TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: Informační Technologie

Návrh a realizace vizualizačního softwaru pro výuku zpracování řeči

Design and implementation of visualization software for teaching speech processing

Diplomová práce

Autor: Bc. Ladislav Šeps

Vedoucí práce: Ing. Miroslav Holada, Ph. D.

Konzultant: Ing. Josef Chaloupka, Ph. D.

V Liberci 19. 5. 2011

Prohlášení

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

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

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

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

Datum: 19. 5. 2011

Podpis

Abstrakt

Cílem této práce byl návrh a realizace softwarové platformy pro výuku algoritmů z oblasti zpracování signálů s možností vizualizací a posléze v tomto softwaru implementovat ukázkové moduly demonstrující možnosti programu a moduly pro základní algoritmy z oboru rozpoznávání řeči. Požadavky kladené na tento software byly moderní uživatelské rozhraní, snadná manipulace s daty i aplikací a hlavně bezproblémová rozšiřitelnost o nové výukové moduly. Tato zpráva podává stručný přehled o základních hlasových technologiích, věnuje se návrhu a samotné implementaci výukové aplikace včetně modulů prezentujících rozhraní a možnosti programu tedy: práci s daty, spolupráci s API operačního systému, zobrazení zvukových signálů a prezentaci základních algoritmů, rozpoznání izolovaných slov včetně animace ve 2D i 3D zobrazení.

Abstract

The goal of this work was the design and implementation of software platform for teaching signal processing algorithms with the possibility of visualizations and implementation of basic modules for demonstrating program features and for basic speech processing algorithms. Requirements for this software were a modern user interface, simple manipulation with data and application, and last but not least easy creation of new program modules. This report provides a brief overview of speech processing basics, describes the design and implementation of educational software, including modules that shows the interface and capabilities of implemented application such as: working with data, cooperating with operating system, display of sound signals and display of algorithms for recognition of isolated words in 2D and 3D view, including animations.

Klíčová slova

Visper, EasyBlocks, výukový software, zpracování signálu, zpracování řeči

Keywords

Visper, EasyBlocks, Educational software, signal processing, speech processing

Úvod

Na Technické univerzitě v Liberci používá již od roku 1997 výukový program Visper. Byl vytvořen za účelem seznámení předvádění s algoritmů pro rozpoznávání řeči. Jeho předností jsou vizualizace rozpoznávacích algoritmů pomocí animací, které usnadňují jejich pochopení. Aktuálně používaná verze Visper II je založena na uživatelském rozhraní těžícím z technologií Component Object Model a ActiveX uvedených firmou Microsoft v roce 1996. Program bez problému funguje na operačních systémech Windows až do verze Windows XP. V systému Windows Vista došlo k revizi COM a ActiveX, díky které Visper II na tomto a novějším systému Windows 7 nefunguje. Tato situace vedla k úvahám o novém výukovém programu, který by časem mohl Visper II nahradit. Visper je v současném stavu díky použití několika programovacích jazyků (C++, assembler, Visual Basic) a dlouhé době vývoje problematické dále rozšiřovat. Navíc jeho uživatelské prostředí je poněkud nezvyklé a je nutné si něj zvykat. Díky tomu vyvstaly první 3 požadavky na případnou novou aplikaci, tedy snadná rozšiřitelnost, moderní lehce osvojitelné uživatelské rozhraní a pokud možno použití technologií, které se budou používat i v dalších letech.

Také se ukázalo, že specializovaných výukových programů existuje velmi málo a to napříč mnoha technickými obory a bylo by vhodné kdyby případná nová aplikace byla použitelná v širší oblasti, než je samotné rozpoznávání řeči tedy minimálně v oblasti zpracování signálu případně i dalších oborech.

Všechny tyto vlastnosti by teoreticky mohla splňovat nějaká softwarová platforma bez na konkrétní obor specializovaných funkcí, ale se snadnou rozšiřitelností, zaměřením na organizaci obecných vstupních a výstupních dat a s co nejuniverzálnější vnitřní datovou strukturou.

Cílem této práce je právě takovouto platformu navrhnout a realizovat, i když s omezením funkčních modulů na obor zpracování řeči.

Obsah

Úvod....6
1 Hlasové technologie....10
1.1 Přehled nejběžnějších hlasových technologií....10
1.1.1 Převod textu na mluvené slovo....10
1.1.2 Detekce jazyka....10
1.1.3 Detekce mluvčího....11
1.1.4 Převod mluveného slova na text (rozpoznání řeči)....11
1.2 Rozpoznání izolovaných slov....11
1.2.1 Parametrizace....12
1.2.2 Příznaky....12
1.2.3 Rozpoznání izolovaných pomocí srovnání nahrávek....12
1.2.4 Rozpoznání izolovaných slov pomocí statistického modelu....15
1.3 Rozpoznání spojité řeči....16
2 Programy používané při výuce zpracování signálu a zpracování řeči....17
2.1 Visper....17
2.2 Matlab....18
2.3 Programovací jazyky obecně....19
3 Návrh a realizace výukové aplikace EasyBlocks....20
3.1 Idea blokového schematu....20
3.2 Uživatelské rozhraní programu....20
3.3 Balíčkový systém ukládání....22
3.4 Platforma Microsoft .NET....23
3.5 Uživatelské rozhraní detailně....24
3.5.1 Hlavní okno programu....24
3.5.2 Seznam bloků – Panel Blocks....24
3.5.3 Výběr pokusu....24
3.5.4 Designer – Panel Structure....25
3.5.5 Nastavení bloků – panel Properties....26
3.5.6 Přepínání vizualizací - panel Visualizations....27
3.5.7 Detaily pokusu – Panel Experiment Details....27
3.5.8 Správa souborů – Panel Files and Streams....28

3.5.9 Připínání konstant....28
3.6 Funkční bloky....29
3.6.1 Běh na vlastním vlákně....30
3.6.2 Spouštění na základě požadavku na data....31
3.6.3 Ukončení a reakce na ukončení datových kanálů....31
3.6.4 Zveřejnění nastavení funkčního bloku....31
3.6.5 Ukládání dat funkčního bloku....32
3.7 Datové kanály....33
3.7.1 Přenos dat datovým kanálem....34
3.7.2 Netypový přenos datových bufferů....34
3.7.3 Přenos dat jako typované objekty....34
3.7.4 Přenos dat jako typované buffery....34
3.7.5 Přenos datovým kanálem z programového hlediska....35
3.8 Běhové prostředí....36
3.8.1 Potlačení některých bloků v nabídce....36
3.9 Externí knihovny využité při programování EasyBlocks....37
3.9.1 DockPanel Suite....37
3.9.2 OpenTK....37
3.9.3 MathNet Iridium....37
4 Přehled implementovaných funkčních bloků a pokusů....38
4.1 Přehled implementovaných funkčních bloků....38
4.2 Přehled ukázkových pokusů....40
4.2.1 Porovnání 2 slov ze souborů v balíčku....40
4.2.2 Přehrávání souboru....40
4.2.3 Zobrazení a uložení souboru pod jiným jménem....41
4.2.4 Nahrávání z mikrofonu do souboru....41
4.2.5 Porovnání slova v souboru s daty z mikrofonu....42
5 Závěr....43

Seznam obrázků

Obrázek 1: Euklidovská vzdálenost....13
Obrázek 2: LTW přiřazení indexu....13
Obrázek 3: LTW algoritmus....13
Obrázek 4: LTW přiřazení indexů....13

Obrázek 5: DTW přiřazení indexů....14
Obrázek 6: DTW algoritmus....15
Obrázek 7: HMM levo-pravý pěti-stavový model....15
Obrázek 8: Visper II rozhraní....17
Obrázek 9: Visper II DTW porovnávání....18
Obrázek 10: Matlab Uživatelské rozhraní....19
Obrázek 11: Blokové schema tak jak je zobrazované v programu....20
Obrázek 12: Ukázka dokování....21
Obrázek 13: Struktura balíčku: kořenový adresář....22
Obrázek 14: Struktura balíčku: pokus....22
Obrázek 15: Hlavní okno programu....24
Obrázek 16: Výběr z několika pokusů v rámci jednoho balíčku....25
Obrázek 17: Designer reakce na propojení....26
Obrázek 18: Panel properties....26
Obrázek 19: Panel Visualizations....27
Obrázek 20: Panel Experiment Details....27
Obrázek 21: Panel Files and Streams....28
Obrázek 22: Připínání konstant....28
Obrázek 23: Rozhraní pro vytváření funkčních bloků....29
Obrázek 24: Model činnosti funkčního bloku....30
Obrázek 25: Řídící výjimky....31
Obrázek 26: Model datového kanálu....33
Obrázek 27: Programové rozhraní datových kanálů....35
Obrázek 28: Porovnání 2 slov ze souborů v balíčku....40
Obrázek 29: Přehrávání souboru....40
Obrázek 30: Zobrazení a uložení souboru pod jiným jménem....41
Obrázek 31: Nahrávání z mikrofonu do souboru....41
Obrázek 32: Porovnání slova v souboru s daty z mikrofonu....42

1 Hlasové technologie

Hlasové technologie je souhrnné označení pro počítačové zpracování zvukové stopy obsahující lidskou řeč. Cílem hlasových technologií je extrakce určité informace, případně její vytvoření, ve zpracovávané zvukové stopě. Běžnou vlastností systémů hlasových technologií je závislost na konkrétním jazyku. To zabraňuje vytvoření univerzálních systémů. Tedy například v Českých podmínkách je problematické až nemožné využívat systémy, které pracují v rozšířenějších jazycích a je nutné spoléhat na systémy zde vyvinuté. Další detaily je možné zjistit například v [1] [2]

1.1 Přehled nejběžnějších hlasových technologií

1.1.1 Převod textu na mluvené slovo

Patrně nejznámější a nejdéle prakticky používanou hlasovou technologií je převod textu na mluvené slovo, někdy se také označuje TTS (z anglického Text To Speech). Implementace systému pro převod textu na mluvené slovo je celkem nenáročná na výkon, jelikož se v podstatě jedná o sestavování krátkých předpřipravených zvukových nahrávek do většího celku. Kvalita TTS systémů se pohybuje ve velkém rozmezí od takřka nesrozumitelného zvuku, přes zřetelně počítačový, ale srozumitelný hlas až k hlasu, který je takřka nerozeznatelný od lidského. V praxi se často používá jako pomocný systém pro těžce zrakově postižené osoby například ve webových prohlížečích. Nebo pro předčítání e-knih v kapesních prohlížečích.

1.1.2 Detekce jazyka

Detekce jazyka, jak z názvu vyplývá, slouží ke zjištění jazyka, kterým osoba v nahrávce promlouvá. Nejčastější použití je v součinnosti s dalšími rozpoznávacími systémy, kdy je díky detekci jazyka možné vybrat a přiřadit odpovídající systém pro rozpoznání řeči. Jako další vhodné využití se jeví použití v telefonních systémech záchranných služeb nebo mezinárodních společností pro přiřazení odpovídajícího pracovníka.

1.1.3 Detekce mluvčího

Systém detekce mluvčího dokáže zjistit, jestli v dané nahrávce promlouvá mluvčí z nějaké předem připravené množiny mluvčích a případně ho konkrétně identifikuje. Často se využívá v součinnosti s dalšími hlasovými technologiemi, například pro vybrání rozpoznávače mluveného slova přizpůsobeného konkrétní osobě, což zvyšuje přesnost rozpoznání případně minimalizuje specifické problémy, jako jsou vady řeči. Další běžné použití je v zabezpečení přístupu ať už do fyzického objektu nebo přístupu do operačního systému počítače, v tomto případě se často používá jako doplněk k dalšímu identifikačnímu znaku jako otisky prstů nebo přístupové heslo.

1.1.4 Převod mluveného slova na text (rozpoznání řeči)

Systémy pro převod mluveného slova na text převádí mluvenou informaci ze zvukové stopy do textové podoby. Výhodné je, že informace v textové podobě zabírá mnohem méně úložného prostoru než zvuková nahrávka. Navíc je v ní možné snadno a rychle vyhledávat konkrétní slova, bez nutnosti si nahrávku poslechnout. Jedním z možných využití je tedy archivace některých zvukových nahrávek, případně přidání textu k nahrávce pro snadné vyhledání konkrétní části pro přehrání. Dále je ho možné využít v systémech pro diktování textu, nebo se používá jako alternativní ovládací zařízení u počítačových systémů v případech, že využití klávesnice a myši není vhodné nebo dokonce možné. To nastává například v provozech kde hrozí jejich znečistění či zanesení (například autodílna), hygienické problémy (v lékařské praxi), nebo případně v důsledku fyzického handicapu osoby. V obecné rovině je ovládání hlasem pro zkušeného uživatele ne až tak praktické a takřka vždy méně efektivní než v případě použití klávesnice a myši, ale pro nového uživatele působí hlasové povely přirozeněji a jsou snadno zapamatovatelné.

