Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Nástroj pro analýzu rádiových dat

Bc. Martin Jebavý

Diplomová práce

2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martin Jebavý**
Osobní číslo:	**I16220**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Nástroj pro analýzu rádiových dat**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Primárním cílem diplomové práce je návrh a implementace softwarového řešení, které bude umožňovat načítat rádiová data ze souboru a tyto zobrazovat v databázi pro následnou analýzu jednotlivých impulzů.
V rámci diplomové práce bude student ověřovat a dále zkoumat shodu mezi impulzy, přičemž jeden signál tvoří skupinu impulzů.
Nad načtenou databází se budou provádět selekce, které následně mají být zobrazeny v grafické podobě operátorovi. Pro testování GUI bude vytvořen algoritmus pro určování skupin impulzů.
Při zpracování diplomové práce bude student postupovat nejdříve návrhem aplikace pro čtení rádiových dat, dále návrh knihoven, tvorba databáze pro ukládání signálů a vytvoření základního formuláře v jazyce C# (WPF) pro zobrazování signálu (GUI). Závěrem autor provede ladění, testování zátěže a bezpečnosti.

Rozsah grafických prací:	**10**
Rozsah pracovní zprávy:	**60**
Forma zpracování diplomové práce:	**tištěná**

Seznam odborné literatury:

CONNOLLY, Thomas M. a Carolyn E. BEGG. Database systems: a practical approach to design, implementation, and management. 5th ed. Boston: Pearson Education International, c2010. ISBN 978-0-321-60110-0. HOFMAN, Jiří a Jan BAUER. Tajemství radiotechnického pátrače Tamara. Praha: Sdělovací technika, 2003. ISBN 80-86645-02-9. NATHAN, Adam. WPF 4 unleashed. Indianapolis, Ind.: Sams, c2010. Unleashed. ISBN 978-0-672-33119-0. PRICE, Jason. Mastering C# database programming. San Francisco: Sybex, c2003. ISBN 0782141838. PROKOP, Jiří. Algoritmy v jazyku C a C++. 3., aktualizované a rozšířené vydání. Praha: Grada, 2015. Průvodce (Grada). ISBN 978-80-247-5467-3. VIRIUS, Miroslav. Programovací jazyk C++. V Praze: České vysoké učení technické, 2017. ISBN 978-80-01-05961-6.

Vedoucí diplomové práce:	**Ing. Monika Borkovcová, Ph.D.** Katedra informačních technologií
Datum zadání diplomové práce:	**30. října 2017**
Termín odevzdání diplomové práce:	**18. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2017

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 15. 5. 2018

Martin Jebavý

PODĚKOVÁNÍ

Rád bych poděkoval Ing. Monice Borkovcové Ph.D. za poskytnutí užitečných rad, připomínek a možností k vypracování závěrečné práce. Dále bych chtěl poděkovat kolegům z práce, kteří mě byli nápomocni při řešení praktické části práce. V neposlední řadě bych chtěl poděkovat mé rodině a přítelkyni za podporu, kterou mi věnovaly během studia.

ANOTACE

Primární cíle diplomové práce se zabývají návrhem a implementací softwarového řešení, které bude umožňovat načítat rádiová data z binárního souboru a zpracovaná je ukládat v podobě impulsů do databáze pro následnou analýzu. V teoretické části práce jsou popsány jednotlivé technologie, na kterých bude postaveno softwarové řešení. Samotný popis technologií zahrnuje zejména definici jejich klíčových vlastností. Praktická část práce se věnuje samotnému návrhu a implementaci softwarového řešení za použití technologií zmíněných v teoretické části práce. Postupně budou navrženy knihovny pro načítání, analyzování a jednotné rozhraní. Přičemž podstatná část bude věnována navrhnutému algoritmu pro analýzu impulsů a grafickému rozhraní, které bude umožnovat zobrazovat tyto zpracované impulsy z databáze.

KLÍČOVÁ SLOVA

rádiová data, impuls, kontakt, analýza, algoritmus, databáze, knihovna, GUI

TITLE

Radio Data Analysis Tool

ANNOTATION

The primary aim of this master's thesis is to design and implement software solution that will be able to read radio data from its binary representation, process them and save them into a database for a consequential analysis. The theoretical part of this thesis describes technologies used while building the program with emphasis on their key features. The practical part illustrates construction and implementation of this software with the technologies mentioned. The significant part is devoted to the algorithm for analyzing impulses, which was essential for this thesis. Another important topic being described in this thesis is graphical user interface used to present given impulses from a database.

KEYWORDS

radio data, impulse, contact, analysis, algorithm, database, library, GUI

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

3D	Three-dimensional space
ACID	Atomicity Consistency Isolation Durability
ANSI	American National Standards Institute
API	Application Programming Interface
CCM	Current Contacts Map
CLR	Common Language Runtime
COM	Component Object Model
DPI	Dot Per Inch
DWM	Desktop Window Manager
GDI	Graphics Device Interface
GPU	Graphics Processing Unit
GUI	Graphical User Interface
ISO	International Organization for Standardization
JPEG	Joint Photographic Experts Group
MIL	Media Integration Layer
MS	Measuring Section
NARD	Nástroj Analýzy Radiových Dat
OOP	Object-oriented programming
PRI	Pulse Repetition Interval
RAM	Random-Access Memory
RDBMS	Relational Database Management Systém
SCM	Saved Contacts Map
SQL	Structured Query Language
STL	Standard Template Library
VS	Visual Studio
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language

ÚVOD

Primárním cílem diplomové práce je vytvoření softwarového řešení, které bude umožňovat načítat rádiová data z binárního souboru, analyzovat tyto data a uložit je do databáze pro zpětné zobrazení a vyhodnocení v grafickém rozhraní operátorovi. Přičemž stěžejní část práce se bude věnovat návrhu algoritmu, který bude umět analyzovat načtená data, a grafickému rozhraní, jenž operátorovi umožní zobrazit výsledek zpracování a poskytne mu jednoduchou sadu nástrojů pro vlastní analýzu. Pro zkonstruování softwarového řešení bude nutno použít různých technologií s ohledem na charakter jednotlivých funkcí, kterými bude ve výsledku disponovat.

