VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

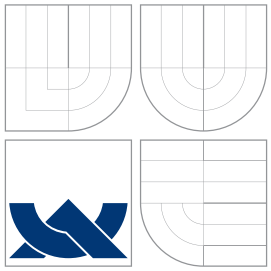
MULTIMEDIÁLNÍ PROHLÍŽEČ PRO PŘEDNÁŠKY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

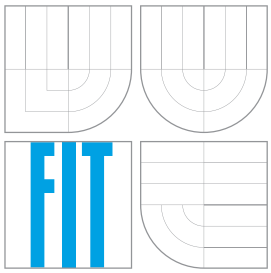
AUTOR PRÁCE
AUTHOR

JAKUB KUBALÍK

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MULTIMEDIÁLNÍ PROHLÍŽEČ PRO PŘEDNÁŠKY

MULTIMEDIA BROWSER FOR LECTURES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAKUB KUBALÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR SCHWARZ
SUPERVISOR

BRNO 2007

Abstrakt

V úvodní části čtenář pochopí význam toho projektu a proč vlastně vznikl. Jedná se o rozsáhlejší týmový projekt, proto popis jeho struktury představuje významnou část práce. Jádro prohlížeče zahrnuje řadu technik, jako např. komponentový systém, speciální systém vnitřní komunikace a další. Vysoký důraz je kladen na přenositelnost a znovupoužitelnost systému — bylo nutné jasně specifikovat pravidla pro syntaxi a používané datové formáty. Nezaměnitelnou úlohu v projektu hraje formát XML. Poslední úpravy představovali především vývoj nových komponent, které současně počítají s budoucím využitím projektu a jeho zapojením do komplexního vyhledávacího systému — ten představuje závěrečná kapitola.

Klíčová slova

identifikace jazyka, rozpoznávání plynulé řeči s velkým slovníkem, detekce klíčových slov, rozpoznávání a ověřování mluvčího, multimediální prohlížeč, grafické uživatelské rozhraní, metadata, obrazový bod

Abstract

In preamble reader will understand the purpose of this project and why it was initiated. It's large team project, so description of its structure represents main part of this report. A core of the browser contain a lot of techniques, e.g. system of components, special internal communication and other. The main goals of this system are portability and reusability — there is exact specification of syntax rules and used data formats. Non-interchangeable role in project has format XML. Invention of new components was the last intervention in project, with intention to use of the project in future and involve in complex search engine — the engine discuss final part of report.

Keywords

language identification, large vocabulary continuous speech recognition, keyword spotting, speaker identification and verification, multi-media browser, graphical user interface, metadata, pixel

Citace

Jakub Kubalík: Multimediální prohlížeč pro přednášky, bakalářská práce, Brno, FIT VUT v Brně, 2007

Multimediální prohlížeč pro přednášky

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval zcela samostatně pod vedením pana Ing. Petra Schwarze.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Kubalík
11. května 2007

Obsah

Zadání bakalářské práce/5811/2006/xkubal05

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Kubalík Jakub**
Obor: Informační technologie
Téma: **Multimediální prohlížeč pro přednášky**
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte s aktuálním stavem vývoje multimediálního prohlížeče a knihovnou wxWidgets, která je používána při jeho vývoji
2. Navrhněte úpravy pro použití prohlížeče pro přístup k videozáznamům přednášek. Prohlížeč musí umět zobrazit videozáznam přenášek, jeho textový přepis vygenerovaný rozpoznávačem řeči a umět v záznamu vyhledávat. Zaměřte se na uživatelskou přívětivost prohlížeče.
3. Navržené úpravy implementujte
4. Systém otestujte na záznamu přednášek jednoho vybraného kurzu.

Literatura:

- P. Schwarz: Initial work on multi-modal browser, seminář SERVITE, FIT VUT BRNO, <http://www.fit.vutbr.cz/research/groups/graph/servite/index.php.cs?page=seminars_06>

Při obhajobě semestrální části projektu je požadováno:

- Bod 1. a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Schwarz Petr, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Jakub Kubalík**
Id studenta: 83962
Bytem: Kvasiny 289, 517 02 Kvasiny
Narozen: 05. 05. 1984, Náchod
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Multimediální prohlížeč pro přednášky
Vedoucí/školitel VŠKP: Schwarz Petr, Ing.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Kapitola 1

Úvod

1.1 Jak všechno začalo?

1.1.1 Historie komunikačních prostředků

Rozvoj řeči jako základního prostředku komunikace mezi lidmi byl zásadním prvkem v rozvoji člověka. Jeho nástup umožnil jednotlivcům vyměňovat si nové informace a poznatky, případně i vynálezy, které ještě více urychlovaly jeho vývoj. Řeč jako taková byl silný prostředek při výměně poznatků hlavně mezi vrstevníky a do jisté míry i mezi generacemi. To značně usnadnil vynález písma.

Psaný text mohl zajistit předávání informace jak mezi jednotlivci a vrstevníky, tak i mnoha a mnoha generacemi. V počátcích byla hlavním problémem skutečnost, že číst a hlavně psát neuměl každý, resp. byla to spíš přednost malé skupiny populace. Nemalý vliv na to měla také finanční stránka věci, právě díky malé dostupnosti cena enormně narůstala. Dalším problémem při šíření psaného textu je značně omezená rychlost, kdy potřeba rozšířit psanou informaci mezi více jednotlivců vyžadovala vytvoření kopie — což znamenalo ruční přepis originálního textu, který podle rozsahu psané informace mohl představovat i několik měsíců práce. Navíc informace uchovaná na papíru neměla příliš dlouhou životnost.

Postupem času s příchodem nových vynálezů a technologií se šíření informací značně usnadnilo a zrychlilo. Zvláště pak s příchodem moderních komunikačních technologií, které umožnily jak hromadné šíření informací, tak i šíření informace na ohromné vzdálenosti. S dnešními technologiemi není problém komunikovat v ,,reálném čase" z jednoho konce světa na druhý.

1.1.2 Orientace v množství informací

Právě s nástupem moderních technologií je na místě automatizované strojové zpracování informací. Kapacita lidských zdrojů je totiž značně omezena a není možné se snadno orientovat v takových kvantech dat a informací, se kterými se člověk denně setkává.

Automatizované zpracování informací — v našem případě se budeme zajímat především o zpracování řeči a přirozeného jazyka — zasahuje do mnoha odvětví a zabývá se jím mnoho skupin lidí. Jednou z nich je právě výzkumná skupina zpracování řeči na naší fakultě vedená panem Doc. Černockým. Tato skupina se zpracováním řeči a přirozeného jazyka zabývá už od svého vzniku v roce 1997. Za tu dobu vznikla řada projektů v tomto oboru, které využívají a dále rozvíjí technologie jako např. identifikace jazyka (LID), rozpoznávání plynulé řeči s velkým slovníkem (LVCSR), detekce klíčových slov (KWS) nebo rozpoznávání

a ověřování mluvčího (SpkID, SpkVer).

Většina těchto aplikací je vytvořena jako samostatný modul a často jsou tyto aplikace snadno transformovatelné do terminálové podoby, která by mohla být součástí jakéhokoli systému. Teorie je totiž taková, že veškeré náročné procesy, jako jsou právě různé rozpoznávače a aplikace vytvářející indexy ke zvukovým případně i video záznamům, by běžely na vybraném výkonném stroji, který by tyto indexy uchovával a jednotliví klienti by pak vyhledávání realizovali pouhým dotazem na takový server. O tom však bude řeč ještě později (viz kapitola 4.3).

1.1.3 Klientský přístup

Doposud byl pro testovaní a prezentaci výsledků různých rozpoznávačů a aplikací využíván software švýcarského výzkumného institutu IDIAP nazvaný JFerret, který je součástí jejich projektu MultiModal Media Fileserver. [6]

JFerret je flexibilní multimediální prohlížeč, který díky své struktuře se zásuvnými moduly umožňuje uživateli definovat konfiguračním souborem jeho vzhled, funkci a chování. Prohlížeč obsahuje zásuvné moduly pro prezentaci video záznamů, audio záznamů, snímků prezentace, dále pak moduly s ovládacími prvky nebo různými časovými osami. [5]

Popsaný typ multimediálního prohlížeče je typ aplikace, která se přesně hodí na prezentaci a testování výsledků různých rozpoznávačů a indexačních aplikací, JFerret má však tu nevýhodu, že je psaný v programovacím jazyce Java, což omezuje výkon aplikace a navíc, pokud se v průběhu vývoje rozpoznávacích systémů vyskytne nějaký požadavek na rozšíření prohlížeče, značně to vývoj zdrží.

Dalším místem, kde vznikla možnost využití multimediálního prohlížeče, je právě naše fakulta, jelikož veškeré přednášky probíhající v hlavních prostorách přednáškových sálů jsou zaznamenávány a ukládány na video servery ve velice dobré kvalitě. Právě tyto záznamy poskytují velké množství testovacího materiálu pro rozpoznávače. Díky tomu také vznikla myšlenka, kdy by možnost vyhledávání ve vybraných záznamech přednášek byla umožněna přímo studentům. To všechno iniciovalo projekt nazvaný Presentation As Synchronized Experience. [2]

1.2 Orientace v dokumentu

Otázka vytvoření multimediálního prohlížeče ve výzkumné skupině zpracování řeči vznikla už před několika lety a intenzivně se na prohlížeči pracuje už více jak rok. I když intenzivně není možná to pravé slovo — do projektu je zapojeno jen několik členů a mimo jiné současně pracují i na jiných projektech, proto není vývoj až tak rychlý.

V době, kdy jsem se do projektu zapojil, bylo jádro prohlížeče bez některých detailů téměř hotové — právě strukturou jádra a vlastně celého prohlížeče se zabývá kapitola 2. Dalším zásadním úkolem ve vývoji prohlížeče byly jednotlivé komponenty, které zajišťují interakci s uživatelem a jsou vlastně stěžejní části prohlížeče, jelikož určují jeho vzhled a funkci, což je pro uživatele to nejdůležitější vodítko — jádro samotný uživatel vůbec nevidí a jeho přítomnost, i když je vlastně základem, vůbec nevnímá. Vývoj jednotlivých komponent byl můj hlavní úkol, tím se zabývá kapitola 3. Mimo popisu komponent se v ní také dozvíte o některých rozšířeních a pomocných třídách, které bylo nutné do prohlížeče přidat a také o několika malých změnách přímo v jádře prohlížeče.

