MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Normative Textual Representation of Mathematical Formulae

MASTER'S THESIS

Maroš Kucbel

Brno, 2013

Declaration

Advisor: assoc. prof. RNDr. Petr Sojka, Ph.D.

Acknowledgement

I would like to thank my supervisor assoc. prof. RNDr. Petr Sojka, Ph.D. for his guidance and advise on the topic throughout this work. I would also like to thank the members of the MIR team at the Faculty of Informatics, Masaryk University for their support and valuable inputs, as well as everyone that helped me during the work in any way.

Work that is described in this thesis was partially co-funded by the European Union through its Competitiveness and Innovation Programme (Information and Communications Technologies Policy Support Programme, "Open access to scientific information", Grant Agreement no. 250,503).

Abstract

The thesis deals with various options of representing mathematical content in electronic formats and their conversion to the plain text. Main focus is given to MathML that is described in detail. The thesis provides a list of current tools available in this field. The findings are applied to the analysis of a MathML-to-plain-text conversion tool. The result of the analysis is taken as a base of the implementation of said tool. The performance aspect of the tool plays an important role in the development process. The tool is then used for a conversion of a substantial corpus of mathematical documents, and the results of the conversion are presented.

Keywords

MathML, StAX, conversion, mathematical formulae, mathematics, MREC, mathematical operations, accessibility

Contents

Chapter 1

Introduction

Mathematics expresses most of its ideas with the help of formulae. To understand a mathematical formula, the reader needs to interpret not only individual symbols but the layout of the formula as well. In this, mathematical equations resemble two dimensional graphics more than ordinary plain text. However, working with images of mathematical formulae is rather impractical. Editing, copying, and manipulation are very difficult, if not impossible. In spite of these shortcomings of images, they were the main method of including mathematical content in Web pages. For display purposes, it is enough. For further processing of mathematical equations, it is not enough.

A new format was introduced - MathML. MathML stems from XML and is, therefore, difficult to read by humans. On the other hand, it is suitable for machine processing. Web browsers are capable of displaying MathML for human readers, and specialized software uses MathML as a format for data storage and exchange.

Now we can easily include mathematical content in Web pages or any other XML document, and display it to the user. But what happens if the user is dyslexic, blind, or otherwise visually impaired? There are multiple solutions: conversion to Braille, screen readers, or text-to-speech software. However, with a few exclusions, their support of mathematics is very slim or nonexistent. Also, search services usually work with plain text only. It is, therefore, desirable to develop a system that will convert mathematical content inputted in MathML into plain text format.

This thesis is a continuation of my Bachelor thesis, *Generování textu z MathML* [10], where a system for converting MathML data to plain text is created. The goal if the thesis is to significantly improve the developed system. The thesis first looks at the structure of MathML (Chapter 2). It then proceeds to list existing accessibility solutions (Chapter 3). Based on the analysis of the problem (Chapter 4) an implementation is created (Chapter 5). The final chapter provides a look at results of running the application on real-life data (Chapter 6).

Chapter 2

Mathematical Markup Language - MathML

World Wide Web pages and their main publishing language, HTML[1], provide many ways to present desired information to the user. However, presenting mathematics is not so easy and straightforward. There aren't any special tags in the HTML specification for including mathematical content. In most cases mathematical equations are presented in the form of an image. On one hand this approach renders the equation the same way in every web browser, on the other hand there is no way to copy the equation for further use, not to mention editing the equation.

MathML[2] was specifically created to circumvent this obstacle. It was designed to be used in web pages along with HTML. It follows that MathML is an application of XML with special set of tags used to capture the content, structure, and even presentation details of mathematical equations. We will take a closer look at how this is achieved in the following sections. In April 1998 MathML became the recommendation of the W3C working group[3] for including mathematics into web pages.

2.1 Structure of MathML

MathML as an application of XML consists of a tree of nodes. Each node is either empty, has textual value, or has a list of descendant nodes. To provide additional information, a node can have an arbitrary amount of attributes (key – value pairs). Each MathML tree has to have a root node named `math` that belongs to the MathML namespace. Also, it is important not to forget to include XML declaration and MathML doctype declaration.

MathML comes with two distinct sets of tags. For the visual form of equations there are Presentation MathML tags, while Content MathML tags

1. HyperText Markup Language
2. `http://www.w3.org/Math/`
3. The World Wide Web Consortium (W3C) is the main international standards organization for the World Wide Web

focus on the semantics and meaning of formulas. Presentation tags are being used primarily by web browsers to display mathematical expressions to the users, Content tags are important for further processing of expressions by specialized mathematical programs.

2.1.1 Presentation MathML

The main focus of the Presentation MathML is displaying the equation. For this purpose, there are around thirty elements - all starting with the prefix "m". Elements are divided into two classes, first of which is called tokens. It consist of elements that represent individual content and do not contain other nested elements. These include:

- `<mi>x</mi>` - identifiers,
- `<mn>2</mn>` - numbers,
- `<mo>+</mo>` - operators, fences, separators,
- `<mtext>`free text`</mtext>` - text.

The content of the token can of course be expressed in more than one character (`<mo>sin</mo>`) or with an XML and HTML character entities. For example, `&#62;` and `&gt;` have the same meaning as `>` - greater than. It is completely up to the users which notation they will choose.

Even though nesting other elements in tokens is not allowed, there are exceptions. For example, HTML5 allows almost any HTML inline tag inside the `mtext` element. So `<mtext><b>free</b> text</mtext>` would be rendered with the bold word free. The other class of elements is layout schemata. This collection of elements is further divided into following groups:

- General Layout:
  - `<mrow>` - general horizontal grouping,
  - `<mfrac>` - fractions and binomial numbers,
  - `<msqrt>`, `<mroot>` - radicals;
- Script and Limit - superscripts, subscripts,
- Tabular Math - tables and matrices,
- Elementary Math - notation for lower grades mathematics.

We can think of the layout schemata as a form of expression constructors, that specify the way in which sub-expression are constructed into larger expressions, starting with tokens and ending with the `math` element. Therefore, elements belonging to layout schemata class do not contain any characters, only other nested elements (layout schemata or tokens).

```
<math>
    <mrow>
        <mi>e</mi>
        <mo>=</mo>
        <mi>m</mi>
        <mo>⋆</mo>
        <msup>
            <mi>c</mi>
            <mn>2</mn>
        </msup>
    </mrow>
</math>
```

Figure 2.1: Expression $e = m \cdot c^2$ in Presentation MathML

Presentation MathML also provides over fifty attributes for fine tuning of expressions, like setting colors, dimensions, alignments, and many others.

2.1.2 Content MathML

Mathematical notation, rigorous as it is, is still not standardized around the world, and there are many different ways of writing mathematical expressions based on cultural customs. Even simple multiplication operation can be written as $x * y$, xy, $x\ times\ y$, $x \cdot y$. In many situations there's no need to actually render the expression; only the underlying meaning is important. For this reason, the Content MathML provides a framework and a markup language that capture the semantics of mathematical expressions.

The structure of the Content MathML tree is based on a very simple principle – applying an operator to sub-expressions. For example, the quotient x/y can be thought of as applying the division operator to two arguments - x and y. It follows that the cornerstone of the Content MathML is the function application element `<apply>`. The first child of the `<apply>` element signifies the operator, which can be an `<apply>` element again, and the rest of the `<apply>` element's direct descendants are arguments to which the operator is applied. Token elements `<ci>` and `<cn>` are available to represent numbers and variables respectively.