Začátek teoretické části závěrečné práce se bude věnovat literární rešerši technologií, která se stane základem pro vyhotovení cílové aplikace. V návaznosti na rešerši následuje několik kapitol, které podrobněji popíší zmíněné technologie.

Kapitoly teoretické části budou zahrnovat technologie C++, WPF (C#), proces *marshaling* s C# a SQLite. Část o technologii C++ se bude zabývat zejména základními vlastnostmi technologie, filozofií, na čem je postaveno C++ a jaké moderní prvky obsahuje. Následná charakterizace technologie WPF obsahuje popis tohoto moderního grafického rozhraní. WPF bude zkoumat, proč je daná technologie lepší než klasická formulářová aplikace od Microsoftu, na jakém *enginu* jsou aplikace WPF postaveny, jakými vlastnostmi disponuje a jak používá značkovací jazyk XAML pro vyhotovení GUI. Druhá polovina teoretické části se bude věnovat procesu marshalingu s C#. U daného procesu budou popsány základní vlastnosti, co vlastně marshaling znamená a k čemu slouží. Poslední část popíše databázovou technologii SQLite. Budou představeny základní rysy, architektura a použití.

Kapitoly praktické části práce se budou věnovat vlastnímu návrhu a implementaci cílové aplikace. Úvodní pasáž objasňuje pochopení řešené problematiky, konkretizaci cíle řešení, využívané technologie v jednotlivých částech aplikace a důvod jejich začlenění do implementace. Následuje deskripce sekvence kroků postupu tvorby a samotné nasazení aplikace od načítání, zpracování, uložení až po samotné zobrazení rádiových dat operátorovi. Stěžejní částí výstupu diplomové práce bude návrh algoritmu pro analýzu načtených dat. Samotná kapitola bude popisovat GUI tedy grafické rozhraní aplikace včetně základního popisu manipulace s aplikací.

V závěru diplomové práce dojde k vyhodnocení navrženého a implementovaného softwarového řešení. K hlavním bodům hodnocení budou patřit zejména části, které se týkají návrhu algoritmu a grafického rozhraní. Na konec bude poukázáno, jaké nové poznatky diplomová práce přinesla.

1 LITERÁRNÍ REŠERŠE

Kapitola představí vybranou literaturu, která zahrnuje charakteristiku a použití různých softwarových technologií. Následující podkapitoly se budou věnovat využití programovacího jazyka C++ a WPF(C#). Pro ukládání a uchovávání dat se představí práce, která využívá koncept databáze SQLite.

Podrobné použití a využití zmíněných technologií budou rozebrány v dalších částech závěrečné práce.

1.1 C++

Autor knihy *„Mistrovství v C++ 4. aktualizované vydání“* Stephen Prata představuje v 18 kapitolách technologii C++. Veškeré teoretické popisy jsou opřeny o praktické ukázky. Kapitola popisuje rozdíly mezi jazykem C a C++, základní tvorbu struktury aplikace, vstupní a výstupní operace a používání proměnných. Zabývá se problematikou práce s daty, jak ukládat data do příslušných datový typů. Kniha postupně vysvětluje práci s objekty, třídami, šablonami, výjimkami, dědičností a dalšími neméně důležitými součástmi. Aktualizované vydání se zejména zabývá standardem C++11, kde se zaměřuje na nové změny v jazyce a příslušné novinky. [10]

Autor Scott Meyers ve své knize *„Effective modern C++: 42 specific ways to improve your use of C++11 and C++14“* se věnuje standardům jazyka C++11 a C++14, jako je deklarace automatického typu, sémantika, lambda a výrazy apod. Vysvětluje, jak správně a efektivně prvky zmíněných standardů, tj. pomocí moderního C++, používat, aby vytvořený software byl správný, umožňoval snadnou modifikaci a zejména byl přenositelný. Hlavní problematika se věnuje kladům a záporům zjednodušené inicializace, inteligentním ukazatelům a technikám správně napsaným lambda funkcím. V neposlední řadě se kniha věnuje, jaké jsou správné postupy pro revizi vývoje softwaru v postarším standardu jazyka (tj. C++98) do moderního C++. [8]

Kniha *„Programming: principles and practice using C++. Second edition“* se skládá ze čtyř částí a příloh. Bjarne Stroustrup představuje v první části knihy základní pojmy, techniky programování spojené s jazykem C ++ a potřebné knihovny k začátku psaní kódu. Zahrnuje používání aritmetických operací, řídících struktur a zpracování chyb. Déle nelze opomenout návrh, realizaci, použití funkcí a typů definovaných uživatelem. Druhá část popisuje způsob získávání číselných a textových dat z klávesnice a ze souboru. Poté ukazuje, jak prezentovat číselné údaje, texty a geometrické útvary jako grafické výstupy. Předposlední část se zaměřuje na zpracování

dat pomocí standardních knihoven C ++ a STL. Přibližuje problematiku implementace kontejnerů (např. vektor, seznam a mapa), dynamického přidělování paměti, použití ukazatelů, výjimek a šablon. Rovněž demonstruje návrhy a použití standardních knihovních algoritmů (např. třídění, vyhledávání apod.). Poslední část nabízí přehled o programování, diskuze o ideálním přístupu, historie, příklady (maticový výpočet, manipulace s textem, testování a programování vestavěných systémů) a stručný popis jazyka C. [11]

1.2 WPF (C#)

Matthew Macdonald představuje v knize „*Windows Presentation Foundation with .NET 4.5. Fourth edition"* moderní grafický systém WPF. Kniha slouží pro profesionální vývojáře WPF, kteří znají platformu .NET, jazyk C# a vývojového prostředí VS. Poskytuje úplný popis všech hlavních funkcí WPF, XAML (značkovací jazyk[1] používaný k definování uživatelského rozhraní WPF) na 3D kreslení a animaci. Věnuje se porovnání klasické formulářové aplikace Windows Form a WPF. Kniha má celkem 33 kapitol, jenž každá kapitola se věnuje jiné problematice. Začínají od základních funkcí WPF, jednotlivých použitelných grafických objektů, svázání s konkrétními daty (tabulka), využití animací, vkládání audio a video souboru, transformace objektů, vytváření grafických stylů, které lze použít na různé grafické objekty a atd. [7]

Kniha *„WPF 4.5 unleashed. Indianapolis"*, kterou napsal Adam Nathan, se zaobírá podobnou problematikou jako předchozí kniha. Pokrývá vše, co zahrnuje značkovací jazyk XAML. Zkoumá oblasti WPF v neuvěřitelné hloubce:

- ovládací prvky,
- rozvržení,
- zdroje a data,
- vazba a styling,
- grafika,
- animace a další.