Vývoj prohlížeče je v současné době v takovém stádiu, že je možné některé základní funkce bez potíží testovat i se stávajícím omezeným počtem dostupných komponent. Jed-

notlivé komponenty jsou však stále ve vývoji a už nyní existují návrhy na další komponenty, které budou v prohlížeči potřeba. Funkce samotného prohlížeče je však značně omezena — v budoucnu by měl být prohlížeč součástí většího projektu, který by společně s různými rozpoznávači a indexačními servery tvořil rozsáhlý systém pracující na mnoha různých počítačích a platformách. S tím se také při jeho vývoji počítalo — prohlížeč tak už v současné době zahrnuje podporu a komponenty pro komunikaci s těmito systémy. Kapitola 4 se zabývá dalšími rozšířeními, které by k prohlížeči měli v budoucnu přibýt a diskutuje také změny, jež prohlížeč bude muset pravděpodobně podstoupit. Nakonec je představen onen rozsáhlý vyhledávací systém, jehož součástí je vyvíjený prohlížeč a osvětluje jeho roli v celém systému.

Kapitola 2

Struktura projektu prohlížeče

2.1 Vstupní předpoklady

Projekt má za cíl vytvořit nástroj, který by mohl sloužit jak pro testování a prezentaci nových vytvářených technologií v oboru rozpoznávačů, tak i pro praktické nasazení mezi běžné uživatele — v současné době jsou technologie podobného typu relativně málo dostupné a hlavně se teprve začínají dostávat na takovou úroveň, kde jejich využití je těžko nahraditelné. Ale hlavním cílem je nahrazení současně používaného prohlížeče JFerret, proto se základní struktura do jisté míry podobá právě tomuto prohlížeči. Pro vývoj byly proto stanoveny tyto hlavní milníky:

- Snadno konfigurovatelný prohlížeč.
- Snadno rozšiřitelný o nové technologie a nové funkce.
- Zpracování toku multimediálních dat (např. přehrávání video či audio záznamů, dále zobrazování snímků prezentací případně různých přepisů řečových rozpoznávačů nebo jejich výstup).
- Možnost online distribuce dat (živé přednášky — zpracování výstupu rozpoznávačů v reálném čase).
- Vyhledávač a vývojové nástroje s otevřenou architekturou a otevřenými zdrojovými kódy.

2.2 Vývojové prostředky

Vzhledem k požadavku na vysoký výkon aplikace a dále s předpokladem vývoje modulárního systému byl použit programovací jazyk C++ se standardním překladačem GCC. Dalším zásadním předpokladem byla přenositelnost mezi různými operačními systémy — v případě použití jazyka C++ je toto splněno, navíc protože se jedna o uživatelské rozhraní bylo s výhodou využito konceptu oken. Toto vše do jisté míry splňuje knihovna *wxWidgets*.

2.2.1 Knihovna *wxWidgets*

wxWidgets je jakýsi programátorský soubor nástrojů pro psaní aplikací s grafickým uživatelským rozhraním (GUI). Je to systém, který za nás provádí spoustu práce a přímo nám

poskytuje standardní aplikační chování. Knihovna *wxWidgets* obsahuje řadu tříd a metod, které jsou programátorovi k dispozici, popř. je možné je podle potřeby přizpůsobit k obrazu svému. Aplikace obvykle zobrazí okno obsahující standardní ovládací prvky, případně vykresluje jiné speciální prvky a zpracovává vstup od myši, klávesnice nebo jiných zdrojů. Aplikace můžou prostřednictvím knihovny také komunikovat s jinými procesy případně řídit další programy. Jinými slovy řečeno, *wxWidgets* velmi usnadňuje práci programátora při psaní aplikace, která provádí obvyklé operace stejně jako moderní aplikace.

Zatímco *wxWidgets* je často označován za soubor nástrojů pro tvorbu GUI aplikací, ale ve skutečnosti je mnohem víc a zahrnuje možnosti, které jsou využívány v mnoha situacích při vývoji aplikací. To je právě v případech, kdy aplikace pod *wxWidgets* musí být přenositelná mezi různými platformami, ne jen část grafického uživatelského rozhraní. Knihovna *wxWidgets* navíc poskytuje mnoho tříd pro práci se soubory, různými toky dat, vícevláknovými aplikacemi, meziprocesorovou komunikací, přístupem k databázím a mnoho dalšího. [7]

2.2.2 Prostředí a přenositelnost

Jak už bylo řečeno zásadní vlastností při návrhu byla přenositelnost. To je samozřejmě při použití standardního GNU překladače GCC s kombinací s knihovnou *wxWidgets* splněno. Pod operačním systémem unixového typu nevznikají žádné větší problémy, jelikož překladač GCC je standardní součástí tohoto systému. U operačního systému Windows je situace trochu jiná, ne však neřešitelná. Navíc její řešení je na místě, protože většina řadových uživatelů bude prohlížeč spouštět právě pod operačním systémem Windows.

Problém operačního systému Windows je, že požadovaný překladač není součástí. Existuje však dokonce více možností jak se s tímto vypořádat a to výběr některého z balíků nástrojů, které mimo jiné obsahují právě GCC překladač. Jednou možností je systém MinGW.

MinGW je kolekce volně dostupných a volně šiřitelných hlavičkových souborů speciálně pro Windows a knihoven kombinovaných s GNU nástroji, které umožňují vytvářet nativní programy pro Windows. Současně vytvořená aplikace není závislá na žádných dalších DLL knihovnách. [3]

Další možností v operačním systému Windows je systém Cygwin. Ten podobně jako MinGW obsahuje hlavičkové soubory a knihovny potřebné k překladu projektu, navíc systém Cygwin emuluje unixové prostředí - např. podporuje unixový zápis cest k souborům (lépe řečeno vyžaduje) nebo jiné systémové proměnné. Systém Cygwin sám o sobě obsahuje také nástroje, které umožňují např. spouštění skriptů napsaných v obvykle používaných skriptovacích jazycích podobně jako v operačních systémech Unix a další nástroje, které jsou v takových systémech běžné. O systému Cygwin a MinGW bude ještě řeč – hlavně problémy spojené s instalací a překladem knihovny *wxWidgets* a překladem projektu prohlížeče. Ty jsou diskutovány v kapitole 3.3.

2.2.3 Datové formáty

S přihlédnutím k snadnější orientaci uživatele je pro uchovávání konfigurace, interní komunikaci a v mnoha situacích a strukturách s výhodou využito formátu XML. Samotný formát XML nedefinuje konkrétní informaci, ale pouze způsob, jakým se zapisuje nebo jakým jsou data uspořádána (tzv. metadata). Právě to umožňuje využít jej na mnoha místech, v prohlížeči téměř všude.

Například rozmístění ovládacích prvků, přehrávačů a dalších grafických prvků v hlavním okně je definováno v souboru `layout.xml` právě ve formátu XML. Zároveň je možné v tomto souboru nalézt základní nastavení jednotlivých prvků, jako např. velikost prvku, vstupní podmínky či inicializační podmínky, omezení či podmínky vnitřní komunikace a další parametry, které je nutné znát při spuštění aplikace. Dále je formát XML využit při interní komunikaci mezi jednotlivými moduly a komponentami (o komponentách bude řeč v kapitole 2.3.1). A v neposlední řadě se formát XML objevuje i na datových vstupech různých komponent.

2.2.4 Pravidla pojmenovávání prvků a syntaxe

Pro lepší orientaci mezi zdrojovými soubory bylo definováno i pojmenovávání souborů. Základním pravidlem pro pojmenovávání je uvození jména písmeny ,,MB" jako Multi-media Browser (anglicky multimediální vyhledávač). Toto však platí pouze pro soubory, které jsou neoddělitelně spjaty s projektem. Soubory a moduly, které jsou více obecnější a můžou fungovat samostatně nebo lépe mohou být prospěšné i u jiných projektů, toto označení nemají. Příkladem může být XML parser, jež je implementován v souborech `sxmlparser.h` a `sxmlparser.cpp`.

Dalším pravidlem je rozšíření označení u modulů pracujících jako komponenty. Takové soubory jsou stejně jako ostatní uvozený písmeny ,,MB", ale navíc je k nim přidáno písmeno 'C' jako Component (anglicky komponenta). Aby bylo rozlišení ještě zřetelnější je k uvození přidán navíc znak '_' (podtržítko). Příkladem může být komponenta `Button`, kde korespondující soubory jsou nazvány `mbc_button.h` a `mbc_button.cpp`.

Podobná pravidla platí i pro pojmenovávání jednotlivých tříd či komponent, tzn. jednotlivá jména jsou uvozena písmeny ,,MB". Výjimkou jsou zde však jména komponent – prefix nezahrnuje znak 'C', ale klíčové slovo ,,Component" je celé přidáno za samotné jméno komponenty. Pokud bychom příklad ukazovali opět na komponentě `Button`, celé její jméno by bylo `MBButtonComponent`, podle kterého je možné implementaci komponenty najít ve zdrojových souborech.

Nicméně toto značení je závazné pouze pro programátora. Pro uživatele využívající hotovou aplikaci je označení komponent už bez jakýchkoli předpon či přípon, které jsou skryté. Také proto pokud budu hovořit o nějakém souboru, třídě popř. o komponentě, budu často uvádět pouze její jméno bez uvozujících dvou písmen ,,MB" a dalších označení. Ovšem při vyhledávání vybraného jména ve zdrojových souborech je nutné s těmito označeními počítat a je dobré je uvést.

Tyto pravidla byla stanovena hlavně pro snadnější orientaci ve zdrojových souborech nejen pro člověka, který je právě píše, ale i pro ostatní členy programátorského týmu. Navíc projekt by měl mít zdrojové soubory veřejně přístupné, v tom případě pak usnadní orientaci i případnému externímu zájemci. Proto byly také přijaty některé zásady psaní samotných zdrojových kódů, které jsou volně přístupné na internetových stránkách [4].