The Content MathML provides two ways of defining the operator. The first one is a set of more than 100 elements each corresponding to some mathematical operator. For example, for the addition operator there is `<plus>`, for logical xor `<xor>`, and so on. Then there is the second way - the element `<csymbol>`. This element contains a textual representation of the operator that is bound to a definition in a content dictionary referenced by either the attribute `cd` or `definitionURL`. External content dictionaries are important for communication between agents, and there exist several public repositories of mathematical definitions. Most notable is the OpenMath Society repository.

```
<math>
    <apply>
        <eq/>
        <ci>e</ci>
        <apply>
            <times/>
            <ci>m</ci>
            <apply>
                <power/>
                <ci>c</ci>
                <cn>2</cn>
            </apply>
        </apply>
    </apply>
</math>
```

Figure 2.2: Expression $e = m \cdot c^2$ in Content MathML

In the MathML 3 a subset of the Content MathML elements is defined - the Strict Content MathML. This uses only minimal, but sufficient, amount of elements. Most importantly, it only allows the usage of `<csymbol>` element for defining operators. The Strict Content MathML is designed to be compatible with the OpenMath standard.

```
<math>
    <apply>
        <csymbol cd="dict">equals</csymbol>
        <ci type="real">e</ci>
        <apply>
            <csymbol cd="dict">times</csymbol>
            <ci type="real">m</ci>
            <apply>
                <csymbol cd="dict">power</csymbol>
                <ci type="real">c</ci>
                <cn type="integer">2</cn>
            </apply>
        </apply>
    </apply>
</math>
```

Figure 2.3: Expression $e = m \cdot c^2$ in Strict Content MathML

2.2 Creating MathML Markup

Creating MathML documents is very simple. All that is needed is a word processor available on every operating system. However, this approach is prone to errors, be it the wrong letter case or unclosed tags. Since MathML is an XML language, the usage of some sophisticated word processor that supports tags highlighting, code completion, and XML validation will greatly improve the efficiency of creating MathML documents. But the whole process is still very time consuming and impractical. As is the case with most XML documents, even writing simple mathematical expressions takes up a lot of space and time. Fortunately, there are many dedicated MathML editors that provide some degree of abstraction from the actual MathML markup.

Multiple software products designated for working with mathematical expressions provide an option to output the results of calculations in MathML format. These include a popular web service Wolfram Alpha, Mathematica, Maple, or Matlab. MathML can also be used as a data exchange format or an input format for aforementioned programs.

Another way of creating MathML documents is a conversion from different formats. Among scientists, mathematicians particularly, TEX is the format of choice. It then comes as no surprise that there are many sources of mathematical texts written in TEX. However, not every publication comes with the source files. Often only print-ready PDF files are released [3]. Also many academic writings, especially older ones, are available only in printed form. These have to be scanned using an Optical Character Recognition (OCR) software.

2.2.1 LATEXML

The lack of a suitable tool for converting LATEX to XML prompted the participants of the Digital Library of Mathematical Functions project to develop their own solution. Main goals of LATEXML [12] design contain the aspiration to faithfully emulate TEX behavior, the ease of extensibility, and the preservation of both semantic and presentation aspects of original documents. To this end LATEXML provides two main commands - `latexml` and `latexmlpost`. The `latexml` command converts the initial TEX document to the XML format based on a set of LATEXML-bindings files that define the mapping of TEX macros to XML. `latexmlpost` then processes resulting XML output by converting mathematics and graphics, cross-referencing, and applying an `XSLT`[4] stylesheet.

Both commands come with a multitude of parameters that allow users to customize the whole process, such as loading user-defined bindings or choosing the output format. Based on the requested output format mathematics is converted to graphics (PNG images for HTML) or Presentation MathML in case of XHTML or HTML5.

LATEXML is freely available online as an installation package for Linux systems as well as Windows and MacOS. An extensive and detailed manual is available online or as a PDF document.

2.2.2 Tralics

Tralics [6, 7] is a freeware software designed to translate LATEX sources into XML documents that can be further converted into either PDF or HTML. The generated XML is conforming to the local ad-hoc DTD (a simplification of the TEI DTD [5]) with mathematical formulas conforming to the Presentation MathML 2.0 recommendations.

Similarly to LATEXML, Tralics provides many ways for customization of resulting documents, among them the possibility to change element and attribute names in the XML file. Besides Presentation MathML, mathematical

4. Extensible Stylesheet Language Transformations `http://www.w3.org/TR/xslt`
5. Text Encoding Initiative `http://www.tei-c.org/index.xml`

formulae can be translated into LaTeX-like elements as well.

Tralics is readily available online in the form of source files or binaries for Linux, Windows, and MacOS operating systems. An extensive documentation regarding customization and usage is also available online.

2.2.3 MaxTract

MaxTract [4] is a tool that through spacial analysis of symbols and fonts in PDF document reverse-engineers its source files in the form of LaTeX or XHTML + MathML documents. The conversion process requires valid PDF files to work correctly as it needs the information about symbols, font encoding, and width of objects contained in the PDF file. PDF documents created via LaTeX fulfill these requirements. As a result, MaxTract is able to process most of the scientific and mathematical material.

2.2.4 INFTY Reader

INFTY [19] is an Optical Character Recognition (OCR) software especially created for mathematical documents. INFTY reads scanned pages (images) and yields their character recognition results in multiple formats, including LaTeX and MathML. It does so in four steps: layout analysis, character recognition, structure analysis of mathematical expressions, and manual error correction (optional).

The most important phase - character recognition - is responsible for distinguishing mathematical expressions and running a character recognition engine originally developed for mathematical symbols together with non-specialized commercial OCR engine.

INFTY Reader is available free of charge for limited amount of time, then it is necessary to purchase a license key.

2.2.5 TeX4ht

TeX4ht [8] is a system for producing structured output in a markup language from sources written in the TeX-based family of languages. It is highly extensible and configurable; most common configurations include the conversion from LaTeX to HTML with MathML, Braille, or DocBook targets.

The conversion is invoked by running the `htlatex` command. To the user the process seems similar to producing standard DVI or PDF outputs. TeX4ht uses hooks within LaTeX constructs and associates configurations to them. By modifying default configuration files, the user can change the resulting HTML document to his liking. TEX4ht comes with the support of Cascading style sheets[6], which provides further possibilities of customizing the output files.

2.3 Canonicalization of MathML

With a wide variety of options for creating MathML documents, it is necessary to modify and unify input documents in such a way that will decrease the number of possible ambiguities (especially in the Presentation MathML); in the best case completely eliminating all ambiguities.

The team standing behind the Universal Maths Conversion library (UMCL) [1] proposed a notion of a unified structure - Canonical MathML [2]. Canonical MathML is valid Presentation MathML and recommends a set of rules that should be followed to make Presentation MathML documents unambiguous. A correct use of `msup` (`msub`) for superscripts (subscripts); parentheses defined in the `mo` element (`mfenced` element is not used); or unified way of writing summations, integrals, and products to name a few.

UMCL incorporates this notion and provides a module for converting Presentation MathML markup into Canonical MathML. The canonicalization module employs the use of XSL stylesheets and transformations to create the desired Canonical MathML. However, XSL Transformations tend to be quite slow, and the UMCL canonicalization is rather error-prone, and sometimes even changes the semantics of mathematical formulae, as was shown in M. Jarmar's thesis [9, chapter 5].

These shortcoming of UMCL prompted a team at Masaryk University to design and implement their own canonicalization solution [5] for use in the (Web)MIaS project [15]. Although the implementation of this solution is still under development, the core functionality is already working, and provides better results than UMCL (from the point of performance as well as the structure of converted documents).

6. `http://www.w3schools.com/css/`

Chapter 3

Status Quo of Current Tools

Making mathematical content available to visually impaired users or those with dyslexia has been a focus of many projects and researches over the last two decades. As a result several products (commercial or open source) have been developed that help the users with working with mathematical formulae. By providing text-to-speech solutions, easy to use and comfortable editors, or support for browsing and searching documents that contain mathematical equations.

In this chapter we will present some solutions that are unique in this field, and had impact on another similar products - either by the theoretical research that fuels such solutions or by general aspects that make them stand out.

3.1 DAISY

DAISY[1] (Digital Accessible Information System) is a technical standard for digital audio books, periodicals, and computerized text. It is specifically intended for people that have problems reading a printed text including blindness, dyslexia, or impaired vision. DAISY is based on combination of XML and MP3, and has various functions that traditional audio books do not provide. These include: line by line navigation, searching in the text, adding bookmarks, or adjusting the speed of the speech. It also provides support for embedded content, such as images, graphics, and MathML.

DAISY books (books conforming to the DAISY standard) can be listened to on designated DAISY readers, computers with installed DAISY software, mobile phones, or MP3 players. There are many implementation of software players - commercial, free, or even open source. They range from standalone applications to addons for internet browsers (especially Mozilla Firefox). Some software players are capable of displaying HTML pages with embedded MathML markup, and subsequently read the math-

1. `http://www.daisy.org/daisy-standard`

ematical expressions (this functionality was tested on a commercial product Dolphin EasyReader that supports Czech language as well).

3.2 MathPlayer

MathPlayer is an application developed by Design Science[2] that enables users of Microsoft Internet Explorer web browser to display mathematical equations written in MathML markup. It uses Microsoft's internal HTML engine (MSHTML) on which Internet Explorer is based. Also, for any application that makes use of MSHTML to display formatted content MathPlayer is able to display MathML content. This may include email clients, alternative browsers, or RSS readers.

