Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra ... (Softwarového inženýrství)

Bakalářská práce

Multiplatformní implementace spike detektoru pro intraoperační elektrokortikografii

Jakub Drábek

Vedoucí práce: Ing. Petr Ježdík, Ph.D.

5. května 2015

Poděkování

Rád bych poděkoval Ing. Petru Ježdíkovi, Ph.D. za vstřícný přístup a trpělivost při vedení mé bakalářské práce a dále Ing. Radku Jančovi za ochotnou pomoc při analýze zdrojového kódu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 5. května 2015 .....

Odkaz na tuto práci

Drábek, Jakub. *Multiplatformní implementace spike detektoru pro intraoperační elektrokortikografii.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato práce se zaměřuje na analýzu, implementaci a optimalizaci algoritmu spike detektoru epileptickoformních výbojů pro intraoperační elektrokortikografii napsaného v prostředí MATLAB.

Cílem práce je implementovat spike detektor v objektově orientovaném jazyce, který je možné kompilovat pod platformami operačních systémů Linux a Microsoft Windows, případně Android. Tato implementace má splňovat možnost přenesení do jiného systému nebo nasazení v embedded systému. Dále je cílem vytvořit uživatelsky přívětivé prostředí pro obsluhu implementovaného algoritmu s možností vizualizace a protokolování výsledků tak, aby aplikace byla použitelná během neurochirurgického zákroku.

Samotná aplikace je napsána v jazyce C++ s využitím knihovny wxWidgets pro zajištění multiplatformní kompilace na operačních systémech Linux a Microsoft Windows. Dále je zde použito několika dalších externích knihoven především pro zpracování digitálního signálu a práci s daty.

Tato implementace dosahuje výsledků referenčního řešení pro výchozí nastavení detektoru. Optimalizace je provedena především na paměťových nárocích aplikace.

Práce obsahuje analýzu algoritmu spike detektoru, popis provedených optimalizací a změn oproti referenčnímu řešení. Dále je zde popis implementace, srovnání funkčnosti a výsledků vlastní implementace oproti stávající implementaci.

Klíčová slova Spike detektor, Analýza algoritmu, Implementace, Optimalizace, Multiplatformní kompilace, wxWidgets

Abstract

This thesis is focused to the analysis, implementation and optimization of the spike detector epileptiform discharges algorithm for intraoperative electrocorticography, which is written in MATLAB.

The aim of this thesis is to implement the spike detector in an object-oriented programming language that is possible to compile under Linux and Microsoft Windows operating systems platforms, possibly also in the Android environment. This implementation has to fulfil the possibility of the transfer to the another system or the possibility of the usage in embedded system. Next aim is to create a user-friendly environment for handling with implemented system with the possibility of visualization and logging the outcomes, so that the application would be applicable during neurosurgery. Next aim is to create a user-friendly front-end application to control the implemented system with visualization and outcoming data logging capabilities, so that the application would be usable and applicable during neurosurgery.

The application is written in programming language C++ using wxWidgets framework to provide cross-platform compilation capability for Linux and Microsoft Windows operating systems. Furthermore, several external libraries are used, particularly for digital signal processing and data handling.

This implementation achieves the results of a reference solution for default setting of the detector. Optimization has been performed mainly on memory demands of the applications.

This thesis includes an analysis of the algorithm of the spike detector, a description of the optimization and changes compared to the reference solution. Furthermore, there is a description of the implementation, functionality comparison and results of the presented implementation compared to the reference implementation.

Keywords Spike detector, Analysis of the algorithms, Implementation, Optimization, Multi-platform compilation, wxWidgets

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Jedním z neurologických onemocnění mozku je epilepsie. Jedná se o onemocnění záchvatovitého charakteru, kde dochází podle rozsahu k postižení části mozku (parciální záchvat) nebo mozku celého (generalizovaný záchvat). Během parciálního záchvatu bývá do určité míry narušeno vědomí a projevy jsou dány umístěním oblasti vzniku záchvatu. Při tomto druhu záchvatu část nemocných pociťuje před samotným záchvatem předzvěst neboli auru. Pokud dochází k šíření záchvatu dále, záchvat se rozvíjí do obrazu takzvaného částečného komplexního, kdy je již porušeno vědomí a projevují se příznaky motorického charakteru. Během generalizovaného záchvatu dochází ihned ke ztrátě vědomí. Těchto typů záchvatů existuje více, přičemž nejtěžší formou jsou generalizované záchvaty s křečemi. Při nich dochází k poruše vědomí až na několik hodin. Během léčby epilepsie se využívá léčby léky (antiepileptiky) nebo v případě, že antiepileptika nedokáží potlačit záchvaty, je možné přistoupit k chirurgické léčbě. [3]

Nalezení charakteristických vlastností epileptogenní tkáně (tzv. biomarkery) je významných krokem pro zlepšení předoperační diagnostiky a pro přesnější lokalizaci části mozku zodpovědné za záchvaty. Jednou z takovýchto vlastností epileptické tkáně jsou například elektrografické mezizáchvatové epileptiformní výboje (spikes, IED).

Tyto výboje jsou využívány pro diagnózu epilepsie, monitorování aktivity nemoci a lokalizaci epileptogenní tkáně.

Lokalizace a určení vlastností mezizáchvatových výbojů je široce využívána v průběhu předoperačních vyšetření za účelem získání informace o uspořádání epileptické sítě a pro naplánování místa a rozsahu chirurgického zákroku. [4]

Za účelem lokalizace těchto výbojů pracovníci výzkumné skupiny ISARG (Intracranial Signal Analysis Research Group) vytvořili a implementovali algoritmus spike detektoru vnitrolebečních záznamů, který pracuje na principu statistického rozložení obalu signálu obsahujícího mezizáchvatové epileptiformní výboje a aktivitu pozadí. Tento algoritmus je implementován ve skriptovacím jazyce MATLAB [5].

Tato práce se zabývá analýzou, optimalizací a implementací tohoto algoritmu spike detektoru. Cílem implementace je převést algoritmus do objektově orientovaného jazyka, který je možné kompilovat pod platformami operačních systému Linux a Microsoft Windows, případně Android. Dále je cílem vytvořit uživatelsky přívětivé prostředí pro obsluhu algoritmu s možností vizualizace a protokolování výsledků tak, aby daná aplikace byla použitelná během neurochirurgického zákroku.

Kapitola 1

Teoretický rozbor úlohy

1.1 Aktuální stav implementace algoritmu

Algoritmus spike detektoru vyvíjený výzkumnou skupinou ISARG je implementován ve skriptovacím jazyce MATLAB. Tato implementace je dostupná z [6] ve verzi 20 v době psaní této práce. Součástí balíčku, kromě algoritmu spike detektoru, jsou anonymizovaná vstupní data sedmi iEEG záznamů, předpočítané výsledky pro tyto záznamy a skript pro spuštění algoritmu a následné grafické zobrazení výsledků detekce jednoho zvoleného kanálu.

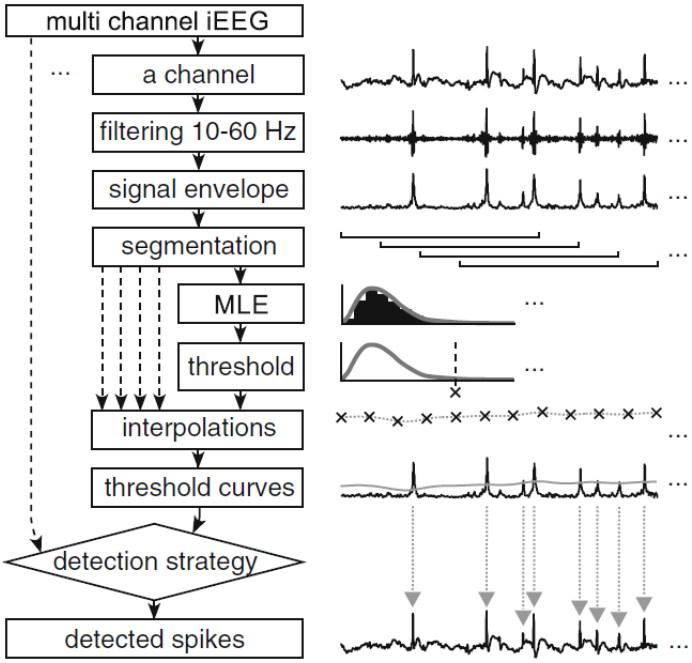
1.2 Analýza předloženého algoritmu

Algoritmus spike detektoru je založen na odlišných morfologických a statistických vlastnostech mezi výbojem a aktivitou pozadí.[1]

Tento algoritmus můžeme rozdělit na tři části:

1. předzpracování vstupního signálu,
2. lokalizaci úseků podezřelých z výskytu IED,
3. vyhodnocení a přesná lokalizace IED.

Schéma algoritmu je zobrazeno na obrázku 1.1.



Obrázek 1.1: Schéma algoritmu [1]

1.2.1 Vstupní hodnoty a nastavení

Vstupní hodnoty

Vstupními hodnotami pro tento algoritmus jsou tyto tři následující parametry:

1. Vstupní signál, tím je myšlena datová sada iEEG záznamů o několika kanálech. Pro lepší pochopení si takovouto sadu můžeme představit jako matici $T \times CH$, kde CH reprezentuje počet kanálů datového záznamu a T počet jednotlivých záznamů v každém kanálu.

2. Vzorkovací frekvence (FS) vstupního signálu.

3. Textový řetězec definující změny výchozího nastavení algoritmu spike detektoru. Tento parametr může být vynechán.

Nastavení detektoru

V algoritmu spike detektoru se pracuje s několika proměnnými, které ovlivňují výsledek a průběh algoritmu. Tyto hodnoty je možné změnit vstupním parametrem nebo je ponechat ve výchozím nastavení. Tyto proměnné jsou uvedeny v následující tabulce s jejich výchozím nastavením a krátkým popisem.

Název	Hodnota	Popis
bandwidth	[10 60]	Spodní a horní frekvence filtrování. Spodní hranice musí být větší nule. Horní hranice musí být menší než hodnota *decimation*.
k_1	3,65	Prahová hodnota k identifikaci jasného výboje.
k_2	0	Prahová hodnota k identifikaci nejasného výboje. Musí platít $0 \leq k_2 < k_1$.
buffering	300	Velikost segmentu v sekundách. Minimální velikost $10 \cdot winsize$.
main_hum_freq	50	Síťový šum v Hz.
f_type	1	Typ filtru: 1 - Chebyshev typ II, 2 - Butterworth, 3 - FIR.
discharge_tol	0,005	Čas mezi výboji v jednom úseku.
polyspike_union_time	0,12	Časový údaj k identifikaci vícenásobné detekce.
decimation	200	Horní mez vzorkovací frekvence v Hz. Signál s vyšší vzorkovací frekvencí bude decimován na tuto hodnotu.
winsize	$5 \cdot fs$	Velikost klouzavého okénka.
noverlap	$4 \cdot fs$	Velikost překrytí segmentů.

Tabulka 1.1: Výchozí nastavení detektoru

1.2.2 Počáteční úprava signálu

V této části se zpracovává a upravuje vstupní signál. Následující kroky jsou aplikovány na jednotlivé kanály zvlášť.

1. V první fázi dochází k převzorkování vstupního signálu v případě, že vstupní signál má vyšší vzorkovací frekvenci než je nastavená hodnota *decimation*. Výchozí hodnota pro decimaci je 200 Hz.

2. V druhé fázi je vstupní signál filtrován pomocí filtrace bez posunu fáze (zero-phase filtering) v nastaveném pásmu pomocí High pass a Low pass digitálního filtru. Pro toto filtrování je použit Chebyshev typ II (zvlnění v závěrném pásmu), Butterworth nebo FIR filter design. Výchozí nastavení je pásmo $10-60$ Hz za použití Chebyshev typ II filter designu.

3. Po fázi filtrování ve zvoleném rozsahu nadchází fáze filtrování síťového šumu pomocí banky dvojpól filtrů s výchozím nastavením mezního kmitočtu 50 Hz a jeho harmonických složek s šířkou pásma 4 Hz.

4. Po fázích filtrování následuje výpočet okamžité obálky pomocí absolutní hodnoty Hilbertovy transformace.

1.2.3 Detekce výbojů

Interiktální výboj reprezentuje ostrý tranzient, který je charakterizován lokálním zvýšením energie zejména v pásmu $10 - 60$ Hz. To se projeví v signálové obálce lokálním maximem.

Detekce výbojů je založena na nalezení lokálních maxim, která jsou dostatečně odchýlena od aktivity pozadí pomocí křivky definované statistickým prahem. Analýza probíhá pro každý kanál signálu samostatně. [4]

1. Obálka vstupního signálu je rozdělena do segmentů za využití klouzavého okénka. Výchozí nastavení velikosti segmentu je pět sekund s 80% překryvem mezi po sobě jdoucími segmenty.

2. Pro každý segment obálky jsou odhadnuty parametry lognormální distribuce μ a σ, sloužící pro statistický popis aktivity pozadí.

   Pro účel výpočtu μ a σ je použit lognormální statistický model, který je charakterizován následující rovnicí:

   $$y = f(x|\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\frac{(\ln x-\mu)^2}{2\sigma^2}}, \tag{1.1}$$

   kde

   - y je hustota pravděpodobnosti,
   - x je amplituda obálky, musí platit $x > 0$,
   - μ a σ jsou odvozené parametry Gaussova rozdělení:

   $$\mu = \frac{1}{N}\sum_{i=1}^{N}\ln(x_i), \tag{1.2}$$

   $$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}[\ln x_i - \mu]^2}, \tag{1.3}$$

   kde N je velikost segmentu obálky. [1]

3. Odhadnutá řada parametrů μ a σ lognormální distribuce je dále vyhlazena filtrem klouzavých průměrů s řádem odpovídajícímu velikosti segmentačního okna.

4. Dále jsou parametry modelu interpolovány (metodou cubic spline interpolation) na vzorkovací kmitočet originálního signálu.

5. Z odhadnutých parametrů modelu jsou spočteny prahovací křivky, po jejichž překročení obálkou je úsek označen jako oblast s výskytem výboje. Prahy th_1 a th_2 slouží pro stanovení různé citlivosti detekce.

$$th_1 = k_1 \cdot [Modus + Median], \tag{1.4}$$
$$th_2 = k_2 \cdot [Modus + Median], \tag{1.5}$$
$$Modus = e^{\mu - \sigma^2}, \tag{1.6}$$
$$Median = e^{u}, \tag{1.7}$$

kde *Modus* a *Median* jsou odhadnuté hodnoty ze statistického modelu používajícího MLE [7].

1.2.4 Vyhodnocení výsledků

V poslední části algoritmu jsou v úsecích obálky přesahující prahovou křivku lokalizovány přesné pozice výbojů.

V označených úsecích jsou hledána lokální maxima obálky (časová značka výboje) a vícenásobné detekce jsou sloučeny do jediné - vícenásobnou detekcí jsou označeny detekce se separací menší než *polyspike_union_time*. Výboje přesahující prahovou hodnotu th_1 jsou označený jako jasné výboje. Akceptovány jsou také nejasné výboje přesahující práh th_2, pokud se ve stejný okamžik nachází v jiném kanále jasný výboj.

1.2.5 Návratové hodnoty

Skript implementující algoritmus spike detektoru má těchto pět návratových hodnot:

- Struktury *MARKER*, *discharges*, *out*, které jsou popsány dále.
- Okamžitou Hilbertovu obálku *envelope*. Jedná se o matici s velikostí decimovaného signálu.
- Prahové křivky *background*. Jedná se taktéž o matici s velikostí decimovaného signálu.

Struktura *MARKER*

Tato struktura obsahuje položky uvedené v tabulce 1.2.

Název	Popis
d	Převzorkovaný vstupní signál.
fs	Aktuální vzorkovací frekvence.
M	Matice o velikosti vstupního signálu. Obsahuje hodnoty siganalizující: 1 ... zřejmý výboj, $\frac{1}{2}$ ... nejednoznačný výboj, 0 ... pozici bez výboje.

Tabulka 1.2: Návratová struktura *MARKER*

Struktura *discharges*

Struktura vícekanálových událostí popisující výskyt výbojů dohromady. Položky *MV* a *MA* jsou matice typu $T \times CH$, kde T je počet záznamů, CH počet kanálů. Jednotlivé položky této struktury jsou uvedeny v tabulce 1.3.

Název	Popis
MV	Matice, obsahující hodnoty, které siganalizují: 1 ... zřejmý výboj, $\frac{1}{2}$ ... nejednoznačný výboj.
MA	Matice obsahující max. amplitudu obálky nad pozadím.
MP	Počáteční pozice multikanálové události.
MD	Doba trvání události.
MW	Hodnota kumulativní distribuční funkce výboje z lognormálního modelu.
MPDF	Hodnota hustoty pravděpodobnosti.

Tabulka 1.3: Návratová struktura *discharges*

Struktura *out*

Struktura popisující jednotlivé detekované výboje. Obsahuje položky uvedené v tabulce 1.4. Proměnná *fs* značí hodnotu vzorkovací frekvence zpracovávaného signálu.

Název	**Popis**
pos	Pozice výboje v kanálu. V sekundách od začátku signálu.
dur	Doba trvání s fixní hodnotou $\frac{1}{fs}$.
chan	Kanál ve kterém byl výboj detekován.
con	Typ výboje: 1 - zřejmý, 0,5 - nejednoznačný.
weight	Statistická významnost "CDF".
pdf	Statistická významnost "PDF".