2.3 Architektura prohlížeče

Prohlížeč je stavěn jako interaktivní aplikace jejíž základním stavebním prvkem je okno. Jinak by se taková aplikace dala nazvat také desktop aplikace nebo přesněji událostmi řízená aplikace. Základem takové aplikace je smyčka (anglicky idle loop) ve které se čeká na příchod nějaké události, která je posléze, je-li relevantní, zpracována. Příkladem takové události může být změna pozice myši či stisknutí nějakého tlačítka, stisknutí nebo uvolnění

klávesy na klávesnici, dokončení zpracování nějakého vstupu či bloku dat apod. V době, kdy není obsluhována žádná událost je možné provádět jiné procedury, jež není nutné provést ihned a mohou být přerušeny právě příchodem události. Tento koncept aplikací umožňuje efektivní využívání času procesoru, kdy daný proces běží jen v době, kdy se zpracovává nějaká událost.

Takto jsou stavěny právě aplikace, jejíchž základem je knihovna *wxWidgets*. Ta obsahuje základní prostředky pro vytvoření ,,idle smyčky" a pro zpracovávání událostí. Klíčovým prvkem v knihovně *wxWidgets* je třída `wxApp`. Přetížením některých jejích metod můžeme nadefinovat chování aplikace, např. metoda `OnInit` je volána ihned po spuštění aplikace — prostřednictvím ní můžeme inicializovat struktury a provést operace, které je nutné provést ještě před uvedením celé aplikace do chodu. Podobně např. u metody `OnExit`, která je volána před ukončením aplikace, a mnoho dalších.

Pro snadnější vytvoření událostmi řízené aplikace knihovna *wxWidgets* obsahuje předdefinovaná makra např. `DECLARE_APP`, která deklaruje hlavní funkci pro vytvoření instance naší aplikace. Dále např. makro `IMPLEMENT_APP`, které onu instanci vytvoří a v neposlední řadě makra `BEGIN_EVENT_TABLE`, `EVT_IDLE` nebo `END_EVENT_TABLE` pomocí nichž vytvoříme idle smyčku pro zpracování událostí. Implementaci základní struktury aplikace nalezneme v modulu `MBApp`.

Hlavní částí, která určuje funkci prohlížeče, je však třída `Desktop`. Ta zajišťuje načtení a zpracování konfiguračního souboru, přípravu pracovního prostředí, inicializaci všech potřebných komponent, jejich zprávu, ustavení komunikační fronty zpráv a zprostředkovává komunikaci mezi jednotlivými částmi prohlížeče. Význam tohoto modulu je zřetelný i z obrázku 2.1, který současně znázorňuje celou architekturu prohlížeče. O dalších částech prohlížeče pojednávají následující podkapitoly.

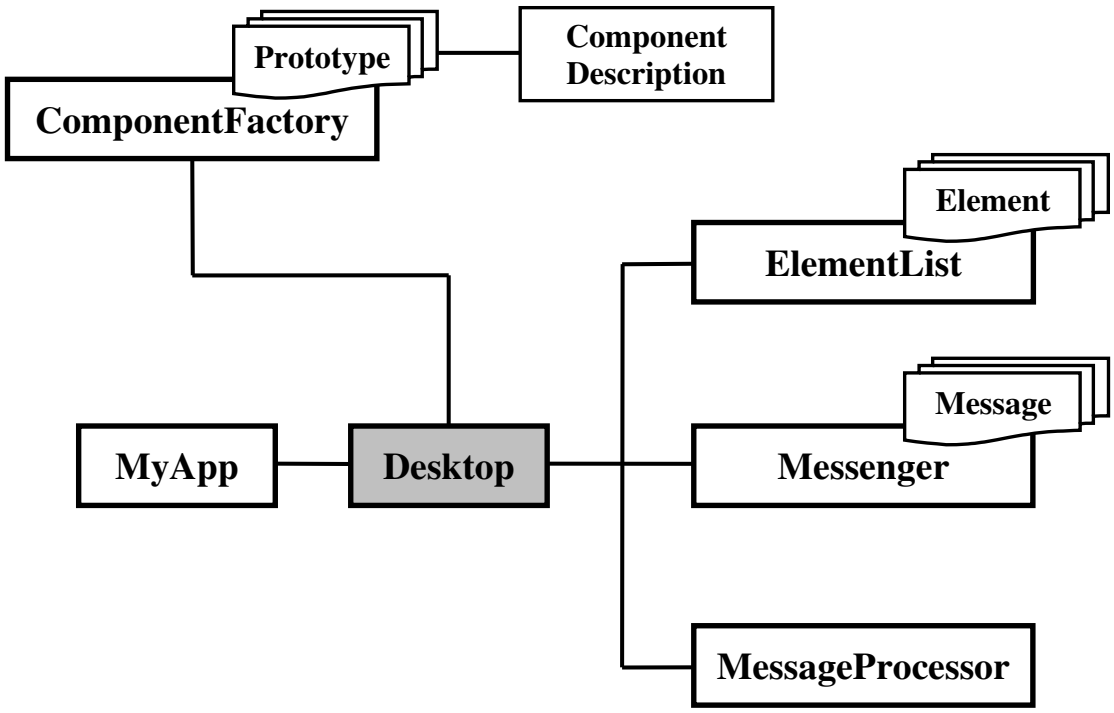
2.3.1 Systém komponent

Cílem návrhu prohlížeče bylo zajistit pozdější snadnou rozšiřitelnost. Z tohoto důvodu byl do návrhu zařazen tzv. komponentový systém. Ten zásadně ovlivňuje celou architekturu prohlížeče, přináší sebou také nutnost speciální interní komunikace — podrobněji se jí zabývá kapitola 2.3.4.

Podpora komponent hnízdí už v samotném modulu Desktop, kde je udržován aktuální seznam dostupných komponent a informací o nich. K tomuto účelu slouží právě modul `ComponentFactory` s třídami `MBComponentFactory` a `MBComponentPrototype`. Třída `MBComponentPrototype` obsahuje struktury a prostředky pro uchování a správu informací o jedné komponentě — vytváří jakýsi popis nebo-li prototyp či vzor komponenty, který mimo jiné zahrnuje i ukazatel na funkci pro vytvoření instance dané komponenty. Třída `MBComponentFactory` pak vytváří seznam prototypů komponent a poskytuje jej hlavnímu modulu.

Struktury s prototypy jednotlivých komponent společně s funkcemi pro vytvoření instance dané komponenty se nacházejí v modulu `IntComp`, konkrétně pak v souboru `mbintcomp.cpp`. Jednotlivé struktury s popisem jsou zřetězeny do lineárního seznamu a exportovány prostřednictvím globální proměnné `gpInternalComponents`. Tento seznam je zpracován modulem `ComponentFactory` při inicializaci aplikace ještě před načtení samotného konfiguračního souboru, jelikož je nutné znát právě ony funkce pro vytvoření instancí jednotlivých komponent.

Dalším modulem, který souvisí s komponentovým systémem, je `ElementList` zahrnující třídu `MBElementList` a třídu `MBElement`. Třída `MBElement` představuje instanci jedné kom-



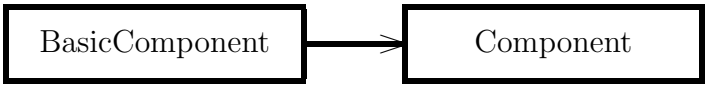
Obrázek 2.1: Architektura multimediálního prohlížeče

ponenty a třída `MBElementList` seznam instancí. `MBElement` v sobě zahrnuje mimo instance dané komponenty také její název a jí příslušející XML konfiguraci a další parametry potřebné k jejímu použití. `MBElementList` v tomto případě nepředstavuje seznam všech dostupných komponent, ale pouze seznam instancí[1] právě použitých komponent, pro něž známe konfiguraci načtenou z hlavního konfiguračního souboru `layout.xml`.

Samozřejmě komponentový systém by nemohl existovat bez samotných komponent. Každá komponenta, aby mohla být zpracována prohlížečem, musí mít standardizované rozhraní. Standardizovaným rozhraním se v objektu komponenty myslí přítomnost metod pro akceptování konfigurační informace, metod pro svázání komponenty s jádrem prostřednictvím ukazatelů na specializované funkce nebo např. metody pro zpracování příchozích dat a interních zpráv. Tyto základní funkce poskytuje třída `BasicComponent`. Další prvky pro snadnou komunikaci komponentám přidává odvozená třída `Component`, a to např. oddělení chybového výstupu od výstupu pro standardní výpisy a varování nebo např. metody pro obousměrnou interní komunikaci, tedy příjem i odesílání zpráv.

Při vytváření nové komponenty tak jednoduše použijeme připravenou třídu `MBComponent` a novou komponentu odvodíme od ní a případně přetížíme vybrané metody pro podrobnější specifikaci funkce dané komponenty. Vztah jednotlivých tříd je vidět na obrázku 2.2.

[1] Instance jednotlivých komponent se vytváří z jejich prototypů, o které se stará modul `ComponentFactory`.



Obrázek 2.2: Třídy nevizuálních komponent

2.3.2 Vizuální komponenty

Třídy `BasicComponent` a `Component` poskytují rozhraní pro propojení komponent s prohlížečem, komunikaci s jádrem a zprávu chyb. Neposkytují nám však prostředky pro vizualizaci a propojení s pracovní plochou a v neposlední řadě také interakci s uživatelem. Tuto funkci nevizuálním komponentám přidává třída `wxComp` odvozená právě od třídy `Component` jak je zřejmé z obrázku 2.3.

Obrázek 2.3: Třída vizuální komponenty

Vizuální komponenta mimo standardního rozhraní, které přebírá od nevizuální komponenty, zahrnuje další parametry jako např. rozměry (pozice a velikost), různé příznaky spojené s vykreslováním a dále metody pro inicializaci a překreslování komponenty atd.