Besides displaying MathML in the browser, MathPlayer comes equipped with a wide range of MathML-related functions. It enables equations to be copied to the clipboard as MathML markup and pasted to any MathML-aware software - be it simple text editor or more sophisticated application. Drag-and-drop functionality is also supported. Another important feature of the MathPlayer is the ability to speak expressions on the web page.

MathPlayer is shipped as a free to use software with English localization only but is still a close source product. The license limits its use to only computers owned by the user and also prohibits translation and reverse engineering of the application.

3.3 AsTeR

AsTeR (Audio System for Technical Readings) is an exceptional piece of software developed by T.V. Raman as a part of his dissertation at Cornell University [13]. AsTeR is capable of producing audio renderings of technical documents, even those containing higher mathematical expressions, written in $\TeX$ or $\LaTeX$. However, processing different markup languages does not pose a problem for AsTeR. All that is required is a recognizer for given markup language.

The logical structure of the document is transformed to an internal representation, which is then rendered in audio using a collection of rendering rules. This enables AsTeR to provide different views of the document. A user can listen to the whole document or select a portion of the document for listening. An important feature of AsTeR makes us of the voice intona-

2. `http://www.dessci.com/en/products/mathplayer/`

tion to read long and complicated formuale. Such formulae are divided into logical parts (like a single summation or a numerator of a fraction) and read with different intonation than the surrounding part of the expression.

3.4 MathTalk

MathTalk [17] is a commercial text-to-speech software that enables visually impaired people to read algebra notation in a quick manner. MathTalk was designed and created as a part of doctoral dissertation of R. D. Stevens [18] and employs the use of prosody in the synthetic voice to decrease the mental workload of the listener. MathTalk enables the users to browse the document and change the speed of reading, which gives them comfortable control of the reading process.

3.5 Lambda

Lambda [14] (Linear Access to Mathematic for Braille Device and Audiosynthesis) aims at solving the problem of mathematics text management by blind users. It consists of two sections: the Lambda code and the editor.

The Lambda code directly derives from MathML. It is automatically convertible into equivalent MathML and through it into the most popular mathematical formats. The Lambda code was designed to:

- have explicit meaning with no ambiguities,
- provide full Braille output of mathematical equations,
- preserve peculiarities of national Braille,
- have a compact linear representation (minimize the movement while reading Braille).

The Lambda editor allows to write and manipulate mathematical expressions. It is essentially a text editor designed to read and write Lambda code. Mathematical elements are grouped into blocks that can be collapsed or expanded for easier reading. Blocks can also be deleted or copied to a different place in the document. The output of the Lambda editor can be written (displayed) in Braille, spoken by speech synthesis, or both at the same time. The speech synthesis exploits the block structure, which enables it to change the speech speed and insert brakes to better communicate the meaning of the mathematical formula.

3.6 Web Browser Support

Strictly speaking, web browsers by themselves are not considered an accessibility software, but they play a key role in sharing information and making mathematical content available to many people across the globe.

MathML was designed as a standard for including mathematical formulae in web pages. It stands to reason that the most well-known and widespread web browsers should have good support of MathML - at least the Presentation MathML. Unfortunately the situation is not that good; developers of these browsers are mainly focused on different areas, and MathML stands very low on the ladder of priorities.

Mozilla Firefox Firefox has the best native support for MathML rendering out of the most popular web browsers and is capable of displaying most elements of the Presentation MathML. Firefox can be enhanced by an extension called FireMath[3] that adds a rich MathML editor capable of generating complicated mathematical expressions quite easily and right into MathML markup.

Google Chrome and Safari Both Google Chrome as well as Safari are based on the WebKit layout engine that has a development version of MathML. The support for MathML is available in latest versions of Safari (since version 5.1). Unfortunately, Chrome does not have native support for MathML but uses the combination of HTML and CSS to render mathematical expression entered in MathML.

Internet Explorer Internet Explorer does not have any native support for MathML, but MathML rendering capability and some additional functionality can be added by the MathPlayer extension that is described in more detail in Section 3.2. However, MathPlayer's support only extends to Internet Explorer 8; support in version 9 is rather spotty, and in version 10 it does not work at all.

Opera Opera was one of the first browsers to include native support for MathML. The rendering of MathML is not as good as in Firefox - Opera has issues with the positioning of elements in more complicated constructs.

3. `http://www.firemath.info`

Browsers for Hand-held Devices At the time of writing of this thesis, none of the browsers for hand-held devices has a satisfactory native support for MathML. However, they are capable of rendering expressions using CSS.

Chapter 4

Analysis and Solution Proposal

The goal of this thesis is the development of a conversion tool that will be able to process an XML document or documents and transform each occurrence of the MathML markup into the appropriate textual representation. For example, $5 \cdot \alpha = x + 3$ should be transformed into "five times alpha equals x plus three". In case of a visual (not semantic) change of the operator, the tool should produce the same result. In our example $5 \cdot \alpha = x + 3$, $5 * \alpha = x + 3$ and $5\alpha = x + 3$ have the same meaning, so all of them should be transformed into identical strings. For better results, the input can be first canonicalized (as described in Section 2.3). From these requirements we can devise a basic top-level workflow diagram of the application (Figure 4.1). All other data outside `math` elements contained in input document has to be preserved and copied verbatim to the output.

This thesis is a continuation of my Bachelor thesis [10] called *Generování textu z MathML*. The bachelor thesis introduces a program for generating plain text from MathML, Math2Text, as described in the paragraph above. However, it has serious limitations when it comes to scalability, supports only a subset of the Presentation MathML elements, and a limited amount of mathematical operations. The program uses XSL Transformations to convert MathML markup into the plain text format. In this thesis we will try to improve the computation speed, provide better scalability, support a greater amount of mathematical operations, and encompass most of the Presentation MathML as well as the Content MathML elements.

Before input documents are loaded into an in-memory structure for representing XML documents, a preprocessing phase will take place. This includes determining the file structure on the hard-drive, and retrieving all files from the input document and all its descendants. This structure will be preserved, and output documents will have the same structure - just in the user-specified directory. Also, input files can be packed using the .zip archive file format (as is the case with documents in the MREC 6.1), and such files have to be unpacked before further processing. Then, if requested, we will canonicalize input files. Since the canonicalization comes from ex-

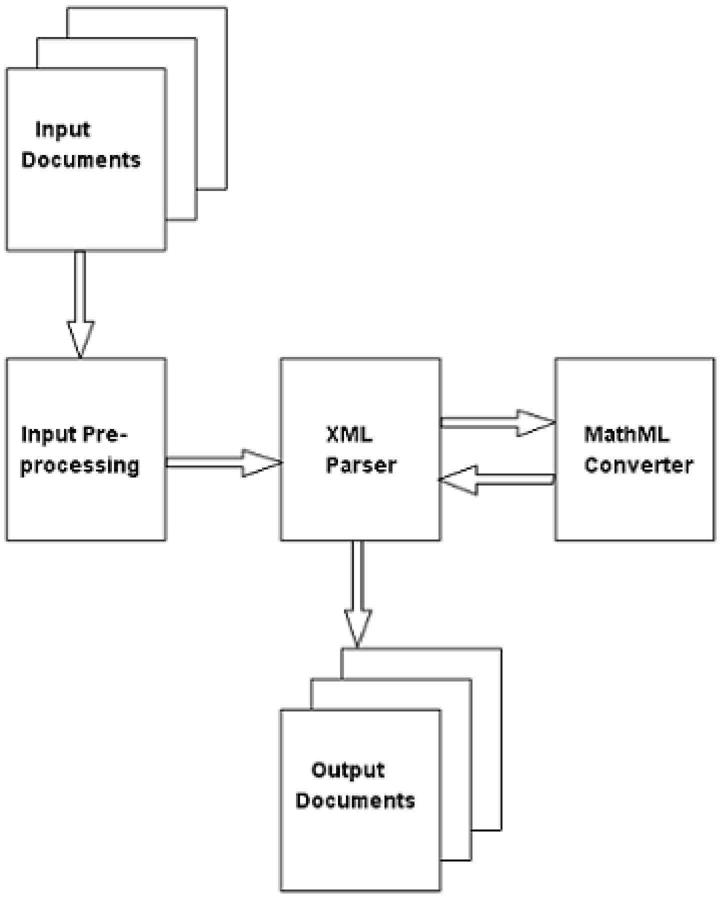

Figure 4.1: Basic diagram of application workflow

ternal module (library), we have to be careful of possible errors or the time efficiency of this process.

