Univerzita Komenského v Bratislave
Fakulta Matematiky, Fyziky a Informatiky
Katedra Informatiky

Inkrementálna editácia matematického textu

(diplomová práca)

bc. Filip Gschwandtner

Štúdijný odbor: 9.2.1 Informatika
Vedúci: prof. RNDr. Branislav Rovan

Bratislava, 2009

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne len s použitím citovaných zdrojov.

. . . . . .

Abstrakt

Práca sa zaoberá ďalším vývinom už existujúceho open source projektu Euromath 2. Cieľom tohto projektu je poskytnutie WYSIWIG6.1 editácie nad xml dokumentami s viacerými mennými priestormi, pričom dôraz je kladený na profesionálne vyzerajúce výsledne dokumenty. V práci sa zaoberám jednak súčasnými problémami projektu tak i jeho budúcim smerovaním. Podľa analýzy celého projektu som zameral svoje úsilie na zlepšenie editačných schopností projektu smerom k používateľovi. Takýto krok znamená viditeľné zlepšenie použiteľnosti. Následkom tohto kroku sa zväčší komunita tohto projektu a teda i spätný tlak na budúce zlepšovanie projektu.
Zanalyzoval som technické možnosti platformy Eclipse, na ktorej základoch je projekt postavený. Zanalyzoval som tiež menný priestor matematického textu(MathML) použitého v projekte a navrhol som grafické rozhranie pre editáciu matematického textu. Zanalyzoval som aj možnosti riešenia požiadavky vkladania matematického textu vo forme Texu.
Medzi priority tejto práce patrí i podpora budúcich vývojárov projektu. Jednak stručné zhrnutie výsledkov predošlých diplomových prác, na ktoré nadväzujem, ale hlavne analýza technických možností Eclipsu a analýza zdrojových súborov projektu umožnia budúcim vývojárom rýchlejšie sa adaptovať problematike a zameriať sa na úzko špecifické problémy a rozšírenia Euromathu. Samozrejme okrem projektového zázemia som črtol i možné vývojové smery vyplývajúce so súčasných problémov a z budúcich rozšírení projektu. Popisované budúce rozšírenia sa vzťahujú na rozšírenia podpory menných priestorov a ich grafického rozhrania editácie.

Kľúčové slová: projekt Euromath 2, editácia matematického textu, MathML

Obsah

Úvod

V súčasnej dobe existuje veľké množstvo softwaru zameraného na tvorbu dokumentov. Niektoré sú úzko špecifické, iné sú orientované všeobecnejšie. Pre bežnú kancelársku tvorbu dokumentov si ľudia vystačia s dostupným softwarom. V prípade vedeckej činnosti sú ale požiadavky na editovanie dokumentov oveľa vyššie. Je nutné podporovať rôzne menné priestory (zložitý matematický text, chemické značky,...) a ich vzájomné zobrazovacie i formátové prepojenie. Z tohto dôvodu je vhodných editačných aplikácií určených pre vedeckú činnosť málo. Jedným z najrozšírenejších nástrojov na vytváranie vedeckého textu je (La)Tex. Obsahuje mnoho užitočných vlastností, avšak jeho hlavnou nevýhodou je nutnosť kompilácie dokumentu pre potrebu zhodnotenia výslednej podoby dokumentu. Túto nevýhodu odstraňujú WYSIWIG6.1 editory. Jedná sa o spôsob editácie, pri ktorej sa dokument vizuálne nemení pri zmene formy(editovacia forma, tlačová forma, súborová forma).
Takýmto editorom sa zaoberá aj projekt Euromath 2. Cieľom mojej diplomovej práce je rozšíriť tento projekt. Jedným z prínosov projektu Euromath 2 je jeho podpora viacerých menných priestorov pri editácií xml dokumentov. V súčasnosti však existuje mnoho kvalitných xml editorov zvládajúcich aj viaceré menné priestory v jednom dokumente. Odlišujú sa od projektu Euromath 2 nedostatkom všeobecnosti a flexibility. Euromath sa totiž zameriava na prácu s ľubovoľnými, t.j. aj používateľom zadefinovanými, mennými priestormi. Iné softwarové riešenia sa zameriavajú skôr len na kombinácie malého počtu všeobecne známych menných priestorov. Z časti je to spôsobené súčasnou neexistenciou jednotného štandartu zobrazovania viacerých menných priestorov v jednom dokumente. Euromath sa v tomto smere snaží ponúknuť komplexné riešenie.
Po hlbšej analýze problematiky projektu Euromath vyplynula potreba zvýšiť použiteľnosť editora,t.j. vytvoriť grafické rozhranie pre editáciu matematic-

kého textu, a poskytnúť budúcim vývojárom projektu lepšie technické základy pre ich prácu. Podpora pre budúcich vývojárov projektu vyplynula z charakteru predošlých diplomových prác zameraných na projekt Euromath 2, ktorý bol prevažne teoretický. Vyššia použiteľnosť editora zas vyplynula z cieľu vytvoriť väčšiu komunitu používateľov, ktorý následne vytvoria dostatočný tlak na ďalší vývoj projektu.
Pre potreby obidvoch cieľov bolo nutné najprv zanalyzovať možnosti platformy Eclipse, technológie na ktorej je projekt postavený. Analyzoval som hlavne vlastnosti funkčného a grafického rozšírenia tejto platformy. Pokračoval som analýzou štandardu MathML, slúžiaceho na zobrazenie a manipulovanie s matematickým textom. Nakoniec som grafické rozhranie navrhol a implementoval. Pre podporu budúcich vývojárov som okrem analýzy platformy Eclipse analyzoval i zdrojové súbory projektu, aby som umožňil rýchlejšiu orientáciu v projekte na programovacej úrovni.
Dodatočným cieľom pre zlepšenie použiteľnosti editora bolo spracovanie požiadavky vkladať matematický text vo forme (La)Texu. Vzhľadom na dodatočnosť tohto cieľa a neexistencie vhodného softwaru tretej strany spĺňajúceho požadované podmienky, som navrhol riešenie tohto problému iba teoreticky.

Kapitola 1

Úvod do projektu Euromath 2

Naším cieľom v tejto kapitole bude poskytnúť ucelený pohľad na open source projekt WYSIWYG6.1 xml editora Euromath 2[3], objasniť základné ciele a motivaciu tohto projektu a priblížit jeho vnútornú štruktúru, jeho naväznosti a teóriu stojacu za ním.

1.1 Predstavenie, ciele a motivácia

Projekt Euromath 2 je projekt WYSIWYG editora xml dokumentov zameraných hlavne na podporu matematických, technických a iných formálnych textov. Je zameraný pre matematikov a iných výskumných pracovníkov, ktorým nepostačuje editor pre bežné kancelárske činnosti. Pozornosť je kladená na profesionálny vzhľad editovaného textu, ktorý sa zachováva aj po opätovnom načítaní či vytlačení do papierovej formy. Táto vlastnosť sa nazýva WYSIWYG vlastnosť. Ďalej sa kladie dôraz na jednoduchosť editovania a na flexibilitu a rozšíriteľnosť editora. Medzi jeho základne črty (v aktuálnej verzií) patrí:

1. open source licencia, konkrétne licenciou MPL 1.1
2. základný XSL-FO editor (schopný editovať XML dokumenty transformovateľné na XLS-FO6.1)
3. plná podpora dokumentov s viacerými mennými priestormi
4. podpora SVG6.1, MathML6.1

5. podpora XHTML (len základná)

6. podpora entít

7. plne nastaviteľný transformátor spúšťajúci XSLT skripty či Eclipse Java pluginy

8. automatické doplňovanie XML elementov/atribútov na základe DTD, XML Schema, Relax, RelaxNG, Trex

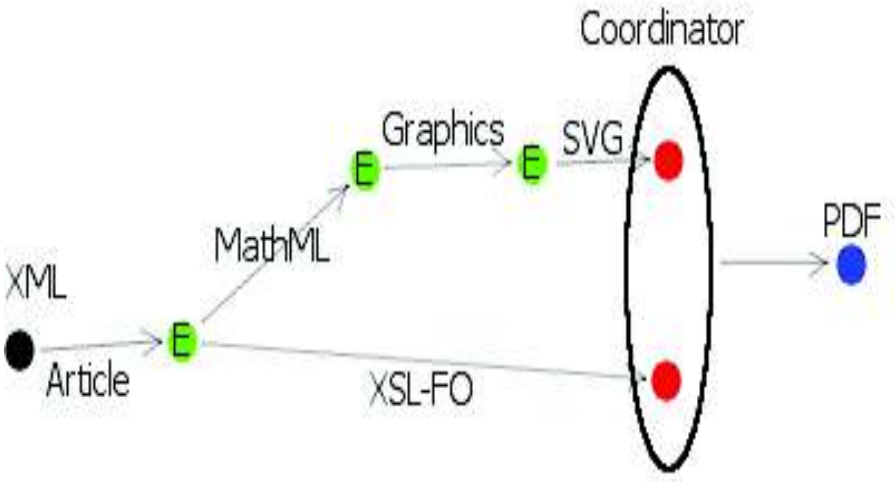
Hlavné pozitívum ale aj negatívum tohto projektu spočíva v použití XSLT transformácii(6.1). Pozitívne je to z dôvodu veľkej vyjadrovacej sily takýchto transformácii, ktorá zaručuje vysokú flexibilitu editora. Nevýhodou je to zas kvôli tomu, že sa v sučasnosti tieto transformácie vykonávajú ako úplné XSLT transformácie. To spôsobuje aj pri minimálnej zmene dokumentu jeho úplnú XSLT transformáciu, ktorá v konečnom dôsledku znemožňuje prácu s editorom v reálnom čase. Riešením je navrhnúť a implementovať XSLT procesor podporujúci čiastočnú XSLT transformáciu. Táto úloha je avšak príliš časovo náročná vzhľadom na čas vymedzený na vypracovanie diplomovej práce. Vhodnou alternatívou by bolo použitie už existujúcich open source XSLT procesorov podporujúcich čiastočnú transformáciu. V skutočnosti ,ale žiadne voľne dostupné, resp. open source, riešenie neexistuje. Existujú len licenčne uzavreté procesory s takýmito vlastnosťami, avšak ich použitie by bolo proti licenčnej filozofii projektu Euromath.
V tomto bode je stále otázny vývoj takéhoto projektu. A to hlavne z dôvodu nespornej existencie veľkého množstva xml editorov. Dôvodom vývoja sú vlastnosti, ktoré sa v žiadnom inom editore nespájajú v jeden celok. Jednou z takýchto vlastnosti je samotná editácia xml dokumentov. Iné dostupné xml editory sa zameriavajú na editáciu xml súborov, to však nie je totožné s editáciou xml **dokumentov**. Rozdiel medzi týmito slovnými spojeniami spočíva v tom, že xml dokument predstavuje komplexný xml súbor obsahujúci veľa rôznych formátov, pričom editor by sa mal zameriavať na ich jednoduchú a prirodzenú editáciu. Bežne dostupné xml editory avšak nepodporujú editáciu viacerých menných priestorov v prirodzenej forme. Zoberme si ako príklad matematický text, ktorý sa dá síce editovať v xml forme (MathML), avšak je to príliš zložité oproti prirodzenej forme matematického zápisu. Ďalším dôvodom použitia editora Euromath je jeho rozšíriteľnosť. Takáto vlastnosť sa ukáže byť užitočná v prípade tvorby vlastného menného priestoru. V prípade tvorby vlastného menného priestoru by sa síce mohli použiť i iné xml editory,

t.j. za predpokladu, že podporujú vlastne definované XSLT transformácie a validáciu dokumentu, problém však nastáva vtedy, keď by sme sa k nemu rozhodli pripojiť iný menný priestor. Zistili by sme, že na niektoré menné priestory XSLT transformácia nevie rozumne pretransformovať (napr. obrázky formátu SVG). Navyše, tiež neexistuje žiadny štandard na spracovanie dokumentov s viacerými mennými priestormi. Projekt Euromath 2 poskytuje v tomto prípade komplexné riešenie.

1.2 Vnútorna Štruktúra projektu a náväznosti

Projekt Euromath 2 bol vytvorený pomocou frameworku vývojového prostredia Eclipse[7]. Software projektu Euromath je publikovaný v dvoch formách. Prvou formou je forma pluginu pre vývojové prostredie Eclipse 3.3, ktorá sa pri správnom umiestnení do adresárovej štruktúry automaticky týmto vývojovým prostredím rozpozná a integruje do seba. Druhou formou je forma rozšírenej RCP(6.1) aplikácie schopnej samostatného behu i bez vývojového prostredia Eclipse. RCP je v skutočnosti pluginovo minimalistické prostredie Eclipsu, ktoré je oproti pôvodnému prostrediu ochudobnené o všetky pluginy, ktoré sú nepotrebné k spustenie systému, správe pluginov a GUI. RCP aplikácie tak ponúkajú pružné prostredie založené na pluginoch, vytvorené GUI a iné výhody systému použitého pre tvorbu prostredia Eclipse.
V architektúre Euromathu sa využíva architektonický vzor MVC(6.1). Jedná sa o vzor obsahujúci 3 časti: Model, View a Controler. Model predstavuje dátový model aplikácie a operácie na ňom. View je časť aplikácie, ktorej jedinou úlohou je vykreslovanie a problémy s tým spojené. Úlohou Controlera nie je len zabezpečovanie komunikácie medzi zvyšnými časťami, ale hlavne zabezpečovanie interakcie s používateľom (GUI6.1). Editor sa celkovo skladá zo 4 častí. Nasledovne si popíšeme ako sú tieto jednotlivé časti vložené do vzoru MVC:

Model(1.časť) je časť zaoberajúca sa validitou dokumentu podľa gramatiky a poskytujúca kontextové editačné operácie, ktoré sú ponúknuté používatelovi na výber. Funkcionalita tejto časti projektu Euromath 2 bola umiestnená do samostatného projektu s názvom **Schematic**[4]. V tomto projekte je základným objektom trieda SchemaPool, starajúca sa o inštancie rozhrania ISchema, ktoré predstavuje validačné pravidlá pre 1 menný priestor. Projekt obsahuje aj podporu (factory) pre tvorbu



Obr. 1.1: Príklad práce projektu GENE

ISchema objektov priamo zo štandartných ISchema súborov(DTD,...). ISchema objekty pre daný dokument naviaže objekt DocumentSchema, ktorá poskytne pre dokument aj tzv. autocompletition, t.j. množinu valídnych operácií pre daný dokument.

Model(2.časť) je časť zaoberajúca sa transformáciou dokumentu pomocou XSLT. Jedná sa o transformovanie, mapovanie a delenie dokumentu podľa menných priestorov pre ďalšie spracovanie. Funkcionalita tejto časti bola taktiež umiestnená do samostatného projektu s názvom **GENE**[5]. Prácu projektu GENE si vysvetlíme na jednoduchom príklade, ktorý je znázornený na obr.1.1. Na začiatku máme xml dokument (čierna farba) a máme k dispozícii sadu exportérov(zelená farba), ktoré dokážu konvertovať 1 formát na iný 1 formát. Úlohou projektu GENE je vytvoriť pomocou týchto exportérov strom výpočtu, aby sa celý xml dokument konvertoval na iný formát. V tomto prípade na formát PDF6.1. Na obrázku je už znázornený takýto výpočet. Xml súbor prejde XSL transformáciou, aby sme oddelili 2 menné priestory: XSL-FO6.1 a MathML. Ďalej sa snažíme dáta v týchto menných priestoroch skonvertovať tak, aby boli akceptované koordinátorom, ktorý je zodpovedný za priamu produkciu výstupného formátu. XSL-FO je

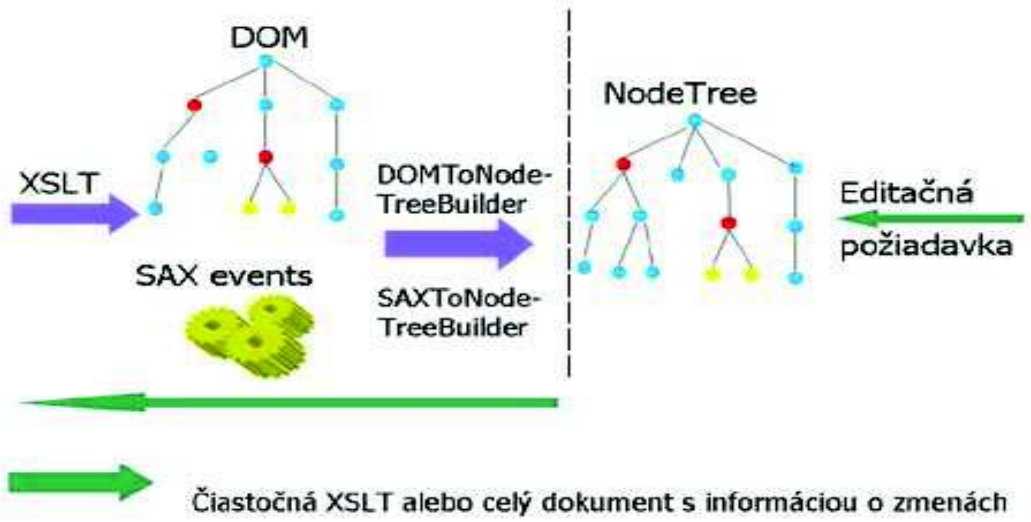
vhodný formát, ale MathML nie. Treba ho skonvertovať na obrázok formátu SVG6.1 pomocou exportéra JEuclid[6]. A nakoniec koordinátor (FOP) vygeneruje požadovaný výstup. Jediné čo GENE potrebuje je vstup, výstupný formát a množinu vhodných exportérov. V prípade EuroMathu sa budú využívať i čiastočné medzivýpočty ako napr. SVG obrázok MathML časti xml dokumentu.

Controller je časť riadiaca editáciu a GUI. V tejto časti sa nachádza inštancia triedy EditorSite, ktorá slúži ako kontajner pre hierarchiu IEditor objektov. Inštancia IEditora predstavuje Editor pre súvislú časť xml dokumentu, ktorý sa nachádza v 1 mennom priestore. Túto hierarchiu editorov tvorí samotný EditorSite podľa informácií z GENE. Zatiaľ existujú len 2 triedy implementujúce rozhranie IEditor. Takouto triedou je XMLEditor, ktorý je používaný ako štandartný editor na editáciu bližšie nešpecifikovaných menných priestorov. Jeho základ spočíva na technológií GEF(6.1) a umožňuje selekciu elementov, editáciu pomocou klávesnice a cez menu, clipboard, undo/redo a zoom funkcionalitu. Druhou triedou je TextEditor rozširujúci XMLEditor o funkcie selekcie textu a písania textu pomocou klávesnice. V tejto časti budeme musieť rozšíriť skupinu editorov o editor matematického textu.

View je časť zodpovedná za vykreslenie výstupu. Euromath pracuje s frameworkom GEF, kde sú základným stavebným kameňom inštancie rozhrania IFigure. Podobne ako bola vytvorená hierarchia inštancií rozhrania IEditor, tak je vytvorená hierarchie inštancií rozhrania IRenderer zodpovedných za vykreslovanie jednotlivých menných priestorov. XSL-FO vykresluje FORenderer a MathML vykresluje MMLRenderer s layoutom poskytnutým projektom JEuclid.