Tabulka 1.4: Návratová struktura *out*

1.3 Optimalizace předloženého algoritmu

Nejvýraznějším bodem optimalizace je použití vícevláknového zpracování ve fázích filtrování dat pomocí filtru bez posunu fáze. Zde probíhá filtrování jedné sady dat (například jednoho kanálu vstupních dat) na nově vytvořeném vláknu určeném jen pro tuto úlohu.

Další optimalizace je provedena při hledání vícenásobných detekcí. Zde dochází k zpracování segmentů signálu označených jako oblast s výskytem výboje. Během tohoto zpracování se vytváří vektory o velikosti zpracovávaného segmentu s mezivýsledky a dále se zpracovávají nebo se pomocí nich dopočítávají výstupní hodnoty. Během tohoto procesu se tyto vektory celé prochází. Optimalizací v tomto bodě je nahrazení těchto vektorů proměnnými a výpočtu odpovídajících výsledků během jednoho průchodu segmentem dat.

Kapitola 2

Požadavky na novou implementaci

Podle [8] dělíme základní požadavky softwarového systému na funkční, které popisují funkčnost systému a obecné (nefunkční), které popisují omezení kladená na systém, určují dodržování standardů a podobně.

2.1 Funkční požadavky

Funkční požadavky na aplikaci spike detektoru jsou tyto:

- načítání vstupních dat ze souborů typu EDF/EDF+ [9],
- nalezení pozic výbojů s maximálně 1% chybovostí,
- výstupem programu struktury *discharges* a *out*,
- zaprotokolovávání výsledků analýzy,
- vizualizace nalezených výsledků,
- načtení zaprotokolovaných výsledků a jejich zobrazení ze souboru,
- možnost změny výchozího nastavení spike detektoru.

2.2 Nefunkční požadavky

Nefunkčními požadavky jsou:

- robustnost – případná chyba algoritmu detekce, vyvolání výjimky nezpůsobí pád aplikace,

- ošetření paměťových nároků v případě velkých[1] vstupních dat,
- možnost kompilování a spuštění aplikace pod platformami vycházejících z operačních systémů Linux a Microsoft Windows, případně Android,
- použití objektově orientovaného programovacího jazyka.

2.3 Rozdíly oproti implementaci v jazyce MATLAB

Implementace spike detektoru v této práci má několik odlišností oproti referenční implementaci v jazyce MATLAB. Tyto odlišnosti vychází z funkčních požadavků a doporučení či kompromisů, ke kterým bylo přistoupeno během konzultací s vedoucím této práce Ing. Petrem Ježdíkem, Ph.D. a autorem referenční implementace Ing. Radkem Jančou. Těmito rozdíly především jsou:

- Během fáze filtrování signálu filtrem bez posunu fáze v nastaveném pásmu je využito Chebyshev typ II filter designu pro výchozí nastavení, kterým je $10 - 60$ Hz . Koeficienty pro toto filtrování jsou v programu předpočítány pomocí MATLABu. V případě nastavení jiného pásma je využito Butterworth filter designu, kde se již koeficienty vypočítávají pro zvolené pásmo.
- Výstupní hodnoty detekce jsou pouze struktury *discharges* a *out*. Struktura *MARKER*, prahové křivky a okamžitá Hilbertova obálka jsou z návratových hodnot vynechány a slouží pouze pro vnitřní potřeby detektoru.
- Grafický interface aplikace.
- Možnost uložení a načtení výsledků analýzy.

[1] Vstupní soubory s velikostí několik stovek MB či několik GB.

Kapitola 3

Návrh vlastního řešení

3.1 Možnosti řešení

Jedním z hlavních kritérií pro implementaci spike detektoru je funkčnost pod různými platformami (multiplatformní aplikace), a to pro platformy vycházejících z operačních systému Linux a MS Windows, případně mobilního operačního systému Android.

Požadavek vytvářet aplikace pro různé operační systémy splňuje více programovacích jazyků, kterými jsou například jazyky C++ [10], Java [11] a skriptovací jazyk Python [12].

Všechny tyto tři zmíněné jazyky taktéž splňují požadavek objektově orientovaného jazyka. Dále, mimo požadované operační systémy založené na jádře Linux a systémy MS Windows, nabízí možnost vytvářet aplikace pro mobilní operační systém Android. K tomu je ale nutné využít dalších externích zásuvných modulů a knihoven, případně i aplikací.

Možnost vývoje pro systém Android

Pro vytvoření aplikace spustitelné na systému Android můžeme použít uvedené jazyky s některými specifiky. V případě jazyka C++ je nutnost využívat sadu nástrojů Android NDK [13] a v případě použití jazyka Python je možnost využít například open source Python knihovny Kivy [14] s použitím aplikace Kivy Launcher [15]. Nicméně obvyklým jazykem pro tvorbu Android aplikací je programovací jazyk Java s Android API a použitím sady nástrojů Android SDK [16].

Tyto rozdíly oproti standardnímu vývoji desktopové aplikace v jazycích C++ a Java mají za následek nemožnost spouštět standardním způsobem vytvořené aplikace na systémech Linux a MS Windows. Je zde nutnost dalších aplikací, které slouží k simulaci prostředí Android na těchto systémech. Výjimkou je skriptovací jazyk Python s knihovnou Kivy, kde je dostatečné použití standardního interpreteru s touto knihovnou.

3.1.1 Zvolené řešení

Zvolený jazyk a podporované platformy

Pro implementaci aplikace spike detektoru je zvolen jazyk C++. Hlavními důvody pro toto rozhodnutí je vysoká rychlost[2] a nepoužívání interpreta pro spuštění aplikace. Dalším nezanedbatelným důvodem jsou vyšší programátorské zkušenosti autora této práce s tímto jazykem oproti jazyku Java nebo jazyku Python, se kterým autor nemá žádné předchozí zkušenosti.

Z důvodu použití jazyka C++, problematičtějšího vývoje aplikací pro systém Android v tomto jazyce a jejich používání na jiných systémech, je implementace spike detektoru pro systém Android vynechána. Tedy výsledná aplikace je vytvořena pro systémy s Linuxovým jádrem a systémy MS Windows.

Multiplatformnost

Vývoj multiplatformních aplikací v jazyce C++ přináší některé problémy vycházejících z používání odlišných API pod různými operačními systémy. Hlavními odlišnostmi, na které můžeme narazit v úloze implementace spike detektoru, jsou využívání vícevláknového zpracování některých úloh a odlišné sady nástrojů pro tvorbu grafického uživatelského prostředí (GUI). Cílové platformy, systémy MS Windows a systémy s Linuxovým jádrem, využívají tyto odlišná API:

- Systémy MS Windows využívají Windows API [17].
- Systémy založené na Linuxovém jádru využívají API odpovídající POSIX definicím [18]. U těchto systémů nalezneme rozdíly i v jednotlivých distribucích, kde různé distribuce používají různá grafická prostředí s různými sadami nástrojů pro tvorbu GUI. Například systémy s grafickým prostředí pracovní plochy GNOME [19] používají sadu nástrojů GTK+ [20] nebo systémy založené na grafickém prostředí plochy KDE [21] používají pro vykreslování GUI Qt framework [22].

Zde se nabízí více způsobů řešení, z nichž jsou zde zmíněny dva spolu s některými jejich výhodami a nevýhodami.

Jedním, méně schůdným řešením, je vytvořit více verzí zdrojového kódu v částech, kde dochází k implementačním odlišnostem. Toto řešení je založeno na principu podmíněného překladu, kde dochází k vložení a přeložení té části zdrojového kódu, pro kterou je splněna nějaká podmínka. Tou může být typ operačního systému, pod kterým probíhá překlad aplikace. Tuto informaci můžeme zjistit pomocí předdefinovaných maker překladače, které jsou uvedeny například zde [23].

[2] Zde záleží především na kvalitě implementace daného algoritmu.

Některé výhody a nevýhody tohoto přístupu jsou:

+ Programátor má kontrolu nad implementací jednotlivých částí – myšleny části, které jsou odlišné pro různé systémy.
– Opakování zdrojového kódu.
– Velké množství zdrojového kódu.
– S předchozím bodem souvisí špatná přehlednost ve zdrojovém kódu.
– V případě požadavku na rozšíření aplikace pro další systémy, nutnost rozšíření zdrojového kódu o API vycházející z tohoto systému.
– V případě změny některé funkčnosti nebo opravy chyby, v některé z částí s více verzemi, nutná úprava korespondujících částí ve všech verzích.

Dalším možným řešením je použít některou z cross-platform knihoven, které umožňují implementovat multiplatformní aplikace za použití jednotně vypadajícího zdrojového kódu pro všechny systémy. Mezi výhody a nevýhody tohoto přístupu patří:

+ Psaní jedné verze zdrojového kódu pro všechny podporované platformy.
+ Komplexní sada nástrojů – nástroje pro tvorbu GUI, použití více vláken a podobně.
– Nutnost naučit se používat API dané knihovny.
– Některé knihovny jsou distribuovány pod placenou licencí.

Zvolený framework

V této práci je využito druhé možnosti, tedy je použita C++ knihovna poskytující sadu nástrojů zastiňujících rozdíly mezi platformami. Je zde použita knihovna wxWidgets [24] ve verzi 3.0.2. Hlavní výhody této knihovny jsou:

- vedle C++ podpora více programovacích jazyků jako je Python, Perl, PHP, Java,
- volná licence umožňující komerční použití,
- používání nativních systémových SDK a modulů,
- podpora platforem MS Windows, Mac OS X, Linux a dalších,
- kvalitně zpracovaná dokumentace s mnoha příklady. [25]

Porovnání této knihovny s některými dalšími cross-platform frameworky naleznete na [25].

3.2 Návrh architektury

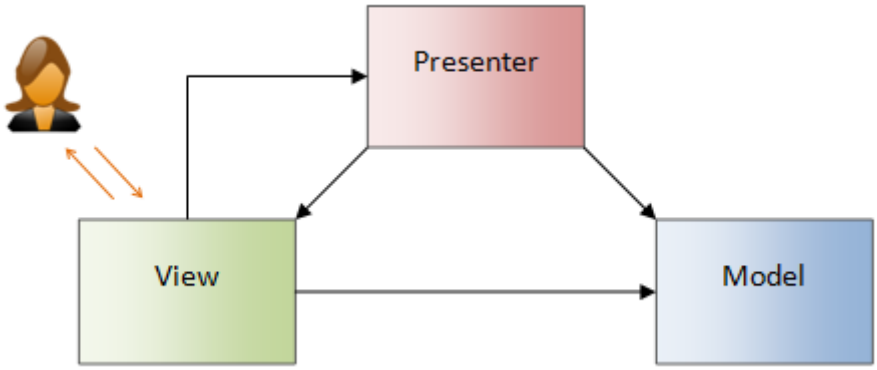
Po zvolení programovacího jazyka a případného frameworku je dalším důležitým krokem zvolení vhodné softwarové architektury. Ta může určovat organizaci softwarových tříd do balíčků, jmenné prostory, uspořádání balíčků do vrstev a podsystémů, a další.

Aplikace spike detektoru je založena na architektonickém vzoru MVP Supervising Controller [26]. Jedná se o třívrstvou architekturu, která dělí aplikaci na tři logické části:

Model
: Vrstva Model reprezentuje data s bussiness logikou.

View
: Vrstva View se stará o zobrazení a obsluhu uživatelského rozhraní, navíc se dále stará o zobrazení dat z modelu.

Presenter
: Vrstva Presenter obsahuje aplikační a prezentační logiku, manipuluje s Modelem a aktualizuje View.



Obrázek 3.1: Schéma architektonického vzoru MVP [2]

Oproti vzoru MVC je zde především odlišnost v kontrole uživatelského vstupu a výstupu. O tu se zde stará vrstva View. [2]

3.3 Doménový model

Koncept doménového modelu vychází ze zvoleného architektonického stylu MVP Supervising Controller. Tedy, třídy jsou rozdělené do tří logických částí a obstarávají pouze tu funkcionalitu, která je očekávaná v dané logické části.

V roli Presenteru zde vystupují třídy:

- *CMain* třída reprezentující samu aplikaci. Definuje, co se provede při spuštění aplikace.
- *CSpikeDetector* jádro aplikace. V této třídě je realizován algoritmus spike detektoru, který je popsán v sekci 1.2.
- *COneChannelDetect* je pomocná třída implementující část funkcionality algoritmu detektoru. Realizuje detekci výbojů v jednom kanálu datového signálu.

Třídy patřící do logické části Model jsou rozděleny na tři skupiny, na skupinu tříd vystupující v roli DAO[3], na skupinu tříd sloužících ke zpracování digitálního signálu a na třídy reprezentující samotná data. Do první skupiny patří:

- *CInputModel* abstraktní třída definující metody a základní implementaci některých metod pro třídy sloužící k načítání iEEG signálu ze souborů.
- *CInputModelEDF* třída založená na abstraktní třídě *CInputModel*. Obstarává načítání vstupních dat ze souborů typu EDF a EDF+.
- *CSettingsModel* třída sloužící k načítání a ukládání nastavení detektoru do dokumentu XML [27]. Výstupem této třídy je struktura obsahující jednotlivá nastavení.
- *CResultsModel* třída sloužící k načítání a ukládání výsledků analýzy do souborů. Více v sekci 3.5.

Do druhé skupiny patří:

- *CDSP* třída poskytující metody pro zpracování digitálního signálu.
- *CResamplingThread* třída realizující převzorkování jednoho kanálu digitálního signálu.
- *CFiltFilt* třída realizující filtr bez posunu fáze.

V roli dat vystupují třídy:

- *CDetectorOutput* třída reprezentující datovou strukturu *out*.
- *CDischarges* třída reprezentující datovou strukturu *discharges*.

[3]Data Access Object – Objekt zapouzdřující přístup k datovému zdroji

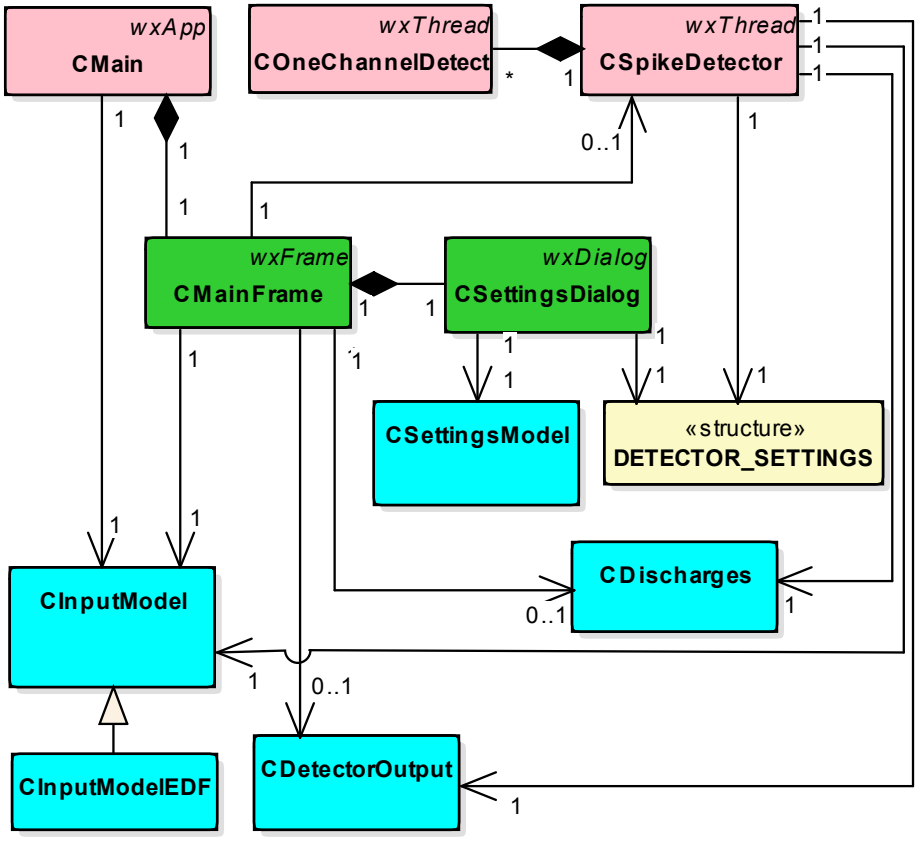
V poslední řadě jsou zde třídy spadající do logické části View:

- *CMainFrame* třída realizující hlavní okno uživatelského rozhraní aplikace.
- *CSettingsDialog* třída realizující dialogové okno pro nastavení detektoru.

Relace mezi třídami

Na následujícím obrázku jsou zobrazeny relace mezi třídami. Model obsahuje pouze názvy tříd. Metody a třídní atributy jsou z důvodu velikosti výsledného modelu vynechány.

Jednotlivé třídy jsou barevně odlišeny dle logických částí vycházejících ze zvolené architektury. Modře znázorněné třídy spadají do vrstvy Model, růžově zabarvené do vrstvy Presenter a zeleně zabarvené do vrstvy View.



Obrázek 3.2: Doménový model – relace mezi třídami

Bližší informace o spolupráci tříd a funkcionalitě se nachází v sekci Implementace a příloze C.

3.4 Návrh uživatelského rozhraní

Jedním z požadavků na implementaci spike detektoru je grafické uživatelské rozhraní umožňující ovládání detektoru, zaprotokolování a vizualizaci výsledků analýzy.

Pro tento účel jsou navrženy tři hlavní grafické elementy, přes které je možné zmíněné požadavky realizovat.

