TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

WYSIWYG editor pro sazbu MathML

WYSIWYG editor for MathML

Diplomová práce

Autor: **Bc. Pavel Svatoň**
Vedoucí práce: Ing. Pavel Tyl
Konzultant: Ing. Jiří Týř

Rozsah práce:

počet stran:	74
počet obrázků:	18
počet tabulek:	7
počet příloh:	7

V Liberci 11. prosince 2008

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.)

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, které má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše)

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum 11. prosince 2008

Podpis

Poděkování:

Děkuji vedoucímu mé diplomové práce Ing. Pavlu Tylovi a stejně tak konzultantovi Ing. Jiřímu Týřovi za ochotu, rady a podněty při realizaci této práce.

Abstrakt

V současnosti je většina matematiky na webu zachycena pomocí obrázků, které jsou ale těžko modifikovatelné a zpracovatelné aplikacemi, jenž manipulují s HTML. Proto World Wide Web Consortium, později W3C Math working group, vytvořilo MathML pro usnadnění práce s matematikou v prostředí WWW. MathML je tedy XML aplikace pro publikaci matematických vzorců na Webu. Tato diplomová práce se zabývá úkolem návrhu a implementace WYSIWYG editoru pro tento matematický značkovací jazyk. Editor bude jak samostatnou aplikací, tak i apletem, který bude možné vystavit na web a komunikovat s ním. K vytvoření aplikace byl použit programovací jazyk Java a to především jeho knihovna Swing pro tvorbu grafického uživatelského rozhraní a Java API for XML Processing, tedy rozhraní pro zpracování dokumentů v jazyce XML.
Klíčová slova: HTML, W3C, MathML, XML, WYSIWYG, Swing, API

Abstract

In present are almost mathematical terms on web captured through the images which are for the applications working with HTML hard to modified and hard to elaborate. That is why World Wide Web Consortium, later W3C Math working group, developed MathML to help with work with mathematical terms on WWW. MathML is thus XML application for publication the mathematical terms on web. This work deals with proposal and implementation WYSIWYG editor for mentioned mathematical marking language. Editor will be both independent application and applet, which will be able to run on web and communicate with it. For creating this editor was used the programming language Java, mainly its library Swing for creating graphical user interface and Java API for XML Processing for elaboration documents in XML language.
Key words: HTML, W3C, MathML, XML, WYSIWYG, Swing, API

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

API	Application Programming Interface
AWT	Abstract Windowing Toolkit
CSS	Cascade StyleSheet
DOM	Document Object Model for XML
DTD	Document Type Definition
GIF	Graphics Interchange Format
GUI	Graphical User Interface
HTML	HyperText Markup Language
JDK	Java Development Kit
JVM	Java Virtual Machine
MathML	Mathematical Markup Language
MIME	Multipurpose Internet Mail Extensions
MML	Mathematical Markup Language
MSIE	Microsoft Internet Explorer
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
UCS	Universal Character Set
UML	Unified Modeling Language
URL	Uniform Resource Locator
UTF-8	UCS Transformation Format
W3C	The World Wide Web Consortium
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSLT	eXtensible Stylesheet Language Transformations
WWW	World Wide Web
WYSIWYG	What You See Is What You Get

1 Úvod

Publikování matematických a jiných vzorců na internetu prostřednictvím obrázků přináší řadu nevýhod, proto byl vytvořen standard MathML. Tvoření samotného kódu MathML je ale velmi složité, pracné a zdlouhavé (viz příloha A) a proto se pro zápis matematických výrazů v tomto značkovacím jazyku využívají editory, které poskytují uživatelsky přívětivější prostředí a odstraňují nutnou znalost jednotlivých tagů a jejich atributů. Cílem této práce je tedy vytvořit editor, pomocí něhož si uživatel sestaví libovolný vzorec a následně si bude moci vygenerovat výsledný MathML kód, který si posléze může vložit do svých HTML stránek. WYSIWYG editor bude naprogramován i jako aplet jazyka Java. MathML je možné zapisovat ve dvou formách a to v prezentační formě určené především pro vysázení vzorce v prohlížeči, nebo ve formě sémantické pro vyhodnocování a interpretování jinými aplikacemi. Tyto formy lze také kombinovat. Vytvořený editor bude implementovat většinu elementů, ale pouze z kategorie prezentačního MathML. Cílem práce je rovněž podpora jednoduchého stylování vzorců.

V současnosti existuje několik WYSIWYG editorů pro MathML. Většina je ale buď ve formě pluginů do prohlížeče nebo se řadí mezi komerční programy. Uvedl bych zde dva zajímavý WYSIWYG editory, které mi byly inspirací a předlohou při vlastní realizaci. Jedním z nich je komerční produkt MathType [7] pro operační systémy Windows a Macintosh. Jde o trial verzi programu, kdy po uplynutí zkušební doby jsou znepřístupněny některé jeho funkce. Výstupem je soubor ve formátu MML, GIF či PostScript. Uživatelské rozhraní je velice intuitivní, nabízí uživateli veškeré matematické funkce a symboly. Zásadním nedostatkem je, že nepodporuje opětovné vložení MathML kódu a jeho přeložení do původního vzorce. Dalším zajímavým editorem je MathML .NET Control [13]. Jde o profesionální nástroj, jenž může být spuštěný buď v plnohodnotném režimu se všemi ovládacími tlačítky či v režimu bez nabídkové lišty. Výstupem je pouze formát XML či MML, uložení do GIF či jiných formátů tento editor nenabízí. Jeho hlavní předností je nabídka široké škály funkcí, symbolů a operátorů. Implementuje většinu elementů a jejich atributů spadajících do kategorie prezentačních prvků MathML. Bohužel tento nástroj je komerční a jenom

pro zajímavost jedna licence pro normálního uživatele stojí $ 299. Přehled dalších editorů pro MathML můžete nalézt na [22].

Práce je rozčleněna do několika kapitol:

Kapitola 2 přibližuje důvody vzniku standardu pro zápis matematických výrazů a nedostatky jiných řešení. Ještě před úvodem do MathML jsou rozebrány základy jazyka XML, ze kterého byly matematické značky odvozeny. Tato kapitola se věnuje hlavně popisu jednotlivých prvků prezentačního MathML. Obsahuje také užitečné informace o zakomponování MathML do jiných webových dokumentů a o podpoře u nejznámějších a nejpoužívanějších internetových prohlížečů na různých platformách.

Kapitola 3 je nejobsáhlejší částí dokumentu, kde je podrobně zachycen postup při tvorbě WYSIWYG editoru. Věnuje se všem fázím vývoje uvedeným na začátku kapitoly 3.

2 MathML

Už v roce 1995 bylo zřejmé, že vkládání matematických výrazů do tehdejšího HTML je dosti omezené, a proto na nátlak především vědecké komunity konsorcium W3C uznalo, že podpora matematiky na webu je nedostatečná. Zobrazování veškerých matematických vzorců bylo řešeno pomocí obrázků. Toto řešení, ale přinášelo spoustu nedostatků:

- datová velikost není ideální,
- složitá tvorba v grafických editorech a modifikovatelnost, takto vysázený vzorec se navíc není schopen přizpůsobit změnám týkající se prohlížeče, jako je třeba změna velikosti písma atd.,
- aplikace pracující s webem není schopna obrázek interpretovat a nevyčte z něj význam,
- není možná dynamická tvorba matematických vzorců.

Z uvedených důvodů se konsorcium snažilo tento problém řešit. Za tímto účelem vznikla skupina W3C Math working group. Původním návrhem bylo začlenění matematiky do HTML 3.0. Z důvodu neustálého rozšiřování internetu a stoupání obliby u různých komunit vznikaly požadavky na neustálé rozšiřování HTML o nové strukturované datové typy netýkající se pouze matematiky. Později bylo zřejmé, že musí vzniknout obecný mechanismus pro vložení nových typů. Tímto mechanismem se stal jazyk XML. Zápis matematických vzorců se stal jedním z jeho podmnožin. Při návrhu MathML se bral ohled především na způsob zakomponování tohoto typu dat do existujícího prostředí. Nabízely se tři varianty:

- konvertováním existujících formátů, jako je například nejrozšířenější TEX, do MathML,
- vložením MathML do HTML, tak aby ho byly schopni webové prohlížeče interpretovat,
- zpřístupněním MathML zápisu i jiným aplikacím, které ho budou umět interpretovat a vyhodnocovat.

Matematický značkovací jazyk je tedy aplikací jazyka XML a proto v následující kapitole uvedu základy tohoto standardu.

2.1 Jazyk XML

XML (eXtensible Markup Language) je značkovací jazyk, který se vyvinul z jazyka SGML (Standard Generalized Markup Language). XML je považovaný za standardní formát pro výměnu a sdílení informací. Tento jazyk (narozdíl od HTML) umožňuje tvorbu vlastní sady značek (tagů), která musí dodržovat určitá syntaktická pravidla. Základním stavebním kamenem každého dokumentu je element, který je v textu vyznačován tzv. tagem. XML je case-sensitive a tak se ve jménech elementů rozlišují malá a velká písmena. Tagy mohou být jak párové tak i samostatné, čímž vyjadřujeme, že element je prázdný. Mezi zahajující a ukončující značkou se může vyskytovat text či další prvky. U každého tagu se může vyskytovat tzv. atribut, který obsahuje doplňkové informace o daném prvku a je vždy umístěn v počáteční značce. Hodnota atributu musí být uzavřena do uvozovek. Dalším prvkem XML jsou entity, kterými zapisujeme speciální znaky. Součástí dokumentu mohou být i komentáře, které jsou při zpracování ignorovány. Jazyk XML má jednoduchá, ale velmi přísná pravidla, která se musí při tvorbě každého dokumentu striktně dodržovat.

Syntaktická pravidla:

- Celý dokument musí být vždy zabalen do jednoho prvku označovaného jako kořenový. Tento element se nesmí vyskytovat v obsahu jiných prvků.
- Názvy elementů odpovídají pravidlům. Při výběru jména elementů existují určitá omezení, která musíme dodržovat. Začátek názvu musí být složen buď z písmena či podtržítka, dalšími znaky mohou být písmena, číslice, podtržítka, pomlčky a tečky. Ostatní znaky nejsou při výběru jména prvku povoleny.
- Každý XML dokument musí obsahovat alespoň jeden prvek.
- V tomto značkovacím jazyce se rozlišují malá a velká písmena.
- Při vnořování elementů do sebe, se musíme vyhnout problému křížení prvků. Znamená to, že pokud prvek obsahuje počáteční tag jiného elementu, musí pak obsahovat i tag ukončující.

- Atributy odpovídají také určitým pravidlům. Jsou složeny z názvu a hodnoty, která je oddělena od názvu rovnítkem. Pro stanovení názvu atributů platí ta samá omezení jako u názvu elementů. Hodnota atributu je vždy ohraničena uvozovkami.

- V dokumentu se nesmí vyskytovat zakázané znaky, které mají svůj specifický význam. Místo těchto znaků se používají tzv. vestavěné znakové entity.

Na prvním řádku XML dokumentu se může také nacházet tzv. XML prolog, který udává podle jaké verze XML byl dokument vytvořen a jeho součástí může také být název použitého kódování, implicitně je nastavené UTF-8. Níže je uveden ilustrativní příklad XML dokumentu. Na prvním řádku se nachází XML prolog, pod nímž je povinný kořenový element `prehled_firem`. Abychom se vyhnuli použití zakázaného znaku ampersandu, vložili jsme do prvku `nazev` vestavěnou znakovou entitu. Na témže řádku je uveden i komentář. Za povšimnutí také stojí element `info`, který není párový a obsahuje dva atributy a to `pocet_zamestnancu` a `zisk`.

```
<?xml version="1.0" encoding="UTF-8"?>
<prehled_firem>
    <spolecnost>
        <nazev>Novak &amp; syn</nazev>  <!-- prodej hardwaru -->
        <adresa>
            <ulice cp="140">Komenského</ulice>
            <mesto>Teplice</mesto>
        </adresa>
        <info pocet_zamestnancu="10" zisk="1000000" />
    <spolecnost>
</prehled_firem>
```

2.2 Prezentační MathML

Vytvořený WYSIWYG editor bude implementovat pouze prezentační elementy a proto se o významových prvcích zmíním jenom ve stručnosti. Prezentační prvky popisují strukturu matematického zápisu, kdežto sémantické prvky popisují přímo matematické objekty, nikoliv notaci. Prezentační forma se používá především pro vizualizaci vzorce a sémantická forma je určena pro aplikace, jenž budou matematiku ze vzorce zpracovávat a vyhodnocovat. Následující MathML kód obsahuje oba typy zápisu

téhož vzorce pro diskriminant kvadratické rovnice. V levém sloupci je použita prezentační forma a ve sloupci pravém je použita forma významová.

```
<msqrt>
  <mrow>
    <msup>
      <mi>b</mi>
      <mn>2</mn>
    </msup>
    <mo>-</mo>
    <mrow>
      <mn>4</mn>
      <mo>&InvisibleTimes;</mo>
      <mi>a</mi>
      <mo>&InvisibleTimes;</mo>
      <mi>c</mi>
    </mrow>
  </mrow>
</msqrt>
```

```
<apply>
  <root/>
  <apply>
    <minus/>
    <apply>
      <power/>
      <ci>b</ci>
      <cn>2</cn>
    </apply>
    <apply>
      <times/>
      <cn>4</cn>
      <ci>a</ci>
      <ci>c</ci>
    </apply>
  </apply>
  <cn>2</cn>
</apply>
```

Většina sémantických prvků představuje příslušné operátory, relace a funkce. Tato forma zápisu odpovídá prefixové notaci, kde nejprve je uvedena nějaká matematická operace a až posléze nalezneme matematické objekty, jimiž mohou být identifikátory, čísla, atd. Těchto prvků je tedy podstatně více než prezenčních elementů.

Obě uvedené formy lze také spojit do jednoho zápisu MathML, protože v některých situacích je lepší nechat až na konkrétní aplikaci, aby si vybrala, jaká forma zápisu ji nejvíce vyhovuje a odpovídá jejímu typu úlohy. Z těchto důvodů je možné spojit oba zápisy, buď způsobem připojení významové poznámky k prezentačnímu výrazu, nebo připojení prezentační poznámky k sémantickému výrazu. V později uvedeném MathML zápisu je už vidět, jak se toto spojení forem realizuje. Důležitým elementem, který zajišťuje připojení obou notací je element `semantics`. Elementem `annotation-xml` vyznačujeme připojenou poznámku a pomocí jeho atributu `encoding` určujeme typ zápisu v poznámce.

```
<semantics>
  <mrow>
    <mfenced>
      <mrow>
```

```
<semantics>
  <apply>
    <factorial/>
    <apply>
```

```
        <mi>a</mi>
        <mo>+</mo>
        <mi>b</mi>
      </mrow>
    </mfenced>
    <mo>!</mo>
  </mrow>
  <annotation-xml
   encoding="MathML-Content">
    <apply>
      <factorial/>
      <apply>
        <plus/>
        <ci>a</ci>
        <ci>b</ci>
      </apply>
    </apply>
  </annotation-xml>
</semantics>
```

```
      <plus/>
      <ci>a</ci>
      <ci>b</ci>
    </apply>
  </apply>
  <annotation-xml
   encoding="MathML-Presentation">
    <mrow>
      <mfenced>
        <mrow>
          <mi>a</mi>
          <mo>+</mo>
          <mi>b</mi>
        </mrow>
      </mfenced>
      <mo>!</mo>
    </mrow>
  </annotation-xml>
</semantics>
```

Prezentační MathML lze rozdělit na několik kategorií. První skupinou jsou symboly a zastupují je elementy uvedené v tab. 2.1. Tyto prvky neumožňují vnořování jiných elementů, obsahují už konkrétní hodnoty. V posledním sloupci je uvedeno, zda vytvořený WYSIWYG editor bude implementovat daný prvek.

Tabulka 2.1: Přehled elementů kategorie symboly

Element	**Popis**	**Podpora**
mi	identifikátor	ANO
mn	číslo	ANO
mo	operátor, závorky či oddělovač	ANO
mtext	text	ANO
mspace	mezera	ANO
ms	řetězec	ANO
mglyph	glyf	NE

Další skupinou jsou obecná schémata. Tyto prvky už umožňují vnořování elementů a proto je tabulka 2.2 rozšířena o údaj počet argumentů, které daný element

může obsahovat. Například element `mfrac` je složen právě ze dvou podvýrazů představující jmenovatel a čitatel, tyto podvýrazy jsou umístěny do elementů `mrow`. Prvek `mphantom` obsahuje podvýraz, který nebude vysázen, ale bude zachovávat jeho velikost. Nikde není stanovené, zda podvýraz, jenž je obsažen například v prvcích `mfenced`, `mphantom`, `msqrt`, musí být uzavřen v elementu `mrow`, či zda může obsahovat konkrétní hodnoty zastoupené elementy `mi`, `mn` a dalšími. Ve všech příkladech uvedených na [18] je použit `mrow` a tak budu toto pravidlo dodržovat i při realizaci editoru. Stylování vzorců v editoru docílím právě pomocí elementu `mstyle` a jeho atributů. Obsahuje atributy pro nastavení pozadí (`mathbackground`), barvy písma (`mathcolor`), velikost písma (`mathsize`), nastavení řezu písma (`mathvariant`) a spoustu dalších. Typ závorek, kterými ohraničíme podvýraz, se přiřazuje pomocí atributů prvku `mfenced`. Je jím atribut `open` a `close`. Například, chceme-li mít počáteční závorku hranatou a ukončující složenou, bude zahajující tag vypadat následovně:

```
<mfenced open="[" close="}">
```

Chceme-li obsah uzavřít například do horizontálního kruhu, odmocniny nebo třeba podtrhnout číslo při součtu, použijeme element `menclose`. Symbol pro ohraničení se nastavuje atributem `notation`.

Tabulka 2.2: Výčet elementů ze sady obecných schémat

Element	**Popis**	**Počet arg.**	**Podpora**
mrow	podvýrazy řazené horizontálně	0 nebo více	ANO
mfrac	zlomek	2	ANO
msqrt	druhá odmocnina	1 a více	ANO
mroot	n-tá odmocnina	2	ANO
mstyle	nastavení stylu	1 a více	ANO
mpadded	přidání mezery okolo obsahu	1 a více	NE
mphantom	neviditelný obsah	1 a více	ANO
mfenced	uzavře obsah do závorek	0 nebo více	ANO
menclose	uzavře obsah přizpůsobeným symbolem	1 a více	NE

Další kategorií prezentačních prvků jsou schémata textů a krajních hodnot. Jejich soupiska je v tabulce 2.3. Prvním argumentem všech uvedených elementů je vždy element představující základ, může jim být `mrow` či nějaký konkrétní prvek (`mn`, `mi`, `ms`). Následující argumenty potom reprezentují indexy, exponenty a obsah nad nebo pod základem. Element `mmultiscripts` navíc může obsahovat prázdný prvek `mprescripts`, který značí, že všechny následující prvky budou umístěny před základem. Použijeme ho v případě, kdy chceme, aby před základem byl zobrazen index či exponent. Po základu uvádíme elementy v pořadí index, exponent ‚`mprescripts`, index, exponent. Pokud nějaký prvek nepotřebujeme uvést pak ho nahradíme prázdným elementem `none`. Vzájemného zarovnání hodnot v indexu a mocnině můžeme dosáhnout zápisem: element indexu, element exponentu, element indexu, element exponentu... Lze tak vytvořit i následující ukázku ${}_{i\ m}^{jln}X$ zápisem:

```
<mmultiscripts>
  <mi>X</mi>                <!-- zaklad -->
  <mi>i</mi><mi>j</mi>      <!-- index, exponent -->
  <none/><mi>l</mi>         <!-- prazdny element, exponent -->
  <mi>m</mi><mi>n</mi>
</mmultiscripts>
```

Tabulka 2.3: Prvky spadající do kategorie schémat textů a krajních hodnot

Element	**Popis**	**Počet arg.**	**Podpora**
msub	připojení dolního indexu k základu	2	ANO
msup	připojení horního indexu k základu	2	ANO
msubsup	připojení horního i dolního indexu	3	ANO
munder	připojí prvek pod základ	2	ANO
mover	připojí prvek nad základ	2	ANO
munderover	připojí prvek nad i pod základ	3	ANO
mmultiscripts	dolní i horní index za i před základem	1 nebo více	ANO

Pro zápis matic a tabulek jsou určeny elementy uvedené v tabulce 2.4. Základním prvkem je element `mtable`, jenž smí obsahovat pouze prvky `mtr` a `mlabeledtr`. Do těchto prvků je povolené vnořovat pouze elementy `mtd` představující konkrétní

položku matice či buňku tabulky. Prázdné značky `malignmark` a `malignmark` využijeme především při zápisu soustav zarovnaných rovnic podle hodnot, proměnných či operátorů. WYSIWYG editor bude podporovat nastavení vlastností matic a tabulek, jakými jsou například stejná výška řádku (`equalrows`) a šířka sloupců (`equalcolumns`), způsoby zarovnání řádků (`rowalign`) i sloupců (`columnalign`), oddělení plnou či přerušovanou čárou řádků (`rowlines`) nebo sloupců (`columnlines`) a ohraničení tabulky (`frame`). Doplnění matice či tabulky o tyto vlastnosti dosáhneme pomocí atributů elementu `mtable`.