1.3 Teoria stojaca za Euromathom

V tejto časti si popíšeme najzaujímavejšie informácie z predošlých diplomových prác([1],[2]), na ktoré nadväzujem. Zameriame sa hlavne na veci súvisiace s témou(cieľom) tejto diplomovej práce, poprípade súvisiace s jej voľbou. Uvedieme si ale aj teoretické riešenia zaujímavých problémov súvisiacich s témou, ktoré by mohli pomôcť budúcim vývojárom lepšie pochopiť problematiku projektu Euromath.

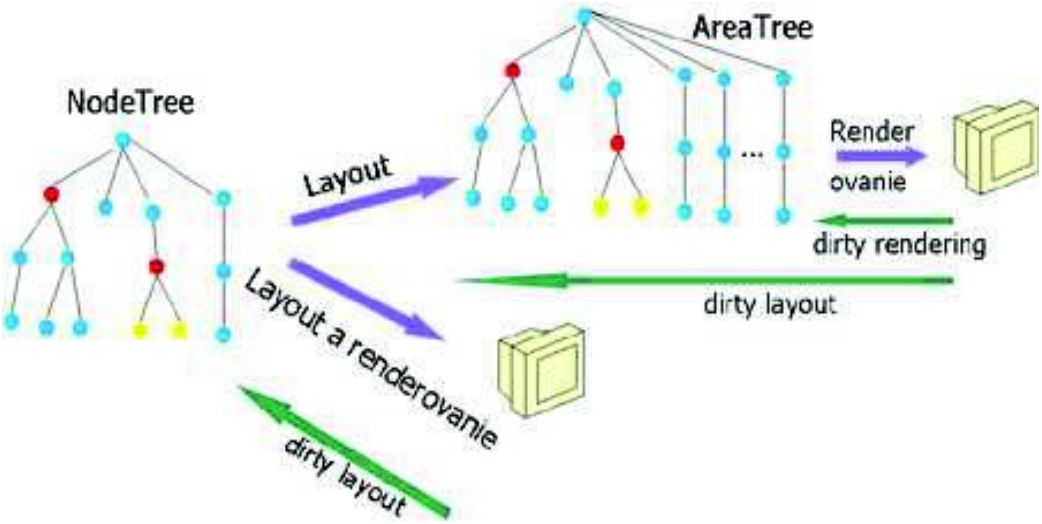


Obr. 1.2: náčrt č.1

1.3.1 Návrh renderovacieho frameworku

V tejto časti budeme čerpať z diplomovej práce Martina Kollára s názvom Renderovanie XML dokumentov WYSIWYG editovaní[2]. Cieľom jeho práce bolo vytvoriť framework pre tvorbu XML rendererov vhodných pre WYSIWYG editáciu XML dokumentov.Výsledky tejto práce,resp. jej implementácia, síce nie sú priamo súčasťou editora Euromath 2, avšak prináša cenné informácie ohľadom WYSIWYG renderovacích problémoch a ich riešení, a tak napomáha k celkovému pochopeniu editora Euromath 2. Vzhľadom na okrajovosť témy len načrtneme celkový problém, ktorý by mal riešiť renderovací framework.
Celkový náčrt jednotlivých častí frameworku a procesov medzi nimi nutných na vykreslenie xml dokumentu je znázornený na obr.1.2, 1.3. Teraz si bližšie tieto náčrtky popíšeme:

DOM Tree je dátová štruktúra, ktorá vznikne z xml dokumentu XSL transformáciou6.1. XSL transformácia je transformácia pôvodného xml dokumentu (povedzme v podobe DOM stromu) definovaná XSL transformačnými pravidlami, ktoré môžu jednak meniť jednotlivé xml elementy ale aj vytvárať nové. XSL transformácie sa môžu použiť napríklad na vytváranie obsahu knihy, či očíslovania strán. Z tohto je očividné, že



Obr. 1.3: náčrt č.2

spätná XSL transformácia je celkom netriviálna záležitosť a pre všeobecnú XSL transformáciu veľmi ťažka (ak vôbec možná).

Node Tree je prvá dátová štruktúra nachádzajúca sa vnútri frameworku. Všetky dátové štruktúry a výpočty, ktoré boli doposiaľ uskutočnené (mimo NodeTree) sa považujú časť nazvanú FATS(6.1). Do FATS patrí i samotný pôvodný xml dokument aj s procesmi jeho načítania zo súborového systému. Hranicu medzi frameworkom a FATS dobre vystihuje zvislá prerušovaná čiara na náčrtku č.1 (1.2). Node Tree predstavuje len vnútornú(frameworkovú) reprezentáciu xml dokumentu. Môže byť vytvorená prehľadávaním DOM stromu (DOM Tree) pomocou DOMtoNodeTreeBuildera alebo zachytávaním SAX udalosti pomocou SAXtoNodeTreeBuildera.

Area Tree je dátová štruktúra obsahujúca okrem informácií obsiahnutých v Node Tree i informácie o rozmiestnení jednotlivých elementov na obrazovke. Tieto dodatočné informácie dostaneme z procesu rozmiestňovania(layoutovania) elementov. Z tejto dátovej štruktúry sa posielajú dáta priamo na vykreslenie (rendering). Táto dátová štruktúra môže byť aj vynechaná v prípade spojenia rozmiestňovacieho a vykreslovacieho procesu.

Pozrime sa bližšie na preposielanie požiadaviek od používatela a na tzv. dirty editačné techniky. Fialové šípky na náčrtkoch zobrazujú smer vyhodnocovania dát od xml dokumentu k vyrenderovanému výsledku(väčšinou k obrazovke). Používateľ však pri použití editora zmení obsah dát a to na strane editora,t.j. na strane vyrenderovaného výsledku. Na správnom fungujúcom frameworku zostáva preposlať editačnú požiadavku používatela smerom k FATS. Presne tento proces zobrazujú na náčrtkoch zelené šípky. Po preposlaní by mal FATS rozpoznať či zmena bola valídna alebo nie. Samozrejme to už je starosť FATS a nie frameworku. Ak bola zmena valídna tak by sa mal celý xml dokument znova spracovať od XSL transformáciu až po AreaTree a nanovo vyrenderovať. Avšak ak by aj tá najmenšia zmena znamenala spracovanie celého xml dokumentu nanovo, tak by žiadny WYSIWYG editor nepracoval v reálnom čase, a preto pán Kollár navrhol použitie tzv. dirty editačných techník. Jedná sa o editačné techniky, ktoré sú použité len v prípade, že framework bezpečne vie ako by xml dokument alebo jeho časť vyzerala po celkovom prechode výpočtovými operáciami od samotného xml dokumentu. Medzi takéto editačné požiadavky patrí napríklad zmena farby textu alebo niečo podobne triviálne, čo očividne framework môže okamžite správne interpretovať a nemusí čakať na znovuspracovanie celého xml dokumentu. Poznáme 2 typy dirty editačných techník: dirty rendering a dirty layout. Dirty layout je použitý v prípadoch, keď vieme zaručiť čo by sa presne vygenerovalo z xml dokumentu do dátovej štruktúry NodeTree, ale nevieme či náhodou editačná požiadavka používateľa nespôsobí iné rozmiestňovanie grafických elementov na obrazovke. V takom prípade je nutné spraviť dodatočné rozmiestňovanie a rendering. V prípade použitia dirty renderingu musíme vedieť zaručiť, že nové rozmiestňovanie nie je potrebné, pretože staré rozmiestňovanie sa nezmení. Medzi takéto prípady patrí i spomínaná zmena farby textu.
Bohužiaľ techniky dirty rendering a dirty layout neboli vhodné zakomponované do projektu Euromath, a tak spracovanie požiadaviek v reálnom čase je ešte stále nesplnenou vlastnosťou editora.

1.3.2 Editačné techniky a XSL transformácia

V tejto podkapitole budeme čerpať z diplomovej práce Tomáša Studva s názvom WYSIWYG editačné techniky XML dokumentov s použitím XSLT[1].

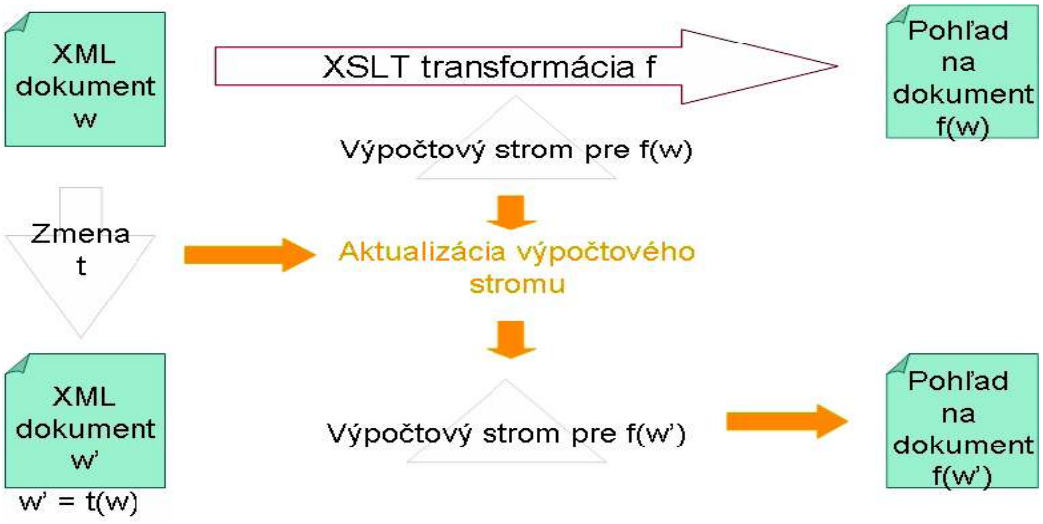
Cieľom tejto práce bolo aplikovať techniku editácie pohľadu na XSLT a implementovať správu pohľadu pomocou čiastočnej XSL transformácie. Druhý cieľ tejto diplomovej práce sa síce nepodaril dosiahnuť, avšak i tak diplomová práca poskytuje cenné informácie o XSL transformáciach, ktoré sú nevyhnutnou súčasťou editora Euromath 2, resp. robia ho výnimočným v porovnaní s inými existujúcimi WYSIWYG editormi, pretože mu dávajú nevýdanú flexibility
Základným predpokladom ďalšieho textu bude fakt existencie gramatiky pre xml dokument. Takáto gramatika je schopná daný dokument vygenerovať. Xml dokument je teda jazykom generovaným svojou gramatikou. Gramatika sa v takýchto prípadoch nevyužíva na generovanie xml dokumentu, ale len na jeho validáciu, t.j. na overenie možnosti generovania obsahu xml dokumentu pravidlami z danej gramatiky. Gramatiky pre xml súbory sa dajú zapísať pomocou DTD či XSchema súborov. Samozrejme, že existujú xml dokumenty bez vopred známej gramatiky, ale nimi sa zapodievať nebudeme.
Editor sa vo všeobecnosti skladá zo 4 častí:

- časť spravujúca dokument a asociovanú gramatiku
- pohľad, t.j. časť vykonávajúcu transformácie a poskytujúca transformovaný dokument
- časť riadiaca editáciu a interakciu s používateľom
- časť zobrazujúca pohľad

Editor je automat pracujúci nad konfiguráciou (dokument, gramatika, transformácia). Proces editácie predstavuje cyklus voľby editačnej požiadavky, jej aplikácie a zobrazenia jej pohľadu. Podľa miesta vykonávania editačnej požiadavky delíme editory na 2 skupiny. Prvá používa techniku editácie dokumentu a druhá techniku editácie pohľadu.

Technika editácie dokumentu

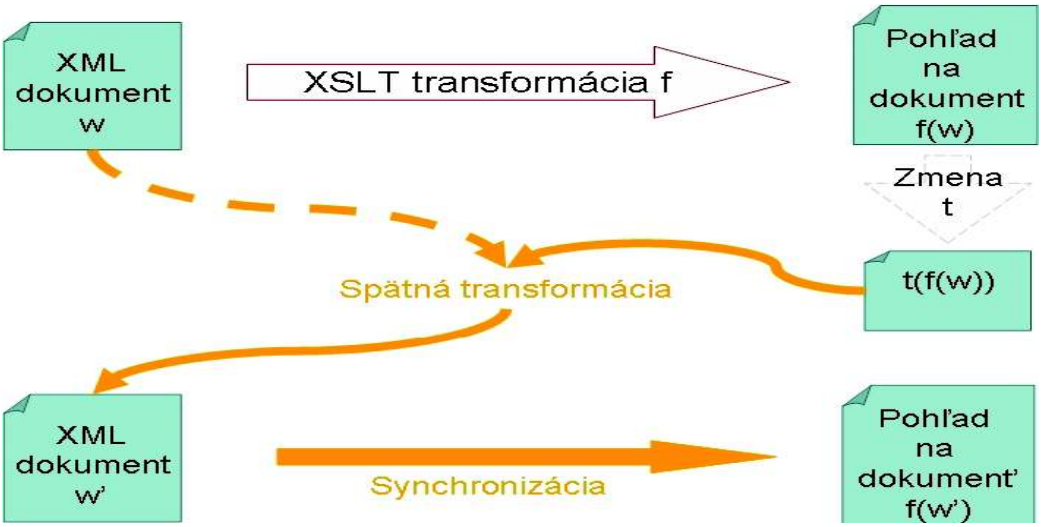
Pri tomto spôsobe editácie sa robia zmeny priamo do dokumentu, pričom sa pri každej zmene nanovo vytvorí pre xml dokument pohľad a vykreslí sa. Tento spôsob editácie je použitý i v projekte Euromath 2. Jedná sa o najjednoduchší spôsob editácie. Medzi najproblematickejšie časti takéhoto riešenia patrí synchronizácia pohľadu a mapovanie grafického rozhrania do



Obr. 1.4: Technika editácie dokumentu

dokumentu. Druhý problém bol vyriešený zavedením identifikátorov (ID) pre xml uzly, ktoré sa nemenili ani v NodeTree či AreaTree. Prvý problém zostáva naďalej problematický, avšak nie z pohľadu možnosti vytvorenia pohľadu ale z pohľadu rýchlosti danej operácie. Pri veľastranových dokumentoch (>50 strán) je celkové znovuvytvorenie pohľadu a jeho vykreslenie (rendering) časovo príliš náročná operácia (>1 sekunda) pre WYSIWYG editor. Dokonca už aj samotné vykreslenie je dosť časovo náročná operácia. Táto situácia[1] sa dá vyriešiť čiastočnou XSL transformáciou. Pán Studva avšak takúto čiastočnú transformáciu neimplementoval vzhľadom na zložitosť riešenia. Naznačme si aspoň jeho hrubú myšlienku, ktorá spočíva v udržovaní výpočtový strom XSL transformácie. Pán Studva označil takýto strom ako XT strom. Uzly XT stromu väčšinou korešpondujú s riadiacími príkazmi jazyka XSLT. Keď nastane v dokumente zmena, tak identifikujeme v XT strome uzly, ktoré potrebujú aktualizáciu a aktualizujeme od nich výpočet. Keď tým aktualizujeme výpočet, tak ešte musíme aktualizovať výstup, aby sme dostali aktuálny pohľad. Celý proces znázorňuje obr1.4.

[1] t.j. aspoň XSL transformácia keď nie vykreslovanie



Obr. 1.5: Technika editácie pohľadu

Technika editácie pohľadu

Pri tejto technike sa needituje dokument, ale samotný pohľad. Po každej zmene pohľadu sa aktualizuje dokument spätnou transformáciou. Samotná spätná transformácia však nestačí, lebo XSL transformácia na novom dokumente môže spôsobovať v pohľade ďalšie zmeny, ktoré bez jej vykonania nezachytíme. Preto nastáva ešte ďalšia aktualizácia pohľadu. Celý proces je zachytený na obr.1.5.

Editácia pohľadu riadená gramatikou

Editácia pohľadu riadená gramatikou je rozšírenie editácie pohľadu o vlastnosť, ktorá nedovolí vykonať operáciu, po ktorej by dokument nebol valídny podľa gramatiky. Pri takejto editácií by používateľ musel mať znalosť gramatiky, podľa ktorej sa validuje dokument a to by bolo obtiažne. Preto je vhodnejšia technika editácia pohľadu gramatikou s výberom, ktorá navyše používateľovi ponúka operácie na výber. Zistiť aké operácie ponúknuť používatelovi nie je jednoduché. Pre gramatiky so silou turingovho stroja by sa muselo nutne dlho simulovať gramatické odvodenia, aby sme sa dopracovali k možným prípustným operáciam. Pán Studva pri hľadaní riešenia tejto otázky nakoniec zvolil postup tvorby vlastného modelu, ktorý nazval TOS

(Transformácia Obojsmerných Šablón). Ako už názov napovedá, jedná sa o rozšírenie XSL transformácie o obojsmerné šablóny. Transformácia prejde cez danú šablónu iba vtedy ak to čo šablóna vytvorí dokáže prejsť i inverznou šablónou a vygenerovať pôvodný vstup. Pre jednoduchosť (resp. pre odvrátenie prílišnej zložitosti) sa model obmedzil na určité výrazové prvky jazykov XSLT a XPath[2]. Výsledkom bola slabá výrazová sila modelu.

Prvotným cieľom mojej diplomovej práce vyplýval zo snahy vylepšiť projekt Euromath, avšak tvorba procesora pre čiastočnu XSL transformáciu bola nad časové možnosti diplomovej práce. Existovala síce možnosť pokúsiť sa vybrať open source procesor a vhodne ho upraviť, avšak i tak by bola takáto snaha príliš časovo náročná. Z podobných zložitostných príčin asi ani neexistuje open source XSLT procesor s takouto vlastnosťou. Nakoniec sa cieľ mojej diplomovej práce zmenil na pridanie podpory editovania matematického textu. Síce sa tým nevyrieši rýchlostná otázka, ale sa aspoň rozšíri funkcionalita, aby sa editor dal používať vo väčšej miere. Takýto krok vytvorí priestor pre používateľov, ktorý by mohli aplikáciu používať a teda v konečnom dôsledku aj väčšiu snahu aplikáciu ďalej vyvíjať.

[2]Jazyk vytvorený na adresovanie rôznych častí xml dokumentov

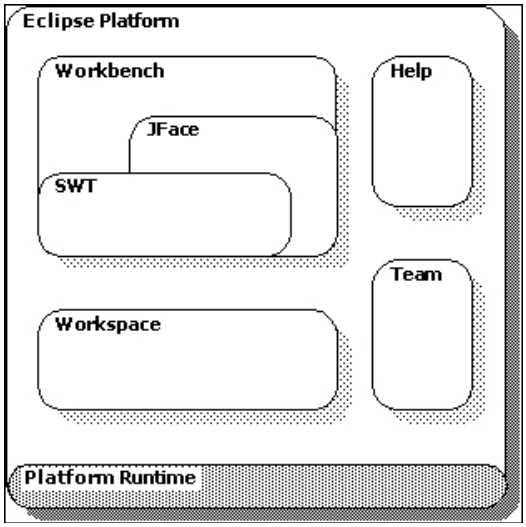
Kapitola 2

Platforma Eclipse

2.1 Základne definície a rozširovacie mechanizmy

Základnou platformou projektu Euromth 2 je platforma Eclipse. Nemýľme si však pojmy platforma pre vývoj a vývojové prostredie. Eclipse je totiž nielen názov pre vývojové prostredie určené pre vývojárov, ale aj základnou platformou pre vývoj, na ktorej môže novo vznikajúca aplikácia stavať. V tejto kapitole budeme hovoriť o Eclipse ako platforme pre vývoj, konkrétne o tom čím Eclipse je a čo ponúka ako platforma na vývoj. Všetky informácie tejto kapitoly pochádzajú z oficiálneho zdroja [9], poprípade z mojej bakalárskej práce [8].