One important factor about the input documents that was not taken into account is their quantity. The application must be able to process large corpora of mathematical documents in reasonable time and with efficient memory usage. Fortunately, modern computation systems provide a way of running the application in multiple threads. The application just has to ensure that all threads work with the same settings. This can be accomplished by providing a single point of entry to the settings instance. A globally visible class (object) that implements the Singleton design pattern is the best solution for accessing settings across the whole application. Also, retrieving external resources for each single thread is very time consuming.

Therefore, an in-memory cache of resources common for all threads will be implemented.

We can now create a more complex and detailed diagram of internal workings and dataflows inside the application (see Figure 4.2).

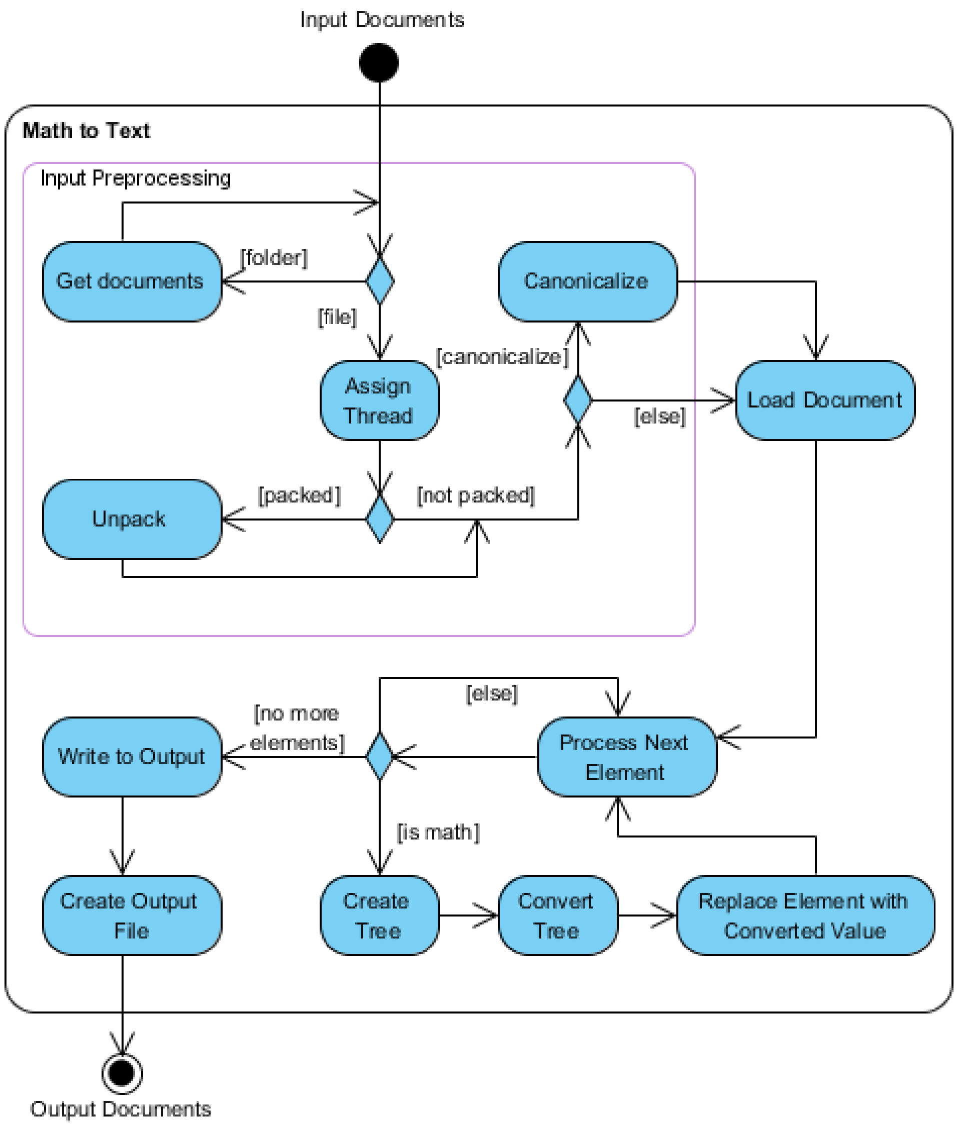

Figure 4.2: Activity diagram of the application

Three activities (processes) still remain to be defined:

1. Load Document,
2. Create Tree,
3. Convert Tree.

4.1 Load Document

When it comes to processing XML documents, there are four common approaches:

1. Document Object Model[1] (DOM) retains the tree structure of the XML document and is composed of nodes. Node represents elements in the XML structure and form a hierarchical structure.
2. Extensible Stylesheet Language[2] (XSL) refers to a family of languages used to transform and render XML documents.
3. Streaming XML is an event-based, sequential, and unidirectional approach of processing XML documents.
4. XML data binding is a process of mapping XML documents to objects in computer memory (deserialization).

	Processing speed	Memory requirements
DOM	slow	high
XSL	slow	high
Streaming XML	fast	low
Data binding	average	high

Table 4.1: XML processing options comparison

The memory requirements shown in Table 4.1 stem from the maximum amount of data that each method uses to process documents. For Streaming XML method, which reads only one event at a time and retains only minimal processing information, these requirements are low. The rest of compared methods need to load the whole document into computer memory,

1. `http://www.w3.org/DOM/DOMTR`
2. `http://www.w3.org/Style/XSL/`

hence their memory requirements are high. The comparison of processing speeds comes from the experience of the author with using these methods.

Based on the comparison in Table 4.1 we choose the fastest and most memory-efficient method for processing our input documents - the Streaming XML method. Algorithm 1 shows a way to process input documents using an in-memory tree structure defined in Section 4.2.

Algorithm 1 Process input algorithm

```
procedure ProcessInput
    clear tree                                        ▷ a DOM-like MathML tree
    while next event exists do
        event ← next event
        if event is a start of math element then
            tree ← create a new tree
        else if event is an end of math element then
            convert tree and write the result to the output
            clear tree
        else if event is inside math element then
            insert a new node, value or attribute in the tree
        else
            write event to the output
        end if
    end while
end procedure
```

4.2 Create Tree

The tree in this case means an in-memory representation of MathML with a tree structure. DOM seems like a good option for this application; however, a fully-fledged DOM representation of the MathML markup is not required. The information this model provides takes up a lot of resources (time and memory) and a big part of it would be discarded. However, we need a structure that will provide a comfortable traversal - moving from parent to children and vice versa.

We have designed a simplified DOM tree - with just the information we actually need. Since we only use it to build a tree representation of MathML markup, every node in this tree has a special property that denotes its type - name of the element and part of MathML it belongs to (Presentation or Content).

Besides the type property, our simplified model contains a list of children, pointer to the parent, a text value, a list of XML attributes (key-value pairs), and a attribute that signifies whether this node has already been processed (we want to process each node only once).

Our very simple tree has one more advantage besides simplicity - it allows the application to provide an option to change the method of loading XML documents.

4.3 Convert Tree

We are starting the conversion process with our tree representation of the MathML markup. At the beginning of the conversion, we need to determine whether the tree consist of only the Presentation markup, Content markup, or both, since each requires a slightly different approach to the conversion. In most cases, the Presentation MathML resides directly in the `math` tree (is a direct descendant of the `math` element) or in the element `semantics`, while the Content MathML is often found enclosed in the `annotation-xml` element with an attribute `encoding` set to the value `MathML-Content`. Or, we can simply traverse the tree till we find an element from either the Presentation or Content MathML markup and continue based on our finding.

The conversion process starts at the top level element, `math`, and then recursively continues to traverse the tree converting the Presentation and Content elements based on their own specifics. Each converted node is marked as processed in order not to be processed twice. Since in some cases the conversion requires to look ahead and jump out of the recursion pattern to process a sibling node (or any other node).

The basic outline of the conversion process can be seen in Figure 4.3 and will be described in detail in the next sections.

4.3.1 Presentation MathML

Every element of the Presentation MathML requires an individual approach to the processing. But there are still groups of elements with similar characteristics.