Rozpoznání řeči se dělí na dvě základní kategorie, tedy rozpoznání izolovaných slov a rozpoznání spojité řeči.

1.2 Rozpoznání izolovaných slov

Problematika rozpoznání izolovaných slov spočívá převážně v tom jak zjistit do jaké míry jsou si dvě nahrávky (dvě nahraná slova) podobné to opakovat pro testovaný vzorek a celou množinu nahrávek referenčních. Nejpodobnější referenční nahrávka tedy obsahuje stejné slovo jako nahrávka testovaná. S velikostí slovníku lineárně narůstá

výpočetní složitost problému a tak tuto metodu nelze použít na velmi rozsáhlé slovníky. Pro ty se vytváří statistický model, který k jakému vzorku zkoumaná nahrávka nejpravděpodobněji patří. Základním problémem je velké množství nadbytečných informací ve zvukové nahrávce, které nesouvisí s obsahem mluveného slova, jako například tón,výška, rychlost a rytmus hlasu (může se lišit u různých jazyků) nebo okolní hluky a šumy zanesené nahrávací soustavou. Převod nahrávky do podoby, ve které je eliminováno co největší množství informace nesouvisející s významem mluveného slova se nazývá parametrizace.

1.2.1 Parametrizace

Parametrizace spočívá v rozdělení nahrávky do velmi krátkých, vetšinou částečně se překrývajících, úseků, kterým se říká framy (z anglického frame=rám vychází z představy posunováním rámu konstantní velikosti po nahrávce, obsah rámu je onen krátký úsek). Délka framu je v rámci jednoho porovnání konstantní a je určena tak aby lidský hlasový aparát nebyl schopen během jednoho framu významně změnit parametry vyluzovaného zvuku.

V rámci parametrizace se framy analyzují za účelem získání parametrů (jinak příznaků). Hodnoty získaných příznaků by měly co nejlépe popisovat významovou informaci v daném framu. Vektor tvořený všemi získanými parametry z framu se nazývá příznakový vektor.

1.2.2 Příznaky

Historický vývoj příznaků směřuje od výpočetně nenáročných příznaků typu počet průchodu nulou a celková energie framu. K výpočetně složitějším spektrálním příznakům udávajícím energie v určitá části frekvenčního spektra signálu až k příznakům kepstrálním, které se získávají z další fourierovy transformace logaritmického magnitudového spektra signálu. Od těchto takzvaných statických příznaků odvozujeme příznaky dynamické(nebo také delta příznaky), které sledují změnu daného parametru v závislosti na čase. Jejich výpočet probíhá derivací příznaků statických.

1.2.3 Rozpoznání izolovaných pomocí srovnání nahrávek

Pokud chceme srovnat zparametrizované nahrávky narážíme na dva problémy. První je jak porovnat dva vektory příznaků a druhý spočívá v různé délce nahrávek.

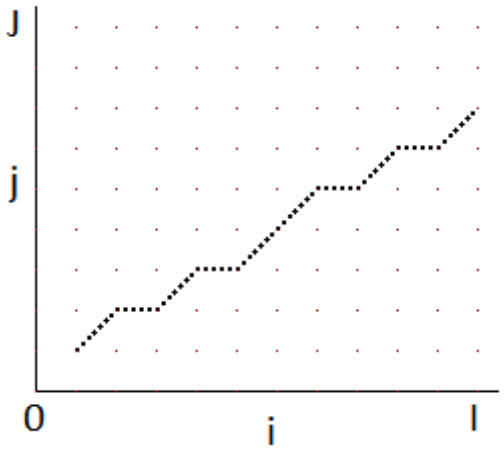
Pro určení rozdílu dvou vektorů se používá takzvaná euklidovská vzdálenost d(x,r) kde x a r jsou příslušné příznakové vektory a P je počet příznaků ve vektoru. Čím větší je tato vzdálenost tím jsou příznakové vektory rozdílnější.

$$d(x,r)=\sum_{p=1}^{P}\sqrt{(x_p-r_p)^2}$$

Obrázek 1: Euklidovská vzdálenost

Druhým problémem je nestejná délka nahrávek, tedy i nestejný počet příznakových vektorů. Dvě porovnávané nahrávky nazveme zkoumaný vzorek a reference. K vzájemnému přizpůsobení délky nahrávky se používají algoritmy borcení času. Jednoduchým případem je algoritmus lineárního borcení časové osy LTW [1][2], (z anglického Linear Time Warping). V podstatě se jedná změnu délky referenční nahrávky na délku zkoumaného vzorku a výpočet příslušných hodnot pomocí lineární interpolace.

Kde slovo má délku I framů a reference J framů, w(i) je index příznakového vektoru z referenční nahrávky se kterým budeme srovnávat příznakový vektor na indexu i z nahrávky testované. D je celková vzájemná vzdálenost nahrávek (čím menší tím jsou si podobnější) Vzájemné přiřazení indexů pak může vypadat třeba takto:



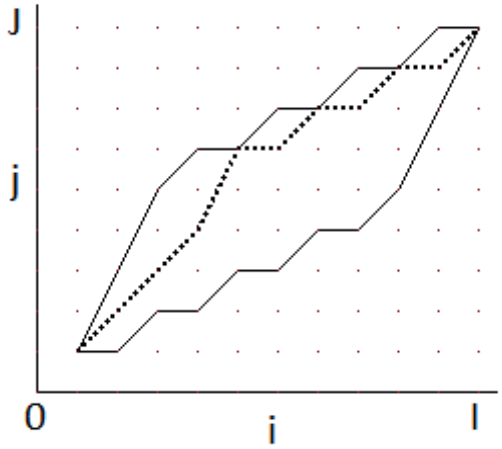
Obrázek 2: LTW přiřazení indexů

Pokročilejší metodou je dynamické borcení časové osy DTW (z anglického dynamic time warping). Vychází se z toho, že tempo lidské řeči se může měnit a tedy různé části slova mohou být u srovnávaných nahrávek různé. Bylo tedy nutné najít metodu jak referenční vzorek přizpůsobit na délku testovacího tak, aby se respektovalo natažení či zkrácení jednotlivých částí slova.

Teoretické množství různých vzájemných přiřazení je I * J. Tento počet se snižuje zavedením podmínek. První podmínkou je podmínka okrajová tedy w(1) = 1, w(I) = J značící, že první i poslední frame testovaného vzorku a reference se boudou porovnávat spolu. Druhou podmínkou je monotonicita w(i)>=w(i-1), tedy, že se nelze po vzorku vracet zpět. Třetí podmínkou jsou takzvané podmínky spojitosti nazívané také itakurovy podmínky podle Fumitada Itakury, který je zformuloval, pro každé vzájemné přiřazení indexu musí platit jedna z těchto podmínek. Podmínky jsou tyto:

1) w(i) = j pouze když w(i-1) != j
2) w(i-1) = j - 1
3) w(i-1) = j - 2

Jedno z možných přiřazení indexu algoritmem DTW [1][2], tedy může vypadat takto: černou čarou jsou zobrazeny podmínky.



Obrázek 3: DTW přiřazení indexů

Je vidět, že v rámci podmínek stále existuje větší množství možných přiřazení, ale už tak je znatelně omezeno. DTW algoritmus hledá takovou cestu, že:

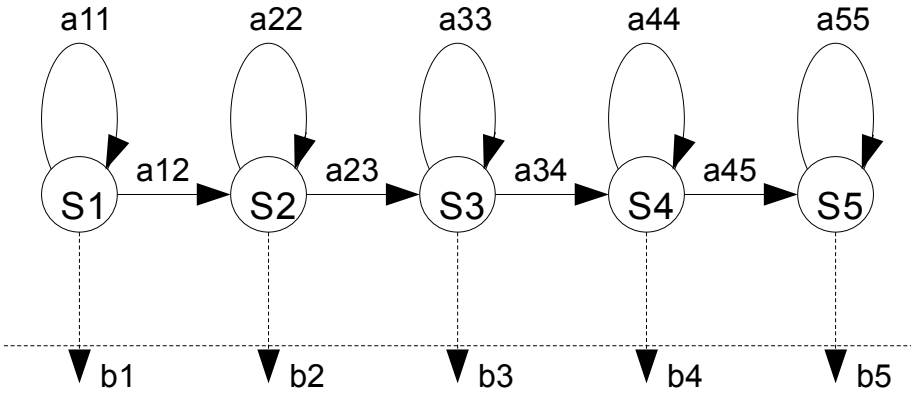
Výpočet lze buď provést hrubou silou nebo využít Bellmanův princip optimality a vzdálenost vyhledat optimálnější cestou.

$$D(X,R)=min(\sum_{1}^{I} d(I_i, J_{w(i)}))$$

Obrázek 4: DTW algoritmus

1.2.4 Rozpoznání izolovaných slov pomocí statistického modelu

Pro statistický model už nelze na nahrávky pohlížet jako řadu framů, ale spíše jako na sadu několika stavů v rámci celého slova, tyto stavy nemají pevný počet framů. Kolem těchto stavů je třeba vytvořit slovní stavový model tvořený malým množstvím stavů, podle kterého budeme nahrané slovo identifikovat. K tomu se využívá metoda skrytých markovských modelů HMM (z anglického Hidden Markov Models) [1][2], která slova reprezentuje stavovým modelem s pravděpodobnostními parametry. Pěti-stavový HMM model slova vypadá takto:



Obrázek 5: HMM levo-pravý pěti-stavový model

Sx jsou stavy modelu.

axy jsou pravděpodobnosti přechodů, tedy že v aktuálním framu přejde model ze stavu x do stavu y.

bs je výstupní pravděpodobnostní rozložení – tedy funkce vracející pravděpodobnost, že daný vstupní příznakový vektor patří do stavu s.

Skryté markovské modely se nazývají skryté kvůli tomu, že ve skutečnosti nevíme jaký frame nahrávky patří ke kterému stavu. Model se proto trénuje iterativní metodou se čtyřmi kroky.

1) Inicializace - Framy všech nahrávek daného slova se rovnoměrně přidělí ke stavům a přejde se na krok 3;
2) Nalezne se rozdělení framů s kterému vychází v modelu vyšší pravděpodobnost, k tomu se využije Viterbiho algoritmus

3) Z těchto vylepšených rozdělení se přepočítají pravděpodobno přechodů v modelu.
4) Pokud se model od poslední iterace dostatečně zlepšil přejdeme do kroku 2 k další iteraci, pokud ne prohlásíme model za dostatečně přesný.

Při rozpoznávání se vyhledá k nahrávce model s nejvyšší pravděpodobností.

1.3 Rozpoznání spojité řeči

Rozpoznávání spojité řeči oproti rozpoznání izolovaných slov vnáší několik dalších problémů.

Velkým problémem je velikost slovníku. Jazyk třeba i omezený na nějakou specifickou oblast (medicína, právo) obsahuje obrovské množství slov (pro češtinu v řádech stovek tisíc). Pokud by měl existovat natrénovaný HMM model pro každé slovo nebylo by možné v rozumném čase vyzkoušet nahrávku na všech modelech a určit nejpravděpodobnější. Navíc ani nejsme schopni určit kde daná slova začínají a kde končí, protože v mluveném projevu se mezi slovy nedělají pomlky, takže s testováním modelů ani nelze začít.