Platforma Eclipse bola vytváraná ako veľmi pružné prostredie skladajúce sa s veľkého počtu dobre integrovaných pluginov. Využíva sa v ňom idea tzv. otvorenej architektúry. Jedná sa o spôsob vytvárania zložitejšej architektúry platformy na základe určitého pridávanie vrstiev pluginov. V úplnom jadre existuje len relatívne malý runtime engine spravujúci všetky pluginy. Jeho prácou je dynamicky detekovať pluginy, načítavať ich a spúšťať. Všetko ostatné je už v Eclipse vytvorené cez pluginy. Nové vrstvy pluginov sa viažu na predošlé pomocou špeciálnych bodov tzv. **extension points**. Samozrejme, že nové vrstvy môžu definovať vlastné extension pointy, na ktoré sa naviažu ďalšie vrstvy. Platforma Eclipse nie je voľne distribuovaná len ako samotný runtime engine, ale už aj s množstvom vrstiev. Rozdiel medzi vývojovým prostredím Eclipse a platformou Eclipse spočíva práve v zakomponovaných

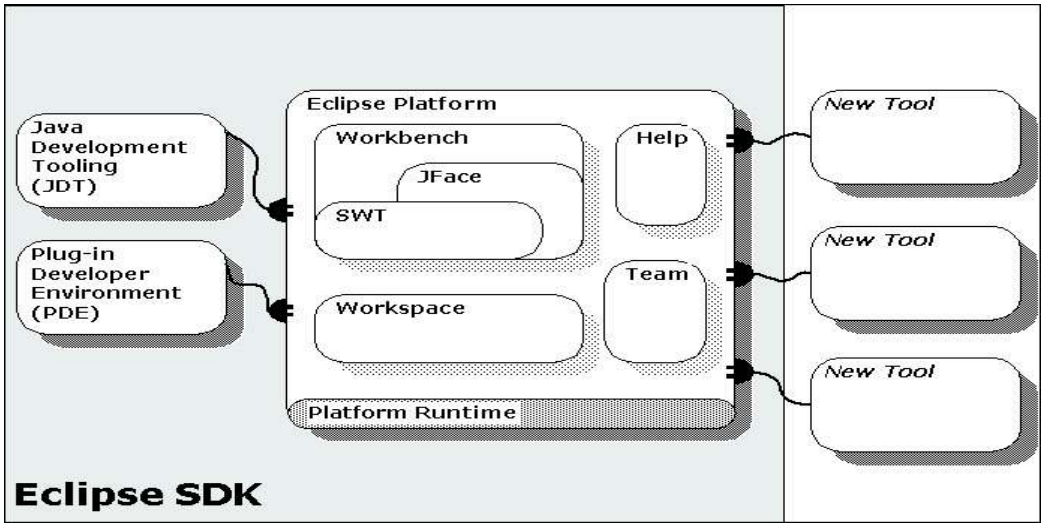


Obr. 2.1: Architektúra platformy Eclipse

pluginoch. Platforma Eclipse(viď. obr. 2.1) obsahuje základné pluginy, ktoré však už poskytujú z programátorského hľadiska netriviálnu funkcionalitu, konkrétne máme na mysli funkčné GUI s editormi, náhľadmi či integrovaným menu, alebo podporu práce so zdrojmi(source management) a iné. Vývojové prostredie Eclipse sa naproti tomu skladá nielen z celej platformy Eclipse ale má i mnohé iné pluginy, ktoré vytvárajú z účelovo nevyprofilovanéj platformy Eclipse ucelené vývojové prostredie(viď. obr. 2.2). Medzi takéto profilovacie pluginy patria napríklad pluginy zabezpečujúce kompiláciu a debugovanie kódu v jazyku Java. Práve samotné vývojové prostredie Eclipse je najlepšou ukážkou softwaru, ktorý si za svoje jadro zvolil platformu Eclipse a vhodne ju rozšíril, aby spĺňal požiadavky na daný software.

Tabuľka 2.1 mapuje hlavné komponenty Eclipse runtime. Z nášho pohľadu budú pre nás zaujímavé časti Platform runtime (definovanie, zapájanie a odpájanie pluginov), Workbench UI (grafické prostredie využité na tvorbu matematického editora) a možno ešte pomocné pluginy (poskytnutie dodatočnej funkcionality v prípade potreby). S ostatnými komponentmi sa nebudeme zaoberať z dôvodu malej či žiadnej potrebnosti pri implementácií.

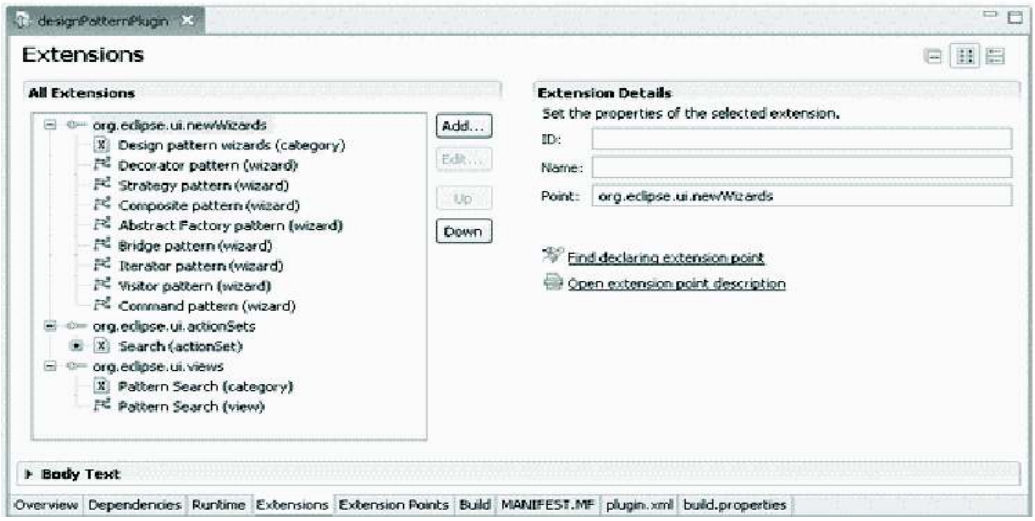
Teoreticky sme si teda už vysvetlili architektúru platformy Eclipse a pripájenie ďalších pluginov k tejto platforme. Presuňme svoju pozornosť ku praktickej stránke pripájania pluginov. Zadefinujeme si požiadavky na plugin,



Obr. 2.2: Architektúra vývojového prostredia Eclipse

Runtime komponent	**Význam**
Platform runtime	Definuje extension point a plug-in model. Spravuje pluginy (hľadanie, spúšťanie).
Resource management (workspace)	Definuje API na vytváranie a správu zdrojov(resources): projekty, súbory a priečinky.
Workbench UI	Implementácia user interface, definuje extension pointy pre ďalšie UI komponenty a poskytuje ďalšie nástroje (JFace,SWT) na tvorbu user interfacov.
Help system	Definuje extension points pre pluginy poskytujúce nápovedu.
Team support	Podpora programovania v skupinach.
Debug support	Definuje model na debugovanie a UI triedy pre tvorbu debugerov a spúšťačov(launchers).
Other utilities	Pomocné pluginy (vyhľadávanie, porovnávanie zdrojov a iné).

Tabuľka 2.1: Komponenty Eclipse runtime



Obr. 2.3: Editor pre plugin.xml

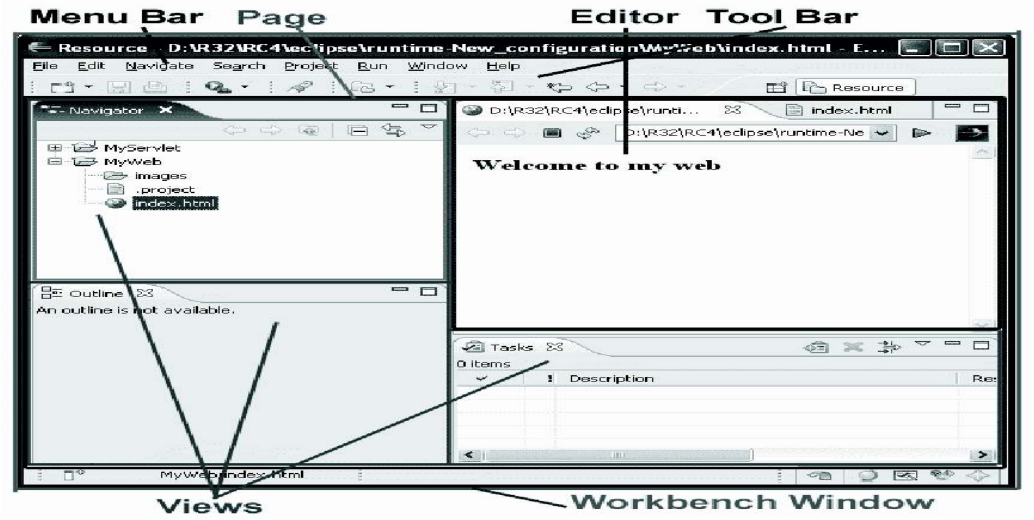
ktoré vytvoria z bežného projektového pluginu plugin rozširujúci platformu Eclipse. Prvým viditeľným rozdielom medzi bežnými pluginmi vo vývojovóm prostredí Eclipse a pluginmi rozširujúcimi platformu Eclipse je existencia dvoch súborov : **manifest.mf** a **plugin.xml**. Súbor manifest.mf obsahuje nízko úrovňové informácie ohľadom zaobalenia pluginu. Samotný platform runtime (viď. obr. 2.1) je implementovaný ako OSGi framework a balík (bundle), základný kameň frameworku, existuje práve v podobe pluginu. Súbor manifest.mf hovorí frameworku základné informácie o balíku, t.j. pluginu. Nejedná sa len o základné informácie ako meno, ale aj o informácie ako napríklad ktoré iné balíky daný balík potrebuje pre svoj správny chod. Súbor plugin.xml je z pohľadu pripájania pluginov oveľa zaujímavejší, pretože obsahuje informácie o všetkých extension pointoch, ktorý daný plugin využíva. Ako prípona súboru prezrádza, jedná sa z pohľadu obsahu súbora o xml súbor. Tento súbor má vo vývojovóm prostredí Eclipse zvláštne postavenie oproti ostatným xml súborom, pretože je mu pre jeho zvláštny význam pri tvorbe pluginov vytvorený samostatný editor (viď.obr.2.3). Neskôr si uvedieme príklad použitia tohto špecializovaného editora.

Už sme si vysvetlili dôležitosť spomínaných dvoch súborov, avšak len samotná existencia týchto súborov v plugine nestačí, aby sa plugin zaradil medzi pluginy rozširujúce platformu Eclipse. Je ešte potrebné zadefinovať v plu-

gine trieda, cez ktorú bude Eclipse runtime (viď. 2.1) spúšťať a ukončovať plugin. Táto trieda musí byť potomkom triedy `org.eclipse.core.runtime.Plugin`. Dôležité sú metódy `start(BundleContext context)` a `stop(BundleContext context)`, ktoré sú volané na odštartovanie pluginu a jeho zastavenie. Ich preťaženie môže plniť rôzne inicializačné či ukončovacie funkcie. Musíme však mať na pamäti, že i keď Eclipse runtime spúšťa plugin až vtedy ak je to naozaj nutné, tak takáto nutnosť môže byť spôsobená napríklad i požiadavkou na získanie ikony daného pluginu a teda inicializácia prípadných veľkých dátových štruktúr daného pluginu v takomto prípade nie je nutná. Takéto prípady by sa mali pri inicializácií ošetriť.
V tejto chvíli už máme vytváranie pluginov dostatočne popísané z technického a pripájacieho hľadiska. Pozrime sa bližšie na tvorbu nových pluginov ako na proces rozširovania už existujúcich pluginov. Medzi prvé kroky pri implementácií pluginu patrí rozhodnutie, ktoré extension pointy chceme využiť. Kedže ideme rozširovať platformu Eclipse o komponenty, ktoré majú vnútri Eclipsu grafický charakter, tak musíme pripájať vytvárané pluginy na extension pointoch definovaných komponentom **Workbench User Interface**. Jeho základom je **workbench window** (implementácia interfacu `org.eclipse.ui.IWorkbenchWindow`), ktorá predstavuje celé grafické okno spusteného prostredia Eclipse. Workbench window obsahuje okrem iného i vlastné stránky (pages), z ktorých je aktívne vždy práve 1 stránka. Stránka predstavuje zoskupenie náhľadov (views) a editorov. Obrázok 2.4 znázorňuje čiastočný rozklad Workbench UI na menšie celky, ktoré definujú väčšinou už len 1 extension point.

Každý extension point pridáva ďalšie súčasti k celkovej architektúre Eclipsu a editor umožňuje ľahké pridávanie, odoberanie a editáciu týchto súčastí. Teraz si uvedieme príklad použitia editora súboru plugin.xml. Konkrétne pridávanie grafického komponentu sprievodca (wizard). Extension point org.eclipse.ui.newWizards pridáva sprievodcov a kategórie, do ktorých môžeme sprievodcov uložiť. Pomocou editora si pridáme 1 kategóriu a pár sprievodcov. Pre kategóriu nastavíme meno, pod ktorým ju chceme vidieť, a jednoznačné[1] ID (identifikačný znakový reťazec). Pre sprievodca sa tiež nastaví meno,ID a ešte navyše trieda, ktorá implementuje wizard (nutne musí implementovať rozhranie `org.eclipse.ui.INewWizard`,väčšinou potomok `org.eclipse.jface.wizard.Wizard`), meno kategórie, v ktorej sa na-

[1]Jednoznačné pre celý Eclipse,t.j. pre všetky pluginy (v prípade napojenia triedy ku súčasti sa použije celé meno triedy)



Obr. 2.4: Workbench Decomposition

chádza (t.j. kategória,ktorú sme si definovali), a ikona, ktorá sa bude zobrazovať pri vstupe do sprievodca. Týmto sme vytvorili nového sprievodcu pre gui našej aplikácie. Možností nastavenia sprievodca je samozrejme oveľa viac, ale pre jednoduchosť príkladu uvádzame len tie najzákladnejšie.

2.2 Popis rozšíriteľných grafických pluginov

V tejto podkapitole si bližšie popíšeme existujúce grafické možnosti platformy Eclipse, ktoré neskôr budeme môcť využiť pri návrhu matematického editora. Popíšeme si použiteľnosť a vlastnosti grafických pluginov a popíšeme si i možnosti ich ďalšieho rozšírenia. Príklad grafických komponentov platformy Eclipsu znázorňuje aj obr. 2.4.

2.2.1 Perspektíva

Platforma Eclipse umožňuje využívanie rôznych grafických prvkov. Používateľ si z nich môže vybrať, ktoré chce používať, a rozložiť ich na pracovnej ploche podľa svojich potrieb. Avšak častým javom je používateľova potreba použiť rôznu sadu nástrojov (v podobe grafických prvkov) v rôznych situáciách. Z

používateľského pohľadu je dosť náročné pri každej zmene potreby druhov a rozloženia nástrojov ručne ale hlavne jednotlivo tieto nástroje odstraňovať a pridávať nové. Kvôli takýmto situáciám existuje v platforme Eclipse perspektíva (Perspective). Rôzne perspektívy sú definované pomocou implementácie rozhrania `org.eclipse.ui.IPerspectiveFactory`. Úlohou takejto implementácie je zkonfigurovať implementácie rozhrania `org.eclipse.ui.IPageLayout`, ktorej funkcionalita spočíva v rozmiestnení jednotlivých grafických prvkov na jednej stránke grafického okna (workbench window) Eclipsu. Po vykonaní konfigurácie sa väčšinou referencia na implementáciu `IPerspectiveFactory` zahodí.
Možné použitie tohto vizuálneho prvku pri návrhu matematického editora spočíva v rozdelení vizuálnych prvkov editácie textu, t.j. vytvorenie samostatnej perspektívy pre editáciu matematického textu.
Nasledovný príklad ilustruje pripojenie novej perspektívy do súbora plugin.xml[2]. Konkrétne sa jedná o perspektívu zdrojov, ktorej definícia je ešte súčasťou platformy Eclipse.

```
<extension point="org.eclipse.ui.perspectives">
  <perspective
        name="%Perspective.resourcePerspective"
        icon="icons/full/cview16/resource_persp.png"
        class="org.eclipse.ui.internal.ResourcePerspective"
        id="org.eclipse.ui.resourcePerspective">
  </perspective>
</extension>
```

Atribút `id` predstavuje identifikátor perspektívy, ktorý musí byť jednoznačný. Atribút `class` predstavuje implementačnú triedu perspektívy.

