A Family of Modular XML Schema for MathML

by

Yuzhen Xie

Department of Computer Science

Submitted in partial fulfillment
of the requirements for the degree of
Master of Science

Faculty of Graduate Studies
The University of Western Ontario
London, Ontario
May, 2002

THE UNIVERSITY OF WESTERN ONTARIO

FACULTY OF GRADUATE STUDIES

CERTIFICATION OF EXAMINATION

Chief Advisors	Examining Board
_____	_____
_____	_____
	_____

The thesis by

Yuzhen Xie

entitled

A Family of Modular XML Schema for MathML

Is accepted in partial fulfillment of the

requirements for the degree of

Master of Science

Date _____ _____

Chair of Examining Board

Abstract

Studying the family of technologies related to MathML, including XML, MathML, XML DTD, and XML Schema, suggests that a Schema for MathML would significantly benefit MathML's usefulness and adoption. This thesis therefore aims to develop a modular XML Schema for MathML. Automatic schema generation tools were explored and found to be insufficient to develop schemas for such a complex XML application as MathML. Based on the analysis of the structure and types of MathML, a family of modular XML Schema for MathML was created from scratch. This has given the first complete Schemas for MathML. It consists of eight schema modules in one XML namespace. The modules are organized to form three independent complete schemas: one for Content MathML, one for Presentation MathML, and one for full MathML. We achieved our goals of small-size and simplicity by using this modular structure. Through this work a few issues regarding the application of the XML Schema language, the need for schema sub-modules, and the representation for DTD entities were also raised.

Key words: MathML, XML Schema, XML DTD, modular structure

Acknowledgements

I wish to extend my appreciation and gratitude to my supervisor Dr. Stephen M. Watt for his guidance, support and encouragement through this entire investigation.

Sincere thanks and appreciation are extended to Mr. Rob M. Corless, Mr. Greg J. Reid, Ms. Bethany Heinrichs, Ms. Dianne McFadzean, Ms. Janice Wiersma, and Ms. Cheryl McGrath for their assistance during the course of the studies.

I also wish to thank fellow graduates and friends at ORCCA, Ms. Huanling Lu, Ms. Elena Smirnova, Ms. Xiaofang Xie, Ms. Lihong Zhi, Mr. Laurentiu Dragan, Mr. William Naylor, Mr. Cosmin Oancea, Mr. Luca Padionov, Mr. Ignor Rodionov, and Mr. Bo Wan for their invaluable assistance to me and wonderful friendship.

My sincere appreciation also goes to my family for their love and support throughout my life and especially the past few years.

Thank God for

"Charm is deceitful and beauty is fleeting,
but a woman who fears the Lord will be praised.
Give her credit for what she has accomplished,
and let her works praise her in the city gates."

(Proverb 31:30-31)

Table of Contents

Certificate of Examination ii

Abstract iii

Acknowledgement iv

Table of Contents v

Chapter 1 Introduction 1

Chapter 2 XML Schema Review.....4

2.1 Overview of XML 4

2.2 The Idea of an XML DTD 5

2.3 The Idea of an XML Schema 9

2.3.1 Basic Mechanisms 10

2.3.2 Namespace and Multiple Schema Documents 14

2.3.3 Conformance 17

Chapter 3 MathML Review.....18

3.1 Goals of MathML 18

3.2 Presentation Markup, Content Markup, and Mixing Presentation and Content 18

3.3 MathML Syntax and Grammar 23

Chapter 4 XML Schema Issues and MathML 24

4.1 Issues that XML Schema Treats Properly 24

4.2 Automatic Schema Generation Tools and Problems ……29

4.3 Conceptual Tools to Develop XML Schema ……33

Chapter 5 Composition of a Modular XML Schema for MathML ……..36

5.1 Analysis of the Structure and Type of MathML ……36

5.2 Significance of Modularization ……..38

5.3 Design Principles ……39

5.4 Creation of the Modular Structure ……..40

5.5 Key Mechanisms used in Organizing Modular Structure ……....43

5.6 Key Mechanisms used in Composing Short Schemas ……47

Chapter 6 Concluding Remarks ……55

Glossary ……57

References ……59

Appendices: A Modular XML Schema for MathML ……....61

1. MathML.xsd ……...62
2. MathML-Presentation.xsd ……..64
3. MathML-Content.xsd ……65
4. Meta-MathML-Presentation.xsd ……....66
5. Meta-MathML-Content.xsd ……...84
6. low-levelContent.xsd ……....103
7. mathAtts.xsd ……110
8. MathMLCommonAtts.xsd ……...112

Vita ……..113

Chapter 1 Introduction

MathML [5] is the first application of XML [1]. The syntax and grammar of MathML is mainly defined in the MathML 2.0 DTD (Document Type Definition for MathML 2.0, February 2001).

XML DTDs are a feature of XML inherited from SGML (Standard Generalized Markup Language), which are used to create a template for document markup. On the other hand, XML Schemas were designed specifically for XML, to describe the structure semantics of sets of XML documents. Their form is specified by the W3C recommendation of May 2001 [3 and 4]. XML Schemas give functionality above and beyond that of DTDs.

The work of this thesis started from investigating the ideas of XML Schemas and XML DTDs as applied to MathML. For a complex XML application such as MathML, an XML Schema for MathML would help to refine its structure semantics and allow better software tools.

As a low-level format for describing mathematics and a basis for machine-to-machine communication, the likely usage of MathML lies in three areas. First, it tends to be used as the source data for the rendering of mathematical expressions within web pages. The second practical goal is to be a common language for the exchange of simple mathematics between applications that process mathematical semantics, for example, Maple and

Mathematica. The third usage is to serve as a source of mathematical structures for conversion to and from other formats and notations, for example, TeX and voice.

XML Schemas describe the possible valid contents of XML data. An XML Schema for MathML would make it easier and more flexible to describe, exchange, and validate mathematical and scientific content encoded in MathML. A schema could aid the MathML processor to enforce MathML semantic structure. Based on a schema, an application such as an editor can access the rules for the document structure to assure that only correct MathML is generated. The schema can also give hints to a MathML processor, for example, to help separate indenting from content. With the default or fixed values of attributes declared in the schema, an info set [18] can be provided to the instance documents.

A modularized XML Schema for MathML would help the end-user to tailor MathML for different purposes. For example, the schema for pure presentation markup in MathML could help increase efficiency and reduce complexity in an application concerning direct rendering to the user, such as screen display, printing, and conversion to speech. A modularized schema can support the creation of subsets and supersets of MathML for specific uses such as handheld devices and special-purpose appliances.

A comparison between Schemas and DTDs was performed, and issues that Schemas treat properly were discovered. Based on the analysis of the structure, types and semantics of MathML, we have seen that a modular XML Schema for MathML would greatly benefit the applications of MathML. The significance of modularization on MathML schemas is

evident. In comparison to other approaches, the complexity and size is reduced by an order of magnitude.

Approaches that are possibly used to create such a modular schema for MathML were evaluated. First, the current automated schema generation tools were investigated. Using these tools, schemas were produced from the conversion of MathML DTDs. The resulting schemas were bulky, repetitive and non-intuitive. We found it better, based on the thorough understanding of MathML structure and semantics, to create the schemas for MathML by hand.

The conceptual tools used to develop schemas were then examined. The organization of the development of the MathML schemas started from the modular structure of MathML. Our focus was on how to better use the schema mechanisms to organize the modules to create small-size schemas.

Our result is eight schema modules, created in one XML namespace. They can be organized to form three independent schemas for MathML and well-defined subsets: one for presentation markup, one for content markup, and one for the whole MathML markup. These are the first complete schemas for MathML. Moreover, the size of these schemas is about one tenth the size of automatically generated schemas. Through this experience with the XML Schema language, questions were also raised about the implementation of the XML Schema definition language, as well as some issues of MathML, such as MathML entities.

Chapter 2 XML Schema Review

2.1 Overview of XML

XML is a regularized subset of SGML. It inherits some characteristics from HTML (Hypertext Markup Language) [16], and contains additional features that are aimed at its use on the Internet and to overcome limitations in HTML.

XML predefines no tags. The authors create tags that are needed for their applications. Furthermore, unlike HTML, XML adopts a very strict syntax that results in smaller, faster, and lighter software tools such as browsers.

There are generally two types of applications for XML: document applications and data applications. Document applications manipulate information that is primarily intended for human consumption. XML is independent of the delivery medium since it concentrates on the structure of the document. Therefore, it is possible to edit and maintain documents in XML and to automatically publish them on different media in different format such as HTML, PostScript, PDF, RTF, etc.

Data applications manipulate information that is primarily intended for software consumption. With the structure of a database expressed in XML, this format can be used to exchange information between organizations. Concentrating on the structure, software can automatically visit the data list, extract the data, and update databases.

The basic form of XML is explained by the W3C recommendation [1]: "each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a 'root' or document entity that is the data file to represent the entire document. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. The logical and physical structures must nest properly".

There are two classes of documents recognized by XML: *well-formed* and *valid*. "Well-formed" documents have the right structure of start and end tags, attributes are properly quoted, entities are acceptable, character sets are properly used, but they have no DTD or Schema. "Valid" documents additionally have a DTD or a Schema. Based on the DTD or Schema, an XML processor can check if the syntax and structure of the documents are correct.

2.2 The Idea of an XML DTD

The Document Type Definition **(**DTD**)** [2] is the original modeling language to describe the tree structure of XML documents. This is a significant feature that XML inherits from SGML. Using DTDs, XML is able to create a template for document markup so that the placement of elements and their attributes can be controlled and validated.

A DTD consists of a number of declarations that are grouped together. Each declaration conforms to the markup declaration format, '<!……>', and is classified using one of the following keywords: `ELEMENT` (tag definition), `ATTLIST` (attribute definitions), `ENTITY` (entity definition), and `NOTATION` (data type notation definition).

An element declaration is used to define a new element and specify its allowed content. A canonically empty element can hold no child elements and also no text. An element declared to have a content of '`ANY`' may contain all of the other elements declared in the DTD. A model group is used to define an element that has specific mixed content or element content.

An *element content* may contain only child elements. A *mixed content* may have a mixture of child elements and free text. Sequence control over many content tokens is achieved through two logical connector operators: ',' (sequence connector) and '|' (choice connector). Quantity indicators such as ?, + and * are used to specify the occurrence rules. A keyword `PCDATA` (Parsable Character Data) is used to indicate the location where document text is allowed.

Attribute declarations define attribute names, the value types and default values. A *value type* can be one of the following: `CDATA`, `NMTOKEN`, `NMTOKENS`, `ENTITY`, `ENTITIES`, `ID`, `IDREF`, `IDREFS`, notation, or name group. The '`#REQUIRED`' keyword is used to indicate that a particular attribute must be present each time the element it belongs to is used. The '`#IMPLIED`' keyword is used to specify that the attribute can be absent. The

keyword '#FIXED' indicates that the provided default value is the only value the attribute can take. In the XML standard, reserved attributes that begin with 'xml' such as xml:lang may already have assigned meanings. xml:lang is used to specify the language used in the contents and attribute values of any element in an XML document.

Conditional sections indicated by the key word 'INCLUDE' or 'IGNORE' are used to identify portions of a DTD as optional segments, which can be easily included or excluded from processing to build alternative document models. *Notation declarations* define which format may be embedded when an element or entity contains non-XML format data. CDATA sections are used where character data may occur to escape blocks of text containing characters that would be, if unspecified, recognized as markup.

White space (spaces, tabs, and blank lines) used in editing XML documents to set apart the markup for greater readability is typically not intended for inclusion in the delivered version of the document. When white space in an element should be preserved by applications, an attribute xml:space can be attached to it to indicate the intention.

The following "Note.dtd" given by Neil Bradley (1999) [17] illustrates most of the important features of the XML DTDs. The file of "Note.xml" is a valid instance for the class of XML documents defined by "Note.dtd".

When a DTD is processed, an internal DTD subset must be processed. Typically, this subset contains document-specific declarations, such as local entity definitions. The

`standalone` parameter of the XML declaration is used to inform a non-validating XML processor whether or not the external subset of the DTD must be read for the document to be processed accurately. Furthermore, there are some restrictions on the use of the markup for a document containing neither an internal nor an external DTD subset.

Note.dtd:

```
<!-- The Note DTD version 1.3 -->
<!NOTATION TIFF SYSTEM
      "TIFFVIEW.EXE" >
<!ENTITY % images "IGNOR" >
<![%images[<!ENTITY % noteContent "p |
      image">]]>
<!ENTITY % noteContent "p">
<!element note (%noteContent;)*>
<!element p (#PCDATA)>
<!element image EMPTY>
<!attlist image filename ENTITY
      #REQUIRED>
```

Note.xml:

```
<?XML version="1.0" encoding="UTF-8"
      standalone="no" ?>
<!DOCTYPE Note SYSTEM "Note.DTD" [
      <!ENTITY XML "eXtensible Markup Language">
      <!ENTITY history SYSTEM "History.XML">
      <!ENTITY XMLimage SYSTEM "/ents/XML.TIF"
```

```
            NDATA TIFF>
      <!ENTITY % image "INCLUDE">
]>
<note>
<p>The &XML; format is a very important move to bringing the
benefits of structured markup to the masses.</p>
&history;
<p>The following image shows a fragment of XML:</p>
<image filename="XMLimage" />
<p>The tags <![CDATA [<note>, <p> and
       <image../> are used in this document]]>.</p>
</note>
```

2.3 The Idea of an XML Schema

An XML Schema [1] provides a mechanism to specify the structural semantics of XML documents. It aims at defining a class of XML documents so that, as an "instance document", an XML document can be described to conform to a particular schema. The XML Schema language is grammar-based, represented with an XML 1.0 syntax, and uses namespaces. Schemas substantially reconstruct and considerably extend the capabilities found in document type definitions (DTDs). The features of the XML Schema language provide functionality above and beyond what is provided by DTDs.

2.3.1 Basic Mechanisms

The basic mechanisms of XML Schema include element and attribute declarations, and simple and complex type descriptions and definitions. Elements are said to have *complex type* if they contain subelements or carry attributes. Elements have *simple type* when they contain numbers (and strings, dates, etc.) but do not contain any subelements. Attributes always have simple types.

Simple types can be either built-ins in XML Schema, such as *string* and *decimal*, or derivations from the built-ins. Using the keyword "`restriction`", new simple types can be defined by deriving them from existing simple types (built-ins and derived). For example, we can define a simple type named fontStyle by the following definition:

```
<simpleType name=”fontStyle”>
      <restriction base="string">
            <enumeration value="normal"/>
            <enumeration value="bold"/>
      </restriction>
</simpleType>
```

Simple types can also be list types or union types. *List types* consist of sequences of atomic types. *Union types* consist of one or more instances of one type.

Using the `complexType` element, new complex types can be defined. These typically consist of a set of element declarations, element references, and attribute declarations.