- Prvním je hlavní okno aplikace, které je zobrazeno při spuštění aplikace. V tomto okně se nachází ovládací prvky a oblast pro zobrazení výsledků analýzy. Ty jsou vypsány do dvou oddělených tabulek.
- Druhým prvek je dialogové okno, přes které je možné manipulovat s nastavením detektoru
- Posledním prvkem je dialogové okno zobrazující postup analýzy. Pomocí něj lze proces analýzy zastavit.

Ostatní grafické části jako jsou dialogová okna pro otevření a uložení souboru vychází z výchozího vzhledu pro danou platformu.

3.4.1 Hlavní okno aplikace

V hlavním okně aplikace se nachází lišta obsahující menu pro ovládání detektoru, panel nástrojů obsahující prvky menu zobrazené pomocí ikon a dvě tabulky pro zobrazení výsledků analýzy. Vzhled tohoto okna je navržen na obrázku 3.4.

Lišta menu je rozdělena do tří submenu s těmito položkami:

- File
  1. Open file – zobrazí okno pro otevření souboru s daty k analýze.
  2. Load results – zobrazí okno pro otevření souboru s uloženými výsledky analýzy.
  3. Save results – zobrazí okno pro uložení výsledků do souboru. Tato položka je přístupná jen pokud jsou v aplikaci zavedené některé výsledky analýzy.
  4. Quit – ukončí aplikaci spike detektoru.
- Detector
  1. Run – spustí analýzu. Je přístupné pouze v případě, že je načten soubor s vstupními daty k analýze.
  2. Options – zobrazí dialogové okno pro nastavení detektoru.
- Help
  1. About – zobrazí informace o aplikaci.

3.4.2 Dialogové okno nastavení

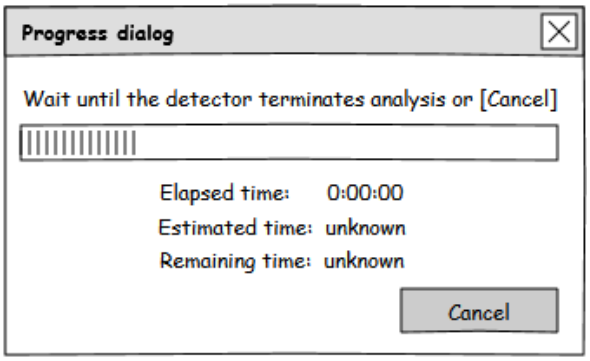
V tomto okně je možné změnit nastavení analýzy. Vzhled tohoto okna je navržen na obrázku 3.5. V tomto okně se nachází tyto tlačítka:

1. Load default – tlačítko pro načtení výchozího nastavení ze souboru.
2. Load saved – tlačítko pro načtení uloženého nastavení ze souboru.
3. Save – tlačítko pro uložení aktuálního nastavení. Tato operace přepíše předchozí uložené hodnoty.
4. Use – tlačítko pro nastavení vyplněných hodnot. Bez potvrzení změněných hodnot nebudou tyto hodnoty použity.

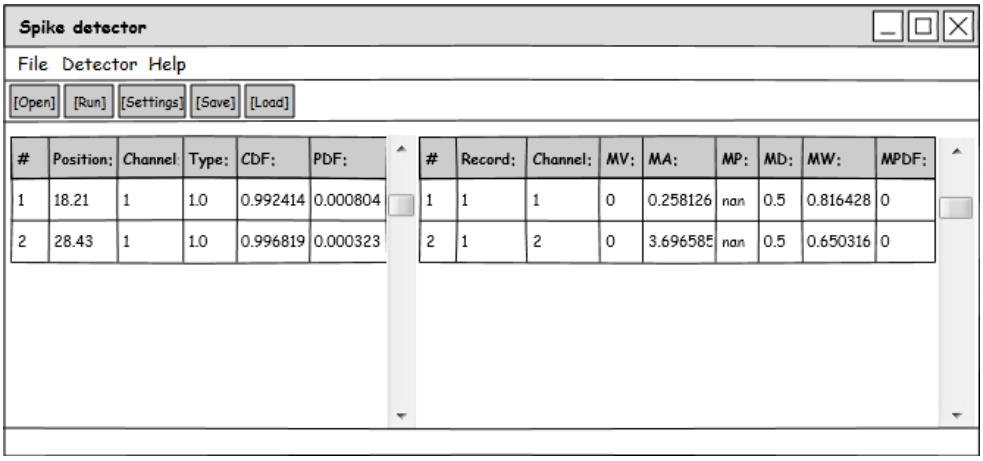
V případě nekorektně vyplněné hodnoty nastavení je vstupní pole označeno červenou barvou a je zobrazeno varování. V tomto případě nejsou nastavené hodnoty použity dokud nedojde k vložení korektních hodnot.

3.4.3 Dialogové okno s postupem analýzy

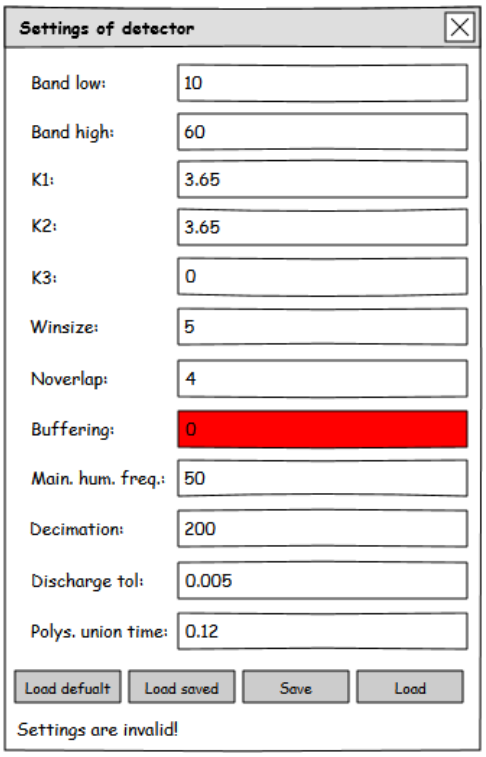
Dialogové okno s ukazatelem postupu (progress bar) je zobrazeno při spuštění analýzy. Dále jsou zde uvedeny informace o uplynulém čase, odhadovaný čas a zbývající čas analýzy. Také se zde nachází tlačítko pro přerušení analýzy.



Obrázek 3.3: Návrh GUI – dialogové okno průběhu analýzy



Obrázek 3.4: Návrh GUI – hlavní okno



Obrázek 3.5: Návrh GUI – dialogové okno nastavení

3.5 Forma ukládání výsledků

Výsledky detektoru, struktury *discharges* a *out*, je možné uložit a zpětně načíst ze dvou podporovaných datových formátů.

Prvním je binární datový formát MAT [28], který je výchozím formátem pro ukládání proměnných do souboru v prostředí MATLAB. Struktury jsou zde zachovány ve stejném formátu jako tomu je v referenční implementaci.

Druhou možností je použít dokumenty XML [27]. Tyto dokumenty mají definovanou následující strukturu.

Výsledky jsou uloženy mezi kořenovým elementem *spikedetektor*. Dále:

- Záznamy struktury *out* jsou uloženy mezi elementem *Output*, kde je každý záznam výboje uložen v elementu *Position* s následujícími atributy:
  - *pos* – pozice výboje v kanálu,
  - *chan* – doba trvání výboje,
  - *dur* – kanál, ve kterém výboj vznikl,
  - *con* – typ výboje,
  - *weight* – statistická významnost "CDF",
  - *pdf* – statistická významnost "PDF".
- Záznamy vícekanálových událostí jsou uloženy mezi elementem *Discharges*. Zde, protože se jedná o dvourozměrné matice typu $T \times CH$, kde T je počet záznamů a CH počet kanálů, jsou jednotlivé záznamy rozděleny mezi elementy *Channel* s atributem *number* značící číslo kanálu, kterému přísluší vnořené záznamy. Záznam je reprezentován elementem *Disch* s následujícími atributy:
  - *number* – číslo záznamu v daném kanálu
  - *MV* – identifikátor typu výboje,
  - *MA* – maximální amplituda obálky nad pozadím,
  - *MP* – počáteční pozice události,
  - *MD* – doba trvání události,
  - *MW* – hodnota kumulativní distribuční funkce výboje z lognormálního modelu,
  - *MPDF* – hodnota hustoty pravděpodobnosti.

Příklad takovéhoto dokumentu:

```
<?xml version="1.0" ?>
<spikedetector>
    <Output>
        ...
        <Position
            pos="75.395" chan="1" dur="0.005" con="1"
            weight="0.999743" pdf="2.52558e-005" />
        <Position
            pos="77.71" chan="1" dur="0.005" con="1"
            weight="0.998042" pdf="0.000242417" />
        ...
    </Output>
    <Discharges>
        <Channel number="0">
            <Disch
                number="0" MV="0" MA="0.258126" MP="nan"
                MD="0.005" MW="0.816428" MPDF="0" />
            <Disch
                number="1" MV="0" MA="8.19371" MP="nan"
                MD="0.005" MW="0.962851" MPDF="0" />
            ...
        </Channel>
        ...
    </Discharges>
</spikedetector>
```

Více informací o realizaci ukládání a načítání výsledků se dozvíte v části Implementace.

3.6 Forma nastavení detektoru

Výsledky analýzy vstupního signálu jsou ovlivňovány daným nastavením detektoru. Možná nastavení s jejich výchozími hodnotami jsou popsána v sekci 1.2.1.

Možnost měnit nastavení, ukládat přednastavené hodnoty a zpětně načíst výchozí hodnoty v roli uživatele je řešena pomocí dialogového okna.

Hodnoty daných nastavení jsou uloženy v dokumentu XML s názvem „settings.xml“, který se nachází ve stejné složce jako aplikace spike detektoru. V případě, že tento dokument není nalezen, je použito výchozí nastavení.

Struktura toho dokumentu je následující:

```
<?xml version="1.0" ?>
<spikedetector>
    <!-- DEFAULT SETTINGS -->
    <settings type="1">
      <band_low>10</band_low>
      <band_high>60</band_high>
      <k1>3.65</k1>
      <k2>3.65</k2>
      <k3>0</k3>
      <winsize>5</winsize>
      <noverlap>4</noverlap>
      <buffering>300</buffering>
      <main_hum_freq>50</main_hum_freq>
      <discharge_tol>0.005</discharge_tol>
      <polyspike_union_time>0.12</polyspike_union_time>
      <decimation>200</decimation>
    </settings>
    <!-- SAVED SETTINGS -->
    <settings type="2">
      <band_low>10</band_low>
      <band_high>60</band_high>
      <k1>3.65</k1>
      <k2>3.65</k2>
      <k3>0</k3>
      <winsize>5</winsize>
      <noverlap>4</noverlap>
      <buffering>300</buffering>
      <main_hum_freq>50</main_hum_freq>
      <discharge_tol>0.005</discharge_tol>
      <polyspike_union_time>0.12</polyspike_union_time>
      <decimation>200</decimation>
    </settings>
</spikedetector>
```

Atribut *type* u elementu *settings*, udává o jaké nastavení se jedná, možné hodnoty jsou hodnota 1 udávající výchozí nastavení detektoru a hodnota 2 značící nastavení uložené uživatelem.

Odlišnosti oproti implementaci v prostředí MATLAB jsou:

- Elementy *band_low* a *band_high* reprezentují hodnotu *bandwidth*, tedy spodní a horní frekvenci filtrování.
- Je vynechána možnost zvolení typu filtru – ten je pevně nastaven.

Kapitola 4

Implementace

4.1 Vývojové prostředí

Implementace spike detektoru je vyvinuta a testována na 64bitových operačních systémech MS Windows 7 Home Premium (dále jen Windows 7) a Ubuntu 14.04 (dále jen Ubuntu). Na obou systémech je použito pro účely analýzy a zhodnocení výsledků prostředí MATLAB R2014a verze 8.3.0.532 pro 64bitové systémy.

Projekt aplikace je veden tak, aby nebyla nutnost použití IDE[4]. Pro úpravu a sestavení výsledné aplikace si uživatel vystačí pouze s textovým editorem, překladačem a nástrojem pro automatizaci překladů. Je zde pouze požadavek na překladač s podporou standardu C++11. Další nástroje pro použité systémy jsou:

MS Windows 7 Home Premium

Na tomto systému je kompilace realizována pomocí nástroje MSYS [29] ze sady nástrojů MinGW [30]. Odpovídající Makefile[5], určený pro sestavení aplikace spike detektoru pod systémem MS Windows, je určen pro použití sady MinGW.

V případě použití jiného nástroje je nutné tento Makefile patřičně upravit. Více informací o sestavení aplikace naleznete v příloze B.

[4]Integrated Development Environment – vývojové prostředí v rámci jednoho software.

[5]Soubor definující postup překladu a závislosti mezi zdrojovými soubory pro nástroj GNU Make.

Nástroje použité na systému Windows 7:

Název	Verze	Popis
MinGW x86	1.0.18	Sada nástrojů poskytující podobné funkce jako Linuxové distribuce pod systémy Windows.
GCC	4.8.1	Sada překladačů.
GNU Make	3.81	Nástroj pro automatizaci překladů [31].
Dr. Memory	1.8.1	Nástroj pro ladění chyb v paměti [32].

Tabulka 4.1: Nástroje použité na systému MS Windows 7

Ubuntu 14.04

Na systému Ubuntu jsou jednotlivé nástroje spouštěny pomocí terminálu, těmito nástroji jsou:

Název	Verze	Popis
GCC	4.8.2	Sada překladačů.
GNU Make	3.81	Nástroj pro automatizaci překladů.
Valgrind	3.10.0	Sada analytických nástrojů [33].

Tabulka 4.2: Nástroje použité na systému Ubuntu 14.04

4.2 Realizace tříd

Jak je uvedeno v Návrhu architektury, aplikace je složena z několika logických částí – vrstev. Každá vrstva se skládá z několika tříd, které poskytují danou funkcionalitu.

V následujícím textu je implementace rozdělena do těchto částí:

- Implementace samotné aplikace a GUI.
- Implementace algoritmu spike detektoru – jádro aplikace.
- Implementace tříd poskytujících zpracování digitálního signálu.
- Implementace tříd pro práci se soubory.
- Reprezentace dat.
- Ošetření chyb a nestandardních stavů.

Toto rozdělení částečně odpovídá rozdělení dle zvolené architektury, však některé části se mohou prolínat.

Jednotlivé třídy jsou zde stručně popsány. Podrobnější popis zahrnující důležité metody těchto tříd naleznete v příloze C a ve vygenerované dokumentaci ze zdrojového kódu. Pro generování dokumentace je použit nástroj Doxygen [34].

4.2.1 Implementace aplikace a GUI

Implementace aplikace

Samotná aplikace je reprezentována třídou *CMain*. Tato třída dědí od *wxApp* [35] z knihovny wxWidgets. To ji umožňuje definovat operace, které se provedou při inicializaci aplikace a při jejím ukončení.

Při inicializaci aplikace je vytvořena instance třída pro přístup k vstupním datům analýzy vycházející z abstraktní třídy *CInputModel*. V případě této implementace je zde inicializována třída *CInputModelEDF*, která je odvozená od *CInputModel*.

Dále je vytvořena instance hlavního okna aplikace, třídy *CMainFrame*. Toto okno je posléze zobrazeno na pracovní plochu.

Při ukončení aplikace dochází k uvolnění alokované paměti.

Implementace GUI

GUI aplikace spike detektoru realizují třídy *CMainFrame* a *CSettingsDialog*. Obě tyto třídy jsou inicializovány při startu aplikace.

Třída *CMainFrame*, reprezentující hlavní okno aplikace, dědí od třídy *wxFrame* [36], která zastiňuje všechnu potřebnou funkcionalitu pro práci s grafickým objektem okna pod danou platformou.

Tato třída v sobě obsahuje nebo vytváří všechny grafické objekty. Jedná se jak o objekty pro zobrazení informativní zprávy nebo objekty pro uživatelskou interakci (menu, panel nástrojů, dialogové okno s postupem analýzy). Jedním z těchto objektů je instance třídy *CSettingsDialog*.

Dalším úkolem této třídy je obsluha uživatelských akcí a reakce na průběh analýzy, to je vyvoláno přijetím událostí typu *wxThreadEvent* vyvolané třídou *CSpikeDetector*. Ty mohou obsahovat zprávu o postupu analýzy, dokončení analýzy nebo o chybě. Dle zprávy je buďto aktualizován zobrazený postup analýzy, nebo zobrazeny výsledky analýzy, nebo zobrazeno chybové hlášení.

Třída *CSettingsDialog* reprezentuje dialogové okno pro obsluhu nastavení detektoru. Tato třída dědí od třídy *wxDialog* [37].

Vedle grafických prvků (tlačítka, pole pro vyplnění hodnot) se v třídě *CSettingsDialog* nachází instanci třídy *CSettingsModel*, která obstarává ukládání a načítání nastavení ze souboru.

Podobně jako *CMainFrame* implementuje metody pro obsluhu uživatelských požadavků, které jsou představeny tlačítky v zobrazeném dialogovém okně.

Výsledný vzhled aplikace spike detektoru, realizovaný těmito třídami, naleznete v sekci 4.4.