2.2.2 Menu

Hlavné menu je v aplikáciách všeobecne považované za jeden z najdôležitejších ovládacích prvkov. Avšak v prípade položkovo veľkého menu sa prehľadnosť viditeľne znižuje. Tvorcovia dizajnu tohto vizuálneho prvku sa snažili vytvoriť dodatočné vizuálne pomôcky, aby zredukovali daný problém. Medzi takéto pomôcky patrí aj zoskupovanie položiek do skupín (väčšinou podľa

[2]Výpis príslušnej časti zo súbora plugin.xml či obrázok z gui editora tohto súboru sú ekvivalentné, a preto je irelevantné, ktoré z nich použijeme.

funkcie položiek). Podobné vlastnosti má aj menu implementované v platforme Eclipse.
Pridávanie prvkov do menu poskytuje extension point `org.eclipse.ui.actionSets`. Pomocou neho sa definuje tzv. actionSet, t.j. dátová štruktúra definujúca nové prvky určené pre hlavné menu na platforme Eclipse (prvky typu menu) ale aj nové prvky reprezentujúce akciu (prvky typu action). Nasleduje ilustračný príklad štruktúry takéhoto využitia extension pointu.

```
<extension point = "org.eclipse.ui.actionSets">
       <actionSet id="org_eclipse_ui_examples_readmetool_actionSet"
           label="%ActionSet.name"
           visible="true">
           <menu id="org_eclipse_ui_examples_readmetool"
               label="%ActionSet.menu"
               path="window/additions">
               <separator name="slot1"/>
               <separator name="slot2"/>
           </menu>
           <action id="org.eclipse.ui.examples.readmetool.readmeAction"
               menubarPath="window/org_eclipse_ui_examples_readmetool/slot1"
               label="%ReadmeAction.label"
               tooltip="%ReadmeAction.tooltip"
               icon="icons/ctool16/openbrwsr.png"
               class="org.eclipse.ui.examples.readmetool.WindowActionDelegate"
               definitionId="org.eclipse.ui.examples.readmetool.readmeAction"
               enablesFor="1">
               <selection class="org.eclipse.core.resources.IFile"
                    name="*.readme">
               </selection>
           </action>
       </actionSet>
   </extension>
```

Menu mechanizmus platformy Eclipse funguje nasledovne. Prvok typu menu vytvorí top level prvok (ako sú napr. „File“,„Search“ vo vývojovóm prostredí Eclipse) a definuje mu určité skupiny v určitom poradí. Tieto skupiny budú neskôr slúžiť prvkom typu akcia(action) na určenie ich cesty(path) v menu. Takto sa prvok typu akcia pomocou správne nastavenej cesty atribútom `menubarPath` zaradí do menu. Presnejšie povedané, zaradí sa na ko-

niec za všetky doteraz pridané prvky typu akcia v danej skupine vo vnútri menu. V prípade vyvolania tohto prvku akcie v menu sa presúva tok riadenia do triedy asociovanej ku prvku typu akcia, ktorá implementuje rozhranie `org.eclipse.ui.IWorkbenchWindowActionDelegate` (v určitých prípadoch `org.eclipse.ui.IWorkbenchWindowPulldownDelegate`). Konkrétne sa spustí vykonávanie metódy `run(IAction action)`, ktorá je súčasťou predka obidvoch rozhraní (`org.eclipse.ui.IActionDelegate`).
Okrem vytvorenie potrebného menu a jeho skupín, nastavenia atribútu `menubarPath` a `class` pre prvok akcie, treba ešte dodefinovať už nám známe atribúty (`id`, `icon`, `label` pre menu i akciu)
Prvok typu menu vytvorí vo všeobecnosti nový top level prvok v menu platformy Eclipse, avšak dá sa použiť aj na prepísanie už existujúceho top level prvku menu a práve táto možnosť nám umožňuje zbytočne neporušovať hierarchiu stromovitej štruktúry menu pre potreby pridania menej významných akcií. Predstavme si existujúce menu, ktoré sme nevytvorili, ale bolo už vytvorené iným pluginom, ktorý si zadefinoval vlastné skupiny vnútri tohto menu. Na existenciu tohto menu a teda aj na existenciu daných skupín sa však nedá vo všeobecnosti spoľahnúť, kedže pluginová štruktúra aplikácie sa môže meniť, a tak musíme vytvoriť presne také isté menu aké pozorujeme. Funkcie nášho pluginu ani pluginu pôvodne vytvárajúceho menu nebudú narušené, avšak musíme poznamenať, že takéto jednanie nie je vôbec štandartné. Neštandartnosť spočíva v tom, že sa spoliehame na fakt, že vyššie verzie cudzích pluginov budú naďalej používať menu v rovnakej forme (t.j. rovnaké skupiny) ako doteraz a tak je funkčnosť vyšších verzií cudzích pluginov ohrozená. V prípade projektu Euromath neexistujú cudzie pluginy, ktoré by prispievali do štruktúry menu a zároveň by hrozila ich zámena za aktuálnejšie vývojové verzie. Z tohto dôvodu by sme mohli vyššie popísaný postup v prípade potreby uplatniť.

2.2.3 Editor

Vizuálny prvok editora sa funkčne príliš nelíši od editora súborov, ktorý ponúka operačný systém. V podstate nič dôležité nebráni používateľovi použiť externý editor súborov ak sa jeho práca zameriava len na editáciu daného súbora. Rozdiel v použití editora platformy Eclipse spočíva v jeho miere prepojenosti s ostatnými časťami aplikácie. Prepojenosť umožní využívať rôzne špecifické funkcie aplikácie a to bude v konečnom dôsledku znamenať vysoký nárast komfortu a efektivity oproti editácií externým editorom.

Z pohľadu vývoja projektu Euromath by sa mohlo zdať skúmanie tohto vizuálneho prvku ako zbytočné vzhľadom na jeho existujúce zapojenie v projekte. Avšak z dôvodu vylepšovania editora v projekte je nutné, aby sme si objasnili celkové fungovanie tohto komponentu spolu s jeho celkovými možnosťami, ktoré pravdepodobne nie sú všetky využité v projekte Euromath.
Nový editor sa pripája pomocou extension pointu `org.eclipse.ui.editors`. Obsahovo je naviazaný na vstupný objekt (väčšinou súbor), ktorý je implementáciu rozhrania `org.eclipse.ui.IEditorInput`. Okrem očakávaných atribútov ako `id`, `name`, `icon`, `class` sa vyskytujú i nové atribúty (viď. príklad uvedený nižšie). Atribút `extensions` definuje príponu súborov, pre ktoré je daný editor určený. Zaujímavým atribútom je tiež `contributorClass`, ktorý určuje triedu poskytujúcu akcie orientované na editor. Takáto trieda musí implementovať rozhranie `org.eclipse.ui.IEditorActionBarContributor` a dokáže pridávať akcie do menu a nástrojovej lišty (toolbar).

```
<extension point = "org.eclipse.ui.editors">
    <editor
       id = "org.eclipse.ui.examples.readmetool.ReadmeEditor"
       name="%Editors.ReadmeEditor"
       icon="icons/obj16/editor.png"
       class="org.eclipse.ui.examples.readmetool.ReadmeEditor"
       extensions="readme"
       contributorClass="org.eclipse.ui.examples.readmetool.Readme-
                         EditorActionBarContributor">
    </editor>
</extension>
```

Trieda predstavujúca samotný editor musí byť implementáciou rozhrania `org.eclipse.ui.IEditorPart`, pričom sa väčšinou použije trieda `org.eclipse.ui.part.EditorPart` ako východiskový implementačný bod.

2.2.4 Náhľad

Náhľad (view) má predovšetkým pomocnú funkciu. Vo všeobecnosti dokáže zobrazovať ľubovoľné hierarchické dáta. Táto vlastnosť sa často využíva v spojení s editorom, pričom náhľad zobrazuje vlastnosti objektu, ktorý sa práve edituje v editore. Zmeny vo vlastnostiach v náhľade prebiehajú okamžite a automaticky pri editačnej zmene. Rovnaké použitie môžeme pozorovať

i v projekte Euromath popri WYSIWYG editácií vo forme stromu pre editovaný xml dokument.
Tvorba nových náhľadov spočíva v rozšírení pomocou extension pointu `org.eclipse.ui.views`. Tak ako už spomínané editory, tak i náhľad potrebuje svoju implementačnú triedu. Jedná sa o implementáciu rozhrania `org.eclipse.ui.IViewPart`, avšak môže sa použiť aj rozšírenie minimalistickej implementácie tohto rozhrania, triedy `org.eclipse.ui.part.ViewPart`. Na jednoduchom ilustračnom príklade si vysvetlíme ďalšie atribúty nutné k správnemu zkonfigurovaniu nového náhľadu.

```
<extension point="org.eclipse.ui.views">
    <category
       id="org.eclipse.ui.examples.readmetool"
       name="%Views.category">
    </category>
    <view
       id="org.eclipse.ui.examples.readmetool.views.SectionsView"
       name="%Views.ReadmeSections"
       icon="icons/view16/sections.png"
       category="org.eclipse.ui.examples.readmetool"
       class="org.eclipse.ui.examples.readmetool.ReadmeSectionsView">
    </view>
</extension>
```

Okrem už spomenutých atribútov či podobných atribútov z predošlých prípadov rozšírenia pomocou extension pointu, existuje ešte jeden neznámy atribút (`category`). Tento atribút určuje identifikátor(`id`) kategórie náhľadov, do ktorej by mal patriť novovytvorený náhľad. V príklade sme si pre ilustráciu jednu kategóriu vytvorili.

Kapitola 3

MathML

V projekte Euromath 2 sa za účelom zápisu matematického textu využíva formát MathML. Kvôli návrhu editora matematického textu si tento formát bližšie predstavíme. Budeme čerpať z oficiálnych zdrojov konzorcia W3C ([11],[12]).

3.1 Definície a história

V minulosti, po vzniku elektronickej komunikácie, vznikla potreba elektronickej výmeny matematického textu. Takáto komunikácia s obsahom matematického textu neprebiehala len medzi matematikmi, ale aj medzi inými vedcami, medzi učiteľmi a študentmi, a medzi mnohými ďalšími ľuďmi. Z tohto dôvodu sa vyvinul nie príliš praktický systém prenášania a publikovania matematických textov pomocou obrázkov. Takýto systém mal mnoho nevýhod. Tieto nevýhody by sa dali zoskupiť do dvoch skupín.

Zobrazovacie problémy , ktoré spočívajú v nesprávnom zobrazení matematického textu v inom prostredí (iný browser, iné grafické motívy operačného systému). Medzi takéto problémy patria problémy s farbou pozadia matematického textu, ktoré vznikajú pri odlišnej celkovej farbe pozadia v koncovej zobrazovacej aplikácií. Patrí sem i problém veľkosti riadka, ktorý môže byť na rôznych koncových systémoch rôzny, avšak matematický text na obrázkoch bol vytvorený podľa fixnej výšky a rozloženia riadku.

Kódovacie problémy spočívajú v ťažkej manipulovateľnosti s matematic-

kým textom v obrázkovej podobe. Požiadavky na vyhľadávanie, indexovanie, čítanie a editáciu matematického textu sú príliš ťažko realizovateľné. Bitová veľkosť obrázku je tiež príliš vysoká vzhľadom na prenášanú informáciu.

V skutočnosti sa s takýmto systémom publikácie matematického textu stretneme na internete i dnes, hlavne v prípadoch nevedeckých stránok. Avšak v roku 1994 sa vedci zhodli na potrebnosti nového systému/formátu na prenášanie matematického textu. Najprv sa diskutovalo o vhodnom rozšírení formátu HMTL 3.0, ktorý bol v danom čase vhodný formát publikovania textu na internete. Nakoniec sa však rozhodlo o vzniku úplne nového systému na zobrazovanie a zaznamenávanie matematického textu. Pomenovali ho Mathematical Markup Language, skrátene MathML.
Požiadavky na tento nový systém sa zameriavali jednak na odstraňovanie problémov s obrázkovými systémami prenášania matematického textu, ale i na iné vlastnosti vhodné pre budúci rozvoj MathML. Konkrétne sa jedná o nasledovné požiadavky:

- Schopnosť pojmúť matematický text vhodný pre učiteľskú a vedeckú komunikáciu na všetkých úrovniach.
- Schopnosť zachytiť jednak matematický význam ale aj matematickú notáciu daného matematického textu.
- Schopnosť konverzie z a do iných matematických formátov bez rozdielu medzi formátmi zameranými na sémantiku matematického textu a formátmi zameranými na prezentáciu matematického textu. Výstupné formáty, ktoré dostaneme konverziou z MathML, by mali taktiež obsahovať nasledovné typy formátov:
  - formáty pre grafické zobrazenie matematického textu
  - formáty pre syntetizátory reči
  - formáty pre vstup do algebraických počítačových systémov
  - iné matematické formáty (Tex a iné)
  - formáty bežného (nematematického) textu
  - formáty pre tlač (i pre tlač Braillového písma)
- Schopnosť preposlať informácie špecifické pre aplikáciu, ktorá s MathML pracuje.

- Podpora efektívneho prechodu cez dlhé matematické výrazy.
- Schopnosť jednoducho rozširovať funkcionalitu.
- Schopnosť jednoduchej vykonateľnosti editačných techník matematického textu a podpora šablón (templates).
- Schopnosť jednoduchej spracovateľnosti a generovateľnosti pre software, ktorý s ním bude primárne pracovať, ale taktiež schopnosť čitateľnosti pre ľudí, aby ho mohli ľudia ľahšie pochopiť a v prípade jednoduchých matematických textov i ručne vytvárať.

Bohužiaľ, hore uvedené požiadavky na MathML nezaručia jeho úspech. Problém spočíva v kvalite jeho implementácie tretími stranami. Preto je vhodne spomenúť i dodatočná implementačné ciele, ktoré však už priamo záležia na autoroch implementačného softwaru.

- Matematický text vo forme MathML vnútri HTML kódu by sa ma zobrazovať správne v najpoužívanejších internetových prehliadačoch a to vzhľadom ku čitateľovým zobrazovacím nastaveniam. Mala by sa tiež dosahovať maximálne kvalita zobrazenia aké ponúka daná platforma.
- Taktiež tlač HTML dokumentov s matematickým textov vo forme MathML by mala byť maximálnej kvality.
- Matematický text vo forme MathML by mal byť schopný reagovať na akcie používateľa a schopný koordinovať komunikáciu s ostatnými aplikáciami pomocou prehliadača tohto matematického textu.
- Vývoj editorov a konvertorov matematického textu vo forme MathML by mal smerovať k tomu, aby tieto nástroje napomáhali k vytváraniu internetových stránok s obsahom matematického textu vo forme MathML.

Pri celkovom zhrnutím požiadaviek, je MathML XML formát určený na popis matematickej notácie a na zachytenie štruktúry a obsahu matematického textu. Jeho primárnym cieľom je umožniť publikovať a manipulovať matematické texty na internete, tak ako to umožňuje HTML s bežným textom.

3.2 Design MathML

MathML ako značkovací(markup) jazyk si dalo za cieľ zachytiť prezentačnú a obsahovú stránku matematického textu. Prezentačná stránka zachycuje ako má byť matematický text zobrazený zatiaľ čo obsahová stránka zachycuje matematický text skôr ako súvislosti a vzťahy v abstraktnom matematickom svete.
V reálnom svete však často notácia, t.j. súčasť prezentačnej stránky matematického textu, vo veľkej miere ovplyvňuje naše vnímanie matematického textu. Notácia nás môže nasmerovať určitým smerom v oblasti chápania daného matematického textu. Tento fakt je vo veľkej miere využívaný v učiteľských kruhoch. Pre dizajnérov bol však problém zabezpečiť korektnú prácu s nepresnými matematickými tvrdeniami, i keď väčšina ľudí pochopí správny význam tvrdenia vďaka jeho notácií. Ako protipól k takémuto prípadu existuje matematický text, ktorý je použitý ako vstup pre algebraické výpočtové systémy, kde sa vôbec neberie v úvah notácia ale len obsahová stránka. Funkčne musí MathML zvládnuť obidva extremálne prípady a tak riešením bolo oddeliť zápis prezentačnej a obsahovej stránky úplne. Z toho vyplýva aj nasledujúce členenie MathML elementov:

Prezentačné elementy popisujú vizuálnu dvojdimenzionálnu štruktúru matematickej notácie. Príkladom takýchto elementov sú `mrow` a `msup`. Element `mrow` zabezpečuje horizontálny riadok pre elementy, ktoré sa v ňom bezprostredne nachádzajú. Element `msup` zas zabezpečuje vizuálny prvok horného indexu.

Obsahové elementy (content elements) popisujú obsahovú stránku matematického textu.

Elementy rozhrania (interface elements) zabezpečujú prepojenie MathML s editormi, konvertormi a iným softwarom, ktorý s MathML pracuje.

Každý prezentačný element korešponduje s určitou notačnou schémou, pričom rekurzívnou dekompozíciou sa dá každý matematický výraz rozložiť na základné notačné schémy ako napríklad riadok, horný index, dolný index a pod. Takéto notačné schémy sa samozrejme ešte dajú rozdeliť na tie najjednoduchšie prvky ako sú číslice, písmena a podobne. Pre obsahový popis matematického textu sa dá aplikovať podobná dekompozícia. Podobnosť takýchto dekompozícií nie je náhodná, pretože notácie a obsahový význam sú

často úzko prepojené.
MathML výrazy sa tiež dajú zo štrukturálneho hľadiska chápať ako stromy. Každý vrchol takéhoto stromu korešponduje s niakým MathML elementom. Listy stromu korešpondujú s atomickými notáciami (prezentačné listové elementy) alebo so základnými obsahovými elementami ako sú čísla, znaky a podobne. Existujú 3 druhy listov:

Prázdne kanonické elementy , ktoré nemajú žiadne telo. Typickým príkladom je element operácie `<plus/>`.

Anotačné elementy , ktorých význam spočíva v pripájaní dát v inej forme ako XML. Nasledovný príklad znázorňuje pripojenia matematického textu vo forme texu.

```
<annotation encoding="TeX">
    \sin x + 5
</annotation>
```

Token elementy sú najbežnejším druhom listu. Najčastejšie reprezentujú identifikátory, čísla alebo operácie. V prípade prezentačných tokenov sú to konkrétne elementy `mi` (identifikátor), `mn` (number) a `mo` (operation). V prípade obsahových tokenov sa jedná o elementy `ci`, `cn` a `csymbol` (zadefinovanie nového symbolu).

Väčšina MathML elementov definuje svoj obsah pomocou svojho začiatočného a ukončovacieho tagu, pričom môžu obsahovať aj iné elementy. V prípade tokenov sa medzi začiatočným a ukončovacím tagom môže nachádzať iba reťazec charakterov, ktorý je považovaný za obsah tokena. V prípade prázdnych kanonických elementov sa nevyžaduje žiadny obsah, pretože existencia samotného kanonického elementu poskytuje dostatočnú informáciu o liste stromu.

3.3 Prezentačné a obsahové elementy

3.3.1 Prezentačné elementy

Tak ako už bolo vyššie uvedené, prezentačné elementy popisujú vizuálnu dvojdimenzionálnu štruktúru matematickej notácie. Okrem už spomenutých prezentačných tokenov, existujú aj prezentačné elementy využívajúce sa na

zarovnávanie textu. Dokopy existuje asi 30 prezentačných elementov a tie sú schopné akceptovať ďalších 50 nastaviteľných atribútov.
Prezentačné elementy korešpondujú väčšinou s rozmiestňovacími (layout) schémami. Tie sa dajú rozdeliť do viacerých skupín podľa zamerania. Prvá skupina sa zameriava na skripty (`msub`, `munder`, `mmultiscripts`), druhá skôr na všeobecne rozmiestnenie elementov (`mrow`, `mstyle`, `mfrac`) a posledná špeciálne na tabuľky. Rozmiestňovacie schémy sú špecifické z pohľadu poradia schém ich potomkov. V iných schémach nezáleží na poradí ich potomkov, avšak v rozmiestňovacích schémach je to dôležité. Dobrým príkladom je schéma elementu `frac`, ktorá predpokladá ako schému prvého potomka schému čitateľa až potom schému menovateľa. Takáto vlastnosť rozmiestňovacích schém sa nedá zachytiť(vynútiť) na úrovni spracovania XML dokumentu skrz DTD6.1. Z tohto dôvodu jasne vyplýva, že sa MathML nedá vždy spracovávať generickým XML procesorom a je tu potreba špeciálneho procesora pre MathML.
Nasleduje jednoduchý ilustračný príklad použitia prezentačných elementov na zobrazenie matematického textu $(a + b)^2$.