Global elements and attributes are declared as the children of the `schema` element. They can be referenced elsewhere in the schema to create new elements or attributes. The occurrence of an element is defined by the `minOccurs` and `maxOccurs` attributes in its declaration. The appearance of an attribute can be declared by using a `use` attribute with a value of either `required`, or `optional`, or `prohibited`. The following schema section illustrates these features. The `mms` denotes the target namespace of our Schemas for MathML. It is defined as one of the `schema` properties in the schema files, here is `xmlns:mms="http://www.orcca.on.ca/MathML/Schema"`.

```
<attribute name="fontsize" type="string"/>
<attribute name="fontfamily" type="string"/>
<attribute name="alt" type="string"/>
<element name="mglyph">
     <complexType>
          <attribute ref="mms:alt" use="optional"/>
          <attribute ref="mms:fontfamily" use="optional"/>
          <attribute ref="mms:index" use="optional"/>
     </complexType>
 </element>
 <complexType name="mglyphType">
     <sequence>
          <element ref="mms:mglyph"
               minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
 </complexType>
```

There are other variations in the content models of elements. A complex type can be derived from a simple type by extending it, using a `simpleContent` element to indicate that the new type contains only character data and no elements. The *mixed content* model constructs schemas where character data can appear alongside sub-elements, and character data is not confined to the deepest sub-element. The key word `sequence` is used to specify that the order and number of child elements appearing in an instance must agree with the order and number of child elements defined in the model.

The *empty content* model defines elements that have no content at all. `anyType` is used in places where no constraints are required on the content of an element. An XML schema uses an attribute `nillable` for an element to indicate whether an element can appear with or without a non-nil value. The following examples describe the situations discussed herein.

```
<complexType name="carPrice">
      <simpleContent>
            <extension base="decimal">
                  <attribute name="currency" type="string"/>
            </extension>
      </simpleContent>
</complexType>

<complexType name="salutation" mixed="true">
      <sequence>
            <element name="title" type="string"/>
            <element name="name" type="string"/>
      </sequence>
</complexType>


<element name="somebody" type="anyType" nillable="true"/>
```

In order to help both the human readers and applications to understand schemas, three elements are designed to annotate them. The `annotation` element can be used at the beginning of most schema constructions. Its child elements include `documentation` and `appInfo`. In most cases, the `xml:lang` attribute of the `documentation` element specifies the language of the information for the definition of an element or a complex type. Information for tools, stylesheets and other applications can be indicated by the `appInfo` element.

Named and un-named groups of elements and attributes fill the role of parameter entities in XML 1.0 DTDs. Like the globally declared elements and attributes, the various groups can be declared and edited in one place and referenced in multiple definitions and declarations later. This approach can increase the readability of schemas, and ease their maintenance. The XML schema language provides four elements to define groups appearing in a content model: `sequence`, `choice`, `group`, and `all`. The `attributeGroup` element provides a way to define and edit a list of attributes together. All of them may contain `minOccurs` and `maxOccurs` attributes.

The `group` element can be used to bind some elements or attributes together, which can be named or un-named. The `sequence` element indicates the order the elements or attributes must be present. The `choice` element declares that only one of its children can be appear in an instance. The `all` element constrains cases where at the top-level of a content model all the elements in the group must be individual elements which may appear once or not at all, and they may show in any order. Any content models that are

expressible with an XML 1.0 DTD can be represented by XML schema by combining and nesting the various groups and by setting up the appropriate constraints. We give the following as an example:

```
<group name="ptoken">
      <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="mms:mi"/>
            <element ref="mms:mn"/>
            <element ref="mms:mo"/>
            <element ref="mms:mtext"/>
            <element ref="mms:ms"/>
      </choice>
 </group>
```

2.3.2 Namespaces and Multiple Schema Documents

The "target namespace" mechanism is a facility to identify a collection of type definitions and element declarations of a schema which belong to the particular namespace. The target namespace is important to schema validation, for which elements and attributes in an instance document are checked against the elements and attributes declarations and type definitions in the schemas. The definitions and declarations between schemas can be re-used by referring types in another schema with a different target namespace.

The `schema` element has `elementFormDefault` and `attributeFormDefault` to specify the qualification of local elements and attributes. The value of `unqualified`

or `qualified` is used to describe whether locally declared elements and attributes must be qualified or not. Qualification can also be controlled on a declaration by declaration basis using the `form` attribute. An example of this would be:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:mms="http://www.orcca.on.ca/MathML/Schema"
      targetNamespace="http://www.orcca.on.ca/MathML/Schema"
      elementFormDefault="qualified"
      attributeFormDefault="qualified">
      …
      <element name=”currency” type=”string”
            form=”qualified”/>
</schema>
```

It is not necessary to declare a prefix associated with the XML Schema elements and types when a target name space is declared and the default name space is “`http://www.w3.org/2001/XMLSchema`” in a schema. In such a schema, a prefix denoting the target name space must be associated to the user-defined global elements and types. When a schema is designed without a target namespace, a prefix such as `xsd:` must be used to associate the XML Schema elements and types with the XML Schema namespace “`http://www.w3.org/2001/XMLSchema`”. Its purpose is to identify the elements and simple types as belonging to the vocabulary of the XML Schema language rather than the vocabulary of the schema author.

To improve the ease of maintenance, re-usability and readability, the contents of a larger schema can be divided into several schema documents. The XML Schema language provides the `include` and `redefine` mechanisms to allow re-use of schema components, and redefinition of simple and complex types, groups, and attribute groups from external schema files in one namespace. The `import` mechanism enables schema elements and types to be used across multiple namespaces. New types can be created based on existing ones by extension or restriction. Existing types and groups can be redefined by extension or restriction. Moreover, elements can be substituted for other elements by substitution groups. An example would be:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:mms="http://www.orcca.on.ca/billSchema"
        targetNamespace="http://www.orcca.on.ca/billSchema"
      <include schemaLocation=
            "http://www.orcca.on.ca/billSchema/commonAtts.xsd"/>
      <include schemaLocation=
            "http://www.orcca.on.ca/billSchema/stub.xsd"/>
      <import schemaLocation=
            "http://www.orcca.on.ca/customerSchema/customer.xsd"/>
      …
</schema>
```

2.3.3 Conformance

To validate an XML document, conformance checking is needed to verify whether the rules defined in the schema are followed. There are two tasks performed in this processing. One is "schema validation", in which an instance document is checked against a schema for conformance. Another one is called "infoset contributions", which adds supplementary information such as types and default values.

All the elements and types used in the instance document are processed against the schema for validity. Starting from the root element, the sub-elements are checked one by one until the whole document is processed. Their contents are evaluated to follow the content models defined in the schema. Attributes and types including simple types and complex types are verified against the attributes and types permitted by the schema.

Chapter 3 MathML Review

3.1 Goals of MathML

MathML is an application of XML for describing mathematical notation and capturing both its structure and content. It aims at enabling mathematics to be served, received, and processed on the World Wide Web and in digital libraries.

Due to the complex structure of mathematical expressions, it is not practical to describe them in raw HTML. Presently, most mathematical contents on the Web are created in images of mathematical notation (in GIF or JPEG format), or in PDF format for the whole document. This makes the interconnectivity impossible. MathML is an important step in finding ways of communicating mathematics that assist automatic processing, searching and indexing, and reuse in other mathematical applications and contexts.

Typically, MathML is also useful for web services. Mathematical web services are designed in a two-layer architecture. The lower layer is comprised of general software tools exchanging, processing, and rendering suitably encoded mathematical data. MathML is designed to provide the encoding of mathematical information for the bottom of this layer. The top layer consists of specialized software tools for interchange of information and rendering to multiple media, for example, conversions between T_EX markup to HTML and MathML, and conversions between MathML and the format type in speech or print, or a computer algebra system.

3.2 Presentation Markup, Content Markup, and Mixing Presentation and Content

MathML can encode both the presentation of a mathematical notation and the content of the mathematical idea or object which it represents. There are three categories of MathML elements: *presentation elements*, *content elements* and *interface elements*. Being modeled as trees, MathML markup can reflect the recursive nature of mathematical objects and notation. Each MathML element corresponds a node in the tree. The branches under a parent node represent its children. The leaves in the tree are primitive notation or content units such as numbers, characters, etc.

The visually oriented two-dimensional structure of mathematical notation can be described in *presentation markup*, which consists of about thirty elements accepting over 50 attributes. Most of them belong to layout schemata, containing other presentation elements. Each two-dimensional device, such as a superscript or subscript, fraction or table, is a layout schema. The most commonly used *token elements* are `mi`, `mn` and `mo` for representing identifiers, numbers and operators respectively.

The layout schemata are classified into several groups. One group contains `mrow`, `mstyle`, and `mfrac`, which focus on more general layout. Another group concerning with scripts includes elements such as `msub`, `munder`, and `mmultiscripts`. Tables are dealt with in the third group. The `maction` element in a separate category is used to encode various kinds of actions on notation. The order of child schemata is very

important for many layout schemata. For example, the first child of a `mfrac` element denotes the numerator and second child denotes the denominator. An example of the presentation markup of $(a+b)^2$ is given as below.

```
<mrow>
      <msup>
            <mfenced>
                  <mrow>
                        <mi>a</mi>
                        <mo>+</mo>
                        <mi>b</mi>
                  </mrow>
            </mfenced>
            <mn>2</mn>
      </msup>
</mrow>
```

Content markup describes the semantics of the mathematical notations. It includes about 120 elements that accept about a dozen attributes. Most of them are empty elements such as `partialdiff`, `leq`, and `tan`. These denote a wide variety of operators, relations and named functions. The various mathematical data types, such as `matrix` and `set`, lie in an important category of "container elements".

The other elements such as `apply` are used to combine operations into expressions and to make new mathematical objects from others. The order of the child elements is usually

significant. For example, the first child of apply element is the function to be applied and the remaining children are the arguments of the function in order. The `declare` element is used to declare a variable to be a certain type with a certain value. It is especially important when the content markup needs to be evaluated by a computer algebra system. For example, the content markup fragment for $(a+b)^2$ is:

```
<apply>
      <power/>
      <apply>
            <plus/>
            <ci>a</ci>
            <ci>b</ci>
      </apply>
      <cn>2</cn>
</apply>
```

Mixed presentation and content markup should be used in places where it makes sense. Content markup embedded in presentation markup can be used when any content fragment is semantically meaningful and does not require additional arguments or quantifiers to be fully specified. Presentation markup embedded in content markup requires that presentation markup must be contained in a content token element. The `semantics` element can also be used to provide a content expression to serve as a 'semantic annotation' for a presentation expression. The following is a mixed markup example given by W3C [5]:

Notation: $\int_1^t dx/x$

```
<mrow>
      <semantics>
            <mrow>
                  <msubsup>
                        <mo>&int;</mo>
                        <mn>1</mn>
                        <mi>t</mi>
                  </msubsup>
                  <mfrac>
                        <mrow>
                              <mo>&dd;</mo>
                              <mi>x</mi>
                        </mrow>
                        <mi>x</mi>
                  </mfrac>
            </mrow>
            <annotation-xml encoding="MathML-Content">
                  <apply>
                        <int/>
                        <bvar><ci>x</ci></bvar>
                        <lowlimit><cn>1</cn></lowlimit>
                        <uplimit><ci>t</ci></uplimit>
                        <apply>
                              <divide/>
                              <cn>1</cn>
                              <ci>x</ci>
                        </apply>
                  </apply>
            </annotation-xml>
      </semantics>
</mrow>
```

3.3 MathML Syntax and Grammar

Since MathML is an application of XML, it follows the rules of XML syntax. Part of its grammar is specified by the MathML Document Type Definition (DTD) [5]. The MathML DTD defines the details about element and attribute names, which elements can be nested inside each other, etc. The details about how to use tags, attributes, entity references and so on are governed by the XML language specification. However, MathML specifies additional syntax and rules, to encode a great deal more information than with pure XML. These rules place additional criteria on attribute values and more detailed restrictions on the child elements.

The XML Schema language has been created as a normative recommendation by W3C ([2] and [3]). To increase the flexibility of the use of XML for the Web, W3C encourages use of Schema for application built with XML. Thus, a schema would be highly desirable for MathML to enhance the MathML syntax and grammar.

Chapter 4 XML Schema Issues and MathML

4.1 Issues that XML Schema Treats Properly

In general, an XML Schema refines the semantic structure of an XML document. The XML Schema content model provides functionality to better express the syntactic, structural and value constraints applicable to an XML application. Since the XML Schema language takes the syntax and grammar of XML, it can benefit from all the advantages of XML, such as independence of transport medium, possibility to use XML tools such as parsers, editors and browsers to process schemas.

XML Schemas are relatively strong in data typing, compared with the limited textual content such as CDATA, PCDATA, "true", and "false" available with DTDs. There are more than forty built-in simple types, such as `string`, `token`, `integer`, and `negativeInteger`.

The XML Schema language also provides flexible approaches for users to create derived simple or complex types based on the built-in types, such as `restriction`, `enumeration`, `pattern`, `list`, and `union`. For example, we can use the `enumeration` to define a new simple type called CanadaProvince. This would be derived from `string`, and must have a value of one of the standard Canada Province names.

```
<xsd:simpleType name="CanadaProvince">
      <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Ontario"/>
            <xsd:enumeration value="British Columbia"/>
            <xsd:enumeration value="Manitoba"/>
            <! -- and so on -- >
      </xsd:restriction>
</xsd:simpleType>
```

Complex types can be derived from simple types and existing complex types using extension, restriction, or substitution. Abstract elements and types can be declared so that new elements and types can be created later by substitution, which mimics the inheritance mechanism in object-oriented concepts. The `simpleContent` element is used to indicate a complex type containing only character data and no elements.

XML Schema also provides the mixed content mechanism to construct schemas where character data can appear alongside sub-elements, and character data is not confined to the deepest sub-elements. To achieve this, the `mixed` attribute on the `complexType` definition is set to `true`.

Fundamentally, the *mixed* model in XML Schema is different from the *mixed* model in XML 1.0 DTD. Under the XML 1.0 DTD mixed model, it is not possible to constrain the order and number of child elements appearing in an instance document. In contrast, XML Schema gives full validation of mixed models so that the order and number of child

elements appearing in an instance document must agree with the order and number of child elements specified in the model.

Beyond the built-in type library provided by XML Schema, schema authors are free to create their own type libraries saved as separated files so that they can be shared and used as building blocks for creating new schemas.

Similar to the parameter entities in DTD, the XML Schema language provides a grouping mechanism to define and name groups of elements and attributes. They can be used later to build content models. Moreover, the combined and nested use of `group`, `choice`, `sequence`, and `all` elements together with the occurrence constraints `minOccurs` and `maxOccurs` provide additional expressive power. For example, the `all` element defines a group in which all the elements may appear once or not at all, and they may show in any order.

