Univerzita Komenského
Fakulta Matematiky, Fyziky a Informatiky
Ústav Informatiky

Tomáš Hanakovič

Diktovanie matematických formúl

Diplomová práca

Školiteľ: Mgr. Marek Nagy

Bratislava 2006

Abstrakt

HANAKOVIČ Tomáš. *Diktovanie matematických formúl* [diplomová práca]. Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky; Katedra informatiky. Bratislava: FMFIUK, 2006. 52 s.

Práca popisuje analýzu, návrh a implementáciu multimodálnej[1] webovskej aplikácie, ktorá slúži na zadávanie jednoduchých matematických formúl. Používa sa vizuálne i hlasové rozhranie v podobe dialógu. Dôraz je kladený na prehľadnosť a jednoduchú orientáciu v používateľskom rozhraní. Aplikácia je plne funkčná aj s výlučným použitím hlasového vstupu, čím získava ambície byť prístupnejšou aj pre zrakovo postihnutých. Spomínané vlastnosti sa dosiahli použitím zaužívaných grafických prvkov a rečových konštrukcií. Pre lepšiu orientáciu vo formule sa využila hierarchická identifikácia jej prvkov. Z technologického hľadiska sú použité existujúce nekomerčné štandardy, čo má za následok otvorený produkt založený na spoľahlivých technológiách s dobrým potenciálom na ďalší vývoj.

[1] Aplikácia, ktorá sa dá ovládať viac ako jedným typom modality

Obsah

Abstrakt.....**1**
Obsah.....**3**
1 Úvod.....**5**
1.1 Motivácia.....5
1.2 Realita.....5
1.3 Diktovanie a matematika.....6
1.4 Cieľ.....6
2 Prehľad.....**7**
2.1 Formulácia problémov.....7
2.2 Diktovanie.....8
3 Analýza.....**13**
3.1 Analýza požiadaviek.....13
3.2 Práca s aplikáciou.....15
3.3 Reprezentácia formuly.....16
4 Návrh.....**19**
4.1 Technológie a architektúra.....19
4.2 Klient.....20
4.3 Server.....27
5 Implementácia.....**33**
5.1 Klient.....33
5.2 Server.....40
6 Záver.....**45**
7 Referencie.....**47**
8 Prílohy.....**49**

1 Úvod

1.1 Motivácia

Nedávny rozkvet multimediálnych technológií dal osobným počítačom (PC) nový rozmer. I keď mnoho ľudí používa PC stále len ako písací stroj, je zrejmé, že dnes dokáže oveľa viac. Je tomu tak určite aj vďaka stále rastúcej výpočtovej rýchlosti – práve kvôli nej dnes môžu bežné PC spracovávať video či hudbu v reálnom čase.

Aj keď sa vývoj poberá dopredu, zdá sa, že širokej pospolitosti používateľov stačí na uspokojenie relatívne málo a v povedomí ľudí pomaly bledne sen o „počítači, ktorý mi bude rozumieť“. Sny zo sci-fi filmov o inteligentných počítačoch, s ktorými sa bude dať „porozprávať“, sa vytrácajú.

1.2 Realita

Z názvu práce si možno domyslieť, že sa nebudem zaoberať tým, ako zostrojiť inteligentný počítač, s ktorým sa bude dať konverzovať. Chcem len poukázať na fakt, že vízia toho, že nám počítač „porozumie“, nie je vôbec taká vzdialená, ako by sa mohlo niekedy zdať. Osobne sa mi zdá, že techniky spracovania zvuku sú dnes dostatočne prepracované a ustálené. Preto som presvedčený, že číra hlasová interakcia s počítačom nie je žiadne „sci-fi“. Treba si len uvedomiť niektoré veci. Jednou z hlavných je, že (podľa väčšinového názoru) počítače nie sú inteligentné, len inteligenciu simulujú. Je teda neprirodzené očakávať od počítača, aby simulovaním inteligencie obsiahol sémantiku prirodzenej reči. Mám pocit, že práve toto nedorozumenie odradilo mnohých ľudí od používania hlasu pri počítači. Nedovolím si hodnotiť, či je vývoj hlasom ovládaných aplikácií dostatočne bohatý alebo nie, no dúfam, že každá jedna multimodálna aplikácia pomôže oživiť túto myšlienku.

Treba uznať, že technológie v nedávnych rokoch dosť pokročili. Hlavne výpočtová sila umožnila rozpoznávaniu reči (aspoň teoreticky) celkom slušný pokrok. Bohužiaľ, programové zázemie v dobe písania tejto práce nie je práve na najlepšej úrovni, tým viac pre spracovanie slovenčiny. Pravdepodobne každý z nás si uvedomuje, že konverzácia s počítačom je stále raritou. Ale myslím si, že netreba byť skeptický. Aj keď s nami počítače nikdy nebudú vedieť „voľne konverzovať“, no určite sú nám už dnes schopné porozumieť, ak sa s nimi budeme rozprávať podľa určitých pravidiel.

1.3 Diktovanie a matematika

Viem, že matematika nie je zrovna konverzačný jazyk, no podľa mňa je dobrou ukážkou toho, že vety komplexného jazyka s dobre definovanými pravidlami (akými sú i matematické výrazy) sa už v dnešnej dobe dajú „dostať do počítača“ pomocou diktovania. V akademickej oblasti sa možno často stretnúť s odvážnymi teóriami, ktoré čakajú, až svet „dozreje“ na ich praktickú aplikáciu. Myslím si, že hlasový vstup sa už svojho času dočkal. Dnes bežne fungujú veľké komerčné systémy, v ktorých si je pomocou telefónu možné objednávať tovar či služby. Ja chcem však ukázať, že nielen komerčné aplikácie, ale aj tie akademické či aplikácie pre „bežného človeka“, sú rovnako dobrí kandidáti pre implementáciu hlasového rozhrania. Stačí len dobre sformulovať, o čom konverzácia bude a ako bude prebiehať.

V dobe písania tejto práce som sa stretol s naozaj veľmi malým počtom hlasom ovládaných aplikácií pre PC, nie to ešte matematického razenia. Svetlým bodom je program MathTalk [1], ktorý je však komerčný a založený na proprietárnej technológii. A práve preto mám dôvod túto situáciu aspoň trochu zlepšiť. Pre nedostatočné programové zázemie je vznik nekomerčného, univerzálneho a dostatočne komplexného diktovacieho programu zatiaľ iba víziou. Avšak realizácia aspoň nejakej podmnožiny toho, čo sa z matematiky nadiktovať dá, môže pozitívne prispieť k rozvoju v tejto oblasti.

1.4 Cieľ

Ako som už vyššie spomenul, mojím hlavným cieľom je „ukázať svetu“, že hlasová komunikácia s počítačom má svoje uplatnenie všade. Chcem vytvoriť aplikáciu schopnú rozpoznávať diktované matematické výrazy. Odozva aplikácie by mala byť vizuálna i zvuková a jej ovládanie by malo byť umožnené jednak „klasickým“ spôsobom (myš, klávesnica) a rovnako pomocou hlasu. Aplikácia bude v anglickom jazyku z vyššie spomenutých technologických dôvodov.

Hotovú aplikáciu môže využiť prakticky ktokoľvek, komu je hlasový vstup príjemnejší ako písanie v TeX-u [2] či „klikanie“ v rôznych komerčných editoroch. V neposlednom rade treba mať na pamäti, že zrakovo postihnutí môžu silu hlasu využiť v plnej miere, nakoľko je mojím cieľom aplikáciu spraviť intuitívnu s efektívnym hlasovým výstupom aj bez potreby vizuálnej odozvy. I keď výber technológie, na ktorej je postavená táto práca, zadávanie značne spomaľuje (nútenými pauzami pri rozpoznávaní), dovolím si tvrdiť, že s pokrokom a hlavne rozšírením technológii sa môžeme s hlasom dostať oveľa ďalej.

2 Prehľad

2.1 Formulácia problémov

Na úspešné dosiahnutie hlavného cieľa bolo treba vyriešiť viacero súvisiacich problémov uvedených v tejto kapitole.

2.1.1 Nejednoznačnosť

Majme úlohu nadiktovať matematický výraz. Väčšina ľudí rieši toto zadanie bez zdĺhavého uvažovania v intuitívnej rovine so slovami: „Už som toľkokrát počul hovorenú matematiku, že mi je jasné, ako niečo nadiktovať!“. Treba si však uvedomiť, že i napriek tomu, že matematické zápisy majú tendenciu byť jednoznačné, pri ich čítaní sa často spoliehame na intuíciu. Ako príklad skúsme zapísať vetu „*a i plus b j*“. Do úvahy prichádzajú minimálne 2 interpretácie: $a_i + b_j$ a $ai + bj$. Často býva z kontextu zrejmé, ktorá z nich je tá správna, no pri formalizácii problému sa nemožno spoliehať na nejaký externý kontext.

Riešenie tohoto problému možno nájsť v syntaktickej rovine - v jednoznačnom pomenovaní štruktúr zápisu[2] [3].

2.1.2 Linearizácia

Štruktúra matematických výrazov je pomerne komplikovaná – pri budovaní výrazu možno používať kompozíciu, „neskorú definíciu“[3], premenné, rekurzívne vnorené štruktúry (napr. zlomky) a iné. Problém nastáva, ak je nutné tieto štruktúry prečítať – zo štruktúrovaného zápisu treba urobiť jednu súvislú vetu (resp. jednoriadkový zápis). Tento proces nazvem linearizácia. V niektorých zložitejších prípadoch sa môže stať, že jednoducho stratíme pojem o štruktúre. Preto je nutné navrhnúť spôsob, ako linearizovať efektívne. Tu sa dajú uplatniť rôzne konvencie, nie všetky sú však rovnako prehľadné a jednoduché.

2.1.3 Navigácia

Navigovať v matematických výrazoch je v podstate možné dvomi spôsobmi: v pôvodných štruktúrach alebo v ich linearizovanom tvare. Štruktúrovaný tvar má

[2] napríklad dolný index
[3] výraz sa definuje až po jeho použití, napr. „pre všetky $|\mathbf{A}| \neq 0$, kde $\mathbf{A}$ je matica súradníc...“

výhodu vo vizuálnej prehľadnosti, no identifikovanie (jednoznačné určenie) jednotlivých elementov výrazu je pomerne zložité. Riadkový tvar naopak postráda prehľadnosť (pre množstvo vnorených zátvoriek), no objekty sa dajú jednoducho očíslovať a teda okamžite identifikovať. Tiež je potrebné zvoliť kompromis medzi počtom atomických štruktúr[4] (ktoré vzniknú pri linearizácii) a prehľadnosťou. Ideálnym riešením sa zdá byť hierarchický prístup. Tento prístup je založený na sledovaní úrovní v matematickej štruktúre, jednotlivé bloky (časti výrazu) v rovnakej úrovni sa lineárne očíslujú. Každý blok je možné expandovať a tým vytvoriť očíslovanie v ďalšej úrovni. Opakovaním tohto postupu sa dá hierarchicky každý element jednoznačne identifikovať, pričom je zachovaná prehľadnosť i rozumný počet krokov.

2.2 Diktovanie

S interaktívnym diktovaním súvisia v zásade štyri problémy:

- rozpoznávanie reči
- gramatiky a riadenie rozpoznávania
- spôsob reprezentácie výstupu
- voľba vhodného spôsobu riadenia dialógu a navigácie vo výstupe

2.2.1 Rozpoznávanie reči

História

Vývoj rozpoznávania reči [4] bol ako vo väčšine prípadov spojený s vývojom technológií a výskumom z iných vedných oblastí (fyzika, matematika, akustika, fonetika...). Už koncom 18. storočia sa ľudia pokúšali zostrojiť mechanický syntetizátor ľudského hlasu. Neskôr sa objavili práce podrobne analyzujúce ľudský rečový trakt, ktoré dali základy moderným výpočtovým metódam ako napr. LPC (linear prediction coding), ktoré sa snažia popísať hlasovú vzorku pomocou sady vektorov, a tak minimalizovať množstvo potrebnej informácie pri zachovaní charakteristík vzorky.

Dôležitým míľnikom vo vývoji bolo použitie digitálneho spracovania analógového signálu, ktorý podnietil vznik množstva rôznych prístupov ku klasifikácií vzoriek, akými sú napríklad DTW (dynamic time warping), ktorého hlavnou výhodou je schopnosť kompenzovať časové rozdiely porovnávaných vzoriek, či HMM (hidden markov model - skryté markovove modely), ktoré sa pre svoju univerzálnosť stali jadrom dnešných

[4] podvýraz, ktorý sa už ďalej nebude linearizovať, ani deliť na menšie podvýrazy

rozpoznávačov. Neskôr sa tieto techniky zdokonalili a z niekoľko slovných spojení sa prešlo na súvislú reč.

Skryté markovove modely

HMM je model založený na pravdepodobnostnom princípe, čo mu dodáva veľkú výhodu oproti exaktným výpočtovým modelom, podobne ako je to u človeka: niekedy nemusí porozumieť úplne všetko, no „domyslí si“. Preto sa ľahšie vyrovnáva s nedostatočnou či zašumenou vstupnou informáciou.

HMM možno formálne zapísať ako

$$G=(Q, V, \mathbf{N}, \mathbf{M}, \boldsymbol{\Pi})$$

kde

- $Q=\{q_{1,}\ldots, q_N\}$ sú stavy modelu
- $V=\{v_{1,}\ldots, v_L\}$ sú indexy do kódovej knihy
- $\mathbf{N}=[n_{ij}]$ je matica pravdepodobnosti prechodu zo stavu q_i v čase t do stavu q_j v čase $t+1$
- $\mathbf{M}=[m_{jl}]$ je matica pravdepodobností generovaných vzorov, jej prvky určujú, s akou pravdepodobnosťou je v čase t a stave q_j vygenerovaná l-tá položka vzorov.

HMM si možno predstaviť ako konečný automat, ktorý na prechod medzi stavmi používa pravdepodobnosť a miesto čítania vstupu generuje pri prechodoch indexy do kódovej knihy. HMM umožňujú zreťazovanie, čo implikuje ich univerzálnosť. V praxi sa totiž používajú modely pre fonémové alebo subfonémové jednotky reči. Potom možno ľubovoľné slovo „vyskladať“ zreťazením jeho fonémových modelov. Ako príklad možno uviesť slovo „dictation“. Najprv je nutné v slove identifikovať jednotlivé fonémy, v angličtine sa na to dajú použiť rôzne prepisovacie programy, ktoré tento proces zjednodušia. Fonetický zápis spomínaného slova je „dɪktɛɪʃɨn“. Jednotlivé grafické reprezentácie foném sú štandardizované medzinárodnou organizáciou IPA (International Phonetic Association) pod rovnomennou skratkou IPA (International Phonetic Alphabet).