The first group consists of token elements. The conversion is straightforward and depends only on the value of the element:

- `mn` - convert number to string or take the original value,

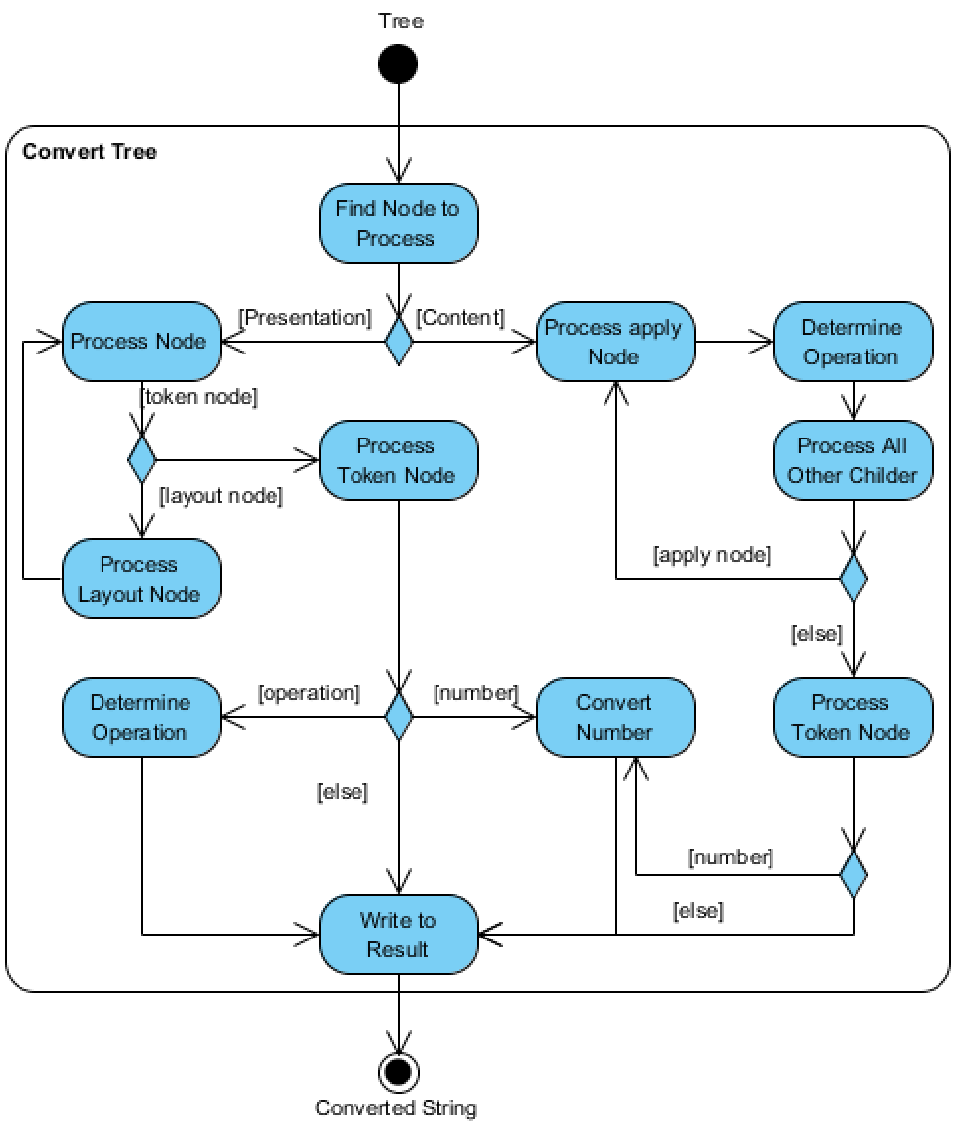

Figure 4.3: Tree conversion process

- `mi` - take the original value if it is in Latin script, otherwise convert the value if possible (e.g. Greek alphabet),
- `mo` - determine the operation defined by the `mo` element value ($\Sigma \rightarrow$ sum, $\int \rightarrow$ integral, ...).

It is important to remember that one mathematical operation can be expressed with various symbols, and the conversion has to unify them into the same string representation.

The second group contains elements with a specific purpose. In other words, these elements clearly state (with their names) what is their intended function. The conversion uses this fact and is therefore very simple. For example:

- `mfrac` - fractions or binomial numbers,
- `msqrt` - square root,
- `mfenced` - subexpression is enclosed in parenthesis.

The third group is composed of elements that specify only the layout of the subexpression. To determine the exact function the author wanted to convey, we need additional information provided by child elements, and in some cases it may also depend on sibling or parent elements.

```
<math>
    <munder>
        <mo>lim</mo>
        <mrow>
            <mi>x</mi>
            <mo>&rarr;</mo>
            <mn>0</mn>
        </mrow>
    </munder>
    <mrow>
        <msqrt>
            <mi>x</mi>
        </msqrt>
    </mrow>
</math>
```

Figure 4.4: Expression $\lim_{x \to 0} \sqrt{x}$ in Presentation MathML

As we can see in Figure 4.4, the `munder` element has to be interpreted in the context of its child elements (especially the first child) and its first sibling element. Similar rules apply to elements `mover`, `munderover`, `msub`, `msup`, and `msubsup` that describe various mathematical constructs, such as summations, integrals, products, logarithms, and many more.

The last group is formed by elements that have a purely presentation function: spacing, padding, and so on. These can be ignored altogether, since they do not provide any information about the meaning of presented expressions.

4.3.2 Content MathML

The cornerstone of the Content MathML is without a doubt the element `apply` - the function application. It describes the application of its first child element on the rest of child elements - all of which can be `apply` elements themselves. The `apply` element will, therefore, serve as a hub - assigning the processing of expression or subexpression based on the operator (the first child element).

To be able to convert the expression correctly, we need a different approach than for the Presentation MathML, where the elements are ordered for display purposes and can be converted basically from top to bottom (with a few exceptions). In the Content MathML a function can be applied to multiple elements, but it will be declared only once as seen in Figure 4.5 (in the Presentation MathML the plus symbol would be declared twice - between x and y, y and z).

```
<math>
    <apply>
        <plus/>
        <ci>x</ci>
        <ci>y</ci>
        <ci>z</ci>
    </apply>
</math>
```

Figure 4.5: Expression $x + y + z$ in Content MathML

Therefore, we can not determine word order of the expression from the position of elements in the document (as is the case in the Presentation MathML), but we have to specify the desired word order for each operation separately. Fortunately, operations can be sorted into logical groups based on the word order we use when presenting them.

- Infix form - the operator is used between pairs of inputs - starting with the first and the second input, then the second and the third and so on.

These include operators for division ($x/y/z$), multiplication ($x \cdot y \cdot z$), comparison ($x = y = z, \geq, <$), and many more.

- Prefix form - the operator is used at the beginning; preceding inputs. In this case, the operator is used only once at the beginning followed by converted inputs. Examples include absolute value ($|x|$), negation ($\neg$), or floor ($\lfloor x \rfloor$).

- Prefix form with multiple inputs - a special case of the prefix form, where there are multiple inputs. In this instance, the operator is also used only once, but inputs have to be divided by commas, and the last two inputs divided by the word "and". Function $min(x, y, z)$ should be converted to string: minimum of x, y and z. Another examples might be sets, lists, functions greatest common divisor, maximum, or lowest common multiple.

- Quotient and remainder - $rem(x, y)$ should be converted to: reminder of x divided by y. Similarly the quotient operator.

- Others - operators that do not belong to any of abovementioned categories or belong to more than one (like plus or minus, which can be used in both the prefix and infix form) have to be treated in a separate way.

4.3.3 Operators and Symbols

As we mentioned before, many mathematical operations can be expressed using more than one operator or symbol. Our task lies in identifying operators and symbols belonging to each operation, creating a storage structure for this data, and developing a method or methods for searching in this structure, i.e., finding an operation for a given operator.

This way, all operators that denote the same operation are grouped together, and we can easily unify the way each operation is converted.

4.3.4 Adding Parentheses

Imagine someone read you the following sentence: "x to the power of three plus y". What equation would you imagine? Is it $x^3 + y$ or x^{3+y}? Lets assume the original equation was the latter, x^{3+y}. In the MathML markup this ambiguity does not exist - as can be seen in Figure 4.6. However, the Presentation MathML relies on the rendering of the equation and the ability of the user to visually distinguish between the two possibilities (standard font vs superscript), and so does not need to explicitly include parentheses.

The Content MathML is not predominantly intended to be read by human users, and, therefore, it does not provide explicit parentheses.

```
<math>
    <mrow>
        <msup>
            <mi>x</mi>
            <mrow>
                <mn>3</mn>
                <mo>+</mo>
                <mi>y</mi>
            </mrow>
        </msup>
    </mrow>
    <annotation-xml encoding="MathML-Content">
        <apply>
            <power/>
            <ci>x</ci>
            <apply>
                <plus/>
                <cn>3</cn>
                <ci>y</ci>
            </apply>
        </apply>
    </annotation-xml>
</math>
```

Figure 4.6: Expression x^{3+y} in Presentation and Content MathML

Since the conversion results is just a plain text with no visual aid to determine what the original equation was expressing, we have to enclose parts of equations that logically belong together and can be ambiguous with parentheses ourselves.