Tabulka 2.4: Elementy pro konstrukci tabulek a matic

Element	**Popis**	**Počet arg.**	**Podpora**
mtable	tabulka nebo matice	0 nebo více řádků	ANO
mlabeledtr	řádek s popisem či očíslováním	1 nebo více	NE
mtr	řádek v tabulce či matici	0 nebo více	ANO
mtd	jedna položka či buňka tabulky	1 a více	ANO
malignmark	značka pro zarovnání	—	NE
malignmark	značka pro zarovnání	—	NE

Pro úplnost uvádím poslední prvek (tab. 2.5), kterým přidáváme dynamičnost k podvýrazu. Atributem `actiontype` nastavujeme typ akce, o kterou daný prvek rozšiřujeme. Jenom pro představu jednou z akcí může být například změna vzorce při stisku tlačítka myši na vzorci.

Tabulka 2.5: Oživující element

Element	**Popis**	**Počet arg.**	**Podpora**
maction	připojí akci k podvýrazu	1 nebo více	NE

Podrobný výčet a popis všech elementů včetně jejich atributů, příklady použití a další užitečné informace ohledně standardu MathML naleznete na [18].

2.3 MathML interface

Pro zapojení MathML do jiných dokumentů (HTML, XHTML, XML...), se používají prvky rozhraní. Pro integrování se matematický zápis vloží na požadované místo v dokumentu a uzavře se do elementu `math`. Tento element disponuje množstvím atributů, některé z nich jsou v tabulce 2.6. Atribut `id` a `class` se používají pro odkazování na daný prvek či skupinu prvků ze skriptů či CSS, nebo se používají pro účely transformace (např. do jiného formátu). Posledním uvedeným atributem specifikujeme URL adresu MathML schématu, podle kterého se kontroluje validita dokumentu. Pomocí jmenného prostoru jednoznačně identifikujeme, ke které sadě značek patří která značka.

Tabulka 2.6: Vybrané atributy prvku rozhraní

Atribut	**Popis**
class	přiřazení do jedné nebo více tříd
id	přiřazení jedinečného jména v celém dokumentu
style	nastavení vizuálních, popř. hlasových vlastností (CSS)
display	specifikuje, jak má být MathML zobrazeno (`display`, `inline`)
macros	specifikuje URL adresy ukazující na externí macro soubory
schemaLocation	deklarace MathML jmenného prostoru

Webové prohlížeče pro interpetování MathML vyžadují některé náležitosti dokumentu:

- DOCTYPE deklaraci, kterou odkazujeme na DTD. Jde o soubor podmínek pro tvorbu XML dokumentu. Jak již bylo uvedeno v kapitole 2.1, každý XML dokument musí odpovídat syntaktickým pravidlům, abychom o něm mohli říci, že je správně strukturovaný, nebo-li well-formed dokument. To je ovšem pouze první úroveň ověření správnosti XML dokumentů. Pokud vytvořený XML dokument odpovídá i všem pravidlům zapsaným v DTD, říkáme, že je validní a splňuje tak i druhou úroveň správnosti. Například pokud chceme, aby MathML tvořil samostatný dokument, vložíme do standardního XML dokumentu následující deklaraci

```
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
  "http://www.w3.org/Math/DTD/mathml2/mathml2.dtd">
```

- deklaraci jmenného prostoru. Pokud je v dokumentu použito více aplikací XML, pak je zapotřebí jednotlivé značky rozlišovat. Takový mechanizmus, kterým tento problém zabezpečíme, je označován jako jmenný prostor (namespace). Budeme-li mít XHTML dokument, do kterého chceme vložit MathML, pak uvozující značka dokumentu bude mít podobu

  ```
  <html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:mml="http://www.w3.org/1998/Math/MathML">
  ```

  Veškerý elementy spadající do MathML se pak budou uvádět s prefixem `mml`.

  ```
  <mml:math>
    <mml:mrow>
      <mml:mo>-</mml:mo>
      <mml:mn>1</mml:mn>
    </mml:mrow>
  </mml:math>
  ```

- objektové tagy a procesní instrukce. Jeden z nejpoužívanějších prohlížečů Microsoft Internet Explorer (MSIE) potřebuje pro korektní zobrazení MathML doplnění o plug-iny. Tyto podpůrné programy ale vyžadují buď uzavření MathML do elementu `embed`, nebo hlavička dokumentu musí obsahovat element `object`. Více informací o podpoře MathML ze strany internetových prohlížečů je v kapitole 2.4.

- určitý typ dokumentu. Prohlížeče Netscape, Mozilla vyžadují XML dokument a MSIE vyžaduje HTML dokument. Součástí každého dokumentu je MIME typ, který předchází datům a určuje co vlastně předávaná data mezi serverem a klientem (prohlížečem) představují. Příklad některých MIME typů naleznete v tabulce 2.7

Při publikování matematiky na webu je zároveň našim cílem zpřístupnit tyto stránky co nejširšímu spektru webových prohlížečů. Univerzálním řešením je vytvoření XML dokumentu, ve kterém budeme měnit MIME typ podle potřeb klienta (prohlížeče). Pro přepnutí MIME typu na uživatelově straně lze použít standardní

Tabulka 2.7: Přehled některých MIME typů

MIME typ	**Dokument**
text/html	HTML, XHTML 1.0 při dodržení pravidel kompatibility, ostatní XHTML pro MSIE
text/xml	veškeré XHTML
text/css	stylový jazyk CSS
application/xml	veškeré XHTML
text/xsl	stylový jazyk XSL

XSL stylesheet přímo od W3C Math Group. Nejpoužívanější a nejdůležitější částí XSL standardu je transformační jazyk XSLT. Pomocí tohoto jazyka se budou nejenom měnit MIME typy, ale budou se i upravovat DOCTYPE deklarace, přidávat se prvky pro podpůrné plug-iny atd. Při vystavení dokumentu na internetu je zapotřebí spolu s ním umístit na server také soubor se stylovým jazykem XSL. Musíme tak učinit, jelikož MSIE z bezpečnostních důvodů nebude provádět potřebné transfromace. Pokud budeme pracovat offline, pak bychom samozřejmě měli mít tento soubor na lokálním počítači. V případě, že MathML dokument obsahuje pouze prezentační prvky, je možné použít soubory `pmathml.xsl` a `pmathmlcss.xsl`. Toto řešení je rychlejší a proto i doporučované. Způsob zadání preference při určování nástroje pro zpracování MathML a jiné informace včetně výše uvedených souborů je možné nalézt na [21].

V příloze můžete shlédnout ukázkový XHTML dokument (viz příloha B) obsahující MathML a také samostatný XML (viz příloha C) obsahující pouze matematický značkovací jazyk.

2.4 Podpora MathML v internetových prohlížečích

I přes veškerou snahu W3C, hned na začátku této kapitoly musím bohužel konstatovat, že podpora MathML v prohlížečích je nedostatečná. Je to jediný faktor, který brzdí rozšíření používání tohoto značkovacího jazyka. Začíná ale svítat na lepší zítřky a nové verze prohlížečů jako například Opera začínají tento standard zahrnovat. Jak

to vypadá s podporou MathML u konkrétních prohlížečů na různých platformách je uvedeno v následujícím souhrnu:

- Windows
  - Amaya, všechny verze
  - MSIE 5.0 pomocí Techexplorer plug-inu
  - MSIE 5.5 použitím jednoho z pluginů MathPlayer, Techexplorer plug-inu
  - MSIE 6.0+ pomocí MathPlayer nebo Techexplorer plug-inu
  - Mozilla FireFox 0.9.9+
  - Netscape 6.1 pomocí Techexplorer plug-inu
  - Netscape 7.0+
  - Opera 9.5 s Weekly build - 9656
  - Opera 9.6+
- Linux/Unix
  - Amaya, všechny verze
  - Mozilla FireFox 0.9.9+
  - Netscape 6.1 pomocí Techexplorer plug-inu
  - Netscape 7.0+
- Macintosh
  - Mozilla FireFox 0.9.9+
  - MSIE 5.0+ pomocí Techexplorer plug-inu

Nejlépe je na tom freewarový webový prohlížeč a editor Amaya vytvořen konsorciem W3C. Prohlížeč a editor byl vytvořený za účelem získání systému, který umožní spojit více technologií konsorcia. K editaci matematických vzorců využívá WYSIWYG editor.

Až na menší nedostatky neměl prohlížeč Mozilla s interpretací MathML žádné problémy. Proto jsem pro veškeré testování realizovaného WYSIWYG editoru používal právě tento moderní a velice rozšířený webový prohlížeč. V příloze A se můžete podívat na vysázení kvadratické rovnice tímto interpretem.

Opera až do verze 9.5 matematický značkovací jazyk úplně opomíjela. Teprve modul pro rozšíření přinesl základní podporu pro jazyk MathML, kromě toho vylepšoval i současnou podporu XSLT a odstraňoval spoustu chyb prohlížeče.

Nejnovější verze nejrozšířenějšího prohlížeče MSIE sám o sobě matematický zápis vůbec nepodporuje. Microsoft popisuje na svých stránkách řešení, které spočívá

v doplnění prohlížeče o podpůrné plug-iny. Doporučuje instalování programu MathPlayer [8] od společnosti Design Science. Tento produkt je zcela zdarma. Pokud máme tento plug-in nainstalovaný, pak lze využít buď XSL stylesheet od W3C Math Group, jenž se postará o doplnění dokumentu o potřebné elementy, bez nichž by ke správnému zobrazení matematiky nedošlo, nebo se o vložení potřebných prvků postaráme sami. V druhém případě musíme nejprve rozšířit element `html` o jmenný prostor

```
<html xmlns:mml="http://www.w3.org/1998/Math/MathML">
```

a následně vložit prvek `object` do hlavičky dokumentu.

```
<object id="MathPlayer"
  classid="clsid:32F66A20-7614-11D4-BD11-00104BD3F987"
  codebase="http://www.dessci.com/dl/mathplayer.cab"
</object> <?import namespace="mml" implementatio="#MathPlayer">
```

Pak už lze do dokumentu zapisovat jednotlivé matematické značky uvedené s prefixem `mml`.

Kromě MathPlayeru lze využít i jednoho z nejznámějších a nejkvalitnějších nástrojů umožňující zobrazení jak MathML tak i TEXu a LATEXu, a to Techexplorer Hypermedia Browser [10]. Tento vysoce výkonný podpůrný program, jenž byl původně vyvinut firmou IBM a nyní spadá pod společnost Integre, lze využít jak pro MSIE tak i pro webový interpret Netscape verze 6.1. Využíváme-li tento plug-in, pak veškerý MathML prvky budou uzavřeny v elementu `embed`. Zápis vzorce $c = \sqrt{a^2 + b^2}$ by tak vypadal následovně:

```
<embed type="text/mathml" mmldata="
<math>
  <semantics>
    <mrow>
      <mi>c</mi><mo>=</mo>
      <msqrt><mrow>
        <msup><mi>a</mi><mn>2</mn></msup>
        <mo>+</mo>
        <msup><mi>b</mi><mn>2</mn></msup>
      </mrow></msqrt>
    </mrow>
   <annotation encoding='MathType-MTEF'>
  </annotation>
  </semantics>
</math>">
```

Pro testování jednotlivých webových prohlížečů byly použity vytvořené příklady MathML přímo od W3C. Příklady prezentačních i sémantických elementů jsou uvedeny na adrese [19].

3 Realizace WYSIWYG editoru

Ke tvorbě WYSIWYG editoru byl použit programovací jazyk Java (JDK 6 Update 7) [16] a open source vývojové prostředí NetBeans 5.5.1. [12]. Vývoj WYSIWYG editoru probíhal v následujících několika fázích:

1. **Návrh a tvorba grafického uživatelského rozhraní (GUI).** Podmínkou, kterou jsem si v této fázi kladl je, aby vytvořené prostředí bylo přehledné, intuitvní a celkově uživatelsky přátelské. Jelikož má být editor jak samostatnou aplikací tak i apletem, tak další podmínkou při realizaci bylo sestavení GUI pouze z jednoho hlavního okna.

2. **Výběr způsobu uchování MathML dokumentu**. Java je vybavena dvěma doplňkovými API pro zpracování XML dokumentů, bylo tedy zapotřebí vybrat vhodnější nástroj pro naše potřeby.

3. **Spojení MathML dokumentu s grafickými komponentami**. Veškeré uživatelské změny při tvorbě a editaci matematického vzorce je zapotřebí ihned promítnout do uchovávaného MathML dokumentu. Je tedy zapotřebí navrhnout systém, který mi bude řešit problém vyhledávání, mazání, přidávání a editaci konkrétních elementů.

4. **Návrh a řešení výstupu výsledného MathML dokumentu**. Ještě před samotnou implementací jednotlivých elementů bylo (už jenom z důvodu testování) zapotřebí zajistit výstup editoru.

5. **Implementace základních MathML prvků** (viz tabulka 2.1)

6. **Implementace MathML elementů umožňující vnořování jiných prvků**. Jde například o element `mfrac` představující zlomek, `mtable` používající se pro konstrukci tabulek či matic a spoustu dalších elementů.

7. **Generování grafiky k některým matematickým objektům** Tou je například odmocnina, tabulka, různé typy závorek jenž se musí přizpůsobovat obsahu a další objekty.

8. Jednou z nejkomplikovanějších fází bylo **programování metod, které se postaraly o vzájemné zarovnání** a přizpůsobení jednotlivých **matematických objektů** jejím **obsahům**. Náročnost vývoje těchto metod byla dána možností neomezeného vnořování elementů.

9. **Zajištění možnosti jednoduchého stylování vytvořených vzorců**. V této etapě vývoje aplikace bylo cílem vytvořit nástroje, které by uživateli nabízely změnu řezu písma, nastavení velikosti, barvy fontu a pozadí.

10. **Implementace ostatních funkcí** jako například možnost vracet se a opakovat jednotlivé kroky při konstrukci vzorce, funkce pro tvorbu nového dokumentu, atd.

11. **Provedení importu uživatelova MathML dokumentu a opětovné sestavení vzorce** pro případné pozdější doplnění a editaci vzorce.

12. **Zajištění internacionalizace programu**. V této části práce jsem řešil automatické přepínání jazykových mutací podle konkrétní lokalizace. Budou k dispozici anglická a česká mutace.

13. **Vytvoření archivů pro distribuci** obsahující zdrojové i spustitelné soubory s potřebnými knihovnami, soubory s jazykovými mutacemi a sestavovacím schématem pro nástroj Ant.

14. Návrh a popis způsobu **komunikace apletu s dokumentem**, v němž je zobrazen.

3.1 Základní metody a třídy editoru

Hlavní vytvořenou třídou obsahující prvně prováděnou metodu `main`() je `Editor`, která je odvozena od `javax.swing.JApplet`. Objekt typu `Editor` představuje jak samostatnou aplikaci tak i aplet. Pro aplet je nezbytná inicializační metoda `init`(), jenž je volána prohlížečem na začátku provádění apletu. V této metodě se mimo jiné volá i `jbInit`() inicializující veškeré komponenty GUI. Pro aplet je také nezbytná implementace metody `start`() umístěné v těle metody `main`(), která spouští aplet v prohlížeči. Tato metoda je prováděna až po metodě `init`(). Automaticky je volána

prohlížečem při uživatelově návratu na stránku s apletem. Pokud je ukončená práce prohlížeče, je volána metoda `destroy`() starající se o uvolnění veškerých prostředků zabraných apletem.

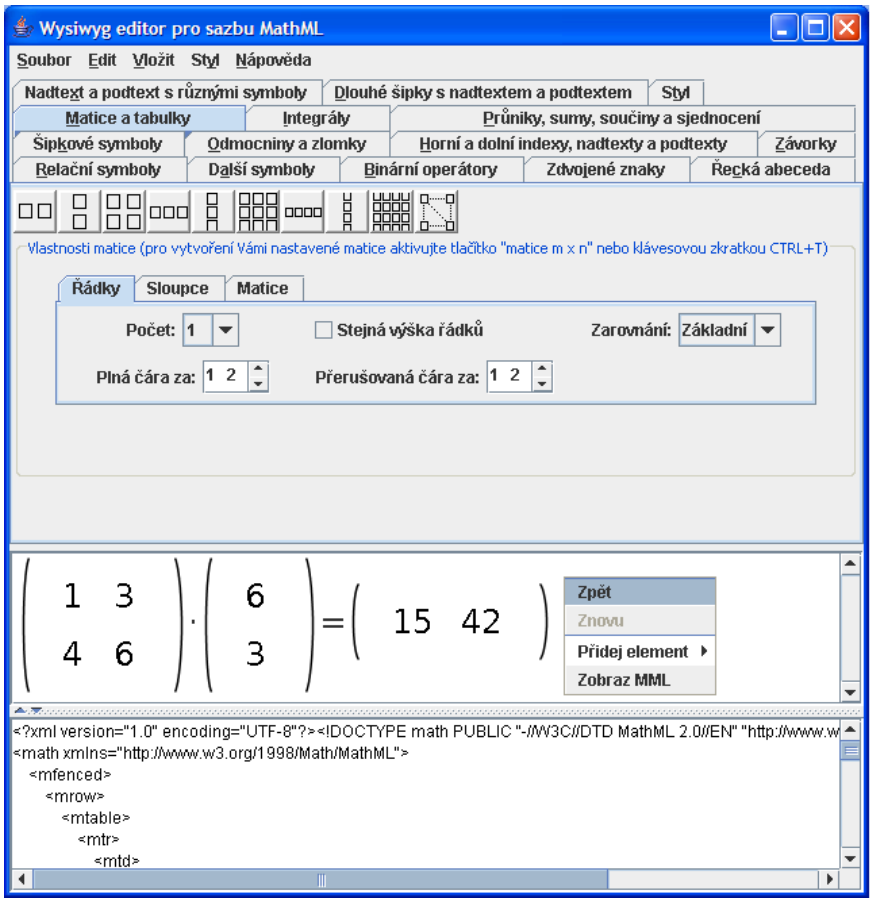
V začátku realizace WYSIWYG editoru bylo nejprve zapotřebí vhodně navrhnout GUI a poté zvolit, jakým způsobem bude k MathML dokumentu vůbec přistupováno. Veškeré změny v instancích třídy `JTextPane` se museli okamžitě odrážet do obsahu MathML dokumentu. Za tímto účelem vznikla třída, jenž definuje přijímač událostí pro všechna potřebná textová pole. Posledním základním problémem bylo vyřešit výstup editoru.

3.1.1 Návrh grafických komponent

Systém NetBeans nabízí programátorům tvorbu GUI pomocí vizuálního vývojového prostředí, kde si lze jednoduše sestrojit vlastní uživatelské rozhraní bez hlubší znalosti grafických komponent. Výhodou tohoto způsobu návrhu je především rychlost a okamžitá názornost bez nutnosti kompilace zdrojových souborů. Je možné si vybírat mezi komponentami ze dvou balíků tříd a to `java.awt` nebo `javax.swing`. Třídy z balíku AWT (sada abstraktních nástrojů pro okna) jsou zastaralé a v současné době se využívají vylepšené prvky GUI tzv. swingové komponenty. Při tvorbě vlastní aplikace jsem dal i přes veškeré zmiňované výhody přednost ručnímu psaní kódu, hlavně z důvodu osvojení si jednotlivých prvků, jejich atributů, událostí a z důvodu většího přehledu nad vlastním kódem. K sestavení aplikace jsem použil knihovnu Swing.

Při návrhu jsem se snažil, aby vytvořené GUI (obrázek 3.1) bylo přehledné a intuitivní, aby nebyl problém pro žádného potencionálního uživatele WYSIWYG editor ihned používat.

