \ldots A_n \xrightarrow{\text{elaborate}} \begin{array}{c} p.\,c_1 : \gamma(\tau_1) = \gamma(D_1) \\ \vdots \\ p.\,c_n : \gamma(\tau_n) = \gamma(D_n) \end{array}
$$

γ substitutes A_i for x_i and $p.c_i$ for c_i.

Various Extension Principles

Mizar-Style Functor Definitions

$\mathtt{theory}\ \mathit{FunctorDefinitions} = \{$

$\quad\mathtt{meta}\ \mathit{Propositions}$

$\quad\mathtt{extension}\ \mathit{functor}\ = \lambda\alpha : \mathtt{type}.\ \lambda\beta : \mathtt{type}.$

$\qquad\lambda \mathit{means} : \alpha \to \beta \to \mathit{prop}.\ \lambda \mathit{existence} : \mathit{proof}\ \ldots.$

$\qquad\lambda \mathit{uniqueness} : \mathit{proof}\ \ldots.\ \{$

$\qquad\quad f \quad : \quad \alpha \to \beta$

$\qquad\quad \mathit{definitional\_theorem} \quad : \quad \mathit{proof}\ \forall x : \alpha.\ \mathit{means}\, x\, (f\ x)$

$\quad\}$

$\}$

Various Extension Principles

HOL-Style Type Definitions

$$
\begin{array}{l}
\texttt{theory}\ \mathit{Types} = \{\\
\quad \texttt{meta}\ \mathit{Propositions}\\
\quad \texttt{extension}\ \mathit{typeDef} = \lambda\alpha : \texttt{type}.\ \lambda A : \alpha \to \mathit{prop}.\ \lambda\rho : \mathit{proof}\ \ldots.\ \{\\
\qquad \begin{array}{lcl}
T & : & \texttt{type}\\
\mathit{Rep} & : & T \to \alpha\\
\mathit{Abs} & : & \alpha \to T\\
\mathit{Rep}' & : & \mathit{proof}\ \forall x : T.\ A\,(\mathit{Rep}\,x)\\
\mathit{Rep\_inverse} & : & \mathit{proof}\ \forall x : T.\ \mathit{Abs}\,(\mathit{Rep}\,x) \doteq x\\
\mathit{Abs\_inverse} & : & \mathit{proof}\ \forall x : \alpha.\ A\,x \Rightarrow \mathit{Rep}\,(\mathit{Abs}\,x) \doteq x
\end{array}\\
\quad \}\\
\}
\end{array}
$$

Concrete Syntax for Our Extension Layer

- For bidirectional pragmatic-to-strict translation
- extension $e = \lambda x_1 : E_1. \ldots \lambda x_n : E_n. \{\Sigma\}$ is written as

<**extension** name="e">
<**parameter** name="x_1">E_1</**parameter**>
$\vdots$
<**parameter** name="x_n">E_n</**parameter**>
<**theory**>
Σ
</**theory**>
</**extension**>

- pragmatic $c : e\ A_1 \ldots A_n$ is written as

<**pragmatic** name="c" extension="$\langle e \rangle$">
$A_1 \ldots A_n$
</**pragmatic**>

$\langle e \rangle$ denotes e's URI.

Pragmatic Surface Syntax

- Notation parser specific to each pragmatic surface syntax

  ongoing work for our Twelf surface syntax

  done for our sTeX surface syntax

pragmatic surface syntax

theorem	$1+1 \doteq 2$	(foo)
proof	$\mathit{proof\text{-}term}$	

↓ notation parser

pragmatic OMDoc 2 abstract syntax

$\mathit{foo} \quad : \quad \mathit{theorem}\ \ 1+1 \doteq 2\ \ (\mathit{proof\text{-}term})$

↓ pragmatic-to-strict

strict OMDoc 2 abstract syntax

$\mathit{foo} \quad : \quad 1+1 \doteq 2 \quad = \quad (\mathit{proof\text{-}term})$

Conclusion and Future Work

- User-definable, constrained, statement level extensions in MKM formats
- Generic: applicable to virtually any declarative language
- Realized within the OMDoc/MMT language
- Expressed common conservative extension principles
- Future: test with extension principles from widely used MKM formats
  - create library of extension principles
  - find limitations (candidates: abstract data types, proof schemas)