In the Presentation MathML logical parts of equations reside inside a single `mrow` element. So the straightforward solution is to prepend and append parentheses to the `mrow` element content. However, we do not always need (or want) parentheses in this place, like in the case of the `msup` element in Figure 4.6. The easiest, and in most cases sufficient fix, is to add parentheses only if the `mrow` element has at least two child elements.

In the Content MathML the element `apply` acts as a grouping element.

Similarly to the `mrow` element, we do not need to add parentheses every time. In Figure 4.6 the first `apply` element does not need to use parentheses - only the second one does. For deciding whether to add parentheses or not, we will use the division of operations introduced in Section 4.3.2. Only operators belonging to the *infix* form group will add parentheses.

The resulting string procured from the conversion of equation x^{3+y} should be: "x to the power of open braces three plus y close braces".

4.3.5 Localization

Every string that is outputted has to be localized to the user specified language. The exception are token elements that are copied verbatim. There is a problem with different word order in different languages. We will, however, work with just Germanic and Slavic languages that have very similar word order (as opposed to Japanese, which puts verb at the end of the sentence). We can, therefore, work with a simple word substitution for localization and don't have to occupy ourselves with diverse word orders.

Chapter 5

Implementation

This chapter describes various implementation aspects that occurred during the creation of the conversion application. The application is written in the Java programming language - mainly because of personal preference of the author. Also, the existence and availability of many frameworks and tools for Java language (for working with XML documents among others) are big advantages of using this language.

The application uses Apache Maven for build automation, distribution management, and dependency management.

The application provides a command line interface and can be customized by setting input parameters (see Appendix B for the enumeration of arguments).

The structure of the application can be seen on the component diagram 5.1 below.

5.1 DOM-like Representation of MathML

The implementation of the tree designed in Section 4.2 can be seen in Figure 5.2.

5.2 Input Processing

As we outlined in Section 4.1, the streaming approach with custom DOM-like internal representation of the MathML markup is used.

In Java, two major APIs[1] for streaming processing of XML documents are popular among developers:

- SAX (Simple API for XML) implements the push principle - the reporting of events as they are encountered.

1. Application programming interface

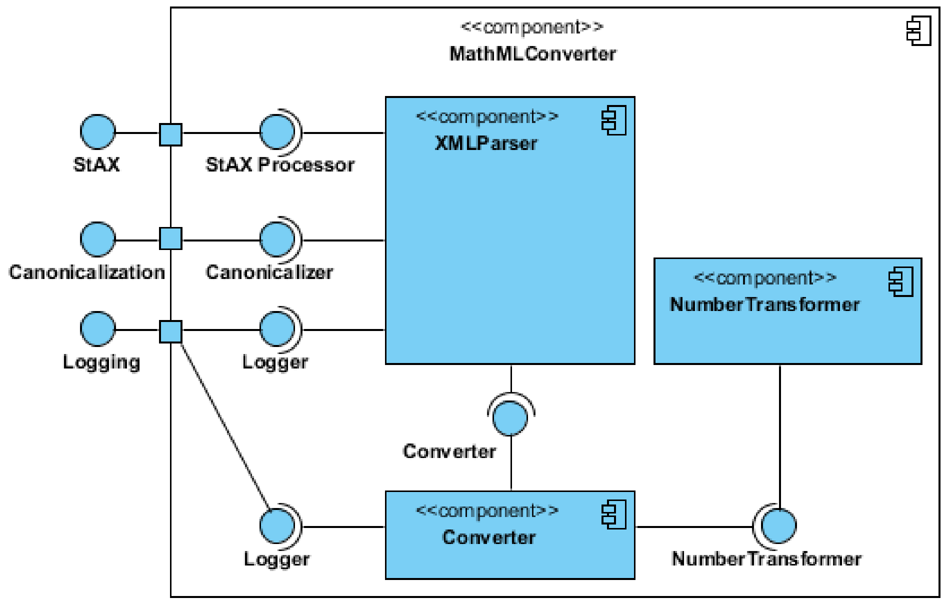

Figure 5.1: Component diagram of the application

- StAX (Streaming API for XML) - the pull principle. The application requests events from the StAX processor. (StAX is a specification defined by the JSR 173[2].)

This application uses default StAX implementation that is shipped with Java Standard Edition 6 runtime, but there are also other implementations available, such as Woodstox[3] or Aalto[4]. There is a possibility to request one of these two implementations to use instead of the default implementation.

StAX offers two ways of traversing documents: Cursor API and Iterator API. We use the first, Cursor API, because it should be faster and more memory-efficient[5]. Two main interfaces are available in the Cursor API (Figure 5.3): `XMLStremReader` for accessing all possible information retrievable from XML documents and `XMLStreamWriter` which in turn provides methods for outputting this information.

2. `http://www.jcp.org/en/jsr/detail?id=173`
3. `http://woodstox.codehaus.org`
4. `http://wiki.fasterxml.com/AaltoHome`
5. `http://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/SJSXP3.html`

```
public final class MathMLNode {
   /**
    * Type of this node.
    */
   private MathMLElement type;
   /**
    * List of all children of this node. Empty if
    * there are none. In this case value must be
    * set.
    */
   private List<MathMLNode> children = new
      ArrayList<MathMLNode>();
   /**
    * Text value of this node. {@code null} if
    * there are some child nodes.
    */
   private String value;
   /**
    * Parent node.
    */
   private MathMLNode parent;
   /**
    * Was this node already processed? Useful when
    * you have to "look ahead" and process
    * element sooner.
    */
   private boolean processed = false;
   /**
    * Set of attributes.
    */
   private Set<XmlAttribute> attributes = new
      HashSet<XmlAttribute>();

   ... getters
   ... setters
```

Figure 5.2: Simplified DOM for internal representation of MathML markup

```
public interface XMLStreamReader {
    public int next();
    public boolean hasNext();
    public String getText();
    public String getLocalName();
    public String getNamespaceURI();
    ...
}

public interface XMLStreamWriter {
    public void writeStartElement(String localName)
        ;
    public void writeEndElement();
    public void writeCharacters(String text);
    ...
}
```

Figure 5.3: Example of methods in `XMLStreamReader` and `XMLStreamWriter` interfaces of the Cursor API

5.2.1 XML Parser

The implementation of the `XmlParser` interface - `XmlParserStAX` - is responsible for processing input XML documents using the StAX Cursor API, and at the same time creating output documents in the same run through the file. The internal processing follows steps outlined in the Algorithm 1. The parser also provides a method for processing a string input and producing a string output. In this case, the parser processes the document defined by this string the same way as any other input document. However, the output contains only converted strings - without any of original documents elements outside the MathML markup.

The parser is capable of accepting a folder as an input (instead of a document) and processing all files in this folder, while preserving the file structure on the path entered by the user.

A thread pool with a user-specified size is initialized at the beginning of the conversion process, and for every file from the input a `java.util.concurrent.Callable` instance is created. Using the `java.util.concurrent.ExecutorService` with our thread pool, these callables are concurrently invoked (whenever there is a free thread in

```
public class XmlParserStAX implements XmlParser {
    public String parse(final String inputString,
       final Locale language);
    public File parse(final File file, final Locale
        language);
    public List<File> parse(final List<File> files,
        final Locale language);
}
```

Figure 5.4: Public methods of `XmlParserStAX` class

the thread pool, the next callable is invoked).

A JDOM[6] implementation of the `XmlParser` is also provided and will be used to compare the performance of StAX implementation in Section 6.2.

5.3 Conversion

When the parser creates a complete tree, it immediately initializes the conversion of that tree by calling the `convert(tree, language)` method of the `MathMLConverter` class. The converter then decides which element of the input tree will be used as the root element of the conversion. The Presentation MathML markup has precedence over the Content markup, unless the opposite was requested by the user.

The converter consists of several classes that represent elements of MathML markup - a separate class for each element of the Presentation MathML markup and three classes (`Apply`, `Cn`, `Ci`) for the Content MathML markup. Every class has only one static method, `process(node, settings)` that is called whenever the traversal of the input tree encounters corresponding element. The run through the tree is done via the class `Node` and its static method `process(node, settings)`. This method works as a switch for every type of node and delegates the processing to a specific class. It also marks each encountered node as processed (see Figure 5.5 for illustration). The resulting string is gradually built from partial results of individual nodes.