[1] Způsob anotace dokumentu, který je syntakticky odlišitelný od textu.

Zdůrazňuje nejnovější funkce *multi-touch*[2], vylepšení vykreslování textu a další nové ovládací prvky. Rozděluje se na témata, která pokrývají 3D technologii, audio, video efekty a další. Zobrazuje, jak vytvářet oblíbené prvky uživatelského rozhraní (např. galerie, vlastní ovládací prvky apod.). Doplňuje o praktické ukázky a postupy použití funkcí a jednotlivých objektů WPF. [9]

1.3 SQLite

Jay Kreibich rozdělil svoji knihu *„Using SQLite“* do dvou hlavních částí, přičemž soubor kapitol se nachází v pořadí, jak by měla být kniha čtena. První dvě kapitoly poskytují podrobný pohled na SQLite a jeho použití. Třetí kapitola zahrnuje stahování a vytváření knihovny. Kapitoly čtyři a pět představují úvod do jazyka SQL. Šestá kapitola se zabývá koncepty návrhu databáze. Kapitola sedmá se zabývá základy C API a osmá kapitola pokrývá pokročilejších témata jako ukládání dat a časů, používání SQLite ze skriptovacích jazyků[3] a využívání pokročilejších rozšíření. Kapitoly devět a deset pokrývají psaní vlastních funkcí SQL, rozšíření a modulů. Následuje několik referenčních příloh pokrývající příkazy SQL. [5]

[2] Schopnost snímat více dotyků najednou.
[3] Programovací jazyk pro rychlý vývoj samostatných programů (např. JavaScript)

2 TECHNOLOGIE C++

Programovací jazyk C++ se stal jedním z nejdůležitějších jazyků na přelomu dvacátého století. Základy C++ jsou postaveny na jeho předchůdci jazyku C, který se prezentuje výkonem, ucelenostÍ, rychlostí a přenositelností. Objektově orientované vlastnosti poskytují jazyku C++ nové možnosti řešení zadaných úkolů v dnešní době. Jedním ze základních rozšíření je možnost využití šablon, která otevírají cestu k obecnému programování. Jazyk spadá mezi velmi výkonné, ale zároveň vyžaduje hlubší znalosti k jeho využívání.

Objektově orientovaná vlastnost patří mezi základní důvody k použití jazyka C++. Hlubší znalost základů ve standardu jazyka C vedou k využití této vlastnosti, protože jazyk C poskytuje základní datové typy, operátory, řídící struktury a syntaktická pravidla. [1], [10]

2.1 Filozofie jazyka C++

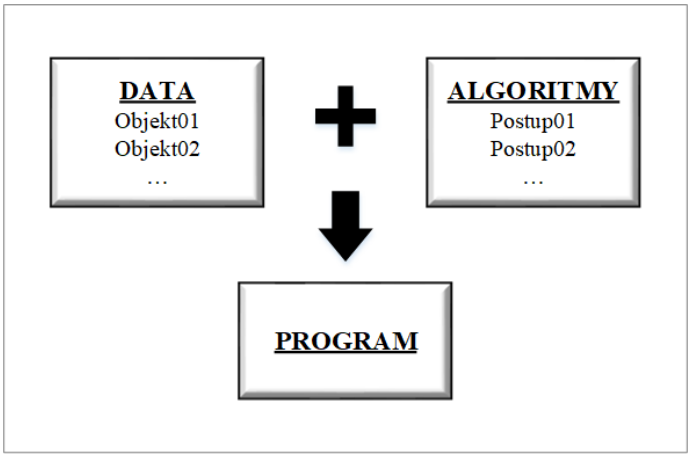
Se zvyšováním výkonu počítačů přichází vetší nároky na programy, který se tím stávaly složitějšími. Na základě inovování směřoval vývoj počítačových jazyků k jednodušší správě programovacího procesu. Jazyk C byl obohacen o řídící struktury a funkce. Přínosem jazyka C++ je podpora objektově orientovaného a generického programování kódu, který šetří čas a zvyšuje spolehlivost programů.

C++ umožňuje velký počet implementací na mnoha platformách, jehož základem je ISO/ANSI standard, který slouží jako základna pro jejich vzájemnou kompatibilitu. Definuje vlastnosti, které by měl jazyk mít, samotné chování a chování standardních knihoven (funkce, třídy, šablony apod.) Standard prezentuje jazyk jako přenositelný mezi různými počítačovými platformami s různými implementacemi. [1], [10]

2.1.1 Programování v jazyce C

C++ staví svoje základy a principy programování na jazyku C. Jazyk C patří mezi *procedurální*[4] tzn., že stejnou vlastností disponuje i novější jazyk C++. Obecně se počítačové jazyky zabývají daty a algoritmy. Data zahrnují informace, která program zpracovává a používá. Naproti algoritmy představují programem používané metody (viz. Obrázek 1).

[4] zdůrazňuje hledisko algoritmu programu.



Obrázek 1 - Data a algoritmy

Zdroj: přepracováno dle (PRATA, 2013)

Procedurální programování se skládá z činností, která by měl počítač provádět, a jejich implementace pomocí programovacího jazyka. Program popisuje posloupnost procedur, kterými se počítač řídí při tvorbě specifického výstupu. Starší procedurální jazyky (Frotran, Basic) trpěly problémy při vytváření rozsáhlejších problémů. Programy často využívaly větvení, které mělo za následek rozdělení programu na dvě části. Výsledkem dělení byl zamotaný průběh vykonávání, kde bylo těžké porozumět kódu programu a požadavek na modifikaci se mnohdy stával katastrofou. Nedostatek odstranil ukázněnější styl nazývaný *strukturované programování*.