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

3.3.2 Obsahové elementy

Obsahové elementy majú za úlohu zaznamenávať obsahovú stránku matematického textu. Takýchto elementov je asi 120, pričom nastaviteľných atribútov je asi 12. Väčšinou sa jedná o prázdne elementy korešpodujúce k množstvu rôznych operátorov, relácií a pomenovaných funkcií. Príkladom môže byť element `leq` (relácia $\leq$) či `tan` (funkcia tangens). Ďalšou podskupinou obsa-

hových elementov sú elementy zaznamenávajúce matematické dátové typy. Príkladom je napríklad element `matrix` a `set`. Poslednú, avšak nemenej dôležitú, podskupinu tvoria elementy schopné vytvoriť nové matematické objekty či aplikovať (element `apply`) operácie na matematické výrazy, ktoré sú pri takomto použití chápané ako argumenty pre danú operáciu.
Spomenutý element `apply` je spomedzi obsahových elementov jeden z najdôležitejších. Elementy, ktorý sú jeho potomkovia, predstavujú jednu operáciu aj s potrebnými argumentmi. Poradie potomkov je tiež dôležité, pretože sa predpokladá prefixový zápis operácie, t.j. prvý potomok je element operácie, druhý potomok je element prvého argumentu, tretí potomok je element druhého argumentu a tak podobne.
Nasledujúci príklad demonštruje takúto konštrukciu. Jedná sa o jednoduchú operáciu odčítania identifikátora b od identifikátora a.

```
<mrow>
  <apply>
   <minus/>
   <ci>a</ci>
   <ci>b</ci>
  </apply>
</mrow>
```

Existujú však aj operácie, ktoré okrem zadefinovania argumentov požadujú aj zadefinovanie kvalifikátorov. Príkladom takejto operácie je integrácia (element `int`) alebo derivácia (element `diff`). Nasledovný príklad znázorňuje jednoduchú operáciu integrácie $\int_a^b f(x)dx$. Pomocou elementu `bvar` sa vo všeobecnosti definuje viazaná premenná (Bound VARiable) a hornú a dolnú hranicu integrálu definujeme cez elementy `uplimit` a `lowlimit`.

```
<apply>
  <int/>
  <bvar><ci> x </ci></bvar>
  <lowlimit><ci> a </ci></lowlimit>
  <uplimit><ci> b </ci></uplimit>
  <apply>
     <ci type="function"> f </ci>
     <ci> x </ci>
  </apply>
</apply>
```

3.3.3 Spojenie prezentačných a obsahových elementov

Prezentačné a obsahové elementy sa od seba funkčné dosť odlišujú. Prezentačné elementy sú určené na zachytávanie notácie matematického textu. Takáto funkčnosť sa využíva hlavne pri spracovávaní starších matematických textov, ktoré boli zaznamenávané prevažne notačným spôsobom. V dnešnej dobe však existuje potreba sémantickej spracovateľnosti matematického textu, ktorá jednoznačne podmieňuje zápis matematického textu vo forme obsahových elementov. Dôsledkom toho mnoho dnešných aplikácií musí kombinovať obidve formy zápisu matematického textu, aby spĺňala svoju funkcionalitu.
Existujú rôzne spôsoby spájania elementov. V prípade zapájania obsahových elementov do prostredia prezentačných elementov, by sa mali zapájať takým spôsobom, aby sa z okolitých presentačných elementov dali určité fakty pre obsahové elementy vyčítať ihneď a nemuseli sa dodatočne a zbytočné dodefinovávať v obsahových elementoch. V prípade zapájania prezentačných elementov do prostredia obsahových elementov by sa zas malo predovšetkým jednať iba o notačné doplnenia obsahových tokenov. Existuje avšak aj iné riešenie prepojenia elementov. Jedná sa o úplné oddelenie obsahovej a prezentačnej časti, pričom obidve popisujú totožný matematický text. Takúto konštrukciu ponúka element `semantics`. Vo všeobecnosti úlohou tohto elementu je asociovať obsahovému popisu matematického textu (element prvého potomka) rôzne reprezentácie (elementy ďalších potomkov v poradí). V prípade definovania reprezentácie v MathML (MathML-Presentation) sa vykreslovací mechanizmus môže rozhodnúť pre takto definovanú reprezentáciu namiesto implicitne získanej podľa obsahového popisu. Nasleduje jednoduchý príklad použitia elementu `semantics`.

```
<semantics>
  <apply>
    <plus/>
    <apply>
      <sin/>
      <ci> x </ci>
    </apply>
    <cn> 5 </cn>
  </apply>
  <annotation-xml encoding="MathML-Presentation">
    ...
```

```
    ...
  </annotation-xml>
  <annotation encoding="Maple">
    sin(x) + 5
  </annotation>
  <annotation encoding="TeX">
    \sin x + 5
  </annotation>
</semantics>
```

Kapitola 4

Zdokumentovanie implementačnej stránky projektu Euromathu

V predošlých kapitolách sme sa venovali rôznym témam. Analyzovali sme dizajnové myšlienky Eclipsu. Hovorili sme si o princípoch projektu Euromath 2, pričom sme nevynechali ani zaujímavé problémy, ktoré s jeho vývojom súviseli. Tieto témy slúžili nielen ako potrebný analytický materiál pre návrh grafického rozhrania, ale slúžili aj ako oporný bod pre budúcich vývojárov projektu Euromath 2. V tejto kapitole postúpime týmto smerom ďalej a ponúkneme stručnú analýzu zdrojového kódu projektu. Najprv sa zameriame na pluginovú štruktúru všeobecne a neskôr i na konkrétne časti, ktoré by sme mohli tiež využiť pri návrhu grafického rozhrania pre matematický editor.

4.1 Všeobecná analýza

Všeobecná analýza by mala predovšetkým zabezpečiť všeobecnú znalosť zdrojového kódu. Mali by sme získať prehľad o jednotlivých pluginoch, o ich obsahu a funkčnosti. Z dôvodu prehľadnosti sa zameriame na jednotlivé pluginy postupne.

General je plugin bez funkčného zdrojového kódu. Obsahuje rôzne dokumenty a materiály k projektu.

sk.baka.eclipse.plugin-parent-pom je plugin obsahujúci konfiguračný sú-

bor pre buildovací nástroj Maven, pomocou ktorého sa vytvárajú tzv. buildy celého projektu.

sk.baka.ikslibs je plugin obsahujúci podprojekt ikslibs. Jedná sa o rozšírenú knižnicu podporujúcu prácu s xml dokumentami.

sk.baka.xml.gene je základným pluginom pre podprojekt GENE. Obsahuje celú funkcionalitu podprojektu okrem niektorej špeciálnej funkcionality, ktorá je obsiahnutá v niekoľkých nasledujúcich pluginoch.

sk.baka.xml.gene-docbook je súčasťou podprojektu GENE a obsahuje exportéry pracujúce s formátom DocBook.

sk.baka.xml.gene-mathml je súčasťou podprojektu GENE a obsahuje exportéry pracujúce s formátom MathML. Obsahuje tiež knižničnú referenciu na open source projekt JEuclid, ktorý zabezpečuje vykreslovanie matematického textu vo formáte MathML.

sk.baka.xml.gene-pom je plugin s podobnou funkciou ako plugin sk.baka.-eclipse.plugin-parent-pom, avšak konfiguračne slúži len podprojektu GENE.

sk.baka.xml.schematic je plugin obsahujúci celú funkcionalitu podprojektu Schematic.

sk.uniba.euromath je plugin obsahujúci samotné jadro projektu Euromath 2. Funkčnosť pluginu zahŕňa konfigurovanie projektu pomocou konfiguračných súborov, pripojenie všetkých grafických prvkov projektu ku platforme Eclipse, podporu grafických prvkov (listeners, akcie,. . .), rôzne podporné knižnice a iné potrebné triedy a rozhrania, ktoré sa ideologicky nepodarilo zaradiť do iných pluginov.

sk.uniba.euromath.docbook je gui plugin, ktorý slúži len na zaobalenie funkcionality spojenej s formátom DocBook pre potreby prezentácie v grafickom rozhraní.

sk.uniba.euromath.fop je plugin obsahujúci podporu menného priestoru XSL-FO. V projekte Euromath je tento formát využitý na zobrazovanie formátovaného textu. Plugin obsahuje vizuálny editor, exportér, vykresľovací mechanizmus a rôzne podporné nástroje.

sk.uniba.euromath.mathml je plugin podporujúci formát MathML. Obsahuje prázdnu implementáciu vizuálneho editora, prepojenie na vykresľovací mechanizmus projektu JEuclid, preprocesor pre MathML a podporné dátové štruktúry.

sk.uniba.euromath.pom je plugin rozširujúci funkcionalitu pluginu sk.baka.eclipse.plugin-parent-pom.

sk.uniba.euromath.rcp je plugin určený na build RCP6.1 aplikácií.

sk.uniba.euromath.xhtml je plugin plniaci podobnú funkciu ako plugin sk.uniba.euromath.docbook, avšak pre formát XHTML.

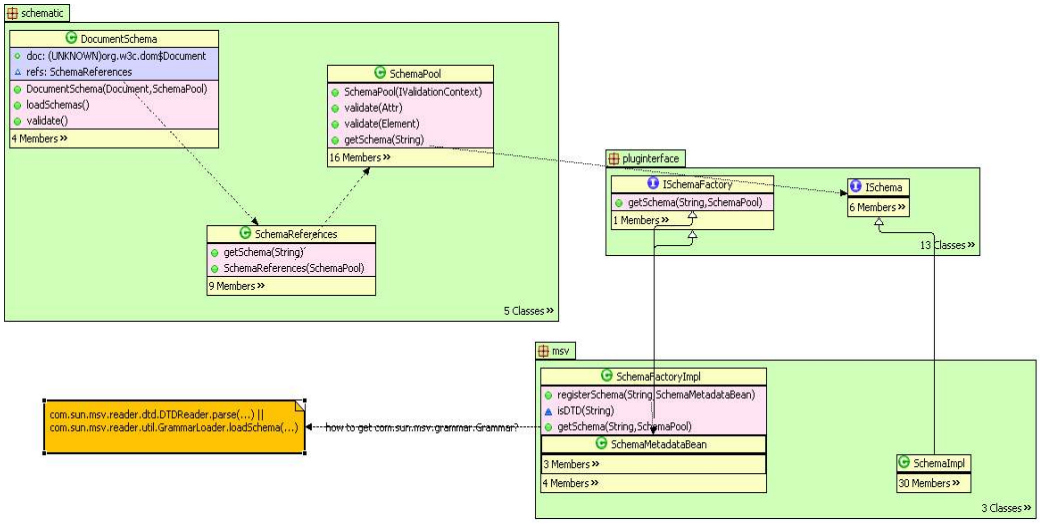
test je plugin obsahujúci len testovacie a ukážkové súbory editácie editorom projektu Euromath.

4.2 Podrobnejšia analýza vybraných častí

V tejto časti zameriame svoju pozornosť na 2 podprojekty projektu Euromath 2 (Schematic a Gene) a zanalyzujeme implementáciu všeobecného editora xml dokumentov, z ktorého sa dá neskôr vychádzať pri implementácií špecifickejších editorov.

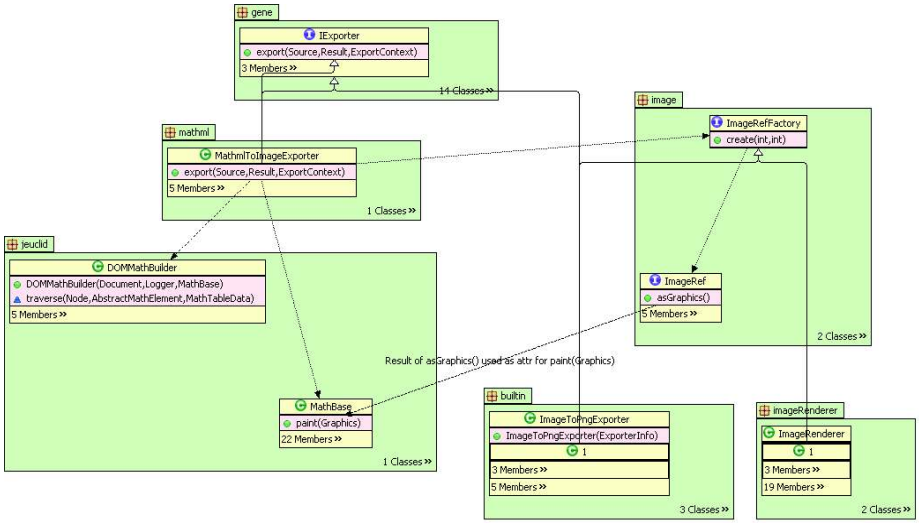
4.2.1 Schematic a trieda DocumentSchema

Základným kameňom projektu Euromath je časť starajúca sa o model (xml dokumenty a ich objektové reprezentácie). Táto časť je nazvaná schematic a jej hlavnou triedou je trieda `sk.baka.xml.schematic.SchemaPool` (viď. obr. 4.1). Funkciou schematic nie je len poskytnutie modelu ako takého, ale i poskytnutie množiny operácií nad ňou či poskytnutie funkcionality validácie dokumentu podľa schémy. Všetky schémy, t.j. triedy implementujúce rozhranie `sk.baka.xml.schematic.pluginterface.ISchema`, sú uložené v triede `SchemaPool`. Táto trieda poskytuje okrem validačných metód (`validate(Attr)`, `validate(Element)`) i jednotlivé schémy načítané v systéme. Získame ich podľa ich mena pomocou metódy `getSchema(String)`.
Ďalšou dôležitou triedou je trieda `sk.baka.xml.schematic.DocumentSchema`, ktorá spája xml dokument a jej schému. Pomocou metódy `loadSchemas()` získa od triedy `SchemaPool` schémy k dispozícií a zapamätá si ich v premennej



Obr. 4.1: DocumentSchema

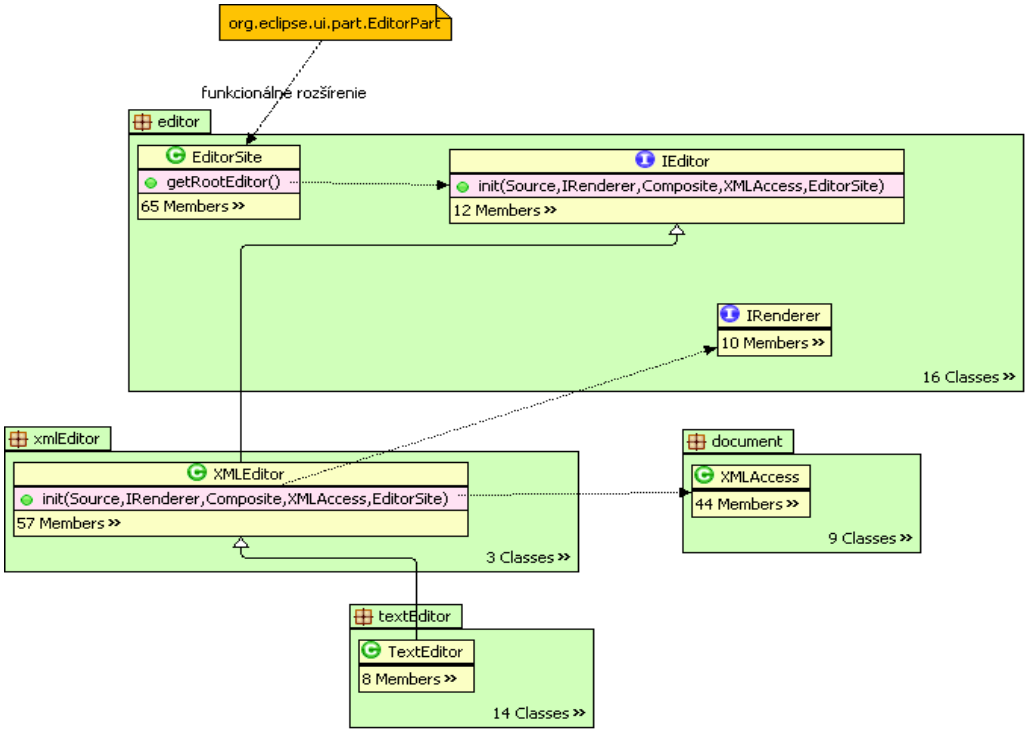
`refs`. Na základe týchto schém a dokumentu potom vie poskytnúť operácie na dokumente zachovávajúce valídnosť dokumentu. Už sme si síce povedali ,že schémy sú inštancie rozhrania `ISchema`, avšak všetky inštancie spĺňajúce túto podmienku sú vždy len inštancie triedy `sk.baka.xml.schematic.msv.SchemaImpl`. Nové schémy sa pridávajú do systému (resp. načítavajú sa) napríklad z DTD súborov pomocou inštancie triedy `sk.baka.xml.schematic.msv.SchemaFactoryImpl` (implementácia rozhrania `sk.baka.xml.schematic.plugin-terface.ISchemaFactory`). Konkrétne sa používa jej metóda `getSchema(String, SchemaPool)`, ktorá načíta súbor s gramatikou (schémou) pomocou metódy `com.sun.msv.reader.dtd.DTDReader.parse(...)` alebo pomocou metódy `com.sun.msv.reader.util.GrammarLoader.loadSchema(...)`. Načítaná schéma sa dá aj zaregistrovať medzi už známe menné priestory pomocou metódy `sk.baka.xml.schematic.msv.SchemaFactoryImpl.register-Schema(String,SchemaMetadataBean)`. Dá sa dokonca zistiť aj dodatočná informácia či daná schéma pochádza z DTD súboru a to pomocou metódy `sk.baka.xml.schematic.msv.SchemaFactoryImpl.isDTD(String)`.



Obr. 4.2: MathMLToImageExporter

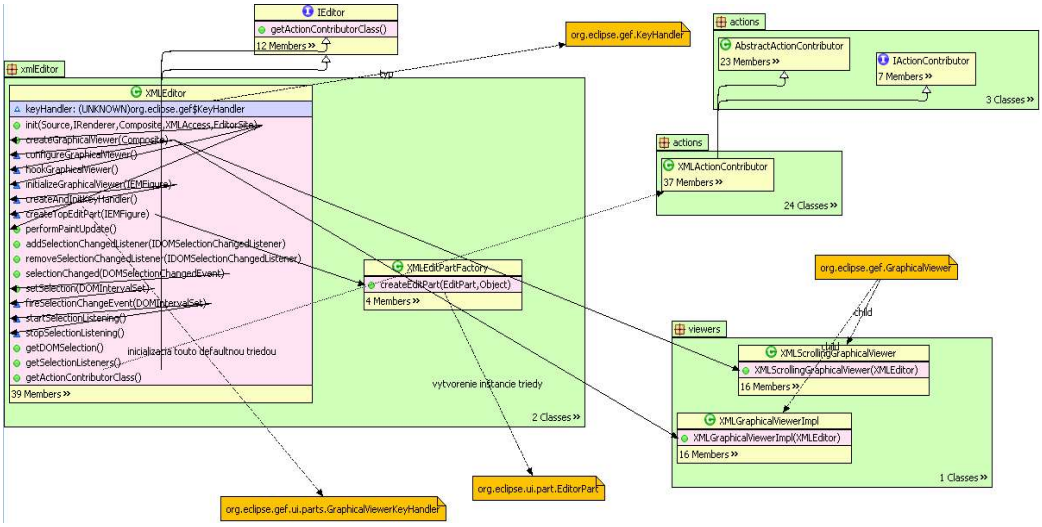
4.2.2 Gene a trieda MathMLToImageExporter

Pozrime sa bližšie na časť projektu Euromath nazvanú GENE. Úlohou tejto časti projektu je konvertovať časti XML dokumentu na zobraziteľné formáty, pričom túto konverziu zabezpečujú tzv. Exportéry. Jedná sa o triedy implementujúce rozhranie `IExporter`. Konkrétne pre menný priestor MathML je to formát SVG (6.1) a exportér triedy `MathmlToImageExporter`(viď. obr. 4.2). Zameriame sa hlavne na túto triedu z dôvodu jej veľkého významu pre editáciu matematického textu. Najdôležitejšou metódou exportéra je metóda `export(Source,Result,ExportContext)`, ktorá spracováva xml zdroj typu `javax.xml.transform.Source` a vyexportuje ho v podobe objektu typu `sk.baka.xml.gene.ExportContext`. Pozrime sa bližšie na tento export. Najprv sa xml zdroj zpracuje metódou `sk.baka.ikslibs.DOMUtils.toDocument(Node)`. Získame objekt typu `org.w3c.dom.Document` a ten už môžeme použiť ako parameter pre vytvorenie triedy `net.sourceforge.jeuclid.DOMMathBuilder`, ktorá je už súčasťou iného open source projektu tretej strany. Tento projekt má názov JEuclid a je zameraný predovšetkým na zobrazovanie(vykreslovanie) menného priestoru MathML. Trieda `DOMMathBuilder` poskytne inštancií triedy `net.sourceforge.jeuclid.MathBase` potrebné informácie, aby mohla táto trieda vykresliť matematický text ako obrázok.