V horní části aplikačního okna se nachází nabídková lišta vytvořená standardní třídou `JMenuBar`. Nabídka `Soubor` (`JMenu`) nabízí vytvoření nového MathML dokumentu (`JMenuItem`), načtení uloženého dokumentu, zobrazení MathML kódu odpovídající vytvořenému vzorci a standardní funkci pro ukončení editoru. Posunu v krocích zpět a dopředu lze dosáhnout položkami v nabídce `Edit`. Další nabídku používáme pro rychlejší vkládání nejčastěji používaných matematických objektů. Pod oddělovačem přidaným do nabídky pomocí metody `addSeparator`() třídy `JMenu` nalezneme objekty pro vložení textu, řetězce a neviditelného obsahu. Ke všem těmto ob-



Obrázek 3.1: Vzhled WYSIWYG editoru

jektům i většině položkám v jiných nabídkách jsou pro urychlení volby přiřazeny klávesové zkratky metodou `setAccelerator(KeyStroke keyStroke)` třídy `JMenuItem`. Styl vzorců lze přidávat v předposlední nabídce. Pro nastavení velikosti, barvy fontu a pozadí jsem využil přepínačů definovaných jako objekty typu `JRadioButton`, které jsou seskupeny přidáváním do objektu typu `ButtonGroup`, jenž se stará o stavy těchto tlačítek a zaručuje, že bude v aktivním stavu právě jedno z nich. Řez písma je volen pomocí dvou zaškrtávacích polí typu `JCheckBox`. Poslední nabídka obsahuje pouze informace o aplikaci.

Veškeré matematické objekty, operace, symboly a funkce jsou přehledně uspořádány do záložek, které získáme vytvořením instance třídy `JTabbedPane`. Na obrázku 3.1 je vidět, že veškeré vlastnosti tabulky a matic jsou nastavovatelné opět ve vnořených záložkách. Tímto způsobem je řešena i záložka Styl, do které jsou přidány další objekty typu `JTabbedPane` pro barvu fontu, barvu pozadí a ostatní styl. Pro toto řešení jsem se rozhodl z toho důvodu, abych se vyhnul použití nového

dialogu (instancí třídy `JDialog`) pro vstup dat, jak je to řešené v jiných obdobných programech. Vytvořené matematické objekty se do editovatelného pole vkládají tlačítky, ke kterým jsou přidány ikony metodou `setIcon`, jíž předáváme argument typu `ImageIcon`. Bylo zapotřebí vytvořit celkem 118 ikonových souborů. K tlačítkům pro sazbu symbolů, operátorů a jiných znaků ikony připřazené nejsou, daný znak je přímo vysázen do textu objektu typu `JButton`.

Hlavní částí aplikačního okna je editovatelné pole typu `JTextPane`, jelikož tato textová komponenta narozdíl od `JTextArea` umožňuje nastavovat styl a vkládat jiné komponenty, čehož se využije při vytváření složitějších objektů (zlomek, objekt s obsahem nad i pod základem atd.) K této komponentě a ke všem ostatním stejného typu (budou popsány v pozdějších kapitolách) je připojená instance třídy `JPopupMenu` představující plovoucí kontextové menu vyvolávané pravým tlačítkem myši. Tato nabídka poskytuje vkládání nejpoužívanějších objektů, vracet se a opakovat jednotlivé kroky a hlavně zobrazit požadující MathML kód. Zápis matematického vzorce v MathML bude generován do textové komponenty u spodního okraje editoru. Komponenta má znepřístupněnou editaci, je určená pouze pro zobrazení MathML, k načítání uživatelova zápisu výrazu a pro případné varovné hlášení o chybějících fontech pro korektní zobrazení všech symbolů v nabídce.

3.1.2 Fonty pro WYSIWYG editor

Java veškerý zdrojový kód před kompilací interně převádí do znakové sady Unicode. Všechny speciální znaky vkládané pomocí nabídky jsou zapsány ve formátu Unicode. Zápis řeckého písmena λ (lambda) by vypadal takto \u03BB, kde 03BB je vyjádření znaku v šestnáctkové soustavě. WYSIWYG editor také nabízí vložení zdvojených znaků, které jsou kódovány více jak 2 bajty. Jsou to znaky, jenž nespadají do základní znakové sady a v Javě se zadávají jako dvojice znaků, kde první leží v rozmezí \uD800–\uDBFF a druhý se nachází v rozmezí \uDC00–\uDFFF. Například $\mathbb{A}$ (zdvojené A) se bude zapisovat jako dvojice \uD835\uDD38. Každý znak v tomto programovacím jazyku totiž zabírá 16 bitů. Podpora těchto speciálních znaků je zahrnuta až v Unicode 4 a Javě 5. Pro vysázení všech uvedených znaků jak v nabídce tak i v pracovním textovém poli je zapotřebí nainstalovat pouze jednu sadu fontů [9]. Pokud na lokálním počítači nebude tato sada nainstalovaná, pak

uživatel bude upozorněn hlášením s odkazem ke stažení sady fontů v textovém poli u spodního okraje editoru. Na chybějící fonty bude uživatel upozorněn ihned při spuštění editoru.

3.1.3 Zpracování dokumentu v MathML jazyce

XML dokumenty lze v Javě zpracovávat dvěma aplikačními rozhraními:

- **API pro analýzu XML, nebo-li SAX (Simple API for XML)**. Jde o proudové zpracování dokumentu, který je rozdělen na jednotlivé události (počátek dokumentu, začátek elementu, konec elementu, atd.). Každá událost zavolá metodu (handler), jejíž funkce je plně na programátorovi. Hlavní výhodou tohoto sekvenčně přistupujícího rozhraní je rychlost zpracování dokumentu a menší paměťová náročnost.

- **objektový model dokumentu pro XML, nebo-li DOM (Document Object Model for XML)**. Tento typ zpracování je úplně odlišný. Celý dokument je uložen do stromové struktury v paměti počítače. V Javě tuto strukturu získáme vytvořením instance třídy `Document`. Pomocí metod tohoto objektu lze dokumentem libovolně procházet, vyhledávat konkrétní elementy, nahrazovat je, mazat atd. Hlavní nevýhodou oproti SAX je přítomnost celého dokumentu v paměti. Jednou z podstatných výhod tohoto rozhraní je možnost vytváření a editace existujících dokumentů. SAX tuto přímou možnost vytváření dokumentů a možného uložení do souboru nenabízí.

Z popisu uvedených rozhraní je zřejmé, že pro naše potřeby bude vhodnější využít objektové rozhraní DOM. Vytvoření prázdného MathML dokumentu má za úkol metoda `initDOM()` třídy `Editor`. Hlavní část kódu této metody:

```
// vytvoreni implementace
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
DocumentBuilder builder = factory.newDocumentBuilder();
DOMImplementation impl = builder.getDOMImplementation();

// vytvoreni dokumentu
DocumentType mathml = impl.createDocumentType("math",
                      "-//W3C//DTD MathML 2.0//EN",
                      "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd");
```

```
Document doc;
doc = impl.createDocument("http://www.w3.org/1998/Math/MathML",
                          "math", mathml);
```

Statickou metodou `newInstance`() třídy `DocumentBuilderFactory` získáme objekt, pomocí něhož vytvoříme DOM parser (`builder`). Ten umí rozpoznat jednotlivé součásti dokumentu. Voláním metody `setNamespaceAware`() oznamujeme parseru, že bude pracovat s jmenným prostorem. Abychom získali prázdný dokument, musíme nejprve zavolat metodu `getDOMImplementation`() třídy `DocumentBuilder`, čímž dostaneme instanci třídy `DOMImplementation` a poté zavoláme metodu `createDocument`() vytvořené instance spolu s argumentem udávajícím jmenný prostor MathML. Metodou `createDocumentType`() nastavujeme typ dokumentu s uvedením kořenového elementu a URL adresy DTD.

3.1.4 Model řízení událostí

V Javě jsou události zpracovány tak, že zdroj události (například tlačítko, položka v menu, stisk tlačítka na myši, klávesnici) předává přijímači (objekt naslouchající událostem, tzv. cíl či posluchač) objekt, identifikující danou událost (`ActionEvent`, `KeyEvent`, atd.) a obsahující informace nejenom o dané události, ale i o zdroji. Tento přeposílaný objekt vzniká v okamžiku vyvolání události. Přijímač může být definován tak, že vznikne instance třídy, v níž implementujeme rozhraní posluchače. Těchto rozhraní přijímače existuje mnoho (z důvodu více druhů událostí). Teprve konkrétní implementovaná metoda rozhraní (`actionPerformed`, `valueChanged`, `caretUpdate`, atd.) obsluhuje vzniklou událost. Poslední podmínkou pro fungování obsluhy události je, že naslouchající objekt musí být zaregistrovaný u zdroje (například u instance třídy `JButton` existuje metoda `addActionListener`(), které předáváme jako argument objekt posluchače).

Na schématu v příloze D je zachycen model událostí pro vytvořený WYSIWYG editor. Pro zjednodušení znázornění zde nejsou uvedeny konkrétní zdroje událostí, ale pouze hlavní části aplikace, v nichž se konkrétní zdrojové komponenty nacházejí. Ohodnocení hran na schématu představuje předávaný objekt při vzniku události. Hrana vstupuje do zaregistrovaného posluchače pro objekt události. Jsou zde uvedeny i třídy (`Editor`, `UndoAction`...), ve kterých se přijímače nacházejí. U každého

posluchače jsou navíc zobrazeny jeho akce, jenž jsou implementovány. Například vložení některého matematického objektu je možné několika způsoby (pomocí menu, záložek, vysakovacího menu, klávesovou zkratkou), proto posluchači události pouze volají příslušnou metodu, která daný problém (vysázení objektu) řeší. Tímto způsobem je řešeno více posluchačů. Jak uvidíme později, tak matematické objekty budou složeny z více instancí třídy `JTextPane`, tyto instance budou umístěny na panelu, jenž bude vložen do základního editovatelného textového pole v editoru. Z tohoto důvodu jsou v modelu znázorněny jako zdroje události editovatelná pole. Každý z těchto objektů bude mít zaregistrované uvedené posluchače.

Z modelu například plyne, že stisk standardního windows tlačítka pro ukončení aplikace vyvolá událost `WindowEvent` a u zaregistrovaného posluchače se zavolá implementovaná metoda `windowClosing`(), která ukončí aplikaci a uvolní používané systémové zdroje metodou `exit`() třídy `System`. Konkrétní implementované metody rozhraní posluchačů budou podrobně popsané až v příslušných částech práce.

3.1.5 Propojení MathML dokumentu s komponentami

Pro sledování veškerých změn v hlavním textovém poli je nejvhodnější připojit objekt implementující rozhraní `DocumentListener`, jenž příjímá události vzniklé přidáním, smazáním nebo náhradou znaku či řetězce (viz model v příloze D popsaný v kapitole 3.1.4). Připojený objekt je vytvořeného typu `MMLDocumentListener`.

Před implementací elementů představujících složitější matematické objekty bylo nutné nejprve vyřešit problém rozpoznávání a roztřídění ručně napsaného vzorce v editoru bez využití menu či záložek. Napsaný výraz se musí rozložit na jednotlivé elementy odpovídající číslům (`mn`), operátorům (`mo`) a identifikátorům (`mi`) a ty následně přidat na správné místo do instanční proměnné `doc` typu `Document`. Rozpoznávání jednotlivých elementů je řešeno pomocí velice silného nástroje pro práci s textem, tzv. regulárních výrazů. Tento nástroj je určen pro vyhledávání částí řetězců a nahrazování do požadované podoby. Na následující ukázce kódu z metody `parser`() třídy `Editor` je vidět regulární výraz předávaný metodě `compile`() statické třídy `Pattern` jako argument. Regulární výraz je použit na řetězci znaků hlavního textového pole. Výraz je složen ze tří hlavních částí uzavřených do kulatých závorek. Ohraničením závorkami dosáhneme toho, že pokud se najde část řetězce odpovídající

regulárnímu výrazu uvnitř této konstrukce, pak tato skupina znaků bude zapamatována.

1. část představuje elementy `mn` nebo-li čísla. Tento prvek MathML může být tvořen číslicemi nebo tečkou.

2. část představuje elementy `mo` či-li operátory. Výčet všech možných operátorů je uložen v proměnné `operatory` a jsou zapsány jako Unicode znaky.

3. část vyhledává identifikátory, prvky `mi`. Identifikátory jsou určeny v regulárním výrazu jako doplněk k předchozím dvěma skupinám znaků.

```
String operatory = "";
// unicode pro operatory ze zakladni latinky
operatory += "\\u0021-\\u002D\\u002F\\u003A-\\u0040\\u005B\\u005D";
operatory += "\\u005F\\u0060\\u007B-\\u007E";
// unicode pro operatory z rozsireni latinky
operatory += "\\u00A7\\u00B1\\u00B7\\u00D7\\u00F7";
// unicode pro sipky
operatory += "\\u2190-\\u21FF";
// unicode pro matematicke operatory
operatory += "\\u2200-\\u22FF";
// unicode pro Miscellaneous Technical operatory
operatory += "\\u2200-\\u22FF";
// unicode pro geometrický tvary
operatory += "\\u25A0-\\u25FF";

Pattern vzor = Pattern.compile("([0-9\\.]*)" +
                               "(["+operatory+"]*)" +
                               "([^0-9\\."+operatory+"]*)");
Matcher match = vzor.matcher(str);
```

Jednou z vlastností regulárních výrazů je jejich „hladovost“, tzn. že se snaží nalézt co nejdelší řetězec vyhovující výrazu. Nachází-li se tedy více operátorů v řetězci vedle sebe, pak to regulární výraz vyhodnotí jako jeden element `mo`. V MathML je takový zápis přípustný a validní. Zápisem do WYSIWYG editoru:

```
123+x=-54y
```

se budou ukládat elementy s hodnotami

`mn` → `123`, `mo` → +, `mi` → `x`, `mo` → = −, `mn` → `54`, `mi` → `y`

Registrovaný posluchač hlídá jak vložení tak i mazání každého znaku. Pokud se zavolá jedna z jeho implementovaných metod, pak se vždycky bude volat metoda parser pro celý řetězec objektu typu `JTextPane` a to z důvodu, že lze těžko dohledávat v MathML dokumentu k jaké změně došlo a dokument patřičně upravit (přidat, smazat element).

3.1.6 Výstup editoru

Druh programu nazývaný aplet má z bezpečnostních důvodů implicitně nastavena tato bezpečnostní omezení:

- Aplet má zakázáno přistupovat k souborům na lokálním počítači.
- Aplet nesmí spouštět žádný jiný program na lokálním počítači.
- Aplet může komunikovat pouze s počítačem na němž je umístěna (X)HTML stránka obsahující tento program.

Tuto bezpečnostní politiku lze obejít explicitním povolením některých omezení. Svoji nastavenou bezpečnostní politiku lze uložit více způsoby, nejčastěji se používá soubor. Soubor (`.java.policy`) pro uživatelská nastavení musí být umístěn v domovském adresáři. Kdybychom například chtěli povolit veškerá omezení (zápis do souborů), bude obsah souboru vypadat takto:

```
grant {
  permission java.security.AllPermission;
};
```

Důrazně ale upozorňuji, že tomuto přístupu bychom se měli vyhnout a povolovat pouze potřebná konkrétní omezení. Více o bezpečnostní politice Javy naleznete na stránkách [6].

V editoru jsem se rozhodl, že explicitně žádná omezení povolovat nebudu a tudíž aplikace nebude nabízet možnost zápisu kódu do XML souboru. Výstup tedy bude spočívat pouze v zobrazení MathML zápisu našeho vytvořeného vzorce. V GUI je za tímto účelem přidané textové pole typu `JTextArea`. Toto pole je především určené pro zobrazení touzebného MathML kódu a proto je v něm zakázané editování.

Výstup je už v takové podobě, že stačí pouze vytvořit prázdný XML soubor a vygenerovaný kód do něj vložit.

Generování MathML obstarává metoda `zobrazKod`() třídy `Editor`. Jelikož máme výsledný zápis uložený v objektu typu `Document`, je zapotřebí ho převést na objekt typu `String`, který předáme jako argument při volání metody `setText`() pro naše textové pole. Řetězec obsahující MathML je výstupem metody `DOM2String`(), jejímž argumentem je objekt typu `Document`. V dokumentu jsou uložené ale i pomocné atributy, například pro určování polohy elementů v textovém poli a spousta dalších. Proto si nejprve vytvoříme kopii tohoto dokumentu. Význam a důvod rozšíření elementů o tyto atributy bude rozebrán v dalších kapitolách. Kopie je získána tak, že je metodou `initDOM`() vytvořen prázdný MathML dokument a do něj potom naimportován kořenový element včetně všech podstromů původního dokumentu. K tomu je použita standardní metoda třídy `Document` a to `importNode`(). V novém dokumentu už posléze lze odstranit zmiňované atributy. K tomu byla vytvořená metoda `projdiDom`() třídy `Editor`, která nedělá nic jiného, než že rekurzivně prochází stromovou strukturu a tyto atributy ruší. Převod dokumentu na řetězec řeší následujících pár řádků kódu.

```
String xslt =
    "<xsl:stylesheet version='1.0' " +
    "xmlns:xsl='http://www.w3.org/1999/XSL/Transform' " +
    "xmlns:xalan='http://xml.apache.org/xslt' " +
    "exclude-result-prefixes='xalan'>" +
    "<xsl:output method='xml' indent='yes' xalan:indent-amount='4'/>" +
    "<xsl:template match='@*|node()'>" +
    "<xsl:copy><xsl:apply-templates select='@*|node()'/>" +
    "</xsl:copy></xsl:template></xsl:stylesheet>";
Transformer transformer;
transformer = TransformerFactory.newInstance().newTransformer(
                new StreamSource(new StringReader(xslt)));
transformer.setOutputProperty(OutputKeys.METHOD, "xml");
transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
StreamResult result = new StreamResult(writer);
DOMSource source = new DOMSource(dok);
transformer.transform(source, result);
```

O převod dokumentu na řetězec se stará objekt `transformer` třídy `Transformer` a to konkrétně jeho metodou `transform`(). Proměnná `xslt` obsahuje kód v XSLT

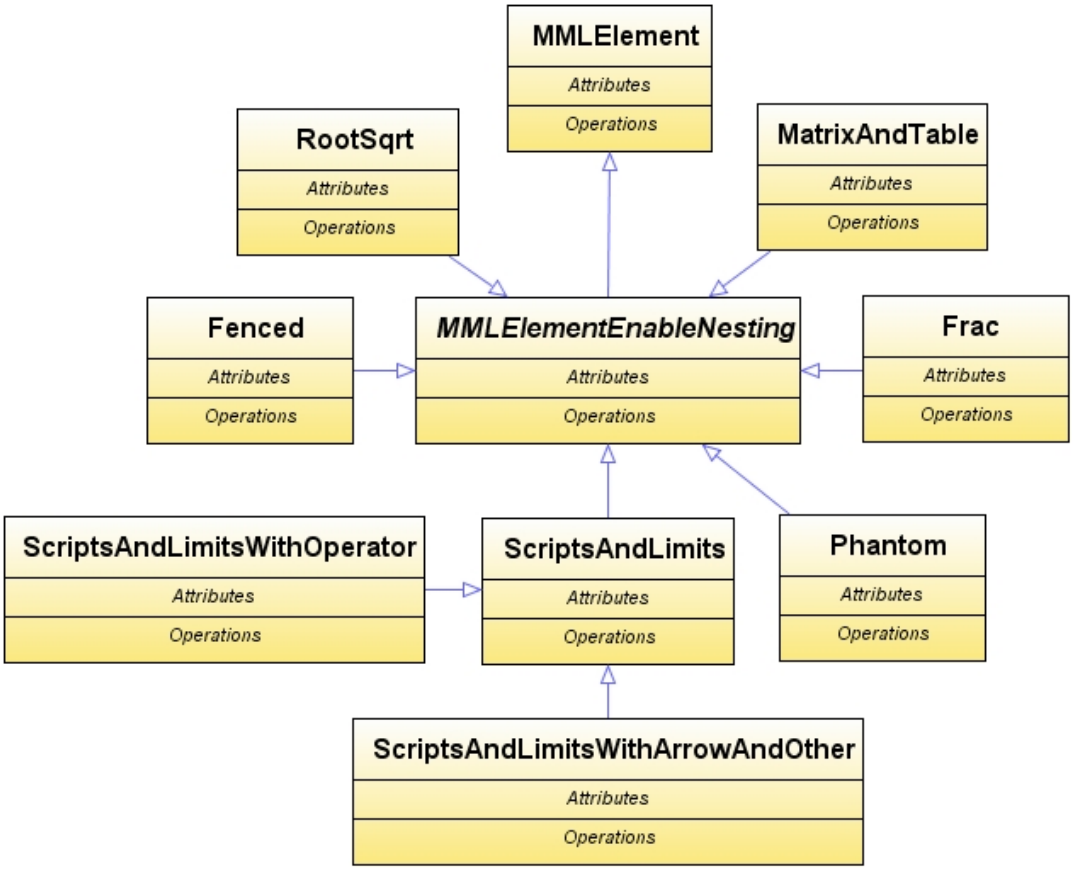
jazyku, který upraví naše MathML tak, aby jednotlivé elementy byly patřičně odsazeny a odřádkovány, jinak bychom měli jednořádkový řetězec, v němž bychom se těžko orientovali.