Strukturované programování omezuje větvení (rozhodnutí o další zpracovávané instrukci) na malou množinu zvládnutelných konstrukcí. Jazyk C obsahuje konstrukce, které představuje například cyklus `for, while, do-while` a příkaz `if-else`.

Nový modernějším rozšířením je návrh stylu *shora dolů*. Rozdělení obsáhlejšího programu do menších zvládnutelnějších celků vystihuje hlavní myšlenku zmíněného stylu. Jestliže i po rozdělení je jeden z celků stále rozsáhlý, lze ho rozdělit do ještě menší celků. Postup se opakuje, dokud program neobsahuje pouze jednoduše programovatelné moduly. Jazyk C tento přístup realizuje programovatelnou jednotkou zvanou *funkce.* [1], [10]

2.1.2 Objektově orientované programování

Vlastnosti strukturovaného programování vylepšily srozumitelnost, spolehlivost a modularitu programů, nadále zůstává problém při programování obsáhlejších celků.

Objektově orientované programování (OOP) přináší nové způsoby k řešení problému programování rozsáhlejší celků programu. Od procedurálního programování, které se zaobírá zejména algoritmy, OOP přikládá váhu a důraz na samotná data. Usiluje o přednost, aby jazyk hlavně vyhovoval danému problému. Realizace spočívá v návrhu datových forem, které představují majoritní rysy problému V jazyce C++ existuje identifikátor `class`, který specifikuje nové formy dat. Tyto formy dat představují základ pro datovou strukturu nazývanou *objekt*. Obecně lze konstatovat, že třída definuje všechny potřebné údaje, jejíž realizací je objekt, který pracuje s těmito údaji. Postup, začínající od návrhu tříd po návrh programu se nazývá programování *zdola nahoru*.

OOP nabízí mnohem více než pouhé spojení dat a metod s definicí tříd, ale například poskytuje vytváření znovupoužitelného kódu. Skrývání informací před vnějškem zabezpečuje, aby se k datům přistupovalo nevhodným způsobem. Polymorfismus zpřístupňuje realizaci vícenásobných definic operátorů a funkcí, přičemž použití specifické definice závisí na aktuální souvislosti. Dědičnosti umožňuje odvozovat nové třídy od již vytvořených. Zmíněné pojmy zavedené OOP vyžaduje odlišný přístup k programování než u procedurálního programování.

Místo použití návrhu shora dolů je vhodné občas přistupovat k problému a jeho řeší s využitím návrhu zdola nahoru. Navržení užitečné, spolehlivé a implementačně správné třídy se stává mnohdy těžkým úkolem. Naštěstí OOP nabízí možnost do vyvíjeného programu začlenit již existující třídy. Jednou z hlavních výhod C++ je schopnost použít jednoduchým způsobem dobře otestovaný kód, který je součástí existujících knihoven. Knihovny jsou navrženy tak, aby usnadňovali vytváření nových programů. [1], [10]

2.1.3 Obecné programování

Mezi další programovací modely, podporovaným jazykem C++, patří *obecné programování.* Pojem obecný lze definovat jako vytváření typově nezávislého kódu. Jazyk C++ má množství datových typů (např. celá čísla, znaky apod.). Obecné programování rozšiřuje jazyk takovým způsobem, že lze napsat obecný předpis funkce pro neurčený datový typ. Za výhodu se považuje využití jedné funkce pro více datových typů. Nemusí se tedy vytvářet odlišné funkce pro

jednotlivý datový typ, kde by se funkce lišila pouze datovým typem nikoliv postupem a smyslem vykonávání. Obecnost se realizuje v jazyku C++ pomocí *šablon.*

S OOP spojuje společný cíl o vytváření jednoduššího znovupoužitelného kódu a technikou abstrakce obecných návrhů. OOP zdůrazňuje datové hledisko a poskytuje prostředky pro správu velkých programů. Obecné programování se věnuje zejména algoritmu a disponuje nástroji pro vykonávání běžných úkolů, jako je třídění dat nebo spojování seznamů. [1], [10]

2.2 Moderní standardy C++11 a C++14

Standardy C++ vylepšují jazyk o řadu nových funkcí. Některé nové možnosti mají za úkol zjednodušit pochopení jazyka C++, jeho učení a v neposlední řadě jeho použití.

Mezi základní příklady použití standardů je použití inicializace seznamem uvedených ve složených závorkách, automatické odvození datového typu pomocí funkce `auto`. Mezi další změny je zařazeno rozšíření a zpřehledňování tříd (např. specifikátory `override` a `final`).

Některá vylepšení zefektivňují programování i samotný výsledek. Lambda výrazy nabízí oproti klasickým funktorům a ukazatelům funkcí nové přednosti. Šablona `function` se používá ke snižování počtu nových instancí šablon. Pro lepší práci se zacházením s alokací paměti pomocí funkce `new` slouží šablony `unique_ptr` a `shared_ptr`.

Standardy přináší rozšíření knihoven a nové třídy STL. Šablony `tuple` a `regex` poskytují možnosti, jak vyřešit běžné programovací úkony. Mezi přínosy se řadí použitelnost a spolehlivost jazyka, které ovlivňují propagaci jazyka mezi programátory. [6], [8]

2.2.1 Terminologie a konvence

Pro správné pochopení výkladu C++ je důležité si ujasnit některou terminologii. Standardy jsou pojmenovány po roce, ve kterém byly přijaty (např. C++11, C++14). Názvy korespondují normám ISO. Tabulka 1 představuje používané terminologie, když se hovoří o jazyku C++, a jaké verze standardů jazyka spadají pro běžné pojmenování mezi programátory.

Tabulka 1 - Terminologie jazyka C++

Používané názvy	**Verze jazyka**
C++	Všechny
C++98	C++98 a C++03
C++11	**C++11 a C++14**
C++14	**C++14**

Zdroj: přepracováno dle (MEYERS, 2014)

C++ si zakládá na efektivnosti (platí pro všechny verze). Zmíněné verze v Tabulce 1 lze s polehčujícími okolnostmi popsat následovně:

- C++98 postrádá podporu souběžnosti (verze C++98 a C++03),
- C++11 podporuje lambda výrazy (verze C++11 a C+++14)
- a C++14 nabízí zobecněni návratového typu funkce.