Before we get to the explanation of the conversion inside these classes, we have to define how mathematical operators and operations are defined in the application. The `Operation` class is an enumeration of all operations

6. `http://www.jdom.org`

```
public class Node {
    public static String process(final MathMLNode
        node, final ConverterSettings settings) {
         final StringBuilder builder = new
            StringBuilder();
       switch (node.getType()) {
          ...
          case MN: {
             builder.append(Mn.process(node,
                settings));
             break;
          }
          case MFRAC: {
             builder.append(Mfrac.process(node,
                settings));
             break;
          }
          ...
       }
       node.setProcessed();
    }
}
```

Figure 5.5: Excerpt from the `Node` class

known to this application. Each operation is assigned a unique key that is also used to retrieve appropriate textual representation of this operation from the localization files. Next, there is a type of the operation as defined in Section 4.3.2. The last property is an array of possible operators or symbols that might be used to represent this operation in documents. These can be standard HTML character entities entered by their name (`&minus;`) or unicode code point (decimal or hexadecimal) or a name of the corresponding element in the Content MathML markup as seen in Figure 5.6.

The conversion of Presentation MathML elements is very simple and derives from the names of respective nodes. We offer special treatment for layout elements (such as `munder`, `msup`, ...) and distinguish a few notable operations that occur often in mathematical texts - limits, integrals, or summations.

On the other hand, the conversion of Content MathML elements is more

```
public enum Operation {
    MINUS_PLUS("minus_plus", OperationType.INFIX,
        "&#8723;", "&#x2213;", "∓");
}
```

Figure 5.6: An example of an operation

complicated. Everything important is happening inside the class `Apply`. The first child of the `apply` element denotes the operation. The operation can be defined by the appropriate Content MathML element, in which case we use the element name to retrieve the operation from the enumeration. It can also be defined by the `csymbol` element, then the value of the element is used, or by another `apply` element, for which we invoke the `Node.process()` method.

When we have determined the exact operation, we proceed by processing it based on its type, i.e., based on its word order when spoken. Operations with a type `OperationType.SPECIAL` have to be processed individually since they require a distinctive approach (e.g., logarithm), or there are multiple ways of expressing them (e.g., summation, integral).

Lastly the conversion of numbers to strings ($11 \rightarrow$ eleven) is optionally executed, and contents of identifiers (elements `mi` and `ci`) are verbatim copied to the output.

Example of results of the conversion can be found in Appendix C.

Chapter 6

Results

6.1 Mathematical Retrieval Collection MREC

MREC [11] is a large corpus of mathematical texts (numbering close to 450 thousand documents). MREC[1] is based on documents downloaded from the arXMLiv[2] [16], which in turn is created by transforming TEX documents from arXiv[3] - a huge library of freely available texts from multiple scientific fields including Physics, Mathematics, and Computer science. Using LATEXML(described in the Subsection 2.2.1), these texts are converted into the XML format with mathematics being represented in MathML. Not all documents from the arXiv are part of MREC - only documents for which the conversion yielded results in categories successful and complete with errors are included.

6.2 Comparing `XmlParser` Implementations

The application comes with two implementations of the `XmlParser` interface: one uses DOM, the other StAX. Since the StAX implementation uses only general API, we have an option to use and compare different StAX API implementations. Together we have four implementations to test:

- JDOM - a DOM implementation,
- Woodstox - a StAX implementation,
- Aalto - another StAX implementation,
- StAX - a StAX implementation included in Java 6.

The comparison will be performed on a standard desktop computer with the following configuration: CPU Intel Core i5-2400 (4 cores @3.1GHz),

1. `https://mir.fi.muni.cz/MREC/index.html`
2. `http://kwarc.info/projects/arXMLiv/`
3. `http://arxiv.org/`

8 GB RAM, standard HDD (7200 RPM) with Windows 8 64-bit and Java 7u21. Before we start comparing individual implementations, we will find the optimal number of threads to use. As we can see in Table 6.1, increasing the number of threads decreases computation time only until we use 10 threads, then the times stagnate or even slightly increase. Therefore, we will use 10 threads for comparison of `XmlParser` implementations.

Thread count	Time (s)
1	69.892
2	38.295
5	31.112
10	21.414
15	21.787
25	22.114

Table 6.1: Running the application on 200 documents from MREC with variable thread count

Now we can start testing different implementations. For this purpose we will use a sample of 2,000 MREC documents. We will focus not only on the computation time, but also on memory requirements represented by the heap size used by Java.

	Time (s)	Max. heap size (MB)	Max. used heap (MB)
StAX	282.895	531.7	430.6
Woodstox	161.172	796.5	754.4
Aalto	167.264	797.5	740.5
JDOM	185.198	796.8	698.7

Table 6.2: A comparison of `XmlParser` implementations

From Table 6.2 we can readily see that alternative StAX implementations are much faster than the implementation shipped with Java and also faster than JDOM. The memory usage of the three fastest implementations are roughly similar. However, from Table 6.3 we can read that the average memory usage of JDOM is much higher than the rest of implementations. From this data we can deduce that Woodstox and Aalto implementations provide the best performance results. From this point forward we will use Woodstox implementation, because it has a bigger user base, a better documentation, and an active development.

	Average heap size (MB)
StAX	350
Woodstox	500
Aalto	500
JDOM	700

Table 6.3: A comparison of `XmlParser` implementations' average heap sizes

6.3 Converting the MREC Corpus

The conversion was executed on a fairly weak server. The same sample of 2,000 that was used in comparison of different `XmlParser` implementations took 37 seconds to complete. After a preliminary testing, we found an optimal number of threads - 10. The conversion process was run with the language option set to English, in 10 threads, and using the Woodstox StAX implementation.

Files converted	439,423
Time taken	9,828,023 ms $\approx$ 164 min
Peak memory usage	850 MB
Unrecognized operations	4483 (unique)
Number of errors	780

Table 6.4: Results of the conversion of the MREC corpus

As we can see in Table 6.4 the conversion finished after 164 minutes and produced 780 errors. Most of these errors are caused by invalid or unexpected elements' position in the MathML structure. For example, the token element `mi`, which should contain only a textual value, had a child element. The `munder` element did not have two children.

The number of unique unrecognized operations is quite high. However, upon closer examination most of these operations are not defined by mathematical symbols (i.e., `swap`, `clique`, `prev`, ...).

Overall the conversion was successful. Every input document was converted (judging by the number of resulting files). The number of unrecognized operations is reasonable (one unique operation per 98 files), the number of errors even lower (one error per 563 files).

Chapter 7

Conclusion

At the beginning, the thesis examines MathML as a format for storing and exchanging mathematical content. Tools for converting various formats into MathML and the need of a single canonical representation of mathematical formulae in MathML are outlined and described. The thesis also introduces notable solutions that make mathematical content accessible for users with dyslexia or some form of visual impairment.

With the acquired information the analysis of the problem is performed. During the analysis it became clear that the system developed as a part of my Bachelor thesis is unsatisfactory in terms of MathML elements coverage and performance. A completely new system is designed. Streaming XML is proposed as the best method of processing XML documents. Every occurrence of the MathML markup in the input document is loaded into a custom simplified tree and thereafter converted into plain text format.

The resulting implementation provides a localization into three languages - English, Slovak, and Czech. Also, many settings for the application are available, including a possibility to swap StAX implementations.

In the last chapter the results of running the application on a substantial amount of documents from the MREC corpus are shown.

For a future work, a framework for defining mathematical operations via external files could be implemented. This will pose a few problems with the word order of user defined operations, especially in the Content MathML. Also, the application could be translated into more languages.

Bibliography

[1] Dominique Archambault, Donal Fitzpatrick, Gopal Gupta, Arthur I Karshmer, Klaus Miesenberger, and Enrico Pontelli. Towards a universal maths conversion library. In *Computers Helping People with Special Needs*, pages 664–669. Springer, 2004.

[2] Dominique Archambault and Victor Moço. Canonical MathML to simplify conversion of MathML to Braille mathematical notations. In *Computers Helping People with Special Needs*, pages 1191–1198. Springer, 2006.

[3] Josef B Baker, Alan P Sexton, and Volker Sorge. Towards reverse engineering of PDF documents. *Towards a Digital Mathematics Library. Bertinoro, Italy, July 20-21st, 2011*, pages 65–75, 2011. `http://dml.cz/dmlcz/702603`.