Obr. 4.3: EditorSite

`MathBase` má na tento účel metódu `paint(java.awt.Graphics2D)`, pričom parameter typu Graphics2D je použité ako plátno(canvas), na ktoré sa kreslí. Musíme teda získať ešte inštanciu plátna. To získame vytvorením obrázka (resp. referencie na obrázok), ktorý bude typu `sk.baka.xml.gene.image.ImageRef`, pretože obrázok poslúži ako plátno pri použití metódy `asGraphics()`. A nakoniec vytvorenie referencie na nový obrázok vytvoríme pomocou triedy `sk.baka.xml.gene.image.ImageRefFactory`. Ďalšími triedami stojacími za zmienku je i trieda `sk.baka.xml.gene.builtin.ImageToPngExporter`, ktorá už podľa názvu exportuje obrázky typu `Image` na obrázky formátu png, či trieda `sk.uniba.euromath.editor.imageRenderer.ImageRenderer` slúžiaca na vykreslovanie priamo z xml zdroja.



Obr. 4.4: XMLEditor

4.2.3 Trieda EditorSite

Rozšírením triedy `org.eclipse.ui.part.EditorPart`, ktorá predstavuje základný editor v platforme Eclipse, si vytvoríme vlastný editor. Za účelom docielenia požadovanej funkcionality musíme ešte túto funkcionalitu doimplementovať. V prípade projektu Euromath sa mierne rozšírený základ triedy EditorPart implementoval v podobe triedy EditorSite (viď.obr.4.3).
EditorSite je v skutočnosti kontajnerom pre inštancie implementácií rozhrania `IEditor`. Tieto implementácie majú na starosť súvislú časť dokumentu, ktorá spadá pod rovnaký menný priestor. Takýmito implementáciami sú `XMLEditor` a jeho potomok `TextEditor`. Obidve implementácie sa inicializujú pomocou metódy `init(Source, IRenderer, Composite, XMLAccess, EditorSite)`. Výsledok poskytne podprojekt GENE. Bude typu `Source` a zároveň je už súčasťou renderera (`IRenderer`), ktorý pomocou inštancie triedy `XMLAccess` je schopný aj tento zdroj upravovať. Inicializácia by mala získať z renderera všetky `IFigure` inštancie a zobraziť ich na danom komponente.

4.2.4 Trieda XMLEditor

Zamerajme sa na triedu `XMLEditor` a inicializáciu v nej(viď.obr.4.4). V prípade tejto triedy prebieha inicializácia pomocou troch postupne volaných

metód:

1. Metóda `createGraphicalViewer(Composite)`, ktorá najprv vytvorí inštanciu triedy `org.eclipse.gef.GraphicalViewer`. Konkrétne sa bude jednať o inštancia triedy `XMLScrollingGraphicalViewer` (alebo triedy `XMLGraphicalViewerImpl`). Následne sa nakonfiguruje prehliadač(viewer) (metóda `configureGraphicalViewer()`) a nakoniec sa pripoja do systému listeners (metóda `hookGraphicalViewer()`), aby dokázal prehliadač reagovať na udalosti typu selection.

2. Metóda `initializeGraphicalViewer(IEMFigure)` nastaví prehliadaču `KeyHandler`, aby reagoval na udalosti vyvolané klávesnicou, ale predovšetkým sa v tejto metóde nastaví prehliadaču obsah (`Content`). Obsah sa získa pomocou metódy `createTopEditPart(IEMFigure)`, resp. sa použije factory na tvorbu inštancie triedy `EditPart` z koreňového figure(`IEMFigure`) stromu, ktorý má `EditPart` predstavovať.

3. Metóda `performPaintUpdate()` už len prekreslí celý prehliadač.

`XMLEditor` je najvšeobecnejšia implementácia rozhrania `IEditor`. Je založený na frameworku grafickéj editácie (GEF6.1) a obsahuje funkcionalitu označovania (selekcie) elementov, editácie pomocou klávesnice a cez menu, clipboard, undo/redo a zoom. Pozrime sa na jeho metódy bližšie a analyzujme jednotlivé funkcie tejto triedy.

- Preťažené metódy `addSelectionChangedListener(IDOMSelection-Changed Listener), removeSelectionChangedListener(IDOMSelectionChangedListener)` a `selectionChanged(DOMSelectionChanged-Event)` umožňujú pridávať, odoberať listener objekty a preposlať udalosti označenia (`IXMLSelection`) ku zodpovedným listener objektom. Posledná spomenutá akcia sa vykonáva pomocou vnorených volaní metód `selectionChanged`, `setSelection` a `fireSelectionChangeEvent`. Posledná metóda využíva zastavenie a spustenie počúvania listener objektov pomocou metód `stopSelectionListening()` a `startSelection-Listening()`. Takéto prerušenie ich činnosti je nutné z dôvodu možnosti nekonečného cyklu pri pridávaní novej udalosti. Získať aktuálne označenie je možné pomocou metódy `getDOMSelection()`, ktorá vyvolá rovnomennú metódu z inštancie triedy `EditorSite` spojenú s týmto editorom. Aktuálny zoznam listenerov možno získať pomocou metódy

`getSelectionListeners()`. Pomocou týchto metód je celkovo zabezpečená správa udalostí označenia.

- Platforma Eclipse pracuje s globálnymi lištami (menu, status line, tool bar alebo cool bar) ako s kontajnerom pre inštancie typu akcia. Trieda `XMLActionContributor` pridáva akcie do týchto líšt. Táto trieda sa dá získať pomocou metódy `getActionContributorClass()`, ktorá je preťaženou metódou rozhrania `IEditor`. Rozhraním, ktoré táto trieda implementuje, je napríklad `IActionContributor`, ktorý je základným rozhraním pre všetky `ActionContributor` triedy. Tiež implementuje rozhranie `AbstractActionContributor`, ktoré implementuje základne operácie pre `ActionContibutor` triedy.

- Trieda `XMLEditor` obsahuje aj premennú typu `org.eclipse.gef.KeyHandler`, ktorej názov predznamenával funkciu spravovania všetkých udalostí klávesnice ohľadom daného editora (resp. `EditPartViewer`). V skutočnosti však táto premenná spravuje len udalosti klávesnice, ktoré nie sú súčasťou akceleračného navigačného systému celého menu. Daný keyHandler sa inicializuje pomocou metódy `createAndInitKeyHandler()`, pričom sa inicializuje len implicitnou triedou z GEF-u (`org.eclipse.gef.ui.parts.GraphicalViewerKeyHandler`).

Kapitola 5

Grafické rozhranie editácie matematického textu

V tejto kapitole sa zameriame na tvorbu grafického rozhrania pre editáciu matematického textu. Kedže projekt Euromath obsahuje podporu editácie xml dokumentu v rozsahu pridávania, odoberania a zmeny xml elementov, tak je zabezpečená i editácia menného priestoru MathML. Problém je vo všeobecnosti takejto editácie. Mnohé matematicky atomické elementy sú totiž vo forme XML zapísané pomocou viacerých elementov, ktoré sú atomické pre formát XML. Z tohto pohľadu nevytvárame len rozmiestnenie grafických komponentov spúšťajúce už existujúce operácie, avšak vytvárame aj špecifický editor tohto menného priestoru. V tomto prípade je špecifickým editorom už zabudovaný editor xml dokumentu, ktorý obsahuje navyše určité makrá existujúcich príkazov predstavujúce editačné príkazy s matematicky atomickými elementami.
V tejto kapitole si nebudeme popisovať technické detaily Eclipsu či menného priestoru MathML. Za týmto účelom boli vytvorené predošlé analytické kapitoly. Do tejto kategórie patrí aj tvorba makier existujúcich príkazov. Zameriame sa skôr na myšlienkový sled tvorby grafického rozhrania. Popíšeme si rôzne možnosti rozhraní a ich analýzou sa dopracujeme ku konečnému návrhu grafického rozhrania.
Prvotné myšlienky tvorby rozhrania spočívali v analýze odpovedí na základne zobrazovacie otázky. Existovali rôzne možnosti zobrazenia celého editora vzhľadom na rozloženie grafického rozhrania aplikácie projektu Euromath. Jednou možnosťou bolo presunúť funkčné prvky editácie či celý editor matematického textu do samostatného okna (dialog), ktorý by sa vyskytoval

mimo hlavného okna aplikácie. Aplikácia by nadobudla viacoknovú charakteristiku. Pozitívum takéhoto kroku by spočívalo v lepšej nastaviteľnosti celkovej aplikácie na úrovni jej okien. Negatívum zas spočíva v trende rozširovania projektu. V budúcnosti by sa totiž mohli ďalšie rozšírenia projektu uberať rovnakou cestou pridávania ďalších dialógov, čo spôsobí ťažšiu celkovú prácu s prispôsobovaním aplikácie hlavne kvôli reorganizácií dodatočných dialógov. Dôsledkom tohto faktu je preferencia alternatívy jednooknovej aplikácie[1].
Ďalšie rozhodnutie spočívalo vo výbere spôsobu zobrazovania dodatočných kontrolných komponentov (tlačidlá vkladania matematicky atomických elementov, editácia hodnôt listových tokenov v mennom priestore MathML). Na jednej strane bola možnosť vlastnej správy vykreslovania pomocou plátna (canvas) a na strane druhej použitie grafických komponentov platformy Eclipse. Na prvý pohľad je zrejmá možnosť využitia existujúcich grafických komponentov platformy Eclipse, ktoré uľahčia implementáciu z dôvodu nutnej implementácie určitého grafického výzoru a správania sa v prípade vlastných grafických prvkov. Dôvodom zváženia i inej alternatívy je zámer použitia grafických komponentov platformy Eclipse v rôznych častiach grafického rozhrania platformy Eclipse (viď. obr. Workbench decomposition2.4 v kapitole o platforme Eclipse). Problém v tomto prípade nenastáva pri rozložení či zakomponovaní týchto komponentov do platformy Eclipse, ale pri možnej budúcej snahe o zmenu týchto komponentov. Môže vzniknúť potreba na možnosť aktívnej konfigurácie pozície a grafického zobrazenia týchto komponentov zo strany používateľa. V takom prípade je nutné aktívne pluginy s danými konfigurovateľnými grafickými komponentmi pozastaviť. Takáto akcia je síce z teoretickej pohľadu na pluginovú architektúru platformy Eclipse možná, avšak úspešná deaktivácia a reaktivácia pluginu nemusí byť možná. Jednotlivé pluginy môžu totiž medzi sebou závisieť. V konečnom dôsledku môže takéto prepojenie spôsobiť funkcionálne problémy celej aplikácie, pričom to vyústi do reštartu aplikácie. Snaha o zistenie množiny vzájomne závislých pluginov by mohla byť taktiež neúspešná z dôvodu vnútroplatformových procesov, ktoré sa nedajú preskúmať. Výsledkom je teda reštart aplikácie. Z tohto dôvodu sa ponúka možnosť meniť vlastné konfiguračné súbory inštalovaného pluginu v jeho neaktívnom stave pomocou externej aplikácie vyrobenej pre tento účel. Tento krok sa nám zdá príliš nesystémový vzhľadom na neštandartný prístup do už nainštalovaných pluginov.

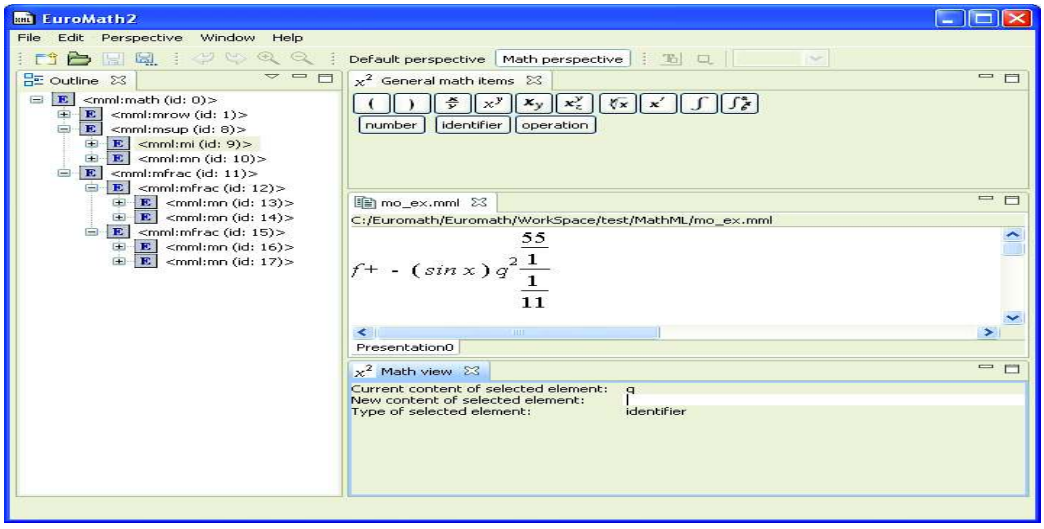
[1]Výnimkou sú rôzne dočasné okná pre potreby výberu súboru, potvrdzovania požiadaviek a podobne.

Výsledkom procesu analýzy možností by teda malo byť použitie plátna na vykreslovanie grafických komponentov, avšak daný spôsob nás núti implementovať grafické komponenty, ktoré sú už vytvorené. Takýto krok je taktiež nesystémový. Existuje však i ďalšia alternatíva. Jedná sa o modifikáciu metódy použitia grafických komponentov platformy Eclipse. Modifikácia spočíva v obmedzení umiestnenia grafických komponentov v rámci aplikácie. Za normálnych okolností by sa dali grafické komponenty pripojiť vo forme položiek hlavného menu, tlačidiel pomocnej lišty (toolbar) či zakomponovať do náhľadov(views). Obmedzenie zúži tieto oblasti iba na náhľady. Takéto obmedzenie sa môže zdať nepostačujúce pre potreby zobrazenia grafického rozhrania, avšak pôvodná funkcia náhľadov, zobrazovať hierarchické štruktúry či zoznamy, sa dá jednoducho modifikovať. Náhľad vo svojej podstate ponúka možnosť vložiť do seba ľubovoľný grafický komponent platformy Eclipse (panel, tlačidlá, zoznam, viacriadkový editor textu, štítok s textom a iné). Takýmto spôsobom sa dá v kóde programu dynamicky vytvoriť požadovaná štruktúra grafických komponentov. Takéto riešenie má aj svoje nevýhody. Pri prekonfigurovaní takýchto komponentov sa musí reinicializovať náhľad a táto operácia nemusí byť ľahko dosiahnuteľná vzhľadom na fakt, že náhľad sa v platforme Eclipse inicializuje zväčša iba raz a to vtedy keď tento komponent prvý krát potrebujeme. Avšak v najhoršom prípade požiadame používateľa o reštart celej aplikácie, ktorá problém vyrieši. Oproti predošlému riešeniu nemusíme nesystematicky meniť vlastné konfiguračné súbory pri deaktivovanom plugine, stačí zmeniť konfiguračné súbory počas behu pluginu v aplikácií a použiť zmenenú konfiguráciu pri ďalšom štarte aplikácie na dynamickú tvorbu komponentov vnútri náhľadu.

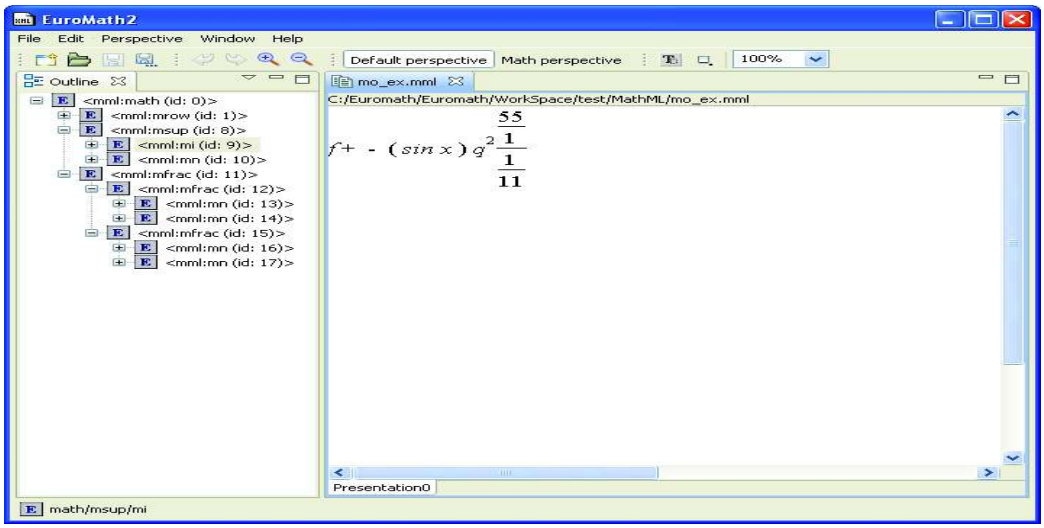
Postupnou analýzou sme dospeli do štádia, v ktorom sme schopný zobraziť tlačidlá (náhľady s grafickými komponentmi tlačidiel), ktoré budú vkladať za označený element matematického textu ďalšie matematicky atomické elementy. Pre prípad akcie odstránenia označeného elementu asociujeme klávesovú skratku s daným makrom odstraňovania elementu. Problémom však ešte zostáva editácia obsahu listov (prezentačné tokeny štandartu MathML) matematického stromu editovaného textu.