3.2 Implementace prezentačních elementů MathML

Všechny MathML elementy mají některé společné vlastnosti, jakými jsou například název, rodičovský element, textová komponenta v níž jsou vložené a další. Prvně je tedy definována třída `MMLElement`, která zahrnuje všechny společné atributy i metody a používající se pro vytvoření objektů představujících elementy `mspace`, `mi`, `mo`, `mtext` a `ms`. Z této třídy je odvozená nová třída `MMLElementEnableNesting`, v níž jsou definované atributy a metody společné pro všechny objekty představující elementy, jejichž obsahem už nemusí být prostý text či čísla, ale i jiné MathML prvky. Tato třída je abstraktní a je základní nebo-li rodičovskou třídou pro poslední vrstvu přímých podtříd definujících konkrétní elementy. Celá struktura tříd pro implementaci MathML prvků je zachycena na zjednodušeném UML (objektově orientovaný modelovací jazyk) diagramu (viz obrázek 3.2). Přehled všech datových složek a metod prvních dvou základních tříd je v příloze E.

3.2.1 Řádkování v editoru

Ještě před samotnou implementací MathML elementů je důležité se zmínit o tom, jak se bude promítat odřádkování v hlavním textovém panelu do dokumentu. V komerčním programu MathML .NET Control [13] způsobuje přechod na nový řádek vložení elementu `mtable` na místo potomka kořenového elementu `math`. Prvek `mtable` obsahuje tolik elementů `mtr`, kolik je řádků v editoru. Každý element představující jednotlivé řádky má jediného potomka `mtd`. Tento jediný sloupec je atributem `columnalign` s hodnotou `left` (zarovnaný vlevo). Řádkování je v našem editoru zajištěno stejným způsobem. Pokud posluchač pro hlavní textové pole zaregistruje stisk klávesy ENTER, pak se všechny dosavadní elementy v dokumentu přesunou na úroveň potomků příslušných nově vytvořených `mtd` elementů. Editor patřičně reaguje na přidávání a ubírání libovolných řádků a na základě toho elementy přeskupuje. V případě, že uživatel všechny řádky opět zruší, vrátí se dokument do původní



Obrázek 3.2: Diagram tříd implementující MathML elementy v UML

struktury. V ostatních textových polích, ze kterých budou sestavené složitější matematické objekty, bude řádkování znemožněné (kapitola 3.2.3).

3.2.2 Podpora základních prvků

Základním stavebním kamenem všech matematických objektů jsou elementy popsané v tabulce 2.1. V kapitole 3.1.5 byl popsán způsob rozeznávání elementů z napsaného řetězce. Pokud bychom ale chtěli do editoru zapsat text či řetězec odpovídající elementům `mtext` a `ms`, pak v této fázi vývoje WYSIWYGu by byl vždy zpracován jako identifikátor (element `mi`) pomocí regulárního výrazu. Abychom tedy nějakým způsobem mohli určit, kdy se jedná o jaký objekt, na jaké pozici v textovém panelu je uložen, kde se nachází a pod jakým názvem v našem uchovávaném MathML dokumentu, byla vytvořena jedna ze základních tříd a to zmiňovaná třída `MMLElement`. Objekty této třídy se ještě používají pro sazbu speciálních znaků, operátorů a symbolů, které na klávesnici nenajdeme. Do dokumentu je ukládána entita,

kdežto do textového panelu je vkládán Unicode znak pro zobrazení. Veškeré tyto objekty, které by jinak automaticky nebyly správně rozpoznány, je zapotřebí v dokumentu nějakým způsobem rozlišovat od ostatních prvků. Elementy jsou označené pomocí atributu `menu`. Element je dále rozšířen o informace poskytující aktuální polohu v textovém panelu (atribut `pocPozice` a `konPozice`). Těmito atributy je zajištěno propojení elementu v dokumentu s odpovídajícím objektem v textovém poli. Třída `MMLElement` disponuje jedinou metodou, která je volána v konstruktoru při tvorbě objektu. Cílem této metody `init`() je začlenění nového elementu představujícího daný objekt do MathML DOMu a jeho příprava pro vložení obsahu elementu.

Metoda `init`()

Předpokládejme, že v editoru máme zapsaný následující výraz:

$$1.vlastnost \ldots 5x + 7y \in P$$

Dokument by potom obsahoval uvedenou část kódu:

```
<mtext konPozice="10" menu="true"
pocPozice="0">1.vlastnost...</mtext>
<mn>5</mn><mi>x</mi><mo>+</mo><mn>7</mn><mi>y</mi> <mo
konPozice="19" menu="true" pocPozice="19">&Element;</mo>
<mi>P</mi>
```

Vidíme, že první a předposlední element vznikl vytvořením objektu typu `MMLElement` a zbývající elementy byly vygenerovány metodou `parser`() třídy `Editor`.

V případě, že bychom chtěli vložit například nějaký operátor ze záložek či jiný objekt, metoda `init`() by postupovala podle těchto kroků:

1. Vytvoří se nový element, který je doplněn o počáteční a koncovou pozici v textovém poli. Je označen atributem `menu` a atributem `novy` (význam tohoto atributu bude uveden později při zápisu obsahu do elementu).

2. Určí se rodičovský prvek nově vzniklého elementu. Pokud je textové pole rozděleno do více řádků, je zapotřebí určit správný `mtd` element. Do pomocného objektu typu `Vector` vložíme všechny jeho sourozence. V případě více řádků tedy veškeré prvky ve vyhledaném `mtd`.

3. Prochází se všechny odkazy na prvky uložené v objektu typu `Vector` a určuje se správná pozice pro vložení elementu do dokumentu.

(a) Je-li prvek vložený do obsahu jiného prvku označeného atributem `menu`, pak se vytvoří ještě jeden element stejného typu jako je původní a obsah bude rozdělen podle pozic. Mezi tyto elementy je poté umístěn nově vkládaný prvek.

(b) Element je buď vložený mezi prvky označené atributem `menu`, nebo mezi ostatní elementy. U těchto zbývajících prvků nemusíme řešit, kam umístíme nový element (bude vysvětleno v následující podkapitole věnující se metodě `insertUpdate`()).

4. Posledním krokem je posunutí pozic pravých sourozeneckých elementů.

Nyní pro názornost předpokládejme, že chceme vložit například symbol $\mp$ mezi hodnotu 5 a x tak, aby vzorec měl následující podobu:

$$1.vlastnost \ldots 5 \mp x + 7y \in P$$

V této fázi přidávání symbolu by dokument tvořily tyto upravené prvky:

```
<mtext konPozice="10" menu="true" pocPozice="0">1.vlastnost...</mtext>
<mn>5</mn>
<mo jednoznakovy="true" konPozice="15" menu="true" pocPozice="15"/>
<mi>x</mi><mo>+</mo><mn>7</mn><mi>y</mi>
<mo konPozice="20" menu="true" pocPozice="20">&Element;</mo>
<mi>P</mi>
```

O samotné vložení obsahu se stará až posluchač typu `MMLDocumentListener` pro dané textové pole. Hodnota už v této fázi není vložená do obsahu elementu, protože cílem bylo vytvořit univerzální systém pro veškeré MathML prvky. V případě vložení jednoznakového elementu by to možné bylo, ale pokud bychom se rozhodli vložit například řetězec představovaný elementem `ms`, pak v tomto okamžiku ještě nevíme co bude obsahem prvku a proto ukládání hodnoty přenecháme až na instanci konkrétního přijímače.

Metoda `insertUpdate`()

Tato implementovaná metoda rozhraní posluchače reaguje na vložení znaku do textového pole. Nejprve se určí pozice změny v textovém poli a na základě té se prohledává MathML dokument. Mohou nastat tři situace:

1. Počáteční pozice některého z elementů uložených v dokumentu je shodná s pozicí změny. V této situaci se musí zjistit, zda nejde o nově vytvořený prázdný

element podle toho, zda je rozšířen o atribut `novy`. Pokud atribut existuje, pak se pozice pravých sourozenců upravovat nemusí, to bylo provedeno v metodě `init`() při vytváření instance třídy `MMLElement`. Musí se pouze odstranit atribut `novy`, protože při dalším vyvolání této metody už se o nový element nebude jednat. V opačném případě je ještě nutné určit, zda nejde o element představující operátor, symbol či funkci vloženou přes záložky. Tyto speciální znaky jsou v dokumentu představovány elementy označený atributem `jednoznakovy`. Tento atribut je přidáván při vzniku objektu `MMLElement` v metodě `init`(). Obsah elementu je tvořen entitou znázorňující daný znak a proto ho nelze doplňovat o jiné znaky. Nejedná-li se o tento jednoznakový element, pak jde o přidání znaku do existujícího elementu a pozice pravých sourozenců se musí obnovit. Pokud je znak vložen na pozici jednoznakového elementu pak se pozice elementů počínaje tímto prvkem posunou.

2. Pozice změny v textovém poli odpovídá pozici uvnitř obsahu elementu. Obsah prvku je upraven včetně jeho koncové pozice a pozice všech pravých sourozenců, u kterých tuto informaci máme poznamenanou.

3. Pozice kurzoru textového pole neleží v rozmezí pozic žádného uloženého elementu. V takovém případě se posunou pozice pravých sourozeneckých elementů označených atributem `menu`.

Nyní je již dokument v uvedené podobě. Před výstupem budou z dokumentu odstraněny pomocné atributy.

```
<mtext konPozice="10" menu="true" pocPozice="0">1.vlastnost...</mtext>
<mn>5</mn>
<mo jednoznakovy="true" konPozice="15" menu="true" pocPozice="15">
    &amp;mp;
</mo>
<mi>x</mi><mo>+</mo><mn>7</mn><mi>y</mi>
<mo konPozice="20" menu="true" pocPozice="20">&Element;</mo>
<mi>P</mi>
```

V prvním a třetím případě mohlo dojít k přidání znaku uživatelem a tudíž se nevytvářela instance třídy `MMLElement`. V tomto okamžiku nevíme o jaký element se jedná a proto se nejprve odstraní veškeré neoznačené sourozenecké elementy a potom se zavolá metoda `pruchod`() tohoto přijímače. Ke smazání neoznačených prvků došlo z důvodu, že by bylo zbytečné nějakým způsobem určovat, zda jde o nový prvek či

se nějaký původní rozšiřuje a podobně. Jak se dále dočteme, elementy budou totiž záhy opět vytvořeny.

Úkolem metody `pruchod`() je procházení elementů, které byly předány jako argument této metodě. Jedná se o prvky vytvořené objekty typu `MMLElement`. Na základě pozic těchto elementů se budou určovat části řetězců v textovém panelu, které se nacházejí mimo rozsah těchto pozic a neodpovídají žádné instanci třídy `MMLElement`. Tyto výseky z řetězce textového pole jsou předávány metodě `parser`(), jejíž činnost je popsaná v kapitole 3.1.5. Jenom ve zkratce, tato metoda rozděluje řetězec na části odpovídající konkrétním elementům pomocí regulárních výrazů a tyto výseky jsou ukládány do nově vytvořených elementů.

Ukažme si příklad, kde bychom místo entity upravovali první hodnotu elementu `mn` ($5 \rightarrow 54$) našeho ilustrativního příkladu. Nejprve bychom odstranili všechny neoznačené elementy. Dokument by tedy vypadal následovně:

```
<mtext konPozice="10" menu="true" pocPozice="0">1.vlastnost...</mtext>
<mo konPozice="20" menu="true" pocPozice="20">&Element;</mo>
```

Následně by byly metodě `parser`() předávány postupně řetězce neodpovídající označeným elementům a metoda by tyto řetězce analyzovala a vytvářela nové MathML prvky. Jejím úkolem je i vložení těchto nových prvků na správnou pozici do dokumentu.

$54x + 7y : 54 \rightarrow$ `mn`, $x \rightarrow$ `mi`, $+ \rightarrow$ `mo`, $7 \rightarrow$ `mn`, $y \rightarrow$ `mi`

$P : P \rightarrow$ `mi`

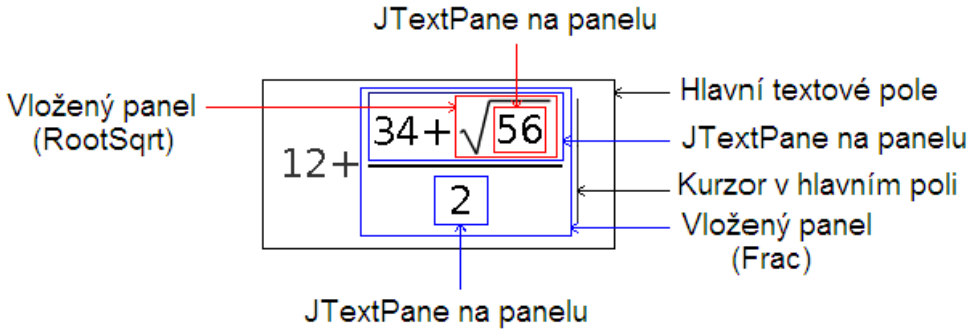
Element `mspace` představující mezeru o velikosti 0.2 em nastavenou atributem `height` vzniká v metodě `textPanelKeyPressed`(), která zprostředkovaně obsluhuje událost vyvolanou stiskem mezerníku. Metoda vytvoří objekt typu `MMLElement` a nastaví atribut pro jednoznakový prvek. Stejně tomu je u znaků vložených přes záložky. Jediným rozdílem je, že element `mspace` je prázdný.

Z popisu činností metod pro vkládání nových základních matematických objektů je vidět, že tento proces není jednoduchou záležitostí a musí se brát v úvahu všechny možné situace, jenž mohou nastat. Obdobným způsobem je řešena i metoda `removeUpdate`(), která se stará o mazání znaků z textového pole a tím tedy i o rušení elementů uložených v dokumentu. Opět se zde analyzuje řetězec v textovém poli, na základě něhož jsou upravovány elementy v dokumentu. V případě, že je část vzorce

označená a nahrazená za jiný objekt, je nejprve volána metoda `removeUpdate`() a po té metoda `insertUpdate`().

3.2.3 Podpora ostatních prvků

Pro vytváření složitějších matematických objektů, jako je například zlomek, odmocnina nebo základ s indexy, je definovaná abstraktní třída `MMLElementEnableNesting`, z níž jsou teprve odvozeny třídy pro konkrétní objekty. Přehled všech atributů a metod je v příloze E. Hlavní textové pole představuje záměrně objekt typu `JTextPane`. Jednou z jeho vlastností je totiž možnost vkládání komponent. Všechny složitější matematické objekty jsou tvořeny vrstveným panelem, který je právě možné vkládat do textového pole zmiňovaného typu. Ve vrstveném panelu jsou opět umístěny komponenty typu `JTextPane`, čímž je umožněno neustálé vnořování objektů do sebe. Vzorec tedy bude strukturovaný podle obrázku 3.3. Pro každé textové pole objektu bude rovněž zaregistrovaný posluchač typu `MMLDocumentListener`, který bude obsluhovat vzniklé události.



Obrázek 3.3: Uspořádání komponent představující vzorec

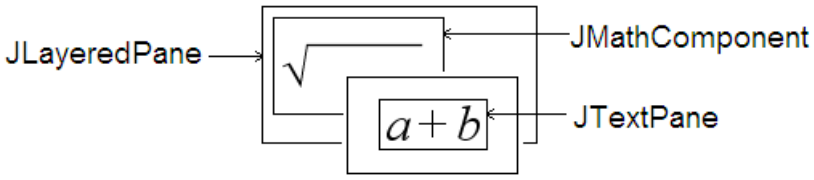
Možnosti systému JEuclid

Pro sazbu odmocninové čáry, zlomkových čar, různých typů závorek, tabulek a jiné potřebné grafiky je v původním řešení použit systém JEuclid určený právě pro MathML. Tento kompletní balík v sobě zahrnuje například aplikaci MathViewer, jenž zobrazuje MathML zápis, řádkový konvertor z MathML do jiných formátů, grafické komponenty Swing či AWT a jiné nástroje. Více informací o tomto projektu naleznete na adrese [11]. V realizovaném editoru jsem použil pouze grafickou komponentu `JMathComponent`. Předáme-li této komponentě jednou z metod

(`setContent(String str)`, `setDocument(Document doc)`) MathML zápis, pak se postará o jeho vizualizaci.

Původní řešení přizpůsobené nástrojům JEuclid

Každý složitější matematický objekt je tvořen vrstvenou plochou (`JLayeredPane`), v níž můžeme jednotlivé komponenty hloubkově uspořádat (viz obr. 3.4). Na tento panel je v nejnižší vrstvě přidána JEuclid komponenta a ve vrstvě vyšší je umístěn textový panel (`JTextPane`), který překrývá obsah. I přesto že textový panel je umístěn přes obsah objektu, jsou veškeré vnořené elementy ukládané do `JMathComponent` vloženy do prvku `mphantom` a tak na komponentu vysázeny nebudou s tím, že svou velikost si budou zachovávat. Součástí každé instance třídy odvozené od rodičovské třídy `MMLElementEnableNesting` je dokument, ve kterém je uložen pouze příslušný element představující celý objekt včetně všech jeho dětí (v příkladě by se jednalo o prvek `msqrt`). Tento upravený dokument je při každé změně v některých z jeho textových polí předáván komponentě `JMathComponent`, čímž je zajištěno přizpůsobování grafiky našemu obsahu.



Obrázek 3.4: Původní návrh sazby grafiky

Vytváření objektů a propojení MathML s komponentami

Jelikož editor bude umožňovat vnořování objektů a každý objekt si bude uchovávat svůj MathML dokument z důvodu sazby grafiky na komponentu `JMathComponent`, je zapotřebí ukládat informace o tom, v jaké textovém poli se zrovna nacházíme a do jakého MathML se budou promítat změny z tohoto pole. Za tímto účelem existují statické složky `aktTextKomp`, `aktDokument` třídy `Editor` (nastavení těchto atributů bude popsáno v kapitole 3.3.1). Konstruktor při vytváření objektů (např. typu: `Frac`, `RootSqrt`, `Phantom`, atd.) přidá nový element včetně jeho dětí (zatím prázdné elementy `mrow`) představující tento objekt do aktuálního MathML dokumentu na správnou pozici. Zároveň vytvoří nový dokument, jenž bude předáván komponentě pro vysázení grafiky a to metodou `initDOM`(). Do tohoto dokumentu je

přidán stejný element co se vkládal do aktuálního dokumentu, pouze místo dětí typu `mrow` budou prvky `mphantom`, aby byl obsah neviditelný. Tento element bude navíc označen pomocným atributem `vnorovani`, podle něhož budeme vědět, že element představuje objekt umožňující vkládání jiných prvků.

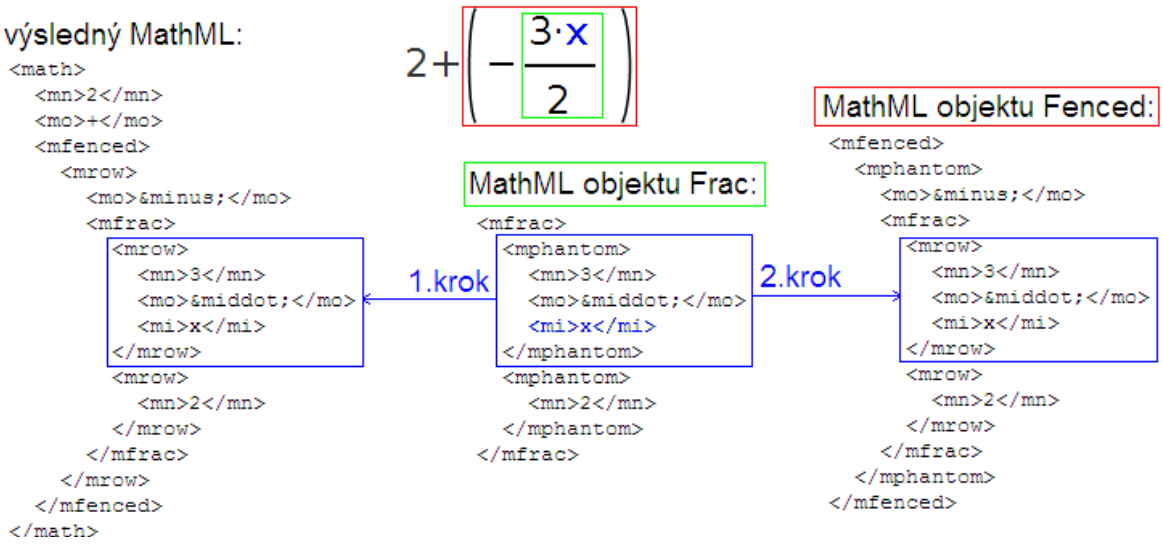
V později popsaných metodách je někdy zapotřebí procházet výsledný MathML dokument s tím, že musíme vědět co daný element představuje za komponentu neboli vědět, kde se fyzicky nacházíme. Například zlomek je představován komponentou `JPanel` a čitatel s jmenovatelem jsou tvořeny komponentami `JTextPanel`. Je tedy zapotřebí spojit element `mfrac` s komponentou `JPanel` a elementy `mrow` s komponentami `JTextPanel`. Z tohoto důvodu byl definován atribut `citacMatKomp` v třídě `Editor`. Ten je při každém vytváření objektu složeného z komponent zvětšen o jejich počet. Komponentě je přiřazeno jednoznačné jméno složené z názvu odpovídajícího elementu a tohoto čísla. Element je zase rozšířen o atribut `id` obsahující vytvořenou unikátní hodnotu. Pro zajištění obousměrného propojení elementu s komponentou nám standardizované API rozhraní pro XML nabízí metodu `setUserData`(), pomocí které element rozšíříme o uživatelská data, v našem případě tedy o odkaz na objekt typu propojované komponenty.