4.2.2 Implementace jádra aplikace

Jádrem aplikace je samotná implementace algoritmu spike detektoru popsaného v sekci Analýza. Implementace tohoto algoritmu je navržena tak, aby byla nezávislá na GUI a samotné aplikaci, která tento algoritmus spouští. Samotný algoritmus je realizován ve třídách *CSpikeDetector* a *COneChannelDetect*. Průběh analýzy po spuštění uživatelem je zobrazen v sekvenčním diagramu 4.1.

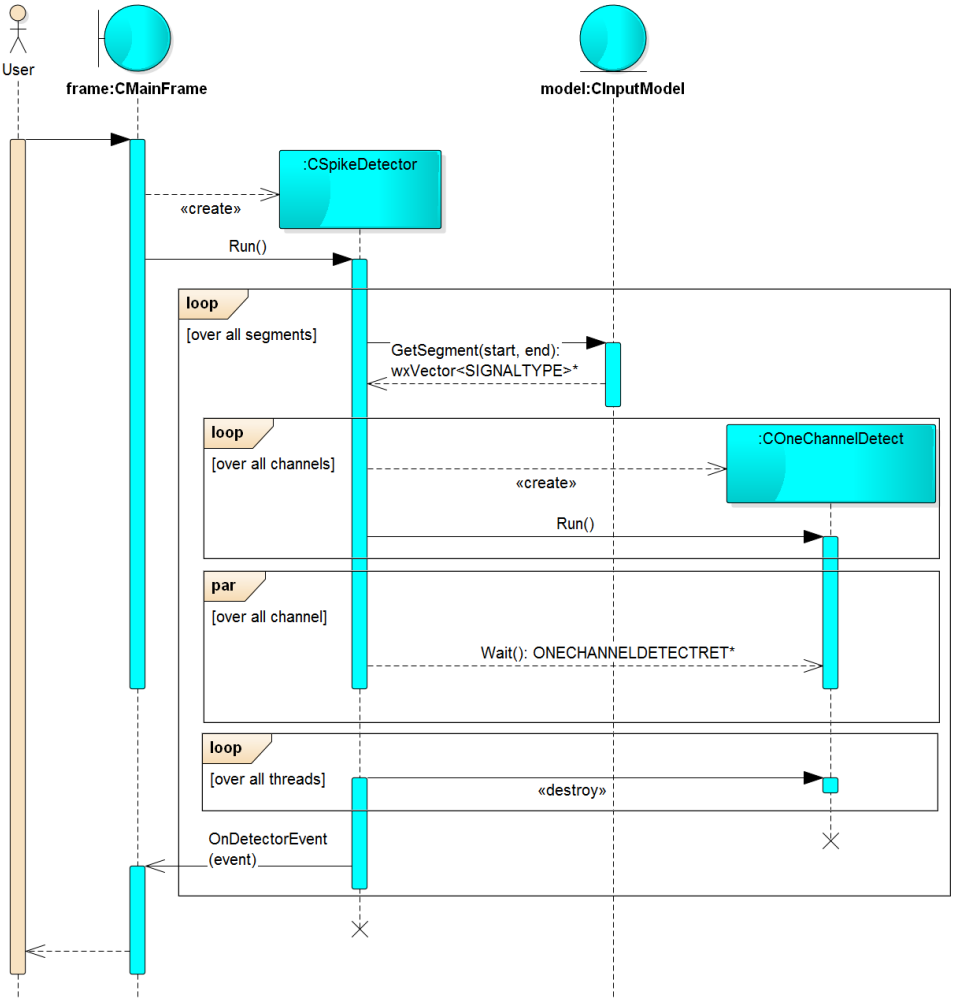
Důvodem pro rozdělení implementace do dvou tříd je běh procesu detekce jednokanálových událostí na novém vlákně pro každý kanál analyzovaných dat. Tento proces je reprezentován třídou *COneChannelDetect*.

Obě třídy *CSpikeDetector* a *COneChannelDetect* dědí od třídy *wxThread* [38]. To umožňuje pomocí instancí těchto tříd vytvořit nová pracovní programová vlákna. Ty dále vykonávají další zpracování.

Knihovna wxWidgets umožňuje vytvářet dva různé typy vláken *detached*[6] a *joinable*. Rozdíl mezi těmito typy je ve způsobu ukončení práce vlákna. Vlákna typu *detached* jsou po skončení práce automaticky odstraněna ze systému. Oproti tomu, status vlákna typu *joinable* je udržován, dokud není toto vlákno vyzvednuto z jiného vlákna. To například umožňuje při vyzvednutí ukončeného vlákna získat výstupní hodnotu z tohoto vlákna pomocí návratového parametru.

Analýza je spouštěna na samostatném vlákně oproti vláknu, na kterém běží sama aplikace a GUI. Vstupním bodem analýzy je třída *CSpikeDetector*, která vytváří vlákno typu *detached*. Oproti tomu třída *COneChannelDetect* vytváří vlákno typu *joinable*.

[6] Tento typ vláken je definován v POSIX API, Win32 API definuje všechna vlákna jako *joinable*. Tento rozdíl je eliminován knihovnou wxWidgets, která umožňuje použít typ *detached* i pod systémy MS Windows.



Obrázek 4.1: Diagram průběhu analýzy

Stručný průběh analýzy:

1. Výpočet segmentace vstupních dat.
2. Postupné načítání segmentů vstupních dat (iEEG signál) pomocí metody *GetSegment* instance třídy *CInputModelEDF*.
3. Každý segment je dále analyzován v metodě *spikeDetector*:
   a) V případě vyšší vzorkovací frekvence vstupních dat než je požadovaná, dochází k jejich převzorkování. To realizuje statická metoda *Resample* třídy *CDSP*.
   b) Dále je provedeno filtrování síťového šumu a filtrování v nastaveném pásmu. Tyto operace jsou realizovány statickými metodami *Filt50Hz* a *Filtering* třídy *CDSP*.
   c) Po fázích filtrování dochází k detekci jednokanálových událostí. Ta je provedena pro každý kanál vstupního signálu zvlášť.
      Samu detekci realizuje třída *COneChannelDetect*. Zde je pro každý kanál zpracovávaného signálu vytvořena vlastní instance této třídy.
   d) Na závěr této metody jsou v označených úsecích lokalizovány přesné pozice výbojů.
4. Po analýze jednoho segmentu vstupních dat jsou odstraněny oboustranné překrývající se detekce. A zbývající detekce jsou zaznamenány do objektů tříd *CDetectorOutput* a *CDischarges*.

Stručný průběh algoritmu detekce jednokanálových událostí:

1. Výpočet okamžité obálky pomocí absolutní hodnoty Hilbertovy transformace.
   To je provedeno pomocí statické metody *AbsHilbert* třídy *CDSP*.
2. Výpočet odhadů parametrů lognormální distribuce pro každý segment vstupních dat.
3. Vyhlazení odhadnuté řady parametrů filtrem klouzavých průměrů.
   Ten jen reprezentován metodou *FiltFilt* třídy *CDSP*.
4. Interpolace parametrů modelu na vzorkovací kmitočet originálního signálu.
   Pro interpolaci je zde použita Spline interpolace z knihovny ALGLIB [39]. Přesněji funkce *spline1dconvcubic* [40].
5. Výpočet prahovacích křivek.

6. Označení oblastí s výskytem výboje.

   Tyto oblasti jsou vyhledávány a označovány v metodě *localMaximaDetection*.

7. Na závěr jsou detekovány vícenásobné detekce se separací menší než *polyspike_union_time*, které jsou následně sloučeny do jediné.

   Toto je realizováno metodou *detectionUnion*. Zde dochází k výpočtu dilatace a eroze, pročež je použit výpočet diskrétní konvoluce. Ten je realizován pomocí funkce *convr1d* [41] z knihovny ALGLIB.

4.2.3 Implementace zpracování digitálního signálu

V algoritmu spike detektoru je používáno několika operací pro zpracování digitálního signálu. Jedná se o proces převzorkování signálu, filtrování signálu a výpočet absolutní hodnoty Hilbertovy transformace.

Tato funkcionalita je implementována ve statické třídě *CDSP*, která dále využívá pomocné třídy *CResamplingThread* a *CFiltFilt*. Operace tříd *CResamplingThread* a *CFiltFilt* jsou prováděny na nově vytvořených vláknech, jako tomu je u třídy *COneChannelDetect*. Tyto vlákna jsou typu *joinable*.

Třída *CResamplingThread* slouží k převzorkování jednoho kanálu digitálního signálu. Toho je využíváno v metodě *Resampling* třídy *CDSP*, která obstarává decimaci analyzovaného signálu. Pro převzorkování signálu je použita knihovna libsamplerate [42].

Třída *CFiltFilt* představuje implementaci digitálního filtru bez posunu fáze. Implementace tohoto filtru je převzata z [43]. V této implementaci je využita externí knihovna Eigen [44] pro maticové násobení a inverzní operace.

Implementace filtru využívá lambda funkcí a datového typu *auto*. Pro to je nutné použít kompilátor s podporou standardu C++11.

4.2.4 Implementace práce se soubory

V této podsekci se nachází třídy pro práci se soubory. Tyto třídy pouze zastiňují práci s daty na úrovni Modelu, kde jsou data načítána ze souborů nebo naopak do souborů ukládána.

Tyto třídy jsou rozděleny do tří kategorií dle druhu dat, se kterými pracují.

4.2.4.1 Vstupní data analýzy

Po zvolení souboru s daty, která jsou určená pro analýze, je cesta k tomuto souboru předána instanci třídy zpracovávající tento soubor. Tato třída tento soubor otevře a načte základní informace pro analýzu. Dále nabízí rozhraní pro přístup k iEEG signálu.

Pro tuto funkčnost jsou definovány předpisy metod v abstraktní třídě *CInputModel*. Ty dále implementuje třída *CInputModelEDF*, která zpracovává soubory typu EDF a EDF+.

Pro práci s těmito datovými typy je použita knihovna EDFLib [45]. Zde je pro čtení vzorků signálu použita funkce *edfread_physical_samples*, která převede záznamy na jejich fyzikální veličiny. Ty jsou reprezentovány datovým typem *float*.

4.2.4.2 Výsledky analýzy

Výsledky analýzy, struktury *out* a *discharges*, je možné uložit a zpětně načíst z binárního datového formátu MAT nebo z dokumentu XML. Struktura dokumentu XML je popsána v 3.5.

Pro práci se soubory obsahující výsledky analýzy je zde statická třída *CResultsModel*. Ta pro práci se souborovým typem MAT používá knihovnu MAT File I/O Library [46] a pro práci s dokumenty XML knihovnu TinyXML [47].

4.2.4.3 Nastavení detektoru

Nastavení analýzy je uloženo v dokumentu XML popsaného v 3.6. Ten je zpracováván třídou *CSettingsModel*, která čte nastavení uložená v tomto dokumentu nebo ukládá uživatelem zadané hodnoty.

Pro práci s tímto dokumentem je opět použita knihovna TinyXML.

4.2.5 Reprezentace dat

Výsledky analýzy jsou reprezentovány třídami *CDetectorOutpu* a *CDischarges*. Ty drží výsledky v podobné struktuře jako tomu je v implementaci v prostředí MATLAB.

4.2.6 Ošetření chyb a nestandardních stavů

V průběhu životního cyklu aplikace může dojít k chybě nebo dojde k situaci, kdy nelze pokračovat v určité činnosti. Tím může být například:

- chyba při otevírání souboru s daty,
- neočekávaná chyba v průběhu detekce,
- volba souboru nepodporovaného datového typu.

Pro tyto situace je zde implementována třída *CException*, která dědí od třídy *exception* ze standardního balíku knihoven. Tedy tato třída reprezentuje výjimku vyvolanou při vzniku některé chyby.

Při zachycení vyvolané výjimky dochází k dvěma možnostem:

1. Výjimka byla vyvolána v průběhu analýzy nebo zpracování dat v procesu analýzy.

   V případě, že je výjimka vyvolána ve vláknu sloužícímu pro analýzu vstupních dat nebo některém vláknu zpracovávajícího digitální signál (třídy *CFiltFilt*, *CResamplingThread*). Je tato výjimka propuštěna až do vstupní metody *Entry* třídy *CSpikeDetector*. Zde je tato výjimka zachycena – jedná se o zachycení jakékoliv výjimky vycházející ze třídy *exception*.

   Po zachycení výjimky je přerušen proces analýzy a je zaslána událost, obsahující chybu, instanci třídy, která byla nastavena pro zachytávání zpráv z implementace algoritmu – v této aplikaci třída *CMainFrame*. Ta po přijetí této události převede aplikaci do výchozího stavu (metoda *clear*) a zobrazí chybovou zprávu uživateli.

2. Výjimka je vyvolána v hlavním vlákně aplikace. Zde se jedná o výjimky vyvolané při obsluze uživatelských požadavků mimo analýzu.

   V této situaci je výjimka zachycena a zpracována přímo třídou *CMainFrame*. Zde dojde k ošetření výjimky a zobrazení chybové zprávy uživateli pomocí logovací funkce *wxLogError* [48].

4.3 Použité knihovny

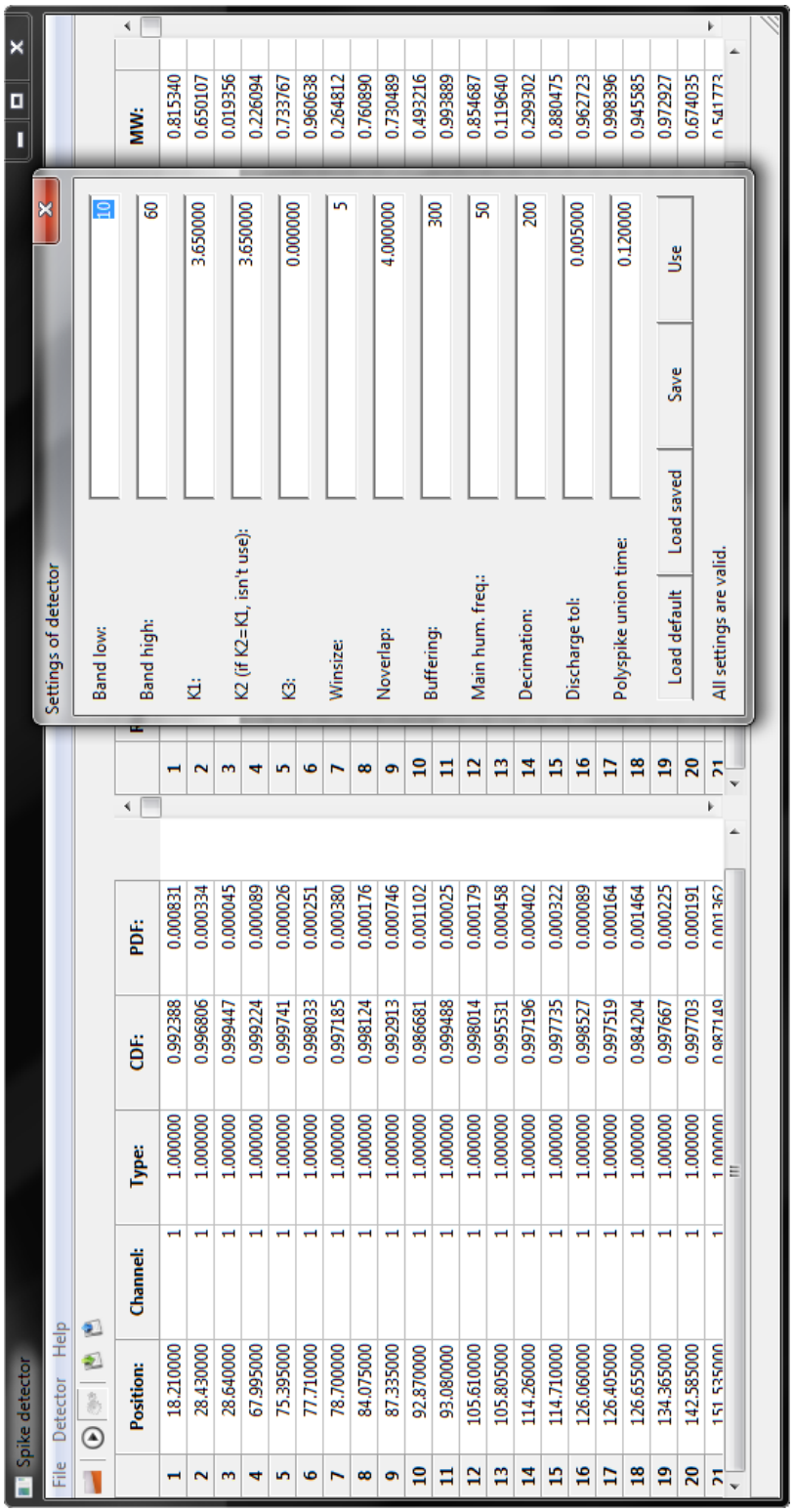
V této implementaci je využito vedle standardních knihoven a knihovny wxWidgets několika dalších externích knihoven. Jejich rekapitulace s verzí dané knihovny je uvedena v následující tabulce:

Knihovna	**Verze**	**Licence**
ALGLIB	3.9.0	GPL 2+ [49]
libsamplerate	0.1.8	GPL [50]
Eigen	3.2.4	MPL2 [51]
MATio	1.5.2	Open source
TinyXML	2.6.2	zlib/libpng License [52]
EDFlib	1.11	BSD [53]