Při vytváření vizuální komponenty však nestačí pouze vytvořit instanci třídy `wxComp`, navíc je nutné vytvořit speciální třídu odvozenou od zvolené třídy *wxWidgets* implementující vizuální komponentu a přidat k ní rozhraní pro interní komunikace s jádrem prohlížeče. K tomuto účelu bylo vytvořeno speciální makro `MBVISUAL_INTERFACE`, které jednoduchým vložením do vytvořené třídy přidá požadované rozhraní. Poté už programátor pouze při vytváření komponenty vytvoří instanci vizuální části komponenty pomocí makra `MBVISUAL_LINK`, které propojí komponentu s její vizuální částí a vytvořenou instanci si uloží do parametru `mpInstance` třídy `wxComp`.

2.3.3 Přizpůsobitelný vzhled

Jak už bylo několikrát zmíněno, uživatel si může jednoduše upravit vzhled celého prohlížeče (resp. rozmístění jednotlivých komponent) podle svých požadavků. Nastavení vzhledu se definuje v XML formátu a celé je uložené v souboru `layout.xml`, který se při každém spuštění prohlížeče načte a teprve poté se vytvoří seznam komponent, jež uživatel v souboru nadefinoval. To umožňuje snadnou dynamickou změnu vzhledu aplikace aniž by bylo nutné znovu celý projekt překládat. Konfigurační soubor má hierarchickou strukturu právě diky použitému formátu XML a je potom možné se v něm snadno orientovat. Pro jistotu si ukážeme krátký příklad:

```
<layout name="main">
    <window name="window1" title="Testovací okno"
        left="25%" top="33%" width="50%" height="33%"/>
        <textbox name="textbox1" text="Můj text"
            left=10 top=10 width=250/>
        <button name="button1" text="Moje tlačítko"
            left="10%" top="50%" width="250"
            onclick="<message name=\"settext\" text=\"TEXT\"/>"/>
    </window>
</layout>
```

Hierarchie souboru naznačuje vztahy mezi jednotlivými objekty — kořenem konfiguračního souboru je položka `layout`, která uzavírá nastavení celého prohlížeče a představuje jakousi virtuální plochu, na kterou budeme umisťovat jednotlivé komponenty. Základním objektem který umístíme na plochu je potom okno (položka `window`). Jedná se o klasické okno se záhlavím, kde se objeví systémové menu a napravo tlačítka pro uzavření, minimalizaci a maximalizaci. Jako titulek okna se použije parametr `title` položky `window`, v našem případě ,,Testovací okno".

Rozměry každého prvku je možné zadat buď absolutně (v obrazových bodech — pixelech) nebo relativně vůči nadřazenému prvku (v procentech). V případě našeho okna je nadřazený prvek celá obrazovka — okno by podle nastavených rozměrů mělo být umístěno ve středu obrazovky a mělo by zabírat polovinu plochy v horizontálním směru a třetinu ve vertikálním. Na pracovní plochu je samozřejmě možné umístit více oken a né pouze jedno, jak je ukázáno v příkladu.

Parametrem každé komponenty a každého objektu je `name`, který dává danému objektu jméno, přes něj se poté na vybraný objekt můžeme odkazovat nebo jej identifikovat[2]. Naše okno je pojmenované `window1`.

V uvedeném konfiguračním souboru okno `window1` obsahuje dvě komponenty, a to jedno editační okno (`textbox`) a jedno tlačítko (`button`). Rozměry obou komponent se nastavují stejnými parametry jako v případě hlavního okna, tedy parametry `left`, `top` či `width`, `height`. Navíc každá komponenta definuje další akceptovatelné parametry, které jsou specifické právě pro danou komponentu — např. u editačního okna je to text, který se uvnitř objeví po spuštění aplikace. Tlačítko navíc akceptuje parametr `onclick`, který definuje akci, která se provede při jeho stisku. V uvedeném příkladu se po stisku tlačítka odešle interní zpráva s názvem `settext` a parametrem `text` nastaveným na hodnotu ,,TEXT". Co toto znamená si řekneme v následující kapitole.

2.3.4 Interní komunikace

Pro synchronizaci a komunikaci jednotlivých částí prohlížeče a hlavně jednotlivých komponent byl v prohlížeči vytvořen speciální modul pro interní komunikaci pomocí zpráv, resp. jedná se o více souvisejících modulů. Klíčovým modulem zpracování zpráv je modul `Messenger`. Jeho základem je fronta zpráv, do které se zařazují příchozí právy. Fronta je

[2]Pokud není jméno v konfiguračním souboru uvedeno, automaticky je mu vygenerováno jméno skládající se z názvu komponenty a indexu v hranatých závorkách, např. `window[1]`

opatřená i jednoduchou správou paralelního přístupu pomocí semaforu — zpracování je však zatím sekvenční proto není aktuálně využit.

Zařazování zpráv do fronty probíhá asynchronně podle toho, kdy nějaká komponenta nebo modul potřebuje odeslat zprávu. Na druhé straně rozesílání zpráv je prováděno v idle smyčce tak, aby nebyly porušeny zásady interaktivní aplikace a rozesílání nezdržovalo aplikaci v době, kdy to není třeba. Každá zpráva je totiž implicitně rozesílána všem ostatním modulům a komponentám v prohlížeči. Omezení příjmu některých zpráv na jednotlivých komponentách je možné definovat v konfiguračním souboru přímo parametrem dané komponenty. K tomuto účelu slouží dva parametry — jeden pro specifikaci zpráv, které má komponenta přijímat (`amsg` — Accept Messages) a druhý pro filtraci zpráv, které komponenta přijímat naopak nemá (`dmsg`). Oba parametry specifikace akceptují zápis ve formě standardních regulárních výrazů ze standardu POSIX:

- `.` → reprezentuje jakýkoli znak
- `*` → nula, jeden nebo více opakování předchozího znaku
- `+` → jeden nebo více opakování předchozího znaku
- `[xyz]` → jeden z množiny znaků

Pokud bychom jedné komponentě chtěli přiřadit více specifikací zpráv, zadáme je jednoduše za sebe do příslušného parametru oddělené ';' (středníkem). Abychom mohli sestavit nám vyhovující regulární výraz musíme ovšem znát způsob, jakým prohlížeč zprávy formátuje a adresuje. Základní prototyp adresy zprávy je:

```
layout@component_name:message_name
```

kde `layout` je název nadřazené pracovní plochy — v konfiguračním souboru uzel `layout` a jeho parametr `name`. Položka `component_name` představuje název komponenty, která je odesílatelem zprávy a `message_name` název samotné zprávy. Takže pokud bychom např. chtěli přijímat zprávy od jakékoli komponenty s názvem `settext`, do parametru `amsg` dané komponenty zapíšeme `.*:settext`. Konkrétní vnitřní reprezentaci zprávy a její adresy si ukážeme na příkladu:

```
<message name="settext" text="TEXT"/>
```

Pokud by tuto zprávu odeslala komponenta tlačítka se jménem `button1` mohla by její adresa vypadat například takto:

```
main@button1:settext
```

Adresa této zprávy odpovídá právě uvedenému regulárnímu výrazu a proto by byla komponentou přijata a posléze zpracována.

Jak jste si jistě všimli, samotná zpráva je předávána jako text formátovaný pomocí XML. Tím je docíleno nezávislosti na systému a zpráva může mít teoreticky neomezenou délku a komponenty si tak mezi sebou mohou vyměňovat např. zpracovávaná data. Vnitřně je zpráva zabalena do třídy `Message`, která kromě samotného XML textu zprávy obsahuje také odesílatele zprávy, tedy odkaz na komponentu, která danou zprávu vytvořila, a další potřebné parametry.

V prohlížeči kromě klasických zpráv, kterými si komponenty předávají informace a komunikují, existují ještě zprávy systémové. Tyto zprávy jsou místo rozeslání všem komonentám z fronty zpráv předány modulu `MessageProcessor`. Formát systémové zprávy i její adresa je shodná s obyčejnou zprávou, pouze je jméno zprávy uvozeno klíčovým slovem „`system.`". Pokud modul `Messenger` narazí na takovou zprávu, okamžitě ji předá modulu `MessageProcessor` a zpracování nechá na něm. Příkladem takové zprávy je zpráva `remove`, která způsobí odstranění komponenty nebo dokonce i celé skupiny komponent z pracovní plochy za běhu aplikace.

```
<message name="system.remove" component="button1">
<message name="system.remove" group="group1"
```

Další systémovou zprávou je `loadlayout`, která způsobí načtení nového konfiguračního souboru, opět za běhu aplikace. S tímto systémem je možné dynamicky měnit vzhled celé aplikace aniž bychom museli prohlížeč znovu spouštět.

```
<message name="system.loadlayout" url="file.xml">
```

2.3.5 Zpracování XML

Jak jsme se přesvědčili, ve formátu XML probíhá interní komunikace, definuje vzhled prohlížeče v konfiguračním souboru a navíc některé komponenty data zpracovávají také ve formátu XML. Formát XML hraje v prohlížeči významnou roli, z tohoto důvodu byl také vytvořen modul pro zpracování XML přímo pro prohlížeč.

Existuje sice spousta dostupných knihoven zpracovávajících tento formát, ale při použití externí knihovny je v mnoha případech nutné dodržovat jistá licenční pravidla a ustanovení nebo při nejhorším knihovna nevyhovuje přesně požadavkům programátora. Navíc při překladu prohlížeče je nutné aby použitá externí knihovna byla dostupná použitému překladači, což vždy nemusí být implicitní. Vlastní zabudovaná knihovna tak usnadňuje práci jak programátorovi, tak případně uživateli, který by prohlížeč sestavoval ze zdrojových kódů.

Zabudovaný modul pro zpracování XML (zkráceně parser) je v celku jednoduchý a přitom umožňuje snadnou manipulaci a použití. Co se týče čtení můžeme funkci parseru shrnout do dvou fází — v první vstupní soubor popř. samotný XML text zpracuje a celý

strom uloží do paměti. Takový přístup může v některých případech, zvláště pokud nás zajímá jen malá část vstupního XML, znamenat zdržení, většinou se však zpracovává celý vstup a přístup k jednotlivým uzlům je velice efektivní – ten probíhá pomocí operátoru `[]` (indexace). To si ostatně můžeme ukázat na krátkém příkladu:

```
SXMLDocument doc;           // objekt XML parseru
doc.Load("layout.xml");     // načtení stromu ze souboru
doc.ParseText(text);        // načtení stromu přímo z proměnné
```

Nyní je celý strom vstupního XML načtený v paměti a můžeme s ním snadno pracovat. Prostřednictvím operátoru `[]` můžeme jak číst:

```
text = doc["main.window#title"];
```

tak i zapisovat:

```
doc["main.window.textbox(name=textbox1)#text"] = "text1";
```

Použijeme-li například XML uvedené v kapitole 2.3.3, v prvním případě potom proměnná `text` bude obsahovat titulek okna `window1` a to ,,Testovací okno". Po provedení příkladu zápisu změníme parametr `text` komponenty `textbox1` z původního ,,Můj text" na ,,text1".