[4] Josef B Baker, Alan P Sexton, and Volker Sorge. MaxTract: Converting PDF to\ mbox\ LaTeX, MathML and Text. In *Intelligent Computer Mathematics*, pages 422–426. Springer, 2012.

[5] David Formánek, Martin Líška, Michal Růžička, and Petr Sojka. Normalization of Digital Mathematics Library Content. In James Davenport, Johan Jeuring, Christoph Lange, and Paul Libbrecht, editors, *24th OpenMath Workshop, 7th Workshop on Mathematical User Interfaces (MathUI), and Intelligent Computer Mathematics Work in Progress*, number 921 in CEUR Workshop Proceedings, pages 91–103, Aachen, 2012.

[6] José Grimm. Tralics, a LaTeXto XML Translator. *TUGboat*, 24(3):377–388, 2003.

[7] José Grimm. Producing MathML with Tralics. In Petr Sojka, editor, *Proceedings of DML 2010*, pages 105–117, Paris, France, July 2010. Masaryk University. `http://dml.cz/dmlcz/702579`.

[8] Eitan M Gurari. TEX4ht: HTML Production. *TUG-boat*, 25(1):39–47, 2004.

[9] Martin Jarmar. *Conversion of Mathematical Documents into Braille*. PhD thesis, Master's thesis, Faculty of Informatics (Jan 201 2), `https://is.muni.cz/th/172981/fi_m/?lang=en`, 2012.

[10] Maroš Kucbel. Generování textu z MathML, 2011. Bachelor Thesis, Masaryk University, Brno, Faculty of Informatics (advisor: Zuzana Nevěřilová), `http://is.muni.cz/th/324736/fi_b/?lang=en`.

[11] Martin Líška, Petr Sojka, Michal Růžička, and Petr Mravec. Web Interface and Collection for Mathematical Retrieval: WebMIaS and MREC. *Towards a Digital Mathematics Library. Bertinoro, Italy, July 20-21st, 2011*, pages 77–84, 2011.

[12] Bruce Miller. LaTeXML: A LaTeXto xml converter. *Web Manual at http://dlmf.nist.gov/LaTeXML/, seen April 2013*, 2013.

[13] TV Raman. *Audio System for Technical Readings*. PhD thesis, Cornell University, 1994.

[14] Waltraud Schweikhardt, Cristian Bernareggi, Nadine Jessel, Benoit Encelle, and Margarethe Gut. LAMBDA: A European system to access mathematics with Braille and audio synthesis. In *Computers Helping People with Special Needs*, pages 1223–1230. Springer, 2006.

[15] Petr Sojka and Martin Líška. Indexing and Searching Mathematics in Digital Libraries. In *Intelligent Computer Mathematics*, pages 228–243. Springer, 2011.

[16] Heinrich Stamerjohanns, Michael Kohlhase, Deyan Ginev, Catalin David, and Bruce Miller. Transforming Large Collections of Scientific Publications to XML. *Mathematics in Computer Science*, 3:299–307, 2010. `http://dx.doi.org/10.1007/s11786-010-0024-7`.

[17] Robert Stevens and Alistair Edwards. Mathtalk: The design of an interface for reading algebra using speech. In *Computers for Handicapped Persons*, pages 313–320. Springer, 1994.

[18] Robert David Stevens. *Principles for the design of auditory interfaces to present complex information to blind people*. PhD thesis, University of York, 1996.

[19] Masakazu Suzuki, Fumikazu Tamari, Ryoji Fukuda, Seiichi Uchida, and Toshihiro Kanahori. INFTY: an integrated OCR system for mathematical documents. In *Proceedings of the 2003 ACM symposium on Document engineering*, pages 95–104. ACM, 2003.

Appendix A

Electronic Attachments

`/examples/`	. . . MathML files before and after conversion
`MREC/`	. . .A few documents from the MREC corpus
`converter.jar`	. . .The developed application with all dependencies
`MathML2Text.zip`	. . .MathML2Text application
`apidocs/`	. . .Javadoc documentation
`src/`	. . .Source code
`target/`	. . .Built jar archives
`pom.xml`	. . .Maven configuration file
`README.txt`	. . .Instructions for running and building the application
`thesis.zip`	. . .PDF and source files of the thesis

Appendix B

Command Line Arguments

The application can be run from command line using standard Java invocation: `java -jar converter [options] <file...>`, where `<file...>` denotes one or more input documents. `[options]` is used to set up the application using available options described below.

Options to customize the conversion process:

`-h,-help`

- print available options

`-c,-canonicalize`

- every input document will be canonicalized before conversion

`-cm,-content-markup`

- the Content MathML will be used for conversion if available

`-l,-language <LANGUAGE>`

- set desired language of conversion; English (`en`), Slovak (`sk`), and Czech (`cs`) are available

`-n,-transform-numbers`

- all numbers will be transformed into strings

`-o,-output <PATH>`

- set output directory

`-p,-parser <PARSER>`

- choose parser implementation that will be used for loading input documents; default Java StAX (`stax`), Woodstox (`woodstox`), Aalto (`aalto`), and JDOM (`dom`) are available

`-r,-replace-spaces`

- all spaces in localization files will be replaced with underscores, i.e., open_braces

`-t,-threads <NUMBER>`

- set desired number of threads the application will use for parallel processing

A possible application invocation can be: `java -jar converter -c -l en -p woodstox -r -t 10 inputFile.xml`

Appendix C

Conversion Examples

```
<?xml version="1.0" encoding="UTF-8"?><document
   xmlns:m="http://www.w3.org/1998/Math/MathML"
   xmlns:conv="http://code.google.com/p/mathml-
   converter/">
    <child id="42">
        <content>content</content>
        <conv:math>logarithm to the base two of open
            braces x plus five point two close
           braces</conv:math>
        <content></content>
    </child>
    <conv:math>logarithm to the base two of open
       braces x plus five point two close braces</
       conv:math>
</document>
```

Figure C.1: Figure C.2 converted using StAX

```
<?xml version="1.0" encoding="UTF-8"?>
<document xmlns:m="http://www.w3.org/1998/Math/
   MathML">
    <child id="42">
        <content>content</content>
        <m:math>
            <m:mrow>
                <m:msub>
                    <m:mo>log</m:mo>
                    <m:mn>2</m:mn>
                </m:msub>
                <m:mrow>
                    <m:mi>x</m:mi>
                    <m:mo>+</m:mo>
                    <m:mn>5.2</m:mn>
                </m:mrow>
            </m:mrow>
        </m:math>
        <content/>
    </child>
    <m:math>
        <m:semantics>
            <m:annotation-xml encoding="MathML-
               Content">
                <m:apply>
                    <m:csymbol>log</m:csymbol>
                    <m:logbase><m:cn>2</m:cn></m:
                       logbase>
                    <m:apply>
                        <m:plus/>
                        <m:ci>x</m:ci>
                        <m:cn>5.2</m:cn>
                    </m:apply>
                </m:apply>
            </m:annotation-xml>
        </m:semantics>
    </m:math>
</document>
```

Figure C.2: Expression $log_2(x + 5.2)$ in MathML

```
<?xml version="1.0" encoding="UTF-8"?><math xmlns="
   http://code.google.com/p/mathml-converter/">open
    braces sum from open braces i equals zero close
    braces to ten of open braces absolute value of
   i factorial minus x with lower index i end
   absolute value close braces close braces times
   definite integral from 0 to 1 of x with respect
   to x</math>
```

Figure C.3: Figure C.4 converted using StAX

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
    <mrow>
        <mrow>
            <munderover>
                <mo>&#x2211;</mo>
                <mrow>
                    <mi>i</mi>
                    <mo>=</mo>
                    <mn>0</mn>
                </mrow>
                <mn>10</mn>
            </munderover>
            <mrow>
                <mo>|</mo>
                <mi>i</mi>
                <mo>!</mo>
                <mo>-</mo>
                <msub>
                    <mi>x</mi>
                    <mi>i</mi>
                </msub>
                <mo>|</mo>
            </mrow>
        </mrow>
        <mo>*</mo>
        <msubsup>
            <mi>&#x222b;</mi>
            <mi>0</mi>
            <mi>1</mi>
        </msubsup>
        <mi>x</mi>
        <mo>&#x2146;</mo>
        <mi>x</mi>
    </mrow>
</math>
```

Figure C.4: Expression $\Sigma_{i=10}^{10}|i! - x_i| * \int_0^1 x\mathrm{d}x$ in MathML

Index