Při vytváření objektů je tedy nutné nastavit popisované propojení. Posledním úkolem konstruktoru je inicializace grafických komponent pro vizualizaci objektu metodou `jbInit`(). Jaké komponenty se budou inicializovat je patrné z obrázku 3.4. V metodě `jbInit`() jsou také zaregistrované posluchače pro veškerá textová pole.

Proces ukládání změny v textovém panelu

Vznikne-li událost, jejímž zdrojem je instance třídy `JTextPane`, pak tato změna bude zpracována implementovanými metodami třídy `MMLDocumentListener`, tak, jak je popsáno v kapitole 3.2.2. Na konci těchto metod se navíc testuje, zda objekt umožňuje vkládání jiných prvků (atribut `vnorovani`). V kladném případě je zavolána metoda `importElementu`() a potom `prizpusobeniZmeneObsahu`(). Druhé uvedené metodě se budeme věnovat později. Tato metoda kromě jiného předává dokument objektu komponentě `JMathComponent`, čímž je přizpůsobena grafika obsahu.

Změna se zatím projevila pouze do dokumentu objektu a proto musíme nějakým způsobem oznámit nadřazeným prvkům, že došlo k editaci a je nutné upravit jejich dokumenty. Tento cíl má stanovený právě metoda `importElementu`() (viz obr. 3.5).



Obrázek 3.5: Činnost metody `importElementu`()

Nejprve vyhledá element spojený s komponentou v níž došlo ke změně ve výsledném MathML dokumentu a to podle zavedeného atributu `id` standardní metodou `getElementById`(). Potom jsou ve výsledném dokumentu všechny děti nalezeného prvku odstraněny a znovu naimportovány elementy z dokumentu objektu, aby se nemuselo zjišťovat, jaká změna nastala a jakých elementů se týká. Pro import elementů z jednoho dokumentu do jiného existuje metoda API, tzv. `importNode`(). Ta ale bohužel nekopíruje uživatelská data a proto byla vytvořena vlastní metoda `myImportNode`() fungující obdobně. Poslední pomocnou metodou využitou při importu prvků je `myReplaceElement`(), která zaměňuje názvy prvků `mphantom` za `mrow` (pokud nejde o prvek `mphantom` vložený uživatelem). Tím jsou zkopírovány všechny prvky z dokumentu objektu do výsledného dokumentu. Další částí metody je importování elementů do dokumentů nadřazených objektů, čehož je dosaženo voláním metody `importPredci`(). Jejím argumentem je element představující objekt z jeho dokumentu. Jde o rekurzivní metodu, která nejprve vyhledá ekvivalentní element předanému argumentu ve výsledném dokumentu. Poté zjistí jeho předka a určí zda, jde o prvek umožňující vnořování (podle pomocného atributu). V kladném případě se získá odkaz na instanci představující matematický objekt. Metoda `getUserData`() vrací objekt typu `Object`. Jde o univerzální nadtřídu všech tříd v Javě. Objekt tohoto typu je tedy zapotřebí ještě přetypovat. Jelikož přesně nevíme o jaký konkrétní typ objektu se jedná (`Frac`, `MatrixAndTable`, `ScriptsAndLimits`...), odkazujeme

se na něj pomocí společného předka typu `MMLElementEnableNesting` (tzv. polymorfismus). Jednou z datových složek instance je právě jeho dokument, do kterého naimportujeme změněné elementy. Tímto způsobem se rekurzivně obnoví všechny dokumenty nadřazených objektů. V následujícím zjednodušeném kódu je popisovaná metoda.

```
protected void importPredci(Element element, boolean importMstyle) {
    Element element_doc = getElementById(element.getAttribute("id"),
                                         Editor.doc);
    Element predek = (Element) element_doc.getParentNode();

    while (predek != null && !predek.getNodeName().equals(Editor.MATH)){
        if (!predek.getAttribute("vnorovani").equals("") &&
             predek.getAttribute("textovaKomponenta").equals("") &&
            !predek.getNodeName().equals(Editor.MSTYLE)) {

            MMLElementEnableNesting objekt =
              (MMLElementEnableNesting) predek.getUserData("instance");
            ...
            Element dup;
            if (importMstyle) {...}
            else
                dup = myImportNode(element, objekt.docElement, true);
            ...
            Element docEl =
              (Element) objekt.docElement.getDocumentElement();
            Element firstCh = docEl.getFirstChild().getFirstChild();
            importPredci(firstCh, false);
        }
        predek = (Element) predek.getParentNode();
    }
}
```

Při testování sazby grafiky pro první realizovaný matematický objekt, kterým byl zlomek, bylo zjištěno, že zlomková čára se vertikálně posouvá na základě výšky znaků ve jmenovateli nebo čitateli. Tzn. byl-li tvořen znak dolním dotahem v čitateli (např. při vložení textu obsahující znak p), pak zlomková čára byla odsunuta přes jmenovatel a naopak obsahoval-li znak ve jmenovateli horní dotah (např. znak d), grafika byla vysázena přes čitatel. Problém mohl být vyřešen větším odstupem textových polí, což by ale výsledný vzorec zbytečně roztáhlo a při psaní složitějšího výrazu by mohl být výstup nepřehledný. Další nevýhodou použití komponenty `JMathComponent` bylo, že k vlastní knihovně bylo zapotřebí připojit celý

balík JEuclid, přestože využita jediná komponenta. Zároveň pokud by aplikace měla být i apletem, muselo by se povolit jedno z bezpečnostních omezení, které zakazuje používání nestandardních knihoven.

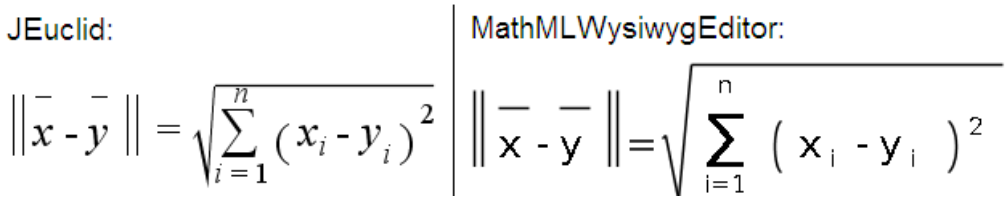
Vlastní řešení grafiky

Vlastní řešení sazby grafiky spočívá v založení nové třídy `Grafika` odvozené od základní grafické komponenty `JComponent`. Navrhnutý systém pro ukládání změn a vytváření objektů je zachován, jen je pouze místo komponenty `JMathComponent` používaný objekt typu `Grafika`. Třída `JComponent` obsahuje metodu `paintComponent`(), jejímž argumentem je tzv. grafický kontext zapouzdřený v objektu typu `Graphics`. Instance tříd grafického kontextu jsou používány pro kreslení čar, křivek, tvarů, vyplněných tvarů a obrázků. Pro kreslení v komponentě se využívají nástroje poskytované objektem typu `Graphics2D`, což je novější kontext odvozený od uváděné třídy `Graphics`. Pro vykreslení něčeho vlastního tedy používáme metodu `paintComponent`() a grafický kontext definovaný třídou `Graphics2D`. Veškeré vykreslené tvary jsou složené z přímek a oblouků (definovány třídou `Arc2D.Double`).

Při vytváření instance třídy `Grafika` je konstruktoru předáván jako argument objekt typu `Rectangle` představující obdélník s veřejnými datovými členy definujícími levý horní roh, šířku a výšku obdélníku. Tento obdélník vyznačuje umístění textového pole na panelu, kolem kterého se má kreslit požadovaný tvar. Dalším argumentem konstruktoru je objekt typu `MMLElementEnableNesting`, abychom mohli určit o jaký matematický objekt se jedná a mohli přistupovat k jeho datovým složkám potřebným pro vykreslení. Při vytváření matematického objektu přidáváme na vrstvenou plochu instanci třídy `Grafika` (jako v původním řešení využívající systém JEuclid). Pro přizpůsobení grafiky změnám v textových polích není předáván MathML dokument, ale jsou pouze upravené pozice obdélníku a údaje o jeho velikosti. Na obrázku 3.6 můžeme porovnat vlastní řešení použité v editoru s výstupem aplikace MathViewer, která je součástí balíku JEuclid.

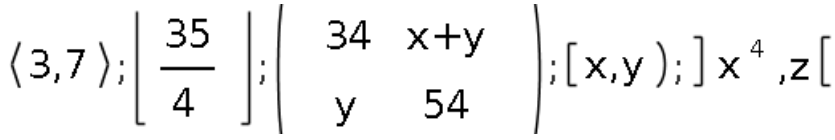
Objekty typu `Fenced`

V této části kapitoly jsou popsány již samostatné třídy definující konkrétní matematické objekty. Jednou z nich je třída `Fenced`. Objektem tohoto typu můžeme ohraničit libovolnou část vzorce od jednořádkového výrazu až po matice či jiné vy-



Obrázek 3.6: Srovnání výstupu systému JEuclid se sazbou WYSIWYG editoru

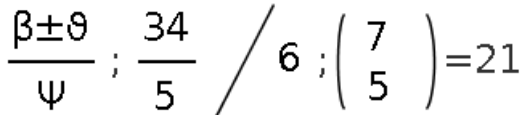
tvořené matematické objekty (mohou být do sebe i vnořovány, závorky se jim vždy přizpůsobí). Na výběr máme ze široké škály typů, například klasické oblé, hranaté, složené, kolmé, úhlové nebo méně používané konstrukce pro označení dolní a horní celé části, kombinované závorky a další. Všechny typy závorek jsou vykreslené na komponentě `Grafika`, kterou v metodě `jbInit`() přidáváme na hlavní panel představující tento objekt. Sazbu některých závorek vidíme na obr. 3.7.



Obrázek 3.7: Ukázka sazby elementu `mfenced`

Objekty typu `Frac`

Sazbu zlomků zajišťuje instance třídy `Frac` (viz obr. 3.8). Jedním z argumentů konstruktoru si vybíráme mezi třemi typy zápisů. Kromě klasického zobrazení můžeme zvolit i zápis zlomku do řádku nebo výpis bez zlomkové čáry. Poslední typ najde uplatnění pro sazbu kombinačních čísel, resp. binomických koueficientů. Internetovým prohlížečům oznamujeme nestandardní typ zápisu zlomku pomocí atributů `bevelled` s hodnotou `true` (zápis do řádky) a `linethickness` s hodnotou `0` (tloušťka zlomkové čáry).



Obrázek 3.8: Ukázka sazby elementu `mfrac`

Objekty typu `MatrixAndTable`

Třídou `MatrixAndTable` definujeme matice a tabulky (viz obr. 3.9). Konstruktoru tohoto objektu předáváme jak počet sloupců a řádků tak i seznam atributů uložený

v instanci třídy `ArrayList`. Vlastnosti matic a tabulek nastavujeme v příslušné záložce. Volíme si zde mezi způsobem zarovnání řádků či sloupců, zda mají být stejně vysoké řádky či stejně široké sloupce. Na této záložce také najdeme seznamy řádků a sloupců. Jejich výběrem určujeme oddělení plnou či přerušovanou čárou. V záložce matice vybíráme jeden ze tří typů orámování tabulky či matice. Po nastavení těchto voleb stiskneme tlačítko pro sazbu matice m x n. Následně bude vygenerována matice či tabulka s vybranými vlastnostmi. O vykreslení oddělovacích čar či orámování se opět stará objekt typu `Grafika`.

$$\left(\begin{array}{ccc|c} 1 & \alpha & \alpha^2 & 1 \\ 1 & \alpha & \alpha\beta & \alpha \\ \beta & \alpha^2 & \alpha^2\beta & \alpha^2\beta \end{array}\right);$$

x	2	$\frac{13}{3}$	566
y	34	433	44322

Obrázek 3.9: Ukázka sazby elementu `mtable`

Objekty typu `Phantom`

Tento objekt je určen pro sazbu částí vzorců, které nebudou zobrazeny, ale svoji velikost si budou ponechávat. Instance třídy `Phantom` je složena pouze z panelu a jednoho textového pole. Své uplatnění může najít u soustav rovnic, jenž mají být zarovnány například podle proměnných (viz obr. 3.10). Jelikož se jedná o třídu odvozenou od `MMLElementEnableNesting`, můžeme do textového pole vnořovat libovolné matematické objekty. Veškerý obsah se stane neviditelným.

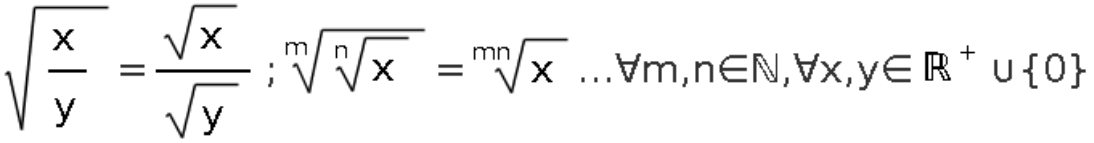
$$\begin{aligned} -2x_1 + x_2 + 5x_3 &= 19 \\ 3x_2 + x_3 &= 9 \\ 2x_3 &= 6 \end{aligned}$$

Obrázek 3.10: Ukázka sazby elementu `mphantom`

Objekty typu `RootSqrt`

Je-li náš vzorec složen z druhé či n-té odmocniny, pak byl použit objekt typu `RootSqrt` (viz obr. 3.11). Konstruktoru tohoto objektu předáváme argument typu `Boolean`, který pokud je pravdivý, pak se bude jednat o n-tou odmocninu předsta-

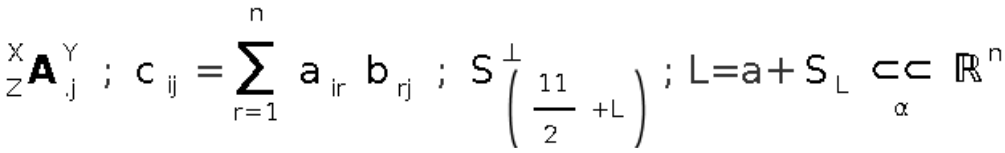
vovanou elementem `mroot`. Pokud bude hodnota nepravdivá, pak se bude jednat o druhou odmocninu či-li element `msqrt`. V případě n-té odmocniny bude grafika vykreslována podle pozic, výšky a šířky základu tvořeného objektem `JTextPane`.



Obrázek 3.11: Ukázka sazby elementů `msqrt` a `mroot`

Objekty typu `ScriptsAndLimits`

Tímto objektem můžeme vysázet elementy `msup`, `msub`, `msubsup`, `mover`, `munder`, `munderover` a `mmultiscritp` (viz obr. 3.12). Tento objekt nám také umožňuje vytvářet více úrovní indexů a exponentů před i za základem. Totéž platí i pro prvky připojené nad či pod základ. Tato třída je navrhnuta tak, že teprve až při vytvoření její instance se zadává základ a k němu připojené indexy či jiné prvky. Tzn. že nelze objektem tohoto typu připojit indexy, exponenty atd. k existujícím matematickým prvkům. Chceme-li zadat pouze některý z indexů či exponentů u objektu představujícím element `mmultiscripts`, pak ostatní textová pole necháme nevyplněná, čímž bude vysázen jenom námi zadaný obsah.



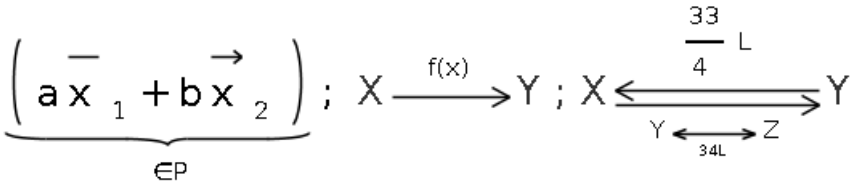
Obrázek 3.12: Ukázka sazby elementů `msub`, `munder`, `mmultiscripts` a dalších

Pro první úroveň je automaticky použité písmo o stupeň menší než je v základu, pro druhou úroveň o dva stupně atd. Velikost písma v základu je určována podle fontu aktuální textové komponenty, do níž je vkládán tento objekt (odkaz na tuto komponentu je uložen ve statické datové složce `aktTextKomp` třídy Editor).

Objekty typu `ScriptsAndLimitsWithArrowAndOther`

Třída definující tento typ objektu má zděděné atributy a metody od základní třídy `ScriptsAndLimits`. Matematické objekty tohoto typu (viz obr. 3.13) lze získat i pomocí rodičovské třídy, kde do indexů, mocnin, nadtextů či podtextů vložíme pří-

slušný operátor. Tento typ objektů ale uživateli usnadňuje sestavení požadovaného matematického vzorce. Dalším rozdílem při použití objektu typu základní třídy je, že operátory (šipky, derivace...) se nebudou přizpůsobovat (roztahovat, měnit pozici...) vloženému výrazu, kdežto u objektů tohoto typu se tyto operátory přizpůsobovat budou, jelikož jsou vykresleny na komponentu `Grafika`. Tyto operátory tak nelze měnit, jsou napevno vysázeny na komponentu. Všechny instance této třídy lze vytvořit tlačítky na záložkách `Nadtext a podtext s různými symboly`, `Dlouhé šipky s nadtextem a podtextem`.



Obrázek 3.13: Ukázka mat. objektů složených z prvků kategorie schémat textů a krajních hodnot

Objekty typu `SriptsAndLimitsWithOperator`

Na záložce `Integrály a Průniky, sumy, součiny a sjednocení` jsou objekty definované třídou `SriptsAndLimitsWithOperator`. Tato třída je odvozená od rodičovské třídy `ScriptsAndLimits`. Operátory jako je součin, sjednocení, sumy atd. nejsou vykreslovány, ale tvořeny znakem většího fontu než je v aktuální textové komponentě (viz obr. 3.14). V textovém poli, v němž je zobrazen operátor, není povolena editace a proto operátor nelze smazat. Stejně jako u předchozího popisovaného typu objektu je možné i tento typ realizovat pomocí indexů, exponentů, nadtextů a podtextů připojených k základu, do kterého se vloží příslušný operátor.

Obrázek 3.14: Ukázka mat. objektů tvořených pomocí některých operátorů

3.2.4 Zarovnání a přizpůsobení matematických objektů

V tomto stádiu realizace wysiwyg editoru, byly vytvořeny veškeré matematické objekty a tak zajištěna implementace většiny prezentačních prvků jazyka MathML. Všechny objekty popsané v kapitole 3.2.3 je zapotřebí ještě přizpůsobovat přidávaným nebo mazaným znakům či vnořeným objektům. Pod přízpůsobením je myšleno zvětšení resp. zmenšení textového pole, v němž došlo ke změně, následně překreslení grafické komponenty a nakonec úprava velikosti panelu, na který jsou všechny komponenty přidány. U zlomků, n-tých odmocnin, indexů, exponentů a dalších objektů složených z více vstupních textových panelů je zapotřebí hlídat, aby si i ostatní panely, u nichž posluchač nezaregistroval změnu, zachovávali předepsané odstupy a rozložení na panelu.

Metoda `prizpusobeniZmeneObsahu`()

Tato metoda je již definovaná v abstraktní třídě `MMLElementEnableNesting`. V odvozených třídách je potom překryta pro konkrétní typ matematického objektu. Pokud událost vyvolá obsluhu u posluchače typu `MMLDocumentListener`, pak pro objekty označené v MathML dokumentu atributem `vnorovani` je volána společně s metodou `importElementu`() i tato metoda. Jejím cílem je určit šířku pro textové pole tak, aby odpovídala jeho obsahu. Hodnota je stanovena podle šířky řetězce (metodou `getFontMetrics`()) a šířek vložených komponent. Podle nově nastavených rozměrů vstupního pole jsou upraveny pozice ostatních textových panelů. Například dojde-li ke změně v čitateli a jmenovatel je užší, pak se jmenovatel musí znovu centrovat podle nové šířky čitatele. U každého typu objektu jsou textová pole jinak rozmístěna a proto implementace metody `prizpusobeniZmeneObsahu`() je přenechána až na konkrétní třídu. Objekty do sebe mohou být vnořovány a proto se nová šířka panelu musí nějakým způsobem předat nadřazeným instancím. Je tak volána metoda `vnoreni`(), jejímž argumentem je instance třídy `JTextPane`, tedy rozměrově upravené textové pole, a informace o metodě, jenž má být volána pro nadřazené textové pole (jde o metody `prizpusobeniZmeneObsahu`(), `vypocetNovychVysek`(), `prizpusobeniVyskyZmeneObsahu`()). Jediným cílem této metody je získat odkaz na textové pole, ve kterém je uložené aktuální pole. Pro získané nadřazené pole se bude opět volat metoda `prizpusobeniZmeneObsahu`() pro přizpůsobení nejen jeho, ale

i celého objektu, ve kterém je toto nadřazené pole umístěno. Tímto způsobem se bude pokračovat, dokud nezískáme odkaz na hlavní textové pole. Jde o tzv. nepřímou rekurzi.