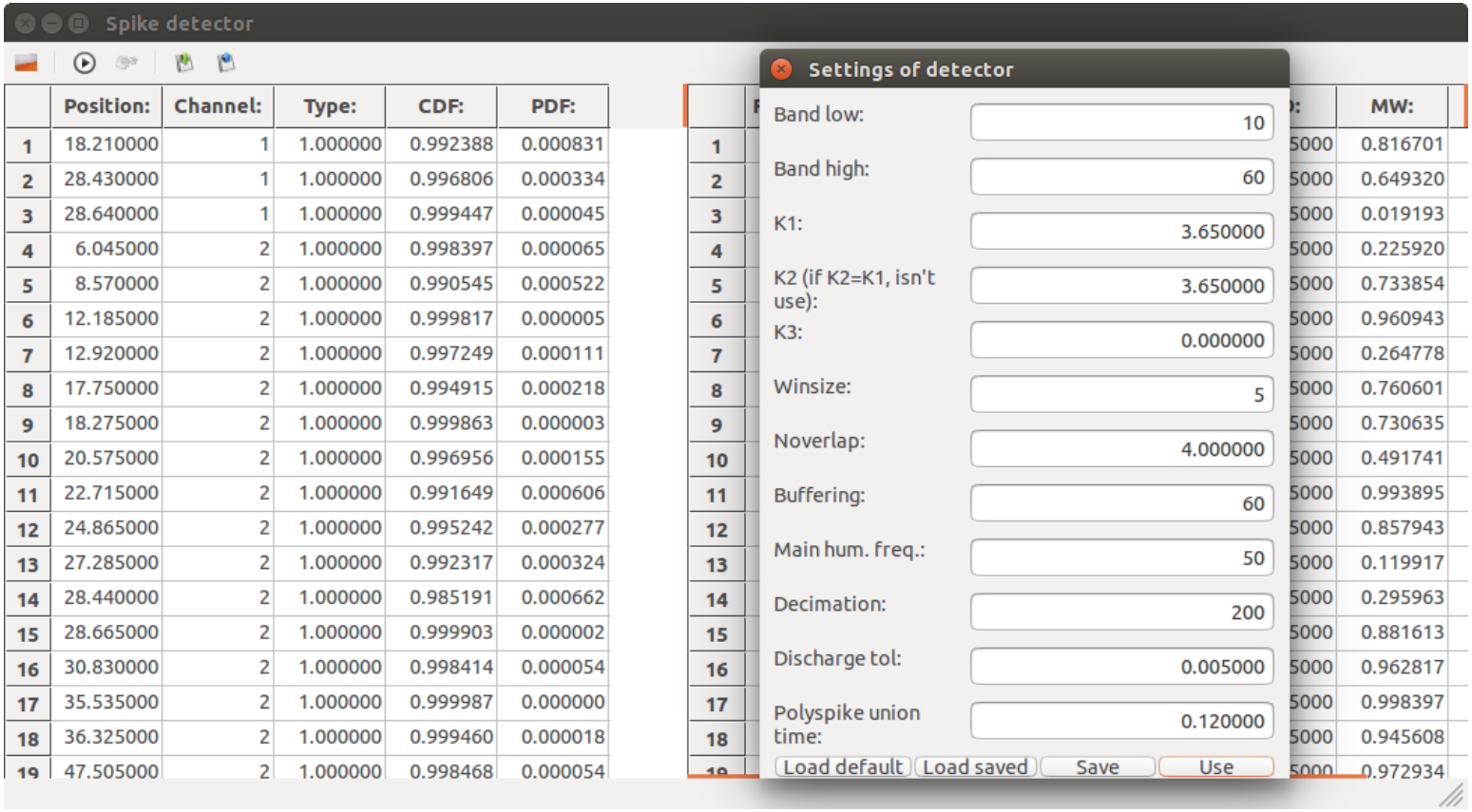
Tabulka 4.3: Externí knihovny využité v implementaci

4.4 Vzhled aplikace

Výsledný vzhled aplikace je zobrazen na obrázku 4.2, pro systém MS Windows 7, a na obrázku 4.3, pro systém Ubuntu 14.04.



Obrázek 4.2: Zobrazení aplikace na systému MS Windows 7



	Position:	Channel:	Type:	CDF:	PDF:
1	18.210000	1	1.000000	0.992388	0.000831
2	28.430000	1	1.000000	0.996806	0.000334
3	28.640000	1	1.000000	0.999447	0.000045
4	6.045000	2	1.000000	0.998397	0.000065
5	8.570000	2	1.000000	0.990545	0.000522
6	12.185000	2	1.000000	0.999817	0.000005
7	12.920000	2	1.000000	0.997249	0.000111
8	17.750000	2	1.000000	0.994915	0.000218
9	18.275000	2	1.000000	0.999863	0.000003
10	20.575000	2	1.000000	0.996956	0.000155
11	22.715000	2	1.000000	0.991649	0.000606
12	24.865000	2	1.000000	0.995242	0.000277
13	27.285000	2	1.000000	0.992317	0.000324
14	28.440000	2	1.000000	0.985191	0.000662
15	28.665000	2	1.000000	0.999903	0.000002
16	30.830000	2	1.000000	0.998414	0.000054
17	35.535000	2	1.000000	0.999987	0.000000
18	36.325000	2	1.000000	0.999460	0.000018
19	47.505000	2	1.000000	0.998468	0.000054

	MW:
1	0.816701
2	0.649320
3	0.019193
4	0.225920
5	0.733854
6	0.960943
7	0.264778
8	0.760601
9	0.730635
10	0.491741
11	0.993895
12	0.857943
13	0.119917
14	0.295963
15	0.881613
16	0.962817
17	0.998397
18	0.945608
19	0.972934

Obrázek 4.3: Zobrazení aplikace na systému Ubuntu 14.04

Kapitola 5

Testování

Výsledná aplikace je otestována na splnění hlavních dvou požadavků stanovených pro novou implementaci.

Nejdůležitějším požadavkem je bezesporu shoda výsledků analýzy s referenční implementací algoritmu v prostředí MATLAB verze 20. Zde je povolena maximálně 1% chybovost.

Druhým bodem je optimalizace, zde jsou rozebrány paměťové nároky aplikace a dále je porovnána doba trvání analýzy.

5.1 Testovací data

Pro účely testování je použito sedm souborů s vstupními daty pro analýzu. Tyto soubory jsou dostupné z adresy [6] nebo jsou přiloženy na CD této práce.

Tyto soubory jsou dále převedeny z binárního datového formátu MAT do formátu EDF+.

Každý ze souborů obsahuje pět minut záznamu iEEG signálu a další potřebné informace k analýze. Charakteristika vstupních dat obsažených v těchto souborech:

Soubor	FS [Hz]	Počet kanálů	Počet vzorků kanálu
IED_P001	250	15	75000
IED_P002	1000	15	300000
IED_P003	200	15	60000
IED_P004	200	15	60000
IED_P005	200	15	60000
IED_P006	1000	15	300000
IED_P007	200	15	60000

Tabulka 5.1: Charakteristika testovacích dat

5.2 Porovnání výsledků analýzy

Vzájemná shoda pozic detekovaných výbojů je vyjádřena v tabulce 1.1. Ta obsahuje tyto údaje:

- TP – počet detekcí odpovídajících referenční implementaci,
- FP – počet falešných detekcí,
- FN – počet přehlédnutých detekcí,
- PPV – prevalence, míra správných detekcí v procentech, vypočtena ze vztahu: $PPV = \frac{TN}{TN+FN} \cdot 100$
- SEN – senzitivita, míra skutečně pozitivních detekcí v procentech, vyjádřena vztahem: $SEN = \frac{TP}{TP+FP} \cdot 100$

Soubor	**TP**	**FP**	**FN**	**PPV** [%]	**SEN** [%]
IED_P001	894	0	0	100	100
IED_P002	497	0	0	100	100
IED_P003	382	0	0	100	100
IED_P004	22	0	0	100	100
IED_P005	1367	0	0	100	100
IED_P006	648	0	0	100	100
IED_P007	439	0	0	100	100

Tabulka 5.2: Porovnání výsledků pro výchozí nastavení detektoru

Z těchto výsledků lze pozorovat, že pro použitá vstupní data a výchozí nastavení analýzy jsou detekované pozice shodné s referenční implementací.

Toho bylo dosaženo až v průběhu testování. Původní testovaná implementace obsahovala FN v řádu jednotek pro některé testovací sady. Během zkoumání vzniku těchto chyb byla nalezena chyba ve výpočtu variance. Ta byla následně opravena.

5.3 Paměťové nároky

Paměťové nároky aplikace jsou závislé na charakteristice vstupních dat pro analýzu a daném nastavení detektoru. Nejvýraznějším parametrem nastavení ovlivňujícím paměťové nároky je hodnota *buffering*, která udává velikost načteného segmentu v sekundách pro každý kanál vstupního signálu.

Velikost segmentu je dána vztahem:

$$n_1 = ch \cdot (fs \cdot buffering + 2 \cdot 3 \cdot winsize), \tag{5.1}$$

kde

- n_1 je celkový počet načtených vzorků,
- ch je počet kanálů vstupního signálu,
- fs je vzorkovací frekvence vstupního signálu,
- *buffering* délka načteného záznamu,
- *winsize* velikost klouzavého okénka[7].

V případě segmentace je první a poslední segment zkrácen o velikost překrytí $3 \cdot winsize$. Dále, v případě kratší velikosti posledního segmentu, je tento segment spojen s předchozím. Datový typ jednoho vzorku signálu je v této implementaci *float*.

Dalším elementem, který může výrazněji ovlivnit paměťovou náročnost, je počet zachycených detekcí. Zde, v případě mnoha vícekanálových událostí, je počet všech záznamů vlastností dán vztahem:

$$n_d = d \cdot ch \cdot 6, \tag{5.2}$$

kde

- n_d označuje celkový počet uložených záznamů pro vícekanálové události,
- d je počet vícekanálových událostí.

Zde je každý záznam reprezentován datovým typem *double*. Tento počet také ovlivňuje paměťové nároky na zobrazení těchto záznamů v okně aplikace po procesu analýzy.

Paměťová asymptotická složitost procesu analýzy je lineární $O(n)$, to je určeno z přesnějšího vyjádření:

$$2 \cdot n_1 + 13 \cdot n_2 + O(1), \tag{5.3}$$

kde n_2[8] je počet vzorků signálu po převzorkování.

[7] Výchozí nastavení $winsize = 5 \cdot fs$.

[8] $n_2 = n_1$, nebo $n_2 < n_1$ pro vyšší vzorkovací frekvenci vstupních dat oproti pracovní frekvenci.

Měřená data

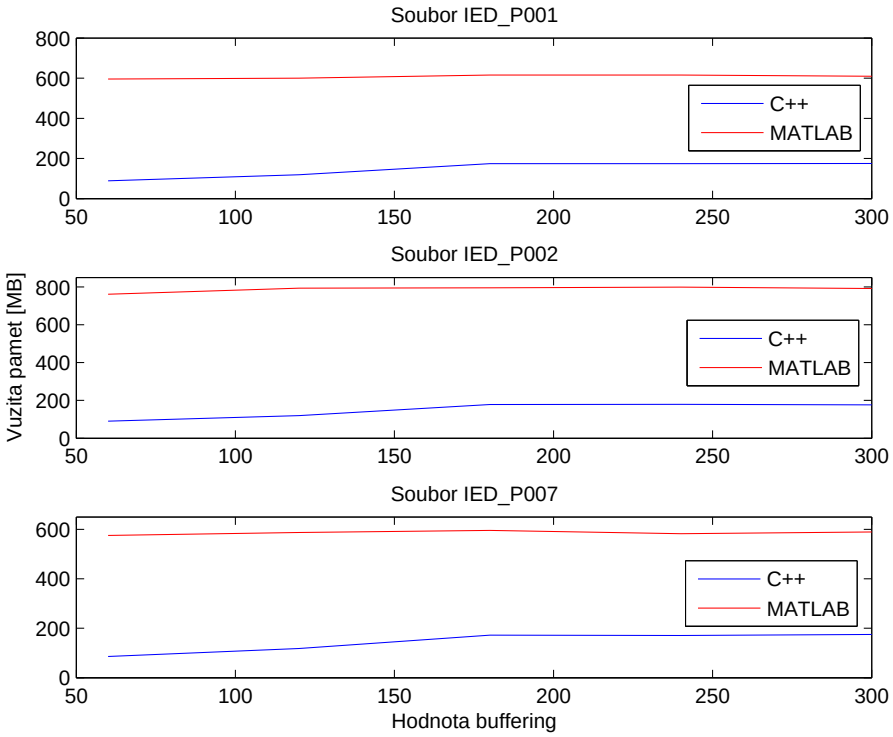
Na obrázku 5.1 jsou zobrazeny grafy využité paměti při procesu analýzy a následném zobrazení výsledků pro obě implementace. Data jsou měřena na systému Ubuntu 14.04. Naměřená data zahrnují paměť pro celou aplikaci spike detektoru a prostředí MATLAB. Vstupními soubory jsou IED_P001, IED_P002 a IED_P007. Ty jsou vybrány z důvodu odlišných vzorkovacích frekvencí a tedy rozdílnému počtu záznamů signálu.

Dále tyto grafy zachycují paměťové nároky pro různé hodnoty *buffering* s nastavenou délkou segmentu jedna až pět minut.

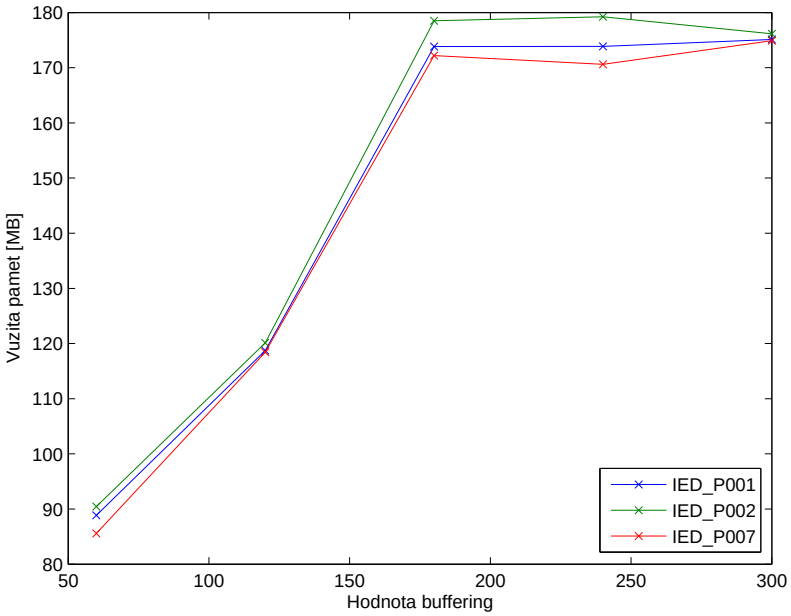
Z těchto grafů lze pozorovat, že analýza pomocí prostředí MATLAB využívá zaokrouhleně v průměru o 600 MB paměti navíc. To je především dáno paměťovými nároky samotného prostředí, které po spuštění používá okolo 500 MB paměti.

Závislost použité paměti na hodnotě *buffering* je lépe zobrazena na obrázku 5.2. Ten zobrazuje hodnoty pro tuto implementaci. Zobrazený graf ukazuje přímou úměrnost použité paměti na velikosti načteného segmentu signálu. Vstupní signál je pro tyto případy rozdělen na pět segmentů pro *buffering* nastavený na 60 s a dva segmenty pro *buffering* nastavený na 120 s. Pro zbylé hodnoty *buffering* již k segmentaci nedochází.

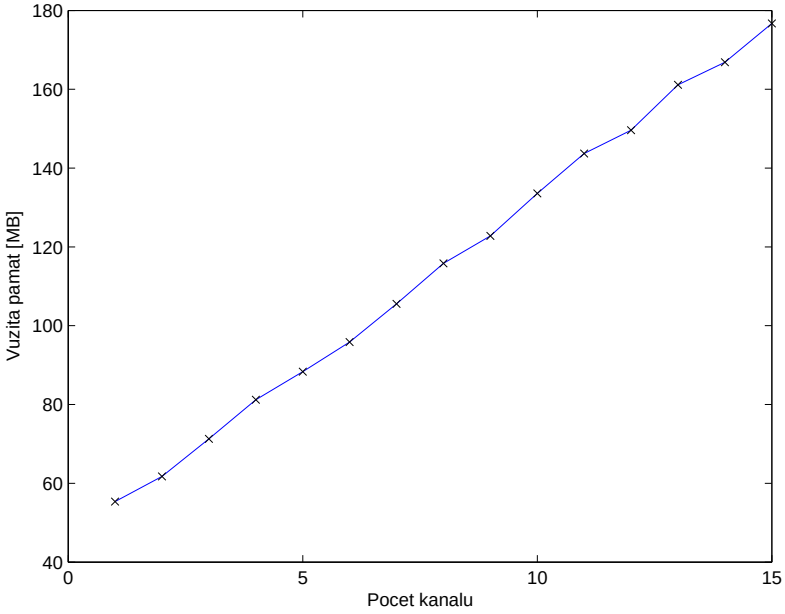
Obrázek 5.3 zobrazuje lineární nárůst paměti pro zvyšující se počet kanálů vstupního signálu s nastavenou hodnotou *buffering* na 300 s.