Mezi nejhlavnější rysy C++11 patří pohyblivá sémantika, kde základ tvoří *r-hodnota* a *l-hodnota*. R-hodnoty označují objekty vhodné pro pohyb, zatímco l-hodnoty obecně nikoliv. V konceptu (ne vždy v praxi) odpovídají r-hodnoty dočasným objektům vrácených z funkcí, na které se lze odvolávat, buď podle jména, ukazatele nebo reference na l-hodnotu. Užitečnou heuristikou pro určení, zda se jedná o l-hodnotu, je požádat, jestli existuje možnost vzít si adresu l-hodnoty. Pokud možnost neexistuje jedná se obvykle o r-hodnotu. [6], [8]

2.2.2 Koncept auto

Auto vyjadřuje jednoduchý koncept, který je mnohem rafinovanější, než se může zprvu zdát. Jeho použití se šetří typování, ale také zabraňuje problémům, týkající se výkonu a správnosti určení datového typu, které mohou nastat při deklaraci datového typu manuálním způsobem. Přesto se vyskytují situace, kdy volba konceptu auto není vhodná, proto se nelze úplně vyhnout manuální deklaraci. [6], [8]

2.2.3 Lambda výrazy

Lambda výrazy přinášejí změnu v pojetí programování jazyka C++. Překvapující je fakt, že použití těchto výrazů nepřináší žádnou novou sílu do programování. Lambda má za úkol pohodlným způsobem vytvářet funkční objekty. Funkční objekty lze sestrojit i s trochou psaní bez použití lambdy. Použití v dnešní době má obrovský dopad na vývoj softwaru C++.

STL algoritmy, bez implementace lambdy, využívají nejzákladnější predikáty pro jejich funkčnost (hledání, odebírání apod.). Pokud jsou k dispozici, algoritmy se stávají netriviálními a lze měnit jejich původní mentalitu pro vykonávání. Mimo STL umožňuje lambda vytváření vlastních konstrukcí pro práci s chytrými ukazateli. [6], [8]

2.2.4 Chytré ukazatelé

Aby bylo možné pochopit, proč vznikly chytré ukazatele, je potřeba si přiblížit nepraktičnost prostých ukazatelů jazyka C++.

- Deklarace neindikuje, zda ukazatel odkazuje na jeden objekt nebo pole.
- Deklarace neumožňuje zjistit, zda lze ukazatel odstranit, když se stal pro další průběh
- Pokud se zjistí, že je potřeba ukazatel odstranit, neexistuje žádný způsob, jak mu to říci. K odstranění se musí explicitně použít `delete`.
- Může existovat situace, kdy bude nejasné, jakým způsobem bude potřeba odstranit ukazatele. Zda jako ukazatele na jeden objekt nebo na pole objektů. Pokud se použije nesprávná konstrukce, výsledky mohou skončit nedefinovaným způsobem.
- Obvykle neexistuje způsob, jak zjistit, na čem ukazatel závisí, tj. jestli ukazatel směruje na paměť, který již objekt nedrží (ukazuje do prázdna).

Prosté ukazatele se prezentují, jak silnými nástroji mohou být. Za desetiletí používání demonstrují, že jediný moment nepozornosti s jejich zacházeními, může vést ke složitému hledání a odstranění problému, který vznikl jejich špatnou manipulací.

Chytré ukazatelé jsou jedním z možných řešení těchto popsaných problémů. Tyto ukazatelé si lze představit jako obálky kolem prostých ukazatelů, které se chovají stejně jako prostý ukazatelé, ale poskytují možnost, jak se vyhnout mnoha úskalím. Chytré ukazatelé dokáží prakticky vše, co zvládnou prostý ukazatelé, ale s mnohem méně množství chyb. Proto jejich využití by mělo být maximální.

Existují čtyři inteligentní ukazatele `auto_ptr, unique_ptr,shared_ptr a weak_ptr` v jazyce C++ 11. Všechny ukazatelé fungují na principu, aby mohly spravovat dynamicky přidělené objekty během jejich existence, tj. zabránit úniků zdrojů tím, že zajistí vhodným způsobem zničení objektů (včetně kontroly výjimek), když se již stanou nepotřebnými pro další kontext.

Ukazatel `auto_ptr` je pozůstatkem z C++98. Byl to pokus o standardizaci toho, co se později přinesl `unique_ptr` v C++11. Pro provedení správné práce ukazatele je vyžadována pohyblivá sémantika, kterou C++98 nedisponovalo. Řešením se stalo `auto_ptr`, který spolupracuje s kopírovacími operacemi pro přesun. Vedlo to ke vzniku překvapivého kódu (kopírování nastavilo `auto_ptr` na hodnotu `null`) a k frustrujícímu omezení použití (např. není možné ukládat `auto_ptr` do kontejnerů).

Inteligentní ukazatel `unique_ptr` umí vše plus něco navíc jako `auto_ptr`. Pracuje stejně efektivně, ale bez deformace při kopírování). Je lepší ve všech ohledech už jen tím, že k použití `auto_ptr` se potřebuje kompilovat s kompilátory C++98. Pokud neexistuje omezení kompilace s kompilátory C++98, je vhodné se vyhnout použití `auto_ptr`.

C++11 přináší sdílené ukazatelé `shared_ptr`, které spravují objekt prostřednictvím sdíleného vlastnictví. Žádný specifický sdílený ukazatel nevlastní objekt, místo toho všechny směřují ke spoluprací, aby zajistily jeho zničení v místě, kde objekt již není potřeba. Když poslední ukazatel přestane ukazovat na objekt (např. ukazatel je zničen nebo začne ukazovat na jiný objekt), zničí objekt, na který odkazoval. Závěrem lze konstatovat, že sdílené ukazatelé nabízejí pohodlí, pří řízení a úklidu libovolných zdrojů. Ve srovnání s `unique_ptr`, jsou `shared_ptr` dvakrát větší a vykazují větší režijní náklady na řídící bloky.