Při vzniku instance třídy odvozené od `MMLElementEnableNesting` vytvoříme celý matematický objekt, který se následně vloží do aktuálního textového pole, což vyvolá událost, jenž obsluhuje zaregistrovaný objekt typu `MMLDocumentListener`. V implementovaných metodách příjimače je volána pro objekt výše popisovaná metoda, která pouze přizpůsobí šířku panelů provedené změně. Při vkládání složitějších objektů je ale zapotřebí přizpůsobit i výšky panelů a provést vzájemné zarovnání, které je provedeno v metodě `vlozMMLElementEnableNesting`(), v níž je vytvářená i příslušná instance. Obsluha událostí a vykreslování GUI je zajištěno pouze v jednom vlákně, zvaného `event-dispatching thread`. Události se obslouží až po skončení předchozí obsluhy a nepřerušují vykreslování. Většinu operací s GUI komponentami je nutné provádět v uvedeném vlákně. Aby byl kód vykonán právě ve vlákně message dispatcheru, musí se použít statická metoda `invokeLater`(`Runnable`) třídy `SwingUtilities`, která počká až budou všechny události obslouženy a GUI bude vykresleno. Přizpůsobení výšky je nutné provést až v této metodě v kódu podprogramu `vlozMMLElementEnableNesting`(), jelikož výška a zarovnání je prováděna na základě umístění komponent na obrazovce. Kdyby byla upravována v zaregistrovaném posluchači, neznali bychom přesné pozice komponent nového objektu. V těle `invokeLater`() se pouze volají statické metody `vypocetNovychVysek`(), `prizpusobeniVyskyZmeneObsahu`(). V těle metody `vlozMMLElementEnableNesting`() nalezneme následující kód:

```
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        MMLElementEnableNesting.vypocetNovychVysek(Editor.aktTextKomp);
        MMLElementEnableNesting.prizpusobeniVyskyZmeneObsahu(
            Editor.aktTextKomp, true);
    }
});
```

Při mazání znaků a objektů tvořených panely je přizpůsobení ponechané na posluchači, protože veškeré komponenty jsou už vykresleny.

```
public void removeUpdate(DocumentEvent e) {
```

```
    ...
    if(!rodic.getAttribute("vnorovani").equals("")) {
        MMLElementEnableNesting objekt =
          (MMLElementEnableNesting) rodic.getUserData("instanceObjektu");
        objekt.importElementu(textKomp);
        objekt.prizpusobeniZmeneObsahu(textKomp);
        objekt.vypocetNovychVysek(textKomp);
        objekt.prizpusobeniVyskyZmeneObsahu(textKomp, true);
    }
}
```

Metoda `vypocetNovychVysek`()

Tato metoda prochází všechny vložené komponenty v textovém poli. Odkaz na pole je jedním z argumentů této metody. Po průchodu je určena nová výška, kterou je zapotřebí nastavit pro textové pole a také pro vrstvený panel. Výška pole je dána rozdílem horní pozice nejvyšší komponenty a dolní pozice nejnižší komponenty. Pozice jsou určovány metodou `getLocationOnScreen`(), která je součástí standardních grafických komponent balíku Swing. Nové rozměry jsou uloženy do statické datové složky typu `ArrayList` třídy `Editor`. Následně se přejde do nadřazeného textového panelu metodou `vnoreni`(). Proces se bude rekurzivně opakovat.

Metoda `zarovnaniPodleVnorElem`()

Cílem této metody je zajistit vzájemné zarovnání matematických objektů v textovém panelu. Je nastaveno metodou panelu `setAlignmentY`(), na němž jsou rozmístěny další komponenty. Snahou je tedy zarovnat veškeré vložené komponenty do jedné horizontální osy podle standardního zápisu matematických výrazů. Například objekt představující zlomek je zarovnaný podle zlomkové čáry, odmocnina je zarovnaná na základě vnořené hodnoty či objektu. Základní zarovnání je stanoveno po vytvoření příslušné nové instance a po jejím vložení do textové komponenty. Tato metoda vrací hodnotu typu `float` odpovídající potřebnému zarovnání objektu na základě jeho obsahu. Většina matematických objektů je tímto způsobem zarovnaná (např.: odmocnina, horní či dolní index, matice, neviditelný obsah...). Hodnota předávaná metodě `setAlignmentY`() je v rozmezí $0-1$, kde 0.5 je zarovnání na střed. K určování veškerých pozic je použita metoda `getLocationOnScreen`(). Tato metoda je volána v těle následujícího podprogramu. Průběh metody lze popsat uvedeným algoritmem (znázorněn na obr. 3.15):

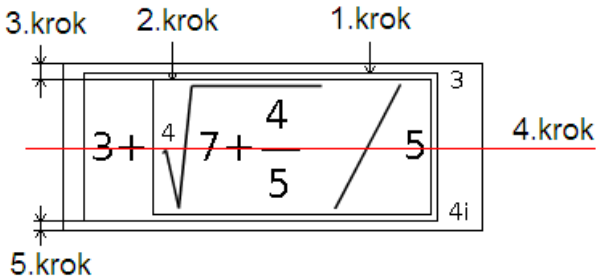
1. krok: Určí se typ objektu, jenž má být zarovnaný, a na základě jakého textového pole má být zarovnán. U většiny objektů se jedná o pole představující základ, u matice či tabulky s lichým počtem řádků se hledá textové pole umístěné nejvýše v rámci prostředního řádku, u zlomku zapsaného do řádku se musí určit, zda jmenovatel nebo čitatel je umístěn výše.

2. krok: Uvnitř vyhledaného pole se nalezne prvek umístěný nejvýše k hornímu okraji, neexistuje-li takový objekt, tzn. obsahem jsou pouze základní elementy (uvedené v tab 2.1), pak přechod na 5. krok.

3. krok: Pro objekt jenž se zarovnává, musíme vypočítat rozdíl horních pozic tohoto objektu a objektu určeného v 2. kroku.

4. krok: Je potřeba stanovit pozici osy pro zarovnání v rámci panelu objektu. Zjistí se zarovnání objektu z 2. kroku a vynásobí se jeho výška panelu s jeho hodnotou zarovnání. K výsledku se přičte rozdíl horních pozic objektů z 3. kroku, čímž se získá pozici osy. Výsledné zarovnání je rovno podílu pozice osy a výšce panelu, z něhož je zarovnávaný objekt tvořen.

5. krok: Určí se rozdíl dolních pozic nalezeného textového pole a panelu tvořícího objekt, pro který se počítá hodnota zarovnání. Od výšky tohoto objektu se odečte vypočítaný rozdíl a výsledná hodnota je vydělená znovu výškou panelu. 4. a 5. krokem jsme získali hodnotu zarovnání pro metodu `setAlignmentY`().



Obrázek 3.15: Průběh metody `zarovnaniPodleVnorElem`()

Metoda `prizpusobeniVyskyZmeneObsahu`()

Metodou `vypocetNovychVysek`() jsme si připravili nové rozměry textových komponent a panelů i u nadřazených objektů, které je zapotřebí nastavit. Průběh metody lze rozdělit na několik částí:

1. Po určení typu objektu je nejprve upravena velikost textového pole na uloženou hodnotu. Následně se zjistí, zda se jedná o objekt, v němž došlo ke změně, budou se totiž procházet a upravovat i nadřazené objekty. Pouze u prvně upravovaného objektu známe přesné pozice jednotlivých komponent díky metodě `invokeLater`(). Celý dále popisovaný proces se bude rekurzivně opakovat a postupně se budou upravovat všechny objekty vynořováním k hlavní textové komponentě. Z tohoto důvodu se bude lokace u nadřazených komponent určovat jiným způsobem než pomocí metody `getLocationOnScreen`(). Jedná-li se tedy o objekt, kde vznikla událost, pak se patřičně upraví pozice textového pole a dalších polí, pokud je jich více. U některých komponent je umístění dané na základě zarovnání vnořených objektů. Pokud je součástí objektu komponenta `Grafika`, pak se provede její překreslení. Po úpravě atributů všech vložených objektů je zapotřebí také upravit rozměry hlavního panelu. Nakonec se panel ještě zarovná. U většiny matematických objektů je zarovnání prováděno na základě jeho obsahu. Správnou hodnotu pro metodu `setAlignmentY`() vrací další vytvořená metoda `zarovnaniPodleVnorElem`(), kterou lze volat pouze v případě, že jsou známy všechny pozice a GUI je vykresleno, tedy v případě objektu, v němž se udála změna.

2. Z aktuálního objektu se přejde na nadřazený a bude se znovu určovat výška textového panelu, jelikož zarovnáním vloženého objektu mohlo dojít k její změně. Výška je opět určovaná na základě rozdílu horní pozice nejvyššího objektu a dolní pozice nejníže umístěného objektu. Při určování těchto pozic nelze použít metodu `getLocationOnScreen`() na objektu, jenž byl v předchozím kroku upraven, jelikož ještě není vykreslen a pozice by nebyly správné. Hodnoty jsou tedy vypočteny pomocí pozice zarovnání sourozeneckého objektu, který se nachází v témže textovém poli. Pokud je zjištěná výška odlišná od uložené ve statické složce, pak je přepsána na nově vypočítanou hodnotu.

3. Došlo-li ke změně rozměrů komponent, pak většina z nich bude upravena už v této fázi. K úpravě zbylé části dojde v dalším volání téže metody v 1. části. Nová výška textového panelu může znamenat posunutí jiného textového panelu tak, aby bylo zachováno požadované rozmístění komponent na panelu. Přeuspořádání polí znamená, že je opět zapotřebí zkontrolovat velikost vrstveného

panelu. V této fázi dojde ještě k zarovnání objektu a to obdobným způsobem jaký byl popisovaný v metodě `zarovnaniPodleVnorElem`() s tím rozdílem, že pracuje-li se s upraveným objektem, nemůžeme pozice určovat standardní metodou `getLocationOnScreen`(). K tomuto účelu vystačí hodnoty zjištěné v předchozí fázi.

4. Poslední část tohoto podprogramu spočívá pouze v přechodu do nadřazeného textového pole metodou `vnoreni`(), pro který se bude opět volat metoda `prizpusobeniVyskyZmeneObsahu`(), aby šlo rekurzivně projít a případně i upravit rodičovské komponenty.

Implementace 1. a 3. části jsou závislé na konkrétním objektu, proto mohou být řešeny jinak pro každou instanci odvozenou od třídy `MMLElementEnableNesting`.

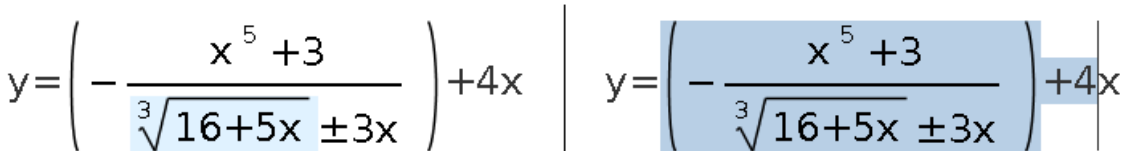
3.3 Rozšiřování funkčnosti editoru

V tomto stádiu vývoje aplikace bylo dosaženo základní funkčnosti WYSIWYG editoru. Editor je vhodné doplnit o další základní funkce, které uživateli nejen zpříjemní a ulehčí práci při vytváření vzorců, ale umožní mu zapsanou matematiku i stylovat a načítat pro pozdější editaci.

3.3.1 Zvýrazňování a označování jednotlivých elementů

Aby uživateli bez jakékoliv znalosti matematického značkovacího jazyka bylo zřejmé, jaké části vytvořeného vzorce odpovídají konkrétním MathML elementům, bylo zavedeno vyznačování těchto prvků ve výrazu. Elementy jsou při procházení vzorce zvýrazněny světle modrou barvou pozadí. V místě kurzoru textového pole tedy bude daný úsek výrazu vyznačen. Při označování části vzorce bylo také zapotřebí zajistit, aby se selektovali i vytvořené matematické objekty tvořené vloženými komponentami. Na obrázku 3.16 vidíme v levé půli zvýraznění elementu `mroot` v textovém poli představujícím jmenovatel zlomku a v půli pravé označení celého obsahu hodnoty y kromě identifikátoru x.

Z událostního modelu v příloze D lze vypozorovat, že každé textové pole může být zdrojem události `CaretEvent`. Tuto událost (změnu pozice kurzoru) zpracovává posluchač `MMLCaretListener` odvozený od třídy `CaretListener`, který implementuje



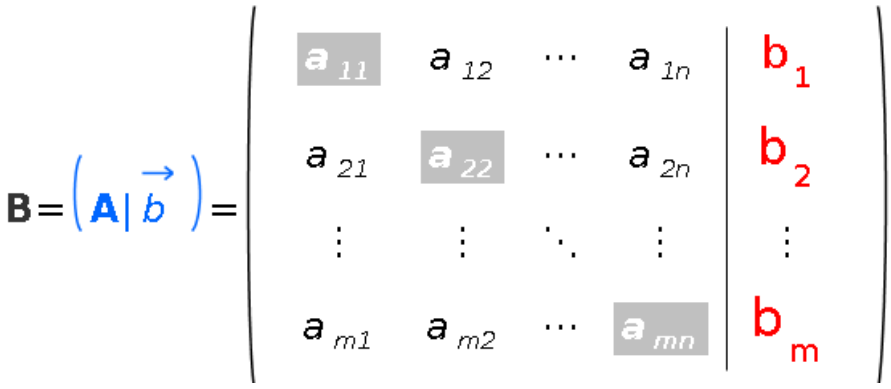
Obrázek 3.16: Zvýraznění a označení MathML elementů ve vzorci

jedinou metodu `caretUpdate`(). Při každé změně kurzoru se zároveň aktualizuje aktuální textová komponenta, v níž se nacházíme a s ní spojený MathML dokument. Kromě předchozí aktualizace se v těle metody také testuje, zda je část dokumentu označena či nikoliv. Součástí objektu nesoucího informace o události typu `CaretEvent` jsou metody `getMark`() a `getDot`(). Vrací nám pozici začátku a konce selekce či jenom pozici kurzoru. Při vytváření instance `MMLCaretListener` je předáván konstruktoru element z dokumentu objektu spojený s textovou komponentou. Při označení části vzorce se tedy prochází všechny děti předaného elementu. Pomocí atributů `pocPozice` a `konPozice` se určuje, zda se objekt spojený s elementem se nachází v označené části a zda je tvořen panelem. Každá instance třídy odvozené od `MMLElementEnableNesting` má definovanou metodu pro označení objektu. Jejím jediným argumentem je barva označení. V případě, že objekt chceme označit jí předáme barvu získanou metodou `getSelectionColor`() textového panelu. Pokud chceme objekt pouze zvýraznit, pak argumentem bude světle modrá barva. Objekty lze do sebe vnořovat, je tedy nutné zajistit označení resp. zvýraznění i vnořených instancí. K tomu slouží metoda `projdiDOM`(), která rekurzivně prochází dokument do hloubky. Pro zvýraznění části řetězce v textovém poli je použit objekt typu `Highlighter`, který je pro tento účel konstruovaný. Rušení označení a zvýraznění se provádí analogickým způsobem.

3.3.2 Stylování vzorců

Jedním ze základních cílů této práce je, aby WYSIWYG editor podporoval jednoduché stylování vzorců. Aplikace bude nabízet změnu barvy písma, pozadí, změnu velikosti fontu a matematických objektů a nakonec nastavení řezu písma. Více možností neposkytují ani komerční produkty. Příklad vzorce, na němž je použito stylování je znázorněn na obr. 3.17.

Veškerá změna stylu je nastavena buď pomocí menu a nebo záložek. Metody (jako



Obrázek 3.17: Příklad použití jednoduchého stylování

jsou např.: `nastavVelikost`(), `nastavTucny`(), `nastavBarvuFontu`() atd.) a třídy implementující posluchače (`HandlerListSelection`, `HandlerColorChooser`) připravují argumenty pro hlavní metodu obsluhující změnu stylu a to podprogram `nastavStyl`().

Metodou `addMathMLStyle`() jsou rozšířena všechna textová pole o vytvořený styl `regular`, `italic`, `bold` a `boldItalic`. Od každé vytvořené instance třídy `JTextPane` je získán metodou `getStyledDocument`() objekt typu `StyledDocument`. Výše vypsané styly v sobě bude zapoudřovat právě tento objekt.

Metoda `nastavStyl`() vyhledá element odpovídající části vzorce, na níž je umístěn kurzor. V případě označení vyhledá všechny prvky spojené s označenou částí. Styl se tak nastavuje vždy pro celý obsah elementu. Elementy se klasifikují na základní (viz tab. 2.1) a ostatní prvky, jejichž obsahem může být jiný prvek. U první skupiny se styl nastavuje rozšířením prvku o tyto atributy:

- `mathcolor` (#RGB hexadecimálně),
- `mathbackground` (#RGB hexadecimálně),
- `mathsize` (velikost v procentech),
- `mathvariant` (hodnoty `bold`. `italic`, `bold-italic`)

Elementy z druhé skupiny jsou vnořeny do elementu `mstyle`, který je doplněn o uvedené atributy. Styl je tak nastaven i pro všechny vnořené prvky. Je-li u elementu zjištěno, že daný atribut už obsahuje, pak je tento atribut odstraněn. Provedené změny v dokumentu objektu jsou importovány jak do výsledného dokumentu, tak i do

dokumentů všech nadřazených objektů metodou `importPredci`(). MathML dokumenty jsou tedy upravené, nyní je zapotřebí změnu promítnout i do textového pole. Jaký styl je pro danou část vzorce již nastavený je určeno náhledem do dokumentu objektu. Instance třídy `StyledDocument` narozdíl od rozhraní `Document` podporuje formátovaný text. Její metodou `getStyle`() je vrácen objekt typu `Style`. Získáme tedy vytvořený styl v metodě `addMathMLStyle`(). K úpravě zobrazení částí vzorce v textovém poli se běžně používá kolekce klíčů atributů `StyleConstants`. Tato třída poskytuje nástroje jako například `StyleConstants.setForeground`() pro nastavení barvy písma, `StyleConstants.setFontSize` pro úpravu velikosti fontu a další. Těmito statickými metodami se rozšíří připravený styl. Třída `StyledDocument` disponuje metodou `setCharacterAttributes`(), které se přiřadí námi vytvořený styl části řetězce v textovém poli. Mění-li se velikost písma a objektů musí se přizpůsobit i veškeré komponenty tvořící část vzorce a to metodami popsanými v kapitole 3.2.4. V podprogramu `prizpusobeniZmeneObsahu`() se navíc šířka řetězce určuje tak, že text je rozdělen na menší části tzv. elementy (`javax.swing.text.Element`), ke kterým je přiřazená množina atributů (`AttributeSet`), obsahující informace nutné pro správné zobrazení elementu na obrazovku. Metodami `isBold`(), `isItalic`() a dalšími se pak už jen zjistí nastavené formátování a podle toho stanoví výsledná šířka komponent. Část kódu řešící tento problém:

```
StyledDocument sDoc = f.getStyledDocument();
AttributeSet atrSet;
Font fnt;
javax.swing.text.Element run;
int delka = rozmery.width;

while (i < str.length()) {
    run = sDoc.getCharacterElement(i);
    atrSet = run.getAttributes();
    fnt = sDoc.getFont(atrSet);
    ...
    if (fnt.isBold() || fnt.isItalic() ||
          fnt.getSize() != Editor.aktTextKomp.getFont().getSize())
        delka += f.getFontMetrics(fnt).stringWidth(ret);
    else
        delka += f.getFontMetrics(f.getFont()).stringWidth(ret);
    ...}
```

Formátuje-li se objekt tvořený komponentou `Grafika`, pak případná změna barvy

fontu předává metodou `setColor`(), kterou se mění barva pera a překresluje komponenta. Pokud upravovaný objekt obsahuje další do sebe vnořované instance, pak je volána metoda `nastavStylVnorenychElementu`(). Jde o rekurzivní metodu procházející veškeré vnořené objekty a nastavující požadovaný styl. Pokud nějaká část vzorce už formátována je, pak ji nový styl překryje.