Obrázek 5.1: Porovnání využití paměti



Obrázek 5.2: Využití paměti implementace v C++



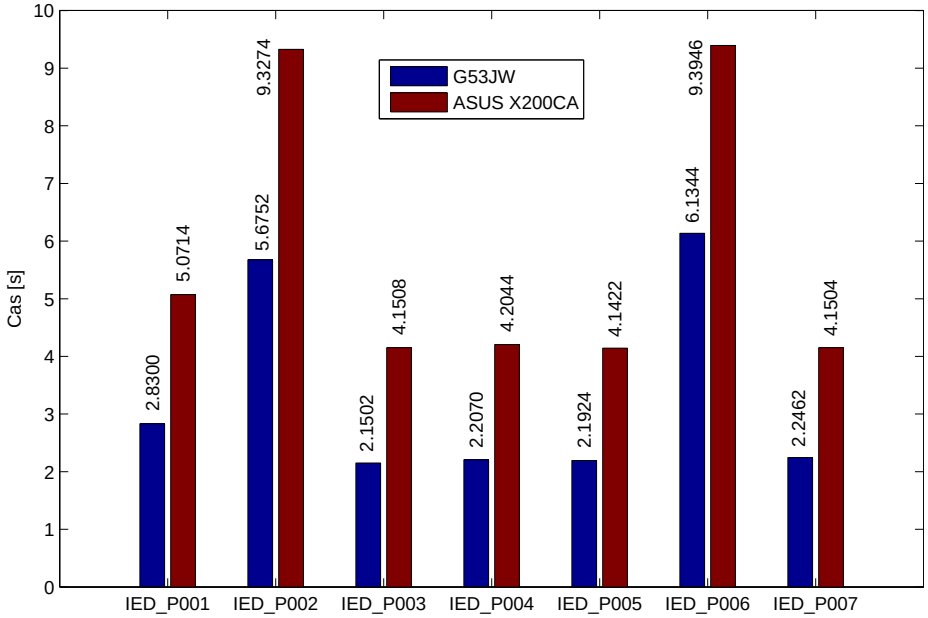
Obrázek 5.3: Využití paměti při zvýšení počtu kanálů

5.4 Porovnání doby běhu analýzy

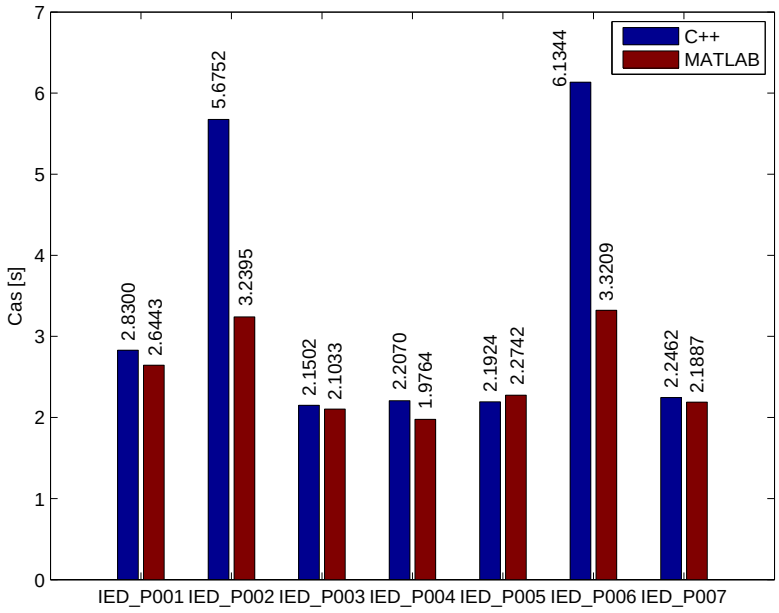
Měření doby analýzy je provedena na 64bitových operačních systémech MS Windows 7 a Ubuntu 14.04 pro obě implementace. Dále je porovnána doba analýzy této implementace na dvou noteboocích s odlišnou konfigurací.

Prvním notebookem je ASUS G53JW s dvoujádrovým procesorem Intel Core i5 460M (2,53 GHz) s 4-mi thready a se 4 GB DDR2 RAM paměti. Na tomto notebooku proběhlo měření doby analýzy obou implementací pod systémy MS Windwos 7 a Ubuntu 14.04. Druhým notebookem je ASUS X200CA s dvoujádrovým procesorem Intel Celeron 1007U (1,50 GHz) a 2 GB DDR2 RAM paměti. Na druhém notebooku je použit 64bitový operační systém MS Windows 8 Pro.

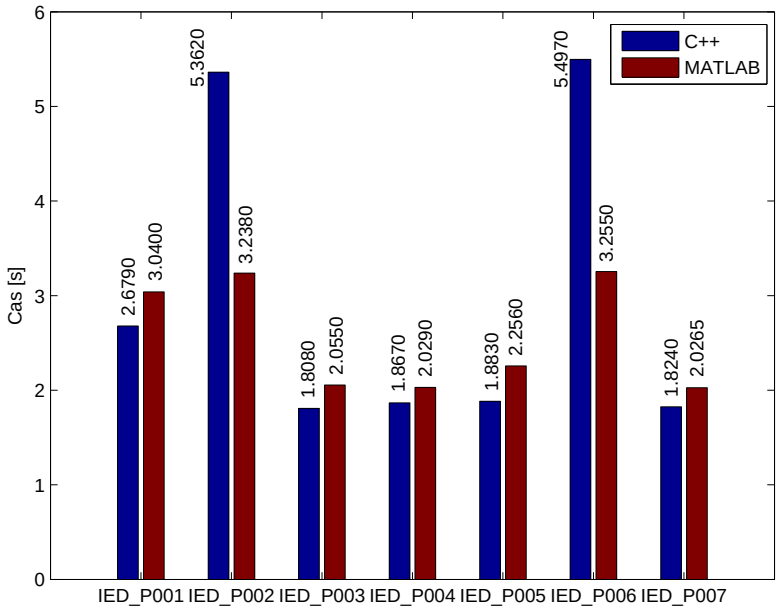
Následující obrázky obsahují grafy zobrazující průměrné časy z pěti měření pro každý vstupní soubor. V systému MS Windows 7 je aplikaci spike detektoru a prostředí MATLAB nastavena priorita běhu procesu na hodnotu „Vysoká“. Pro tyto měření je použito výchozí nastavení detektoru.



Obrázek 5.4: Doba běhu analýzy ASUS G53JW vs. ASUS X200CA



Obrázek 5.5: Doba běhu analýzy MS Windows 7



Obrázek 5.6: Doba běhu analýzy Ubuntu 14.04

Z obrázků 5.5 a 5.6 lze pozorovat, že časová náročnost analýzy této implementace je podobná časové náročnosti referenční implementace v případě vstupních dat se vzorkovací frekvencí blížící se pracovní frekvenci analýzy, signál se vzorkovací frekvence na 200 Hz a 250 Hz. V systému MS Windows 7 vychází průměrná časová náročnost analýzy této implementace přibližně o 4 % hůře než v MATLABu. Naopak pro systém Ubuntu 14.04 bylo naměřeno zlepšení přibližně o 12 %.

V případě vyšší vzorkovací frekvence je nárůst doby analýzy této implementace pro oba systémy vyšší. Pro vzorkovací frekvenci 1000 Hz je čas analýzy téměř dvojnásobný. Tento nárůst času je zapříčiněn decimací vstupního signálu, která je provedena knihovnou *libsamplerate*.

Graf na obrázku 5.4 zobrazuje razantní snížení doby analýzy při použití výkonnějšího procesoru s více logickými jádry. Zde došlo k zrychlení analýzy v mezích 40 % a 50 % časové náročnosti.

5.5 Hypotetická maxima a doporučená konfigurace

V současné době je výzkumná skupina ISARG schopna zaznamenávat iEEG signál s maximální vzorkovací frekvencí 2048 Hz pro maximálně 256 kanálů. Při těchto parametrech analyzovaných dat a výchozím nastavení detektoru vyšplhají paměťové nároky na několik stovek megabajtů. Také se výrazně prodlouží doba analýzy z důvodu převzorkování. Celková doba je ovšem závislá na délce vstupního signálu.

Pro vstupní data s maximálními parametry je doporučeno mít alespoň 2 GB paměti RAM a procesor se čtyřmi jádry s frekvenci alespoň 2 GHz.

V případě analyzování dat s parametry podobnými testovaným datům, je dostačující 512 MB paměti RAM a procesor s dvěma fyzickými jádry. Zde pro urychlení procesu je ovšem vhodné, aby daný procesor měl více logických jader.

Závěr

Cílem práce bylo analyzovat, implementovat a optimalizovat algoritmus spike detektoru epileptickoformních výbojů. Implementace tohoto algoritmu má splňovat požadavek objektově orientovaného jazyka, který je možné kompilovat pod platformami operačních systémů Linux a Microsoft Windows, případně Android.

Po analýze algoritmu a možností řešení je pro implementaci zvolen jazyk C++ s využitím knihovny wxWidgets. To umožňuje kompilovat výslednou aplikaci na systémech Microsoft Windows a Linux pro stejnou verzí zdrojového kódu. To je ověřeno sestavením a testováním aplikace na systémech MS Windows 7 a Ubuntu 14.04. Dále je tato aplikace testována na systému MS Windows 8, kde je použito sestavení ze systému MS Windows 7.

Pomocí knihovny wxWidgets je dále vytvořeno grafické uživatelské prostředí, které umožňuje obsluhovat implementovaný algoritmus a protokolovat výsledky.

Správnost implementace algoritmu je ověřena porovnáním výsledků pro sedm sad testovacích dat a výchozí nastavení detektoru. Zde je absolutní shoda detekovaných pozic s pozicemi detekovanými referenčním řešením. Je tedy splněn požadavek na maximálně 1% chybovost.

Optimalizace je splněna částečně. Implementovaná aplikace je paměťově méně náročná oproti dodané implementaci algoritmu ve skriptovacím jazyce MATLAB. Do těchto nároků ovšem spadají paměťové nároky celé implementované aplikace a celé paměťové nároky prostředí MATLAB, na kterém je spouštěna referenční implementace.

Z pohledu časových nároků na průběh algoritmu dosahuje tato implementace velice podobných časů jako implementace v jazyce MATLAB pro vstupní signál se vzorkovací frekvencí blížící se pracovní frekvenci analýzy. Tyto časy jsou rozdílné v řádu maximálně desetin sekund, což v poměru s časem procesu činí přibližně 4 % - 12 %. Pro systém MS Windows 7 vychází měřené časy hůře, oproti tomu pro systém Ubuntu 14.04 lépe.

Pro vstupní signál s vyšší vzorkovací frekvencí tato implementace již zaostává. To je způsobeno operací decimace vstupního signálu, která je v této implementaci časově náročnější. Pro převzorkování digitálního signálu je použita knihovna *libsamplerate*. Ta je z časových důvodů na tuto práci pro proces převzorkování ponechána.

Výsledná aplikace je připravena pro použití v produkčním prostředí. Toto řešení je otevřené pro další rozvoj nebo úpravy. Zde se nabízí použití jiného nástroje pro převzorkování digitálního signálů, který by urychlil proces analýzy. Dále se nabízí vytvořit komponentu zobrazující výsledky analýzy prostřednictvím grafu.

Literatura

[1] Janča, R.; Ježdík, P.; Cmejla, R.; aj.: *Detection of Interictal Epileptiform Discharges Using Signal Envelope Distribution Modelling: Application to Epileptic and Non-Epileptic Intracranial Recordings. Spike Detector Algorithm.* Springer Science+Business Media New York: New York, 2014, 12 s. Dostupné z: http://link.springer.com/article/10.1007%2Fs10548-014-0379-1

[2] Bernard, B.: *Prezentační vzory z rodiny MVC [online].* 2009, [cit. 2015-03-27]. Dostupné z: http://www.zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/

[3] Nemocnice na Homolce: *Typy epileptických záchvatů [online].* 2015, [cit. 2015-03-14]. Dostupné z: http://www.homolka.cz/cs-CZ/oddeleni/specializovana-centra/centrum-pro-epilepsie/pro-pacienty/typy-epileptickych-zachvatu.html

[4] Janča, R.; Ježdík, P.; Cmejla, R.; aj.: *Detection of Interictal Epileptiform Discharges Using Signal Envelope Distribution Modelling: Application to Epileptic and Non-Epileptic Intracranial Recordings.* Springer Science+Business Media New York: New York, 2014, 12 s. Dostupné z: http://link.springer.com/article/10.1007%2Fs10548-014-0379-1

[5] The MathWorks, Inc: *MATLAB The language of Technical Computing [online].* 2015, [cit. 2015-03-14]. Dostupné z: http://www.mathworks.com/products/matlab/

[6] ISARG: *ISARG [online].* 2015, [cit. 2015-03-14]. Dostupné z: http://isarg.feld.cvut.cz/downloads.html

[7] Weisstein, E. W.: *Maximum Likelihood [online].* The MathWorks, Inc, 2015, [cit. 2015-03-16]. Dostupné z: http://mathworld.wolfram.com/MaximumLikelihood.html

[8] Mlejnek, J.: *BI-SI1.2 - Přednáška č.3 - Analýza a sběr požadavků.* Říjen 2014, [cit. 2015-03-20]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-SI1/_media/lectures/03/03.prednaska.pdf

[9] Kemp, B.: *European Data Format [online].* 2015, [cit. 2015-03-20]. Dostupné z: http://www.edfplus.info/

[10] cplusplus.com: *Information [online].* 2015, [cit. 2015-03-20]. Dostupné z: http://www.cplusplus.com/info/

[11] ORACLE: *What is Java? [online].* 2015, [cit. 2015-03-20]. Dostupné z: http://java.com/en/download/whatis_java.jsp

[12] Python Software Foundation: *Python [online].* 2015, [cit. 2015-03-20]. Dostupné z: https://www.python.org/

[13] Google: *Android NDK | Android Developers [online].* 2015, [cit. 2015-03-20]. Dostupné z: https://developer.android.com/tools/sdk/ndk/index.html

[14] Google: *Kivy Launcher [online].* 2015, [cit. 2015-03-20]. Dostupné z: https://play.google.com/store/apps/details?id=org.kivy.pygame

[15] kivy.org: *Kivy [online].* 2015, [cit. 2015-03-20]. Dostupné z: http://kivy.org/

[16] Google: *Android Studio and SDK Tools [online].* 2015, [cit. 2015-03-20]. Dostupné z: http://developer.android.com/sdk/index.html

[17] Microsoft: *Windows API Index [online].* 2015, [cit. 2015-03-21]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516%28v=vs.85%29.aspx

[18] The IEEE and The Open Group: *The Open Group Base Specifications Issue 7 [online].* 2013, [cit. 2015-03-21]. Dostupné z: http://pubs.opengroup.org/stage7tc1/

[19] The GNOME Project: *GNOME [online].* 2015, [cit. 2015-03-21]. Dostupné z: https://www.gnome.org/

[20] The GTK+ Team: *The GTK+ Project [online].* 2014, [cit. 2015-03-21]. Dostupné z: http://www.gtk.org/

[21] KDE and the K Desktop Environment: *KDE [online].* 2015, [cit. 2015-03-21]. Dostupné z: https://www.kde.org/

[22] Qt Project Hosting: *Qt Project [online].* 2014, [cit. 2015-03-21]. Dostupné z: http://qt-project.org/

[23] Nadeau, D. D. R.: *C/C++ tip: How to detect the operating system type using compiler predefined macros [online]*. 2012, [cit. 2015-03-21]. Dostupné z: http://nadeausoftware.com/articles/2012/01/c_c_tip_how_use_compiler_predefined_macros_detect_operating_system

[24] wxWidgets: *wxWidgets Cross-Platform GUI Library [online]*. 2015, [cit. 2015-03-21]. Dostupné z: http://wxwidgets.org/

[25] wxWidgets: *WxWidgets Compared To Other Toolkits [online]*. 2015, [cit. 2015-03-21]. Dostupné z: https://wiki.wxwidgets.org/WxWidgets_Compared_To_Other_Toolkits#Java_GUI_toolkits

[26] Fowler, M.: *Supervising Controller [online]*. 2006, [cit. 2015-03-27]. Dostupné z: http://martinfowler.com/eaaDev/SupervisingPresenter.html

[27] W3Schools: *Introduction to XML [online]*. 2015, [cit. 2015-03-25]. Dostupné z: http://www.w3schools.com/xml/xml_whatis.asp

[28] The MathWorks, Inc.: *MAT-File Versions [online]*. 2015, [cit. 2015-03-25]. Dostupné z: http://www.mathworks.com/help/matlab/import_export/mat-file-versions.html?searchHighlight=MAT-file

[29] MinGW.org: *MSYS [online]*. 2015, [cit. 2015-03-26]. Dostupné z: http://www.mingw.org/wiki/msys

[30] MinGW.org: *Home of the MinGW and MSYS Projects [online]*. 2015, [cit. 2015-03-26]. Dostupné z: http://www.mingw.org/

[31] Free Software Foundation, Inc.: *GNU Make [online]*. 2015, [cit. 2015-03-26]. Dostupné z: https://www.gnu.org/software/make/

[32] DynamoRIO: *Dr. Memory Memory Debugger for Windows and Linux [online]*. 2015, [cit. 2015-03-27]. Dostupné z: http://www.drmemory.org/

[33] Valgrind™ Developers: *Valgrind [online]*. 2014, [cit. 2015-03-27]. Dostupné z: http://valgrind.org/

[34] doxygen: *Doxygen [online]*. 2015, [cit. 2015-04-25]. Dostupné z: http://www.stack.nl/~dimitri/doxygen/