Ak máme teda sadu slov, ktoré chceme rozpoznávať, vytvoríme si ku každému z nich zreťazený model. Danú výpoveď potom kvantizujeme, oindexujeme pomocou kódovej knihy a zistíme, ktorý model ju generuje s najvyššou pravdepodobnosťou.

HMM sú v dnešnej dobe jedinou rozumnou voľbou pre univerzálny rozpoznávač. Táto skutočnosť je dostatočne zrejmá aj v praxi, väčšina moderných rozpoznávačov je totiž založená práve na nich. Jeden z takých tiež využijem aj v tejto práci.

2.2.2 Gramatiky

Schopnosť rozpoznávať ľubovoľnú (vopred zadanú) množinu slov je síce veľmi užitočná, no v praxi je potrebné rozpoznávať zložitejšie štruktúry, ako sú slovné spojenia či vety. Jednou možnosťou by bolo určiť množinu viet, ktoré treba rozpoznať, čo by však viedlo ku veľkým problémom, pretože z n slov možno teoreticky poskladať až n^k rôznych viet dĺžky k. Keď sa však zamyslíme nad štruktúrou bežných viet, zisťujeme, že sú štruktúrované a teda ich možno efektívne popísať gramatikami.

Úlohou gramatiky je efektívnym spôsobom striktne určiť množinu viet, ktoré bude rozpoznávač akceptovať. Samotná gramatika popisuje štruktúru platnej vety najčastejšie tak, že určuje spôsob rozvoja všeobecných vetných foriem na sled atomických elementov (terminálov). Pri rozpoznávaní reči sa najčastejšie používajú regulárne, lineárne či bezkontextové gramatiky.

Každú gramatiku možno rozvinúť do stromu odvodenia. Každá vetva stromu znamená jednu akceptovateľnú výpoveď. Keďže HMM možno efektívne reťaziť, je možné pre každú vetvu stromu vytvoriť postupnosť HMM, ktorá bude reprezentovať celú vetvu. Počet vetiev v strome závisí hlavne od počtu neterminálov na pravej strane pravidiel. Ak je tento počet vysoký, potom na pokrytie celej gramatiky stromu potrebujeme príliš veľa postupností modelov, čo je pri výpočtoch neúnosné. Naviac možno povoliť aj potenciálne nekonečné vety (jednoduchým pravidlom majúcim rovnaký neterminál na ľavej aj pravej strane), teda pokrytie takejto vetvy je nemožné.

V praxi sa tento problém rieši dynamickým vytváraním a prehľadávaním postupností na „okolí práve aktívneho vrchola" stromu a orezávaním nepravdepodobných vetiev (a s nimi celých podstromov). Dnešné rozpoznávače sú už dostatočne prepracované a s únosnými výpočtovými nárokmi, takže je možné rozpoznávať v reálnom čase aj dostatočne zložité vety.

Vo svojej práci som využil gramatiky vo formáte JSGF (JSpeech Grammar Format) [5]. Ich výhodou je prehľadnosť a jednoduchá integrácia sémantiky. Za nevýhodu možno považovať ich slabú popisnú silu (sú iba regulárne), čo však v tomto prípade neprekážalo.

2.2.3 Reprezentácia výstupu

Nadiktovaný výraz je potrebné reprezentovať. Program poskytuje grafickú interpretáciu matematického vzťahu využitím zásuvného modulu do prehliadača. Interne je využívaný MathML (Mathematical Markup Language) [6], ktorý umožňuje reprezentovať výrazy zapísaním v prezentačnej (bez sémantiky) alebo v obsahovej forme (funkcie a operátory majú daný sémantický význam). Zápis v MathML nielenže poskytuje dáta pre zásuvný modul, ale slúži aj ako jazyk pre export výslednej formuly. V dobe písania tejto práce bola prezentačná forma lepšie podporovaná, preto sa exportuje do nej.

Na zvukovú reprezentáciu výrazov som nenašiel žiaden riadne písomne definovaný štandard čítania. Na túto tému však existuje niekoľko článkov a prác (napr. [7]), ktoré vychádzajú z hovorovo zaužívaného spôsobu čítania štruktúr. Je však nutné dbať na to, aby mal výstup čo najjednoduchšiu formu a zároveň jednoznačnú interpretáciu. To sa dosahuje pridávaním rôznych intonačných či prozodických prvkov. V prípade tejto práce bolo využité SSML (Speech Synthesis Markup Language) [8].

2.2.4 Riadenie dialógu

Najdôležitejšou časťou aplikácie je systém na riadenie dialógu. Tento systém zabezpečuje životný cyklus aplikácie. V podstate ide o vykonávanie akcií programu podľa kontextu – teda podľa toho, čo sa práve diktuje. Cyklus sa opakuje po celú dobu behu aplikácie a pozostáva z nasledujúcich krokov systému:

- získa hlasový vstup od používateľa
- vyhodnotí ho a overí platnosť v danom kontexte
- vykoná adekvátne akcie (produkuje príslušný výstup, zmaže element...)
- určí nový kontext

Technologicky sú prvé dva kroky realizované v jazyku VoiceXML (Voice Extensible Markup Language) [9], presnejšie využitím technológie X+V (XHTML+Voice) [10], ktorá integruje hlasovú interakciu do XHTML (Extensible HyperText Markup Language) [11] stránok. Jej praktické využitie v mojej práci je, že jednoduchá XHTML stránka slúži ako vizuálne rozhranie, ktoré umožňuje diktovanie a navigáciu pomocou hlasu. Po každom cykle sa na stránke prostredníctvom servera vykonajú príslušné vizuálne úpravy a pokračuje sa ďalej v interakcií.

3 Analýza

3.1 Analýza požiadaviek

Je potrebné vytvoriť aplikáciu, ktorá umožní zadať stredne komplikovanú[5] matematickú formulu. Spôsob zadávania má byť umožnený manuálne (myš) i hlasom. Výstup aplikácie musí byť multimodálny.

V zásade sa možno na aplikáciu pozerať z dvoch hľadísk, ktoré pomôžu lepšie sformulovať funkčné požiadavky. Tieto hľadiská sú použiteľnosť a prístupnosť.

Použiteľnosť

Pojem použiteľnosti popisuje, či je možné danou aplikáciou dosiahnuť želaný cieľ. Je dôležité, aby aplikácia obsahovala všetky potrebné funkcie. Rovnako dôležité je však aj to, aby neobsahovala funkcie, ktoré sú nepotrebné. Takéto funkcie zbytočne zväčšujú množinu fráz, ktoré musí aplikácia rozpoznávať, čím sa samozrejme znižuje presnosť. Z týchto dvoch vecí vyplýva, že je nutné nájsť čo najmenšiu množinu takých funkcií, ktoré spolu umožnia vykonať všetky požadované akcie.

Prístupnosť

Pod prístupnosťou sa myslí schopnosť jednoducho a intuitívne vykonať želanú akciu. Tomu napomáha prehľadné používateľské rozhranie (UI[6]). Keďže UI „obaľuje" funkcionalitu, je v ňom skrytá požiadavka optimálnosti. Prehľadné UI poskytuje prístup práve k optimálnej množine funkcií.

Pre vizuálne prvky sú dôležité poloha, veľkosť a vzhľad. Tieto symbolizujú rôzne kategórie a dôležitosť danej funkcie. Pre hlasové vstupy sú zasa dôležité dobre navrhnuté a racionálne kategorizované gramatiky, ktoré majú čo najlepšie kopírovať zaužívané frázy a pritom zaručiť jednoznačnú interpretáciu výpovede (2.1.1).

Takisto treba dbať na to, že má byť aplikácia prístupná aj zrakovo postihnutým. Preto je nutné, aby bol hlasový výstup ekvivalentný grafickému, t. j. s rovnakou výpovednou hodnotou.

[5] podpora známych a zaužívaných syntaktických prvkov, nie špeciálnych
[6] z anglického user interface

3.1.1 Funkčné požiadavky

Nasledujú funkčné požiadavky, ktoré špecifikujú očakávania kladené na program. Takisto sú tu spomenuté aj požiadavky na spôsob realizácie funkcionality. Okrem iného prihliadajú aj na horeuvedenú prístupnosť a použiteľnosť:

- Vstup manuálne aj s použitím hlasu. K vizuálnym elementoch UI vytvoriť hlasový ekvivalent a čo najviac zjednotiť text elementu s hlasovým vstupom.
- Možnosť vytvárania formuly, navigovania v nej a takisto možnosť jednoduchých úprav.
- Rozdelenie funkcionality do kategórií. Zoskupenie funkcií, ktoré spolu súvisia a oddelenie od ostatných skupín. Možno sformulovať tieto kategórie:
  - ◇ Základné funkcie programu: nová formula, ukončenie, export finálnej formuly
  - ◇ Vkladanie blokov formuly: jednotlivé bloky možno tiež kategorizovať, napríklad funkcie, operátory, zátvorky, zlomky,...
  - ◇ Navigácia po blokoch formuly: navigácia musí byť jednoduchá, musí poskytovať intuitívny spôsob identifikácie ľubovoľného elementu. Najlepšie sa tu uplatní hierarchický prístup (2.1.3).
  - ◇ Čítanie blokov formuly: vzhľadom na prístupnosť pre zrakovo postihnutých je nutné, aby bol výstup stručný a pritom výstižný. K tomu pomôže aj využitie prozodických vlastností, ako sú napríklad zmena výšky hlasu či vloženie prestávok.
- Možnosť exportovať vytvorenú formulu. Nie je požadované vedieť importovať, pretože primárnym cieľom programu je byť pomocným nástrojom pri vkladaní formúl do iných aplikácií. Vhodným by bolo zobrazovanie formuly počas jej tvorby v „klasickom" zápise, na aký je zvyknutý matematický svet.
- Konfigurovateľnosť a rozšíriteľnosť: všetky nastavenia musia byť uložené v externých súboroch, kde budú ľahko dostupné. Keďže formula sa skladá z blokov rozličných kategórií, tieto kategórie musia byť rozšíriteľné, napríklad musí byť možné rozširovať zoznam operátorov pridávaním nových. Vhodná je tiež možnosť upravovať a nastavovať samotné UI.

3.1.2 Využitie štandardov

Používanie overených štandardov pomôže zlepšiť kvalitu aplikácie (znížiť chybovosť, zvýšiť presnosť rozpoznávania reči) a zabezpečí interoperabilitu. Štandardnosť sa týka nasledovných častí:

- Vizuálne rozhranie: je potrebné ho navrhnúť tak, aby sa vzhľadom i správaním podobalo na dnes bežné desktopové aplikácie, nezávisle na platforme.
- Zvukové rozhranie: zvukový výstup sa musí generovať buď ako zvuk samotný, alebo ako zápis frázy pre hlasový syntetizátor. Keďže daná fráza môže obsahovať formátovanie, zápis by mal byť štandardizovaný.
- Formát finálneho výstupu: finálna formula bude používaná v iných aplikáciach, preto je nutné robiť export do štandardizovaného jazyka.

3.2 Práca s aplikáciou

Po definovaní požiadaviek je ďalšou dôležitou časťou definovanie spôsobu práce s aplikáciou. V prvom rade je dôraz kladený na intuitívnu prácu. Tento vágny pojem zhruba znamená snahu o to, aby sa používateľ nemusel príliš zaoberať tým, „ako niečo spraviť", ale aby mu to bolo viac-menej po niekoľkých skúsenostiach zrejmé. Tento pojem je však nutné pre lepšie pochopenie ďalej analyzovať.

Vyššie spomínaná použiteľnosť a prístupnosť pozitívne ovplyvnia intuitívnosť ovládania, avšak nezaručujú nutne aj intuitívnu prácu. Príkladom môže byť fakt, že optimálne navrhnuté UI síce poskytuje prehľadné grafické prvky umožňujúce vykonávať optimálnu sadu akcií, avšak vôbec nemusí byť zrejmé, ako danú sadu vybrať a v akom poradí akcie vykonávať.

3.2.1 Módy práce

Keďže primárnou funkciu aplikácie je tvorba formuly, hlavným módom práce bude vkladací mód. Tento mód musí umožniť za sebou vkladať jednotlivé bloky formuly, pričom musí dbať na zachovanie správnej štruktúry (napríklad vnorenie zátvoriek). Pri vkladaní objektu je vhodné vložiť aj špecifické informácie s ním spojené, napríklad pri vkladaní zlomku určenie toho, čo je čitateľ a čo menovateľ. Štruktúry rozličných typov (funkcie, zlomky,...) je dobré vizuálne odlíšiť.

Ďalším módom práce bude navigačný mód, ktorý umožní vybratie zvoleného bloku či elementu[7]. S vybratým elementom sa potom bude dať manipulovať – bude ho možnosť zmeniť, odstrániť, či prípadne doplniť. Systém sa musí postarať, aby po prípadnej zmene či odstránení ostali zachované syntaktické vlastnosti formuly, napríklad binárne operátory musia spájať práve dva operandy a odstránenie jedného z nich bez príslušných úprav má za následok nekorektnú formulu.

[7] blok je postupnosť elementov, element je atomická časť formuly (konštanta, operátor...)

3.3 Reprezentácia formuly

Ako už bolo spomenuté v 2.1.3, matematické formuly možno v zásade reprezentovať dvoma spôsobmi: v štruktúrovanom alebo linearizovanom tvare. Prvý spôsob je prirodzený pre vizuálny spôsob zápisu, druhý sa prirodzene používa pri diktovaní. Keďže aplikácia má byť multimodálna, je nutné zvoliť niektorý z horeuvedených spôsobov ako spoločný.

Ak zoberieme do úvahy spomínané výhody a nevýhody každého z nich, dospejeme k hybridu: hierarchicky číslovaný lineárny tvar (HČLT). Formula v tomto tvare je v každej úrovni rozdelená na sekvenciu blokov, oddelených binárnym operátorom so strednou prioritou[8]. Táto sekvencia má lineárne číslovanie, teda každý blok sa dá v danej úrovni identifikovať na jeden krok. Expandovanie bloku je jeho rozbitie na podbloky ďalšej úrovne podobným spôsobom. Expandovať možno len bloky, ktoré obsahujú operátor, zátvorky, funkčný symbol alebo inú podobnú štruktúru spôsobujúcu zoskupenie. Úroveň bloku je definovaná ako počet expanzií potrebných na jeho očíslovanie.