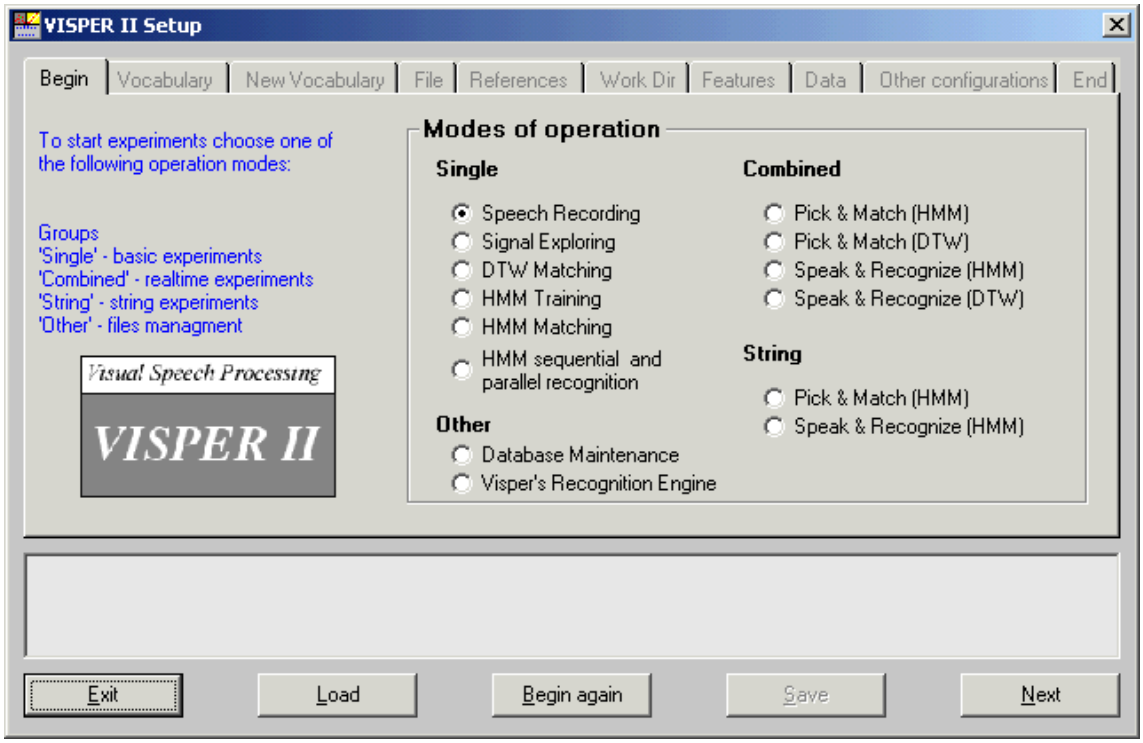
Proto se u spojité řeči používají jednotky menší než slova, většinou fonémy nebo skupiny fonémů, tedy nejmenší zvukové jednotky které mohou nést nějaký význam. Vztah mezi mluveným slovem a fonémy by se dal přirovnat k psanému textu a písmeny. Počet fonémů je pro každý jazyk omezený a pohybuje s maximálně v řádech několika desítek. Rozpoznáním nahrávky se spojitou řečí se získá řada fonémů a k těm je nějak nutné přiřadit text. Přiřazení textu není triviální záležitost, jelikož ve většině jazyků se výslovnost liší od psaného textu a navíc v běžném projevu nejsou mezi slovy mezery takže nelze jednoznačně určit hranice slov. A právě v této oblasti aktuálně probíhá největší část výzkumu a vývoje na pracovištích zabývajících se zpracováním řeči.

2 Programy používané při výuce zpracování signálu a zpracování řeči

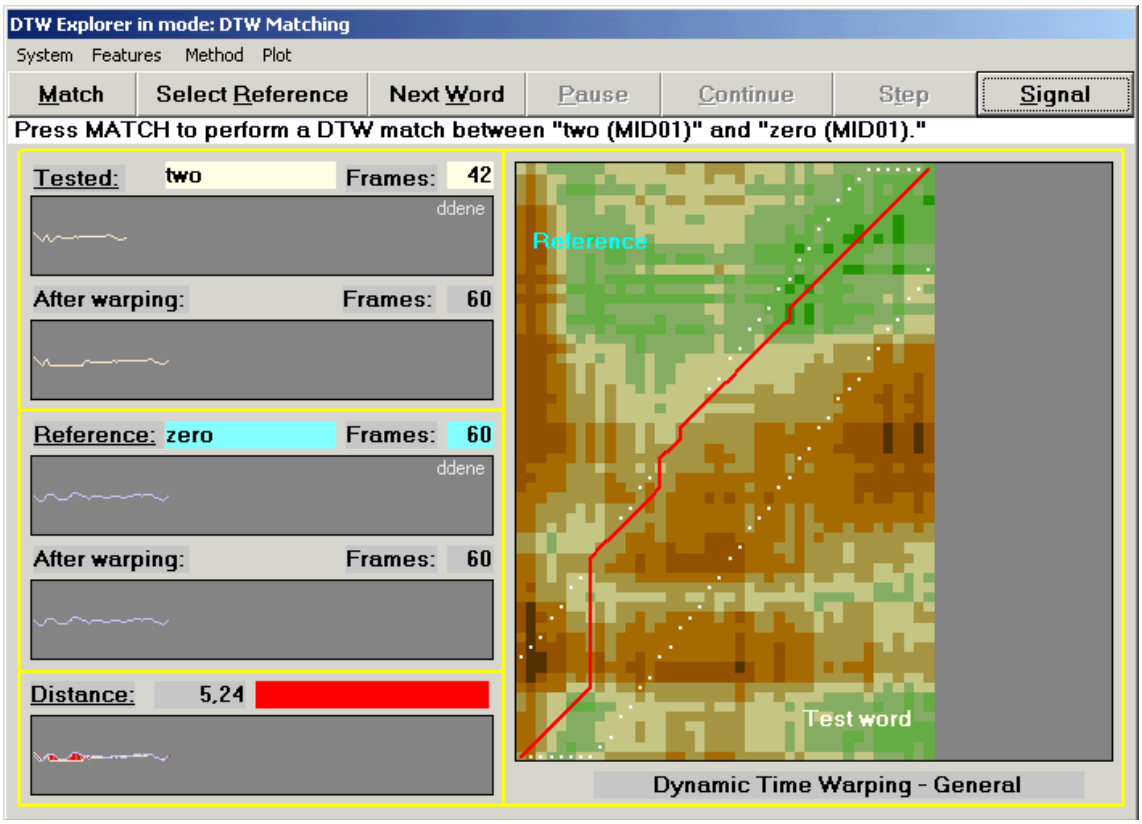
2.1 Visper

Visper je výukový a experimentální software pro pro prezentaci metod rozpoznání řeči vyvinutý Laboratoří počítačového zpracování řeči na Technické univerzitě v Liberci. První verze byla uvedena roku 1997 obsahovala moduly pro tvorbu a zprávu databáze nahrávek řečových vzorků, modul pro grafickou prezentaci trénování a rozpoznávání pomocí skrytých markovských modelů a modul pro experimentování s algoritmy borcení času. V roce 1993 byla vydána verze Visper II, která byla rozšířena o vizualizační modul pro sekvenční paralelní rozpoznání a o moduly práce s textem.

Jedná se o unikátní pomůcku používanou převážně v hodinách výuky zpracování řeči. Takto úzce zaměřený výukový program je při vysokoškolské výuce ojedinělým jevem, většinou se používají programy univerzálnější. Rozhraní celé aplikace je postaveno na dnes už málo používaném konceptu wizardů. Ovládání je tak poněkud nezvyklé, ale lze ho pochopit během několika minut.



Obrázek 6: Visper II rozhraní



Obrázek 7: Visper II DTW porovnávání

2.2 Matlab

Matlab je komplexní matematický program pro numerické výpočty. Název vzniky spojením anglických slov Matrix Laboratory, které znamenají Laboratoř pro práci s maticemi. Vývoj Matlabu započal někdy koncem 70. let, v současnosti za ním stojí firma Mathworks.

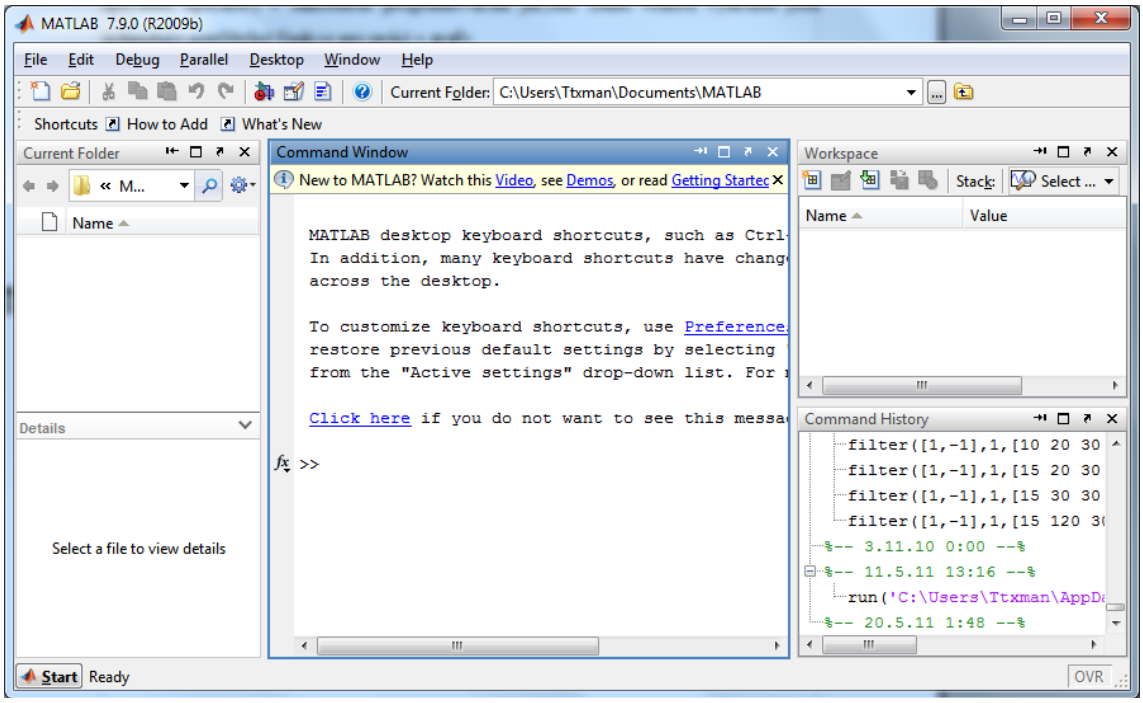
Jak už vyplývá z názvu jedná se o aplikaci pro práci s maticovými a vektorovými výpočty. Primární prací v programu je psaní skriptů v programovacím jazyce MATLAB. Další možností jsou rozšíření programu takzvané toolboxy, z nichž některé umožňují navrhovat různé systémy a simulace v grafickém prostředí.

Při výuce zpracování signálů a řeči je jeho velkou výhodou přímá práce s jednorozměrnými vektory dat, se kterými bez problému pracují vestavěné funkce i speciální operátory v samotném programovacím jazyku. Další velkou výhodou jsou jednoduše použitelné funkce pro práci s grafy.

Nevýhodná se jeví převážně pomalost a občasná nestabilita aplikace a pak také nutnost zakoupit licenci i pro studijní použití. A také nepoužitelnost zdrojového kódu z Matlabu v jiných aplikacích bez nutnosti zakoupení velmi drahých kompilátorů. Proti

programu typu Visper se špatně jeví i komplikovaná příprava animací a jiných prezentačních funkcí.

Uživatelské prostředí programu odpovídá aktuálním standardům skládá se z volně přeskupitelných panelů a přizpůsobitelných toolbarů. Okna programovacího jazyka podporují grafické zvýraznění syntaxe a kód je možné ladit a krokovat.



Obrázek 8: Matlab Uživatelské rozhraní

2.3 Programovací jazyky obecně

Velmi zřídkavou metodou při výuce zpracování signálů případně řeči je použití běžných vyšších programovacích jazyků jako je C++. Jejich výhodou je naprostá volnost uživatele v tom jak má zadaný problém řešit. To je zároveň i hlavní nevýhoda, jelikož pro efektivní použití je nutné dokonale ovládat jak samotný jazyk tak vývojové prostředí. Díky přílišné volnosti a absenci jakýchkoliv pevných základů je k vytvoření funkčního kódu pokusu oproti Matlabu nutné vynaložit příliš mnoho času a úsilí jen k provedení základních vektorových výpočtů a zobrazení grafů.

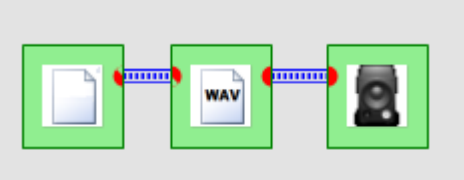
Uživatelské prostředí vývojových nástrojů je převážně obdobné prostředí Matlabu a jedná se tedy o směs volně přeskupitelných panelů a modifikovatelných toolbarů.

3 Návrh a realizace výukové aplikace EasyBlocks

Úkolem této práce bylo navrhnout vizualizační aplikaci určenou pro výuku. Zadané parametry jsou jasné: moderní a jednoduché uživatelské rozhraní, snadná rozšiřitelnost a co nejjednodušší správa dat a pokusů.