Jestliže existuje situace, kdy se ukazatelé mohou prolínat, je vhodné použít `weak_ptr`. Ukazatel `weak_ptr` nabízí možnosti ukládání do mezipaměti, seznamy pozorovatelů a prevenci cyklu sdílených ukazatelů `shared_ptr`. [6], [8]

2.2.5 Přechod na moderní C++

Pokud jde o inovativní a moderní funkce, nelze opomenout velmi přínosné standardy C++11 a C++14. Auto, inteligentní ukazatelé, pohyblivá sémantika a lambda je výčet funkcí, které jsou nezbytné proto, jak se stát efektivním moderním programátorem C++.

Při vytváření nových objektů se rozlišuje úhel pohledu, kdy inicializace objektu C++11 ztělesňuje buď přínos nebo matoucí nepořádek. Inicializační hodnoty objektu mohou být specifikovány pomocí závorek, znaménkem rovnosti nebo složenými závorkami. Složené závorky jsou nejpopulárnější inicializační syntaxe, která zabraňuje zúžení konverzí a imunitu vůči nejproblémovější analýze C++. Významný rozdíl volby mezi klasickými závorkami a složenými závorkami může nastat při tvorbě *vektoru* numerického typu s dvěma argumenty, kdy jejich jednotlivé použití může mít odlišný význam. Výběr mezi těmi to typy závorek, může být velmi náročný při vytváření objektů uvnitř šablon.

Mezi další moderní přístupy se považuje preferování `nullptr` před hodnotou `0` nebo `NULL`. Podstatným poznatkem je vyvarování se přetížení numerických typů a ukazatelů.

Mezi další moderní přístupy se považuje:

- preferování `nullptr` před hodnotou `0` nebo `NULL`,
- upřednostnění deklarace *aliasu* před `typedef`,
- upřednostnění uzavřeného rozsahu *enumu* před neuzavřeným,
- deklarace přepsání nadřazených fuknací,
- preferování konstantních *iterátororů* před běžnými nekonstantními *iterátory,*
- deklarace funkce bez výjimek (pokud ji funkce nebude vyvolávat)
- a používání `constexpr` tam, kde to bude jen možné. [6], [8]

3 WPF (C#)

Windows Presentation Foundation (WPF) je moderní grafický zobrazovací systém pro platformu Windows. Přináší radikální změnu od předešlých technologiích (např. klasická formulářová aplikace C#). Zabudovaná hardwarová akcelerace a nezávislost řešení vnáší inovativní funkce do grafického vývoje aplikací. WPF je nejlepším nástrojem k vytvoření bohaté desktopové aplikace pro systémy Windows (např. Vista, 7, 8, 10). Ve skutečnosti nabízí jedinečnou univerzální sadu nástrojů, zaměřené na různé typy verzí systému Windows. Aplikace WPF lze dokonce spustit na počítačích, kde stálé běží systém Windows XP. [7], [12]

3.1 DirectX: Nový grafický engine

Před WPF Windows vývojáři téměř 15 let používali stejnou technologii zobrazení, která závisela na dvou technologiích operačního systému Windows k vytvoření uživatelského rozhraní (GUI):

- User32
- a GDI/GDI+.

User32 poskytuje tradiční vzhled Windows pro prvky, jako jsou okna, tlačítka, textová pole apod. Kdež to GDI nebo jeho rozšířená verze GDI+, poskytuje podporu kreslení pro vykreslování tvarů, textů a obrázků za cenu dodatečné složitosti (často s nedostatečným výkonem). V průběhu let byly obě technologie vylepšeny a API, kterou vývojáři používají k interakci s nimi, se dramaticky změnily. [7], [12]

3.1.1 Vývoj DirectX

Společnost Microsoft vytvořila DirectX[5], jako novou cestu kolem omezení, které přinášely knihovny User32 a GDI/GDI+. DirectX začal jako nástroj pro vytváření her na platformě Windows. Společnost Microsoft úzce spolupracovala s dodavateli grafických karet, aby pro DirectX poskytla hardwarovou akceleraci potřebnou pro složité textury, speciální efekty, jako je čás-

[5] sada knihoven poskytujících aplikační rozhraní přímého ovládání moderního hardwaru

tečná průhlednost a trojrozměrná grafika. DirectX je mnohem účinnější, protože vykresluje grafické objekty (např. textury) přímo grafickou kartou. GDI/GDI+ potřebuje pro vykreslení převést pixel po pixelu, což je pro nové moderní grafické karty mnohem pomalejší.

V dnešní době je DirectX nedílnou součástí systému Windows s podporou všech moderních grafických karet. Programovací API pro rozhraní DirectX slouží zejména jako sada nástrojů pro vývojáře her. Z důvodu složitosti se DirectX nepoužívá v tradičních Windows aplikací (např. podnikový software). [7], [12]

3.1.2 DirectX ve WPF

WPF přinesl velký převrat. V grafické technologii WPF již není technologie GDI/GDI+, ale na místo toho je použit DirectX. Bez ohledu na charakter vyvíjeného uživatelského rozhraní, aplikace vždy spolupracuje s DirectX. Znamená to, že veškerá práce prochází přes DirectX, ať už se jedná o trojrozměrnou grafiku či prostá tlačítka. Výsledkem je, že i nejrozsáhlejší podnikové aplikace mohou využívat propracované efekty (např. transparentnost, přítomnost *anti-aliasing*[6] apod.). WPF překvapivě nadále spoléhá v omezeném rozsahu na knihovnu User32. Vyžaduje ji pro určité služby, jako je manipulace a směrování vstupů, třídění výstupů apod. Nicméně, veškeré vykreslování má na starosti DirectX.

Velký benefit přináší i hardwarová akcelerace. Jednoduše lze definovat, že DirectX se snaží odevzdat co nejvíce práce jednotce GPU. WPF disponuje dostatečnou inteligencí, aby co nejlépe využívala optimalizaci hardwaru, ale v případě potřeby může provádět stejnou práci s pomocí softwarových výpočtů. Pokud se spustí aplikace WPF v počítači se starší grafikou, rozhraní se bude stále zobrazovat tak, jak bylo navrženo. Alternativa softwarových výpočtů bude pomalejší na počítačích se staršími grafickými kartami v případě, kdy WPF aplikace obsahuje složitější animace nebo jiné propracované grafické efekty. [7], [12]

[6] technologie zpracování obrazu, sloužící k vyhlazování hran 3D grafiky

3.2 API vyšší úrovně

WPF nenabízí pouze hardwarovou akceleraci prostřednictvím DirectX, ale obsahuje i pro programátory aplikací velmi kvalitní služby. Níže budou představeny některé z nejvíce dramatických změn, které WPF přineslo do programovacího světa Windows:

- webový model rozvržení,
- bohatý kreslící model,
- kvalitní textový model,
- animace prvotřídního konceptu,
- podpora pro audio a video média,
- styly a šablony,
- příkazy,
- deklarativní uživatelské rozhraní,
- a stránkově založená aplikace.