Pre lepšie pochopenie uvažujme nasledujúcu formulu:

$$a+2b-\sin(3a-4b)=0$$

Podľa vyššie uvedeného popisu urobme rozbitie formuly na bloky prvej úrovne. Dostaneme formulu v nasledovnom HČLT :

a	$2b$	$\sin(3a-4b)$	0
1	2	3	4

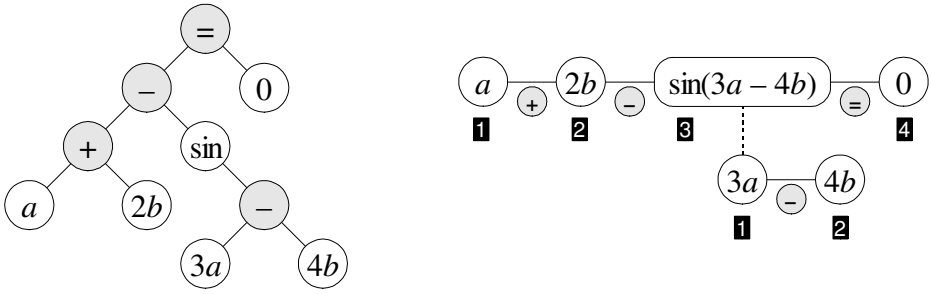
Zlom nastal v „+”, „–” resp. „=”, pretože sú to operátory s menšou resp. nízkou prioritou. Bloky sa teraz dajú sekvenčne očíslovať, ako je to znázornené pod nimi. Ako si možno všimnúť, bloky 1 a 4 sa už nedajú expandovať, pretože sú to atomické elementy. Blok 2 sa dá expandovať na „2” a „*b*”, pretože obsahuje operátor neviditeľného násobenia. Pre zostručnenie sa expanziou podľa neviditeľného násobenia nebudeme ďalej zaoberať. Zamerajme sa teda na blok 3, ktorý sa dá úspešne expandovať na bloky druhej úrovne:

3a	4b
1	2

Tento postup sa dá robiť aj na strome operandov (SO), čo je strom, ktorý vo vrcholoch obsahuje operandy a hrany má označené operátormi. SO sa dá vytvoriť

[8] priorita menšia alebo rovná priorite operátora +

z aritmetického stromu (AS) jednoduchým postupom. AS a SO pre uvedenú rovnicu vyzerajú nasledovne:



Obrázok 1: Aritmetický strom a strom operandov pre uvedenú rovnicu

Sivou farbou sú označené operátory, prerušovanou hranou expanzia bloku. Ako vidno, postupným „rozptylom" sivých vrcholov AS do hrán dostávame z AS naľavo SO napravo. Úroveň bloku zodpovedá úrovni v SO a bloky v danej úrovni sa číslujú postupne, ako je to znázornené číslami v čiernych obdĺžnikoch.

Dôležitou vlastnosťou aplikácie je schopnosť dobrej navigácie – označenie[9] zvoleného elementu. Môžeme sformulovať jednoduchý postup, ako označiť zvolený element: v každej úrovni sa najprv urobí očíslovanie. Potom sa zistí číslo bloku, v ktorom sa element nachádza a daný blok sa expanduje. Toto sa opakuje, pokým element nedostane číslo (je ho možné označiť udaním tohto čísla). Z postupu je zrejmé, že na označenie ľubovoľného elementu potrebujeme k krokov, pričom k je jeho úroveň. Zapisovaním čísel blokov, cez ktoré sa prechádzalo, dostávame k-prvkovú navigačnú postupnosť. Napríklad pre element „4" je táto postupnosť $\{3,2,1\}$.

[9] vizuálne označenie v aplikácii

4 Návrh

V predošlej kapitole boli sformulované požiadavky kladené na aplikáciu a urobená analýza práce a reprezentácie matematickej formuly. V tejto časti predložím návrh, ako uvedené požiadavky splniť z technologického hľadiska, sformulujem možné riešenia problémov a načrtnem základné algoritmy a techniky.

4.1 Technológie a architektúra

Ako už bolo spomínané v 2.2, potreba hlasového vstupu resp. výstupu vyžaduje moduly na rozpoznávanie reči resp. jej syntézu. Interaktívna hlasová aplikácia navyše musí obsahovať modul na riadenie dialógu, pretože je nutné používateľa informovať o stave, vyzvať k akcii, či prečítať výstupy. Riadenie dialógu umožňuje kontrolovať poradie akcií, čím sa dosahuje väčšia spoľahlivosť a zmenšuje dezorientácia používateľa. Požiadavkou je, aby tieto moduly resp. ich implementácie boli založené na existujúcich štandardoch. Je preto vhodné hneď na začiatku zvoliť technológie, na ktorých bude aplikácia postavená, aby sa mohlo stavať na dobre definovaných základoch a určili sa technologické hranice.

Pre rozpoznávač reči existuje niekoľko typových alternatív. Prvým typom je rozpoznávač dopredu definovaných slovných spojení. Tento má však veľké nevýhody, pretože striktne obmedzuje množinu akceptovaných viet. Pri diktovaní matematiky by to nemuselo robiť problém, pokiaľ by sa dopredu určilo, čo všetko sa bude dať nadiktovať. Tým by sa ale porušila požiadavka rozšíriteľnosti programu. Preto jedinou reálnou voľbou je využitie systémov založených na HMM, u ktorých tento problém nie je.

Existuje viacero použiteľných implementácií rozpoznávačov reči, fungujúcich ako pomocné knižnice (Sphinx-4 [12], HTK [13]). Rovnako existuje aj množstvo hotových rečových syntetizátorov (Festival [14], MBROLA [15]). Použitím nezávislých modulov však vniká problém dobrej kooperácie. Ten by sa dal riešiť vybudovaním rozhraní nad nimi a centralizovaním spracovávania v riadiacej aplikácii. K týmto modulom by pribudol ďalší, ktorý by mal na starosti grafický výstup a ovládanie vizuálnych prvkov. Navyše, tento modul by sa musel nejakým spôsobom synchronizovať s modulom na rozpoznávanie reči, aby sa hlasové príkazy prejavili aj v UI. Zdá sa, že takáto „skladačka" by síce zaručovala vysokú kvalitu každého modulu, no zhoršovala by prehľadnosť riadiaceho jadra a vynucovala písanie rozhraní a ďalšieho režijného kódu.

Ideálnym riešením tohto problému sa ukázal prehliadač *Opera* [16]. Ten v sebe zahŕňa moduly na rozpoznávanie a syntézu reči a jeho veľkou výhodou je podpora

technológie X+V ([10], [17]), ktorá zabezpečuje jednoduché riadenie dialógu. Ako prehliadač sa dá navyše veľmi jednoducho použiť ako UI, čo je dnes bežný postup pre webovské aplikácie. Využitím prehliadača Opera odpadá písanie rozhraní a režijného kódu a využitím technológie X+V je zabezpečená aj synchronizácia hlasovej a vizuálnej časti.

Využitie webovského prehliadača (čiže klienta) automaticky implikuje potrebu servera – ide teda o architektúru klient-server. V súčasnosti prichádza do úvahy použitie buď PHP, alebo Java serveru. Vzhľadom na moje preferencie som sa rozhodol použiť aplikačný Java server *Jetty* [18], ktorý sa zdal pre túto aplikáciu vhodný, pretože je dostatočne kompaktný a poskytuje všetky potrebné služby.

4.2 Klient

4.2.1 Používateľské rozhranie

Väčšina požiadaviek súvisiacich s funkcionalitou sa priamo odráža na UI, preto sa nasledujúca časť sústredí na jeho návrh aj s popisom funkcionality jednotlivých prvkov. Pretože sa na vizuálnu interakciu bude používať webovský prehliadač s podporou X+V, rozhranie bude písané v XHTML. Tento štandard umožní využitie niektorých postupov pri tvorbe dynamických stránok, avšak je potrebné sformulovať jedno obmedzenie. Dynamické stránky sú založené na stále novom generovaní ich kódu, čo síce umožňuje vysokú flexibilitu, no znižuje rýchlosť a spôsobuje známy efekt „prekresľovania" stránky. Pretože počet a rozmiestnenie komponentov UI bude nemenné, je zbytočné (ba dokonca nežiadúce) po každej komunikácii so serverom stránku obnovovať. Dáta na stránke však obnovovať treba, preto je nutné používať na zmenu dát skripty alebo vložené rámce. Týmto spôsobom sa zachová pocit spojitosti a pritom sa umožní zmena.

Menu

Požiadavka rozdelenia funkcionality do kategórií sa ľahko realizuje použitím menu. Keďže HTML štandard nepodporuje menu komponent, menu aplikácie bude urobené v jazyku JavaScript. Menu by malo pokrývať spomínané kategórie, teda na prvej úrovni možno navrhnúť nasledujúce položky:

- *Program*: príkazy základných funkcií programu (nová formula, export,...)
- *Command*: všeobecné príkazy (krok späť, zmena módu)
- *Function*: vkladanie funkcií
- *Operator*: vkladanie operátorov

- *Structure*: vkladanie štruktúr
- *Symbol*: zoznam rôznych symbolov
- *IP*: manipulácie s „Insert point", čiže bodom vkladania. Bod vkladania je miesto, na ktoré sa vo vkladacom móde do formuly pridávajú zadané bloky. Jeden bod vkladania je vždy na konci, ostatné môžu byť napríklad v miestach parametrov funkcie.
- *Selection*: manipulácie s výberom v navigačnom móde
- *Say*: čítanie rôznych častí (výberu, blokov, formuly,...)
- *Search*: rýchle nájdenie a vybratie (označenie) danej štruktúry (napr. funkcie)

Matematický riadok a náhľad

Podľa požiadavky aplikácia musí obsahovať miesto, kde sa bude zadávaná formula zobrazovať v HČLT. Toto miesto nazvem matematickým riadkom (MR). Keďže HČLT je vlastne číslovaná sekvencia blokov, možno formulu v tomto tvare reprezentovať tabuľkou, kde v prvom riadku budú bloky a v druhom ich očíslovanie pre danú úroveň, ako to bolo naznačené v 3.3. Na odlíšenie kategórií elementov sa využijú rôzne farby a štýly písma, čím sa zabezpečí lepšia orientácia. Farebne sa tu zvýraznenia aj vybraté elementy a body vkladania. Ďalším vylepšením by bolo zobrazovanie pomenovaní jednotlivých blokov, ktoré by sa zobrazovalo nad nimi. Toto by umožnilo lepšiu vizuálnu orientáciu, a v prípade ovládania hlasom možnosť vyhľadávania a okamžitého vybratia bloku podľa jeho mena.

Okrem MR sa bude na obrazovke ukazovať náhľad zadávanej formuly v používateľsky prívetivej forme, teda v „klasickom" zápise tak, ako ho možno nájsť bežne v literatúre. Na to sa použije zásuvný modul (plugin) TechExplorer [19].

Ostatné panely

Nie všetky funkcie je vhodné umiestňovať do menu. Napríklad písmená a číslice je oveľa pohodlnejšie zobrazovať v tabuľke, ako mať pre každé písmeno položku v menu. Rovnako to platí aj pre často používane operátory, akými sú napríklad „+" alebo „–". Všetky takéto objekty sa umiestnia do pomocných panelov a ich manuálne zadanie bude možné jednoduchým kliknutím na príslušnú položku. Ďalšou výhodou tohto prístupu je, že napríklad pri písmenách je možné hneď vedľa znaku zobraziť jeho slovný ekvivalent, či vedľa symbolu operátora jeho názov.

Okrem panelov, ktoré slúžia na zadávanie, sú potrebné aj informačné panely. V prvom rade je vhodné užívateľa informovať o móde práce. Na ostatné informácie sa

využije dobre známy stavový riadok, ktorý bude zobrazovať informácie, varovania alebo chyby.

Týmto zoznam grafických elementov končí. Akékoľvek ďalšie vizuálne prvky by nepridávali žiadnu užitočnú funkcionalitu, a teda by boli v rozpore s ideou prehľadného a intuitívneho UI.

4.2.2 Hlasové rozhranie

Popri návrhu UI aplikácie je nutné dbať aj na návrh hlasového rozhrania. Spôsob generovania je výberom spomínaných technológií vcelku jednoznačne určený. VoiceXML definuje jednoduchý spôsob hlasovej interakcie s používateľom. Jeho využitie uľahčuje získavanie vstupu aj generovanie výstupu. VoiceXML stránku je nutné po každom cykle aplikácie obnoviť (aktualizovať výzvy, vybrať príslušné gramatiky platné pre aktuálny kontext), preto sa umiestni do vnoreného rámca. Príklad na VoiceXML formulár je v prílohe č. 1.

Vstup

Hlasovým vstupom sa rozumie zvuk prichádzajúci do mikrofónu. Tento zvuk je nutné zachytiť, rozpoznať v ňom slová, a potom posunúť na ďalšie spracovanie. Rozpoznávač reči v prehliadači Opera dokáže rozpoznávať ľubovoľnú zadanú množinu fráz. Táto množina sa určí prostredníctvom gramatík, ja som si zvolil podporovaný JSGF [5]. Príklad na gramatiku v tomto formáte môžno nájsť v prílohe č. 2.

Gramatiky

Analýza toho, čo sa má „dať nadiktovať" a zostavenie potrebných gramatík je najnáročnejšou časťou pri tvorbe hlasového rozhrania. Počas behu aplikácie je možné mať k dispozícii viacero aktívnych gramatík, čo umožňuje roztriediť funkcionalitu do kategórii a oddeliť frázy pre zadávanie a manipuláciu s dátami do samostatných gramatík.

Gramatiky pre manipuláciu sa dajú navrhnúť relatívne priamočiaro, základom je nejaké kľúčové slovo (napríklad *read*) a potom upresnenie akcie (napríklad *selection*). Tieto gramatiky sú veľmi jednoduché a nie je potrebné im v tejto fáze venovať príliš veľa pozornosti.

Vzhľadom na požiadavku čo najjednoduchšieho použitia sa treba hlbšie zamyslieť nad gramatikami, ktoré zadávajú dáta. Bolo by jednoduché urobiť gramatiku pre každý atomický typ, ako sú čísla, písmená, operátory, a pod. To by však veľmi spomalilo zadávanie, pretože naraz by sa dal rozpoznať (a vložiť) len jeden element. V praktickej

hovorenej matematike však väčšinou zoskupujeme elementy do väčších logických blokov a tie od seba oddeľujeme pauzami. Preto by bolo dobré tieto celky sformulovať a navrhnúť gramatiku, ktorá bude v rámci možností zachovávať toto delenie.