3.1 Idea blokového schematu

Základní myšlenkou kolem které je program postaven je blokové schema. Pokud se pokusíme analyzovat výklad v jakémkoliv technickém předmětu, tak výklad většiny algoritmických problémů začíná blokovým schematem. Algoritmus se rozdělí do dílčích bloků, které jsou posléze detailně popisovány. Jelikož EasyBlocks je primárně výukový software a bloková schemata jsou neocenitelná pomůcka pro snadné pochopení algoritmů, bylo záhodno je do aplikace nějak efektivně začlenit. Výsledkem této snahy je to, že modularita aplikace spočívá v relativně jednoduchém vytváření funkčních bloků, které jsou v grafickém prostředí propojovány datovými cestami. Bloky spolu s cestami jsou zobrazené tak, aby právě blokové schema připomínaly. Samotný experiment (pokus) určený k předvedení tedy obsahuje blokové schema, vizualizace funkce některých bloků ve schematu a možnosti úpravy parametrů jednotlivých bloků.



Obrázek 9: Blokové schema tak jak je zobrazované v programu

3.2 Uživatelské rozhraní programu

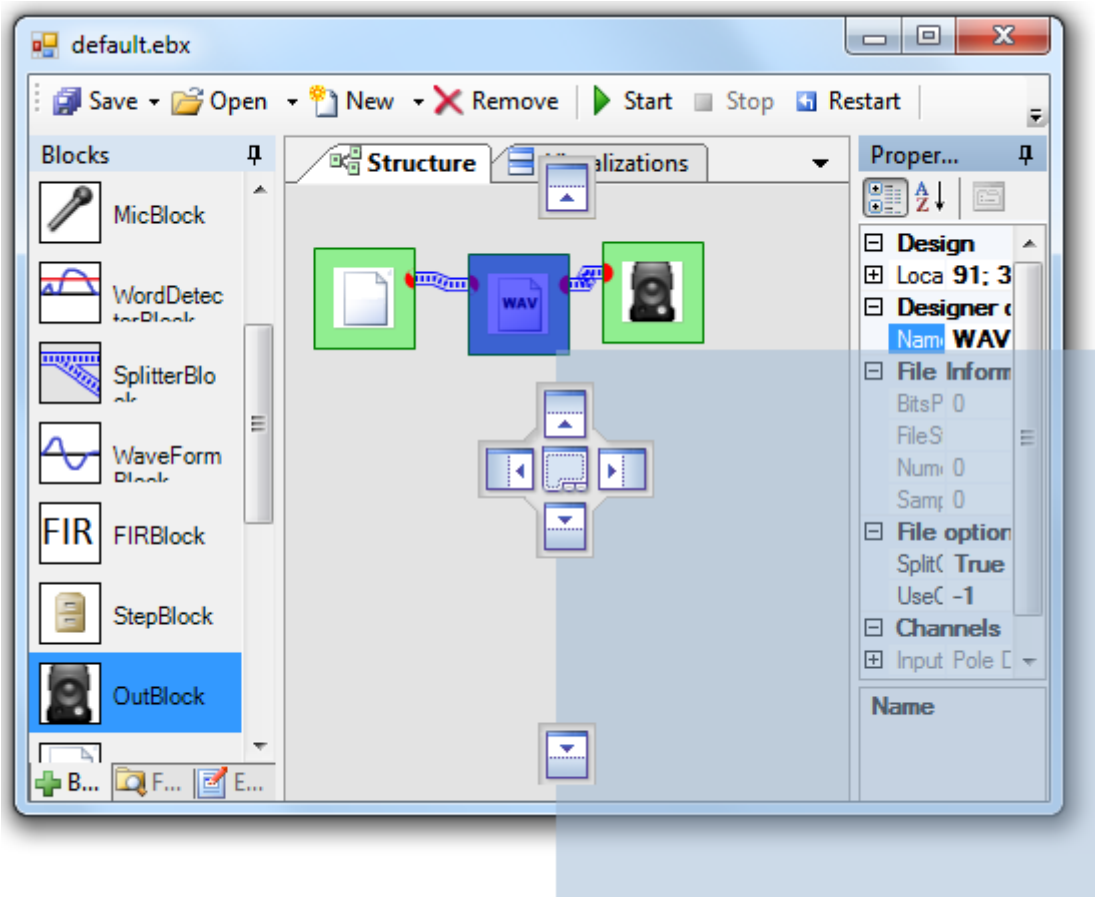
Jako základ uživatelského prostředí byl zvolen koncept známi ze všech integrovaných vývojových prostředí, jako je Microsoft Visual Studio, NetBeans nebo Eclipse. Tedy modulární panely, které je možno libovolně přeskupovat a seskupovat včetně jejich uvolnění do samostatných oken. Navíc toto přeskupení není vázáno na aplikaci, ale přímo na experiment, takže v závislosti na počtu a velikosti vizualizací funkčních bloků může být rozhraní specificky upraveno.

Použitý dokovací systém podporuje vzájemné spojování oken pomocí sjednocení dvou a více oken do společného prostoru, při tom vzniknou známé záložky

umožňující přepnout na libovolné okno ve společném prostoru. Druhou možností je dokování okna dokovat na části do sebe.

Uchopením panelu za jeho horní okraj (jak v zadokovaném tak v nezadokovaném stavu), případně za jeho záložku a tažením se začne přetahovat částečně průhledný světle modrý obdélník a na oknech, ke kterým lze zadokovat se pro vybrání cílové lokace lokace objeví čtyř směrová růžice, případně jenom obdélníky bez středu, které fungují stejně jako růžice. Při přetahování obdélník zobrazuje tvar v jakém se bude okno nacházet při ukončení přetahování na aktuálním místě. Přetáhnutím do volného prostoru se vytvoří plovoucí okno. Přetáhnutím nad růžici se okno zadokuje do části okna podle části růžice, nad kterou se obdélník upustí.

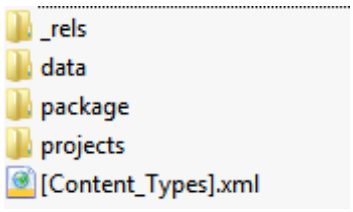
Obrázek 12 zobrazuje dokování ve fázi přetahování panelu. Světle modrý obdélník ukazuje aktuální cílovou pozici přetahovaného panelu. Také jsou zde vidět růžice pro dokování.



Obrázek 10: Ukázka dokování

3.3 Balíčkový systém ukládání

Pro práci s daty byl přijat koncept balíčků známý například z posledních verzí formátů z kancelářských balíků. V podstatě se jedná o soubor komprimovaný metodou ZIP obsahující množinu souborů s různým využitím. V případě EasyBlocks byla balíčkům určena přípona .ebx. Struktura balíčku je následující:

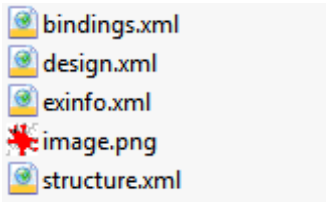


Obrázek 11: Struktura balíčku: kořenový adresář

Složky _rels, package a soubor [Content_Types].xml jsou zachovány kvůli kompatibilitě s použitým balíčkovacím systémem.

Složka data obsahuje datové soubory, ke kterým je možné z aplikace přistupovat, soubory zapsané funkčními bloky do balíčku budou také umístěny sem.

Složka projects obsahuje seznam pokusů v balíčku. Tyto pokusy jsou nezávislé ale sdílejí společná data.



Obrázek 12: Struktura balíčku: pokus

Jak je vidět z obrázku 13, tak nastavení a struktura jednotlivých pokusů je uložena v XML souborech.

bindings.xml obsahuje definované konstanty a jejich případné připojení k prametrům bloků.

design.xml obsahuje nastavení interfacu pokusu, tedy poskládání ovladacích a vizualizačních panelů programu.

exinfo.xml obsahuje detaily o pokusu tak jak je vyplnil autor.

image.png obsahuje náhled na blokové schema pokusu. Používá se při výběru pokusů z balíčku.

Structure.xml obsahuje záznamy o umístění funkčních bloků a jejich propojení.

3.4 Platforma Microsoft .NET

Tato platforma má mnoho vlastností, které vedou k její popularitě. Mezi nejdůležitější patří asi tyto:

Rozsáhlá vnitřní knihovna funkcí pro nejrůznější účely.

Plně objektový model aplikací.

Mocné nástroje pro vývoj a ladění programu.

Mechanizmy pro automatické uvolňování objektů z operační paměti, které zabraňují jinak celkem častým problémům s neuvolňováním paměti programu někdy vedoucí až k jeho pádu.

Podpora několika programovacích jazyků. Jejich počet stále narůstá. (například C#, Visual Basic.NET, F#, J#, C++/CLI, BOO, Iron Python, nebo dokonce PHP pomocí kompilátoru Phalanger). Knihovna vytvořená v jednom jazyku jde automaticky připojit k projektu v jiném jazyku a programátor se nemusí nic řešit.

Jako hlavní vývojová platforma Microsoftu se .NET framework používá v programech vytvořených pro provoz webových a databázových serverů, v webových aplikacích postavených na technologii Siverlight a i jako primární platforma pro programy běžící na mobilních telefonech s operačním systémem Windows Phone 7.

Mezi nevýhody patří pomalejší chod programu díky režii na složitou zprávu paměti. Dopad u správně psaných aplikací ale bývá minimální a celkové zpomalení se udává maximálně do 10%

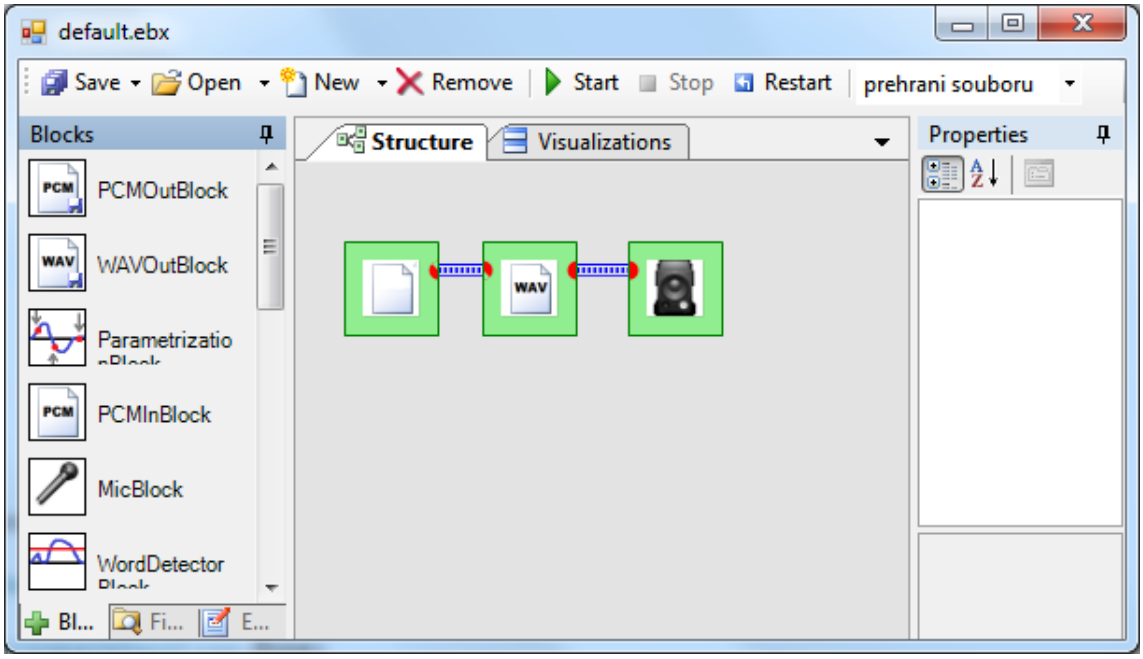
Méně snadné používání funkcí Windows API, než z nativního kódu, ale i přesto jsou součástí .NET frameworku knihovny a funkce zajišťující interoperabilitu s nativním kódem, nebo s programy těžící z technologie Component Object Model.

Možné problémy s portováním na jiné operační systémy, protože pro ně neexistuje oficiální podpora od Microsoftu a běhové prostředí tak musí řešit software třetích stran (mono-project.org).

Někomu také vadí přehnaně upsaná syntaxe jazyků jako je C#, nebo C++/CLI při operacemi s managed objekty.

3.5 Uživatelské rozhraní detailně

3.5.1 Hlavní okno programu



Obrázek 13: Hlavní okno programu

Hlavní okno programu je dostupné od okamžiku s puštění a s jeho uzavřením se EasyBlocks ukončí. Skládá ze se 2 částí.