Jak je vidět práce s tímto XML parserem je celkem snadná a intuitivní.

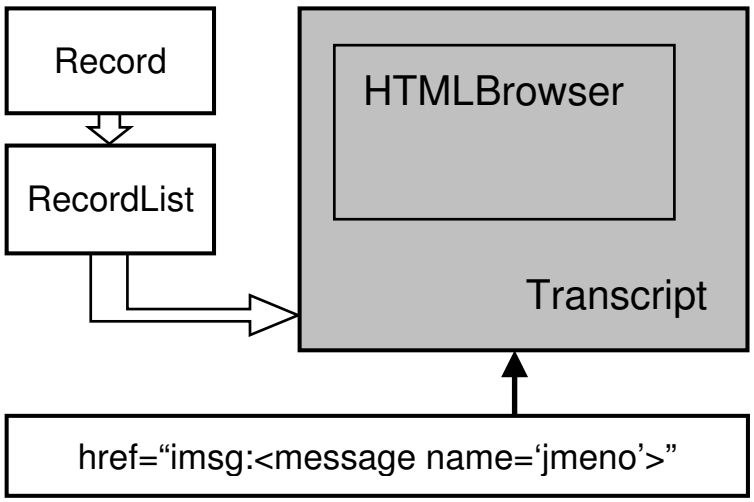
Kapitola 3

Současný stav vývoje

Tato kapitola popisuje a vysvětluje hlavně současný stav vývoje prohlížeče a nové součásti navržené a naimplementované od doby, kdy jsem se do projektu zapojil. Na projektu současně spolupracovalo více členů týmu, ale vesměs zde budu popisovat pouze ty součásti, na kterých jsem pracoval přímo já. Pokud se nebude jednat o mou práci, bude to explicitně zdůrazněno.

3.1 Pomocné třídy

Prvním problémem, který bylo nutné vyřešit, byla potřeba nějakým způsobem uchovávat textové přepisy jednotlivých video resp. audio záznamů. Původním návrhem byla struktura, která by zahrnovala jednotlivé parametry každého záznamu jako je např. samotný text přepisu, čas zahájení, čas ukončení, kód aktuálního kanálu nebo i barva pozadí příslušného kanálu a další informace relevantní danému předpisu. V tomto projektu byl však vždy kladen důraz na univerzálnost, proto vznikly třídy Record a RecordList.



Obrázek 3.1: Komponenta Transcript a použití tříd Record a RecordList

3.1.1 Třída Record

Primárně tato třída slouží k uchovávání parametrů jednoho záznamu nějakého textového přepisu a hodnoty tohoto parametru. Vnitřně se vlastně jedná o seznam dvojic — jméno parametru a jeho hodnota. Formát záznamu textových přepisů byl pro prohlížeč definován v XML a jeho současná podoba je následující:

```
<record channel="channel_id" start="start_in_seconds"
        end="end_in_seconds" bgcolor="background_color"
        timelabel="label" text="transcription"/>
```

nebo

```
<record channel="channel_id" start="start_in_seconds"
        end="end_in_seconds" bgcolor="background_color"
        timelabel="label">
    transcription
</record>
```

Význam jednotlivých parametrů je následující — `channel` udává identifikátor kanálu resp. mluvčího, `start` udává počáteční čas události, `end` koncový čas a `text` obsahuje samotný textový přepis. Poslední dva parametry jsou specifické spíše pro komponentu `Transcript`, tou se zabývá kapitola 3.2.1. Parametr `bgcolor` specifikuje barvu pozadí zobrazeného přepisu a `timelabel` je většinou přímo časový údaj v čitelném formátu, např. MM:SS — to už záleží, odkud daný přepis získáváme.

Pokud tedy máme požadovaný záznam můžeme jednotlivé parametry ukládat do objektu třídy `Record` jeho operátorem `[]` (indexace):

```
record["channel"] = "1";
record["start"] = "18.6";
...
```

Stejným způsobem je samozřejmě možné hodnoty jednotlivých parametrů číst. Tato třída neakceptuje pouze uvedené parametry, je samozřejmě možné vytvořit i jiné parametry — jméno parametru může být jakýkoli řetězec (kromě prázdného). V případě čtení parametru, který v objektu třídy není obsažen, je vrácen prázdný řetězec. Navíc pokud bychom nevěděli jaké všechny parametry vytvořený objekt obsahuje je možné je všechny získat opakovaným voláním metody `GetName`. Před jeho zahájením je však nutné nastavit vnitřní ukazatel na začátek seznamu parametrů metodou `FirstName`.

Třída `Record` implementuje navíc operátor `<` (menší) pro podporu řazení více záznamů. Ten je však definován pouze pro implicitní formát záznamu textového přepisu. Rozhodujícím parametrem při porovnávání záznamů je parametr `start`, tedy počáteční čas.

V případě, že jsou tyto parametry shodné porovnají se koncové časy (parametr `end`) a nakonec parametr `text`. Nebude-li definován ani jeden z těchto parametrů, nebude mít řazení záznamů žádný efekt.

3.1.2 Třída RecordList

Třída `RecordList`, jak už vypovídá název a je zřejmé i z obrázku 3.1, představuje seznam záznamů třídy `Record`. Jistě že není těžké vytvořit takový seznam ve vlastní režii, třída `RecordList` však navíc obsahuje metody pro vytváření a správu takových seznamů. Tak jako třída `Record` obsahuje podporu pro práci s XML záznamy, třída `RecordList` zahrnuje podporu pro práci s celými sekvencemi těchto záznamů a jejich import z XML souborů.

Po vytvoření instance této třídy je seznam implicitně prázdný — pro vkládání záznamů do seznamu můžeme použít metodu `Add`, která na vstupu očekává přímo objekt třídy `Record`. Užitečnější pro nás však budou zřejmě metody `ImportXML` a `ImportXMLFile` popř. i metoda `Import`, tu použijeme v případě, že máme již zpracované XML. Metoda `ImportXMLFile` zpracovává přímo XML soubor — ten může být umístěn jak lokálně, tak i vzdáleně např. na HTTP serveru. Tuto skutečnost zajišťuje použití právě tříd knihovny *wxWidgets*, konkrétně třídy *wxURL*, která akceptuje cestu k souboru ve standardním formátu URL, tzn. není žádný problém přistupovat k souboru přes HTTP či FTP protokol aj. Pokud se jedná o lokální soubor, indikujeme tuto skutečnost uvedením klíčového slova „file" na místě protokolu v URL k souboru, např. `file://data/soubor.xml`. Třída *wxURL* si však snadno poradí i se standardním zápisem cesty k souboru: `./data/soubor.xml`.

Metoda `ImportXML` na vstupu očekává řetězec, který obsahuje celý seznam záznamu v XML formátu, stejně jako jsou zapsané v uloženém XML souboru. Obě metody, `ImportXMLFile` i `ImportXML`, akceptují sekvenci záznamů zapsané v XML formátu, jak bylo specifikováno v předchozí kapitole, navíc však můžou být uzavřeny do uzlů:

```
<transcription>
    <data>
        <record channel=...
        <record ...
        ...
    </data>
</transcription>
```

Přistupovat k jednotlivým záznamům v seznamu můžeme podobným způsobem jako u přístupu k jednotlivým parametrům v třídě `Record`. Nejprve tedy nastavíme vnitřní ukazatel na začátek seznamu metodou `First` a poté už jen čteme sekvenčně jednotlivé záznamy metodou `Get`. Pokud už jsme prošli celý seznam a ukazatel je na konci, metoda vrací nulu.

V neposlední řadě je nutné se zmínit o metodě `Sort`, která seřadí celý seznam podle počátečního času (proto byl v předchozí kapitole explicitně uveden operátor `<` menší, kde byla také uvedena konkrétní kriteria řazení záznamů). Po provedení této metody jsou záznamy procházeny už ve správném pořadí.

3.2 Nové komponenty

Dalším z hlavních bodů mého zadání bylo navrhnout úpravy multimediálního přehrávače tak, aby bylo možné zobrazit videozáznam přednášek, společně s ním jeho textový přepis vygenerovaný rozpoznávačem řeči a umět v něm vyhledávat. A jelikož jádro prohlížeče bylo v zásadě hotové, další vývoj se ubíral směrem k návrhu a implementaci jednotlivých komponent, které by tyto funkce poskytovali. Jedná se hlavně o komponentu `Transcript` pro zobrazení textového přepisu, `MediaPlayer` pro zobrazení videozáznamu přednášky a další komponenty, které budou důležité hlavně do budoucna pro komunikaci s indexačními servery a dalšími prostředky důležitými pro funkci prohlížeče.

Následující podkapitoly se mimo obecného popisu funkce a vzhledu jednotlivých komponent budou detailně zabývat také jejich konfigurací, kterou samotné komponenty potřebují ke správné funkci. Ta je uložená v hlavním konfiguračním XML souboru společně s rozmístění komponent a jinými parametry. Každý popis bude doplněn praktickými příklady.

3.2.1 Komponenta Transcript

Explicitně byla komponenta `Transcript` vytvořena, jak už název napovídá, k zobrazení textového přepisu z rozpoznávače řeči. Později si však ukážeme, že způsob jakým její struktura navržena umožňuje širší užití a je v podstatě na uživateli, jak si její vzhled a typ přepisu nadefinuje. Návrh značně rozšířil možnosti nastavení této komponenty, proto její znalost vyžaduje více pozornosti, nicméně ve výsledku je vytváření konfigurace celkem intuitivní.