Zamerajme sa teda najprv na jednoduché výrazy, konkrétne na lineárne (LV). LV vo všeobecnosti najčastejšie pozostáva zo sekvencie skupín oddelených binárnym operátorom. Skupinu tvorí väčšinou konštanta a niekoľko premenných (písmen latinky) za sebou. Keď vezmeme do úvahy najbežnejšie binárne operátory (+, -, /, *, =), môžeme definovať typ bloku *výraz* nasledovne:

<*výraz*> = <*skupina*> (<*operátor*> <*skupina*>)*

<*skupina*> = (<*konštanta*> <*symboly*>) | <*konštanta*> | <*symboly*>

<*symboly*> = (<*symbol*>)+

V definícii je použitá zaužívaná syntax, „*" a „+" označujú uzávery, „|" označuje alternatívu, zátvorky zoskupenie. Takto definovaný LV umožňuje zadávať jednoduché a často sa vyskytujúce sekvencie relatívne pohodlne. V praxi sa však väčšinou stáva, že výrazy nie sú lineárne, ale polynomické, pričom mocnina nemusí byť nutne konštanta. Preto je múdre rozšíriť *symbol* o jednoprvkový horný index. Podobne často sa v praxi vyskytujú aj postupnosti, kde má zase význam dolný index. Dostávame definíciu prvku *symbol*:

<*symbol*> = <*latinka*> [<*horný_index*> | <*dolný_index*>]

Hranaté zátvorky označujú nepovinný prvok. Takto definovaná gramatika umožňuje vyjadriť jednoduché, bežne používané výrazy, ako sú napríklad:

$$a^2+b^2=c^2 \qquad 3a_1+5a_2+4a_3 \qquad 3x^4+5{,}4x^3-12{,}7x^2-16x+24{,}6=0$$

Pozrime sa teraz na komplexnosť tejto gramatiky z hľadiska rozpoznávania. Dôležitý je hlavne počet prvkov jednotlivých „terminálnych" množín, v tomto prípade *konštanta*, *latinka*, *grécky_znak* a *operátor*. Už pri pohľade na konštanty vieme povedať, že táto množina je nekonečná, dokonca nespočítateľná. Ďalšou veľkou nevýhodou je zaužívané čítanie čísel, napríklad číslo *2 567 874* sa prečíta ako „*two milion five hundered sixty seven thousand eight hundered seventy four*", čo je na prvý pohľad absolútne nevhodné pre rozpoznávanie. Jedným riešením je zakázať „príliš veľké" čísla, no oveľa prípustnejšie je sa s problémom vyrovnať spôsobom diktovania čísla po cifrách. Gramatika, ktorá to umožní, je veľmi jednoduchá – iterovaný výber z 10 možností. Pritom sa nekladú žiadne obmedzenia na veľkosť čísla.

Po vyriešení problému s číslami sa dostávame k ďalšiemu praktickému problému pri rozpoznávaní, ktorým je zvuková podobnosť výpovedí. Anglická výslovnosť znaku „*a*" sa od výslovnosti „*8*" líši iba koncovým „*t*", ktoré niekedy v kontexte splýva

s nasledujúcim slovom. Podobne pri nekvalitnom mikrofóne či neporiadnom vyslovovaní sa ťažko rozlišujú znelé a neznelé dvojice hlások (napríklad „*d*"/„*ť*" a „*b*"/„*p*"). Preto sa môže stať, že pri diktovaní „*a plus b*" môže počítač porozumieť „*eight plus p*". Tomuto problému sa dá predísť zaužívaným spôsobom – pre každú hlásku sa zavedie slovo, ktoré ju bude zastupovať – vytvorí sa takzvaná fonetická abeceda. Asi najznámejšou takouto abecedou je NATO phonetic alphabet [20]. Ja som sa rozhodol pre použitie vlastnej sady slov, pretože sa mi po praktických pokusoch zdala foneticky lepšia. Tá sa však dá ľahko zmeniť v konfiguračnom súbore podľa prianí používateľa.

Rozoznávanie hlások od čísel možno zlepšiť využitím malého triku: namiesto diktovania samostatného „*a*" je možné použiť frázu „*one a*", čím sa zachová význam a zároveň výrazne zníži pravdepodobnosť zámeny s osmičkou.

I keď spomínaná gramatika napríklad nepokrýva súčasné zadanie horného aj dolného indexu, po analýze gramatiky z hľadiska počtu fráz a dopade tohto počtu na rozpoznávač (s prihliadnutím na praktické skúsenosti) možno usúdiť, že daná gramatika je už dostatočne komplexná a pridávanie ďalších prvkov by bolo na škodu rozpoznávania.

VÝSTUP

Využitie hlasového syntetizátora prehliadača Opera redukuje problém syntézy na vygenerovanie sekvencie slov[10]. Ako už bolo v požiadavkách spomínané, hlasový výstup je nutné formátovať, aby sa tak zvýšila prehľadnosť. Použitím SSML [8] možno dosiahnuť zaujímavé efekty, ktoré znížia komplexnosť výstupu pri zachovanej výpovednej hodnote. Ako príklad sa dá použiť spôsob čítania dolných indexov. a_k je možné čítať ako „*a with lower index k*", avšak oveľa pohodlnejšie je prečítať dolný index nižšie položeným hlasom a zbaviť sa spojenia „*with lower index*". Príklad na text v SSML možno nájsť v prílohe č. 3.

4.2.3 Konfigurácia

Dôležitým prvkom aplikácie je prispôsobiteľnosť potrebám používateľa. Bolo by veľmi nerozumné obmedzovať množinu funkcionality len na programátorom stanovenú. V prípade kategorizovania funkcionality treba stanoviť spôsob, ktoré kategórie a akým spôsobom sa budú dať rozširovať či modifikovať.

[10] slovo ako postupnosť textových znakov, nejde teda o žiadne špeciálne fonetické či iné symboly

Ak vezmeme do úvahy vyššie vymenované kategórie, kandidátmi na modifikovateľnosť sú položky *Function*, *Operator*, *Structure* a *Symbol* umožňujúce vkladať špecifický typ objektu.

Bolo by ideálne, aby používateľ mohol jednoduchou zmenou konfiguračného súboru pridať napríklad novú funkciu a hneď ju používať. Je však nutné najprv definovať atribúty každej kategórie, aby bolo zrejmé, ako má pridávaný záznam vyzerať. Možno sformulovať tri atribúty spoločné pre všetky kategórie:

- *Symbol*: záznam možno jednoznačne identifikovať symbolom, ktorý sa zobrazí v matematickom riadku i v náhľade.
- *Popis*: text, ktorým bude zobrazený v menu.
- *Slovo*: slovo (slovné spojenie), ktorým bude vyslovený. Je žiadúce, aby bol atribút *slovo* čo najviac podobný atribútu *popis*, ako to bolo sformulované v požiadavkách.

Každá kategória môže mať ešte svoje špecifické atribúty, napríklad funkciu charakterizuje okrem mena aj počet parametrov, operátor má definovanú prioritu, štruktúra počet parametrov a spôsob ich umiestnenia.

4.2.4 Formát dát posielaných na server

Dáta, ktoré posiela klient na server, nesú okrem dátovej zložky so sebou aj riadiace informácie, ktoré určujú ich význam. Možno sformulovať tieto atribúty:

- *akcia*: popis akcie, ktorá sa má s dátami vykonať. Je to napríklad vloženie, vyhľadanie, výber...
- *dáta*: samotné dáta získane hlasom alebo priradené k manuálne zvolenej položke
- *typ*: typ dát (vkladanie funkcie, operátora...)

Výber webovského klienta priamo určuje, v akom formáte budú presúvané dáta medzi klientom a serverom: využije sa klasický prístup – webovské formuláre. Ich používanie je zaužívané a ich možnosti postačujúce. Keďže na samotnej obrazovke sa nevyskytne ani jedno formulárové pole a ani tlačidlo, bude dokonca postačovať iba používanie skrytých polí – každé pole bude reprezentovať jeden atribút vstupu. Vypĺňanie týchto polí musí byť umožnené aj hlasovému vstupu, aj manuálne zvolenej vizuálnej položke. Toto zabezpečí globálny skript, na ktorý sa budú obe modality odvolávať a ktorý po naplnení skrytých polí formulár odošle.

Samotné dáta (atribút *dáta*) väčšinou reprezentujú postupnosť elementov formuly, ktoré sa zadávali. Ako bolo popísané, každý element patrí do nejakej kategórie,

čiže okrem hodnoty má aj svoj typ. Tieto dva atribúty tvoria záznam s úplnou informáciou o danom elemente. Dáta budú teda tvoriť postupnosť takýchto záznamov.

4.2.5 Formát dát prijímaných servera

Aplikácia produkuje okrem samotného UI a informačných polí rôzne dátové výstupy. Sú to matematický riadok, výstup v „klasickom" zápise a hlasový výstup. Výsledná formula sa musí dať navyše aj exportovať.

Export

Jednou z požiadaviek na aplikáciu je štandardnosť exportu – čiže štandardnosť formátu, do ktorého sa bude exportovať výsledná formula. Tento štandard by mal mať nielen syntakticky presne definovanú formu, ale aj spätosť s matematikou. Mal by to byť teda jazyk určený práve na zapisovanie matematických formúl. Do úvahy prichádzajú minimálne dve riešenia. Prvé, trochu všeobecné, no dobre známe, je zapisovanie matematických formúl TeX-ovskou syntaxou. Aj keď TeX nie je primárne určený priamo pre písanie matematiky, schopnosť zapisovať výrazy v tomto systéme je vo všeobecnosti dostatočne uspokojivá a hojne využívaná.

Druhým je W3C (World Wide Web Consortium) riešenie – MathML [6]. Ako všetky W3C riešenia, aj toto poskytuje výhody ako sú exaktná definícia, univerzálnosť a rozšírovateľnosť. I keď je zápis v MathML na rozdiel od TeX-ovských zápisov oveľa horšie čitateľný, XML syntax mu dodáva viditeľnú štruktúrovanosť, čo je pri matematických zápisoch obzvlášť dôležité. Príklad na zápis formuly v MathML možno nájsť v prílohe č. 4.

Matematický riadok

MR je tabuľka tvorená postupnosťou očíslovaných blokov. V HTML existuje jednoduchý spôsob zápisu tabuľky, takže sa zdá, že problém má priamočiare riešenie. Ako však bolo uvedené v 4.2.1, je nežiadúce obnovovať hlavnú stránku, čiže všetky zmeny dát sa musia udiať prostredníctvom skriptu. Preto sa budú dáta pre MR posielať ako pole záznamov, ktoré potom vstúpi do metódy skriptu na vizualizáciu.

„Klasický" zápis

Dáta pre klasický zápis sa budú generovať vo formáte MathML, pretože tak ich očakáva zásuvný modul do prehliadača. Modul bude vo vlastnom, vloženom rámci, a tak sa bude môcť aktualizovať nezávisle od aktualizácie hlavnej stránky.

Hlasový výstup

Použitím VoiceXML sa problém generovania hlasového výstupu zredukoval na vygenerovanie textovej vety. Syntetizátor, ktorý používa prehliadač Opera podporuje aj niektoré prvky SSML, preto je možné priamo generovať text aj so SSML značkami.

Pretože sa hlasový výstup generuje až po vykonaní akcie, keď sa už spracoval celý pôvodný VoiceXML formulár a vygeneroval sa nový, je nutné tento výstup dať na začiatok nového formulára ako výzvu.

4.3 Server

Jadrom aplikácie bude server, ktorý bude mať tieto základné funkcie:

- spracovávať vstup
- udržiavať dátovú štruktúru reprezentujúcu zadávanú formulu
- udržiavať stavové informácie behu programu
- generovať výstupy

4.3.1 Spracovanie vstupu a reprezentácia formuly

Dáta sa na server dostanú po tom, čo klient odošle HTML formulár. Ako bolo vyššie uvedené, formulár bude obsahovať len skryté polia mapujúce atribúty vstupu. Server sa na základe nich (napríklad podľa atribútu *akcia*) rozhodne, čo sa bude s dátami ďalej diať.

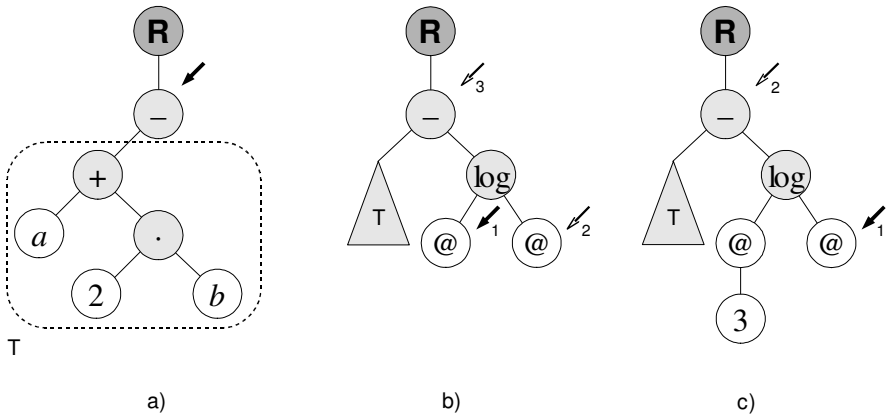
Najčastejšou akciou je vloženie. Pri tomto type akcie dáta reprezentujú postupnosť elementov, ktoré sa majú vložiť. Pretože každý element si okrem hodnoty zo sebou nesie aj svoj typ, je nutné ho identifikovať. Na to poslúži jednoduchá trieda `Parser`, ktorá zo vstupu vyprodukuje pole objektov typu `Token`. Každá podtrieda triedy `Token` zastrešuje jeden typ (kategóriu) elementov. Dajú sa navrhnúť podtriedy `AlphaToken` (písmená), `NumToken` (čísla), `FuncToken` (funkcie), `OperToken`, (operátory), `StructToken` (štruktúry). Vlastnosti každého typu sú odlišné, preto podtriedy prekrývajú zdedené metódy. Tým je zabezpečený polymorfizmus a dodržaný striktný objektový dizajn, čo napomáha jednoduchšiemu spracovaniu.

V 3.3 bol opísaný spôsob reprezentovania doteraz zadanej formuly pomocou aritmetického stromu. Je preto vhodné si vytvoriť programovú reprezentáciu tohto stromu, vstupnú postupnosť elementov možno potom doň použitím určitých pravidiel vložiť, čo sa pokúsim demonštrovať na nasledujúcom príklade.

Uvažujme formulu „$a+2b-\log_3\left(\frac{c}{4d}+e\right)$”. Predpokladajme, že už máme zadané „$a+2b-$”, ostáva teda ešte zadať „$\log_3\left(\frac{c}{4d}+e\right)$”. AS pre zadanú časť je na obrázku 2a). „**R**” označuje koreň, šípky body vkladania, plná šípka práve aktívny bod.