In terms of the occurrence constraints, the minimum and maximum number of times an element may appear is determined by the value of attributes `minOccurs` and `maxOccurs` respectively in XML Schema. For example, when the `graduateClass` element must contain between 3 and 10 `graduateStudent` elements, the XML Schema declaration of the `graduateClass` element type with the desired number of `graduateStudent` children might look like:

```
<xsd:complexType name="graduateClass">
      <xsd:element name="graduateStudent"
           minOccurs="3" maxOccurs="10"/>
</xsd:complexType>
```

The special characters in DTDs ((none), "+", "?" and "*") have limitations in defining the number of a given element that may appear. The 'no specification' indicates that the child must occur only once. The "+" specifies that the child may occur one or more times. The "?" denotes that the child may occur once or not at all. The "*" says that the child may appear zero or more times. It is frustrating to define the situation described above in DTDs. This can be done only as follows.

```
<!ELEMENT graduateClass (
      (graduateStudent, graduateStudent, graduateStudent) |
      (graduateStudent, graduateStudent,
            graduateStudent, graduateStudent) |
      (graduateStudent, graduateStudent, graduateStudent,
           graduateStudent, graduateStudent) |
      … …
>
```

or

```
<!ELEMENT graduateClass (graduateStudent, graduateStudent,
graduateStudent, graduateStudent?, graduateStudent?,
graduateStudent?, graduateStudent?, graduateStudent?,
graduateStudent?, graduateStudent?) >
```

XML Schema also provides more expressive power to modularize and help manage large schemas and large XML document sets: the namespace mechanism enhances the XML Schema extensibility. Large schemas can be divided into small pieces and saved as

individual schema files. Although DTDs can use external entities to achieve this effect, it is limited in reusing the contents of the schema sub-modules.

In XML Schema, schema modules can be reused in another module in the same namespace by `include` or `redefine` mechanism. The `import` element is used to include a schema file from an external namespace. Groups and types can be redefined by extension or restriction. New types can be created based on existing ones using extension or restriction. This approach can accomplish a modularized and more legible schema for both human readers and software tools.

Another advantage of Schema is the flexibility in the use of attributes and attribute groups in the building of the content models. In XML Schema, it is not necessary to define the use for an attribute when globally declared. The use can be specified later in the group definition and content model construction. This way increases the flexibility and reusability of the attributes. For example, the attribute named `fontsize` here is declared globally, and later its use is defined as optional when used in building a content model of an element named label.

```
<attribute name=”fontsize” type=”string”/>
<element name=”label”>
      <complexType>
            <attribute ref=”mms:fontsize” use=optional/>
      </complexType>
</element>
```

In DTD, the use of an attribute is defined at its declaration, and there does not seem to exist a way to redefine it later when constructing the attribute groups and building the content models.

4.2 Automatic Schema Generation Tools and Problems

We have examined two programs for automatically translating DTD into XML Schema: dtd2xsd.pl [7] and dtd2xs [8]. One is written in Perl [7] and one is in Java [8]. Their idea is to use pattern matching to map meaningful DTD entities onto XML Schema constructs. Both claim to handle encoding of elements, attributes, simple types, attribute groups, and model groups. For example, `<!ELEMENT ROOT (A, B)>` in DTD was translated into the following construct for an element declaration in XML Schema [7].

```
<element name=”ROOT”>
      <complexType content=””elementOnly”>
            <element ref=”t:A”/>
            <element ref=”t:B”/>
      </complexType>
</element>
```

The declaration of `<!ATTLIST ROOT a CDATA #FIXED "x">` in DTD was converted into:

```
<element name=”ROOT”>
      <complexType content=”elementOnly”>
            <attribute name=”a” type=”string”
                  use=”fixed” value=”x”/>
      </complexType>
</element>
```

In this study, with the MathML DTD 2.0 as input, we tried both of the programs to generate a schema file for MathML. These results indicate that these automatic schema generation tools are inadequate for such a complex schema as MathML. They might work for simple DTDs, as tested by their authors. However, for a complicated application like MathML, problems, even errors, have been identified in many aspects, including attribute and element declaration, type definition, group definition, as well as content model construction. Furthermore, modularization of schemas was not considered at all by these tools.

In the schema files converted by the two tools from the MathML 2.0 DTD, most of the attributes were declared locally with an element when defining its content, resulting in tens of repeated attribute declarations, such as the following list of attributes.

```
<attribute name='xmlns' type='string' use='fixed'
      value='http://www.w3.org/1998/Math/MathML'/>
<attribute name='xmlns:xlink' type='string'
      use='fixed' value='http://www.w3.org/1999/xlink'/>
<attribute name='xlink:href' type='string' minOccurs='0'/>
```

```
<attribute name='class' type='string' minOccurs='0'/>
<attribute name='style' type='string' minOccurs='0'/>
<attribute name='id' type='ID' minOccurs='0'/>
<attribute name='xref' type='IDREF' minOccurs='0'/>
<attribute name='other' type='string' minOccurs='0'/>
<attribute name='definitionURL' type='string' minOccurs='0'/>
<attribute name='encoding' type='string' minOccurs='0'/>
```

The attribute declarations generated by the dtd2xsd.pl have invalid properties. For example, the `minOccurs` and `maxOccurs` were used as attribute properties. In the schema file generated by converting MathML DTD 2.0 by the dtd2xs tool, element names were not correctly defined, for example, the element `mi` was named as %mi.qname which took the way of referring to a parameter entity in MathML DTDs.

In neither case were named types defined. There was therefore no way to re-use the same types in building the element content models. This resulted in a very large amount of repeated code in the schema files. For example, the sixty content markup elements and thirty presentation markup elements that can be present as child elements (under some of the presentation and content elements) were repeatedly listed by reference in the content models many times.

There were errors in declaring the complex types for the contents of the elements. For example, the "`content = 'elementOnly'`" used as a property for complexType declaration. The group mechanism was misused in the conversion. We noted that all the

elements were declared separately, and there were no group definitions in the context. However, `group` was used to refer to the single element in the complex type definitions. The following code generated by dtd2xs can be an example.

```
<xs:element name="%apply.qname;">
      <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                  <xs:group ref="csymbol.qname" />
                  <xs:group ref="ci.qname" />
                  <xs:group ref="cn.qname" />
                  <xs:group ref="apply.qname" />
                  …
                  <xs:group ref="malignqroup.qname" />
                  <xs:group ref="malignmark.qname" />
                  <xs:group ref="maction.qname" />
            </xs:choice>
            <xs:attributeGroup ref="NamespaceDecl.attrib" />
            <xs:attribute ref="xlink:href" />
            <xs:attribute name="class" type="xs:string" />
            …
            <xs:attribute name="other" type="xs:string" />
      </xs:complexType>
</xs:element>
```

Since no named types or groups were considered, no derived types and redefined groups were used. Due to the repeated attribute declarations and element references in building element content models, the files were quite large in size. One was about 230 KB (by dtd2xsd.pl), and the other was about 300 KB (by dtd2xs). Furthermore, the schemas were not at all modular despite the structure of the DTD. These generated schemas would be unsuitable for tailoring MathML for different use scenarios.

These observations indicate that, with the current state of schema generating tools, in order to better use XML Schema mechanisms to create suitable MathML schemas, it is still necessary to hand write schemas, based on thorough understanding of the structure, types, constraints, and use scenarios of MathML.

4.3 Conceptual Tools to Develop XML Schema

Various kinds of tools have been developed or are under development to work with the W3C XML Schema Definition Language [9]. Among them, TIBCO TurboXML [10], XML Spy [11], XML Schema Quality Checker[12], XSV (XML Schema Validator) [13], and XSDComp (XML Schema Compilation) [15] are of particular interest in creating, validating, and managing XML schemas.

As the first product to comprehensively support for the latest XML standard, TIBCO TurboXML is an Integrated Development Environment (IDE) for authoring and validating XML schemas, as well as XML files and DTDs. It also provides tools for

converting older XML schemas which correspond to October 24 Candidate Release [19] and April 7 Draft [20 and 21] to conform to the current specification. Its management console allows users to perform batch validation, conversion, transformation, and reporting on schemas, DTDs, and XML files.

XML Spy includes a graphical XML Schema Editor, which consists of functions including a Schema Design Menu, XML Schema Design View for both global particles and content models, and a Schema Documentation. Major aspects of the XML Schema language such as element and attribute declarations can be edited in-place using a graphical schema representation. All globally defined items, i.e. elements, attributeGroups, complexTypes, etc., can be displayed using a list in the XML Schema Design View. The content model of an element or an attribute can be shown and edited in a tree view of the XML Spy. The editor also offers a configurable interface for automatically generating XML Schema documentation. XML Spy appears to have been well accepted, having won a number of awards.

XML Schema Quality Checker, XSV, and XSDComp aim at validating XML Schema instances. XML Schema Quality Checker [12] is a Java program to check the validation of schema files against XML Schema specifications. It takes as input an XML Schema and examines improper uses of the XML Schema language. In addition to reporting error messages, it gives suggestions about how to make corrections where it is not obvious. Schemas in multiple documents that are connected via `<include>`, `<import>` or `<redefine>` mechanisms can also be checked in a schema-wide verification.

At time of writing, XSV and XSDComp are under development. XSV is a reference implementation (a proof-of-concept implementation for the W3C) developed by Henry S. Thompson and Richard Tobin. As reported in [14], the current status of the XSV version 1.203.2.20/1.106.2.11 is that it has implemented partially the basic framework of schema checking and instance schema-validation, such as attribute validation, content-model validation, and `include`. There are still Schema mechanisms which it has not implemented yet, such as redefinition of named groups and attribute groups. XSV can be used as an online service, or under Windows, or on any Python platform. XSDComp is an open source project proposed by Curt Arnold to create a W3C XML Schema validator in XSLT. According to the status report, about thirty percent of the development has been done recently.

In addition, it is of course possible to write XML schemas using a normal text editors, but with no support for XML Schema features. We found that the Emacs editor was a useful tool to write XML Schema files, with the support for creating HTML and XML files that provides basic check on tag matching.

Chapter 5 Composition of a Modular XML Schema for MathML

5.1 Analysis of the Structure and Types of MathML

As observed in Chapter 3, MathML consists of three categories of elements: presentation, content, and interface. To recursively build a mathematical object, it is necessary for most presentation or content elements to enclose some number of other MathML elements for the constituent pieces. Thus, MathML expressions can be represented as trees.

The leaf nodes in a mathematical expression tree are classified as *canonically empty* elements, *token* elements, and *annotation* element. Canonically empty elements contain no bodies. They describe basic concepts directly in MathML, such as the content element `<plus/>`. Token elements include presentation token elements such as `mi` and `mn` and content token elements such as `ci`, `cn` and `csymbol`. Token elements are the only MathML elements allowed to accept MathML character data in MathML. MathML character data is composed of Unicode characters with the occasional addition of special characters constructed with the `mglyph` element. Data that is not in MathML format can be included using the `annotation` element.

In terms of the type definitions in XML Schema, all the MathML elements have complex types. Although some of them contain no elements (for example, the presentation element `mspace` and the content element `sep`), they still have attributes.

Due to the recursive expression of the mathematical objects in MathML, the content models can include presentation constructs at the lowest level so that presentation forms can hold presentation or content elements. They can also include content constructs at lowest level so that content elements hold PCDATA or presentation elements at leaf level (for permitted substitutable elements in context).

It is, therefore the case that most of the presentation elements can contain other presentation elements including themselves. We see this, for example, with the general layout forms and the script and limit schemata. Most of the content elements can contain other content elements including themselves. We see this, for example, with forms such as `lambda` and `apply`.

To perform mixed presentation and content markup, a list of presentation elements such as `mi` and `mn` are allowed to be present under some of the content elements. Likewise, a list of content elements such as `ci` and `cn` are allowed to be contained in some of the presentation elements. Therefore most of the elements in MathML have complex types, which are composed of both elements and attributes coming from both presentation and content MathML. Mixed content and presentation markup can also arise using the

semantics element, but this does not introduce any new requirements from the XML Schema point of view.

Most of the attributes have a basic simple type of *string*. Some of them have enumerated textual values, which can be derived by restriction from *string*. A few of the attributes have Boolean types.

In addition, MathML has more than two thousand character entities which play an import role in describing mathematical notations and symbols, for example, `&alpha;` and `&proportionalto;` defined in MathML DTD 2.0.

5.2 Significance of Modularization

Due to the complexity of MathML structure and content, we expect modularization to help increase the conceptual clarity and readability of an XML schema, as well as being more flexible and maintainable.

Decomposition of the whole functionality can help organize the development of the MathML schema, thus reducing its complexity. This will facilitate reuse of common modules and hence reduce repetition by creating well-defined modules to perform particular tasks. Since single modules can be maintained independently, a modular XML Schema for MathML will be more flexible.

Most importantly, a modular XML Schema for MathML in the MathML Schema namespace would allow end users to tailor MathML for different purposes, such as handheld devices and special-purpose applications, to reduce the overall work load in accessing MathML schemas, and in processing based on MathML schemas.

For example, on some occasions such as print publication a pure MathML presentation markup that has nothing to do with the MathML content markup might be desired. In other situations, such as mathematical computation, a pure MathML content markup that has nothing to do with the MathML presentation markup might be needed. There are also situations where mixed MathML presentation and content markup is required, for instance web-based mathematical communication and computation.

5.3 Design Principles

The main goal is to create coherent sets of semantically related modules within a MathML schema namespace to allow pure presentation markup and pure content markup, as well as mixed presentation and content markup.

Naturally, the two types of MathML markup (presentation and content) should be the basic starting points in modularizing the XML Schema for MathML. However, due to the recursive expression of mathematical objects in MathML, particular considerations should be given in defining modules for separating presentation markup and content

markup, as well as in creating supersets for a whole MathML schema for mixing presentation markup and content markup.

Since many attributes in MathML are common to most elements, it is more appropriate to declare them globally or in separate modules so that they will be easy to reuse. Furthermore, the *use* of the attributes is not necessarily defined when declaring them globally. This provides considerable flexibility for reuse in defining the groups and types by means of appropriate instances of the `use` property.

Reusability, readability, and extensibility of the schema modules are important factors. For example, groups of attributes and elements should be properly defined to provide a logical, easy to understand, and consistent organization. Basic complex types should be correctly identified and built so that (based on them) more complex types can be derived, allowing extension and noticeably reducing amount of code.

5.4 Creation of the Modular Structure

Eight modules are defined and created in a hierarchical structure for the XML Schema for MathML. A description of the hierarchical modular structure of the XML Schema for MathML is illustrated in Figure 5.1.

The attributes shared by all the elements in MathML are defined in one module, called "`MathMLCommonAtts.xsd`" (2 KB). All the attributes accepted by the top-level

element `math` are given in "`mathAtts.xsd`" (2 KB). The referencing use of these attributes can be achieved by including these files in a module. This avoids the repeated declaration for common attributes in many modules.