Textový přepis je implicitně akceptován ve formátu XML se strukturou definovanou v kapitole 3.1. Pro vstupní přepis, jeho umístění a další související náležitosti, platí pravidla uvedená v kapitole 3.1.2. Objekt třídy `RecordList` je přímo členem komponenty `Transcript` a využívá veškerých jeho výhod — XML přepis z rozpoznávače může být umístěn na webovém serveru stejně tak jako na lokálním disku, a další výhody popsané v předchozí kapitole.

Samotná komponenta `Transcript` textový přepis ve formátu XML převádí do HTML přepisu a až ten teprve zobrazuje. Právě proto má komponenta svůj základ v komponentě `HTMLBrowser`, která má funkci jednoduchého HTML prohlížeče. Mým úkolem bylo komponentu `HTMLBrowser` rozšířit o možnost interakce uživatele s jinými komponentami. Konkrétně bylo myšleno rozšíření funkce odkazů zapsaných v HTML kódu.

Vizuální část komponenty `HTMLBrowser` je odvozena od třídy *wxHtmlWindow* knihovny *wxWidgets*. Změnu funkce odkazů bylo zajištěno přetížením metody `OnLinkClicked` volané při přechodu na odkaz v aktuální načtené HTML stránce. Metoda provádí analýzu odkazu v parametru `href` HTML tagu `<A>` — pokud se jedná o standardní odkaz na lokální HTML dokument, komponenta tento dokument načte a zobrazí. V případě odkazu na vzdálený dokument přes HTTP či jiný známý protokol, je zavolán externí webový prohlížeč (implicitní prohlížeč nastavený v systému). Pro interakci uživatele a interní komunikaci prohlížeče byl stanoven speciální formát odkazu.

Pro snadné rozlišení tohoto typu odkazu bylo zavedení klíčového slova ,,imsg" na místě protokolu na začátku odkazu. Kompletní tvar odkazu je:

```
imsg:<message name="" ...>
```

Klíčové uvození `imsg:` slouží pouze pro identifikaci, že se jedná o speciální odkaz a zbytek odkazu je standardní zpráva v XML formátu pro interní komunikaci. Při aktivaci takového odkazu se neprovede načtení nového HTML dokumentu, ale zpráva obsažená v odkazu je odeslána prohlížeči a zařazena do fronty zpráv a posléze standardně zpracována. Uživatel tak v podstatě může pomocí komponenty `HTMLBrowser` ovládat celý prohlížeč načtením příslušné stránky, která by obsahovala odkazy se zprávami pro kompletní komunikaci s ostatními komponentami. Komponenta `HTMLBrowser` při spuštění načte HTML dokument zadaný v konfiguraci této komponenty jako parametr `url`, popř. je možné načíst HTML dokument přímo z interní zprávy `htmlbrowser.load` a samotný dokument umístit do zprávy jako parametr `text`. Uživatel tak zde má značnou volnost a velkou část interní komunikace může ovlivňovat příslušným nastavením popř. vytvořením zvláštního HTML dokumentu, který by tyto prvky zahrnoval.

Jelikož je komponenta `Transcript` odvozena od komponenty `HTMLBrowser`, přebírá toto rozšíření automaticky od ní a toho je s výhodou využito. Mimochodem tato situace je znázorněna na obrázku 3.1. Komponenta `Transcript` však při spuštění aplikace nenačítá čistý HTML dokument, ale na vstupu očekává XML soubor s textovým přepisem, který je posléze transformován na HTML přepis a zobrazen. Aby uživatel nebyl ochuzen o nastavení vzhledu tohoto přepisu byly do konfigurace této komponenty přidány parametry pro definování jednotlivých částí HTML přepisu.

Textový XML přepis je v podstatě sekvence jednotlivých záznamů, proto je v HTML přepisu implicitně reprezentován jako tabulka. Každý záznam tak představuje jeden řádek vygenerované tabulky. Pro uživatele je nastavení vzhledu rozděleno do třech částí. První část definuje hlavičku HTML dokumentu, druhá definuje HTML kód pro jednotlivé záznamy a poslední část definuje patu dokumentu. HTML přepis se pak sestavuje tak, že se na začátek uvede hlavička, k ní se připojí kód těla jednotlivých záznamů opakovaně tolikrát, kolik je celkem záznamů a nakonec se připojí pata dokumentu. Nejlepší bude si celý zápis ukázat na příkladu, který je současně implicitním nastavením vzhledu, pokud jej uživatel nedefinuje přímo v konfiguračním souboru.

Hlavička je definována takto:

```
<HTML>
    <HEAD>
        <meta http-equiv="Content-Type" content="text/html
            charset=CP1250">
    </HEAD>
    <BODY>
        <TABLE BORDER COLS=1 WIDTH="100%">
```

kód těla pro jednotlivé záznamy:

```
            <A name="$start"></A>
            <TR bgcolor="$bgcolor">
                <TD align="right" valign="top">
                    <b>[$timelabel]</b>
                </TD>
                <TD align="left">
                    <a href="imsg:<message name=sync.transcript
                                  source=manual start=$start/>">
                        $text
                    </a>
                </TD>
            </TR>
```

a pata dokumentu:

```
        </TABLE>
    </BODY>
</HTML>
```

Jak je vidět, v tomto sledu všechny části tvoří logicky zdrojový kód HTML přepisu. Určitě jste si však všimli, zvláštnosti v těle záznamu. Tělo záznamu se totiž ve výstupním HTML přepisu opakuje pro každý záznam jednou a kdyby byly všechny řádky tabulky stejné, neměl by takový zápis žádný smysl. V části generování těla přepisu je proto zabudovaná podpora proměnných. Ta je umístěna v modulu `common` — konkrétně se jedná o funkci `MBSubstVars`.

Proměnná se v HTML kódu uvozuje znakem `$` (dolar) a bylo myšleno i na sporné situace, kdy by se proměnná pojila s jiným textem. V takovém případě se jméno proměnné uzavře do složených závorek, např. takto: `${var}`.

Co se týče substituce proměnných, ta přímo souvisí s textovým přepisem v XML formátu, tj. např. proměnná v HTML přepisu s názvem `start` bude nahrazena obsahem parametru s názvem `start` v uzlu `record` v XML přepisu. Ukážeme-li si konkrétní příklad jednoho záznamu:

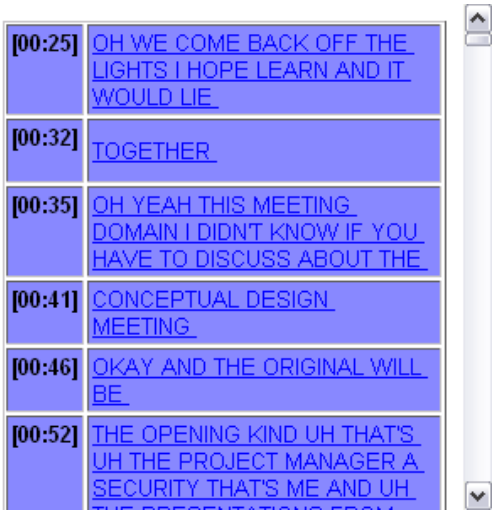
```
<record channel="0" start="25.789" end="30.816" bgcolor="#8888ff"
        timelabel="00:25">
    OH WE COME BACK OFF THE LIGHTS I HOPE LEARN AND IT WOULD LIE
</record>
```

tělo vygenerovaného HTML kódu, za předpokladu, že použijeme implicitní nastavení HTML přepisu, by vypadalo takto:

```
<A name="25.789"></A>
<TR bgcolor="#8888ff">
    <TD align="right" valign="top">
        <b>[00:25]</b>
    </TD>
    <TD align="left">
        <a href="imsg:<message name=sync.transcript
                        source=manual start=25.789/>">
            OH WE COME BACK OFF THE LIGHTS I HOPE
            LEARN AND IT WOULD LIE
        </a>
    </TD>
</TR>
```

Navíc jak je vidět, obsah uzlu `record` je brán jako parametr `text`. Pokud by byl uveden samotný parametr a současně by uzel obsahoval nějaký text přednost by měl uvedený text a pravě ten by byl substituován za proměnnou `text`. Příklad textového přepisu s implicitním nastavením vzhledu komponenty `Transcript`, kde je zahrnut i záznam uvedeny v předchozím příkladě, je vidět na obrázku 3.2.

Definici obsahu jednotlivých částí má uživatel plně pod kontrolou a pokud mu vyhovuje úplně jiný zápis, bez problému jej může použít a výstup sledovat okamžitě po novém spuštění aplikace. Tabulka zde byla použita pro názornost a výsledek přepisu, který je vidět na obrázku, nezastiňuje žádný detail.



Obrázek 3.2: Ukázka komponenty Transcript

3.2.2 Komponenta Execute

V následujících dvou podkapitolách si povíme něco o nevizuálních komponentách, které sice nejsou v testovací verzi aplikace použity, ale jejich význam značně stoupá s budoucím vývojem. Mým úkolem bylo vytvořit komponentu, jež by umožnila prohlížeči komunikaci s externími aplikacemi. Význam a schopnosti této komponenty budou postupně ozřejmeny ve výčtu jejích vlastností.

Komponenta ve zkratce při vzniku požadavku spustí externí aplikaci popř. příkaz a jeho výstup (pouze standardní výstup, ne chybový) předá prohlížeči. Volaný příkaz je nutné napřed specifikovat v nastavení komponenty v hlavním konfiguračním souboru parametrem `command`. Možnosti jsou s části závislé na operačním systému, obecně však příkaz může obsahovat cestu ke spouštěné aplikaci následovanou parametry oddělenými mezerami. Pod unixovým operačním systémem navíc přibývá možnost spouštět skripty popř. jiné příkazy podobně jako v příkazovém řádku. Pod operačním systémem windows je tato možnost dostupná pouze pod systémem cygwin, s kterým jsou částečně problémy spojené s používáním knihovny *wxWidgets*. Komponenta `Execute` do zápisu příkazu přináší navíc možnost použití

proměnných pro větší volnost v používání této komponenty. Formát použití proměnných i s příklady byl uveden v předchozí kapitole, proto jej nebudu znovu rozebírat.