Listy matematického stromu sa zapisuje v štandarte MathML pomocou textových XML elementov. Editácia listov je teda problémom editácie textu (String). Riešení daného problému existuje viacero. Najvizuálnejšie orientovaným je editácia priamo v grafickom komponente editora. Toto riešenie nie je však najjednoduchšie vzhľadom na zložitý proces vykreslovania matematického textu, ktorý by sa musel upravovať. Konkrétne by sa musel

netriviálne upraviť open source software tretej strany, JEuclid. Z tohto dôvodu pripadajú v úvah skôr iné riešenia. Ďalšou možnosťou je v prípade označenia listu v editore vyvolať pri jeho pozícií dialógové okno s editovacím riadkom, kde vykonáme požadovanú editáciu. Takáto realizácia je už v porovnaní s predošlým riešením celkom jednoduchá. Existuje ale i riešenie podobnej jednoduchosti, ktoré prinúti používateľa vykonať menej akcií na vstupných zariadeniach počítača za účelom vykonania editačnej zmeny na dokumente. Dôsledok použitia takejto metódy bude spočívať v rýchlejšej editačnej práci. Samotná metóda spočíva v použití podobného editačného riadka avšak zakomponovaného do samostatného náhľadu. V prípade označenia listu sa automaticky zmení komponentové zameranie platformy (focus) na editačný riadok, kde používateľ zadá text, stlačí klávesu enter a editačná požiadavka je aplikovaná.
Ďalšou rozumnou požiadavkou je potreba zadefinovať rozličné editačné režimy, medzi ktorými budeme schopný podľa potreby prechádzať. V kapitole o platforme Eclipse sme si popísali takýto grafický nástroj s názvom Perspektíva. V prípade projektu Euromath sme zadefinovali matematickú perspektívu za účelom editácie matematického textu. Pôvodnú perspektívu sme definovali ako implicitnú (default) perspektívu.
Celkový vzhľad oboch perspektív projektu Euromath je znázornený na obrázkoch 5.1, 5.2.



Obr. 5.1: Matematická perspektíva



Obr. 5.2: Implicitná perspektíva

Kapitola 6

Rozšírenia matematického editora

Táto kapitola sa venuje rozšíreniam matematického editora, ktoré nespadajú do oblasti nutnej pre správne fungovanie editora, avšak rozširujú a zlepšujú použiteľnosť či pohodlie používateľa.

6.1 Vkladanie matematického textu pomocou LATEXu

Prvotnou ideou stojacou za projektom Euromath bola snaha spojiť kladné vlastnosti texu(latexu), menovite jeho výrazovú silu, s jednoduchosťou editovania vo WYSIWYG editoroch v reálnom čase. Táto idea však v súčasnosti ešte nie je v projekte plne realizovaná a preto používateľ, na ktorého je projekt zameraný, používa v súčasnej dobe s najväčšou pravdepodobnosťou tex(latex). Z tohto pohľadu by bolo výhodné využiť znalosti texu nášho cieľového používatela k tomu, aby sa lepšie a rýchlejšie adaptoval pri prechode na editor Euromath.
Pri prvotnom porovnaní spôsobu členenia zapisovanej informácie v prostredí texu a v prostredí Euromath musíme naraziť na určité podobné črty. Za hlavné a najdôležitejšie považujem oddelenie matematického textu od slovného textu a štruktúrovanosť matematického textu v oboch prostrediach. Tex oddeluje matematický text od zvyšného textu pomocou vlastných štruktúračných príkazov (`\begin{math\...\end{math}`, `$...$`, `\(...\)`). V prípade Euromathu je matematika spracovaná ako samostatný menný priestor.

Takéto oddelenie nám umožní sa zamerať len na matematický text v oboch prostrediach a nezaoberať sa zvyškom dokumentu. Ďalšia spoločná vlastnosť, štruktúrovateľnosť matematického textu, je základom pre konverzie matematického textu medzi týmito prostrediami. Z týchto predpokladov vyplýva nová vlastnosť matematického editora, ktorá by mala veľký praktický význam. Jednalo by sa o konverziu matematického textu z prostredia tex do prostredia Euromath, konkrétne do menného priestoru MathML.
Kedže ťažisko tejto diplomovej práce spočíva hlavne vo vytvorení funkčného matematického editora, tak na tvorbu konverzie matematického textu budeme klasť menší dôraz. Z tohto dôvodu bolo prvotným cieľom pre splnenie tejto úlohy nájsť existujúci open source projekt, ktoré by riešil danú úlohu a bolo by ho možné ľahko zakomponovať do prostredia Euromath. Za účelom nájdenia vhodného softwaru sme museli zadefinovať určité požiadavky, ktoré by mal daný software spĺňať.

Licencia: Euromath je chránený open source licenciou MPL 1.1 6.1. Jedná sa o hybridnú licenciu stojacou medzi modifikovanou BSD licenciou6.1 a GPL licenciou6.1. Licencia GPL umožňuje voľne modifikovať a distribuovať software, avšak modifikovaná BSD licencia je už oveľa reštriktívnejšia. Licencia MPL teda síce umožňuje voľne modifikovať a distribuovať software znova pod licenciou MPL, avšak software môže obsahovať aj proprietárne časti, na ktoré sa možnosť modifikácie nevzťahuje. Môžeme si teda vybrať typ licencie pripojeného softwaru, ktorý by plnil úlohu konvertora. Rozhodli sme sa ale len pre software s open source licenciami a to z dôvodu ďalšej modifikácie pridaného softwaru. Pod modifikáciou sa v tomto prípade nemyslí iba modifikácie spadajúce do okruhu nutných zmien pre bezproblémové zapojenie do projektu ale i prípadne riešenie ďalších požiadaviek na konvertor akými sú napríklad špecifické chybové hlásenia parsovania texu, zvýrazňovanie textu a iné požiadavky, ktoré sa môžu neskôr ukázať ako zmysluplné. V ne-open source alternatívach by takéto dodatočné požiadavky nebolo možno riešiť[1].

Programovací jazyk: Vzhľadom na fakt, že projekt Euromath je naprogramovaný v jazyku Java, tak je preferencia tohto jazyka samozrejmosťou.

[1]Požiadavky na ďalšiu funkcionalitu obrátené na autora softwaru sú síce možné, avšak záruka, že ich autor v blízkej dobe implementuje alebo vôbec bude brať v úvah sú otázne a preto sa na takúto možnosť nedá vôbec spolahnúť

Otázkou však zostáva možnosť použitia iných programovacích jazykov. V jazyku Java existujú určité mechanizmy na pripojenie zdrojového kódu napísanom v inom programovacom jazyku, avšak má to svoje nevýhody. Jednou z nich je dodatočná starostlivosť o zdrojový kód. V budúcnosti by vývojár musel spracovávať viac zdrojových jazykov a ich spoluprácu, čo vytvára dodatočné podmienky na vývojára. To avšak nie je úplne neprípustná požiadavka. Neprípustné kritérium vidím v dodatočných požiadavkách na používatela projektu. Použitie iného jazyka totiž spôsobí dodatočné úkony pri inštalácií programu Euromath. Problém spočíva v tom, že zdrojový kód v nejavovskom jazyku nedokáže využiť virtuálny stroj Javy a preto sa bude musieť z takéhoto kód znova vytvárať binárna podoba pre daný operačný systém, na ktorý sa bude projekt Euromath inštalovať. Takýmto spôsobom sa vo vysokej miere zužuje ľahká platformová nezávislosť zabezpečená ideou virtuálneho stroja. Toto je už príliš neprípustny dôsledok. Z tohto dôvodu pripadá v úvah len software napísaný jazykom Java.

Pri hladaní vhodného softwarového riešenia sme využili internet a našli sme mnoho zaujímavých softwarových riešení ([13],[14]). Avšak ani jedno z nich plne nespĺňalo nami požadované kritéria či existoval iný závažný dôvod nepoužiť daný software. Nebudeme spomínať všetky odmietnuté softwarové riešenia. Spomenieme len tie, ktoré boli nášmu cieľu najbližšie, pričom popíšeme aj ich nedostatky kvôli ktorým neboli tieto softwarové riešenia vhodné.

TTM (TexToMathML) ([15]) Podľa domovskej stránky tohto projektu sa jedná o projekt s presne požadovanou funkcionalitou, avšak jeho licencia je komerčnejšieho rázu a obmedzuje použitie tohto projektu v plnom rozsahu. Tento fakt spolu s použitým implementačným jazykom, jazykom C, vylučujú tento projekt z výberu.

Tex4ht ([16]) Projekt je zameraný na konverziu texu do html podoby, pričom je schopný vytvoriť z matematického textu i MathML podobu. Jednou nevýhodou tohto projektu je, že sa zameriava na zložitejší problém ako ten, ktorý v skutočnosti potrebujeme vyriešiť. Pripojenie takéhoto projektu by znamenalo nutne pripojiť i určitý nevyužiteľný balastný kód. Toto by v danom rozsahu projektu nepredstavovalo až taký veľký problém, avšak projekt závisí ešte aj od iného softwaru, konkrétne od softwaru umožňujúcemu prácu v Texu. Podstata tohto projektu nespočíva v manipulácií so samotným zdrojovým súborom texu (*.tex) ale

až s jeho binárnejšou verziou formátu DVI6.1. Avšak k vytvoreniu DVI súboru sa musí použiť software k Texu, čo v konečnom dôsledku znamená potrebu nainštalovať tohto softwaru pre správny chod programu. A práve toto kritérium si protirečí s filozofiou rozšírenia matematického editora v projekte Euromath. Euromath existuje za účelom poskytnutia pohodlnejšej alternatívy k používaniu (La)Texu, pričom navrhované rozšírenie matematického editora malo za účel uľahčiť prechod z texu. Použitím projektu Tex4ht by sa však opätovne vytvorila závislosť na texu, hoci by bola iba nepriama v podobe potreby nainštalovania softwaru, ktorý s ním pracuje.

Hermes ([17]) Zameraním sa jedná o projekt, ktorý si kladie za cieľ zkonvertovať tex súbory do xml podoby za účelom archivácie. Kladom projektu je jeho open source licencia (GPL). Nevýhodou však zostáva nutnosť inštalácie softwaru k použitiu texu a to z rovnakého dôvodu ako v predošlom projekte.

Tralics ([18]) Projekt zaoberajúci sa konverziou Latexu do Xml podoby. Dôvodom vylúčenia z množiny vhodných kandidátov bolo použitie nevhodného implementačného jazyka (jazyk C).

Navzdory neúspechu pri hľadaní vhodného softwarového riešenia nášho problému sme sa rozhodli nájsť aspoň teoretické algoritmické riešenie problému. Riešenie, ktoré si bližšie vysvetlíme, zostane len riešením teoretickým(t.j. neimplementuje sa) vzhľadom na fakt, že celý problém je len problémom riešiacim dodatočnú vlastnosť matematického editora, na ktorú nie je kladený dôraz v tejto diplomovej práci.

Pred algoritmickým riešením konverzie je nutné sa zamyslieť nad presnejšou definíciou požiadaviek na algoritmus. V prípade softwaru tretiej strany stačila jednoduchá požiadavka konverzie, pričom takýto software by sa následne modifikoval podľa potrieb Euromathu alebo by sa v prípade nemodifikovateľnosti daného software zvážili určité kompromisné riešenie a modifikoval by sa Euromath. V tomto prípade však vytvárame nový algoritmus, ktorý vytvoríme priamo pre potreby projektu Euromath, a tak je na mieste zvažovať dodatočné požiadavky, kedže kompromisy ohľadom kompatibility nemusia byť príjmané. Rozumnou dodatočnou požiadavkou je podpora chybových hlásení a to hlavne z procesu parsovania texovského zdroja. Pri písaní

matematického textu v texu sa môže jednoducho vyskytnúť preklep a pomocou presných chybových hlásení programu si môže používateľ systematicky a rýchlo opraviť takéto chyby v texte. Ďalšou užitočnou vlastnosťou by mohlo byť automatické doplňovanie, prípadne výber, zvyšku slova. Takýchto a podobných vylepšení sa dá vymyslieť mnoho a preto by mal byť algoritmus odolný voči zmenám, resp. obsahovať či pozostávať z dobre rozšíriteľných a nahraditeľných časti. Kedže myšlienka rozšíriteľných a nahraditeľných častí programu nie je v oblasti programovania vôbec inovátorska, je vysoko pravdepodobná existencia určitého riešenia. Týmto riešením sú nepochybné rôzne návrhové vzory, pomocou ktorých sa dajú skryť implementačné metódy. Takéto skrytie nám zaručí z veľkej časti požadované vlastnosti odolnosti voči zmenám.
Myšlienka hľadania algoritmického riešenia v už existujúcich algoritmických technikách sa dá aplikovať aj na problém konverzie matematického textu z texu do MathML. V informatike existuje totiž nespočetne veľa rôznych dátových formátov a ako dôsledok tohto faktu existuje i veľa požiadaviek na konverziu medzi nimi. Vhodné riešenie nášho problému vidíme v použití teoreticko-praktickom aparáte kompilátorov([19]). Hlavným cieľom kompilátora je totiž tiež konverzia, i keď konverzia medzi zdrojovým jazykom a strojovým kódom. V teoretickej podstate sa však jedná o zložitejší konvertor, ktorý konverguje informáciu zo vstupného formátu do výstupného formátu. Najprv si popíšeme fungovanie kompilátora a následne sa zamyslíme nad množinu modifikácií, ktoré by vytvorili z kompilátora pre nás vhodný konvertor.

Pred popisom fungovania kompilátora si potrebujeme zadefinovať niekoľko pojmov([20]), ktoré sú kľúčové pre správne pochopenie teórie stojacou za kompilátormi.

Abeceda je konečná neprázdna množina písmen(symbolov).

Slovo nad abecedou Σ je konečná postupnosť písmen z abecedy Σ. Prázdnu postupnosť nazývame prázdne slovo ε

Jazyk nad abecedou Σ je ľubovoľná množina slov nad abecedou Σ.

Token je písmeno výstupnej abecedy zo scannera/ vstupnej abecedy pre parser.

Lexéma je slovo zdrojovej abecedy pre kompilátor reprezentované po prechode scannerom jedným tokenom.

Zreťazenie slov - Nech $u = u_1u_2 \dots u_n, v = v_1v_2 \dots v_m$, potom $u \cdot v = u_1u_2 \dots u_nv_1v_2 \dots v_m$. Operácia zreťazenia teda pripojí druhé slovo na koniec prvého. Znak zreťazenia (bodku) budeme väčšinou vynechávať.

Iterácia abecedy Σ (značíme Σ^*) je množina obsahujúca všetky slová, ktoré dostaneme zreťazením niekoľkých písmen z Σ.

Frázová gramatika je štvorica G = (N, T, P, σ), kde N a T sú abecedy neterminálnych a terminálnych symbolov ($N \cap T = \emptyset$), $\sigma \in N$ je začiatočný neterminál a $P \in_{kon} (N \cup T)^*N(N \cup T)^* \times (N \cup T)^*$ je konečná množina prepisovacích pravidiel.

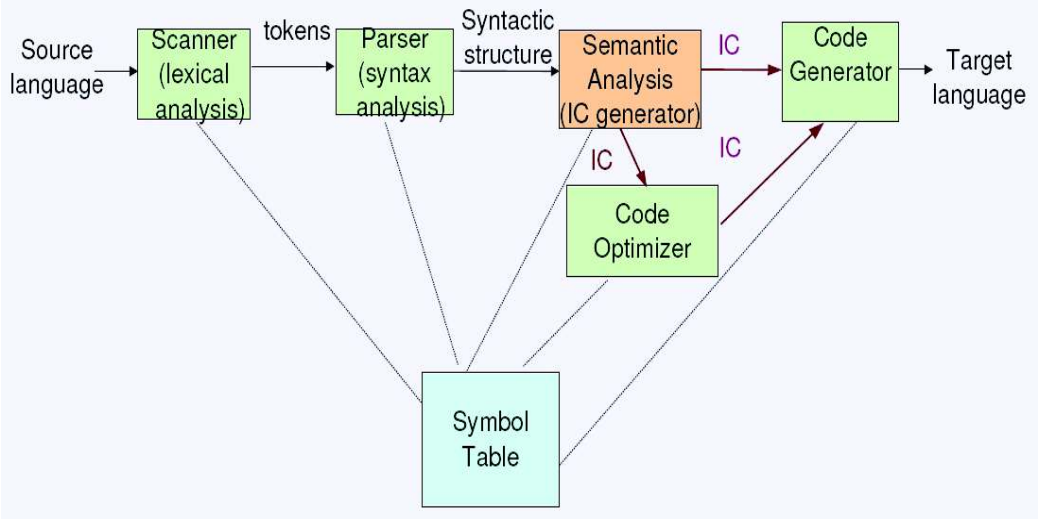
Krok odvodenia v gramatike G je binárna relácia $\Rightarrow$ na $(N \cup T)^*$ definovaná nasledovne: $x \Rightarrow y$ práve vtedy, keď existujú slová w_1;w_2 a pravidlo $u \Rightarrow v \in P$ také, že $x = w_1uw_2$ a $y = w_1vw_2$.

Jazyk generovaný gramatikou G je množina $L(G) = (w \in T^* \mid \sigma \Rightarrow_G^* w)$.

Bezkontextová gramatika je taká frázová gramatika, v ktorej navyše platí $P \in N \times (N \cup T)^*$.

Regulárna gramatika je taká frázová gramatika, v ktorej navyše platí $P \in N \times T^*(N \cup \{\varepsilon\})$.

Syntaktický strom pre danú gramatiku G a dané slovo w je orientovaný usporiadaný stromový graf s vnútornými uzlami označenými neterminálnymi symbolmi a listami označenými terminálnymi symbolmi z gramatiky G, pričom výstupné hrany z vrchola predstavujú aplikáciu 1 kroku odvodenia na neterminál v danom vrchole. Strom je taktiež usporiadaný, t.j. deti vrchola sú vždy usporiadané, a naviac v takom poradí ako v pravidle odvodenia cez ktoré vznikli. Poslednou podmienkou je, že usporiadanie listov tvorí samotné slovo w. Takto definovaných syntaktických stromov môže existovať pre danú gramatiku G a dané slovo w viacero.



Obr. 6.1: Všeobecná štruktúra kompilátora

Ľavé krajné odvodenie je spôsob odvodzovania slova pomocou gramatických pravidiel z počiatočného neterminálu, ktorý aplikuje pravidlo vždy na prvý neterminál zľava. Obdobne sa definuje pravé krajné odvodenie.

Ozdobený syntaktický strom (annotated parse tree) je syntaktický strom, ktorému sú pridané do vrcholov atribúty. Každý vrchol môže mať iné atribúty, pričom ich správne vyhodnotenie (zdobenie stromu) môže závisieť od atribútov jeho synov (syntetizovaný atribút) či od atribútov jeho rodiča a súrodencov (dedený atribút).