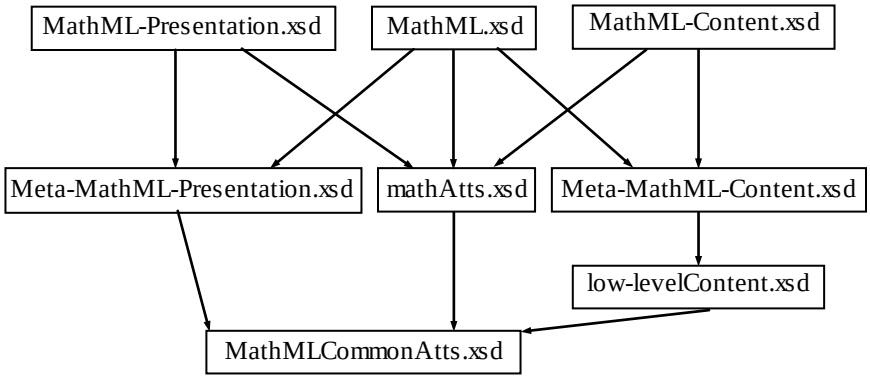

Figure 5.1 Hierarchical Modular Structure of XML Schema for MathML

The canonically empty content elements are declared in a separate module named "`low-levelContent.xsd`" (9 KB). Its size is small and it can be used separately when needed. It is included in "`Meta-MathML-Content.xsd`" (19 KB) to construct the content models for high-level content elements.

The "`Meta-MathML-Presentation.xsd`" (17 KB) defines an XML Schema for a pure presentation world. That is, content elements are not considered in their content models so that the related presentation elements have only presentation elements at the leaf level. Consequently, only the presentation elements are allowed at the leaf level of `<math>` element to form a schema for standalone presentation markup in "`MathML-Presentation.xsd`" (1 KB).

Likewise, the "Meta-MathML-Content.xsd" defines schema for pure content world. That is, presentation elements are not considered in their content models so that the related content elements have only content elements at the leaf level. Only the content elements are allowed at the leaf level of <math> element to form a schema for standalone content markup in "MathML-Content.xsd" (1 KB).

In "MathML.xsd" (2 KB), the content models for presentation and content markup are redefined so that content elements are allowed to be present at the leaf level of presentation elements, and vice versa. A new type that includes both presentation elements and content elements is created for the <math> element.

As a result, three standalone schemas are formed:

(1) The schema for mixed presentation and content markup (51 KB in total) consists of:

- "MathML.xsd",
- "Meta-MathML-Presentation.xsd",
- "Meta-MathML-Content.xsd",
- "low-levelContent.xsd",
- "mathAtts.xsd", and
- "MathMLCommonAtts.xsd".

(2) The schema for pure presentation markup (24 KB in total) includes:

- "MathML-Presentation.xsd",
- "Meta-MathML-Presentation.xsd",
- "mathAtts.xsd", and
- "MathMLCommonAtts.xsd".

(3) The schema for pure content markup (31 KB in total) consists of:

- "MathML-Content.xsd",
- "Meta-MathML-Content.xsd",
- "low-levelContent.xsd",
- "mathAtts.xsd", and
- "MathMLCommonAtts.xsd".

5.5 Key Mechanisms used in Organizing Modular Structure

The eight modules we created as individual schema files are located within the namespace of MathML schema. To allow their flexible reuse, all the elements and most of the attributes are declared globally in each module. To organize the constructs among them, the <include> and <redefine> mechanisms in the XML Schema language are exploited to reuse, derive, and modify schema components with the same target namespace, in this case "http://www.orcca.on.ca/MathML/Schema".

The include element is used to bring in the definitions and declarations contained in the schema document specified by its schemaLocation attribute. For example, the components in "MathMLCommonAtts.xsd" are added to the including schema "mathAtts.xsd", "Meta-MathML-Presentation.xsd", and "low-levelContent.xsd" respectively by using the following statement in each file:

```
<include schemaLocation=
      "http://www.orcca.on.ca/
      MathML/Schema/MathMLCommonAtts.xsd"/>
```

When a schema module covers multiple schema documents, only the top-most document and the common namespace need be referenced. For example, since "MathMLCommonAtts.xsd" has already been included in "low-levelContent.xsd", "Meta-MathML-Content.xsd" only needs to include "low-levelContent.xsd", resulting in both "MathMLCommonAtts.xsd" and "low-levelContent.xsd" being added to the existing target namespace.

Multiple include elements are used to include more than one schema module. For example, both "mathAtts.xsd" and "Meta-MathML-Presentation.xsd" are included in "MathML-Presentation.xsd" by the following two statements:

```
<include schemaLocation=
      "http://www.orcca.on.ca/MathML/Schema/mathAtts.xsd"/>
<include schemaLocation=
      "http://www.orcca.on.ca/
      MathML/Schema/Meta-MathML-Presentation.xsd"/>
```

Once more, the "MathML-Content.xsd" includes both "mathAtts.xsd" and "Meta-MathML-Content.xsd" to bring their components into the existing target namespace:

```
<include schemaLocation=
      "http://www.orcca.on.ca/MathML/Schema/mathAtts.xsd"/>
<include schemaLocation=
      "http://www.orcca.on.ca/
      MathML/Schema/Meta-MathML-Content.xsd"/>
```

The elements, attributes and types in the included schema can be reused in the including schema. New groups and types can be defined or derived based on the existing ones. For example, in "Meta-MathML-Content.xsd" element groups such as c0ary, cfuncoplary and carithoplary are defined according to the element declarations in "low-levelContent.xsd". Another example is the complex type named contentExpressionBaseType, which is constructed by referencing the element group of ContentExpression defined in "Meta-MathML-Content.xsd" and the attribute group of MATHML.Common.attrib, defined in "MathMLCommonAtts.xsd".

In "MathML-Presentation.xsd", the <math> element is defined with the type of mathPresentationType, which is constructed to contain the element group of PresInCont with the appropriate attribute lists. By this approach a partial schema for only presentation markup is formed. In the same way, a partial schema for only content

markup is created in "MathML-Content.xsd", with the type of math element defined to enclose ContInPres along with its attribute list.

The redefine mechanism is used to redefine simple and complex types, groups, and attribute groups that are obtained from external schema files specified in the schemaLocation attribute. It is also required that the external schema files have the same target namespace as the redefining schema. Once the components are redefined they turn out to be part of the redefining schemas target namespace.

In composing the whole XML Schema for MathML in "MathML.xsd", the group of PresExpression is redefined to enclose both the previously defined PresExpression group in "Meta-MathML-Presentation" and the ContInPres group defined in "Meta-MathML-Content.xsd". The group of ContentExpression is redefined to contain the previously defined ContentExpression group in "Meta-MathML-Content.xsd" and the PresInCont group defined in "Meta-MathML-Presentation.xsd". This use of redefine is as follows:

```
<redefine schemaLocation=
      "http://www.orcca.on.ca/
      MathML/schemas/Meta-MathML-Presentation.xsd">
<!-- redefinition of PresExpression group -->
      <group name="PresExpression">
            <choice minOccurs="0" maxOccurs="unbounded">
                  <group ref="mms:PresExpression"/>
                  <group ref="mms:ContInPres"/>
            </choice>
      </group>
</redefine>
```

```
<redefine schemaLocation=
      "http://www.orcca.on.ca/
            MathML/schemas/Meta-MathML-Content.xsd">
<!-- redefinition of ContentExpression group -->
      <group name="ContentExpression">
            <choice minOccurs="0" maxOccurs="unbounded">
                  <group ref="mms:ContentExpression"/>
                  <group ref="mms:PresInCont"/>
            </choice>
      </group>
</redefine>
```

Correspondingly, in "`MathML.xsd`" the content model of the `math` element is built to contain both the element groups of `ContInPres` and `PresInCont` by referencing the group of `MathExpression`. We have used the above techniques to create the whole XML Schema family for MathML, with three standalone schema are formed to allow pure presentation markup, pure content markup, as well as mixed presentation and content markup.

This way of creating and organizing schemas could be achieved by developing a software tool to generate schemas from DTDs, taking into account the logical structure. For example, based on the logical structure of MathML, this tool could be used to identify the points in DTDs to logically separate presentation markup from content markup, then recombine and redefine them when forming schema subsets or supersets. MathML

common attributes and canonically empty elements could also be recognized and shaped as separate schema modules.

5.6 Key Mechanisms used in Composing Short Schemas

All the elements and most of the attributes are declared globally so that they can be referenced anywhere needed. Named groups of attributes and elements are used similarly to the parameter entities defined in MathML DTDs. By simply referencing their names these named groups are used to construct the content models of elements. In constructing content models for the elements, a key scheme is the identification and creation of basic types, which are derived later to build more complex types. Recursive definitions are used to build the content models of high-level elements in both presentation markup and content markup.

The attributes in MathML that are common to all the elements are declared in two attribute group definitions in "`MathMLCommonAtts.xsd`", including `xmlns:m`, `xmlns:xlink`, `xlink:href`, `class`, `style`, `id`, `xref`, and `other`. They are reused by all the other modules. All the attributes for presentation elements and content elements are also declared globally and groups of attributes are defined accordingly. The following example shows the declaration of attribute group of "MATHML.xmlns.attrib".

```
<attributeGroup name="MATHML.xmlns.attrib">
      <attribute name="xmlns:m" type="string"
            fixed="http://www.w3.org/1998/Math/MathML"/>
      <attribute name="xmlns:xlink" type="string"
            fixed="http://www.w3.org/1999/xlink"/>
 </attributeGroup>
```

All the attributes for top-level element `math` and those for browser interface element are declared globally in "`mathAtts.xsd`" module. Attribute groups i.e. `topinfo` and `browif` are then defined according to their grouping rule when presenting in the content model of element `math`. Being included in the top-level modules, they have been reused three times by reference in constructing the content model of the `math` element to form the schemas for pure presentation markup in "`MathML-Presentation.xsd`", pure content markup in "`MathML-Content.xsd`", and mixed presentation and content markup in "`MathML.xsd`".

Definitions for named groups of attributes and elements are extensively utilized to make grouping rules and group names similar to the parameter entities in MathML 2.0 DTD. When declared globally, the use property of an attribute is not necessarily defined, which gives flexibility for reuse. For example, the attributes regarding the font information in presentation markup are declared globally without definition of use, and their uses are specified later in defining the group of `fontinfo`. The attribute group of `fontinfo` is referenced by name in building the types for some of the presentation elements such as `mi`, `mn`, and `mo`. This is done as follows:

```
<attribute name="fontsize" type="string"/>


<attribute name="fontweight">
      <simpleType>
            <restriction base="string">
                  <enumeration value="normal"/>
                  <enumeration value="bold"/>
            </restriction>
      </simpleType>
</attribute>
<attribute name="fontstyle">
      <simpleType>
            <restriction base="string">
                  <enumeration value="normal"/>
                  <enumeration value="italic"/>
            </restriction>
      </simpleType>
</attribute>
<attribute name="fontfamily" type="string"/>
<attribute name="color" type="string"/>
<attribute name="mathvariant" type="string"/>
<attribute name="mathsize" type="string"/>
<attribute name="mathcolor" type="string"/>
<attribute name="mathbackground" type="string"/>
```

```
<attributeGroup name="fontinfo">
      <attribute ref="mms:fontsize" use="optional"/>
      <attribute ref="mms:fontweight" use="optional"/>
      <attribute ref="mms:fontstyle" use="optional"/>
      <attribute ref="mms:fontfamily" use="optional"/>
      <attribute ref="mms:color" use="optional"/>
      <attribute ref="mms:mathvariant" use="optional"/>
      <attribute ref="mms:mathsize" use="optional"/>
      <attribute ref="mms:mathcolor" use="optional"/>
      <attribute ref="mms:mathbackground" use="optional"/>
</attributeGroup>
```

In defining elements with different usage scenarios, basic types are identified and defined. These are used later to derive other related complex types. For example, in "`low-levelContent.xsd`" document, a complex type that describes the basic type of low-level element is defined with the name of `contentBaseType` as follows.

```
<complexType name="contentBaseType">
      <attributeGroup ref="mms:MATHML.Common.attrib"/>
      <attribute ref="mms:definitionURL"/>
      <attribute ref="mms:encoding"/>
 </complexType>
```

This named type is simply referenced by its name by the `type` property in the low-level content element declarations including `integers`, `reals`, `rationals`, and etc, for

instance, `<element name="integers" type="mms:contentBaseType"/>`. A derived type named `tendstoType` is created by extension based on `contentBaseType` to enclose one more attribute. This type is taken by the element of `tendsto` with the `type` attribute assigned to the value of `tendstoType`:

```
<element name="tendsto" type="mms:tendstoType"/>

<complexType name="tendstoType">
     <complexContent>
          <extension base="mms:contentBaseType"
               <attribute ref="mms:type" use="optional"/>
          </extension>
     </complexContent>
</complexType>
```

We have used this scheme widely in creating our MathML schemas.

The other important basic types are identified and defined as follows:

```
<complexType name="ContentExpressionBaseType">
     <group ref="mms:ContentExpression"
          minOccurs="0" maxOccurs="unbounded"/>
     <attributeGroup ref="mms:MATHML.Common.attrib"/>
</complexType>
<complexType name="tokenBaseType">
     <union memberTypes=
          "mms:MathMLCharacters mms:malignmarkType"/>
</complexType>
```

```
<complexType name="PresBaseType">
      <group ref="mms:PresExpression"
            minOccurs="0" maxOccurs="unbounded"/>
      <attributeGroup ref="mms:MATHML.Common.attrib"/>
</complexType>
```

The top-level elements for both the presentation markup and content markup are defined recursively. For example, the following fragment of schema code reveals the recursive definition for content elements `apply`, `reln`, and `lambda`. The type of the three elements is all declared to have `ContentExpressionBaseType`, which is constructed to have zero or more occurrence of the elements in the group of `Content` contained in `ContentExpression`. Meanwhile, the three elements are all contained in the group of `cspecial`, which is a component of group `Content`.

```
<complexType name="ContentExpressionBaseType">
      <group ref="mms:ContentExpression"
            minOccurs="0" maxOccurs="unbounded"/>
      <attributeGroup ref="mms:MATHML.Common.attrib"/>
</complexType>

<!-- Content elements: specials -->
<element name="apply" type="mms:ContentExpressionBaseType"/>

<element name="reln" type="mms:ContentExpressionBaseType"/>
```

```
<element name="lambda" type="mms:ContentExpressionBaseType"/>
<group name="cspecial">
     <choice minOccurs="0" maxOccurs="unbounded">
          <element ref="mms:apply"/>
          <element ref="mms:reln"/>
          <element ref="mms:lambda"/>
     </choice>
</group>
<!-- Content constructs: all -->
<group name="Content">
     <choice minOccurs="0" maxOccurs="unbounded">
          <group ref="mms:ctoken"/>
          <group ref="mms:cspecial"/>
          <group ref="mms:"cother/>
          <group ref="mms:csemantics"/>
          <group ref="mms:c0ary"/>
          <group ref="mms:cconstructor"/>
          <group ref="mms:cquantifier"/>
          <group ref="mms:cop1ary"/>
          <group ref="mms:cop2ary"/>
          <group ref="mms:copnary"/>
          <group ref="mms:copmisc"/>
          <group ref="mms:crel2ary"/>
          <group ref="mms:crelnary"/>
     </choice>
</group>
```

Chapter 6 Concluding Remarks

We have created a modular XML Schema family for MathML. It consists of eight short schema documents in one namespace. Organized properly, they are able to form three independent schemas for presentation markup, content markup, and mixed presentation and content markup, and are intended to serve users to tailor MathML for different purposes. The results show the improvement of XML Schema over DTDs. These are the first complete Schemas for MathML. The size of our schema (53 KB) is much smaller than those generated by the automated tools (230 KB and 300 KB).