[35] wxWidgets: *wxApp Class Reference [online]*. 2015, [cit. 2015-04-03]. Dostupné z: http://docs.wxwidgets.org/trunk/classwx_app.html

[36] wxWidgets: *wxFrame Class Reference [online]*. 2015, [cit. 2015-04-04]. Dostupné z: http://docs.wxwidgets.org/trunk/classwx_frame.html

[37] wxWidgets: *wxDialog Class Reference [online]*. 2015, [cit. 2015-04-04]. Dostupné z: http://docs.wxwidgets.org/trunk/classwx_dialog.html

[38] wxWidgets: *wxThread Class Reference [online]*. 2015, [cit. 2015-03-30]. Dostupné z: http://docs.wxwidgets.org/trunk/classwx_thread.html

[39] ALGLIB Project: *ALGLIB [online]*. 2015, [cit. 2015-04-05]. Dostupné z: http://www.alglib.net/

[40] ALGLIB Project: *Spline interpolation [online]*. 2015, [cit. 2015-04-05]. Dostupné z: http://www.alglib.net/interpolation/spline3.php

[41] ALGLIB Project: *Convolution [online]*. 2015, [cit. 2015-04-05]. Dostupné z: http://www.alglib.net/fasttransforms/convolution.php

[42] Lopo, E. C.: *Secret Rabbit Code [online]*. 2014, [cit. 2015-04-06]. Dostupné z: http://www.mega-nerd.com/SRC/

[43] Pardinas, D.: *MATLAB's filtfilt() Algorithm [online]*. 2014, [cit. 2015-04-06]. Dostupné z: http://stackoverflow.com/questions/17675053/matlabs-filtfilt-algorithm/27270420#27270420

[44] Guennebaud, G.; Benoit, J.; aj.: *Eigen v3 [online]*. 2010, [cit. 2015-04-06]. Dostupné z: http://eigen.tuxfamily.org

[45] Beelen, T.: *EDFlib [online]*. 2015, [cit. 2015-04-06]. Dostupné z: http://www.teuniz.net/edflib/

[46] Dice Holdings, Inc.: *MAT File I/O Library [online]*. 2015, [cit. 2015-03-25]. Dostupné z: http://matio.sourceforge.net/

[47] Dice Holdings, Inc.: *TinyXML [online]*. 2015, [cit. 2015-03-25]. Dostupné z: http://sourceforge.net/projects/tinyxml/

[48] wxWidgets: *Logging [online]*. 2015, [cit. 2015-04-08]. Dostupné z: http://docs.wxwidgets.org/trunk/group__group__funcmacro__log.html

[49] Free Software Foundation, Inc.: *GNU General Public License [online]*. 2007, [cit. 2015-04-08]. Dostupné z: https://www.gnu.org/licenses/gpl-2.0.html

[50] Free Software Foundation, Inc.: *GNU General Public License [online]*. 2007, [cit. 2015-04-06]. Dostupné z: http://www.gnu.org/copyleft/gpl.html

[51] mozilla.org: *Mozilla Public License Version 2.0 [online]*. 2014, [cit. 2015-04-06]. Dostupné z: https://www.mozilla.org/MPL/2.0/

[52] Opensource.org: *The zlib/libpng License [online]*. 2015, [cit. 2015-04-08]. Dostupné z: http://opensource.org/licenses/Zlib

[53] Opensource.org: *The BSD 3-Clause License [online]*. 2015, [cit. 2015-04-08]. Dostupné z: http://opensource.org/licenses/BSD-3-Clause

[54] wxWidgets: *wxVector<T> Class Template Reference [online]*. 2015, [cit. 2015-05-05]. Dostupné z: http://docs.wxwidgets.org/trunk/classwx_vector_3_01_t_01_4.html

[55] wxWidgets: *wxGrid Class Reference [online]*. 2015, [cit. 2015-04-04]. Dostupné z: http://docs.wxwidgets.org/trunk/classwx_grid.html

[56] The MathWorks, Inc.: *hilbert [online]*. 2015, [cit. 2015-04-06]. Dostupné z: http://www.mathworks.com/help/signal/ref/hilbert.html

[57] ALGLIB Project: *Fast Fourier transform [online]*. 2015, [cit. 2015-04-06]. Dostupné z: http://www.alglib.net/fasttransforms/fft.php

[58] The MathWorks, Inc.: *Chebyshev Type II filter design [online]*. 2015, [cit. 2015-04-06]. Dostupné z: http://www.mathworks.com/help/signal/ref/cheby2.html

[59] The MathWorks, Inc.: *Butterworth filter order and cutoff frequency [online]*. 2015, [cit. 2015-04-06]. Dostupné z: http://www.mathworks.com/help/signal/ref/buttord.html

[60] EPFL: *rtfilter [online]*. 2014, [cit. 2015-04-06]. Dostupné z: http://cnbi.epfl.ch/software/rtfilter.html

[61] Free Software Foundation, Inc.: *GNU Lesser General Public License [online]*. 2007, [cit. 2015-04-06]. Dostupné z: http://www.gnu.org/licenses/lgpl.html

[62] Lopo, E. C.: *Miscellaneous API Documentation [online]*. 2011, [cit. 2015-04-06]. Dostupné z: http://www.mega-nerd.com/SRC/api_misc.html

Příloha **A**

Seznam použitých zkratek

API	–	rozhraní pro programování aplikace
CDF	–	kumulativní distribuční funkce
FIR	–	filtr s konečnou impulzní odezvou
FS	–	vzorkovací frekvence
GUI	–	grafické uživatelské rozhraní (Graphics User Interface)
IDE	–	komplexní vývojové prostředí v jednom softwaru
IED	–	vnitrolebeční epileptickoformní výboje
iEEG	–	nitrolebeční elektroencefalogram
ISARG	–	Intracranial Signal Analysis Research Group
MLE	–	maximální pravděpodobnostní algoritmus
MS	–	Microsoft
SDK	–	Software Development Kit
XML	–	Extensible markup language

Příloha B

Příručka sestavení aplikace

Tato příručka popisuje postup pro sestavení aplikace spike detektoru na systémech MS Windows 7 a Ubuntu 14.04. Vedle sestavení samotné aplikace je zde popsáno, jak sestavit použité knihovny a které nástroje pro to využít.

Knihovny, které je nutné sestavit před samotnou aplikací jsou:

- wxWidgets, použitá verze 3.0.2,
- libsamplerate, použitá verze 0.1.8,
- MATio, použitá verze 1.5.2.

Příkazy konzole jsou uvedeny v uzavřených závorkách „[]“, jejich vykonání se provede bez těchto závorek.

Kompletní seznam nástrojů a jejich verzi naleznete v sekci Vývojové prostředí.

B.1 Adresářová struktura

Implementace spike detektoru je rozdělena do následující struktury:

```
/
├── icon ..... adresář obsahující ikony použité v aplikaci
├── libraries ..... adresář pro sestavené externí knihovny
├── obj ..... adresář pro odkládání souborů pro sestavení
├── src ..... adresář se zdrojovými kódy implementace
├── Doxygen ..... Konfigurační soubor pro vygenerování dokumentace
├── Makefile.win ..... Makefile pro systémy Windows
├── Makefile.lin ..... Makefile pro systémy na Linuxovém jádře
└── settings.xml ..... XML dokument obsahující nastavení detektoru
```

B.2 MS Windows 7

Na tomto systému je pro sestavení aplikace a knihoven použita sada nástrojů MinGW. Veškeré níže uvedené příkazy jsou spouštěny v konzoli sady MSYS.

B.2.1 Sestavení knihoven

B.2.1.1 Knihovna wxWidgets

Po stažení a rozbalení archivu jsou provedeny následující kroky:

1. Přesunutí do adresáře se zdrojovými soubory knihovny pomocí příkazu: [cd /cesta/do/adresare]

2. Vytvoření adresáře pro sestavení a přesun do ní: [mkdir build-debug], posléze: [cd build-debug]

3. Spuštění konfigurace:
   [../configure --enable-unicode CXXFLAGS="-std=gnu++11"]

4. Spuštění sestavení: [make]

5. Po dokončení procesu sestavování se soubory sestavené knihovny nacházejí v adresáři „build-debug/lib“. Na systémech Windows je zvoleno použití dynamické podoby této knihovny. Proto je nutné překopírovat soubory končící na „.dll“ do adresáře, kde bude sestavena koncová aplikace.

B.2.1.2 Knihovna matio

Po stažení a rozbalení archivu jsou provedeny následující kroky:

1. Přesunutí do adresáře se zdrojovými soubory knihovny pomocí příkazu: [cd /cesta/do/adresare]

2. Spuštění konfigurace: [./configure]

3. Spuštění sestavení: [make]

4. Po dokončení procesu sestavování se soubory sestavené knihovny nacházejí v adresáři „src/libs“. Ze souborů v tomto adresáři je důležitý soubor libmatio.a, který je nutné překopírovat do adresáře „libraries“ v adresáři detektoru.

B.2.1.3 Knihovna libsamplerate

Po stažení a rozbalení archivu jsou provedeny následující kroky:

1. Přesunutí do adresáře se zdrojovými soubory knihovny pomocí příkazu: [cd /cesta/do/adresare]
2. Spuštění konfigurace: [./configure]
3. Spuštění sestavení: [make]
4. Po dokončení procesu sestavování se soubory sestavené knihovny nacházejí v adresáři „src/.libs“. Ze souborů v tomto adresáři je důležitý soubor libsamplerate.a, který je nutné překopírovat do adresáře „libraries“ v adresáři detektoru.

B.3 Ubuntu 14.04

Na tomto systému je použit standardní terminál. Knihovna wxWidgets je zde sestavena jako statická knihovna.

Nejdříve je nutné nainstalovat potřebné nástroje:

- Sadu překladačů GCC, v případě, že již není nainstalována: [sudo apt-get install g++]
- Vývojové soubory pro knihovnu GTK+: [sudo apt-get install libgtk-3-dev]

B.3.1 Sestavení knihoven

B.3.1.1 Knihovna wxWidgets

Po stažení a rozbalení archivu jsou provedeny následující kroky:

1. Přesunutí do adresáře se zdrojovými soubory knihovny pomocí příkazu: [cd /cesta/do/adresare]
2. Vytvoření adresáře pro sestavení a přesun do něj: [mkdir gtk-build], posléze: [cd gtk-build]
3. Spuštění konfigurace: [sudo ../configure --enable-unicode --disable-shared]
4. Spuštění sestavení: [sudo make]
5. Na závěr instalace knihovny do globálního umístění: [sudo make install]

B.3.1.2 Knihovna matio

Po stažení a rozbalení archivu jsou provedeny následující kroky:

1. Přesunutí do adresáře se zdrojovými soubory knihovny pomocí příkazu: [cd /cesta/do/adresare]
2. Spuštění konfigurace: [./configure]
3. Spuštění sestavení: [make]
4. Volitelně kontrola sestavení: [make check]
5. Na závěr instalace knihovny do globálního umístění: [sudo make install]

B.3.1.3 Knihovna libsamplerate

Po stažení a rozbalení archivu jsou provedeny následující kroky:

1. Přesunutí do adresáře se zdrojovými soubory knihovny pomocí příkazu: [cd /cesta/do/adresare]
2. Spuštění konfigurace: [./configure]
3. Spuštění sestavení: [make]
4. Volitelně kontrola sestavení: [make check]
5. Na závěr instalace knihovny do globálního umístění: [sudo make install]

B.4 Sestavení aplikace

Před spuštěním sestavení aplikace spike detektoru je nejdříve nutné upravit příslušný Makefile. Ve složce s detektorem se nachází dva soubory Makefile.win pro systémy Windows a Makefile.lin pro systémy Linux. Vybereme odpovídající soubor a přejmenujeme ho na Makefile. Dále tento soubor otevřeme v některém textovém editoru.

Zde se na řádce číslo pět nachází definice proměnné *wx_top_builddir*. Ta obsahuje cestu k adresáři se sestavením knihovny wxWidgets. Tuto řádku je nutné upravit tak, aby odpovídala aktuální situaci.

Po této úpravě následuje sestavení samotné aplikace pomocí příkazové řádky:

1. Přesunutí do adresáře detektoru [cd /cesta/do/adresare]
2. Spuštění sestavení: [make]

Příloha C

Popis tříd

V této příloze se nachází dodatečné informace k jednotlivým třídám implementace.

V následujícím textu je použit datový typ *SIGNALTYPE*, který určuje primitivní datový typ vzorků signálu. V této implementaci představuje datový typ *float*. Dále je zde používán termín vektor, ten představuje šablonu třídy datového kontejneru *wxVector* [54].

C.1 Třídy realizující GUI

Třída CMainFrame vedle grafických komponent pro obsluhu detektoru obsahuje dvě členské proměnné, *m_gridOut* a *m_gridDischarges*, typu *wxGrid* [55] představující tabulky pro zobrazení výsledků analýzy.

Dalšími důležitými členskými proměnnými jsou:

- *m_model* – ukazatel na instanci třídy *CInputModelEDF*. Adresa tohoto objektu je předána pomocí konstruktoru při inicializaci třídou *CMain*.
- *m_detector* – ukazatel na instanci třídy *CSpikeDetecor*.
- *m_out* – ukazatel na instanci třídy *CDetectorOutput*.
- *m_disch* – ukazatel na instanci třídy *CDischarges*.

Proměnné *m_detector*, *m_out*, *m_disch* jsou v době inicializace nastaveny na hodnotu *NULL*.

V této třídě jsou dále definovány metody reagující na uživatelské akce. Ty jsou vyvolány při kliknutí na danou položku v menu, nebo při kliknutí na některou ikonu v panelu nástrojů. Těmito akcemi jsou:

- Otevření souboru s daty k analýze. Tuto akci obstarává metoda:
  void OnOpen(wxCommandEvent& WXUNUSED(event)).

  Při tomto požadavku je vytvořeno a zobrazeno dialogové okno pro otevření souboru. Po výběru souboru je absolutní cesta tohoto souboru předána instanci třídy *CInputeModelEDF*, která tento soubor dále zpracuje.

  Po této proceduře je zpřístupněna možnost spuštění analýzy nad zvolenými daty.

- Spuštění analýzy. Tuto akci obstarává metoda:
  void OnRun(wxCommandEvent& WXUNUSED(event)).

  Zde jsou vytvořeny instance tříd *CDetectorOutput*, *CDischarges* a *CSpikeDetector*. Dále je zobrazeno dialogové okno zobrazující postup analýzy a je spuštěn samotný proces analýzy vstupních dat. Ten je realizován třídou *CSpikeDetetor*.

  Tato akce je přístupná jen v případě, že je zvolen a úspěšně načten vstupní soubor s daty k analýze.

- Uložení výsledků analýzy. Tato akce je řešena v metodě:
  void OnSaveResults(wxCommandEvent& WXUNUSED(event)).

  Zde dochází k vytvoření a zobrazení dialogového okna pro uložení souboru. Po pojmenování nově vytvářeného souboru nebo zvolení souboru, který bude přepsán, jsou výsledky uloženy.

  Uložení probíhá pomocí statické metody *SaveResultsXML* pro uložení do dokumentu XML, nebo pomocí statické metody *SaveResultsMAT* pro uložení do binárního datového formátu MAT. Tyto metody jsou definovány ve třídě *CResultsModel*.

- Načtení výsledků analýzy. Metoda:
  void OnLoadResults(wxCommandEvent& WXUNUSED(event)).

  Zde je vytvořeno a zobrazeno dialogové okno pro výběr souboru s výsledky. Po vybrání souboru jsou výsledky načteny a zobrazeny do tabulek.

  Načtení výsledků ze souboru obstarávají statické metody *LoadResultsXML* nebo *LoadResultsMAT* třídy *CResultsModel*. Ty jsou uloženy do třídních proměnných a následně zobrazený metodou *void plotOutputs()* do tabulek.

- Zobrazení dialogového okna nastavení. Metoda:
  void OnOptions(wxCommandEvent& WXUNUSED(event)).

  Tato metoda pouze zobrazí dialogové okno pro práci s nastavením detektoru.

  To je reprezentováno instancí třídy *CSettingsDialog*.

- Zobrazení informací o aplikaci. Při této akci je zavolána metoda:
  void OnAbout(wxCommandEvent& WXUNUSED(event)).

  Dochází zde jen k zobrazení dialogového okna se zprávou obsahující informace o aplikaci.

- Uzavření hlavního okna. Při tomto požadavku je zavolána metoda:
  void OnClose(wxCloseEvent& event).

  V této metodě dochází k úklidu paměti alokované touto třídou. Poté je zavolán příkaz k ukončení aplikace.

Mimo metod pro obsluhu uživatelské interakce jsou v této třída definovány tyto další metody:

- *void OnDetectorEvent(wxThreadEvent& event)* pro zachycení událostí zaslaných z instance třídy *CSpikeDetector*.

  Tyto události určují tři možné stavy analýzy:

  1. Aktualizace stavu průběhu analýzy. Událost obsahuje kladnou hodnotu určující stav průběhu analýzy.
  2. Dokončení procesu analýzy. Událost obsahuje číselnou hodnotu -1.

     V tomto případě je proces analýzy dokončen a jsou zobrazeny výsledky. Také je zpřístupněna možnost tyto výsledky uložit do souboru.
  3. Chyba analýzy. Událost obsahuje číselnou hodnotu -1 a textovou zprávu.

     V případě chyby analýzy je zaslána událost obsahující textový popis této chyby. Ta je dále zobrazena uživateli.

- Metodu *void clear()*, která převede aplikaci do stavu stejnému po spuštění.