Webový model rozvržení zdůrazňuje flexibilní tokové uspořádání, které upravuje ovládací prvky založené na jejich obsahu. Výsledkem je uživatelské rozhraní, které se přizpůsobí vysoce dynamickému obsahu.

Kreslící model se zabývá zejména primitivy jako základní tvary, bloky textu, a další grafické prvky. K dispozici jsou takové funkce, jako například opravdové transparentní prvky, schopnost vrstvit na sebe několik vrstev s různým stupněm krytí a nativní podpora 3D.

Textový model dává Windows aplikacím možnost jedinečně zobrazovat stylovaný text kdekoliv v uživatelském rozhraní. Umožňuje dokonce kombinace textu se seznamy, plovoucími čísly a dalšími prvky uživatelského rozhraní. Pokud nastane potřeba zobrazit velké množství textu, lze použít pokročilé funkce zobrazení dokumentů pro zlepšení čitelnosti, jako zabalení, sloupce a zarovnání.

Animační koncept nevyužívá časovač, který by donutil vyobrazený formulář v aplikaci k překreslení. Místo toho je podstatnou součástí frameworku. Definuje animace s deklarativními tagy, které následně WPF automaticky aktivuje a dojde k překreslení.

Nástroje pro podporu audio a video média byly ve starším uživatelském rozhraní (Windows Forms) překvapivě omezeny při práci s multimédii. WPF však obsahuje podporu přehrávání

jakéhokoliv zvukového či obrazového souboru, který lze přehrát v přehrávači Windows Media Player. Další výhodou nabývá možnost přehrání více než jednoho mediálního souboru najednou. Dokonce působivěji se nabízí možnost nástroje pro integraci video obsahu do zbytku uživatelského rozhraní (aplikace propracovaných triků).

Styly umožňují standardizovat formátování a znovu jej použít ve vyvíjené aplikaci. Šablony nabízí způsob měnit vykreslování libovolného prvku (např. tlačítko). Styly a šablony dopřávají komfort a jednoduchost při vytváření moderního rozhraní.

K dispozici pro vyvíjený kód je možnost abstrakce, kde lze definovat příkazy aplikace na jednom místě a propojit jej s dalšími ovládacími prvky. Většina uživatelů si uvědomuje, že nezaleží na tom, zda spouštějí příkaz *Otevřít* prostřednictvím nabídky nebo pomocí panelu nástrojů, pokud je konečný výsledek akce stejný.

Ačkoliv se nabízí možnost vytvořit okno WPF pomocí kódu, vývojové prostředí Visual Studio od Microsftu používá jiný přístup. Serializuje obsah každého okna do sady tagů XML v dokumentu XAML. Výhoda spočívá v tom, že uživatelské rozhraní je zcela odděleno od implementovaného kódu a grafický návrháři mohou použít profesionální nástroje k úpravě XAML dokumentů pro vznik kvalitního *front endu* aplikace.

Pomocí aplikace WPF se vytvářejí aplikace podobné prohlížeči, který umožní pohybovat se přes kolekci stránek, doplněnou o navigační tlačítka vpřed a vzad. WPF dokáže zpracovat chaotické detaily (např. historie stránek). Dokonce se může vytvořená aplikace, založená na prohlížeči, nasadit přímo v aplikace Internet Explorer[7]. [7], [12]

3.3 Nezávislost rozlišení

Tradiční aplikace systému Windows jsou svázány s určitými předpoklady o rozlišení. Vývojáři obvykle předpokládají se standardním rozlišením monitoru (např. 1366 x 768 pixelů). Navrhuji okna s ohledem na rozlišením monitoru a snaží se zajistit případné přiměřené modifikace velikosti při menším nebo větším rozlišení.

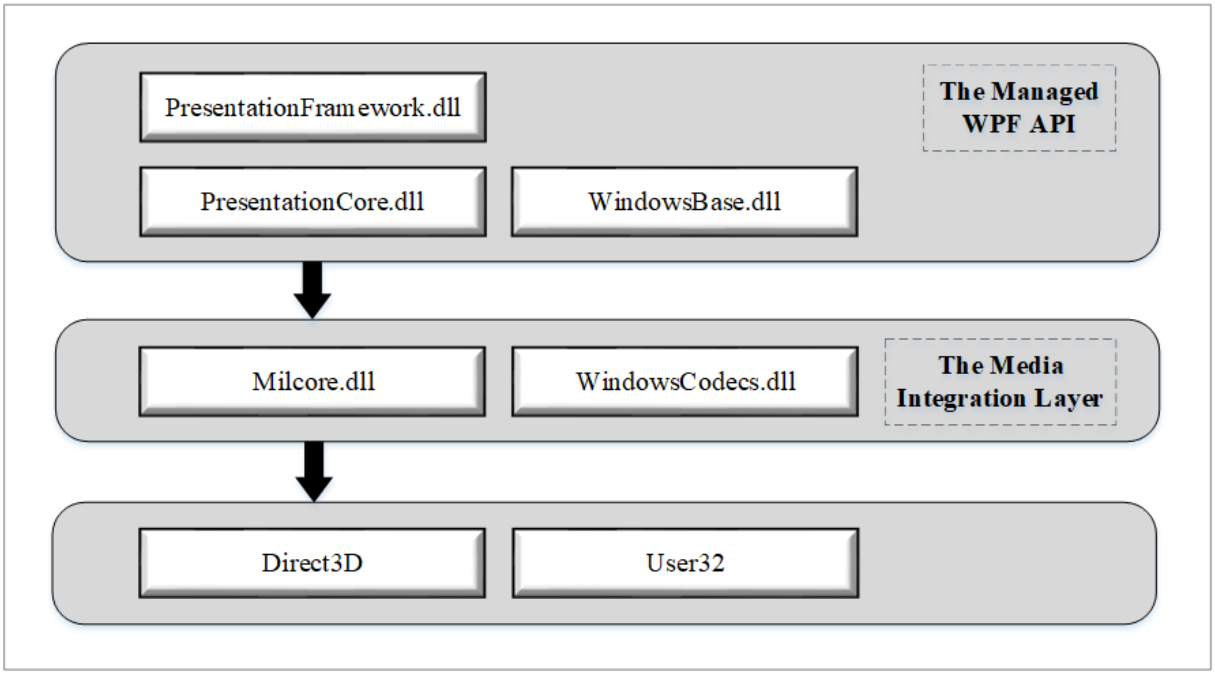
Uživatelská rozhraní obyčejných aplikací Windows nedisponují škálovatelností. Výsledkem je, že pokud se použije rozlišení s vysokým rozlišením, okna aplikace budou při zmenšení hůře čitelné.