We have used many aspects of the XML Schema language in writing these schemas. Some key XML Schema mechanisms such as `group`, `include` and `redefine` were widely used in creating these modularized short schemas. The result demonstrates some strengths of the XML Schema language. Although both Schemas and DTDs are able to characterize sets of XML documents for a particular purpose, Schemas have functionality which is beyond that of a DTD, including mechanisms to construct and organize modular schemas, the methods to create derived simple and complex types, and the ways to reuse the previously defined components.

We note two questions for future attention: Would it be useful to sub-module the generic content elements such as *apply* and *lambda*, particularly as Meta-sub-modules? What would be the best way to represent entities in MathML DTDs? We observe that there

remains a splendid opportunity to create a high-quality tool to generate schemas from DTDs taking into account the logical structure.

Glossary

The following definitions are derived from the "The XML Companion" [17] by Neil Bradly.

DTD (Document Type Definition) — The instructions that codify rules for a particular type of document.

HTML (Hypertext Markup Language) — A non-application-specific DTD developed for delivery and presentation of documents over the Web, to be viewed using a HTML browser.

MathML (Mathematical Markup Language) — An application of XML for describing mathematical expression and capturing both its notation and semantics. It aims at enabling mathematics to be served, received, and processed on the World Wide Web [5].

Namespace — An environment to group element and attribute names and make them unique.

Schema — The definition of a document structure, including value constraints and relationships between objects.

SGML (Standard Generalized Markup Language) — The ISO 8879 standard developed in 1986 to assist electronic delivery and publication of text-based documents.

XML (eXtensible Markup Language) — A generalized markup language based on SGML, with some influence from HTML, aimed primarily at the Web.

W3C (World Wide Web Consortium) — An industry consortium founded in 1994 that comprises over 120 organizations involved in the establishment of standards for the Web, including XML and DTDs for versions of HTML. In agreement with major vendors, responsible for the latest versions of HTML, starting with HTML 3.2in June 1996 and HTML 4.0 on 18 December 1997.

WWW (World Wide Web, or Web) — An Internet service that uses the HTTP protocol and HTML format data files to provide a document delivery service.

References

[1] W3C Recommendation: eXtensible Markup Language (XML) 1.0, February 1998.

[2] W3C Recommendation: Document Type Definition (DTD), Extensible Markup Language (XML) 1.0, February 1998.

[3] W3C Recommendation: XML Schema Part 1: Structures, May 2001.

[4] W3C Recommendation: XML Schema Part 2: Datatypes, May 2001.

[5] W3C Recommendation: Mathematical Markup Language (MathML) Version 2.0, February 2001.

[6] W3C Note: Datatypes for DTDs (DT4DTD) 1.0, January 2000.

[7] Mary Holstege, *et al*, A Conversion Tool from DTD to XML Schema, http://www.w3.org/2000/04/schema_hack/, January 2001.

[8] J. Dudeck, *et al*, dtd2xs, http://puvogel.informatik.med.uni-giessen.de/dtd2xs/, March 2002.

[9] Eric van der Vlist and Lisa Rein, W3C XML Schema Tools Guide, http://www.xml.com/pub/a/2000/12/13/schematools.html, December 13, 2001.

[10] TIBCO Extensibility – XML Infrastructure Solutions, http://www.tibco.com/products/extensibility/solutions/turbo_xml.html, May 2002.

[11] XML Spy, http://www.xmlspy.com, May 2002.

[12] IBM alphaWorks emerging technologies, http://www.alphaworks.ibm.com/tech/xmlsqc?open&l=xml-dev,t=grx,p=shecheck, May 2002.

[13] W3C Validator for XML Schema 20000922 version, XML Output, http://www.w3.org/2000/09/webdata/xsv, May 2002.

[14] Henry S. Thompson and Richard Tobin, Current Status of XSV: Coverage, Known Bugs, etc. Applies to XSV 1.203.2.42/1.106.2.22 of 2002/01/11 16:40:28, http://www.ltg.ed.ac.uk/~ht/xsv-status.html, 15 January 2002.

[15] arstechnica, http://sourceforge.net/projects/xsdcomp, May 2002

[16] W3C Recommendation: HTML 4.01 Specification, December 1999.

[17] Neil Bradley, The XML Companion, Addison-Wesley, 2000.

[18] W3C Recommendation: XML Information Set, 24 October 2001.

[19] W3C Issues XML Schema as a Candidate Recommendation, 24 October 2000

[20] XML Schema Part 1: Structures, ed. Henry S. Thompson et al. W3C Working Draft 7 April 2000, W3C, 7 April 2000.

[21] XML Schema Part 2: Datatypes, ed. Paul V. Biron and Ashok Malhotra. W3C Working Draft 7 April 2000, 7 April 2000.

Appendices

A Modular XML Schema for MathML

1. MathML.xsd
2. MathML-Presentation.xsd
3. MathML-Content.xsd
4. Meta-MathML-Presentation.xsd
5. Meta-MathML-Content.xsd
6. low-levelContent.xsd
7. mathAtts.xsd
8. MathMLCommonAtts.xsd

1. MathML.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:mms="http://www.orcca.on.ca/MathML/Schema"
        targetNamespace="http://www.orcca.on.ca/MathML/Schema"
        elementFormDefault="qualified"
        attributeFormDefault="qualified">

 <annotation>
  <documentation xml:lang="en">
   Module name: MathML.xsd
   Size: 2 KB
   XML schema for top-level element 'math' in MathML, and
   redefinition of PresExpression group and
   ContentExpression group.
   Copyright 2002 ORCCA. All rights reserved.
  </documentation>
 </annotation>

 <include schemaLocation=
      "http://www.orcca.on.ca/MathML/Schema/mathAtts.xsd"/>

 <redefine schemaLocation=
      "http://www.orcca.on.ca/
      MathML/Schema/Meta-MathML-Presentation.xsd">
  <!-- redefinition of PresExpression group -->
  <group name="PresExpression">
   <choice minOccurs="0" maxOccurs="unbounded">
    <group ref="mms:PresExpression"/>
    <group ref="mms:ContInPres"/>
   </choice>
  </group>
 </redefine>
```

```
 <redefine schemaLocation=
      "http://www.orcca.on.ca/
      MathML/schemas/Meta-MathML-Content.xsd">
  <!-- redefinition of ContentExpression group -->
  <group name="ContentExpression">
   <choice minOccurs="0" maxOccurs="unbounded">
    <group ref="mms:ContentExpression"/>
    <group ref="mms:PresInCont"/>
   </choice>
  </group>
 </redefine>

 <group name="MathExpression">
  <choice minOccurs="0" maxOccurs="unbounded">
   <group ref="mms:ContInPres"/>
   <group ref="mms:PresInCont"/>
  </choice>
 </group>

 <complexType name="mathType">
  <group ref="mms:MathExpression"
      minOccurs="0" maxOccurs="unbounded"/>
  <attributeGroup ref="mms:topinfo"/>
  <attributeGroup ref="mms:browif"/>
 </complexType>

 <element name="math" type="mms:mathType"/>

</schema>
```

2. MathML-Presentation.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:mms="http://www.orcca.on.ca/MathML/Schema"
        targetNamespace="http://www.orcca.on.ca/MathML/Schema"
        elementFormDefault="qualified"
        attributeFormDefault="qualified">

 <annotation>
  <documentation xml:lang="en">
   Module name: MathML-Presentation.xsd
   Size: 1 KB
   XML schema for top-level element 'math' for presentation
   MathML. The content elements are not included in the
   content models.
   Copyright 2002 ORCCA. All rights reserved.
  </documentation>
 </annotation>

 <include schemaLocation=
     "http://www.orcca.on.ca/MathML/Schema/mathAtts.xsd"/>
 <include schemaLocation=
     "http://www.orcca.on.ca/
     MathML/Schema/Meta-MathML-Presentation.xsd"/>

 <complexType name="mathPresenationType">
  <group ref="mms:PresInCont"
     minOccurs="0" maxOccurs="unbounded"/>
  <attributeGroup ref="mms:topinfo"/>
  <attributeGroup ref="mms:browif"/>
 </complexType>

 <element name="math" type="mms:mathPresentationType"/>
</schema>
```

3. MathML-Content.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:mms="http://www.orcca.on.ca/MathML/Schema"
        targetNamespace="http://www.orcca.on.ca/MathML/Schema"
        elementFormDefault="qualified"
        attributeFormDefault="qualified">

 <annotation>
  <documentation xml:lang="en">
   Module name: MathML-Content.xsd
   Size: 1 KB
   XML schema for top-level element 'math' for content MathML.
   The presentation elements are not included in the content
   models.
   Copyright 2002 ORCCA. All rights reserved.
  </documentation>
 </annotation>

 <include schemaLocation=
     "http://www.orcca.on.ca/MathML/Schema/mathAtts.xsd"/>
 <include schemaLocation=
     "http://www.orcca.on.ca/
     MathML/Schema/Meta-MathML-Content.xsd"/>

 <complexType name="mathContentType">
  <group ref="mms:ContInPres"
     minOccurs="0" maxOccurs="unbounded"/>
  <attributeGroup ref="mms:topinfo"/>
  <attributeGroup ref="mms:browif"/>
 </complexType>