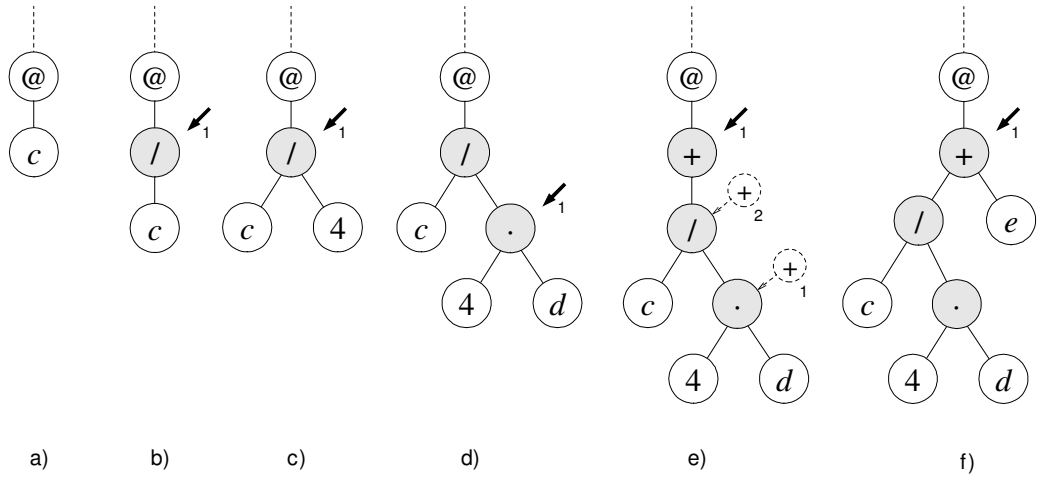
Teraz nasleduje vloženie funkcie „log”. Pri vkladaní funkcie sa vloží aj toľko parametrových vrcholov „@”, koľko má funkcia parametrov, v tomto prípade dva – jeden je základ logaritmu a druhý jeho hodnota. Ako si možno všimnúť, šípka sa presunula na funkčný vrchol (Obrázok 2b)

Ďalej sa vloží základ logaritmu. Po vložení „3” sa musí parameter uzavrieť, aby sa doň už nepridávalo. Bod vloženia sa odstráni a aktivuje sa nasledujúci (Obrázok 2c).



Obrázok 2: Vkladanie elementov do existujúceho AS

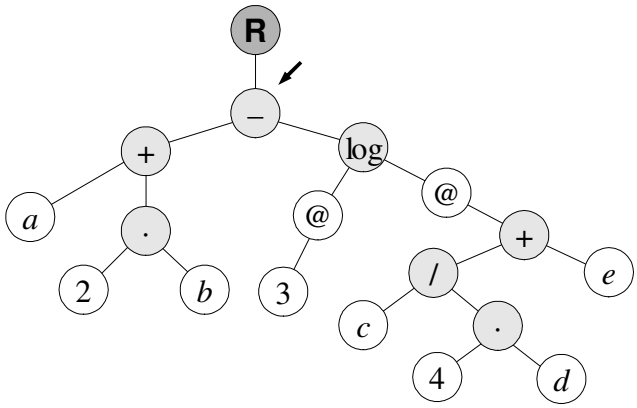
Teraz sa vloží hodnota logaritmu: „$\frac{c}{4d}+e$” ako druhý parameter. Do stromu sa však vkladá každý element tohto bloku osve, čo znázorňujú body a) až f) obrázku 3.



Obrázok 3: Vkladanie hodnoty logaritmu

Vloženie „*c*”, „/” aj „4” je priamočiare. Ako všetky operátory, aj operátor „·” je reprezentovaný objektom z triedy `OperToken` (sivé vrcholy), ktorá obsahuje pravidlo vkladania do stromu zohľadňujúce prioritu. Keďže „·” má vyššiu prioritu ako „/”, bude vložený ako syn „/”. V ďalšom kroku sa však vkladá operátor „+”, ktorý má nižšiu prioritu aj ako „·”, aj ako „/”, teda podľa pravidla musí byť vkladaný ako ich predchodca. V kroku e) je naznačené hľadanie správneho umiestnenia porovnávaním so spomínanými operátormi.

Výsledný strom bude nakoniec vyzerať nasledovne:



Obrázok 4: Výsledný AS

Takto vytvorený strom možno efektívne použiť na reprezentovanie zadanej formuly, lebo jeho prechádzaním ju možno jednoducho spätne zrekonštruovať.

4.3.2 Generovanie výstupov

Ako bolo uvedené v 4.2.5, aplikácia musí generovať štyri druhy výstupov. Sú to dáta pre export, matematický riadok, „klasický” zápis a hlasový výstup. Výhodou je, že dáta pre export a „klasický” zápis sa zhodujú – je to MathML, takže sa počet výstupov zredukoval na tri.

Všetky výstupy je možné vygenerovať prehľadávaním a prácou na AS. Každý výstup je generovaný príslušnou metódou. Pretože každý vrchol je objekt oddedený z typu `Token` a má prekryté jeho metódy, je úlohou vrchola z ľubovoľnej podtriedy postarať sa o správny výstup zo svojho podstromu. Zavolaním príslušnej metódy na koreni potom dostávame výstup vzťahujúci sa na celý strom.

Generovanie matematického riadku

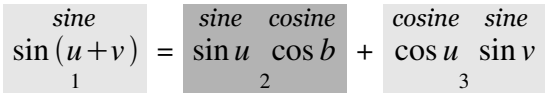
Matematický riadok sa v klientovi prejaví ako tabuľka, ktorá je utvorená skriptom z postupnosti záznamov reprezentujúcich atomické elementy (4.2.5). Aby bolo možné

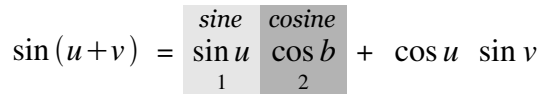
skonštruovať riadok s očíslovaním a pomenovaním blokov, každý záznam o elemente musí obsahovať hodnotu a typ objektu a nepovinne číslo a meno. Keďže sa číslujú celé bloky elementov, musia elementy zoskupené v danom bloku zdieľať spoločné číslo. Práve podľa tohto čísla sa potom určí rozdelenie formuly do blokov – blok končí tam, kde končí postupnosť rovnakých čísel jeho elementov. Pretože pre každú úroveň číslovania formuly v HČLT je rozdelenie na bloky iné (expanzia rozbíja bloky na jeden alebo viac podblokov), je iné aj zoskupovanie elementov do blokov a teda aj ich číslovanie. Toto číslovanie teda nemôže byť statické a musí sa po každom kroku prepočítavať.

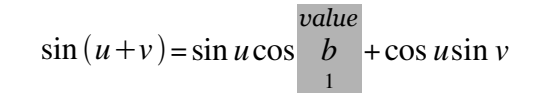
Na výpočet čísla pre konkrétny element treba teda poznať aktuálnu úroveň číslovania formuly resp. počet expanzií vybratého bloku. Ak nie je expandovaný žiadny blok, postupne sa očíslujú všetky bloky formuly. Po zvolení bloku na expandovanie sa už ostatné bloky číslovať nebudú, pretože by to nemalo žiadny praktický význam. Číslovanie sa teda v každej úrovni aplikuje len na podstrom, kde koreňom je vybratý blok (v prípade, že nie je vybratý žiadny, je koreňom celá formula).

Obdobný postup platí aj pri pomenovávaní blokov. Bloku je vhodné priradiť také meno, ktoré ho charakterizuje v danej úrovni číslovania formuly. Teda napríklad v prvej úrovni možno blok „$\log(a, 74\,b)$" pomenovať ako „*logarithm*", pričom po expanzii sa rozbije na podbloky „a" s menom „*base*" (základ logaritmu) a „$74\,b$" s menom „*value*" (jeho hodnota). Takto je možné vyhľadaním „*logarithm*" a „*value*" zmeniť hodnotu logaritmu bez toho, aby bolo nutné poznať jej presnú pozíciu vo formule.

V nasledujúcom príklade je známa identita, v ktorej je však chyba – miesto „v„ je v prvom kosínuse „b": $\sin(u+v)=\sin u\cos b+\cos u\sin v$. Chybu chceme opraviť. V náčrte nižšie je zobrazený MR pre zadanú identitu, svetlosivou sú označené bloky, tmavosivou expanzia a taktiež je znázornené očíslovanie i pomenovanie:







Ako si možno všimnúť, operátory nie sú číslované. Je to preto, lebo sú prirodzene považované za spájajúce resp. oddeľovacie elementy a nie za obsahové. Je však možné prepnúť sa do módu číslovania operátorov, ktoré sa potom očíslujú obdobným spôsobom, ako bol práve uvedený.

Na manipuláciu s MR sa dá navrhnúť trieda LineTable, ktorá v podstate robí transformáciu stromu Token-ov na strom operandov, ktorý sa „vypíše" do postupnosti záznamov – stĺpcov tabuľky. Je to akási inverzná procedúra k rekonštrukcií aritmetického stromu zo vstupu, avšak na rozdiel od vstupu má táto trieda prehľad o štruktúre stromu, pretože výsledná postupnosť vzniká nie jeho rozbitím, ale traverzovaním správnym spôsobom. Špecifickou informáciou pre MR, ktorú si musí LineTable uchovávať, je aktuálne číslovanie, výber a určenie či resp. ktorý blok je expandovaný. Ako bolo vyššie spomínané, očíslovanie ľubovoľného elementu je určené úrovňou expanzie bloku, ktorý ho obsahuje. Pri postupnej expanzii sa vytvára navigačná postupnosť (vo vyššie uvedenom príklade možno sledovaním tmavosivých blokov vytvoriť postupnosť $\{2, 2, 1\}$), pričom *i*-tý prvok hovorí, ktorý blok bol v *i*-tom kroku expandovaný. Teda je nutné si túto postupnosť pamätať, aby sa vedelo ktorý blok je práve expandovaný a v ktorej úrovni číslovania sa forma nachádza. Pri kontrakcii sa z postupnosti uberie posledný prvok a hodnota nového posledného prvku určí index nového vybratého bloku. Okrem toho si LineTable musí ešte uchovávať informácie o aktívnom mieste vkladania a výbere (selection).

4.3.3 Udržiavanie stavových informácií

Ďalšou dôležitou úlohou servera je udržiavanie informácie o stave. Okrem spomínaných inštancií TokenTree a LineTable musí aplikácia vedieť, či je vo vkladacom alebo navigačnom móde.

Aplikácia musí mať tiež prehľad o histórii aktivít. Táto história je dôležitá kvôli odvolávaniu posledných akcií (známe ako funkcia „undo"). Po odvolaní poslednej akcie je žiadúce, aby systém vrátil do pôvodného stavu dáta, texty na obrazovke, výpisy, hlasové výstupy... Toto všetko sa musí evidovať v každom kroku.

Technická realizácia pamätania stavu nie je komplikovaná. Stačí vytvoriť duplikáty všetkých potrebných objektov a vložiť ich do zásobníka. Pri odvolávaní potom stačí vyhodiť zo zásobníka zvolený počet prvkov a posledný prehlásiť za nový stav. Keďže sa pracuje s rádovo malým počtom objektov a nepredpokladá sa enormné množstvo operácií, zdá sa bezpredmetné hľadieť na pamäťové nároky.

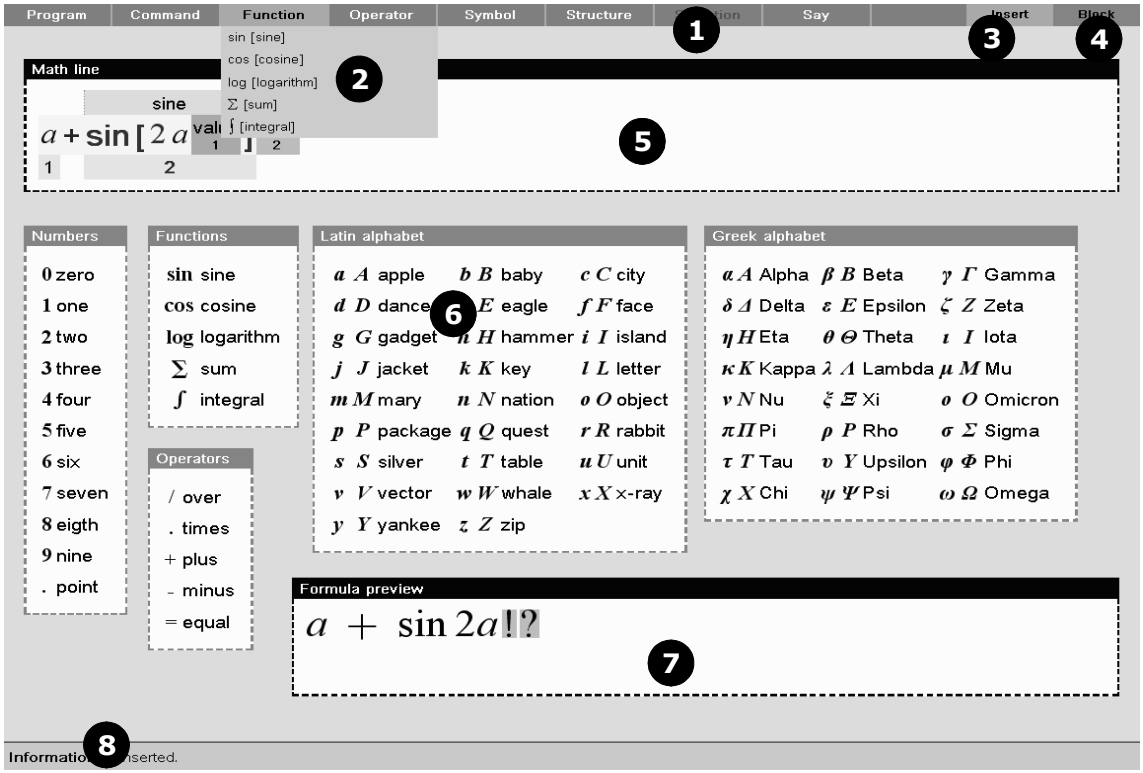
5 Implementácia

5.1 Klient

Aj keď je program webovskou aplikáciou, jeho ambíciou je vyzerať ako klasická desktopová aplikácia. V kapitole 4.2.1 (o návrhu UI) bol uvedený náčrt jednotlivých komponentov UI. V tejto časti sa budem podrobnejšie venovať ich implementácii ako aj stručnému popisu ich fungovania.