  Tedy jsou provedeny operace uzavření vstupního souboru s daty k analýze a smazání výsledků analýzy.

Třída CSettingsDialog má definovány metody pro obsluhu uživatelských požadavků pro práci s nastavením detektoru. Tyto požadavky jsou ovládány pomocí čtyř tlačítek v tomto okně. Těmito požadavky jsou:

- Načtení výchozího nastavení. Obsluhuje metoda:
  void OnLoadDefault(wxCommandEvent& WXUNUSED(event)).

  Nastavení detektoru je načítáno ze souboru pomocí instance třídy *CSettingsModel* metodou *LoadSettings(DEFAULT_SETTINGS)*. Tato metoda vrací strukturu *DETECTOR_SETTINGS* obsahující hodnoty nastavení. Po získání této struktury jsou aktualizována vstupní pole s hodnotami pomocí metody *RefreshControls()*.

- Načtení uloženého nastavení. Obsluhuje metoda:
  void OnLoadSaved(wxCommandEvent& WXUNUSED(event)).

  Operace této metody odpovídají načítání výchozího nastavení. Rozdílem zde je parametr metody *LoadSettings*, který je zde nastaven na *SAVED_SETTINGS*. To označuje požadavek na načtení uživatelem uloženého nastavení.

- Použití nastavení. Po načtení nebo změně nastavení je nutné potvrdit volbu tohoto nastavení. To provede metoda:
  void OnUse(wxCommandEvent& event).

  Tato metoda nejdříve ověří vyplněné hodnoty. V případě korektních hodnot je toto nastavení uloženo do třídní proměnné *m_settings*.

- Uložení nastavení. Uložení nastavení do souboru je řešeno v metodě:
  OnSave(wxCommandEvent& event).

  Nastavení je nejdříve zvalidováno. V případě korektních hodnot se provede uložení do souboru pomocí metody *SaveSettings* instance třídy *CSettingsModel*.

V předchozích metodách je používána metoda *bool ValidateInput()* pro korekci vyplněných hodnot. V případě nekorektních hodnot nastavení je vstupní pole daného nastavení vyplněno červenou barvou a je zobrazeno varování.

C.2 Třídy realizující jádro aplikace

Třída CSpikeDetector představuje hlavní třídu pro spuštění analýzy detekce. Její instance je vytvořena při každém požadavku na spuštění analýzy a po ukončení této práce je zničena – ukončením práce může být dokončení analýzy nebo vznik chyby v průběhu analýzy.

Pomocí konstruktoru jsou instanci třídy *CSpikeDetector* předány tyto parametry:

- Ukazatel na objekt třídy vycházející z třídy *wxEvtHandler*, do této třídy jsou zasílány události o postupu analýzy. V této aplikaci se jedná o instanci třídy *CMainFrame*.
- Ukazatel na objekt třídy *CInputModel* poskytující vstupní data.
- Ukazatel na strukturu *DETECTOR_SETTINGS* s nastaveními pro analýzu.
- Ukazatel na objekt třídy *CDetectorOutput*, do kterého jsou ukládány výsledky detekce jednotlivých výbojů.
- Ukazatel na objekt třídy *CDischarges*, do kterého jsou ukládány výsledky nalezených vícekanálových událostí.

Tyto parametry jsou uloženy do třídních proměnných. Další třídní proměnná *m_progress* udává o jak velkou hodnotu pokroku bude zvýšena hodnota v dialogovém okně zobrazující postup.

Vstupním bodem pro spuštění analýzy na novém vlákně je metoda *Entry()*. Ta pouze spustí analýzu (metoda *RunAnalysis()*) a po jejím dokončení zašle hlavnímu vláknu aplikace zprávu o dokončení nebo chybě.

Třída COneChannelDetect slouží k detekci jednokanálových událostí v jednom datovém kanálu. Instance této třídy je vytvořena v průběhu analýzy třídou *CSpikeDetector*.

Pro úlohu detekce jsou této třídě předány pomocí konstruktoru tyto hodnoty:

- Ukazatel na vektor obsahující vstupní data – jeden kanál segmentu iEEG signálu.
- Ukazatel na strukturu *DETECTOR_SETTINGS* s nastavením analýzy.
- Vzorkovací frekvenci vstupních dat.
- Vektor obsahující indexy segmentů vstupních dat za použití klouzavého okénka.
- Číslo kanálu z kterého pochází vstupní data.

Výstupem práce této třídy je struktura *ONECHANNELDETECTRET*, která obsahuje následující položky:

Název	Popis
m_markersHigh	Vektor obsahující značky lokálních maxim pro identifikaci jasných výbojů.
m_markersLow	Vektor obsahující značky lokálních maxim pro identifikaci nejasných výbojů. V případě, že není nastaven požadavek této detekce, tento vektor odpovídá vektoru *m_markersHigh*.
m_prahInt[2]	Vektory obsahujících spočtené prahovací křivky.
m_envelopeCdf	Vektor se statistickou významností "CDF" lognormální distribuce.
m_envelopePdf	Vektor se statistickou významností "PDF" lognormální distribuce.
m_envelope	Vektor obsahující absolutní hodnoty Hilbertovi transformace vstupních dat.

Tabulka C.1: Struktura *ONECHANNELDETECTRET*

C.3 Třídy realizující zpracování digitálního signálu

Třída CDSP je statická třída nabízející tyto funkce:

- Převzorkování signálu. Metoda pro tuto operaci:
  void Resample(wxVector<SIGNALTYPE>& data, const int& countChannels, const int& actualFS, const int& requiredFS).*

  Tato metoda pracuje s polem vektorů obsahující vstupní data k převzorkování, která jsou nahrazena výstupními. Zde každá položka pole vektorů představuje jeden kanál vstupních dat.

  Samotný proces převzorkování je řešen až třídou *CResamplingThread*, která pracuje s jedním kanálem vstupních dat. Tato metoda pouze vytváří instance třídy *CResamplingThread* pro každý kanál vstupních dat, spouští proces převzorkování a čeká na dokončení.

  Po dokončení převzorkování každého kanálu vstupních dat je ještě ověřováno zda odpovídají velikosti výstupních souborů. Požadovaná velikost je vypočtena vztahem:

$$s_2 = \lceil s_1 \cdot \frac{FS_2}{FS_1} \rceil, \tag{C.1}$$

kde

- s_1 je počet prvků jednoho kanálu vstupních dat,
- s_2 je očekávaný počet prvků v jednom kanálu po převzorkování,
- FS_1 je vzorkovací frekvence vstupních dat,
- FS_2 je požadovaná frekvence výstupních dat.

V případě, že převzorkovaná vstupní data nemají požadovanou velikost, jsou buďto doplněna hodnotami posledního prvku signálu, nebo jsou odebrány poslední přesahující prvky.

- Výpočet okamžité obálky pomocí absolutní hodnoty Hilbertovy transformace. Metoda:
  void AbsHilbert(wxVector<SIGNALTYPE>& data).

  Vstupně-výstupní vektor *data* obsahuje signál, z kterého je vypočtena Hilbertova transformace. Dále jsou data tohoto vektoru přepsána absolutními hodnotami vypočtené transformace.

  Výpočet Hilbertovy transformace je založen na algoritmu funkce *hilbert* [56] z prostředí MATLAB.

  Zde pro výpočet diskrétní Furiérovy transformace (FFT) a inverzní FFT jsou použity funkce *fftc1d* a *fftc1dinv* [57] z knihovny ALGLIB.

- Filtrování v nastaveném pásmu filtrem bez posunu fáze pomocí High pass a Low pass digitálního filtru. Metoda:
  void Filtering(wxVector<SIGNALTYPE> data, const int& countChannels, const int& fs, const BANDWIDTH& bandwidth).*

  Vstupem této funkce je vstupní signál, informace o počtu kanálů a vzorkovací frekvenci signálu. Hodnota *bandwidth* je strukturou obsahující spodní a horní hranici filtrování.

  Vstupní signál v parametru *data* je opět nahrazen výstupem filtrování, filtrovaným signálem.

  Filtrování probíhá pro každý kanál vstupního signálu zvlášť. Nejdříve je provedeno filtrování pomocí High pass digitálního filtru a poté filtrování pomocí Low pass digitálního filtru. Samotný filtr realizuje třída *CFiltFilt*, jejíž instance je vytvořena pro každý kanál signálu. Ta vedle tohoto kanálu vstupního signálu pracuje s dvěma vektory popisující filtr.

  Hodnoty těchto vektorů, koeficienty pro filtrování, jsou v případě výchozího nastavení předpočítány pomocí Chebyshev typ II filter designu [58] v prostředí MATLAB. V případě jiného nastavení je použito Butterworth filter designu.

- Filtrování síťového šumu. Metoda:
  void Filt50Hz(wxVector<SIGNALTYPE> data, const int& countChannels, const int& fs, const int& hum_fs, const BANDWIDTH& bandwidth).*

  Vstupní parametry této funkce odpovídají funkci *Filtering*. Je zde navíc jen parametr *hum_fs* obsahující hodnotu síťového šumu.

  Výstup je opět řešen nahrazením vstupních dat v parametru *data*.

  Filtrování taktéž probíhá pro každý kanál vstupního signálu zvlášť za využití filtru implementovaného v *CFiltFilt*.

- Butterworth filter design je řešen v metodách:
  void Buttord(const double& wp, double& ws, const double& rp, const double& rs, int& order) a
  bool Butter(wxVector<double>& b, wxVector<double>& a, const int& order, const double& Wn, FILTERTYPE ftype).

  – Metoda *Buttord* vypočítává řád filtru a mezní frekvenci. Tyto hodnoty jsou použity pro návrh digitálního filtru metodou *Butter*.

  Tato metoda simuluje algoritmus funkce *buttord* [59] z prostředí MATLAB.

  – Metoda *Butter* slouží pro pro návrh Butterworth digitálního filtru N-tého řádu.

  Parametry této funkce jsou vektory *b* a *a* do kterých jsou uloženy koeficienty návrhu digitálního filtru. Dále řád filtru *order*, mezní frekvence *Wn* a typ filtru. Typ filtru určuje zda se jedná o návrh Higpass nebo Lowpass digitálního filtru.

  Výpočet koeficientů pro digitální filtr typu Butterworth N-tého řádu je převzat z knihovny rtfilter [60] verze 1.1 pod licencí LGPLv3 [61].

Třída CResamplingThread realizuje převzorkování jednoho kanálu vstupních dat.

Pomocí konstruktoru jsou instanci této třídy předány parametry:

- Ukazatel na vektor obsahující signál pro převzorkování. Jedná se o vstupně-výstupní parametr. Vstupní signál je nahrazen převzorkovaným signálem.
- Vzorkovací frekvence vstupního signálu.
- Požadovaná vzorkovací frekvence výstupního signálu.

Pro účely této implementace je použito plné API knihovny libsamplerate s typem konvertoru *SRC_SINC_BEST_QUALITY* [62].

Třída CFiltFilt implementuje digitálního filtr bez posunu fáze.

Pomocí kontruktoru této třídy jsou předány:

- Vektory obsahující koeficienty filtru.
- Ukazatel na vektor obsahující vstupní signál pro filtrování.

  Ten opět slouží i pro výstup filtrování, kdy je vstupní signál nahrazen filtrovaným signálem.

C.4 Třídy realizující práci se soubory

Třída CInputModel je abstraktní třídou deklarující metody pro přístup k datům určeným pro analýzu. Některé metody, nezávislé na způsobu čtení dat z daného typu souboru, jsou zde přímo implementovány.

Metody pro přístup k datům:

- Otevření vstupního souboru. Předpis:
  void OpenFile(const char fileName).*

  Úkolem této metody je otevřít vstupní soubor a načíst základní informace o uloženém signálu. Těmi jsou vzorkovací frekvence, počet kanálů uloženého signálu a počet vzorků v jednom kanálu.

  Tato metoda je zde také předepsána v přetížené variantě, kde parametr *fileName* je typu *const wchar_t**. Toho je využito v případě, že absolutní cesta[9] k souboru je tvořena Unicode znaky.

- Uzavření vstupního souboru. Předpis:
  void CloseFile().

  Tato metoda uzavře otevřený soubor.

- Metoda pro ověření možnosti čtení ze souboru. Předpis:
  bool IsOpen() const.

  Zde je vrácena logická hodnota *true* pokud je vstupní soubor otevřen, v opačném přídě je vráceno *false*.

- Ověření dosažení konce uloženého iEEG signálu. Předpis:
  bool IsEnd() const.

  Zde je taktéž vrácená logická hodnota *true* v případě, že byla přečtena všechna data vstupního signálu.

[9] Úplná cesta k souboru od kořenového adresáře společně s názvem a typem souboru.

- Získání segmentu dat. Předpis:
  wxVector<SIGNALTYPE> GetSegment(const int& start, const int& end).*

  Tato metoda slouží k přečtení segmentu signálu. Ten je uložen v poli vektorů, kde každý jeden vektor představuje jeden kanál vzorků digitálního signálu.

  Segment signálu je určen parametry *start* a *end*, které určují počáteční a koncovou pozici čtení dat signálu v souboru.

- V poslední řadě jsou zde implementovány metody pro získání základních informací o uloženém signálu. Tyto metody jsou:

  – *int GetCountChannels() const* pro získání počtu kanálů digitálního signálu,

  – *int GetCountSamples() const* pro získání počtu vzorků jednoho datového kanálu,

  – *int GetFS() const* pro získání vzorkovací frekvence.

Třída CResultsModel slouží pro ukládání a načítání výsledků analýzy.

Metodami pro uložení výsledků jsou *SaveResultsMAT* pro formát MAT a *SaveResultsXML* pro dokumenty XML. Tyto metody mají vstupní parametry:

- *fileName* znakové pole obsahující plnou cestu k souboru,
- *out* ukazatel na objekt třídy *CDetectorOutput* obsahující data k uložení,
- *disch* ukazatel na objekt třídy *CDischarges* obsahující data k uložení.

Metody pro přečtení výsledků ze souboru jsou *LoadResultsMAT* a *LoadResultsXML*. Tyto metody mají obdobné parametry. Jen zde ukazatelé *out* a *disch* slouží pro předání adres nově vytvořených objektů s načtenými daty.

Třída CSettingsModel nabízí tyto metody:

- *DETECTOR_SETTINGS* LoadSettings(SETTINGS_TYPE type)* pro načtení nastavení detektoru ze souboru.

  Vstupním parametrem je typ nastavení, které má být načteno. Možné hodnoty jsou:

  – *DEFAULT_SETTINGS* pro výchozí nastavení,

  – *SAVED_SETTINGS* pro uživatelem uložené nastavení.

  Výstupem je ukazatel na strukturu *DETECTOR_SETTINGS*, v které jsou hodnoty načteného nastavení.

- *void SaveSettings(DETECTOR_SETTINGS* settings)* pro uložení uživatelem pozměněného nastavení.

  Vstupní parametr *settings* je ukazatel na strukturu s nastavením, které je uloženo do dokumentu XML.

Třída CDetectorOutput reprezentuje datovou strukturu *out* pro popis detekovaných výbojů.

Jednotlivé prvky detekovaného výboje jsou zaznamenávány do vektorů. Ukládané hodnoty jsou datového typu *double*. Výjimkou je číslo kanálu, které je typu *int*.

Vedle přímého přístupu k těmto kontejnerům je zde možné použít metodu *Add*, která přidá záznam detekovaného výboje na konec kontejnerů. Odebrání více záznamů výboje lze učinit metodou *void Remove(const wxVector<int>& pos)*, která odebere výboje na pozicích uložených v parametru *pos*.

Třída CDischarges reprezentuje datovou strukturu *discharges* pro popis vícekanálových událostí.

Implementace reprezentované struktury je tvořena poli vektorů. Ty jsou opět zvenčí volně přístupné. Prvky záznamů jsou datového typu *double*.

Při vytváření instance této třídy je nutné pomocí konstruktoru předat počet kanálu analyzovaného signálu. Tento počet určí velikost polí datových kontejnerů. Pro odebrání více záznamů na určitých pozicích je zde opět definována metoda *Remove*.

C.5 Třída realizující výjimky

Třída CException obsahuje tyto položky:

- *m_message* typu *wxString*. Chybová zpráva určená pro zobrazení uživateli.
- *m_whence* typu *wxString*. Název třídy a metody, kde vniklo k vyvolání výjimky.
- *m_time* typu *wxDateTime*. Čas vyvolání výjimky.

Hodnoty *m_message* a *m_whence* jsou inicializovány pomocí konstruktoru této třídy.

Vedle metod *GetMessage* a *GetLocation* pro získání zprávy a místa vyvolání výjimky, je zde přetížena metoda *const char* what() const throw()* pro získání chybové zprávy jako pole znaků.

Příloha **D**

Obsah přiloženého CD

- `readmea.txt` ..... stručný popis obsahu CD
- `data` ..... adresář s vstupními daty ve formátu EDF+
- `doxygen` ..... adresář s dokumentací
- `exe` ..... adresář se spustitelnými formami implementace
  - `Windows 7` ..... sestavení pod systémem Windows 7 x64
  - `Ubuntu 14` ..... sestavení pod systémem Ubuntu 14.04 x64
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `lib` ..... archívy s použitými knihovnami
  - `ref` ..... referenční implementace v MATLAB s daty
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Jakub_Drábek_2015.pdf` ..... text práce ve formátu PDF
  - `BP_Jakub_Drábek_2015.ps` ..... text práce ve formátu PS