 <element name="math" type="mms:mathContentType"/>
</schema>
```

4. Meta-MathML-Presentation.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:mms="http://www.orcca.on.ca/MathML/Schema"
        targetNamespace="http://www.orcca.on.ca/MathML/Schema"
        elementFormDefault="qualified"
        attributeFormDefault="qualified">

 <annotation>
  <documentation xml:lang="en">
   Module name: Meta-MathML-Presentation.xsd
   Size: 19 KB
   Meta XML schema for presentation MathML. The content
   elements are not included in the content models.
   Copyright 2002 ORCCA. All rights reserved.
  </documentation>
 </annotation>

 <include schemaLocation=
     "http://www.orcca.on.ca/
     MathML/Schema/MathMLCommonAtts.xsd"/>

 <!-- global attribute and attribute group declaration -->

 <attribute name="fontsize" type="string"/>
 <attribute name="fontweight">
  <simpleType>
   <restriction base="string">
    <enumeration value="normal"/>
     <enumeration value="bold"/>
   </restriction>
  </simpleType>
 </attribute>
 <attribute name="fontstyle">
```

```
 <simpleType>
  <restriction base="string">
   <enumeration value="normal"/>
   <enumeration value="italic"/>
  </restriction>
 </simpleType>
</attribute>
<attribute name="fontfamily" type="string"/>
<attribute name="color" type="string"/>
<attribute name="mathvariant" type="string"/>
<attribute name="mathsize" type="string"/>
<attribute name="mathcolor" type="string"/>
<attribute name="mathbackground" type="string"/>

<attributeGroup name="fontinfo">
 <attribute ref="mms:fontsize" use="optional"/>
 <attribute ref="mms:fontweight" use="optional"/>
 <attribute ref="mms:fontstyle" use="optional"/>
 <attribute ref="mms:fontfamily" use="optional"/>
 <attribute ref="mms:color" use="optional"/>
 <attribute ref="mms:mathvariant" use="optional"/>
 <attribute ref="mms:mathsize" use="optional"/>
 <attribute ref="mms:mathcolor" use="optional"/>
 <attribute ref="mms:mathbackground" use="optional"/>
</attributeGroup>

<attribute name="form">
 <simpleType>
  <restriction base="string">
   <enumeration value="prefix"/>
   <enumeration value="infix"/>
   <enumeration value="postfix"/>
  </restriction>
 </simpleType>
```

```
</attribute>
<attribute name="fence" type="boolean"/>
<attribute name="seperator" type="boolean"/>
<attribute name="lspace" type="string"/>
<attribute name="rspace" type="string"/>
<attribute name="stretchy" type="boolean"/>
<attribute name="symmetric" type="boolean"/>
<attribute name="maxsize" type="string"/>
<attribute name="minsize" type="string"/>
<attribute name="largeop" type="boolean"/>
<attribute name="movablelimits" type="boolean"/>
<attribute name="accent" type="boolean"/>

<attributeGroup name="opinfo">
 <attribute ref="mms:form" use="optional"/>
 <attribute ref="mms:fence" use="optional"/>
 <attribute ref="mms:seperator" use="optional"/>
 <attribute ref="mms:lspace" use="optional"/>
 <attribute ref="mms:rspace" use="optional"/>
 <attribute ref="mms:stretchy" use="optional"/>
 <attribute ref="mms:symmetric" use="optional"/>
 <attribute ref="mms:maxsize" use="optional"/>
 <attribute ref="mms:minsize" use="optional"/>
 <attribute ref="mms:largeop" use="optional"/>
 <attribute ref="mms:movablelimits" use="optional"/>
 <attribute ref="mms:accent" use="optional"/>
</attributeGroup>

<attribute name="width" type="string"/>
<attribute name="height" type="string"/>
<attribute name="depth" type="string"/>
<attribute name="linebreak" type="string"/>

<attributeGroup name="sizeinfo">
```

```
 <attribute ref="mms:width" use="optional"/>
 <attribute ref="mms:height" use="optional"/>
 <attribute ref="mms:depth" use="optional"/>
</attributeGroup>

<attribute name="lquote" type="string"/>
<attribute name="rquote" type="string"/>
<attribute name="linethickness" type="string"/>
<attribute name="scriptlevel" type="string"/>
<attribute name="displaystyle" type="boolean"/>
<attribute name="scriptsizemultiplier" type="string"/>
<attribute name="scriptminsize" type="string"/>
<attribute name="background" type="string"/>
<attribute name="veryverythinmathspace" type="string"/>
<attribute name="verythinmathspace" type="string"/>
<attribute name="thinmathspace" type="string"/>
<attribute name="mediummathspace" type="string"/>
<attribute name="thickmathspace" type="string"/>
<attribute name="verythickmathspace" type="string"/>
<attribute name="veryverythickmathspace" type="string"/>
<attribute name="open" type="string"/>
<attribute name="close" type="string"/>
<attribute name="separators" type="string"/>
<attribute name="subscriptshift" type="string"/>
<attribute name="superscriptshift" type="string"/>
<attribute name="accentunder" type="boolean"/>

<attribute name="align" type="string"/>
<attribute name="rowalign" type="string"/>
<attribute name="columnalign" type="string"/>
<attribute name="columnwidth" type="string"/>
<attribute name="groupalign" type="string"/>
<attribute name="alignmentscope" type="string"/>
<attribute name="rowspacing" type="string"/>
```

```
<attribute name="columnspacing" type="string"/>
<attribute name="rowlines" type="string"/>
<attribute name="columnlines" type="string"/>
<attribute name="frame">
 <simpleType>
  <restriction base="string">
   <enumeration value="none"/>
   <enumeration value="solid"/>
   <enumeration value="dashed"/>
   </restriction>
 </simpleType>
</attribute>
<attribute name="framespacing" type="string"/>
<attribute name="equalrows" type="string"/>
<attribute name="equalcolumns" type="string"/>
<attribute name="displaystyle" type="boolean"/>

<attributeGroup name="tableinfo">
 <attribute ref="mms:align" use="optional"/>
 <attribute ref="mms:rowalign" use="optional"/>
 <attribute ref="mms:columnalign" use="optional"/>
 <attribute ref="mms:columnwidth" use="optional"/>
 <attribute ref="mms:groupalign" use="optional"/>
 <attribute ref="mms:alignmentscope" use="optional"/>
 <attribute ref="mms:rowspacing" use="optional"/>
 <attribute ref="mms:columnspacing" use="optional"/>
 <attribute ref="mms:rowlines" use="optional"/>
 <attribute ref="mms:columnlines" use="optional"/>
 <attribute ref="mms:frame" use="optional"/>
 <attribute ref="mms:framespacing" use="optional"/>
 <attribute ref="mms:equalrows" use="optional"/>
 <attribute ref="mms:equalcolumns" use="optional"/>
 <attribute ref="mms:displaystyle" use="optional"/>
</attributeGroup>
```

```
<attribute name="rowspan" type="string"/>
<attribute name="columnspan" type="string"/>
<attribute name="edge">
 <simpleType>
  <restriction base="string">
   <enumeration value="left"/>
   <enumeration value="right"/>
  </restriction>
 </simpleType>
</attribute>
<attribute name="actiontype" type="string"/>
<attribute name="selection" type="string"/>
<attribute name="name" type="string"/>
<attribute name="alt" type="string"/>
<attribute name="index" type="string"/>
<attribute name="bevelled" type="string"/>

<!-- end of attribute and attribute group declaration -->

<!-- Presentation schemata with content -->

<!-- empty elements definition -->
<element name="mglyph">
 <complexType>
  <attribute ref="mms:alt" use="optional"/>
  <attribute ref="mms:fontfamily" use="optional"/>
  <attribute ref="mms:index" use="optional"/>
 </complexType>
</element>

<complexType name="mglyphType">
 <sequence>
  <element ref="mms:mglyph"
```

```
      minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
 </complexType>

 <simpleType name="#PCDATA" type="string"/>

 <!-- PCDATA or MathML character elements -->
 <complexType name="MathMLCharacters">
  <union memberTypes="mms:#PCDATA mms:mglyphType"/>
 </complexType>

 <element name="mspace">
  <complexType>
   <attributeGroup ref="mms:sizeinfo"/>
   <attribute ref="mms:linebreak" use="optional"/>
   <attributeGroup ref="mms:MATHML.Common.attrib"/>
  </complexType>
 </element>

 <!-- Empty presentation schemata -->
 <group name="petoken">
  <element ref="mms:mspace" minOccurs="0" maxOccurs="unbounded"/>
 </group>

 <element name="mprescripts" type="mms:pscreschemaType"/>

 <element name="none" type="mms:pscreschemaType"/>

 <complexType name="pscreschemaType">
  <attributeGroup ref="mms:MATHML.xmlns.attrib"/>
 </complexType>

 <!-- Presentation layout schemata: empty elements for scripts --
>
```

```
 <group name="pscreschema">
  <choice minOccurs="0" maxOccurs="unbounded">
   <element ref="mms:mprescripts"/>
   <element ref="mms:none"/>
  </choice>
 </group>

 <element name="malignmark">
  <complexType>
   <attribute ref="mms:edge" use="optional"/>
  </complexType>
 </element>

 <element name="maligngroup">
  <complexType>
   <attributeGroup ref="mms:MATHML.Common.attrib"/>
   <attribute ref="mms:groupalign" use="optional"/>
  </complexType>
 </element>

 <!-- Presentation elements contain PCDATA or malignmark
constructs. -->
 <element name="mi" type="mms:tokenmiType"/>
 <element name="mn" type="mms:tokenmiType"/>
 <element name="mo" type="mms:tokenmoType"/>
 <element name="mtext" type="mms:tokenmiType"/>
 <element name="ms" type="mms:tokenmsType"/>

 <group name="ptoken">
  <choice minOccurs="0" maxOccurs="unbounded">
   <element ref="mms:mi"/>
   <element ref="mms:mn"/>
   <element ref="mms:mo"/>
   <element ref="mms:mtext"/>
```

```
  <element ref="mms:ms"/>
 </choice>
</group>

<complexType name="malignmarkType">
 <sequence>
  <element ref="mms:malignmark"
     minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
</complexType>

<complexType name="tokenBaseType">
 <union memberTypes="mms:MathMLCharacters mms:malignmarkType"/>
</complexType>

<complexType name="tokenmiType">
 <complexContent>
  <extension base="mms:tokenBaseType">
   <attributeGroup ref="mms:MATHML.xmlns.attrib"/>
   <attributeGroup ref="mms:fontinfo"/>
  </extension>
 </complexContent>
</complexType>

<complexType name="tokenmoType">
 <complexContent>
  <extension base="mms:tokenmiType">
   <attributeGroup ref="mms:opinfo"/>
  </extension>
 </complexContent>
</complexType>

<complexType name="tokenmsType">
 <complexContent>
```

```
  <extension base="mms:tokenmiType">
   <attribute ref="mms:lquote" use="optional"/>
   <attribute ref="mms:rquote" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<!-- Presentation: general layout schemata -->
<element name="mrow" type="PresBaseType"/>
<element name="mfrac" type=mfracType""/>
<element name="msqrt" type="PresBaseType"/>
<element name="mroot" type="PresBaseType"/>
<element name="menclose" type="mencloseType"/>
<element name="mstyle" type="mstyleType"/>
<element name="merror" type="PresBaseType"/>
<element name="mpadded" type="mpaddedType"/>
<element name="mphantom" type="PresBaseType"/>
<element name="mfenced" type="mfencedType"/>

<group name="pgenschema">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:mrow"/>
  <element ref="mms:mfrac"/>
  <element ref="mms:msqrt"/>
  <element ref="mms:mroot"/>
  <element ref="mms:menclose"/>
  <element ref="mms:mstyle"/>
  <element ref="mms:merror"/>
  <element ref="mms:mpadded"/>
  <element ref="mms:mphantom"/>
  <element ref="mms:mfenced"/>
 </choice>
</group>
```

```
<complexType name="PresBaseType">
 <group ref="mms:PresExpression"
     minOccurs="0" maxOccurs="unbounded"/>
 <attributeGroup ref="mms:MATHML.Common.attrib"/>
</complexType>

<complexType name="mfracType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attribute ref="mms:bevelled" use="optional"/>
   <attribute ref="mms:linethickness" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<complexType name="mencloseType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attribute name="notation" type="string" default="longdiv"/>
  </extension>
 </complexContent>
</complexType>

<complexType name="mstyleType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attributeGroup ref="mms:fontinfo"/>
   <attributeGroup ref="mms:opinfo"/>
   <attribute ref="mms:lquote use="optional""/>
   <attribute ref="mms:rquote" use="optional"/>
   <attribute ref="mms:linethickness" use="optional"/>
   <attribute ref="mms:scriptlevel" use="optional"/>
   <attribute ref="mms:scriptsizemultiplier" use="optional"/>
   <attribute ref="mms:scriptminsize" use="optional"/>
```

```
    <attribute ref="mms:background" use="optional"/>
    <attribute ref="mms:veryverythinmathspace" use="optional"/>
    <attribute ref="mms:verythinmathspace" use="optional"/>
    <attribute ref="mms:thinmathspace" use="optional"/>
    <attribute ref="mms:mediummathspace" use="optional"/>
    <attribute ref="mms:thickmathspace" use="optional"/>
    <attribute ref="mms:verythickmathspace" use="optional"/>
    <attribute ref="mms:veryverythickmathspace" use="optional"/>
    <attribute ref="mms:open" use="optional"/>
    <attribute ref="mms:close" use="optional"/>
    <attribute ref="mms:separators" use="optional"/>
    <attribute ref="mms:subscriptshift" use="optional"/>
    <attribute ref="mms:superscriptshift" use="optional"/>
    <attribute ref="mms:accentunder" use="optional"/>
    <attributeGroup ref="mms:tableinfo"/>
    <attribute ref="mms:rowspan" use="optional"/>
    <attribute ref="mms:columnspan" use="optional"/>
    <attribute ref="mms:edge" use="optional"/>
    <attribute ref="mms:actiontype" use="optional"/>
    <attribute ref="mms:selection" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<complexType name="mpaddedType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attributeGroup ref="mms:sizeinfo"/>
   <attribute ref="mms:lspace" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<complexType name="mfencedType">
```

```
 <complexContent>
  <extension base="mms:PresBaseType">
   <attribute ref="mms:open" use="optional"/>
   <attribute ref="mms:close" use="optional"/>
   <attribute ref="mms:separators" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<!-- Presentation layout schemata: scripts and limits -->
<element name="msub" type="msubType"/>
<element name="msup" type="msupType"/>
<element name="msubsup" type="msubsupType"/>
<element name="munder" type="munderType"/>
<element name="mover" type="moverType"/>
<element name="munderover" type="munderoverType"/>
<element name="mmultiscripts" type="msubsupType"/>

<group name="pscrschema">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:msub"/>
  <element ref="mms:msup"/>
  <element ref="mms:msubsup"/>
  <element ref="mms:munder"/>
  <element ref="mms:mover"/>
  <element ref="mms:munderover"/>
  <element ref="mms:mmultiscripts"/>
 </choice>
</group>

<complexType name="msubType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attribute ref="mms:subscriptshift" use="optional"/>
```

```
  </extension>
 </complexContent>
</complexType>

<complexType name="msupType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attribute ref="mms:superscriptshift" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<complexType name="msubsupType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attribute ref="mms:subscriptshift" use="optional"/>
   <attribute ref="mms:superscriptshift" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<complexType name="munderType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attribute ref="mms:accentunder" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<complexType name="moverType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attribute ref="mms:accent" use="optional"/>
  </extension>
```

```
 </complexContent>
</complexType>

<complexType name="munderoverType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attribute ref="mms:accent" use="optional"/>
   <attribute ref="mms:accentunder" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<!-- Presentation layout schemata: tables -->
<element name="mtable" type="mtableType"/>
<element name="mtr" type="mtrType"/>
<element name="mlabeledtr" type="mtrType"/>
<element name="mtd" type="mtdType"/>

<group name="ptabschema">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:mtable"/>
  <element ref="mms:mtr"/>
  <element ref="mms:mlabeledtr"/>
  <element ref="mms:mtd"/>
 </choice>
</group>

<complexType name="mtableType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attributeGroup ref="mms:tableinfo"/>
  </extension>
 </complexContent>
</complexType>
```

```
<complexType name="mtrType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attribute ref="mms:rowalign" use="optional"/>
   <attribute ref="mms:columnalign" use="optional"/>
   <attribute ref="mms:groupalign" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<complexType name="mtdType">
 <complexContent>
  <extension base="mms:PresBaseType">
   <attribute ref="mms:rowalign" use="optional"/>
   <attribute ref="mms:columnalign" use="optional"/>
   <attribute ref="mms:groupalign" use="optional"/>
   <attribute ref="mms:rowspan" use="optional"/>
   <attribute ref="mms:columnspan" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<group name="plschema">
 <choice minOccurs="0" maxOccurs="unbounded">
  <group ref="mms:pgenschema"/>
  <group ref="mms:pscrchema"/>
  <group ref="mms:ptabschema"/>
 </choice>
</group>

<!-- Presentation action schemata -->
<element name="maction">
 <complexType>
  <complexContent>
```

```
   <extension base="mms:PresBaseType">
    <attribute ref="mms:actiontype" use="optional"/>
    <attribute ref="mms:selection" use="optional"/>
   </extension>
  </complexContent>
 </complexType>
</element>

<group name="pactions">
 <sequence>
  <element ref="mms:maction"
     minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
</group>

<!-- The following for substitution into
     content constructs excludes elements that
     are not valid as expressions.
-->
<group name="PresInCont">
 <choice minOccurs="0" maxOccurs="unbounded">
  <group ref="mms:ptoken"/>
  <group ref="mms:petoken"/>
  <group ref="mms:plschema"/>
  <group ref="mms:peschema"/>
  <group ref="mms:pactions"/>
 </choice>
</group>

<!-- Presentation element group: all presentation constructs -->
<group name="Presentation">
 <choice minOccurs="0" maxOccurs="unbounded">
  <group ref="mms:ptoken"/>
  <group ref="mms:petoken"/>
```

```
    <group ref="mms:pscrechema"/>
    <group ref="mms:plschema"/>
    <group ref="mms:peschema"/>
    <group ref="mms:pactions"/>
   </choice>
  </group>

  <!-- Recursive definition for content of expressions -->
  <group name="PresExpression">
   <choice>
    <group ref="mms:Presentation"
      minOccurs="0" maxOccurs="unbounded"/>
   </choice>
  </group>
</schema>
```

5. Meta-MathML-Content.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:mms="http://www.orcca.on.ca/MathML/Schema"
        targetNamespace="http://www.orcca.on.ca/MathML/Schema"
        elementFormDefault="qualified"
        attributeFormDefault="qualified">

 <annotation>
  <documentation xml:lang="en">
   Module name: Meta-MathML-Content.xsd
   Size: 17 KB
   Meta XML schema for high-level content elements in
   MathML. The presentation elements are not included
   in the content models.
   Copyright 2002 ORCCA. All rights reserved.
  </documentation>
 </annotation>

 <include schemaLocation=
      "http://www.orcca.on.ca/
      MathML/Schema/MathMLCommonAtts.xsd"/>
 <include schemaLocation=
      "http://www.orcca.on.ca/
      MathML/Schema/low-levelContent.xsd"/>

 <!-- Content elements: symbols group-->
 <group name="c0ary">
  <choice minOccurs="0" maxOccurs="unbounded">
   <element ref="mms:integers"/>
   <element ref="mms:reals"/>
   <element ref="mms:rationals"/>
   <element ref="mms:naturalnumbers"/>
   <element ref="mms:complexes"/>
```