5.1.1 Vizuálne rozhranie

Na nasledujúcom obrázku je snímok obrazovky bežiacej aplikácie. V čiernych krúžkoch sú číselne označené jednotlivé vizuálne prvky. Ich stručný popis, ako aj popis ich funkcie, nasleduje za obrázkom.



Obrázok 5: Snímok obrazovky bežiacej aplikácie

1. Hlavné menu aplikácie – väčšina funkcií je kvôli prehľadnosti dostupná cez menu. Menu je dvoj-úrovňové a položky sú roztriedené do navrhovaných kategórií. Slovo na otvorenie položky sa zhoduje s jej textom.

2. Rozbalené hlavné menu *Function* obsahujúce zadefinované funkcie. Každá položka je definovaná textom resp. symbolom a slovom. V prípade, že sa text a slovo nezhodujú, je slovo uvedené v hranatých zátvorkách.
3. Indikátor pracovného módu (*Insert*, *Navigation*)
4. Indikátor módu číslovania (*Block*, *Operator*)
5. Matematický riadok vo forme tabuľky. V prvom riadku sú mená blokov, v druhom samotné dáta a v treťom číslovanie. Pri kliknutí na meno alebo číslo bloku sa blok vyberie. Ak sa aplikácia nenachádza v navigačnom móde, prepne sa doň.
6. Panely s číslami, znakmi, symbolmi... Ak sa klikne na symbol (resp. na jeho popis v prípade, že nie sú veľké a malé alternatívy), tak sa vloží. V prípade, že sa pracuje v navigačnom móde, sa prepne späť do vkladacieho módu.
7. Náhľad zadávanej formuly v „klasickom zápise" zobrazovaný pomocou zásuvného modulu.
8. Stavový riadok, ktorý zobrazuje informácie a chyby.

Všeobecne o dizajne stránky

Pri dizajne stránky bola snaha umožniť nastavovať vzhľad i rozmiestnenie panelových elementov. Okrem menu (vrátane identifikátorov módu) a stavového riadku sú vlastnosti panelov definované v externom CSS [21] súbore, čo otvára možnosť meniť polohu, viditeľnosť a ostatné vlastnosti bez nutnosti meniť server.

Aplikácia je dimenzovaná na rozlíšenie 1024 x 768, avšak nie je problém zmenou veľkosti písma v CSS súbore zmeniť veľkosť celej stránky, pretože všetky veľkosti sú počítané relatívne (výnimkou je menu, ktoré sa konfiguruje inak – pozri ďalšiu sekciu). Ďalším spôsobom je použitie integrovanej funkcie „zoom" (priblíženie) v prehliadači Opera.

Ako už bolo spomínané v návrhu, väčšina obsahu hlavnej stránky je statická, aby sa predišlo stálemu prekresľovaniu. Preto sa akékoľvek zmeny robia prostredníctvom JavaScript-u. Vzhľadom na stále praktické problémy s týmto jazykom nie je zaručené, že sa v budúcich verziách prehliadača Opera bude aplikácia správať korektne. Dá sa však predpokladať, že zmeny nebudú kritické, a teda tento problém nenastane.

Menu

Pretože HTML formuláre nemajú menu, musela sa použiť nejaká JavaScript-ová verzia. Dnes je ich na internete veľmi veľa, no nie všetky sú dostatočne jednoduché a prehľadné. Pretože si užívateľ môže definovať niektoré vlastné položky (funkcie,

symboly), bolo potrebné nájsť také menu, ktoré by bolo dobre konfigurovateľné. Po vyskúšaní viacerých implementácií som sa rozhodol pre *Tigra Menu* [22], ktoré je zadarmo a jeho konfigurácia spočíva v nastavení vzhľadu a definovaní položiek. Položky sa pritom definujú veľmi jednoduchým zápisom štruktúry do príslušného JavaScript-ového súboru, ktorý sa generuje pri kompilácii nastavení (pozri 5.2.3, odsek „Balík processing"). Po zvolení položky sa zavolá skript, ktorý naplní príslušné skryté polia formulára a ten sa odošle na server.

Pri zmene módu je potrebné niektoré položky zakázať (disable) a zakázané naopak povoliť. Napríklad vo vkladacom móde nie je možné manipulovať s výberom (selection) a v navigačnom móde sa nedajú zadávať ďalšie bloky. Pretože sa hlavná stránka neobnovuje, bolo nutné túto funkcionalitu znova zabezpečiť pomocou JavaScript-u, čo vyžadovalo menšie úpravy použitej implementácie *Tigra Menu*.

Napriek tomu, že indikátory módu sú umiestnené ako súčasť menu, ich funkcia nemá s menu nič spoločné. Ich jedinou úlohou je informovať o aktuálnom móde práce. Parametre je možné konfigurovať v hlavnom CSS súbore.

Matematický riadok

Ako už bolo viackrát spomínané, MR je reprezentovaný tabuľkou. Každý element (záznam) je reprezentovaný jedným stĺpcom. Aby sa dosiahol efekt blokov, bolo použité zlúčenie buniek susedných elementov s rovnakým názvom a s rovnakým číslom.

Na obrázku možno vidieť použitie rozličných štýlov pre rôzne typy elementov. Toto sa dosiahlo použitím atribútu „*class*" na bunke. Ich definícia je opäť v hlavnom CSS súbore, možno ich nájsť pod tagom „*lineTable*".

Matematický riadok je nutné po každej akcii prekresliť. Zo servera prídu všetky potrebné informácie o každom zázname, no v tvare reťazca, kde sú jednotlivé elementy oddelené špeciálnym oddeľovačom. Je teda úlohou skriptu tieto záznamy získať a postupne pridávať do tabuľky, pričom sa musí realizovať zlučovanie susedných buniek, ako to bolo spomínané vyššie. Hojne sa tu využíva práca s HTML DOM [23].

Panely

Pojmom „panely" sa rozumejú jednotlivé tabuľky rozmiestnené v hlavnom okne, ktoré slúžia jednak ako vizuálny pomocník a tiež umožňujú zrýchlené zadávanie príslušných elementov, pretože kliknutie na symbol resp. jeho popis simuluje kliknutie naň v menu. Tieto tabuľky sú generované (podobne ako položky menu) pri spustení aplikácie z konfiguračného súboru nastavení. Vzhľad a polohu možno konfigurovať v hlavnom CSS súbore, kde vystupujú so sufixom „panel".

Náhľad formuly

Pretože náhľad formuly používa zásuvný modul, je tvorený samostatným rámcom vloženým do hlavnej stránky. Tento rámec sa po každej zmene obnovuje. Aby mohol modul správne fungovať, je potrebné posielať mu dáta vo formáte MathML a nastaviť „Content-Type" na „text/mathml". O to sa stará server.

Stavový riadok

Poslednou, no nemenej dôležitou súčasťou UI je stavový riadok. Po vykonaní akejkoľvek akcie sa tu sa zobrazuje jej výsledok v tvare informácie (ak akcia prebehla úspešne) alebo chyby, pokiaľ nejaká nastala. Tieto dva stavy sú vizuálne (zmenou farby) aj zvukovo (zvukový signál) odlíšené.

5.1.2 Zvukové rozhranie

Spracovanie vstupu

Pri využití spomínaných technológií je z hľadiska implementácie najdôležitejšou časťou zvukového rozhrania systém na spracovanie zvuku. O rozpoznávanie foném v zvukových vlnách, ako aj o prepis slov do foném, sa stará modul v prehliadači Opera, takže mojou úlohou bolo napísať gramatiky pre spracovanie a definovať ich sémantiku.

V návrhovej časti 4.2.2 bolo načrtnuté ako by mali gramatiky vyzerať i vysvetlená potreba kategorizácie na riadiace a dátové gramatiky. Riadiace gramatiky pokrývajú dve oblasti – riadenie prostredia a príkazy na manipuláciu. Za riadenie prostredia sa považuje napríklad zmena stavu alebo odvolanie poslednej akcie, za príkaz na manipuláciu možno považovať príkaz na prečítanie bloku či presun bodu vkladania o jedno doprava. Globálne príkazy, nezávislé od pracovného stavu (napríklad odvolávanie poslednej akcie), sú v jednej spoločnej gramatike, ostatné sú v gramatikách, ktoré sa použijú podľa zvoleného stavu.

Špeciálnou gramatikou je gramatika *menu*, ktorá implementuje hlasový ekvivalent grafického menu a generuje sa pri kompilácii nastavení. V tejto gramatike je mnoho veci podobných ako v ostatných riadiacich gramatikách, pretože mnoho položiek menu slúži práve na riadenie aplikácie. Možno by sa zdalo, že riadiace gramatiky sú v tomto prípade nepotrebné, ale nie je tomu tak. Riadiace gramatiky totiž obsahujú väčšinou zjednodušenú frázu pre danú akciu a niekedy dokonca poskytujú nepovinné parametre (napríklad pri odvolaní poslednej akcie sa dá uviesť aj počet krokov, ktorý v menu nie je).

Čo sa týka dátových gramatík, implementované sú kategórie ako *numbers* (čísla), *alpha* (latinka), *greek* (grécke znaky), *linear* (výrazy), *structures* (zátvorky, zlomky a pod.). Každá s týchto kategórií vkladá špecifický typ objektu.

Vyslovenie frázy musí mať za následok rovnaký efekt, ako kliknutie na príslušný vizuálny element. K tomu slúži sémantika gramatiky – pravé tá totiž určuje ako a čím sa naplní výsledný objekt, ktorý sa po rozpoznávaní vráti. Sémantika všetkých gramatík sleduje rovnaký cieľ – na objekte sa nastaví *akcia*, *typ* a *hodnota*.

Podrobný popis platných vstupov možno nájsť v prílohe č. 5.

Dizajn X+V stránky

Celá X+V stránka sa nachádza v skrytom rámci prehliadača kvôli podmienke čo najmenšieho obnovovania hlavnej stránky. X+V stránka pozostáva z XHTML časti a VoiceXML časti. XHTML časť obsahuje práve ten (už viackrát spomínaný) formulár, ktorý sa posiela na server a okrem neho už nič viac. VoiceXML časť, konkrétne formulár, sa stará nielen o rozpoznávanie reči, ale aj o riadenie dialógu. Je vhodné popísať jeho dizajn, spôsob akým sa dáta z objektu vráteného rozpoznávačom reči dostanú na server a spôsob čítania hlasových výziev.

VoiceXML formulár obsahuje jedno hlavné pole, kde sa deje čítanie hlasových výziev a informácií a rozpoznávanie jazykov všetkých gramatík (okrem menu). Výzvy, rovnako ako informácie zo stavového riadku, sa vložia vždy na začiatok. Po ich prehratí sa podľa aktuálneho stavu aktivujú príslušné gramatiky a čaká sa na vstup od používateľa. Po spracovaní vstupu gramatikou sa vráti JavaScript-ový objekt s vyplnenými vlastnosťami *akcia*, *typ* a *hodnota*. Tieto sa pomocou skriptu prekopírujú do príslušných polí vo formulári a formulár sa odošle.

Okrem hlavného poľa sú vo VoiceXML formulári aj iné polia – pre každú položku hlavného menu jedno. Tieto polia sú automaticky generované pri kompilácii nastavení a starajú sa o manipuláciu s danou položkou menu (čítanie jej podpoložiek, rozbalenie/zbalenie položky prostredníctvom volaného skriptu). Po vyslovení názvu položky hlavného menu sa príslušné pole aktivuje, až pokým sa nepotvrdí podpoložka alebo sa jeho spracovanie nezruší príkazom „*close*”. V druhom prípade sa spracovanie presunie znova do hlavného poľa.

Generovanie výstupu

Hlasový výstup je generovaný prostredníctvom TTS syntetizátora zabudovaného v hlasovom module prehliadača Opera. Ako už bolo viackrát spomínané, problém generovania výstupu sa zredukuje na problém vygenerovania správneho reťazca slov,

ktorý môže obsahovať formátovacie značky. Tento reťazec sa vygeneruje na serveri a vkladá sa ako *prompt* do hlavného poľa VoiceXML formulári.

5.1.3 Skripty

Významnú časť funkcionality na klientovi tvoria práve skripty. Sú dôležité jednak z hľadiska manipulácie údajov na stránke, no taktiež slúžia ako spojivo medzi serverom a klientom, pretože prostredníctvom nich sa odosiela formulár. Rovnako ako väčšina vecí, aj skripty sú kategorizované.

Globálne skripty sa používajú na robenie vizuálnych zmien (v indikátoroch módu či stavovom riadku), na nastavovanie premenných formulára a na jeho odoslanie. Ďalej sú tu skripty na vypĺňanie matematického riadku prijatými záznamami.

Poslednou (a najväčšou) kategóriou sú skripty, ktoré sa starajú o menu a spracovávajú akcie kliknutia na jednotlivé položky. Okrem skriptov aplikácie program obsahuje aj skripty samotného jadra menu vytvorené jeho autormi a tiež skript na jeho konfiguráciu.

5.1.4 Pomocné dialógové okná

Okrem hlavnej stránky program používa pomocné dialógové okná. Jedno okno je použité na potvrdenie zvolenej akcie – užívateľ na otázku odpovie kladne alebo záporne. Dizajn okna spočíva v jednoduchom formulári s jedným centrovaným textom a dvoch tlačidiel pre kladnú a zápornú odpoveď. Z hľadiska ovládania hlasom je vo VoiceXML formulári použité jedno pole so zabudovanou logickou (built-in boolean) gramatikou. Po prehratí výzvy, ktorá sa zhoduje s textom, sa odpoveď spracuje a na server sa odošle buď zadaná akcia alebo informácia o jej zrušení v podobe simulácie kliknutia na zvolené tlačidlo.

Druhé okno slúži na vyhľadávanie v MR. V závislosti na aktívnom stave sa vyhľadáva buď medzi bodmi na vkladanie (vo vkladacom móde), alebo medzi jednotlivými menami blokov (v navigačnom móde). Okno obsahuje zoznam spomínaných prvkov, ktoré sú aktuálne dostupné. Okrem toho obsahuje (rovnako ako v predošlom prípade) dve tlačidlá na potvrdenie alebo zrušenie akcie. Pri zobrazení okna sa prehrá výzva, aby používateľ vyslovil, ktorý prvok chce vyhľadať. Pre prečítanie zoznamu práve dostupných prvkov stačí povedať „*help*". Po vybratí sa dialóg zatvorí a prvok sa vyhľadá, po vyslovení „*close*" sa akcia zruší.