V horní části je panel nástrojů. Obsahuje tlačítka pro základní funkce programu související s ukládáním, načítáním a spouštěním projektu.

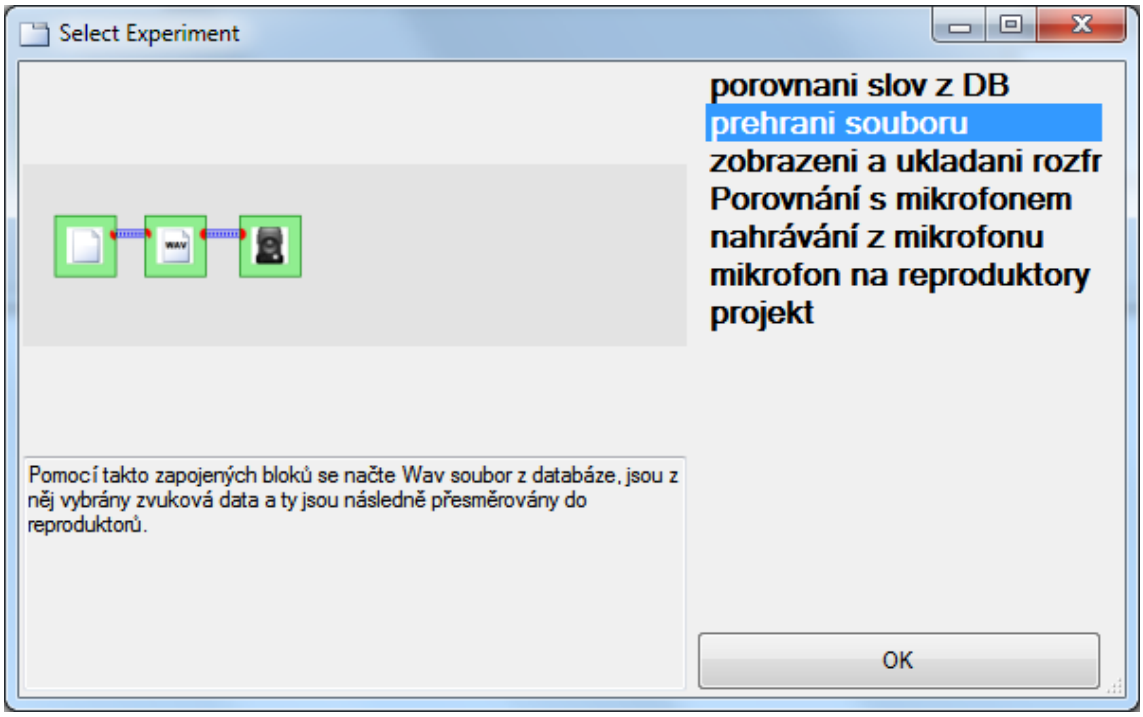
Ve spodní části jsou zadokované panely programu a panely vizualizací funkčních bloků.

3.5.2 Seznam bloků – Panel Blocks

V levé části můžeme vidět zadokovaný seznam bloků, které mohou být v pokusu použity. Z tohoto seznamu je možné bloky pomocí táhni a pusť přidávat do designeru.

3.5.3 Výběr pokusu

Protože jeden balíček může obsahovat více pokusů je pomocí tlačítka Open vybrat i příslušný experiment a zobrazit jeho detaily. Pokud detaily nepotřebujeme je možné rychle se mezi pokusy přepínat pomocí rozbalovacího menu v pravé části panelu nástrojů



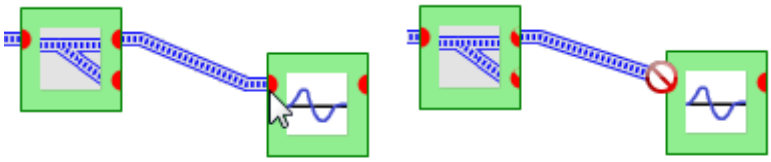
Obrázek 14: Výběr z několika pokusů v rámci jednoho balíčku

3.5.4 Designer – Panel Structure

Designer je vidět ve střední části okna obsahuje blokové schema pokusu. Slouží k sestavení funkčních bloků do projektu a k vytvoření datových kanálů mezi bloky.

Propojení mezi bloky se vytváří pomocí myši. Kanál se započne stiskem levého tlačítka na výstupním bodu funkčního bloku (červený půlkruh na pravé straně bloku) a natažením do vstupního bodu jiného bloku (červený půlkruh na levé straně bloku). Při přiblížení k výstupnímu bodu se kanál přichytne v případě, že bloky lze takto propojit, pokud to nelze kanál se nepřichytne a kurzor myší se změní na zamítavý. Kanál se dokončí povolením levého tlačítka myši ve chvíli kdy je výstup kanálu přichycen.

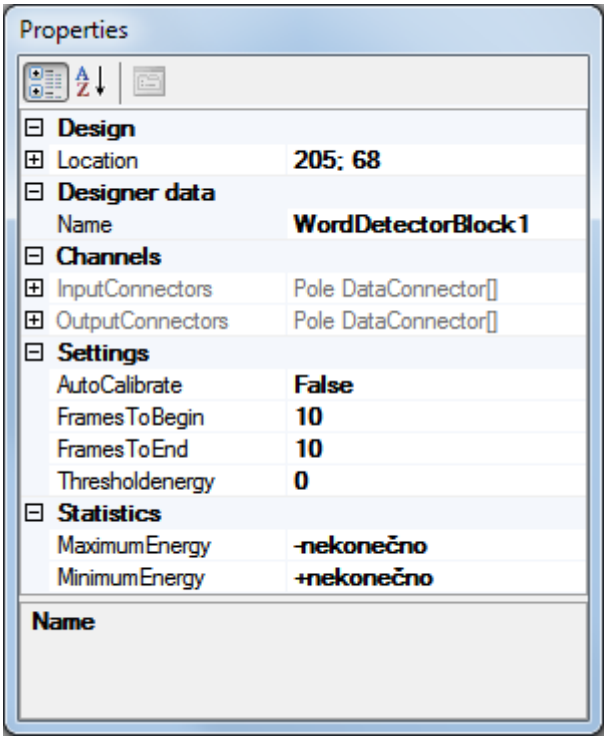
Zobrazené bloky se dají levým tlačítkem vybrat. Vybraný blok lze tažením přesunout na nové místo a klávesou delete smazat. Navíc se při vybrání bloku zobrazí jeho vlastnosti a nastavení v panelu Properties.



Obrázek 15: Designer reakce na propojení

3.5.5 Nastavení bloků – panel Properties

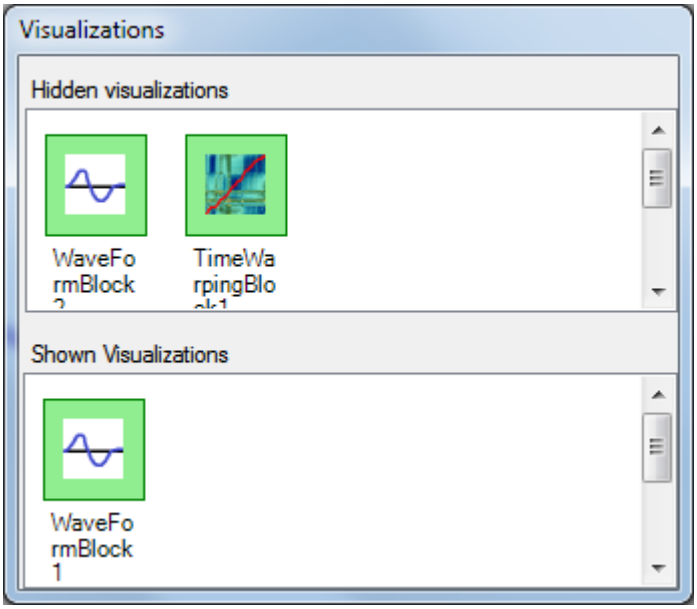
Po vybrání bloku v panelu Structure se jeho vlastnosti a nastavení zobrazí v panelu Properties. Nastavení probíhá po vzoru designéru formulářů Microsoft Visual studia. Každá vlastnost je představována jedním řádkem. Hodnotu vlastnosti lze změnit jednoduchým přepsáním, případně vybráním jedné z hodnot rozbalovacího seznamu. Poslední možností vložení hodnoty u některých vlastností je pomocí tlačítka „...“ v řádku vlastnosti, které otevírá pokročilejší nastavení. Hodnoty také mohou být rozděleny do pojmenovaných kategorií.



Obrázek 16: Panel properties

3.5.6 Přepínání vizualizací - panel Visualizations

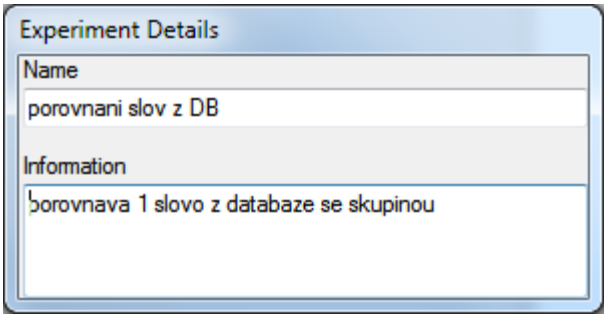
Každý funkční blok může obsahovat i vizualizaci. Například graf nebo panel pro průběžnou změnu nastavení. Panel Visualizations slouží k zobrazení a skrytí vizualizací jednotlivých bloků. V horní části označené „Hidden visualizations“ jsou nezobrazené vizualizace, v dolní části „Shown visualizations“ jsou zobrazené vizualizace bloků. Přesun bloků mezi těmito dvěma částmi je možný buď přetažením bloku z jedné do druhé, nebo je možné bloky přesunovat dvojklikem. Při zobrazení vizualizace se vytvoří nový plovoucí panel. Při jejím skrytí panel zmizí a to i v případě že byla zadokovaná.



Obrázek 17: Panel Visualizations

3.5.7 Detaily pokusu – Panel Experiment Details

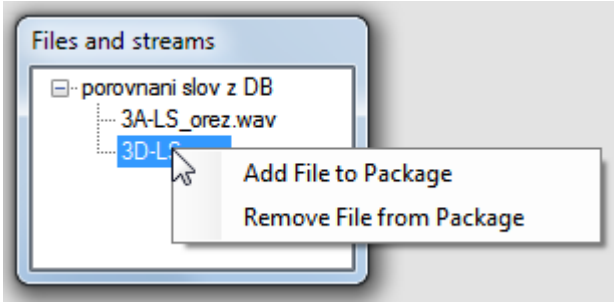
Autor může k experimentu připojit název a popis. K tomu slouží právě tento panel. Název a popis jsou potom vidět při výběru pokusu s balíčku.



Obrázek 18: Panel Experiment Details

3.5.8 Správa souborů – Panel Files and Streams

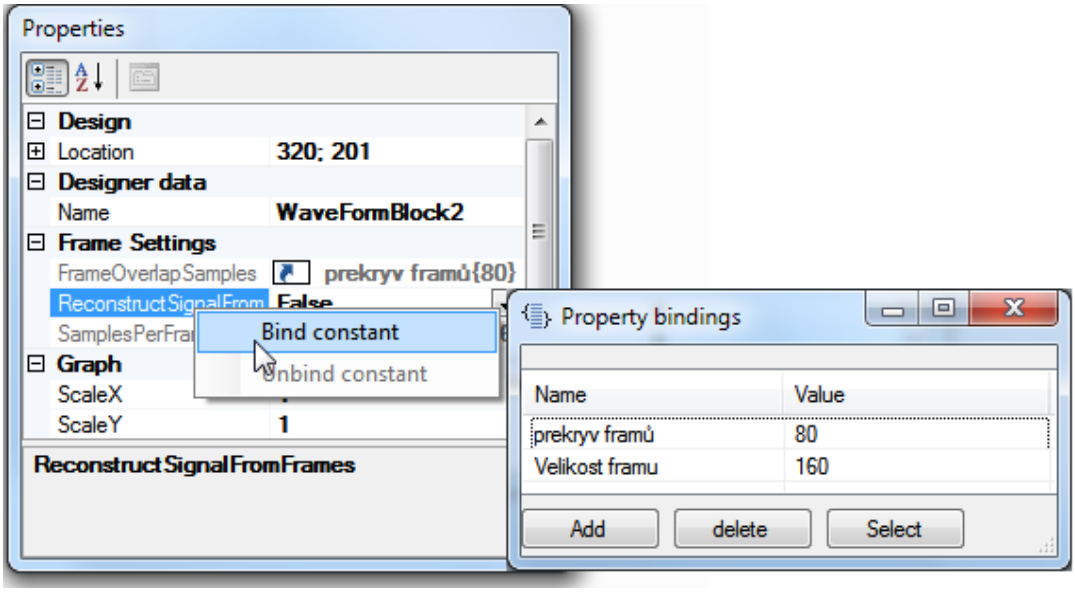
Protože balíček může obsahovat funkčním bloků přístupné soubory je nutné mít z rozhraní programu s těmito soubory manipulovat. Panel Files and Streams tyto soubory zobrazuje a v případě nutnosti je možné je pomocí menu otevíraného pravím tlačítkem myši přidat nebo odstraňovat.