Volby pro formátování matematického výrazu v záložkách či menu jsou přednastavovány podle stylu části vzorce, na němž je umístěn kurzor. Stav komponent je měněn v metodě `caretUpdate`() zaregistrovaného posluchače typu `MMLCaretListener`.

3.3.3 Implementace ostatních funkcí

Pro zpříjemnění práce s editorem jsou implementované funkce *zpět* (undo) a *znovu* (redo). Tyto funkce jsou v menu `Edit` nebo jdou vyvolat přes pravé tlačítko na hlavním vstupním poli, v tzv. popup menu. Samozřejmě v žádném editoru nesmí chybět funkce (*Nový*) pro tvorbu nového vzorce. Pro urychlení vytváření vzorců, je možné procházet vzorec pomocí kurzorových kláves. Pro přechod do textové komponenty objektu se tak nemusí používat myš.

Funkce Zpět/Znovu

Pro všechny instance třídy `JTextPane` je zaregistrovaný další realizovaný posluchač `MMLUndoableEditListener`, který tentokrát zaznamenává veškeré změny v těchto polích. Třída implementuje rozhraní `UndoableEditListener`. Výsledný MathML dokument či dokument matematických objektů je před jakoukoliv změnou v textovém poli ukládán do statického zásobníku `stackUndo` třídy `Editor`. Pro vrácení do předchozího stavu v textovém poli existuje standardní třída `UndoManager`. Funkce *zpět* a *znovu* jsou definovány v jeho metodách `undo`() a `redo`(). Konstruktoru třídy `MMLUndoableEditListener` je zároveň předáván objekt tohoto typu. Ke každému poli je tedy připojen jeho vlastní posluchač a manažer změn. Vytvořený vzorec může být složen z více textových polí a proto při každé změně je v metodě `undoableEditHappened`() vytvořeného posluchače uložena instance tohoto manažeru do statického zásobníku `stackUndoManagers` třídy `Editor`. Máme tedy uchován jak stav MathML dokumentu tak i stav v textovém poli. Délka historie je nastavena na 10 stavů.

Funkce zpět a znovu jsou obslouženy v třídách `UndoAction` a `RedoAction` odvozených od třídy `AbstractAction`. V metodě `actionPerformed`() třídy `UndoAction` je dokument a obsah textového pole vrácen do předchozí podoby (vyzvednutím manažeru a dokumentu ze zásobníku) a zároveň je uložen stejným způsobem současný stav do dalších zásobníků pro funkci znovu.

Je-li potřeba vrátit krok, v němž se upravoval styl matematického objektu tvořený grafickými komponentami, není použit manažer změn a to z několika důvodů:

1. Změnou barvy fontu se mění i barva vykreslené grafiky, kterou nám objekt typu `UndoManager` není schopen vrátit.

2. Ke změně stylu dojde najednou ve všech komponentách (i vnořených). Manažer by vracel předchozí stav postupně.

Tento problém je řešen jiným způsobem využivajícím metodu `nastavStyl`(). Při formátování těchto objektů je na vrchol zásobníku `stackUndo` ukládán objekt typu `Vector`. V této proměnné jsou uloženy veškeré argumenty potřebné pro navrácení stylu metodou `nastavStyl`(). Vracíme-li krok, v němž byl formátován vzorec, pak v metodě `actionPerformed`() třídy `UndoAction` je na zásobník `stackRedo` uložen taktéž seznam argumentů pro funkci znovu.

Funkce Nový

Implementace této funkce není nijak složitá, spočívá v těchto bodech:

1. Z výsledného MathML dokumentu se odstraní všechny děti kořenového elementu `math`.

2. Vyprázdní se hlavní textový panel a vynuluje se čítač matematických komponent (`citacMatKomp`).

3. Vyprázdní se zasobníky pro funkce *Zpět*/*Znovu*. Z objektu typu `UndoManager` se vyřadí veškeré zaznamenané editace. Obnoví se stav nabídek *Zpět* a *Znovu* v menu.

Průchod vzorců pomocí kurzorových kláves

Všechny matematické objekty kromě základních, jenž jsou tvořeny textovými panely, lze procházet kurzorovými šipkami. Veškeré existující instance třídy `JTextPane`

mají přidaný posluchač obsluhující stisk kláves. Každá třída definující objekt tvořený textovými poli má svoji vlastní implementaci reakce na stisk kurzorových šipek. Například zlomek se prochází tak, že šipkou nahoru se kurzor přemístí do čitatele a naopak šipkou dolu do jmenovatele. Dojde-li se kurzorem na levý resp. pravý kraj jednoho z polí a stiskne se levá resp. pravá kurzorová šipka, pak se z objektu vystoupí a přejde se kurzorem před resp. za objekt. Za zmínku stojí ještě trochu neobvyklý přechod mezi poli u objektu představující MathML element `mmultiscript`. Nachází-li se kurzor v základu objektu, pak se do horního pravého exponentu přejde šipkou nahoru, je-li na pravé polovině tohoto pole. Do zbývajících indexů a exponentů přejdeme analogickým způsobem. U instancí tříd `ScriptsAndLimitsWithArrowAndOther`, `ScriptsAndLimitsWithOperator` je vstup do needitovatelných polí zakázán a přejde se tedy do následující komponenty. Řízení je textové komponentě předáno na základě metody `requestFocus`().

3.3.4 Řešení načítání MathML dokumentů

Každý WYSIWYG editor by měl také podporovat načítání uloženého dokumentu. Při realizaci vstupu MathML dokumentů jsou limitující bezpečnostní omezení, která plynou z implicitního nastavení java apletu (viz kapitola 3.1.6). Načtení matematického zápisu je tedy řešené obdobným způsobem, jakým je realizovaný výstup aplikace. Po aktivaci nabídky Soubor → Načti, se musí zkopírovat celý obsah svého XML souboru do označeného textového pole, v němž se zobrazuje výsledný kód. Po vložení MathML zápisu je zapotřebí předchozí nabídku opět aktivovat, čímž se spustí proces načítání uživatelova dokumentu.

Načtený řetězec je zapotřebí převést do strukturované formy (na objekt typu `Document`). Za tímto účelem je vytvořená metoda `string2DOM`(). Která mimo jiné kontroluje, zda dokument neporušuje syntaktická pravidla a je validní. Testování má za úkol třída `MMLErrorHandler`, která implementuje rozhraní `ErrorHandler` umožňující pracovat se vzniklými chybami. V této třídě jsou obslouženy tři typy chyb:

1. `Warning` představuje méně závažné chyby než další uvedené.

2. `Error` je chyba XML 1.0, obvykle se jedná o porušení podmínek validity.

3. `FatalError` je fatální chyba způsobená nesprávným strukturováním XML dokumentu, nerozpoznanou znakovou sadou nebo použitím neplatného odkazu na entitu či znak.

Pokud se nějaká z uvedených chyb vyskytne, pak je na ni uživatel upozorněn. Po zkontrolování předaného dokumentu je editor připraven na tvorbu nového vzorce. Základem procesu načítání MathML je rekurzivní metoda `createDocument`() procházející právě získaný dokument do hloubky. Rozpoznává jednotlivé elementy a zároveň vyskytují-li se v dokumentu prvky umožňující vnořování, pak je měněno aktuální textové pole a aktuální MathML dokument. Elementy jsou zde vkládány metodou `insertElement`(). V tomto podprogramu se nejprve identifikují prvky a následně jsou vytvořeny v aktuálním dokumentu způsobem, jenž je popsán v kapitolách 3.2.2 a 3.2.3. Základní elementy (`mn`, `mi` a `mo` neoznačené atributem `vnorovani`), jsou realizovány tak, že jejich hodnota je přidána do aktuálního textového pole a regulární výraz uvedený v kapitole 3.1.5 je automaticky vytvoří a přidá do dokumentu. U některých elementů (viz tab. 2.3) je nejprve zapotřebí nahlédnout do jeho obsahu, kvůli zjištění, jaký typ objektu pro ně použít (`ScriptsAndLimits`, `ScriptsAndLimitsWithArrowAndOther`, `ScriptsAndLimitsWithOperator`). Pro případné nastavení stylu základních prvků je použita metoda `settingStyle`(), která pouze určuje, jak bude část vzorce spojená s elementem formátována, a připravuje argumenty, jenž jsou předávány již popisovanému podprogramu `nastavStyl`(). Jsou-li v uživatelově dokumentu přítomny elementy, jejichž obsah může být tvořen i jinými prvky, pak jsou zpracovány nejprve všechny jeho děti a až ve zpětném kroku při procházení stromovou strukturou se kontroluje zda jeho rodič je prvek `mstyle`. Objekt představující daný element, je formátován metodou `nastavStyl`() až před procházením jeho pravých sourozenců, jinak by nebyly stylovány všechny jeho vnořené komponenty.

3.3.5 Doplnění o jazykové mutace

WYSIWYG editor bude zpřístupněný na internetu, proto by se mělo počítat s tím, že k němu budou přistupovat uživatelé různých národností. Aplikace tak bude lokalizovaná jak pro češtinu tak i pro angličtinu. Java nabízí pro tento účel třídu `ResourceBundle`, která vyhledává texty nabídek, popisy tlačítek atd. ve správném

jazyce. Aby veškeré tyto řetězce byly odděleny od zdrojového kódu, jsou ukládány v konfiguračních souborech (`Texty_cs_CZ.properties`, `Texty_en_US.properties`). Editor tak lze snadno doplnit o další jazykové mutace. Tato třída používá k určování národního prostředí třídu `java.util.Locale`. Vyhledání souboru odpovídající určenému národnímu prostředí má za úkol metoda `getBundle`() třídy `ResourceBundle`. Předává se jí jako argument řetězec a nepovinný objekt typu `Locale`. V případě, že nechceme automaticky stanovit lokální nastavení. První argument hraje roli při vyhledávání konfiguračních souborů. Název souboru musí být tvořen tímto řetězcem, zkratkou jazyka a státu (buď oba údaje nebo jenom jazyk). Soubor musí mít příponu `.properties`. Soubory by tedy byly vyhledávány například v uvedeném pořadí:

1. `Texty_cs_CZ.properties`
2. `Texty_cs.properties`
3. `Texty.properties`

Na následující ukázce formátu konfiguračního souboru s českou lokalizací si lze všimnout, že národní znaky jsou převáděny do univerzální znakové sady Unicode a to z důvodu toho, že program může být přenesen na počítač, který naší znakovou sadu nemusí podporovat. V Javě lze tohoto převodu dosáhnout programem `native2ascii`, který je součástí JDK. Soubor je složen z klíčů a řetězců v lokalizované verzi. Ve vývojovém prostředí Netbeans existují nástroje pro internacionalizaci aplikace, které obstarají převod národních znaků a generování souboru s tímto formátem automaticky. Zadávají se v nich pouze klíče a hodnoty.

```
# Sample ResourceBundle properties file
# ampersand znaci klavesovou zkratku (Mnemonics)
Soubor=&Soubor

Nacti=Na\u010Dti

Uloz=Ulo\u017E
...
```

Jelikož je editor navržený tak, aby uživatel mohl menu a záložky s matematickými objekty procházet pomocí klávesových zkratek, je zapotřebí řešit problém zkratek pro různé jazykové mutace. Z tohoto důvodu je součástí realizovaného editoru třída

`MMLResourceBundle`, která nabízí mimo metody pro přístup k lokalizovaným řetězcům (`getString(String key)`) i metodu vracející znak pro klávesovou zkratku (`getMnemonicZnak(String key)`). Tento znak je v konfiguračním souboru určován pomocnou značkou & (ampersand), jenž je před něj vložena.

3.4 Vytvoření distribuce

Distribuce realizované aplikace `MathMLWysiwygEditor` je tvořena dvěmi složkama `MathMLWysiwygEditor-0.0.1-dist`, `MathMLWysiwygEditor-0.0.1-src`. První složka obsahuje archiv `MathMLWysiwygEditor.jar`. Java archiv pro svoje spuštění na hostitelském počítači vyžaduje nainstalování javovského virtuálního stroje (JVM). V druhé uvedené složce se nachází veškeré zdrojové a konfigurační soubory, sestavovací schéma (`build.xml`) a ikonové soubory. Je zde zavedena tato adresářová struktura:

```
MathMLWysiwygEditor-0.0.1-src/MathMLWysiwygEditor/
    src         <adr>
    build.xml
./src/
    net         <adr>
    Texty_cs_CZ.properties
    Texty_en_US.properties
./net/sourceforge/MathMLWysiwygEditor/
    images      <adr>
    lib         <adr>
    Editor.java     Fenced.java     Frac.java     Grafika.java
    ...             ...             ...           ...
```

Zkompilované soubory a spustitelný `jar` archiv se získá nástrojem Apache ANT [5] a pro něj vytvořeným sestavovacím schématem (viz příloha F). Tato aplikace poskytuje dobré možnosti řízení překladu programů napsaných v jazyce Java. V současnosti je nejrozšířenějším nástrojem a je využíván i u vývojových prostředí, jako je například NetBeans, Eclipse, IntelliJ IDEA a další. Jeho hlavní výhodou je přenositelnost mezi různými operačními systémy, která byla dosažena implementací celé aplikace právě v jazyce Java. Další výhodou je syntaxe sestavovacího souboru ve formátu XML. Po nainstalování ANTu je překlad souborů proveden následujícím zápisem do příkazové řádky:

```
ant -buildfile build.xml
```

Soubor je zapotřebí uvést i s jeho úplnou cestou. Pro úspěšný překlad je důležité neměnit adresářovou strukturu uvnitř složky `MathMLWysiwygEditor-0.0.1-src`. Proběhlo-li vše v pořádku, byla v aktuálním adresáři vytvořena nová složka `build` se zkompilovanými soubory (*.`class`) a složka `dist` s jar archivem.

3.5 Komunikace (X)HTML dokumentu s apletem

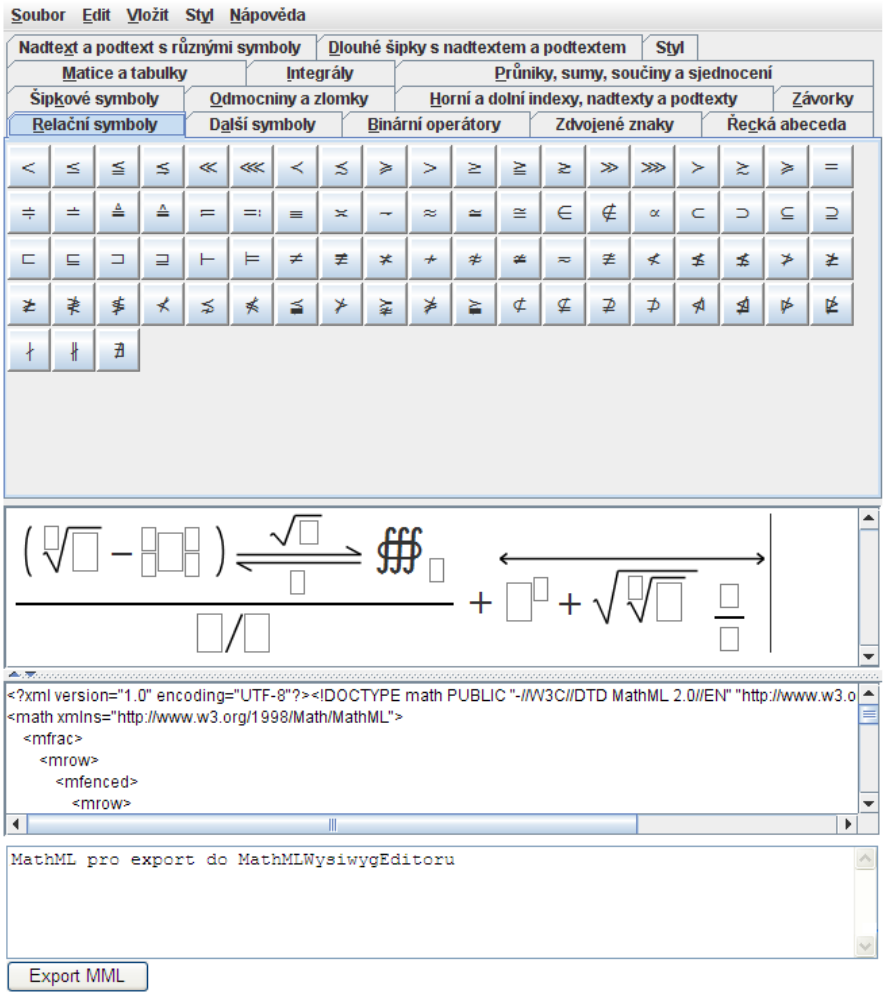
Jedním z hlavních požadavků této práce bylo, aby realizovaný program bylo možno spustit jako aplet i aplikaci. V případě spuštění editoru internetovým prohlížečem by bylo pro názornost přínosné, kdyby webová aplikace umožňovala vytvořený matematický zápis předat XHTML dokumentu a zajistit tak jeho okamžitou interpretaci ve spuštěném prohlížeči. Stejně tak by bylo užitečné zajistit u apletu vstup dat z XHTML dokumentu, například pro předávání MathML pomocí JavaScriptu pro jeho načtení. WYSIWYG editor by tak měl podporovat obousměrnou komunikaci.

Před samotným popisem realizace komunikace je nutné se zmínit o způsobu zakomponování apletu do (X)HTML dokumentu. Ukázkový dokument (výpis v příloze G) je ve formátu XHTML, aby do něj šlo vkládat MathML z apletu. Do HTML se aplet přidává pomocí tagu `applet`, kdežto u XHTML je tento tag zastaralý a nepodporovaný. Je nahrazen elementem `object`. Pro spuštění apletu ve všech typech prohlížečů je zapotřebí uvést i vnořený prvek `object` (určen pouze pro MSIE, ostatní prohlížeče si vystačí s jeho nadřazeným prvkem). Pod prvkem přidávající aplet do XHTML je zakomentovaný element `aplet`, který bychom použili v případě HTML dokumentu. Z kódu je patrná tato adresářová struktura:

```
./
    classes     <adr>
    Editor.xhtml
./classes/MathMLWysiwygEditor/
    lib         <adr>
    MathMLWysiwygEditor.jar
./lib/
    netscape.jar
```

Na obrázku 3.18 je již vidět interpretace ukázkového dokumentu prohlížečem FireFox. Jeho součástí je mimo apletu i formulář pro vstup dat do aplikace. V editoru je vytvořen ilustrativní vzorec bez jediné hodnoty. Teprve při vkládání hodnot se

orámování vstupních textových polí ruší a opět při smazání veškerých hodnot je pole vyznačené zobrazením okraje.



Obrázek 3.18: XHTML dokument s apletem pro testování exportu MathML

Třída Editor kromě povinných veřejných metod (`init`(), `start`()...) nabízí i popisovanou metodu `nactiDokument`() (viz kap 3.3.4) pro načítání uživatelova MathML. Metoda je definovaná jako veřejná (`public`) a tudíž je přístupná i externím programům, v tomto případě JavaScriptu XHTML dokumentu. Funkce přidaná do hlavičky ukázkového XHTML pouze volá veřejnou metodu apletu, jejímž argumentem je řetězec obsahující MathML získaný z formuláře. Při načítání matematického zápisu pomocí JavaScriptu, ale docházelo k nekorektnímu sestavení uživatelova vzorce. Problém spočíval v tom, že webové prohlížeče nepodporují standardní metodu `setCaretPosition`() třídy `TextComponent`. Pozice kurzoru, kterou vrací tato metoda, je neustále nulová. Pokud se metoda při vytváření matematického výrazu

použije, dojde k vyvolání vyjímky a tato část kódu je přeskočena. Bylo tak zapotřebí vytvořený editor přizpůsobit vzniklému problému a to včetně vytvoření nové vlastní metody `insertComp`() pro vkládání komponent do textového pole. Standardní metoda `insertComponent`() totiž pracuje s pozicí kurzoru.

Pro opačný směr komunikace měla být použita knihovna `netscape.jar`, která v Javě zpřístupňuje objekty XHTML dokumentu včetně JavaScriptových funkcí pomocí třídy `JSObject` (více informací na [15]). Pokud by aplikace byla spuštěna v prohlížeči, rozšířila by se nabídka Soubor o novou položku umožňující export vytvořeného vzorce do XHTML dokumentu. Tato komunikace byla navržena tak, že součástí elementu `object` je prvek `param` obsahující název funkce v JavaScriptu a která by byla volána z apletu. Předali bychom jí pouze řetězec obsahující MathML. Komunikace byla funkční pouze v případě, že aplikace byla spuštěna v prohlížeči MSIE. Třída `netscape.javascript.JSObject` byla totiž navržena pro komunikaci s objektem typu `java.applet.Applet` a nikoliv pro novější verzi `javax.swing.JApplet`. Jiné univerzálnější řešení pro vytvořený aplet, které by funguvalo ve většině nejpoužívanějších prohlížečů, jsem z důvodu nedostatku času nedohledal. V implementaci tohoto spojení jsem dále nepokračoval.