Požadavek u komponenty vzniká příjmem specifické interní zprávy. Komponenta tak během svého života pouze čeká na příchod zprávy, která ji aktivuje, provede požadovanou činnost a opět přejde do nečinného stavu. Formát aktivační zprávy je:

```
<message name="exec" param1="hodnota1" param2=... />
```

Po příjmu takovéto zprávy se nejprve provede substituce proměnných v příkazu, které odpovídají jednotlivým parametrům zprávy (v uvedeném příkladu `param1`, `param2` atd.), a posléze se daný příkaz provede. Respektive komponenta pouze příkaz spustí a přejde do nečinného stavu. Řízení předá třídě `Process`, která zajistí zbytek operace. Tento přístup byl zaveden se zřetelem na proměnlivou dobu provádění příkazu. Po ukončení procesu je vyvolána metoda `OnTerminate` třídy `Process`. Ta načte kompletní standardní výstup vygenerovaný za běhu daného příkazu a odešle jej zprávou `output` v parametru `text`. Jméno zprávy prozatím není standardizováno a z testovací důvodu bylo takto nastaveno — v budoucnu jej bude nutné určitě změnit, aby bylo jasně rozlišeno, o jakou zprávu se jedná.

3.2.3 Komponenta UrlGet

Dalším stupněm vnější komunikace měla být komunikace prostřednictvím sítě či internetu. Jako stěžejní byl vybrán protokol HTTP. Z původního záměru však vznikla komponenta `UrlGet`, která jak už název napovídá, načte dokument zadaný pomocí URL. Pro realizaci byla použita třída *wxURL* z knihovny *wxWidgets* — tato třída umožňuje (což je dáno vlastnostmi URL) adresovat dokument právě přes HTTP protokol, ale i jiné jako např. FTP a dokonce i lokální soubory. O adresování dokumentu prostřednictvím URL byla řeč už v kapitole 3.1.2 ve spojení s importem textových přepisů.

Funkce komponenty je v celku jednoduchá. Podobně jako komponenta `Execute` čeká na příchod zprávy po jejímž příchodu se adresovaný dokument načte (resp. stáhne, pokud se jedná o vzdálený dokument) a celý je vložen do zprávy a odeslán prohlížeči ke zpracování[1]. Požadované URL je možné zadat před samotným spuštěním aplikace do konfigurace komponenty jako parametr `url`, popř. je možné jej zadat přímo jako parametr příchozí zprávy — taktéž `url`. V případě, že jsou zadány oba, použije se URL předané ve zprávě, nicméně URL zadané při konfiguraci komponenty zůstane zachováno a při další aktivaci komponenty je možné jej použít. Pokud by se pak často přistupovalo ke stejnému dokumentu, není nutné jej uvádět v každé zprávě a je možné jej za běhu změnit prostřednictvím zprávy `seturl`, požadované URL se pak uvede opět v parametru `url`.

3.2.4 Komponenta MediaPlayer

Jednou z nejvýznamnějších komponent je komponenta `MediaPlayer` pro přehrávání video záznamů. Na této komponentě jsem sice nepracoval já, ale zde si řekneme něco o synchronizačních zprávách v prohlížeči. Jejich formát a použití je totiž už stanoven.

[1] jméno této zprávy opět není zatím standardizováno a může se měnit — v tuto chvíli je nastaveno na `htmlbrowser.load`

Přehrávání videa v této komponentě zajišťuje výchozí modul pro přehrávání video záznamů v daném operačním systému. V operačním systému windows to bude nejspíš Windows media player, pod Unixem jiný apod. Tuto funkci a veškeré ovládání přehrávání video záznamu zajišťuje pro splnění přenositelnosti opět třída knihovny *wxWidgets* a to *wxMediaCtrl*. Pro správnou a souvislou synchronizaci přehrávaného video záznamu komponenta spolupracuje s modulem `Timer`. Ten zajišťuje pravidelnou synchronizaci všech souvisejících komponent. V prohlížeči je implicitně synchronizační interval nastaven na 1 sekundu. Veškerá synchronizace je prováděna prostřednictvím vnitřní komunikace pomocí zpráv. Synchronizační zpráva má tvar:

```
<message name="sync" source="timer" start="999.99">
```

nebo pro synchronizaci na určitou sekci:

```
<message name="sync" source="timer" start="999.99" end="999.99">
```

Synchronizace na určitou sekci by měla umožňovat přehrát pouze určitý úsek video záznamu a poté přehrávání pozastavit. Komponenta `MediaPlayer` zatím podporuje pouze první tvar synchronizační zprávy. Parametr `source` udává typ zdroje, jež zprávu vygeneroval. První možností je právě použitý typ `timer`, ten generuje časovač a slouží pro průběžnou synchronizaci ostatních komponent s přehráváním video záznamu. Na něj však nereaguje pokaždé komponenta `MediaPlayer`, pokud bychom totiž na pracovní plochu umístili současně více přehrávačů, synchronizace by značně zatěžovala procesor a přehrávání by nebylo plynulé. Druhým typem synchronizační zprávy je `manual`, komponenta tuto zprávu generuje např. při pozastavení nebo spuštění přehrávání záznamu a ostatní komponenty ji mohou využívat k synchronizaci video záznamu na požadovaný čas.

Synchronizační zprávy zpracovává ještě například komponenta `Transcript`, která synchronizuje textový přepis současně s přehrávaným záznamem. Toho si můžete všimnout v testovací verzi aplikace na přiloženém médiu CD.

3.3 Další úpravy

Týmový přístup k vývoji vyžaduje vyvážené rozdělení práce mezi jednotlivé členy týmu, proto ne vždy byl vývoj jednotvárný a občas bylo nutné vyřešit jiné problémy související s postupným vývojem, např. při vývoji komponenty `Transcript` se objevil problém při odesílání zprávy a nakonec jsem objevil malou chybičku v modulu `Message` a opravil s tím související slabé místo v modulu `SXMLParser`.

Kromě vývoje komponent jsem dostal za úkol vytvořit makefile pro překlad projektu pod systémem MinGW. Do té doby měl každý člen týmu mezi zdrojovými soubory vytvořený vlastní makefile, jelikož překlad je značně individuální a závisí na použité verzi knihovny *wxWidgets*, na jejím umístění a na mnoha dalších aspektech. Největší problém je překlad na rozdílných operačních systémech či s použitím jiných překladačů. Rozdíl může

být v umístění různých knihoven potřebných k překladu, popř. i ve způsobu zápisu cest apod.

Systém MinGW podporuje zápis cest v unixovém formátu, makefile pro MinGW se tak hodně podobá unixové verzi a několika málo úpravami by mohl být použit i pod unixovým operačním systémem. Stejně tak makefile funguje pod systémem Cygwin. Konkrétní popis testované konfigurace jak systému Cygwin tak MinGW společně s popisem instalace knihovny *wxWidgets* je uveden v komentáři přímo v příslušném makefile. Vytvořený makefile má oproti původní verzi definovány veškeré závislosti mezi jednotlivými moduly a hlavičkovými soubory. Absence těchto závislostí způsobovala nekonzistence mezi jednotlivými moduly přeloženými v různý čas a mohlo tak dojít k neidentifikovatelné chybě za běhu aplikace.

Makefile pro MinGW je logicky v souboru `makefile.mingw`. Kromě makefile pro MinGW jsem vytvořil zvlášť makefile (`makefile.wxdev`) pro uživatele používající vývojový nástroj wxDev-C++. Ten sice zahrnuje také systém MinGW, ale jinak specifikuje cesty ke knihovnám a vyžaduje jiné parametry překladu.

Kapitola 4

Budoucí vývoj a zapojení do rozsáhlého projektu

Multimediální prohlížeč představuje rozsáhlejší týmový projekt a už v předchozích letech byl součástí několika bakalářských či diplomových prací. Projekt není ještě zdaleka dokončený, ale nyní je ve fázi, kdy je možné prezentovat některé z jeho funkcí, které by měli být v budoucnu běžně dostupné i řadovým uživatelům.

4.1 Další užitečné komponenty

Nejbližší fáze vývoje bude směřovat k vytváření dalších potřebných komponent. Inspirace pro nové komponenty je stále možné brát z prohlížeče JFerret. [5] Aktuálně např. komponenta pro zobrazení časové osy průběhu přednášky s indikací právě aktivního mluvčího nebo další komponenta pro zobrazení a procházení prezentací přednášek v různých formátech — v tuto chvíli existuje komponenta schopná zobrazovat jednotlivé snímky prezentace uložené jako obrázek.

Kromě vývoje nových komponent je stále na místě zdokonalování stávajících komponent, např. synchronizace komponenty `MediaPlayer` resp. modulu `Timer` na časový interval a mnoho dalších úprav.

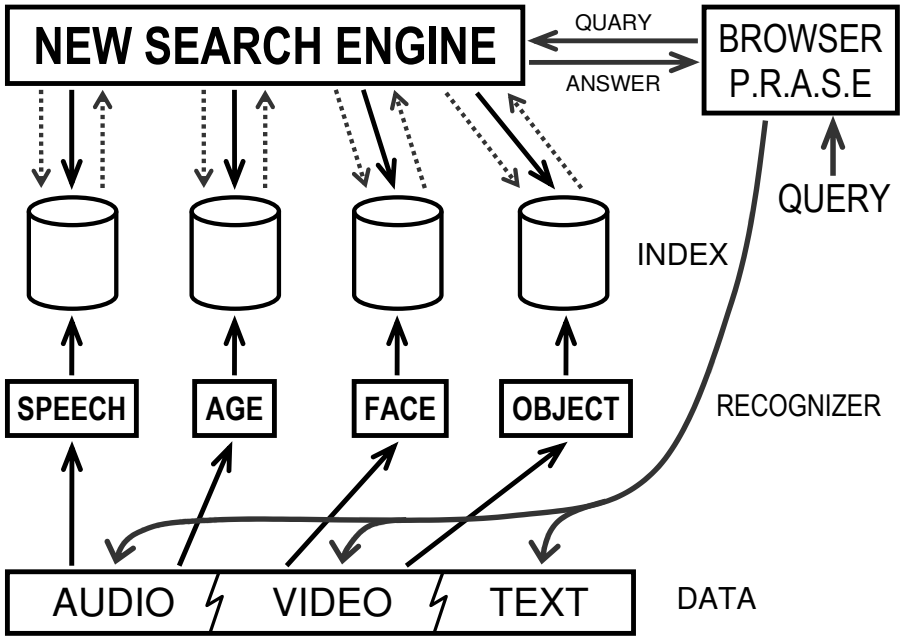
4.2 Pokročilejší úpravy

Dále je nutné standardizovat veškerou interní komunikaci, která není v tuto chvíli ještě ustálena, tj. definovat názvy a formát jednotlivých zpráv, popř. navrhnout modul pro konverzi vnitřní komunikace — což by představovalo zásah do jádra prohlížeče.