Obrázek 19: Panel Files and Streams

3.5.9 Připínání konstant

Až při programování funkčních bloků se ukázalo, že některé parametry bloků se stále opakují na různých blocích v rámci celého pokusu. Proto byla do programu přidána možnost připínat k jednotlivým hodnotám konstanty. Ke správci konstant se lze dostat z menu otevíraném pravým tlačítkem na hodnotách v panelu properties. Ve správci je možné vytvářet mazat a editovat dvojice název – hodnota reprezentující konstantu. Tlačítko Add slouží k vytvoření nové konstanty, tlačítko delete slouží k odstranění konstanty. Tlačítkem Select přiřadíme vybranou konstantu na vlastnost bloku, ze kterého byl správce otevřen.

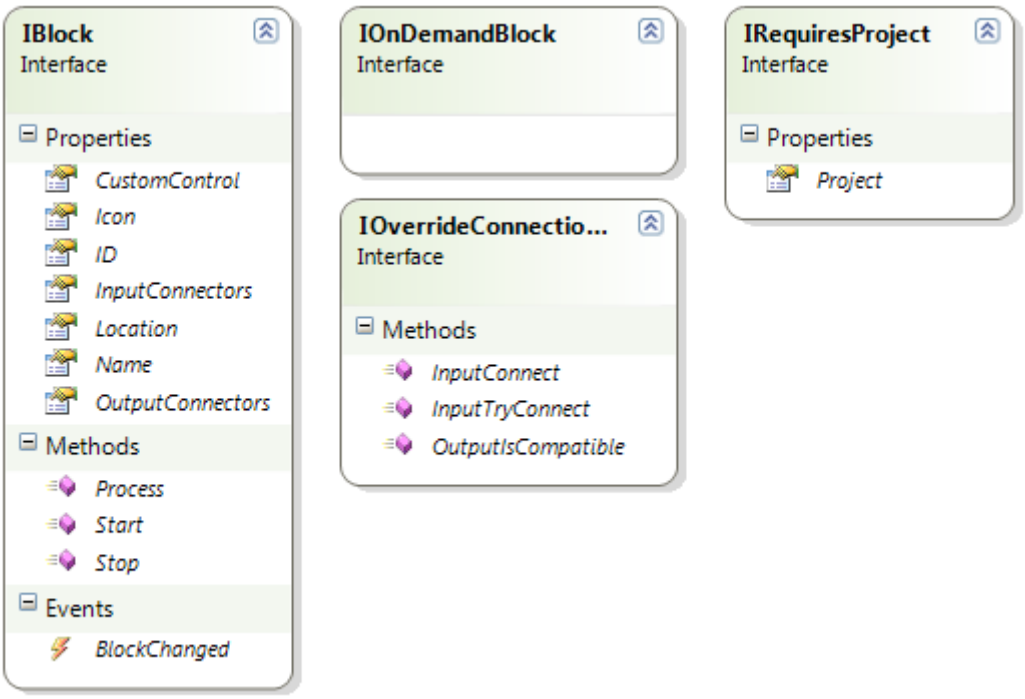


Obrázek 20: Připínání konstant

3.6 Funkční bloky

Funkční blok je grafická reprezentace funkce vytvářející či zpracovávající data. Blok může mít vstupní a výstupní body pro připojení kanálu dat. Je dobré si uvědomit, že kdyby datové kanály mezi bloky byly zcela univerzální tak jdou propojit i bloky, které nejsou datově kompatibilní. To by mohlo způsobit nečekané chování pokusu, který si s takovou situací nedovede poradit. Proto byla do EasyBlocks přidána datová kontrola a všechny vstupy a výstupy z bloků mají zadaný nějaký datový typ, také se u nich určuje jestli se budou přenášet samostatné objekty nebo typované buffery.

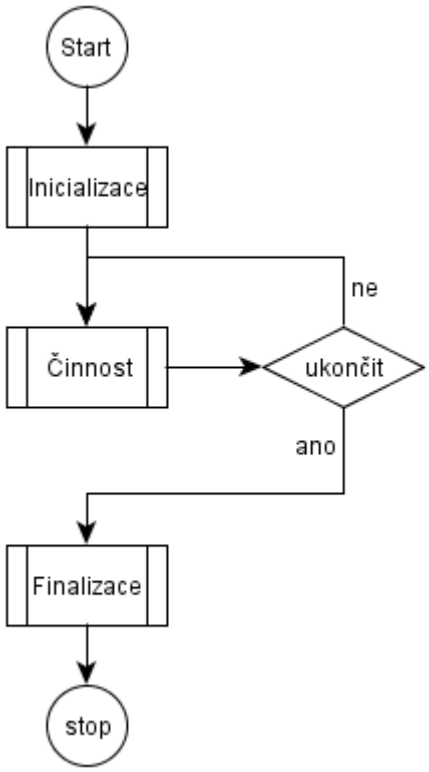
Z programátorského hlediska spočívá vytvoření nového bloku ve vytvoření třídy implementující rozhraní IBlock. V případě, že blok nemá běžet na vlastním vlákně je nutné připojit ke třídě ještě rozhraní IOnDemandBlock. Pokud třída funkčního bloku implementuje rozhraní IOverrideConnection je sama zodpovědná za typovou kontrolu vstupních a výstupních spojení, a případně se jim může přizpůsobovat.



Obrázek 21: Rozhraní pro vytváření funkčních bloků

3.6.1 Běh na vlastním vlákně

Vzhledem k nezávislosti bloků byl s výhodou použit model při kterém každý blok běží na samostatném vlákně a synchronizace probíhá přes datové kanály. Tedy při spuštění pokusu se nejprve bloky inicializují (metodou Start()), poté provádějí činnost a při ukončení se finalizují (metodou Stop()). Vlastní funkce bloku probíhá v metodě Process() která je spouštěna v nekonečném cyklu, k ukončení cyklu dojde pouze v případě, že je v metodě Process() vyvolána neošetřená výjimka, nebo pokud je celý pokus ukončen.



Obrázek 22: Model činnosti funkčního bloku

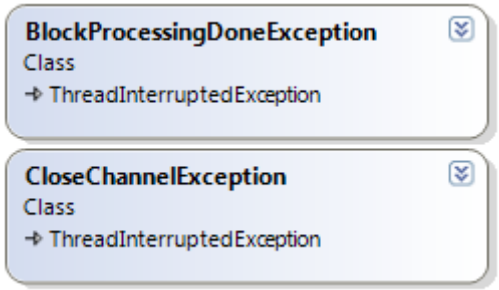
3.6.2 Spouštění na základě požadavku na data

V některých případech je však zbytečné blok provozovat na vlastním vlákně. Pouhou implementací rozhraní IOnDemandBlock a dodržení podmínky, že takovýto blok může mít pouze jeden výstup, běhové prostředí zajistí, aby blok pracoval pouze v případě, že existuje požadavek na data. Funkce bloku bude spuštěna přímo vláknem které data požaduje.

3.6.3 Ukončení a reakce na ukončení datových kanálů

Funkční blok může oznámit, že už do datového kanálu nepošle další data. To se děje zavoláním metody Close() na příslušném kanále. Pokud se jiný blok pokusí z takového kanálu číst a již v něm nejsou akumulovány další data je vyvolána výjimka CloseChannelException, kterou je možné libovolně ošetřit.

Vyvolání výjimky BlockProcessingDoneException slouží jako upozornění běhovému prostředí, že blok dokončil svoji práci bez chyb.



Obrázek 23: Řídící výjimky

3.6.4 Zveřejnění nastavení funkčního bloku

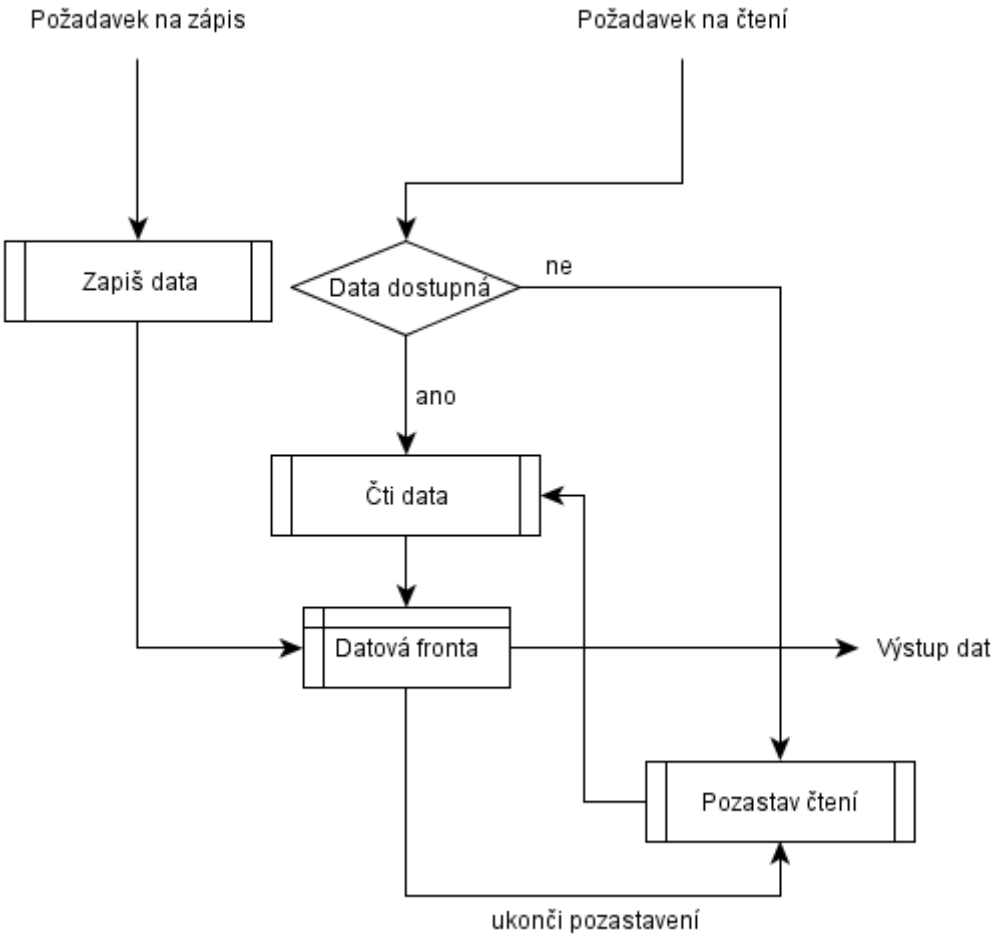
Funkční blok je před používáním nejprve nutné uživatelsky nastavit. Pro zjednodušení práce programátora byl použit shodný model s používaní grafických komponent z jmenného prostoru System.Windows.Forms v editoru Microsoft Visual Studia. Tedy po vybrání bloku se v komponentě odvozené od typu PropertyGrid zobrazí seznam veřejných vlastností třídy bloku. Rozčlenění do kategorií a zobrazení lze u vlastností kontrolovat pomocí Atributů z jmenného prostoru System.ComponentModel shodně jako při návrhu grafických komponent typu winforms na platformě .NET. Viz [7]

3.6.5 Ukládání dat funkčního bloku

Ukládání dat je řešeno serializací do XML pomocí třídy System.Xml.Serialization.XmlSerializer. Tedy je možno implementovat rozhraní IxmlSerializable a ukládání a načítání dat zpracovat ručně. Druhou možností je automatická serializace všech vlastností třídy, kterou je možné řídit pomocí atributů z jmenného prostoru System.Xml.Serialization. Viz [7]

3.7 Datové kanály

Datový kanál jak už bylo dříve naznačeno představuje cestu pro data mezi funkčními bloky. Vzhledem k tomu, že funkční bloky pracují nezávisle na vlastních vláknech, musí být schopný funkční bloky synchronizovat, specificky pokud aktuálně nejsou dostupná žádná data pozastavit vlákno, které data požaduje dokud nebudou nějaká data dodána. V případě, že blok ze kterého jsou požadována data implementuje rozhraní IOnDemandBlock, musí datový kanál zajistit spuštění vnitřní funkce bloku v nekonečném cyklu, dokud není vygenerováno požadované množství dat, potom vrátí kontrolu zpět vláknu které data požaduje.