```
  <element ref="mms:primes"/>
  <element ref="mms:exponentiale"/>
  <element ref="mms:imaginaryi"/>
  <element ref="mms:notanumber"/>
  <element ref="mms:true"/>
  <element ref="mms:false"/>
  <element ref="mms:emptyset"/>
  <element ref="mms:pi"/>
  <element ref="mms:eulergamma"/>
  <element ref="mms:infinity"/>
 </choice>
</group>

<!-- Content elements: operators -->
<group name="cfuncop1ary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:inverse"/>
  <element ref="mms:ident"/>
  <element ref="mms:domain"/>
  <element ref="mms:codomain"/>
  <element ref="mms:image"/>
 </choice>
</group>

<group name="carithop1ary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:abs"/>
  <element ref="mms:conjugate"/>
  <element ref="mms:exp"/>
  <element ref="mms:factorial"/>
  <element ref="mms:arg"/>
  <element ref="mms:real"/>
  <element ref="mms:imaginary"/>
  <element ref="mms:floor"/>
```

```
  <element ref="mms:ceiling"/>
 </choice>
</group>

<group name="carithop1or2ary">
 <choice>
  <element ref="mms:minus" minOccurs="0" maxOccurs="unbounded"/>
 </choice>
</group>

<group name="carithop2ary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:quotient"/>
  <element ref="mms:divide"/>
  <element ref="mms:power"/>
  <element ref="mms:rem"/>
 </choice>
</group>

<group name="carithopnary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:plus"/>
  <element ref="mms:times"/>
  <element ref="mms:max"/>
  <element ref="mms:min"/>
  <element ref="mms:gcd"/>
  <element ref="mms:lcm"/>
 </choice>
</group>

<group name="carithoproot">
 <choice>
  <element ref="mms:root" minOccurs="0" maxOccurs="unbounded"/>
 </choice>
```

```
</group>

<group name="clogicopquant">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:exists"/>
  <element ref="mms:forall"/>
 </choice>
</group>

<group name="clogicopnary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:and"/>
  <element ref="mms:or"/>
  <element ref="mms:xor"/>
 </choice>
</group>

<group name="clogicop1ary">
 <choice>
  <element ref="mms:not" minOccurs="0" maxOccurs="unbounded"/>
 </choice>
</group>

<group name="clogicop2ary">
 <choice>
  <element ref="mms:implies"
     minOccurs="0" maxOccurs="unbounded"/>
 </choice>
</group>

<group name="ccalcop">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:log"/>
  <element ref="mms:int"/>
```

```
  <element ref="mms:diff"/>
  <element ref="mms:partialdiff"/>
  <element ref="mms:divergence"/>
  <element ref="mms:grad"/>
  <element ref="mms:curl"/>
  <element ref="mms:laplacian"/>
 </choice>
</group>

<group name="ccalcop1ary">
 <choice>
  <element ref="mms:ln" minOccurs="0" maxOccurs="unbounded"/>
 </choice>
</group>

<group name="csetop1ary">
 <choice>
  <element ref="mms:card" minOccurs="0" maxOccurs="unbounded"/>
 </choice>
</group>

<group name="csetop2ary">
 <choice>
  <element ref="mms:setdiff"
     minOccurs="0" maxOccurs="unbounded"/>
 </choice>
</group>

<group name="csetopnary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:union"/>
  <element ref="mms:intersect"/>
  <element ref="mms:cartesianproduct"/>
 </choice>
```

```
</group>

<group name="cseqop">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:sum"/>
  <element ref="mms:product"/>
  <element ref="mms:limit"/>
 </choice>
</group>

<group name="ctrigop">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:sin"/>
  <element ref="mms:cos"/>
  <element ref="mms:tan"/>
  <element ref="mms:sec"/>
  <element ref="mms:csc"/>
  <element ref="mms:cot"/>
  <element ref="mms:sinh"/>
  <element ref="mms:cosh"/>
  <element ref="mms:tanh"/>
  <element ref="mms:sech"/>
  <element ref="mms:csch"/>
  <element ref="mms:coth"/>
  <element ref="mms:arcsin"/>
  <element ref="mms:arccos"/>
  <element ref="mms:arctan"/>
  <element ref="mms:arccosh"/>
  <element ref="mms:arccot"/>
  <element ref="mms:arccoth"/>
  <element ref="mms:arccsc"/>
  <element ref="mms:arccsch"/>
  <element ref="mms:arcsec"/>
  <element ref="mms:arcsech"/>
```

```
  <element ref="mms:arcsinh"/>
  <element ref="mms:arctanh"/>
 </choice>
</group>

<group name="cstatopnary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:mean"/>
  <element ref="mms:sdev"/>
  <element ref="mms:variance"/>
  <element ref="mms:median"/>
  <element ref="mms:mode"/>
 </choice>
</group>

<group name="cstatopmoment">
 <choice>
  <element ref="mms:moment"/>
 </choice>
</group>

<group name="clalgop1ary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:determinant"/>
  <element ref="mms:transpose"/>
 </choice>
</group>

<group name="clalgop2ary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:vectorproduct"/>
  <element ref="mms:scalarproduct"/>
  <element ref="mms:outerproduct"/>
 </choice>
```

```
</group>

<group name="clalgopnary">
 <choice>
  <element ref="mms:selector"
     minOccurs="0" maxOccurs="unbounded"/>
 </choice>
</group>

<!-- Content elements: relations -->
<group name="cgenrel2ary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:neq"/>
  <element ref="mms:factorof"/>
 </choice>
</group>

<group name="cgenrelnary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:eq"/>
  <element ref="mms:leq"/>
  <element ref="mms:lt"/>
  <element ref="mms:geq"/>
  <element ref="mms:gt"/>
  <element ref="mms:equivalent"/>
  <element ref="mms:approx"/>
 </choice>
</group>

<group name="csetrel2ary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:in"/>
  <element ref="mms:notin"/>
  <element ref="mms:notsubset"/>
```

```
  <element ref="mms:notprsubset"/>
 </choice>
</group>

<group name="csetrelnary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:subset"/>
  <element ref="mms:prsubset"/>
 </choice>
</group>

<group name="cseqrel2ary">
 <choice>
  <element ref="mms:tendsto"
    minOccurs="0" maxOccurs="unbounded"/>
 </choice>
</group>

<complexType name="PresInContType">
  <group ref="mms:PresInCont"
    minOccurs="0" maxOccurs="unbounded"/>
</complexType>

<complexType name="sepType">
 <sequence>
  <element ref="mms:sep" minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
</complexType>

<complexType name="ciBaseType">
 <union memberTypes="mms:MathMLCharacters mms:PresInContType"/>
</complexType>

<complexType name="cnBaseType">
```

```
 <union memberTypes="mms:MathMLCharacters
     mms:sepType mms:PresInContType"/>
</complexType>

<complexType name="ctokenBaseType">
 <complexContent>
  <extension base="mms:ciBaseType">
   <attribute ref="mms:encoding"/>
   <attribute ref="mms:definition"/>
   <attribute ref="mms:type" use="optional"/>
  </extension>
 </complexContent>
</complexType>

<!-- Content elements: leaf nodes -->
<element name="ci" type="mms:ctokenBaseType"/>
<element name="csymbol" type="mms:ctokenBaseType"/>
<element name="cn">
 <complexType>
  <complexContent>
   <extension base="mms:cnBaseType">
    <attribute ref="mms:type" use="optional"/>
    <attribute ref="mms:base"/>
    <attribute ref="mms:encoding"/>
    <attribute ref="mms:definition"/>
   </extension>
  </complexContent>
 </complexType>
</element>

<group name="ctoken">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:csymbol"/>
  <element ref="mms:ci"/>
```

```
  <element ref="mms:cn"/>
 </choice>
</group>

<complexType name="ContentExpressionBaseType">
 <group ref="mms:ContentExpression"
    minOccurs="0" maxOccurs="unbounded"/>
 <attributeGroup ref="mms:MATHML.Common.attrib"/>
</complexType>

<!-- Content elements: specials -->

<element name="apply" type="mms:ContentExpressionBaseType"/>
<element name="reln" type="mms:ContentExpressionBaseType"/>
<element name="lambda" type="mms:ContentExpressionBaseType"/>

<group name="cspecial">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:apply"/>
  <element ref="mms:reln"/>
  <element ref="mms:lambda"/>
 </choice>
</group>

<!-- Content elements: others -->

<element name="condition" type="mms:ContentExpressionBaseType"/>
<element name="declare">
 <complexType>
  <complexContent>
   <extension base="mms:ContentExpressionBaseType">
    <attribute ref="mms:type" use="optional"/>
    <attribute ref="mms:scope"/>
    <attribute ref="mms:nargs"/>
```

```
    <attribute ref="mms:occurrence"/>
    <attribute ref="mms:definition"/>
    <attribute ref="mms:encoding"/>
   </extension>
  </complexContent>
 </complexType>
</element>

<group name="cother">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:condition"/>
  <element ref="mms:declare"/>
  <element ref="mms:sep"/>
 </choice>
</group>

<!-- Content elements: semantic mapping -->

<element name="semantics">
 <complexType>
  <complexContent>
   <extension base="mms:ContentExpressionBaseType">
    <attribute ref="mms:definition"/>
    <attribute ref="mms:encoding"/>
   </extension>
  </complexContent>
 </complexType>
</element>

<element name="annotation">
 <complexType>
  <complexContent>
   <extension base="mms:#PCDATA">
    <attributeGroup ref="mms:MATHML.Common.attrib"/>
```

```
    <attribute ref="mms:encoding"/>
   </extension>
  </complexContent>
 </complexType>
</element>

<element name="annotation-xml" type="anyType"/>

<group name="csemantics">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:semantics"/>
  <element ref="mms:annotation"/>
  <element ref="mms:annotation-xml"/>
 </choice>
</group>

<!-- Content elements: constructors -->

<element name="interval">
 <complexType>
  <complexContent>
   <extension base="mms:ContentExpressionBaseType">
    <attribute ref="mms:closure"/>
   </extension>
  </complexContent>
 </complexType>
</element>

<element name="set">
 <complexType>
  <complexContent>
   <extension base="mms:ContentExpressionBaseType">
    <attribute ref="mms:type" use=optional/>
   </extension>
```

```
  </complexContent>
 </complexType>
</element>

<element name="list">
 <complexType>
  <complexContent>
   <extension base="mms:ContentExpressionBaseType">
    <attribute ref="mms:order"/>
   </extension>
  </complexContent>
 </complexType>
</element>

<element name="vector" type="mms:ContentExpressionBaseType"/>
<element name="matrix" type="mms:ContentExpressionBaseType"/>
<element name="matrixrow" type="mms:ContentExpressionBaseType"/>
<element name="piecewise">
 <complexType>
  <seuence>
   <element ref="mms:piece"
    minOccurs="0" maxOccurs="unbounded"/>
   <element ref="mms:otherwise" minOccurs="0" maxOccurs="1"/>
  </sequence>
  <attributeGroup ref="mms:MATHML.Common.attrib"/>
 </complexType>
</element>
<element name="piece" type="mms:ContentExpressionBaseType"/>
<element name="otherwise" type="mms:ContentExpressionBaseType"/>

<group name="cconstructor">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:interval"/>
  <element ref="mms:list"/>
```

```
  <element ref="mms:matrix"/>
  <element ref="mms:matrixrow"/>
  <element ref="mms:set"/>
  <element ref="mms:vector"/>
  <element ref="mms:piecewise"/>
 </choice>
</group>

<element name="fn">
 <complexType>
  <complexContent>
   <extension base="mms:ContentExpressionBaseType">
    <attribute ref="mms:definition"/>
    <attribute ref="mms:encoding"/>
   </extension>
  </complexContent>
 </complexType>
</element>

<group name="cfuncopnary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:fn"/>
  <element ref="mms:compose"/>
 </choice>
</group>

<!-- Content elements: quantifiers -->
<element name="lowlimit" type="mms:ContentExpressionBaseType"/>
<element name="uplimit" type="mms:ContentExpressionBaseType"/>
<element name="bvar" type="mms:ContentExpressionBaseType"/>
<element name="degree" type="mms:ContentExpressionBaseType"/>
<element name="logbase" type="mms:ContentExpressionBaseType"/>
<element name="momentabout"
    type="mms:ContentExpressionBaseType"/>
```

```
<element name="domainofapplication"
     type="mms:ContentExpressionBaseType"/>

<group name="cquantifier">
 <choice minOccurs="0" maxOccurs="unbounded">
  <element ref="mms:lowlimit"/>
  <element ref="mms:uplimit"/>
  <element ref="mms:bvar"/>
  <element ref="mms:degree"/>
  <element ref="mms:logbase"/>
  <element ref="mms:momentabout"/>
  <element ref="mms:domainofapplication"/>
 </choice>
</group>
<!-- Operator groups -->
<group name="cop1ary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <group ref="mms:cfuncop1ary"/>
  <group ref="mms:carithop1ary"/>
  <group ref="mms:clogicop1ary"/>
  <group ref="mms:ccalcop1ary"/>
  <group ref="mms:ctrigop"/>
  <group ref="mms:clalgop1ary"/>
  <group ref="mms:csetop1ary"/>
 </choice>
</group>

<group name="cop2ary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <group ref="mms:carithop2ary"/>
  <group ref="mms:clogicop2ary"/>
  <group ref="mms:clalgop2ary"/>
  <group ref="mms:csetop2ary"/>
 </choice>
```

```
</group>

<group name="copnary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <group ref="mms:cfuncopnary"/>
  <group ref="mms:carithopnary"/>
  <group ref="mms:clogicopnary"/>
  <group ref="mms:csetopnary"/>
  <group ref="mms:cstatopnary"/>
  <group ref="mms:clalgopnary"/>
 </choice>
</group>

<group name="copmisc">
 <choice minOccurs="0" maxOccurs="unbounded">
  <group ref="mms:carithoproot"/>
  <group ref="mms:carithop1or2ary"/>
  <group ref="mms:ccalcop"/>
  <group ref="mms:cseqop"/>
  <group ref="mms:cstatopmoment"/>
  <group ref="mms:clogicopquant"/>
 </choice>
</group>

<!-- Relation groups -->
<group name="crel2ary">
 <choice minOccurs="0" maxOccurs="unbounded">
  <group ref="mms:cgenrel2ary"/>
  <group ref="mms:csetrel2ary"/>
  <group ref="mms:cseqrel2ary"/>
 </choice>
</group>

<group name="crelnary">
```

```
  <choice minOccurs="0" maxOccurs="unbounded">
   <group ref="mms:cgenrelnary"/>
   <group ref="mms:csetrelnary"/>
  </choice>
 </group>