5.1.5 Konfigurácia

Jednou z požiadaviek na aplikáciu bola aj možnosť jednoducho meniť nastavenia či meniť množinu poskytovanej funkcionality. Priamočiarym riešením je vytvorenie konfiguračného súboru, ktorý okrem nastavení bude poskytovať aj možnosť rozširovať položky menu o vlastné.

Veľmi dobrým a hojne využívaným formátom na reprezentáciu štruktúrovaných dát je XML (Extensible Markup Language). Základnou ideou ako ho využiť pri zápise nastavení je ich rozdelenie do logických sekcií – užívateľ môže meniť niektoré parametre aplikácie, pridávať funkcie, operátory, symboly a štruktúry. Každá z týchto kategórii bude reprezentovaná príslušným uzlom v XML súbore.

Okrem návrhu kategórií je nutné navrhnúť aj tvar samotných uzlov v danej kategórii – teda obsah a atribúty:

- *Parameter aplikácie* (uzol *setting*): aby bola zabezpečená univerzálnosť, uzol je identifikovaný menom (typu identifikátor) a jeho hodnota určuje hodnotu parametra.
- *Funkcia*: atribúty funkcie sú symbol a fráza na vyslovenie (*speak*). Ďalej obsahuje ako synov svoje pomenované parametre. Okrem toho môže obsahovať šablónu, ako sa má prečítať a ako sa má zapísať do MathML. Tieto šablóny obsahujú zástupné znaky, ktoré sa pri spracovaní nahrádzajú príslušnými prvkami.
- *Operátor*: operátor je okrem symbolu a frázy definovaný ešte prioritou. Takisto tu môžu byť šablóny pre čítanie a matematický zápis.
- *Symbol*: charakterizovaný symbolom frázou. Niektoré symboly sa v MathML zapisujú pomenovanou entitou, je tu teda možnosť uviesť jej meno.
- Štruktúra: najzložitejší stavebný prvok. Rozlíšiť možno jednoduché štruktúry „typu zátvorky" a zložitejšie „typu zlomok". Jednoduché štruktúry sú tvorené len ľavým a pravým symbolom a pomenovaním obsahu. Jadrom zložitejších štruktúr je nejaký binárny operátor, ktorý spája svoje obe strany, a tak vlastne štruktúru vytvára (zlomková čiara spája čitateľ a menovateľ). Niekedy sa za štruktúru považuje iba niektorá strana operátora, napríklad horný index sa spája s elementom na ktorý je aplikovaný, no za samotný index sa považuje iba „symbol hore". Štruktúra teda okrem operátora obsahuje podľa potreby ešte ľavú a pravú stranu, ktoré môžu byť tvorené aj zložitejšími konštrukciami.

Ďalšie informácie možno nájsť v anglickom jazyku v podobe komentárov priamo v konfiguračnom súbore.

5.2 Server

5.2.1 Organizácia a štruktúra servera

V návrhovej kapitole 4.1 bolo spomínané použitie javovského aplikačného servera. Názov tejto kapitoly však neoznačuje dizajn tohto servera, ale dizajn „serverovej časti" aplikácie, ktorá beží na tomto aplikačnom serveri. S prihliadnutím na fakt, že ide o webovskú aplikáciu, je organizácia serverovej časti aplikácie priamočiaro daná – je žiadúce oddelenie kódov tried na spracovanie dát od kódu pre vizuálnu časť aplikácie.

Organizácia kódu pre spracovanie dát

Všetok kód je delený do tried, ich zaradenie do štruktúry sleduje javovské konvencie – balíky. Pre jednoznačnosť bol pre tento projekt zvolený prefix com.hanaxsoftware.dipl, čo je aj meno hlavného balíku. Ten obsahuje triedy Utils a GlobalData, z ktorých prvá poskytuje rôzne užitočné statické funkcie a druhá sa stará o uchovávanie globálnych informácii o stave aplikácie a slúži ako dátová centrála pre všetky ostatné triedy.

Hlavný balík obsahuje tri podbalíky – menu, processing a tokens. Prvý sa stará o objektovú reprezentáciu menu, druhý o kompiláciu konfiguračného súboru a napĺňanie interných štruktúr a tretí obhospodaruje manipuláciu s Token-ami, parsovanie vstupu, vytváranie blokov, budovanie dátového stromu a generovanie matematického riadku.

Organizácia kódu pre vizuálnu časť

Vizuálna časť aplikácie je generovaná prostredníctvom známej technológie JSP (JavaServer Pages [24]). Jedným z postupov, ktorý táto technológia (na generovanie dynamických stránok) umožňuje, je vpisovanie kódu v Jave do špeciálnych tagov (značiek), ktoré sú potom serverom spracované. Práve tento postup je využitý v tejto aplikácii.

Aplikácia sa spúšťa pomocou stránky „index.jsp". Tá však slúži len na inicializáciu inštancie triedy GlobalData a v prípade, že nenastala chyba, je používateľ ihneď presmerovaný na hlavnú stránku „main.jsp". V opačnom prípade sa chyba vypíše.

Vzhľad hlavnej stránky (popísaný v 5.1.1) vyžaduje zobrazenie panelov so symbolmi, funkciami, operátormi a podobne. Tieto množiny sú však čítané

z konfiguračného súboru, a tak musia byť do stránky dynamicky vložené. O to sa postará inštancia triedy GenContent z balíka processing.

Okrem toho sú v hlavnej stránke vložené dva rámce – jeden skrytý, ktorý (ako už bolo spomínané) slúži z pohľadu klienta prevažne na riadenie dialógu, no z pohľadu servera má podstatný význam, pretože práve tu sa nachádza väčšina riadiaceho kódu. Rámec pracuje so stránkou „voiceFrame.jsp". Tá je rozdelená (ako väčšina XHTML stránok) na dve časti: na hlavičku a telo. Hlavička sa väčšinou považuje za informačne nevýznamnú časť. Pretože však práve táto stránka obsahuje VoiceXML formulár (ktorý je podľa špecifikácie X+V celý v hlavičke), obsahuje táto časť všetok javovský kód, ktorý je bližšie popísaný v nasledujúcej kapitole.

5.2.2 Spracovanie dát

V kapitole 4.2.4 bol bol naznačený formát i spôsob, akým sa dáta dostanú na server, no až dosiaľ nebolo explicitne napísané, čo sa s nimi ďalej stane. Táto kapitola sa to bude snažiť objasniť a tiež podrobnejšie popísať formu prijatých dát.

Hlavný formulár obsahuje štyri skryté polia: *action*, *type*, *value* a *value2*. V poli *action* je uložený typ akcie, ktorá sa na klientovi vykonala, v poli *type* je upresnenie typu. Polia *value* resp. *value2* slúžia na posielanie dát a sú myslené ako prvý resp. druhý parameter akcie.

Akcie môžu byť rôzne, napríklad „insert" (vkladanie), „navigate" (navigácia) či „command" (príkaz). Akcie poskytujú všeobecnú informáciu o tom, čo robiť s dátami. Pole *typ* akciu upresňuje, napríklad pre akciu „insert" sú platnými typmi napríklad „function" či „symbol".

Rozoberme si bližšie najčastejšie používanú akciu „insert". Ak používateľ zadal požiadavku vložiť „$a+b$", potom akcia bude „insert" a typ bude „linear". Pozrime sa teraz na dáta. Je zrejmé, že musia byť v tvare jedného reťazca, no jednoduché „a+b" nestačí, pretože by sme radi vedieť že „+" je operátor a „a" a „b" sú písmená latinky. V tomto jednoduchom prípade by sa dalo „domyslieť", že je to tak. Skúsme však vložiť napríklad „*trn*". Ide o funkciu alebo sekvenciu znakov? Z tohto dôvodu je nutné posielať si o každom vkladanom objekte aj jeho typ. Aby sa však stále dali údaje uložiť do jedného reťazca, budeme z klienta posielať vždy záznamy oddelené čiarkou, pričom záznam bude v tvare dvojice hodnota a typ. Hodnotu a typ je treba tiež oddeliť nejakým oddeľovačom, bol zvolený symbol opačného dĺžňa „`", pretože je nepravdepodobné, že by sa inde vyskytol. Teda v prípade „$a+b$" sa pošle „a`alpha,+`op,b`alpha".

Keďže je už zrejmý formát a význam prijatých dát, je možné ich spracovať. Ako bolo napísané, spracovanie sa deje v hlavičke stránky „voiceFrame.jsp". Je tomu tak

preto, lebo akákoľvek akcia môže poskytovať hlasový výstup, najčastejšie ako efekt „opakovacieho papagája", ktorý pre overenie správnosti rozpoznania prečíta to, čo sa povedalo (prípadne na čo sa kliklo). Keďže práve prebieha generovanie stránky, klient bude chcieť po jej skončení prehrať výzvy VoiceXML formulára. Preto sa akékoľvek hlasové výstupy musia generovať do hlavičky, kde sa VoiceXML formulár nachádza.

Samotné spracovanie akcie vyzerá ako blok postupných porovnávaní prijatých hodnôt s konkrétnym reťazcom, teda ako podmienka „v štýle switch" na reťazcoch. Výber je hierarchický – najprv sa vyberá podľa akcie, potom podľa typu. Nájdený kód spracuje dáta, vykoná príslušné akcie a nastaví informácie pre stavový riadok, prípadne vygeneruje skript, ktorý sa spustí na klientovi po načítaní stránky.

5.2.3 Balíky

V kapitole o organizácii kódu pre spracovanie dát bolo spomínané rozdelenie do hlavného balíka a jeho troch podbalíkov (aj s ich stručným popisom). V tejto časti sa chcem podrobnejšie venovať jednotlivým balíkom za účelom objasnenia fungovania servera.

Hlavný balík (com.hanaxsoftware.dipl)

Tento balík obsahuje vyššie uvedené triedy Utils a GlobalData. Triedy nemajú za úlohu nijak spoločne kooperovať a sú z organizačných dôvodov umiestnené v spoločnom balíku len preto, lebo majú globálny význam.

Utils obsahuje zopár užitočných metód, konkrétne: zistenie, či je reťazec platné číslo a metódy na formátovanie zvukového výstupu – pridávanie SSML značiek.

Význam triedy GlobalData je nepomerne väčší. Túto triedu možno označiť za centralizovaný sklad, kde sú uložené všetky informácie, s ktorými sa pracuje. Je to spravené buď v podobe polí alebo v podobe referencií na iné objekty. Takáto vlastnosť mala svoj zámer – bolo cieľom, aby každá trieda mala prístup ku globálnym informáciám, bez potreby dostávať do metód množstvo parametrov. Trieda sleduje programátorskú techniku „singleton", teda je povolená jediná inštancia triedy, ku ktorej sa dá pristupovať odvšadiaľ zavolaním statickej metódy. Okrem uchovávania dát spravuje táto trieda aj odvolávanie akcií (uchovávaním predošlých stavov aplikácie) a zabezpečuje nastavovanie stavového riadku.

Balík menu (com.hanaxsoftware.dipl.menu)

Ako už názov hovorí, úlohou tohto balíka je spravovať menu aplikácie. Grafická reprezentácia v podobe JavaScript-ového menu je iba odrazom internej štruktúry menu

a neobsahuje dôležité údaje o samotných položkách (napríklad parametre funkcie). Preto je nutné mať jeho objektovú reprezentáciu.

Základným stavebným kameňom sú objekty z triedy MenuItem. Každá položka obsahuje titulok a frázu, ktorou má byť čítaná. Okrem toho obsahuje referencie na svoje podpoložky (ak ide o položku hlavného menu) resp. na vlastníka (ak ide o samotnú podpoložku). Hlavnou funkciou položky je „sa vedieť vypísať" do formátu, v akom ju očakáva JavaScript-ové menu.

Od MenuItem sú oddedené štyri ďalšie triedy: Funkcia, Operator, Struktura a Symbol. Ako už z názvu vidno, ide o bližšie určenie položky menu s pridaním ďalších vlastností a funkcionality. Inštancie týchto tried nie sú však len menu položkami, sú to hlavne reprezentanty svojich tried, ktoré pridávajú schopnosť vypisovať sa do MathML, generovať reťazec, ktorým budú prečítané a schopnosť načítať sa z konfiguračného súboru. Práve tieto položky slúžia ako spojivo medzi menu a inštanciami triedy Token, ktoré sa na ne odvolávajú.

Položku hlavného menu priamo určuje kategóriu podpoložiek (položka hlavného menu s názvom „Funkcie" obsahuje funkcie). Preto existuje trieda MenuCategory, ktorá zabezpečuje jednoduchú prácu s celou skupinou položiek, od vkladania až po vyhľadávanie podľa mena, symbolu, či čísla.

Celé menu je nakoniec „poskladané" v triede Menu, ktorá sa okrem toho ešte stará o povoľovanie/zakazovanie jednotlivých položiek a výpis kategórii do polí VoiceXML formulára, ako to bolo napísané v kapitole 5.1.2.

Balík processing (com.hanaxsoftware.dipl.processing)

Tento balík sa stará v podstate o spracovanie konfiguračného súboru (KS). Podľa KS je nutné vyrobiť viacero súborov, ktoré bude aplikácia používať. Sú to hlavne rôzne gramatiky a súbor s položkami pre JavaScript-ové menu. Proces spracovania KS nazvem „kompilácia" KS. Spúšťa sa otvorením stránky „rebuild.jsp".

Proces kompilácie pozostáva z dvoch krokov. Na začiatku statická trieda Config otvorí konfiguračný XML súbor a číta jednotlivé sekcie. Pri čítaní nastavení manipuluje priamo s nastaveniami v triede GlobalData, pri čítaní funkcií, operátorov, symbolov či štruktúr manipuluje s príslušnými inštanciami MenuCategory a pridáva do nich položky vytvorené podľa údajov v súbore.

V druhej fáze sa trieda FileGenerator postará o vygenerovanie príslušných súborov. Volaním metód toJS resp. toMML sa do príslušných súborov zapíše menu, gramatiky pre všetky kategórie a gramatika s latinkou a s gréckymi písmenami.

Balík tokens (com.hanaxsoftware.dipl.tokens)

Tento balík zastrešuje triedy, ktoré sa starajú o spracovanie vstupu, reprezentáciu formuly a generovanie výstupu. Je teda zameraný hlavne na manipuláciu s dátami.

Prípad najčastejšej akcie „insert" už bol z algoritmického hľadiska dosť podrobne popísaný v kapitole 4.3.1. Čo sa týka implementačného hľadiska, najprv sa aktivuje Parser, ktorý dostane ako dáta reťazec záznamov poslaný z klienta. Tento reťazec rozbije podľa oddeľovača a pre každý záznam vytvorí jeho objektovú reprezentáciu ako inštanciu niektorej dcérskej triedy Token-u.