Obrázek 24: Model datového kanálu

3.7.1 Přenos dat datovým kanálem.

Datový kanál a jeho rozhraní podporují tři odlišné typy datových přenosů na základě typu dat specifikovaného autorem funkčního bloku. Všechny přijímají jako vstup instanci třídy DataBuffer, která obaluje přenášený blok dat.

3.7.2 Netypový přenos datových bufferů

Jedná se o přenos bufferů tvořených polem bytů. Zvláštností je, že tyto buffery jsou automaticky při vytváření zamykány na pevnou adresu v paměti a díky tomu je vytvořen ukazatel na přenášené pole. Tento ukazatel může být předán například systémovým API funkcím, což v ostatních módech přenosu přímo nejde.

3.7.3 Přenos dat jako typované objekty

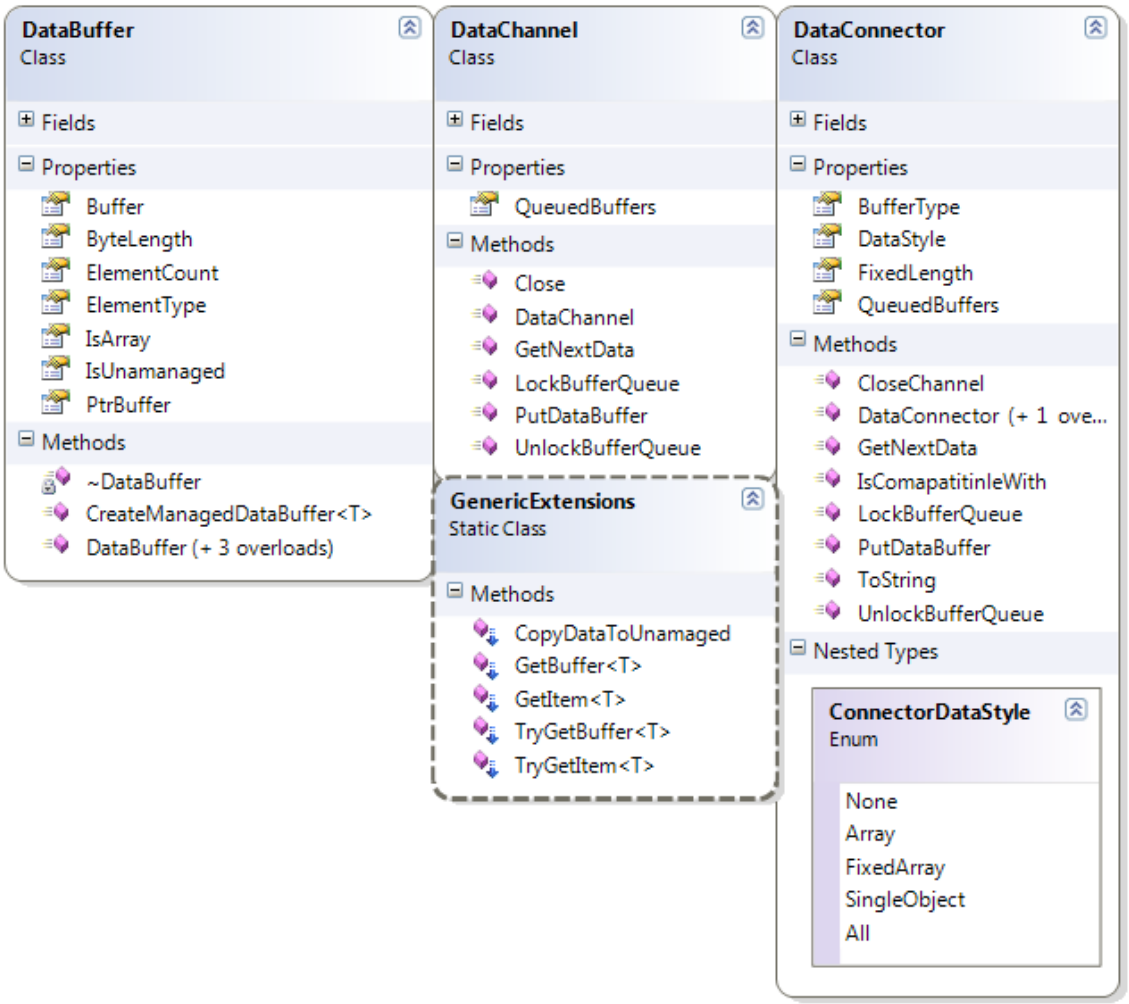
Druhým typem přenosu jsou typované objekty. V tomto režimu se předpokládá, že každá zaslaná položka bude jeden samostatný objekt a nebude to pole primitivních datových typů. Tento režim existuje více méně pouze pro usnadnění práce programátorovy a pro přesnější odlišení typu datového kanálu.

3.7.4 Přenos dat jako typované buffery

Třetím typem přenosu jsou typované datové buffery. V podstatě se jedná o identický způsob přenosu jako u typovaných objektů, ale požaduje se, že vložený objekt musí být pole zadaného typu. Navíc je také možné specifikovat, jestli se v průběhu přenosu může velikost bufferu měnit a případně i, že buffer bude mít vždy nějakou určitou velikost. To znovu slouží k upřesnění typu přenášených dat a zamezuje se tak propojování nekompatibilních funkčních bloků.

3.7.5 Přenos datovým kanálem z programového hlediska

Z programového hlediska je datový přenos tvořen dvěma konektory typu DataConnector, který každý náleží jinému bloku. Při vytváření konektorů se specifikuje typ přenášených dat. Samotný datový kanál typu DataChannel je pro blok nepřístupný a veškerá komunikace probíhá skrze datové konektory. Přenášená data musí být vždy obalená do kontejnerové třídy DataBuffer, která se stará o jejich případný úklid. Vkládání a odebírání typovaných dat do konektorů probíhá pomocí generických rozšiřujících metod implementovaných ve statické třídě GenericExtensions.



Obrázek 25: Programové rozhraní datových kanálů

3.8 Běhové prostředí

Běhové prostředí je v programu reprezentováno třídou Project. Tato třída poskytuje rozhraní pro přístup k vnitřním souborům balíčku. Statické metody pro extrakci informací z balíčků. Dále také spravuje seznamy funkčních bloků a jejich propojení datovými kanály a seznamy konstant, které všechny umí z balíčku načítat a do balíčku ukládat. K samotné práci s balíčkem se používá vestavěné třídy z jmenného prostoru System.IO.Packaging.

Po načtení dat se je jeho činnost omezuje na případné uložení dat a hlavně na správu vláken a datových cest při spuštění pokusu. Průběh spuštění a provedení pokusu je následující:

1) Aplikují se hodnoty konstant na příslušné funkční bloky. V případě, že toto selže je uživateli oznámeno s jakým blokem a jakou hodnotou jsou problémy.
2) Zajistí se případné dotvoření datových kanálů a propojení příslušných bloků podle dat designeru.
3) Inicializují se všechny použité bloky
4) vytvoří se vlákna pro všechny bloky které běží na vlastním vlákně.
5) Postupně se spustí všechny pracovní vlákna
6) Čeká se na zastavení pokusu uživatelem.
7) Pokud existují nějaké bloky, které nedokončily práci je jejich činnost přerušena.
8) Odstraní se případná data která zůstala v datových bufferech
9) Odstraní se vlákna na kterých běžely funkční bloky
10) Proběhne finalizace funkčních bloků.

3.8.1 Potlačení některých bloků v nabídce

Pokud chceme v rámci kompatibility zachovat nějaký funkční blok k načítání, ale nechceme aby se zobrazil v panelu Blocks existuje jednoduchá metoda jak to zařídit. Pokud do souboru SupressBlocks.config, který je v adresáři aplikace umístíme řádek textu odpovídající typu bloku blok se nezobrazí. Jsou podporovány celořádkové komentáře pomocí znaků '#' a ';' jako první znak na řádku. Pokud řádek končí znakem '*' nenačtou se bloky, jejichž typ začíná na text umístěný před hvězdičkou.

3.9 Externí knihovny využité při programování EasyBlocks

3.9.1 DockPanel Suite

Je knihovna pro uživatelské rozhraní, která se snaží napodobovat funkce rozhraní Microsoft Visual Studia. Použito na uživatelské rozhraní.

3.9.2 OpenTK

Je obal kolem tříd 3D API OpenGL. Použito ve vizualizacích LTW a DTW algoritmů.

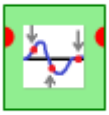
3.9.3 MathNet Iridium

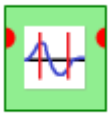
Je knihovna matematických funkcí. Použito na rychlou Fourierovu transformaci v parametrizačním bloku.

4 Přehled implementovaných funkčních bloků a pokusů

4.1 Přehled implementovaných funkčních bloků

V rámci práce vzniklo také 18 jednoduchých prezentačních bloků, jsou to tyto:

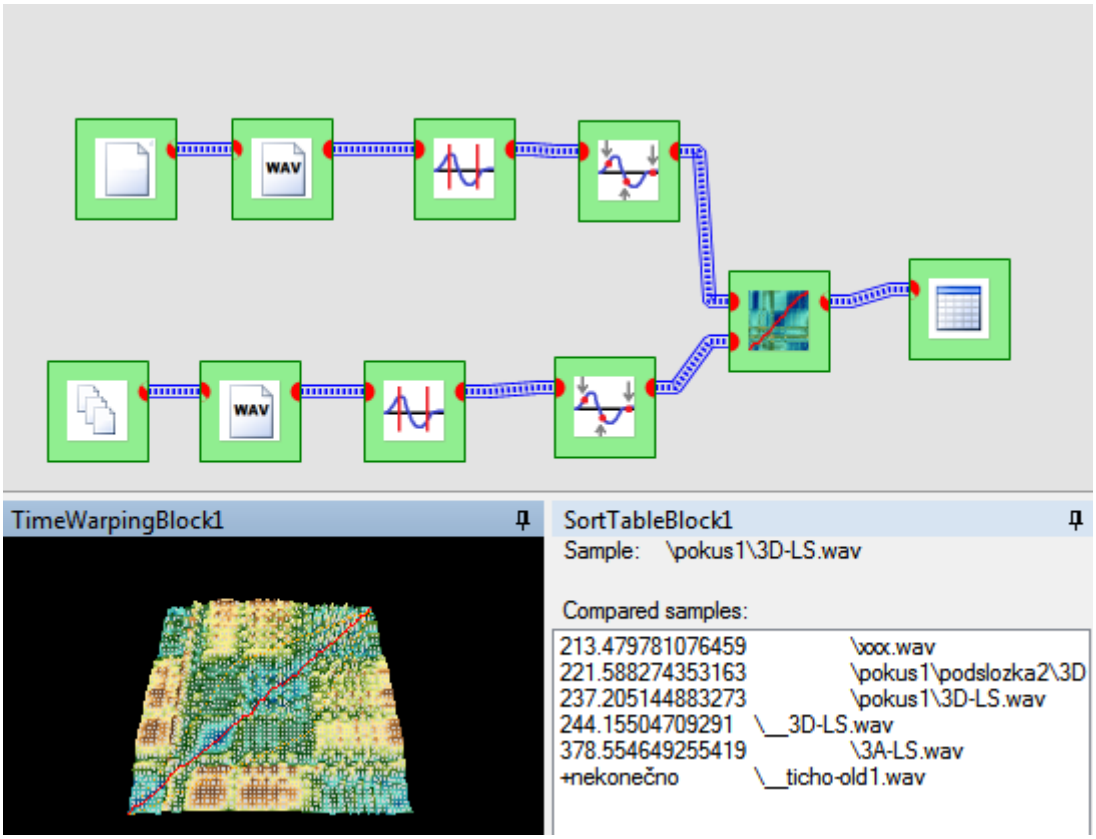
	PCMOutBlock	Záznam dat v PCM formátu (bez hlavičky) do souboru uvnitř nebo i vně balíčku
	WavOutBlock	Záznam dat v PCM formátu s Wave hlavičkou do souboru uvnitř nebo i vně balíčku
	Parametrization Block	Parametrizace zvukového signálu
	PCMInBlock	Načtení PCM dat bez hlavičky ze souboru
	MicBloc	Nahrávání dat z mikrofonu
	WordDetectorBlock	Hladinový detektor pro zjištění hranic slova
	Splitter Block	Rozdvojení datové cesty (například pro zobrazení)
	WaveFormBlock	Zobrazení zvukových dat