 <!-- Content constructs: all -->
 <group name="Content">
  <choice minOccurs="0" maxOccurs="unbounded">
   <group ref="mms:ctoken"/>
   <group ref="mms:cspecial"/>
   <group ref="mms:"cother/>
   <group ref="mms:csemantics"/>
   <group ref="mms:c0ary"/>
   <group ref="mms:cconstructor"/>
   <group ref="mms:cquantifier"/>
   <group ref="mms:cop1ary"/>
   <group ref="mms:cop2ary"/>
   <group ref="mms:copnary"/>
   <group ref="mms:copmisc"/>
   <group ref="mms:crel2ary"/>
   <group ref="mms:crelnary"/>
  </choice>
 </group>

 <!-- Content constructs for substitution in presentation
structures -->
 <group name="ContInPres">
  <choice minOccurs="0" maxOccurs="unbounded">
   <element ref="mms:ci"/>
   <element ref="mms:csymbol"/>
   <element ref="mms:cn"/>
   <group ref="mms:c0ary"/>
   <element ref="mms:apply"/>
```

```
    <element ref="mms:fn"/>
    <element ref="mms:lambda"/>
    <element ref="mms:reln"/>
    <group ref="mms:cconstructor"/>
    <element ref="mms:semantics"/>
    <element ref="mms:declare"/>
   </choice>
  </group>

  <!-- Recursive definition for content of expressions -->
  <group name="ContentExpression">
   <choice>
    <group ref="mms:Content" minOccurs="0" maxOccurs="unbounded"/>
   </choice>
  </group>

</schema>
```

6. low-levelContent.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:mms="http://www.orcca.on.ca/MathML/Schema"
        targetNamespace="http://www.orcca.on.ca/MathML/Schema"
        elementFormDefault="qualified"
        attributeFormDefault="qualified">

 <annotation>
  <documentation xml:lang="en">
   Module name: low-levelContent.xsd
   Size: 9 KB
   XML schema for empty content elements in MathML.
   Copyright 2002 ORCCA. All rights reserved.
  </documentation>
 </annotation>

 <include schemaLocation=
     "http://www.orcca.on.ca/
     MathML/Schema/MathMLCommonAtts.xsd"/>

 <!-- global attribute declaration -->
 <attribute name="base" type="string" default="10"/>
 <attribute name="closure" type="string" default="closed"/>
 <attribute name="definitionURL" type="string" default=""/>
 <attribute name="encoding" type="string" default=""/>
 <attribute name="nargs" type="string" default="1"/>
 <attribute name="occurrence" type="string"
     default="function-model"/>
 <attribute name="order" type="string" default="numeric"/>
 <attribute name="scope" type="string" default="local"/>
 <attribute name="type" type="string"/>

 <element name="sep">
```

```
 <complexType>
  <attributeGroup ref="mms:MATHML.xmlns.attrib"/>
 </complexType>
</element>

<complexType name="contentBaseType">
 <attributeGroup ref="mms:MATHML.Common.attrib"/>
 <attribute ref="mms:definitionURL"/>
 <attribute ref="mms:encoding"/>
</complexType>

<element name="compose" type="mms:contentBaseType"/>

<!-- Content elements: symbols  "c0ary"-->
<element name="integers" type="mms:contentBaseType"/>
<element name="reals" type="mms:contentBaseType"/>
<element name="rationals" type="mms:contentBaseType"/>
<element name="naturalnumbers" type="mms:contentBaseType"/>
<element name="complexes" type="mms:contentBaseType"/>
<element name="primes" type="mms:contentBaseType"/>
<element name="exponentiale" type="mms:contentBaseType"/>
<element name="imaginaryi" type="mms:contentBaseType"/>
<element name="notanumber" type="mms:contentBaseType"/>
<element name="true" type="mms:contentBaseType"/>
<element name="false" type="mms:contentBaseType"/>
<element name="emptyset" type="mms:contentBaseType"/>
<element name="pi" type="mms:contentBaseType"/>
<element name="eulergamma" type="mms:contentBaseType"/>
<element name="infinity" type="mms:contentBaseType"/>

<!-- Content elements: operators  "cfuncop1ary"-->
<element name="inverse" type="mms:contentBaseType"/>
<element name="ident" type="mms:contentBaseType"/>
<element name="domain" type="mms:contentBaseType"/>
```

```
<element name="codomain" type="mms:contentBaseType"/>
<element name="image" type="mms:contentBaseType"/>

<!-- "carithop1ary" -->
<element name="abs" type="mms:contentBaseType"/>
<element name="conjugate" type="mms:contentBaseType"/>
<element name="exp" type="mms:contentBaseType"/>
<element name="factorial" type="mms:contentBaseType"/>
<element name="arg" type="mms:contentBaseType"/>
<element name="real" type="mms:contentBaseType"/>
<element name="imaginary" type="mms:contentBaseType"/>
<element name="floor" type="mms:contentBaseType"/>
<element name="ceiling" type="mms:contentBaseType"/>

<!-- "carithop1or2ary" -->
<element name="minus" type="mms:contentBaseType"/>

<!-- "carithop2ary" -->
<element name="quotient" type="mms:contentBaseType"/>
<element name="divide" type="mms:contentBaseType"/>
<element name="power" type="mms:contentBaseType"/>
<element name="rem" type="mms:contentBaseType"/>

<!-- "carithopnary" -->
<element name="plus" type="mms:contentBaseType"/>
<element name="times" type="mms:contentBaseType"/>
<element name="max" type="mms:contentBaseType"/>
<element name="min" type="mms:contentBaseType"/>
<element name="gcd" type="mms:contentBaseType"/>
<element name="lcm" type="mms:contentBaseType"/>

<!-- "carithoproot" -->
<element name="root" type="mms:contentBaseType"/>
```

```
<!-- "clogicopquant" -->
<element name="exists" type="mms:contentBaseType"/>
<element name="forall" type="mms:contentBaseType"/>

<!-- "clogicopnary" -->
<element name="and" type="mms:contentBaseType"/>
<element name="or" type="mms:contentBaseType"/>
<element name="xor" type="mms:contentBaseType"/>

<!-- "clogicop1ary" -->
<element name="not" type="mms:contentBaseType"/>

<!-- "clogicop2ary" -->
<element name="implies" type="mms:contentBaseType"/>

<!-- "ccalcop" -->
<element name="log" type="mms:contentBaseType"/>
<element name="int" type="mms:contentBaseType"/>
<element name="diff" type="mms:contentBaseType"/>
<element name="partialdiff" type="mms:contentBaseType"/>
<element name="divergence" type="mms:contentBaseType"/>
<element name="grad" type="mms:contentBaseType"/>
<element name="curl" type="mms:contentBaseType"/>
<element name="laplacian" type="mms:contentBaseType"/>

<!-- "ccalcop1ary" -->
<element name="ln" type="mms:contentBaseType"/>

<!-- "csetop1ary" -->
<element name="card" type="mms:contentBaseType"/>

<!-- "csetop2ary" -->
<element name="setdiff" type="mms:contentBaseType"/>
```

```
<!-- "csetopnary" -->
<element name="union" type="mms:contentBaseType"/>
<element name="intersect" type="mms:contentBaseType"/>
<element name="cartesianproduct" type="mms:contentBaseType"/>

<!-- "cseqop" -->
<element name="sum" type="mms:contentBaseType"/>
<element name="product" type="mms:contentBaseType"/>
<element name="limit" type="mms:contentBaseType"/>

<!-- "ctrigop" -->
<element name="sin" type="mms:contentBaseType"/>
<element name="cos" type="mms:contentBaseType"/>
<element name="tan" type="mms:contentBaseType"/>
<element name="sec" type="mms:contentBaseType"/>
<element name="csc" type="mms:contentBaseType"/>
<element name="cot" type="mms:contentBaseType"/>
<element name="sinh" type="mms:contentBaseType"/>
<element name="cosh" type="mms:contentBaseType"/>
<element name="tanh" type="mms:contentBaseType"/>
<element name="sech" type="mms:contentBaseType"/>
<element name="csch" type="mms:contentBaseType"/>
<element name="coth" type="mms:contentBaseType"/>
<element name="arcsin" type="mms:contentBaseType"/>
<element name="arccos" type="mms:contentBaseType"/>
<element name="arctan" type="mms:contentBaseType"/>
<element name="arccosh" type="mms:contentBaseType"/>
<element name="arccot" type="mms:contentBaseType"/>
<element name="arccoth" type="mms:contentBaseType"/>
<element name="arccsc" type="mms:contentBaseType"/>
<element name="arccsch" type="mms:contentBaseType"/>
<element name="arcsec" type="mms:contentBaseType"/>
<element name="arcsech" type="mms:contentBaseType"/>
<element name="arcsinh" type="mms:contentBaseType"/>
```

```
<element name="arctanh" type="mms:contentBaseType"/>

<!-- "cstatopnary" -->
<element name="mean" type="mms:contentBaseType"/>
<element name="sdev" type="mms:contentBaseType"/>
<element name="variance" type="mms:contentBaseType"/>
<element name="median" type="mms:contentBaseType"/>
<element name="mode" type="mms:contentBaseType"/>

<!-- "cstatopmoment" -->
<element name="moment" type="mms:contentBaseType"/>

<!-- "clalgop1ary" -->
<element name="determinant" type="mms:contentBaseType"/>
<element name="transpose" type="mms:contentBaseType"/>

<!-- "clalgop2ary" -->
<element name="vectorproduct" type="mms:contentBaseType"/>
<element name="scalarproduct" type="mms:contentBaseType"/>
<element name="outerproduct" type="mms:contentBaseType"/>

<!-- "clalgopnary" -->
<element name="selector" type="mms:contentBaseType"/>

<!-- Content elements: relations -->

<!-- "cgenrel2ary" -->
<element name="neq" type="mms:contentBaseType"/>
<element name="factorof" type="mms:contentBaseType"/>

<!-- "cgenrelnary" -->
<element name="eq" type="mms:contentBaseType"/>
<element name="leq" type="mms:contentBaseType"/>
<element name="lt" type="mms:contentBaseType"/>
```

```
 <element name="geq" type="mms:contentBaseType"/>
 <element name="gt" type="mms:contentBaseType"/>
 <element name="equivalent" type="mms:contentBaseType"/>
 <element name="approx" type="mms:contentBaseType"/>

 <!-- "csetrel2ary" -->
 <element name="in" type="mms:contentBaseType"/>
 <element name="notin" type="mms:contentBaseType"/>
 <element name="notsubset" type="mms:contentBaseType"/>
 <element name="notprsubset" type="mms:contentBaseType"/>

 <!-- "csetrelnary" -->
 <element name="subset" type="mms:contentBaseType"/>
 <element name="prsubset" type="mms:contentBaseType"/>

 <!-- "cseqrel2ary" -->
 <element name="tendsto" type="mms:tendstoType"/>

 <complexType name="tendstoType">
  <complexContent>
   <extension base="mms:contentBaseType"
    <attribute ref="mms:type" use="optional"/>
   </extension>
  </complexContent>
 </complexType>

</schema>
```

7. mathAtts.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:mms="http://www.orcca.on.ca/MathML/Schema"
        targetNamespace="http://www.orcca.on.ca/MathML/Schema"
        elementFormDefault="qualified"
        attributeFormDefault="qualified">

 <annotation>
  <documentation xml:lang="en">
   Module name: mathAtts.xsd
   Size: 2 KB
   XML schema for the attributes of element 'math' in MathML.
   Copyright 2002 ORCCA. All rights reserved.
  </documentation>
 </annotation>

 <include schemaLocation=
     "http://www.orcca.on.ca/
     MathML/Schema/MathMLCommonAtts.xsd"/>

 <!-- Browser interface definition -->

 <!-- Attributes for top-level element "math" -->
 <attribute name="macros" type="string"/>
 <attribute name="mode" type="string"/>
 <attribute name="display" type="string"/>

 <attributeGroup name="topinfo">
  <attributeGroup ref="mms:MATHML.Common.attrib"/>
  <attribute ref="mms:macros" use="optional"/>
  <attribute ref="mms:mode" use="optional"/>
  <attribute ref="mms:display" use="optional"/>
 </attributeGroup>
```

```
 <!-- Attributes for browser interface element -->
 <attribute name="baseline" type="string"/>
 <attribute name="overflow" default="scroll">
  <simpleType>
   <restriction base="string">
    <enumeration value="scroll"/>
    <enumeration value="elide"/>
    <enumeration value="truncate"/>
    <enumeration value="scale"/>
   </restriction>
  </simpleType>
 </attribute>
 <attribute name="altimg" type="string"/>
 <attribute name="alttext" type="string"/>

 <attributeGroup name="browif">
  <attribute ref="mms:type" use="optional"/>
  <attribute ref="mms:name" use="optional"/>
  <attribute ref="mms:height" use="optional"/>
  <attribute ref="mms:width" use="optional"/>
  <attribute ref="mms:baseline" use="optional"/>
  <attribute ref="mms:overflow"/>
  <attribute ref="mms:altimg" use="optional"/>
  <attribute ref="mms:alttext" use="optional"/>
 </attributeGroup>
</schema>
```

8. MathMLCommonAtts.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:mms="http://www.orcca.on.ca/MathML/Schema"
        targetNamespace="http://www.orcca.on.ca/MathML/Schema"
        elementFormDefault="qualified"
        attributeFormDefault="qualified">

 <annotation>
  <documentation xml:lang="en">
   Module name: MathMLCommonAtts.xsd
   Size: 2 KB
   XML schema for global attributes in MathML.
   Copyright 2002 ORCCA. All rights reserved.
  </documentation>
 </annotation>

 <attributeGroup name="MATHML.xmlns.attrib">
  <attribute name="xmlns:m" type="string"
      fixed="http://www.w3.org/1998/Math/MathML"/>
  <attribute name="xmlns:xlink" type="string"
      fixed="http://www.w3.org/1999/xlink"/>
 </attributeGroup>

 <attributeGroup name="MATHML.Common.attrib">
  <attributeGroup ref="mms:MATHML.xmlns.attrib"/>
  <attribute name="xlink:href" type="string" use="optional"/>
  <attribute name="class" type="string" use="optional"/>
  <attribute name="style" type="string" use="optional"/>
  <attribute name="id" type="ID" use="optional"/>
  <attribute name="xref" type="IDREF" use="optional"/>
  <attribute name="other" type="string" use="optional"/>
 </attributeGroup>
</schema>
```

Vita

Name: Yuzhen Xie

Place of Birth: Henan, China

Year of Birth: 1967

Post-secondary Education and Degrees:

The University of Western Ontario
London, Ontario
2002, M.Sc

The University of Western Ontario
London, Ontario
2000, B.Sc

Tsinghua University
Beijing, China
1988 B.Eng

Honors and Awards:

Special University Scholarship, 2001, 2002
Deans Honor List, 2000
2nd Prize for Scientific and Technological Achievements, 1994
1st Prize for National Academic Progress, 1933
Excellent Graduate Award, 1988

Related Work Experience:

Teaching Assistant and Research Assistant (2001 – 2002),
Ontario Research Center for Computer Algebra
London, Ontario

Software Developer (Internship, 2000 - 2001)
FAG Aerospace Canada
Stratford, Ontario