Z postupnosti Token-ov sa po parsovaní spraví Block. Block nie je len štruktúra na uchovávanie Token-ov, ale obsahuje aj množstvo metód na manipuláciu, vyhľadávanie či výpis do rôznych formátov. Vytvorený Block sa potom pošle aritmetickému stromu (resp. jeho slabej modifikácii) triedy TokenTree na vloženie (spôsobom opísaným v spomínanej kapitole 4.3.1).

Zaujímavým je fakt, že trieda TokenTree je vytvorená len pre uľahčenie reprezentácie stromu. Samotný strom je v skutočnosti tvorený pospájaním Token-ov. Napríklad výpis stromu nie je nič iné, ako volanie výpisu na jeho koreni pričom sa postupuje algoritmom *pre-order*. Je ale oveľa prijateľnejšie a intuitívnejšie mať metódy na strome a nie na jeho vrchole.

O manipuláciu matematickým riadkom sa stará trieda LineTable. Táto trieda nepracuje priamo so stromom, ale so štruktúrou, ktorá vznikne traverzovaním stromu. Väčšina Token-ov v tejto postupnosti je zároveň aj uzlami stromu, no niektoré prvky sú špecifické pre LineTable. Napríklad zátvorky parametra funkcie sa v strome nikde nevyskytujú, no v matematickom riadku sa objavia. Metódy tejto triedy umožňujú napríklad expandovanie blokov, nastavovanie bodu vkladania a vyhľadávanie štruktúr.

Triedy tokenov (com.hanaxsoftware.dipl.tokens.classes)

Trieda Token je základným dátovým objektom, reprezentujúcim jeden element formuly. Táto trieda, ako aj triedy z nej odvodené, umožňujú všetky manipulácie s elementami, ktorými sú hlavne manipulácia s hodnotou, manipulácia v strome (pridávanie synov a bratov, presun v strome, odstránenie), výpisy (do MML, textu) a číslovanie. Význam tejto triedy ukazuje aj fakt, že je spolu so svojimi oddedenými triedami reprezentujúcimi jednotlivé typy elementov vo vlastnom balíku.

6 Záver

Existujúca aplikácia použiteľná na diktovanie matematických formúl je výsledkom analýzy danej problematiky, sformulovania základných požiadaviek a cieľov, teoretického návrhu riešení a nakoniec praktickej implementácie. Aj keď sa počas vývoja vyskytli niektoré problémy, prvotná analýza, ktorej bolo venované optimálne množstvo času, umožnila urobiť dostatočne dobrý návrh bez potreby väčších úprav pri implementácii.

Ako každý program, aj táto aplikácia je limitovaná dobou vývoja i výberom technológie. Je teda množstvo vecí, ktoré by sa dali v budúcnosti vylepšiť. Z technologického hľadiska je to určite modul rozpoznávania reči prehliadača Opera, ktorý by mohol mať podporu rozpoznávania i pre iné jazyky ako angličtina. Takisto modul syntézy s lepšou podporou SSML by pomohol aplikácii pôsobiť prívetivejšie. Z hľadiska prehľadnosti užívateľského rozhrania by pomohlo využitie viacúrovňové menu, ktoré by pomohlo podrobnejšie kategorizovanie položiek (napríklad operátory by sa rozdelili na operátory porovnania, množinové operátory, logické operátory...). Pre zlepšenie orientácie a rýchleho zoznámenia sa s aplikáciou pre nevidiaceho používateľa by pomohol systém kontextovej hlasovej pomoci. Schopnosti aplikácie by sa dali rozšíriť o ďalšie syntaktické prvky, napríklad možnosť aplikovať ľubovoľný symbol na operátor, čím by sa dosiahli efekty ako napríklad „$\vec{+}$”. Avšak po krátkej analýze sa ukázalo, že by to narušilo existujúcu intuitívnu syntax, čo by sa odrazilo napríklad vo zvýšenej potrebe zátvorkovania a znížila sa tak prístupnosť. Čo sa týka vhodnosti pre zrakovo postihnutých, nevýhodou bolo, že sa nenašiel dostatočný počet matematicky zbehlých používateľov na testovanie, takže len veľmi ťažko možno objektívne posúdiť intuitívnosť práce s aplikáciou pre túto skupinu používateľov.

Na záver môžem zhodnotiť, že aplikácia splnila väčšinu mojich očakávaní a má predpokladané schopnosti a vlastnosti, aké som si pri výbere témy predstavoval. Táto práca dala vzniknúť aj článku [25] a referát o nej odznie na výročnej konferencií ICCHP, ktorá sa uskutoční tohto roku (2006) v rakúskom Linzi.

7 Referencie

[1] *MathTalk*, <http://www.metroplexvoice.com/>
[2] Knuth, Donald E.: *The TeXbook*. Addison-Wesley, 1984
[3] Fateman, Richard: *How can we speak math?*, 2004
[4] Psutka, Josef: *Komunikace s počítačem mluvenou řečí*. ACADEMIA, 1995
[5] *JSpeech Grammar Format*, <http://www.w3.org/TR/2000/NOTE-jsgf-20000605>
[6] *Mathematical Markup Language (MathML) Version 2.0 (Second Edition)*, <http://www.w3.org/TR/2003/REC-MathML2-20031021/>
[7] Raman, T. V.: *Audio systems for technical readings*, 1994
[8] *Speech Synthesis Markup Language (SSML) Version 1.0*, <http://www.w3.org/TR/2004/REC-speech-synthesis-20040907/>
[9] *Voice eXtensible Markup Language (VoiceXML™) version 1.0*, <http://www.w3.org/TR/2000/NOTE-voicexml-20000505>
[10] *XHTML+Voice Profile 1.2*, <http://www.voicexml.org/specs/multimodal/x+v/12/spec.html>
[11] *XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition)*, <http://www.w3.org/TR/2002/REC-xhtml1-20020801>
[12] *Sphinx-4*, <http://cmusphinx.sourceforge.net/sphinx4/>
[13] *Hidden Markov Model Toolkit*, <http://htk.eng.cam.ac.uk/>
[14] *The Festival Speech Synthesis System*, <http://www.cstr.ed.ac.uk/projects/festival/>
[15] *The MBROLA Project*, <http://tcts.fpms.ac.be/synthesis/mbrola.html>
[16] *Opera Web Browser*, <http://opera.com/>
[17] IBM: *XHMTL+Voice Programmer's Guide*, 2004
[18] *Jetty Java HTTP Servlet Server*, <http://jetty.mortbay.org/jetty/index.html>
[19] *Integre techexplorer™ Hypermedia Browser*, <http://www.integretechpub.com/products/techexplorer/>
[20] *NATO phonetic alphabet*, <http://en.wikipedia.org/wiki/NATO_phonetic_alphabet>
[21] *Cascading Style Sheets*, <http://www.w3.org/Style/CSS/>
[22] *Tigra Menu*, <http://www.softcomplex.com/products/tigra_menu/>
[23] *Document Object Model (DOM) Level 2 HTML Specification*, <http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109>
[24] *JavaServer Pages*, <http://java.sun.com/products/jsp/>
[25] Hanakovič Tomáš, Nagy Marek: *Speech recognition helps visually impaired people writing mathematical formulas*, 2006

8 Prílohy

1. Príklad na VoiceXML formulár

Nasledujúci VoiceXML formulár demonštruje dialóg, kde je používateľ po otázke „či chce skončiť" požiadaný, aby odpovedal kladne alebo záporne. V prípade, že používateľ neodpovedá, počítač nerozumie alebo si používateľ vyžiada pomoc, prečíta počítač krátku informáciu a dialóg sa opakuje. V prípade, že odpovie záporne, počítač informuje o zrušení akcie.

Z technologického hľadiska je použité jedno pole s logickou gramatikou, spracovanie vyplnenia a odchytávanie udalostí.

```
<vxml:form id="main">
   <vxml:field name="ask" type="boolean">
      <vxml:prompt>Do you want to quit?</vxml:prompt>
      <vxml:filled>
         <vxml:if cond="ask == true">
            <vxml:prompt>Action cancelled.</vxml:prompt>
         </vxml:if>
      </vxml:filled>
      <vxml:catch event="help nomatch noinput">
         <vxml:prompt>
            This is a confirmation dialog, say yes or no.
         </vxml:prompt>
      </vxml:catch>
   </vxml:field>
</vxml:form>
```

2. Príklad na gramatiku v JSGF

Nasledujúca gramatika je (po úprave) prevzatá zo [17] a demonštruje vytvorenie „yes/no" gramatiky s množstvom slovných alternatív pre kladnú aj zápornú odpoveď, ale iba s jednou sémantikou – výrazy v zložených zátvorkách nastavujú „$" – objekt, ktorý bude vrátený ako výsledok rozpoznávania.

```
#JSGF V1.0 iso-8859-1;
grammar yes_no;
public <yes_no> = <yes> { $ = true } | <no> { $ = false };

<yes> = yes [please] | sure | okay | fine | yep | yup | affirmative;
<no> = no | nope | no thanks | negative;
```

3. Príklad na zápis textu v SSML

Nasledujúci príklad demonštruje použitie SSML značiek v hlasovom výstupe aplikácie pre rovnicu „$\frac{a}{b}+\sum_{i=0}^{n} c_i$". SSML značky sú použité na zmenu výšky hlasu a pauzy medzi blokmi. Za povšimnutie stojí použitie zníženého hlasu na indikáciu dolného indexu c_i.

```
a over b <break/>
plus <break/>
sum from <prosody pitch="-3st"> i equals zero </prosody>
to <prosody pitch="+3st"> n </prosody>
of c <prosody pitch="-3st"> i </prosody>
```

4. Príklad na zápis formuly v MathML

Vzorec z predošlého príkladu možno v MathML zapísať nasledovne:

```
<math>
    <mfrac>                          (zlomok)
        <mi> a </mi>
        <mi> b </mi>
    </mfrac>
    <mo> + </mo>                     (operátor +)
    <munderover>                     (horný a dolný index sumy)
        <mo> &Sum; </mo>
        <mrow>
            <mi> i </mi>
            <mo> = </mo>
            <mn> 0 </mn>
        </mrow>
        <mi> n </mi>
    </munderover>
    <mo> &ApplyFunction; </mo>       (aplikácia funkcie)
    <msub>                           (hodnota sumy)
        <mi> c </mi>
        <mi> i </mi>
    </msub>
</math>
```

5. Hlasový vstup

Globálne

Príkazy:

- prečítanie informácii o stave: *say program status*
- odvolanie poslednej akcie: (*undo* | *correction*) [<*number*> (*step* | *steps*)]

Menu:

- prečítanie titulkov položiek hlavného menu: *say menu items*
- otvorenie položky hlavného menu: prečítať titulok
- prečítanie titulkov podpoložiek hlavného menu: *say menu items*
- uzatvorenie položky hlavného menu: *close*
- spustenie podpoložky menu: prečítať titulok

Vkladací mód

Príkazy:

- prepnutie do navigačného módu: *navigation mode*
- presun bodu vkladania doprava/doľava: (*next* | *previous*) *point*
- nastavenie bodu vkladania: go to point <*number*>
- zatvorenie bodu(ov) vkladania: *close* (*point* | <*number*> *points* | *full*)

Vkladanie:

- výraz:
  <*number*> = *0..100* | (*0..9*)+
  <alpha> = [*not*] [*uppercase*] (*a..z*) // „not" je unárne –
  <*alphaS*> = <*alpha*> (and <*alpha*>)* // reťazec znakov
  <*alphaNum*> = (<*number*> <*alphaS*>) | (<*number*> xor <*alphaS*>)
  <*postOp*> = *squared* | *cubed* // na druhú, na tretiu
  <*alphaPost*> = <*alphaNum*> [<*postOp*>] // napríklad $14ab^2$
  <*vyraz*> = [<*alphaPost*>] (<*operator*> <*alphaPost*>)* [<*operator*>]
- grécke symboly: <*greek*> = *greek symbol* [*uppercase*] (alpha...omega)
- vlastná funkcia: function (<alpha> | <greek>)

Navigačný mód

Príkazy

- prepnutie do vkladacieho módu: *insert mode*
- presun výberu: *select* (*next* | *previous* | <*number*>)
- expanzia bloku (so zadaným číslom): (*expand* | *zoom in* | *go in*) [<*number*>]
- kontrakcia blokov: (*contract* | *zoom out* | *go out*)

6. Popis súborov aplikácie

src	***zdrojové kódy tried servera***
WebContent	***súbory pre klientskú časť servera***
js	***skripty***
menu	***skripty menu***
menu.js	*skripty jadra menu, od autorov*
menu_item.js	*položky, generované po kompilácii nastavení*
menu_tpl.js	*nastavenia rozmerov menu*
global.js	*globálne skripty aplikácie*
lineTable.sj	*skripty na generovanie matematického riadku*
menu.js	*aplikačné skripty pre menu*
styles	***štýly***
dialog.css	*štýly pomocných dialógov*
master.css	*štýly hlavnej stránky*
menu.css	*štýly menu*
voice	***súbory pre hlasové rozhranie***
audio	***rôzne zvuky prostredia***
grammars	***gramatiky***
alpha.pat	*šablóna pre písmená, dogenerujú sa slová*
commands.jsgf	*globálne príkazy*
greek.pat	*šablóna pre grécke písmená, dogenerujú sa slová*
ins_commands.jsgf	*príkazy vo vkladacom móde*
ins_navigation.jsgf	*navigácia vo vkladacom móde*
insert.jsgf	*vkladanie objektov*
linear.jsgf	*vkladanie výrazov*
menuItems.jsgf	*frázy vzťahujúce sa na položky hlavného menu*
nav_commands.jsgf	*príkazy v navigačnom móde*
nav_navigation.jsgf	*navigácia v navigačnom móde*
numbers.jsgf	*čísla*
structures.jsgf	*štruktúry*
menu_ins_frag_vxml.jsp	*vygenerované položky menu vo vkladacom móde*
menu_nav_frag_vxml.jsp	*vygenerované položky menu v navigačnom móde*
Web-inf	***skompilované triedy servera***
ask.jsp	*pomocné dialógové okno s otázkou*
index.jsp	*štartovacia stránka aplikácie*
main.jsp	*hlavná stránka aplikácie*
math.jsp	*stránka s MathML pre zásuvný modul*
search.jsp	*vyhľadávacie dialógové okno*
voiceFrame.jsp	*stránka s hlavným formulárom*
vzorecFrame.jsp	*rámec s formulárom obsahujúcim matematický rámec*
config.xml	konfiguračný súbor