	FIRBlock	FIR filtr
	StepBlock	Výstup 2 souborů k porovnání najednou
	OutBlock	Výstup dat do reproduktoru
	DataStreamBlock	Načtení souboru z balíčku nebo i vně balíčku
	TimeWarpingBlock	Blok prezentující animace DTW a LTW algoritmů na vstupních datech
	WavInBlockStream	Načtení PCM dat ze souboru s hlavičkou z datového streamu
	MultipleStreamsBlock	Načtení několika souborů z balíčku nebo i vně balíčku najednou
	FrameBlock	Rozdělení dat na Framy
	WavInBlock	Načtení PCM dat z Wav souboru s hlavičkou ze souboru
	SortTableBlock	Vyhodnocení výsledků TimeWarpingBlocku

4.2 Přehled ukázkových pokusů

4.2.1 Porovnání 2 slov ze souborů v balíčku

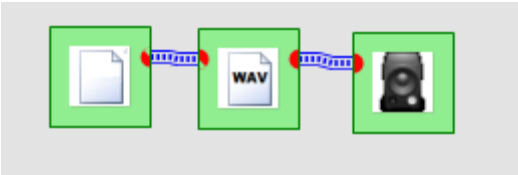
Předvádí připojení dat k TimeWarpingBlocku a jejich vyhodnocení



Obrázek 26: Porovnání 2 slov ze souborů v balíčku

4.2.2 Přehrávání souboru

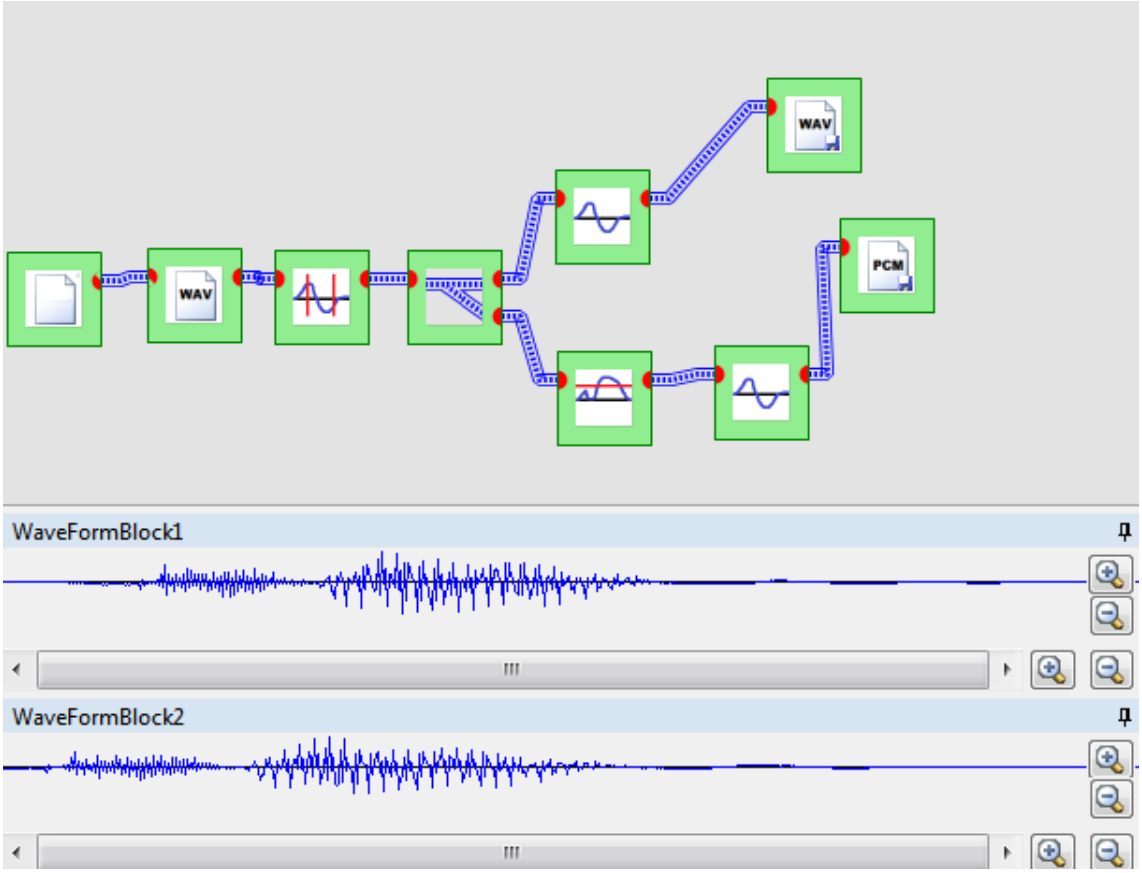
Ukazuje blokovou strukturu pro načtení souboru z balíčku, zpracování načteného streamu a přehrání zvukových dat.



Obrázek 27: Přehrávání souboru

4.2.3 Zobrazení a uložení souboru pod jiným jménem

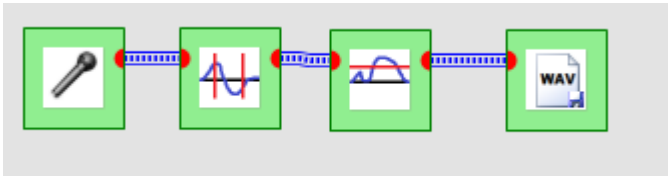
Ukazuje připojení zvukových dat k WaveformBlocku a zobrazení jak dat rozdělených na framy tak dat nedělených.



Obrázek 28: Zobrazení a uložení souboru pod jiným jménem

4.2.4 Nahrávání z mikrofonu do souboru

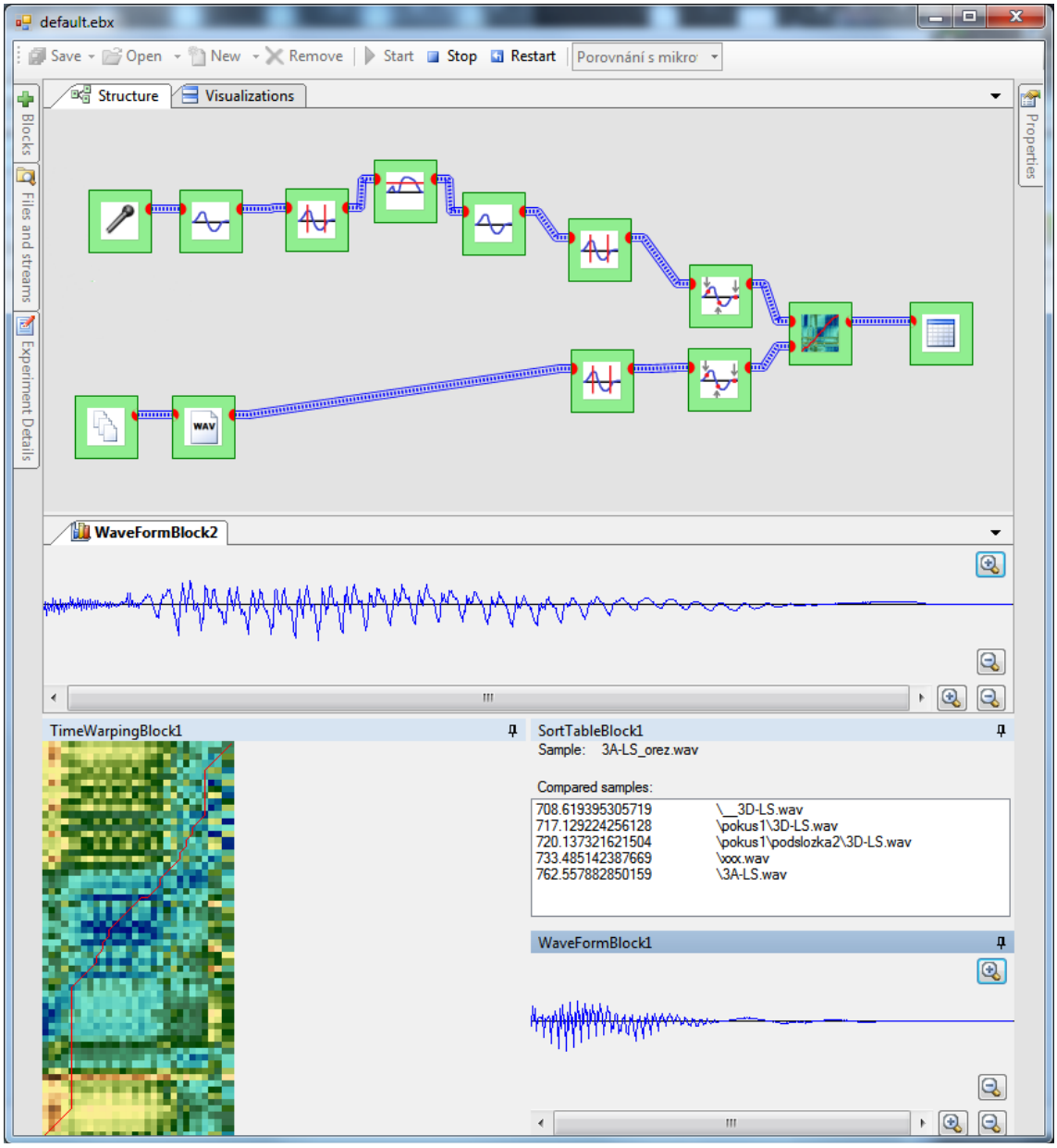
Předvádí použití WordDetectorBlock a mikrofonu k záznamu slova do souboru.



Obrázek 29: Nahrávání z mikrofonu do souboru

4.2.5 Porovnání slova v souboru s daty z mikrofonu

Prezentuje kombinaci mikrofonu slova ze souboru a TimeWarpingBlock v módu 2D zobrazení.



Obrázek 30: Porovnání slova v souboru s daty z mikrofonu

5 Závěr

Aplikace EasyBlocks, která vznikla díky této práci je snadno ovladatelná, snadno rozšiřitelná a jednoduchá na pochopení jak pro uživatele tak pro případného programátora implementujícího nové funkce. Zároveň implementuje silný běhový engine umožňující spouštět, případně nespouštět, funkční bloky na vlastních vláknech podle požadavku programátora daného bloku. Datové cesty mezi bloky jsou schopné přenášet jakákoliv data, požadovaný typ pro konkrétní použití je však díky silné typové kontrole omezen, čímž se eliminuje velká část chyb vycházejících z nekompatibilních bloků a zjednodušuje se práce programátora. Uživatelské prostředí je modulární a pro rozšíření funkcí je možné využít všech možností platformy Microsoft .NET. Výsledný program svým rozsahem překonal 14500 řádků kódu.

Implementace vlastních funkčních bloků pro prezentaci možností aplikace z oblasti zpracování řeči proběhla úspěšně i přes drobné problémy v oblasti API funkcí systému Windows.

Zadání se tedy podařilo splnit v plném rozsahu. Při dalším vývoji aplikace by bylo vhodné zaměřit se na specifické okruhy výuky a vytvořit pro ně samostatné sady funkčních bloků a ukázkové příklady, případně drobné modifikace uživatelského rozhraní podle konkrétních požadavků uživatelů. Také by v případě, že se EasyBlocks začne více používat bylo vhodné připravit instalátor.

Seznam Použité literatury

[1] NOUZA, Jan; KOLDOVSKÝ, Zbyněk; VÍCH, Robert. *Sborník článků ŘEČ A POČÍTAČ*. Liberec : Technická univerzita v Liberci, 2009. 238 s. ISBN 978-80-7372-548-8

[2] HUANG, Xuedong; ACERO, Alex; HON, Hsiao-Wuen. *Spoken Language Processing : A Guide to Theory, Algorithm and System Development*. [Londýn] : Prentice Hall, 2001. 980 s. ISBN 0-13-022616-5

[3] DEMSEY, Seth; WELLS, Jonathan. *MSDN Library* [online]. 2004 [cit. 2011-05-19]. Recording and Playing Sound with the Waveform Audio Interface. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/aa446573.aspx>.

[4] KABAL, Peter. *Audio File Format Specifications* [online]. 2006, Last updated 2011-01-03 [cit. 2011-05-19]. WAVE Specifications. Dostupné z WWW: <http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html>

[7] SELLS, Chriss. *C# a WinForms : programování formulářů windows*. Vydání první. Brno : Zoner Press, 2005. 648 s. ISBN 80-86185-25-0