Závěr

Cíl stanovený na začátku práce byl splněn a výsledkem je plně funkční WYSIWYG editor pro sazbu MathML. Vytvořená aplikace `MathMLWysiwygEditor` spolu s tímto dokumentem nabízí naprosto všem potenciálním uživatelům cestu, jak jednoduše publikovat matematické vzorce na svých internetových stránkách. Editor umožňuje sestavené výrazy načíst i uchovat pro pozdější použití. Stylovat vzorce lze obdobným způsobem jako v komerčních programech stručně popsaných v úvodní kapitole. Kromě běžných funkcí (např. vracet či opakovat jednotlivé kroky) jsou navíc MathML elementy v pracovním panelu zvýrazňovány. Veškerý výstup editoru je validní. Chceme-li jej i přesto ověřit, můžeme použít W3C MathML validátor [20]. Program lze spouštět webovými prohlížeči a může tak být součástí každé internetové prezentace.

Tento dokument popisuje navržené řešení WYSIWYG editoru podporující MathML. Výklad jednotlivých fází vývoje nemohl být detailnější z důvodu rozsahu programu, který čítá okolo 15 000 řádků. Výsledná aplikace je tvořená 19 třídami s mnoha definovanými metodami. V dokumentu jsem se tak snažil věnovat více prostoru pouze důležitým částem návrhu a realizace editoru.

Pokud bychom měli srovnat editor s existujícími programy, pak by byl určitě hlavním nedostatkem nižší výkon aplikace. V dalším rozvoji editoru by tak bylo vhodné se zaměřit především na optimalizaci kódu. Realizovaný editor za komerčními konkurenty dále zaostává v nabídce výstupních formátů. Zde spatřuji další cestu, kudy by se mohl budoucí vývoj ubírat. Jedná se především o podporu obrázkových formátů a formátu PDF. Dalším námětem na vylepšení editoru je zahrnutí elementů sémantického MathML. Uživatel by tak měl na výběr, jakou formu matematického zápisu chce generovat. Uvedené nedostatky však nesnižují funkčnost editoru, která je srovnatelná s dostupnými řešeními.

Literatura

[1] FLANAGAN, David. Java in a Nutschell. 5. vydání, Cambridge: O'Reilly Media, 2005. ISBN 0-596-00773-6.

[2] HORTON, Ivor. Java 5. [s.l.] : [s.n.], 2005. 1443 s. ISBN 80-86330-12-5.

[3] VIRIUS, Miroslav. Java pro zelenáče. Praha : Neocortex, 2001. 240 s.

[4] SATRAPA, Pavel. Regulární výrazy. Root [online]. 2000 [cit. 2008-01-02].

[5] Apache Software Foundation. Ant [online]. [cit. 2008-10-9].
URL: <http://ant.apache.org/>

[6] Linuxsoft. Java – omezování práv I. a II. [online]. [cit. 2007-11-13].
URL: <http://www.linuxsoft.cz/article.php?id_article=1051>

[7] Design Science. MathType [online]. [cit. 2007-12-15].
URL: <http://www.dessci.com/en/products/mathtype/default.htm>

[8] Design Science. MathPlayer [online]. [cit. 2007-10-14].
URL: <http://www.dessci.com/en/products/mathplayer/default.htm>

[9] SourceForge, Inc. DejaVu fonts [online]. [cit. 2008-09-20].
URL: <http://dejavu.sourceforge.net/wiki/index.php/Download>

[10] Integre. Techexplorer Hypermedia Browser [online]. [cit. 2007-10-14].
URL: <http://www.integretechpub.com/techexplorer/>

[11] Max Berger. The JEuclid project [online]. [cit. 2007-11-21].
URL: <http://jeuclid.sourceforge.net/>

[12] NetBeans [online]. [cit. 2007-11-28]. URL: <http://www.netbeans.org/>

[13] soft4science. MathML .NET Control [online]. [cit. 2007-12-15].
URL: <http://www.soft4science.com/products/MathMLControl/>

[14] Sun Microsystems, Inc. [online]. [cit. 2007-10-07]. URL: <http://java.sun.com>

[15] Sun Microsystems, Inc. How Java to Javascript Communication Works in Java Plug-in [online]. [cit. 2008-10-21].
URL: <http://java.sun.com/products/plugin/1.3/docs/jsobject.html>

[16] Sun Microsystems, Inc. Java SE Downloads [online]. [cit. 2007-11-28].
URL: <http://java.sun.com/javase/downloads/index.jsp>

[17] Sun Microsystems, Inc. Java Tutorials [online]. [cit. 2007-12-20].
URL: <http://java.sun.com/docs/books/tutorial/>

[18] W3 Consortium MathML [online]. [cit. 2007-10-02].
URL: <http://www.w3.org/Math>

[19] W3 Consortium MathML. Test Suite [online]. [cit. 2007-10-02].
URL: <http://www.w3.org/Math/testsuite/mml2-testsuite/index.html>

[20] W3 Consortium MathML. W3C MathML Validator [online]. [cit. 2007-12-19].
URL: <http://www.w3.org/Math/validator/>

[21] W3 Consortium MathML. XSL [online]. [cit. 2008-08-19].
URL: <http://www.w3.org/Math/XSL/Overview.html>

[22] W3 Consortium MathML. Software – Editors [online]. [cit. 2007-12-15].
URL: <http://www.w3.org/Math/Software/mathml_software_cat_editors.html>

Příloha A – Ukázka zápisu kvadratické rovnice pomocí MathML

Zápis celkem jednoduchého vzorce v MathML může představovat pro běžného uživatele značný problém. Vedle kódu je zároveň zobrazena interpretace MathML jazyka vybranými internetovými prohlížeči.

```
<math>
 <mrow>
  <mi>x</mi>
  <mo>=</mo>
  <mfrac>
    <mrow>
      <mrow>
        <mo>-</mo>
        <mi>b</mi>
      </mrow>
      <mo>&PlusMinus;</mo>
      <msqrt>
        <mrow>
          <msup>
            <mi>b</mi>
            <mn>2</mn>
          </msup>
          <mo>-</mo>
          <mrow>
            <mn>4</mn>
            <mo>&InvisibleTimes;</mo>
            <mi>a</mi>
            <mo>&InvisibleTimes;</mo>
            <mi>c</mi>
          </mrow>
        </mrow>
      </msqrt>
    </mrow>
    <mrow>
      <mn>2</mn>
      <mo>&InvisibleTimes;</mo>
      <mi>a</mi>
    </mrow>
  </mfrac>
 </mrow>
</math>
```

Mozilla

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

Opera

$$x = \frac{- b \pm \sqrt{b^2 - 4 \quad a \quad c}}{2 \quad a}$$

MSIE

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Obrázek: Zápis kvadratické rovnice jazykem MathML

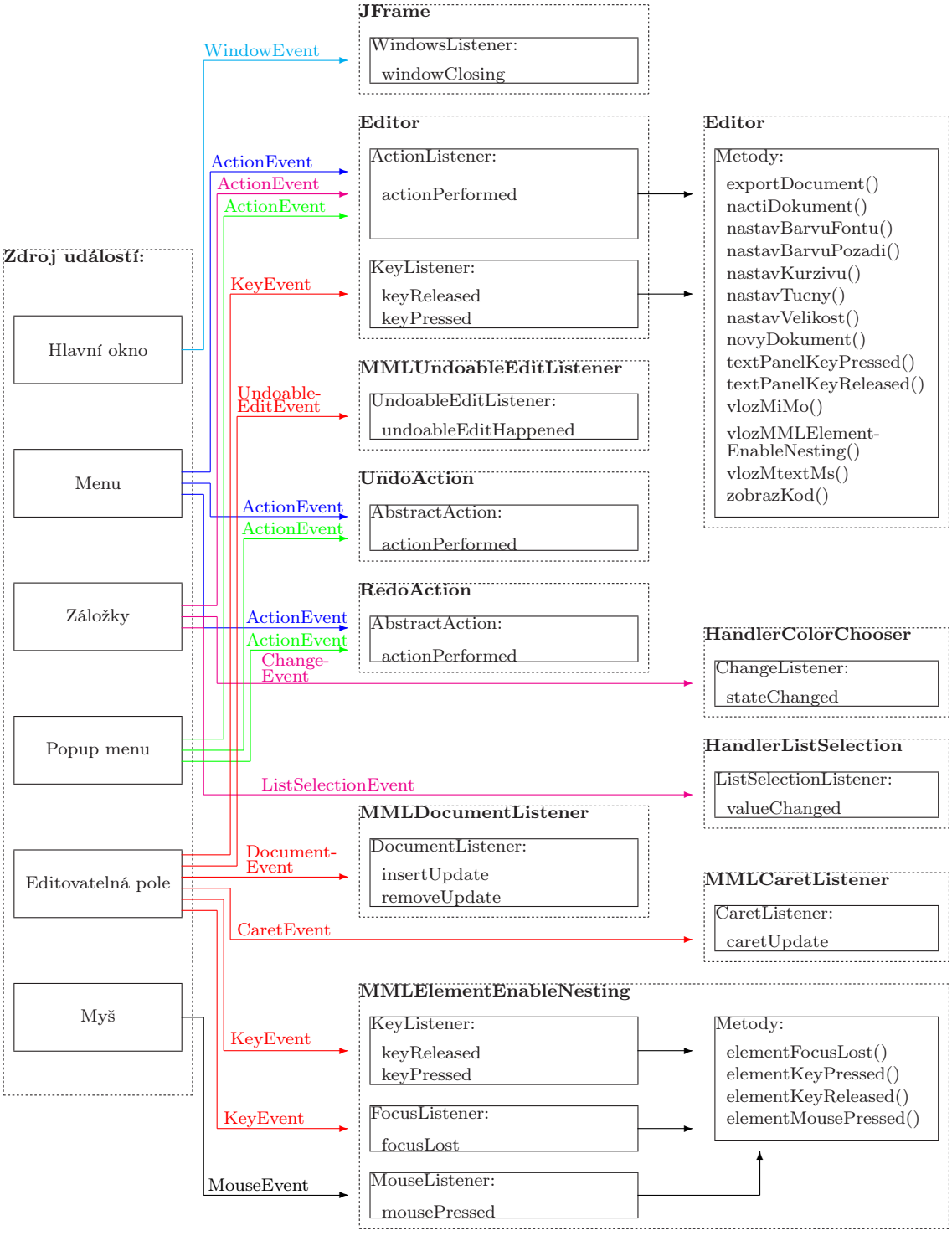
Příloha B – XHTML dokument obsahující MathML

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
  href="http://www.w3.org/Math/XSL/pmathml.xsl"?>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html" />
    <title>Presentation Example</title>
  </head>

  <body>
    <h2>Can your browser display Presentation MathML?</h2>
    <p>
      some text, some text,
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <mfrac>
          <mrow>
            <mn>1</mn>
            <mo>+</mo>
            <msqrt>
              <mn>5</mn>
            </msqrt>
          </mrow>
          <mn>2</mn>
        </mfrac>
        <mo>+</mo>
        <mroot>
          <mrow>
            <mi>a</mi>
            <mo>+</mo>
            <mn>5</mn>
          </mrow>
          <mn>5</mn>
         </mroot>
      </math>
      some text.
    </p>
  </body>
</html>
```

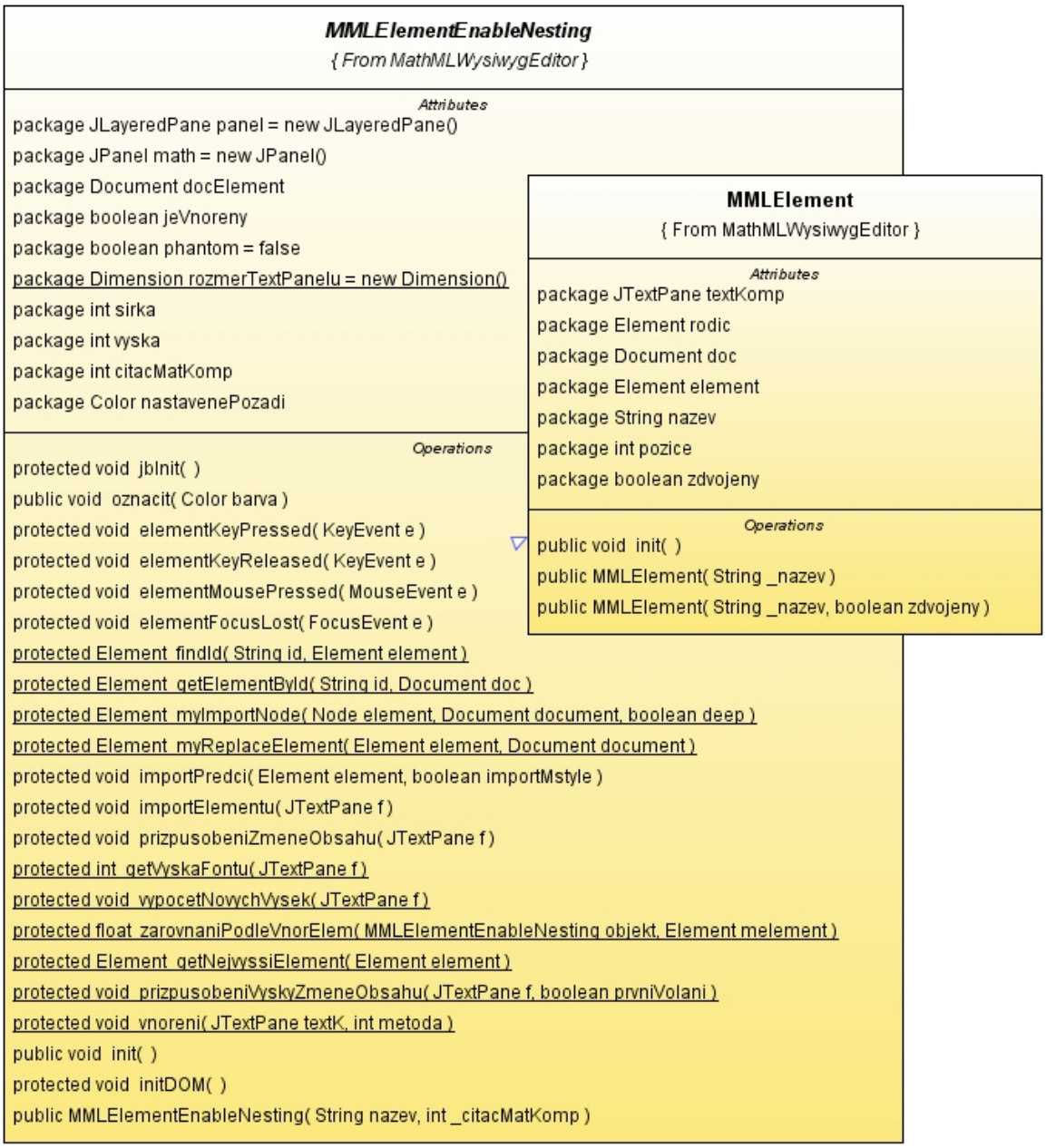
Příloha C – MathML dokument

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
  "http://www.w3.org/Math/DTD/mathml2/mathml2.dtd">

<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mfrac>
    <mrow>
      <mn>1</mn>
      <mo>+</mo>
      <msqrt>
        <mn>5</mn>
      </msqrt>
    </mrow>
    <mn>2</mn>
  </mfrac>
  <mo>+</mo>
  <mroot>
    <mrow>
      <mi>a</mi>
      <mo>+</mo>
      <mn>5</mn>
    </mrow>
    <mn>5</mn>
  </mroot>
</math>
```

Příloha D – Schéma událostního modelu WYSIWYG editoru



Příloha E – Diagram základních tříd implementující MathML elementy



Obrázek: Úplný class diagram pro základní třídy implementující MathML prvky

Příloha F – Ukázka souboru `build.xml`

(sestavovací schéma pro MathMLWysiwygEditor)

```
<?xml version="1.0" encoding="UTF-8" ?>
<project name="MathMLWysiwygEditor" default="jar" basedir=".">
    <target name="initial" description="Initial code">
        <property name="src.dir" value="src"/>
        <property name="build.dir" value="build"/>
        <property name="base.dir" value="net/sourceforge/MathMLWysiwygEditor"/>
        <property name="build.classes.dir" value="${build.dir}/classes"/>
        <property name="build.classes.excludes" value="**/*.java,**/*.form"/>
        <property name="dist.dir" value="dist"/>
        <property name="dist.lib.dir" value="${dist.dir}/lib"/>
        <property name="dist.jar" value="${dist.dir}/MathMLWysiwygEditor.jar"/>
        <property name="main.class" value="net.sourceforge.MathMLWysiwygEditor.Editor"/>
        <property name="lib.dir" value="${base.dir}/lib"/>

        <condition property="have.sources">
            <or><available file="${src.dir}"/></or>
        </condition>
        <path id="libs-build-classpath">
            <fileset dir="${src.dir}/${lib.dir}">
              <include name="*.jar"/>
            </fileset>
        </path>
    </target>

    <target name="compile" depends="initial" if="have.sources" description="Compile the code">
        <mkdir dir="${build.classes.dir}"/>
        <javac destdir="${build.classes.dir}"
            srcdir="${src.dir}"
            debug="true"
            deprecation="false"
            optimize="true" >
            <classpath refid="libs-build-classpath"/>
        </javac>
        <copy todir="${build.classes.dir}">
            <fileset dir="${src.dir}" excludes="${build.classes.excludes}"/>
        </copy>
    </target>

    <target name="jar" depends="compile" description="Create the Jar">
        <dirname property="dist.jar.dir" file="${dist.jar}"/>
        <mkdir dir="${dist.jar.dir}"/>
        <jar jarfile="${dist.jar}" basedir="${build.classes.dir}" index="true"
                includes="**/*.class,**/*.properties,**/*.gif,**/*.jpeg">
            <manifest>
                <attribute name="Main-Class" value="${main.class}" />
            </manifest>
        </jar>
        <copy todir="${dist.lib.dir}">
            <fileset dir="${build.classes.dir}/${lib.dir}/">
                <include name="**/*.jar"/>
            </fileset>
        </copy>
    </target>
</project>
```

Příloha G – XHTML s javascriptem předávající MathML apletu

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/Math/XSL/pmathml.xsl"?>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="content-script-type" content="text/javascript" />
        <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
        <title>Applet HTML Page</title>
        <script type="text/javascript">
            function exportMathML(){
                var mml = document.getElementById('input').value;
                document.getElementById('applet').focus();
                document.getElementById('applet').nactiDokument(null, mml);}
        </script>
    </head>
    <body>
        <!--[if !IE]> Firefox and others will use outer object -->
        <object classid="java:net/sourceforge/MathMLWysiwygEditor/Editor.class" id="applet"
              type="application/x-java-applet" archive="MathMLWysiwygEditor/MathMLWysiwygEditor.jar"
              height="620" width="650" >
            <param name="codebase" value="classes" />
            <param name="nameFunction" value="importMathML" />
            <!-- Konqueror browser needs the following param -->
            <param name="archive" value="MathMLWysiwygEditor/MathMLWysiwygEditor.jar" />
        <!--<![endif]-->
            <!-- MSIE (Microsoft Internet Explorer) will use inner object -->
            <object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" id="applet"
                  codebase="http://java.sun.com/update/1.5.0/jinstall-1_5_0-windows-i586.cab"
                  height="620" width="650" >
                <param name="codebase" value="classes" />
                <param name="code" value="net/sourceforge/MathMLWysiwygEditor/Editor.class" />
                <param name="archive" value="MathMLWysiwygEditor/MathMLWysiwygEditor.jar" />
                <param name="nameFunction" value="importMathML" />
                <strong>
                    This browser does not have a Java Plug-in.<br />
                    <a href="http://java.sun.com/products/plugin/downloads/index.html">
                    Get the latest Java Plug-in here.</a>
                </strong>
            </object>
            <!--[if !IE]> close outer object -->
        </object>
        <!--<![endif]-->
        <!-- XHTML not supported tag applet
        <applet id="applet" codebase="classes"
          code="net/sourceforge/MathMLWysiwygEditor/Editor.class"
          archive="MathMLWysiwygEditor/MathMLWysiwygEditor.jar" width="650" height="620">
            <param name="nameFunction" value="importMathML">
        </applet>-->
        <div id="MathML"></div>
        <form name="form">
            <textarea name="textField" id="input" rows="10" cols="78">
            MathML pro export do MathMLWysiwygEditoru
            </textarea><br />
            <input value="Export MML" type="button" onclick="exportMathML()" />
        </form>
    </body>
</html>
```