Podle původního návrhu prohlížeče měla být každá komponenta umístěna ve zvláštním souboru, aby bylo možné komponenty vytvářet nezávisle na jádře prohlížeče. Takový přístup je však silně závislý na použitém operačním systému a vyžaduje speciální řešení pro každý z nich. V operačním systému Windows by každá komponenta představovala např. jednu knihovnu DLL, v linuxu knihovnu SO apod. Aktuálně jsou komponenty kompilovány přímo k jádru prohlížeče a jiný přístup by představoval velký zásah do jádra prohlížeče.

4.3 Projekt vyhledávacího systému

V současné době se na Ústavu počítačové grafiky a multimédií rozjíždí rozsáhlý projekt vyhledávacího systému jehož součástí by měl být právě vyvíjený multimediální prohlížeč. Zde budou mít uplatnění právě součásti prohlížeče zajišťující externí komunikaci. Jak je vidět na obrázku 4.1, prohlížeč plní úlohu vysoce konfigurovatelného uživatelského rozhraní k celému systému.



Obrázek 4.1: Připravovaný vyhledávací systém [8]

Základem celého systému by byly různé rozpoznávače, které by pracovali nad audio, video či jinými daty a svoje výsledky by ukládaly ve formě indexů na specifická úložiště. Mezi rozpoznávače by se řadil nejen zmiňovaný řečový rozpoznávač, ale i řada dalších jako např. rozpoznávač věku či jiných parametrů mluvčího nebo z oblasti grafiky rozpoznávače obličejů a jiných objektů.

Pro vyhledávání by pak klíčovým prvkem byly indexační servery, jež by spravovali výsledky rozpoznávačů a umožňovali rychlý přístup a vyhledávání mezi nimi. Nad těmito servery by pak měl pracovat speciální vyhledávací systém, který by přímo komunikoval s multimediálním prohlížečem a zodpovídal vyhledávací dotazy. Vyhledávací systém by měl k dispozici výstupy všech různých rozpoznávačů a mohl by tak provádět složitější dotazy, ve kterých by bylo zadáno současně více kriterií a uživatel by tak mohl přesněji specifikovat, co přesně hledá.

Kapitola 5

Závěr

V první fázi mojí práce jsem měl za úkol seznámit se s použitou knihovnou *wxWidgets* a se samotným projektem multimediálního přehrávače, který už v té době byl ve velmi rozpracované fázi. Knihovna *wxWidgets* je sama o sobě velmi rozsáhlý projekt a její studie zabrala značnou dobu — nemalou překážkou byla její konfigurace a překlad. V závislosti na použitém systému je nutné knihovnu přeložit přímo ze zdrojových souborů. Ne však všechny její funkce byly po překladu implicitně dostupné a z překladem prohlížeče jsem tak měl ze začátku problémy. Navíc pro překlad projektu jsem jako výchozí zvolil systém Cygwin, který — jak se později ukázalo — způsobuje v kombinaci s knihovnou *wxWidgets* některé nekonzistence a dále se tak vývoj zdržel. V průběhu této fáze jsem se poučil o dalších problémech, kterým je nutné se vyhnout při překladu a pro snadnější překlad projektu jsem v hlavičkách obou makefile (jak pro systém MinGW, tak pro vývojové prostředí wxDev-C++) uvedl podrobný postup instalace této knihovny.

Poté jsem se seznámil se základní strukturou jádra prohlížeče a jeho komponentovým systémem a s detaily jednotlivých části jsem se podrobně seznamoval v podstatě v celém průběhu vývoje. S dokončením implementace pomocných tříd (`Record` a `RecordList`) a zahájením vývoje první komponenty `Transcript` jsem měl dobrý přehled o komponentovém systému prohlížeče a jeho vnitřní komunikaci. Součástí byla také znalost používaného XML parseru, který bylo nutné znát od prvních zásahu do prohlížeče.

V současné fázi vývoje prohlížeče a výčtem komponent, jež má k dispozici, je možné testovat jeho základní funkce a s komponentami pro externí komunikaci má schopnost téměř okamžitě se připojit k projektu vyhledávacího systému. Vývojový tým vyhledávacího systému bude tak mít možnost bez potíží testovat svůj systém už v průběhu vývoje. Projekt multimediálního prohlížeče je samozřejmě stále ve vývoji a v současném stavu zatím není možné jej vypustit mezi řadové uživatele — navíc s absencí vyhledávacího systému by pro takového uživatele měl velmi malý význam.

Použité zkratky

LID	Language IDentification - identifikace jazyka
LVCSR	Large Vocabulary Continuous Speech Recognition - rozpoznávání plynulé řeči s velkým slovníkem
KWS	KeyWord Spotting - detekce klíčových slov
SpkID	Speaker IDentification - rozpoznávání mluvčího
SpkVer	Speaker Verification - ověřování mluvčího
GCC	GNU Compiler Collection
GUI	Graphical User Interface - grafické uživatelské rozhraní
GNU	GNU's Not Unix - GNU Není Unix
MinGW	Minimalist GNU for Windows [3]
DLL	Dynamic-Link Library - dynamicky linkovaná knihovna
POSIX	Portable Operating System Interface - kolekce standardů specifikovaná institutem IEEE pro definice aplikačního rozhraní kompatibilního mezi jednotlivými unixovými operačními systémy
URL	Uniform Resource Locator - přesná specifikace umístění zdroje informace
IDIAP	Institut Dalle Molle d'Intelligence Artificielle Perceptive (IDIAP Research Institute [1])

Literatura

[1] IDIAP Research Institute. [online].
URL http://www.idiap.ch/

[2] PRASE Multi-media presentation tool. [online], poslední modifikace: 9.4.2007.
URL http://merlin.fit.vutbr.cz/wiki/index.php?title=PRASE_Multi-media_presentation_tool

[3] MinGW - Minimalist GNU for Windows. [online], 2004, poslední aktualizace: 10.4.2007.
URL http://www.mingw.org/

[4] Hoff, T.: C++ Coding Standard. [online], 1995-2006, poslední modifikace: 9.1.2007.
URL http://www.possibility.com/Cpp/CppCodingStandard.html

[5] IDIAP Research Institute: JFerret. [online], 2007, poslední aktualizace: 2.2.2007.
URL http://www.idiap.ch/mmm/tools/jferret

[6] IDIAP Research Institute: MultiModal Media File Server. [online], 2007, poslední aktualizace: 2.2.2007.
URL http://www.idiap.ch/mmm/

[7] Smart, J.; Hock, K.; Csomor, S.: *Cross-Platform GUI Programming with wxWidgets*. Prentice Hall, 2006, ISBN 0-13-147381-6.

[8] Szöke, I.; Fapšo, M.: Killer search in absolutely nonsenique meetings. Prezentace k meetingu 23.3.2007 - 11:00.

Příloha A

Testovací konfigurační soubor

```
<layout name="main">
  <window name="window1" title="Multi-media Browser"
      left=0% top=0% width=75% height=95%>
    <mediaplayer name="player1" left=0% top=0% width=60% height=60%
        url="file://data/iss_prednaska_7.avi" autorun="true" amsg=".*"
        waitfor="1" controls="true" backend="" syncint="1.0"/>
    <transcript name="transcript1" left=0% top=60% width=100% height=40%
        url="./data/iss_prednaska_7.mlf.ok.cp1250.xml" amsg=".*:sync.*">
      <args>
        <head>
          "<HTML>
          <HEAD>
            <meta http-equiv=\"Content-Type\"
                  content=\"text/html; charset=CP1250\">
          </HEAD>
          <BODY>
            <TABLE border=\"0\" cellpadding=\"1\"
                   cellspacing=\"5\" width=\"100%\">"
        </head>
        <body>
          "<A name=\"$start\"></A>
          <TR bgcolor=\"$bgcolor\">
            <TD align=\"right\" valign=\"top\">
              <B>[$timelabel]</B>
            </TD>
            <TD align=\"left\">
              <A href=\"imsg:<message name=sync.transcript
                                      source=manual start=$start/>\">
                $text
              </A>
            </TD>
          </TR>
          <TR>
            <TD> </TD>
            <TD bgcolor=\"#C0C0C0\"></TD>
          </TR>"
        </body>
```

```
        <foot>
          "</TABLE></BODY></HTML>"
        </foot>
      </args>
    </transcript>
    <textbox name="textbox1" text="TRANSFORMACE" left=60% top=0%
        width=30% height=5% onconfirm="<message name=\"exec\"/>"
        amsg=".*:sendText"/>
    <button name="button1" text="Search" left=90% top=0% width=10%
        height=5% onclick="<message name=\"sendText\"/>"/>
    <transcript name="transcript2" left=60% top=5% width=40%
        height=55% amsg=".*:sync.*;.*:output">
      <args>
        <head>
          "<HTML>
          <HEAD>
            <meta http-equiv=\"Content-Type\"
                  content=\"text/html; charset=CP1250\">
          </HEAD>
          <BODY bgcolor=\"#E0E0E0\">"
        </head>
        <body>
          "<A name=\"$start\"></A><BR>
          <B>[$timelabel]</B>
          <A href=\"imsg:<message name=sync.transcript
                                  source=manual start=$start/>\">
            $text
          </A>
          <HR>"
        </body>
        <foot>
          "</BODY></HTML>"
        </foot>
      </args>
    </transcript>
    <execute name="command1"
             command="sh ./data/search/scripts/8_search.sh $text"
             amsg=".*"/>
  </window>
</layout>
```

Příloha B

Ukázková aplikace