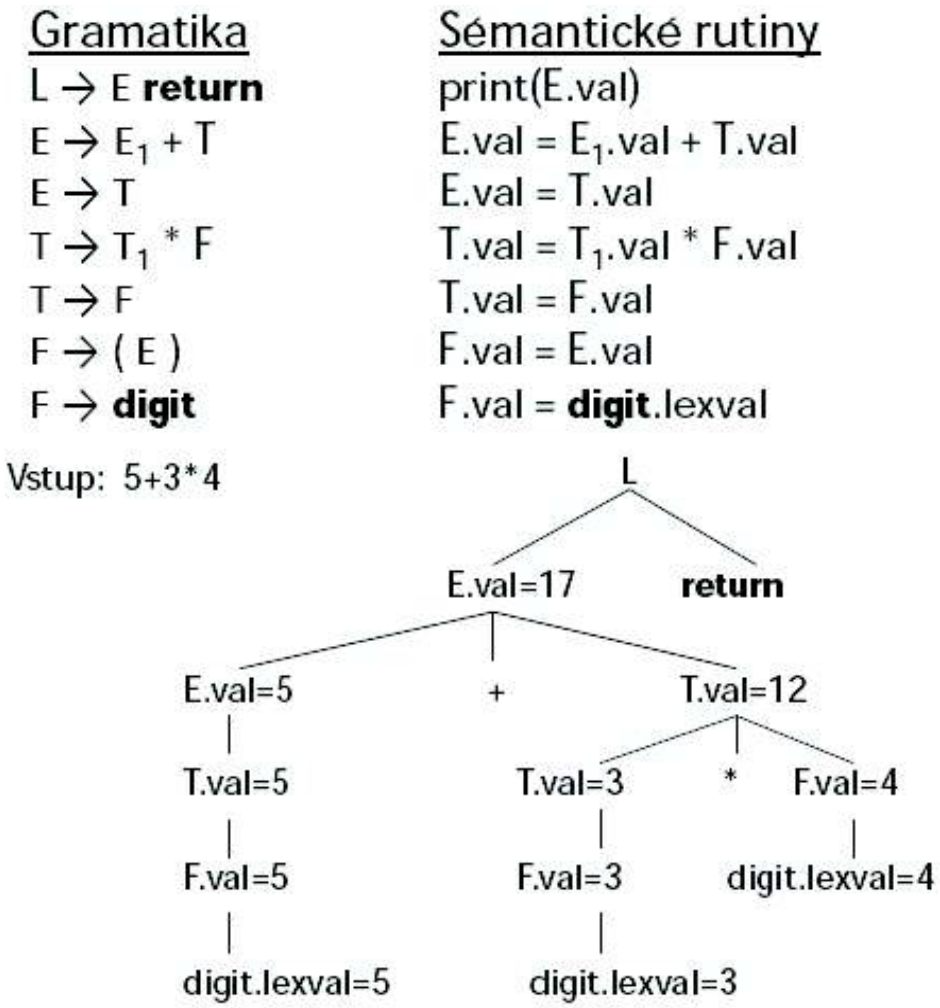
Na obrázku 6.1 je znázornené vnútro všeobecného kompilátora a tok výpočtov medzi jeho jednotlivými časťami. Tok výpočtu sa začína vstupom napísanom v zdrojovom jazyku. Vstup je prvotne spracovaný lexikálnym analyzátorom(scannerom) do postupností tokenov. Hlavnou funkciou lexikálnej analýzy je zjednodušenie práce pre ďalšie časti kompilátora. V podstate sa jedná o odstránenie pre kompilátor nepotrebných častí zdrojového textu. Takýmito nepotrebnými časťami sú zväčša medzery, komentáre, symboly kódujúce koniec riadka a podobne. Vstup v tomto prípade predstavuje postupnosť slov jazyka, ktorý vieme generovať gramatikou, t.j. poznáme generujúcu gramatiku. Pre jednoduchosť predpokladajme, že generujúca gramatika je regulárna. Lexikálna analýza sa pokúša nahradiť jednotlivé slová (postupnosti

terminálnych symbolov) zo vstupu na zvolené neterminálne symboly. Keďže je regulárna gramatika ekvivalentná s regulárnymi výrazmi, tak sa dá proces lexikálnej analýzy popísať ako vyhľadávanie regulárnych výrazov v texte, pričom novovyhľadaný regulárny výraz musí nasledovať tesne po predošlom vyhľadanom regulárnom výraze. Každý regulárny výraz má priradený práve 1 symbol (token), ktorým bude vyhľadané slovo podľa daného regulárneho výrazu nahradené vo výstupe. Môže nastať situácia,že na prefix textu, v ktorom sme ešte regulárne výrazy nevyhľadávali, sa dá aplikovať viacero regulárnych výrazov. V takomto prípade sa dá reagovať rôznymi spôsobmi, avšak heuristické riešenie je použiť regulárny výraz, ktorý nachádza najdlhšie slovo. Najjednoduchším riešením je ale vyhnúť sa takýmto kolíziam pri definovaní regulárnych výrazov(regulárnej gramatiky). Po rozdelení vstupného textu podľa vyhľadaných regulárnych výrazov sa vytvorí postupnosť tokenov podľa postupnosti vyhľadaných regulárnych výrazov a vytvorí sa záznam v tabuľke symbolov spájajúci token na danom mieste v poradí s jeho vyhľadaným slovom(lexémou) v prvotnom vstupe. Samozrejme tokeny predstavujúce výrazy, ktoré chceme vynechať, vynecháme z výstupnej postupnosti a i z tabuľky symbolov. Ukážme si lexikálnu analýzu na príklade :

vstup : `\sum\frac{x}{  a^{2}  }  +  1`
výstup : token sumy, token zlomku, token ľavej riadiacej zátvorky, token identifikátora, token pravej riadiacej zátvorky, token ľavej riadiacej zátvorky, token identifikátora, token horného indexu, token ľavej riadiacej zátvorky,token čísla,token pravej riadiacej zátvorky,token pravej riadiacej zátvorky, token operácie sčítania, token čísla

V danom príklade bolo jasným cieľom odstrániť nežiadúce medzery, t.j. vyhľadať ich regulárnym výrazom a ignorovať ich pri tvorbe výstupu.
Programovací jazyk Java poskytuje vo svojej štandartnej forme balíky podporujúce vyhľadávanie regulárnych výrazov, takže tvorba algoritmu lexikálnej analýzy je pomerne jednoduchou záležitosťou.
Tok výpočtu sa ďalej presunie do časti syntaktickej analýzy, ktorému je daný vstup vo forme postupnosti tokenov. Úlohou syntaktickej analýzy je pre danú postupnosť tokenov a pre danú gramatiku generujúcu danú postupnosť tokenov vytvoriť syntaktický strom daný generujúcou gramatikou pri odvodzovaní danej vstupnej postupnosti tokenov. V tomto prípade gramatika považuje za svoje terminálne znaky tokeny a jej úlohou je povedať ktoré reťazce tokenov sú prípustné, t.j. či vstupný reťazec patrí do nami požadovaného

generovaného jazyka. Bez ujme na všeobecnosti predpokladajme, že generujúca gramatika je bezkontextová, pretože bežné programovacie jazyky sú bezkontextové a zápis matematického textu v texu sa syntakticky príliš nelíši od zápisu výrazu v bežnom programovacom jazyku.
Syntaktický strom sa dá získať vo všeobecnosti viacerými metódami. Najjednoduchšou je rekurzívne skúšanie odvodzovania počiatočného neterminálu v koreni stromu. Táto metóda nie je však najefektívnejšia. Existujú ešte 2 ďalšie efektívnejšie možnosti. Jedná sa o parsovacie metódy zdola-nahor(bottom-up) a zhora-nadol(top-down). Ako ich názvy napovedajú, líšia sa spôsobom tvorby syntaktického stromu. Metóda zhora-nadol postupuje od koreňa syntaktického stromu ľavým krajným odvodením. Pomocou pomocných dátových štruktúr extrahovaných z gramatiky sa vytvorí parsovacia tabuľka hovoriaca pre daný stav aké pravidlo použiť. Pod daným stavom máme na mysli neterminál, ktorý potrebujeme odvodiť, a terminál, ktorý by sa mal vo výslednom slove následne odvodiť. Samozrejme, že pri takomto odvodzovaní z bezkontextovej gramatiky môžu nastať problémy(zacyklenie parsovania, nejednoznačnosť použitia pravidla(2 pravidlá pre 1 bunku tabuľky)) a preto treba bezkontextovú gramatiku ešte vhodne upraviť. Pri metóde zdola-nahor sa pokúšame vystavať syntaktický strom od listov ku koreňu. Podobne ako v predošlej metóde i tu si musíme vytvoriť parsovaciu tabuľku hovoriacu, ktoré pravidlo máme použiť ako ďalšie, avšak nie je nutná konverzia bezkontextovej gramatiky. Táto metóda sa pokúša hľadať odvodzovacie pravidlo v opačnom poradí ako sú pri odvodzovaní aplikované a tak i pri spracovávaní slova zľava doprava bude výsledný syntaktický strom stromom pravého krajného odvodenia. Mohlo by sa zdať, že metóda zdola-nahor je jednoduchším riešením vzhľadom na nepotrebnosť konverzie gramatiky, avšak nie je tomu vždy tak vzhľadom na ďalšie operácie kompilátora.
Výpočtový tok sa presúva do časti sémantickej analýzy, ktorej hlavnou úlohou je vytvoriť tzv. medzikód (IntermediateCode). Jedná sa medziprodukt konverzie do zdrojového kódu. Väčšinou je vo forme jednoduchých inštrukcií pre jednoduchý abstraktný stroj. Dôvod pre prechodný kód je potreba inštrukčnej formy kódu pre potrebu optimalizačných konverzií kódu. Sémantická analýza môže obsahovať i sémantické kontroly vstupu, ktoré nie je možné vykonať v predošlých častiach kompilátora z dôvodu nedostačujúcej sily bezkontextových gramatík. Pri programovacích jazykoch sa jedná hlavne o kontrolu typov. Po optimalizácií medzikódu už nastáva relatívne priamy preklad do výsledného jazyka.
Medzikód sa získava z ozdobeného syntaktického stromu pomocou sémantic-



Obr. 6.2: Príklad gramatiky generujúcej matematický výraz so sémantickými rutinami počítajúcimi hodnotu tohto výrazu. Príklad ozdobeného syntaktického stromu.

kých rutín. Sémantické rutiny(viď. obr. 6.2) sú metódy spustiteľné na jednotlivých vrcholoch ozdobeného syntaktického stromu, ktoré môžu ohodnocovať atribúty vrchola podľa hodnôt atribútov predkov, potomkov či súrodencov vrchola, tvoriť podľa dostupných hodnôt atribútov medzikód či plniť iné rozmanité úlohy. Sémantické rutiny sa definujú pre jednotlivé pravidlá gramatiky a pre daný vnútorný vrchol sa aplikuje sémantická rutina z pravidla, ktoré vytvorilo potomkov tohto vrchola. V prípade nutnosti sa dajú dodefinovať určité inicializačné metódy i pre listy, ktoré obsahujú tokeny. Nie je zriedkavým javom, že syntaktická a sémantická analýza splýva do jedného celku a sémantické rutiny sú volané pri tvorbe ozdobeného syntaktického stromu. V takýchto prípadoch môže byť použitie parsovacej metódy zhora-nadol výhodou oproti metóde zdola-nahor, pretože pri tvorbe stromu vždy vieme, v ktorom odvodzovacom pravidle sa práve nachádzame a nedozvieme sa to až keď je pravidlo celé použité. To značne uľahčuje písanie sémantických rutín ku gramatickým pravidlám.

Po popise fungovania kompilátora by bolo vhodné sa zamyslieť nad jeho modifikáciami, ktoré su nutné pre vytvorenie konvertora matematického textu z texovskej formy do formy MathML. Po krátkej úvahe je zrejmá nepotrebnosť medzikódu, pretože pri konverzií nebude potreba optimalizovať matematický text. Bude prebiehať v celku priamočiara konverzia matematického textu. Generáciu matematického textu vo forme MathML by sme teda presunuli už do sémantickej analýzy a to v podobe generovania cez sémantické rutiny. Bežný jav spájania lexikálnej a sémantickej analýzy by v tomto prípade tiež nebolo vhodné aplikovať vzhľadom na možnosť existencie určitej potreby použiť informáciu v sémantických rutinách, ktoré sú nedostupné pomocou atribútov pri procese tvorby ozdobeného syntaktického stromu. V takomto prípade sa dá informácia získať len na po vytvorení celého stromu. Ostatné časti pôvodného kompilátora môžu zostať zachované, pričom k funkčnému konvertoru treba ešte vytvoriť mapovanie medzi regulárnymi výrazmi pre matematický text v texu a tokenmi, ktoré si vhodne zvolíme, bezkontextovú gramatiku spracovávajúcu štruktúry texu spolu so sémantickými rutinami prekladu do formy MathML.
Za spomenutie stoja i realizácie spomínaných vlastností konvertora, ktoré zavážili pri výbere teórie kompilátorov ako vhodnej myšlienky na modifikáciu. Máme na mysli konkrétne inteligentné chybové hlásenia a automatické dopĺňanie výrazov. Možnosti zakomponovať chybové správy sa vyskytujú v priebehu celého výpočtového toku a to nám umožní vytvoriť tieto správy

presnejšie a teda viac nápomocné používatelovi. V lexikálnej analýze sa môžu vyskytovať chyby súvisiace s nedefinovanými znakmi či spojeniami znakov. V syntaktickej analýze zachytíme problémy s nevhodným reťazením príkazov a v prípade viactokenových príkazov aj preklepové omyly. V sémantickej analýze sa potom môžme zamerať na chyby, ktoré sa nemusia odhaliť na prvý pohľad, a to pomocou zložitejších sémantických kontrôl. Vlastnosť dopĺňania slov je ľahko implementovateľná v syntaktickej analýze, kde pri parsovaní sa ponúkajú z parsovacej tabuľky prepisovacie pravidlá, ktoré napovedajú možný vývoj písania slova.

Záver

Hlavným cieľom tejto diplomovej práce bolo rozšírenie projektu Euromath 2. Upresnením tohto cieľa vznikli 3 samostatné podciele. Prvým cieľom bolo vytvoriť grafické rozhranie pre editáciu matematického textu. Naplnenie tohto cieľu bolo úspešné i keď podpora editácie matematiky sa dá ešte doplniť implementáciu rôznych vylepšení. S tým súvisí aj dodatočný cieľ rozširovania, potreba vlastnosti konvertujúca Tex na MathML. Tento cieľ nebol úspešne dosiahnutý vzhľadom na pôvodný zámer použiť software tretej strany. Software spĺňajúci všetky požiadavky nebol nájdený a preto pre daný problém bolo načrtnuté podrobné algoritmické riešenie, ktoré zostalo iba v teoretickej rovine vzhľadom na dodatočnosť tohto cieľa. Posledným a nemenej dôležitým cieľom bola podpora budúcich vývojárov projektu Euromath 2. Celková podpora bola kompromisom medzi príliš detailnou analýzou kódu či technických detailov platformy Euromath a medzi príliš abstraktnými ideami projektu Euromath 2 a platformy Eclipse. Mohlo by sa namietať, že vývojárska podpora by mala obsahovať ešte rôzne iné veci, avšak zmyseľ takejto podpory bol vo vytvorení východiskového bodu pre budúcich vývojárov. Takýto cieľ bol splnený, pretože som sa zaoberal oblasťami, ktoré budú budúci vývojári určite potrebovať.

Budúce smerovanie projektu vidím hlavne v odstránení rýchlostných problémov projektu implementáciou procesora čiastočnej XSL transformácie. V oblasti rozšírenia funkcionality, vidím hlavný vývoj projektu vo vytváraní podpory pre dodatočné menné priestory. Nové menné priestory sa síce už dajú pripájať avšak nemusia mať renderer či špecializovaný editor so špecializovaným grafickým rozhraním. V tomto smere som v tejto práci ponúkol určitý model postupu vytvárania špecifického grafického rozhrania.

Literatúra

[1] **Tomáš Studva**: WYSIWYG editovacie techniky XML dokumentov s použitím XSLT , Fakulta matematiky, fyziky a informatiky, Univerzita Komenského, Bratislava, 2007.

[2] **Martin Kollár**: Renderovanie XML dokumentov pri WYSIWYG editovaní , Fakulta matematiky, fyziky a informatiky, Univerzita Komenského, Bratislava, 2007.

[3] WYSIWYG xml editor **Euromath2**, Fakulta matematiky, fyziky a informatiky, Univerzita Komenského, Bratislava,http://euromath2.sourceforge.net/

[4] Schematic, Euromath subproject, http://euromath2.sourceforge.net/schematic/

[5] GENE, Euromath subproject, http://euromath2.sourceforge.net/gene/

[6] projekt JEuclid, MathML renderer ,http://jeuclid.sourceforge.net/

[7] vývojové prostredie Eclise založená na programovacom jazyku a technológii Java http://www.eclipse.org/

[8] Java a návrhové vzory, Fakulta matematiky, fyziky a informatiky, Univerzita Komenského, Bratislava, 2007 (moja bakárska práca)

[9] Elektronický help systém integrovaný do vývojoveho prostredia Eclipse 3.3

[10] Dokumentácia k jazyku Java 1.5 (http://java.sun.com/j2se/1.5.0/download.jsp#docs)

[11] Hlavná stránka konzorcia W3C pre matematický text (http://www.w3.org/Math/)

[12] Špecifikácia MathML 2.0 (pdf verzia)(http://www.w3.org/TR/MathML2/mathml-p.pdf)

[13] Internetová stránka venovaná konvertorom z/do MathML, (http://www.w3.org/Math/Software/mathml_software_cat_converters.html)

[14] Prezentácia venovaná nástrojom na písanie publikácií, (http://www.csun.edu/ hcmth008/mathml/converting_to_mathml.pdf)

[15] Stránka projektu TTM (Tex To MathML)(http://hutchinson.belmont.ma.us/tth/mml/)

[16] Stránka projektu Tex4ht (Tex for Html)(http://www.cse.ohio-state.edu/ gurari/TeX4ht/)

[17] Stránka projektu Hermes (http://hermes.roua.org/)

[18] Stránka projektu Tralics (http://www-sop.inria.fr/apics/tralics/)

[19] Stránka k predmetu Kompilátory (Fakulta matematiky, fyziky a informatiky, Univerzita Komenského, Bratislava)(http://www.dcs.fmph.uniba.sk/ sturc/kompilatory/)

[20] Stránka k predmetu Formálne jazyky a automaty (Fakulta matematiky, fyziky a informatiky, Univerzita Komenského, Bratislava)(http://foja.dcs.fmph.uniba.sk/materialy.php)

Zoznam skratiek

WYSIWYG=what you see is what you get

XML=eXtensible Markup Language

XSLT=XML transformácia (XSL transformacion) (používa sa aj spojenie XSLT transformácia)

FATS=File And Transformation System

MathML=Mathematical Markup Language (XML jazyk na zápis matematických symbolov, vzorcov a formúl)

RCP=Rich Client Platform (Minimalistický systém na základe platformy Eclipse)

PDF=Portable Document Format

DTD=Document Type Definition, schématický jazyk umožňujúci definovať reštriktívne obmedzenia na štruktúru XML dokumentu.

XSL-FO=eXtensible Stylesheet Language Formatting Objects, pôvodne označovaný ako XSL (rodina jazykov, ktoré určujú ako majú byt dokumenty v XML formáte formátované alebo transformované)

SVG=Scalable Vector Graphics (Jazyk pre dvojdimenzionálnu grafiku vo formáte XML)

MVC=návrhový vzor Model-view-controller

GUI=Graphical User Interface (grafické používateľské rozhranie)

GEF=Graphical Editing Framework

GPL=GNU GPL=GNU General Public License, licencia umožňujúca modifikovať a distribuovať chránené dielo

MPL=Mozilla Public License, open source licencia vytvorený a používaný organizáciou Mozilla vo svojich softwarových produktoch

BSD=Berkeley Software Distribution, modifikovaná BSD licencia je reštriktívnejší typ licencie ako je licencia GPL

DVI=Device independent file format, výstupný formát sázačského programu Tex

Prílohy (v CD forme)

1. Zdrojové súbory projektu Euromath (workspace ku prostrediu Eclipse 3.3).
2. Vybrané literárne zdroje (uložené internetové stránky, iné dokumenty)