[7] Základní internetový prohlížeč systému Windows

WPF disponuje systémem DPI pro výpočet velikosti rozlišení a řeší problémy při nasazení aplikace na různá zařízení s odlišnými rozlišeními. Systém DPI se používá k nezávislosti rozlišení vyobrazovaných objektů na různých zařízeních. Například tlačítko pracuje stejně jako jiné prvky uživatelského rozhraní v libovolném typu aplikací systému Windows. Rozdíl ve výsledku přináší systém DPI, který ovlivňuje velikost písma systému, ale často ponechává jiné detaily nezměněné. [7], [12]

3.4 Architektura

WPF používá vícevrstvou architekturu. V horní části spolupracuje s vysokou úrovní nabízených služeb, kde kód je zcela napsán v programovacím jazyce C# jako *managed*. Skutečná práce pro překladní .NET objektů do Direct3D objektů (např. textury) se děje v pozadí pomocí nižší úrovně nazvané *Milcore.dll*. Milcore.dll je implementován v *unmanaged* kódu, protože potřebuje těsnou integraci s Direct3D a je velmi citlivý na výkon. Obrázek 2 ukazuje práci jednotlivých vrstev ve WPF aplikaci.



Obrázek 2 - Architektura WPF

Zdroj: přepracováno dle (MACDONALD, 2012)

Obrázek 2 zahrnuje klíčové komponenty jako:

- *PresentationFramework.dll,*
- *PresentationCore.dll,*
- *WindowsBase.dll,*
- *Milcore.dll,*
- *WindowsCodecs.dll,*
- *Direct3D*
- a *User32.*

PresentationFramework.dll obsahuje prvky na nejvyšší úrovni, které přestavují například okna, panely a další typy ovládacích prvků. Odsud pochází většina tříd, která se používají při vývoji WPF aplikace. Rovněž provádí vysokou abstrakci programování (použití stylů).

PresentationCore.dll obsahuje základní typy prvků (např. *UIElement, Visual*), ze kterých jsou odvozeny všechny tvary a ovládací prvky.

WindowsBase.dll obsahuje ještě více základních prvků, které mohou být použity mimo WPF (*DispatcherObject* a *DependencyObject*).

Milcore.dll je jádrem systému pro vykreslování objektů ve WPF. Představuje základní vrstvu pro integraci medií (MIL). Jedná se o ucelený motor pro překlad vizuálních prvků do trojúhelníků a textur, které Direct3D očekává. Milcore.dll není pouze součástí WPF, ale i základní součást systému Windows Vista a Windows 7, který je používán k vykreslení plochy pomocí nástroje DWM.

WindowsCodecs.dll představuje API na nižší úrovni. která poskytuje podporu pro zobrazování (např. zpracování a zobrazení obrázků typu JPEG).

Nízko úrovňová API Direct3D má za úkol vykreslování všech grafických prvků ve WPF. Nejdůležitějším faktem je uvědomění, že nezáleží na tom, zda se používá starší nebo mnohem modernější grafická karta. Vykreslení základních nebo složitějších prvků, běžících na systémech Windows XP, Windows Vista nebo Windows 7, bude stejné.

User32 se používá k určení, které služby program využívá. Není součástí vykreslování běžných ovládacích prvků. [7], [12]

3.5 XAML

XAML je značkovací jazyk používaný k vytváření instancí objektů .NET. Ačkoli XAML technologie může být aplikována na mnoho problémových domén, její primární role je vytváření uživatelských rozhraní WPF. Dokumenty XAML definují uspořádání panelů, tlačítek a ovládacích prvků, které mohou být v aplikaci obsaženy. XAML se nemusí psát pouze ručně, ale lze k jeho vygenerování použít příslušný nástroj. Microsoft Expression Blend představuje grafický návrhový program určený pro grafické designery. Pro běžné vývojáři slouží zejména nástroj Microsoft Visual Studio. Vzhledem k tomu, že oba nástroje mohou vyvíjet grafiku v XAML, lze integrovat pracovní postup mezi vývojáři a designery, kde vývojář vytvoří základní uživatelské prostředí ve Visual Studiu a designer dodá vylepšenou grafiku pomocí nástroje Expression Blend. Tato provázanost spolupráce při vyvíjení aplikace je jedním z hlavních důvodu, proč společnost Microsoft vytvořila XAML. [4], [7]

3.5.1 Grafické uživatelské rozhraní

Při projektování aplikace WPF ve Visual Studiu se XAML kód nepřekládá, jako u běžné Windows Form aplikace. Místo toho je serializován do sady značek, které se při spuštění používají k vygenerování objektů uživatelského rozhraní.

Standard XAML je zcela jednoduchý, jehož základy spočívají v pochopení několika základních pravidel:

- Každý prvek v dokumentu XAML mapuje instanci třídy .NET.
- Název prvku odpovídá přesně názvu třídy.
- Stejně jako u jakéhokoliv dokument XML, lze umístit jeden element do jiného.
- XAML dává každé třídě flexibilitu při rozhodování, jak řešit danou situaci.
- Vlastnosti každé třídy se mohou nastavit pomocí atributů.
- V některých situacích, kdy atribut není dostačující, je nutno použít vnořené tagy se speciální syntaxí. [4], [7]

4 MARSHALING S C#

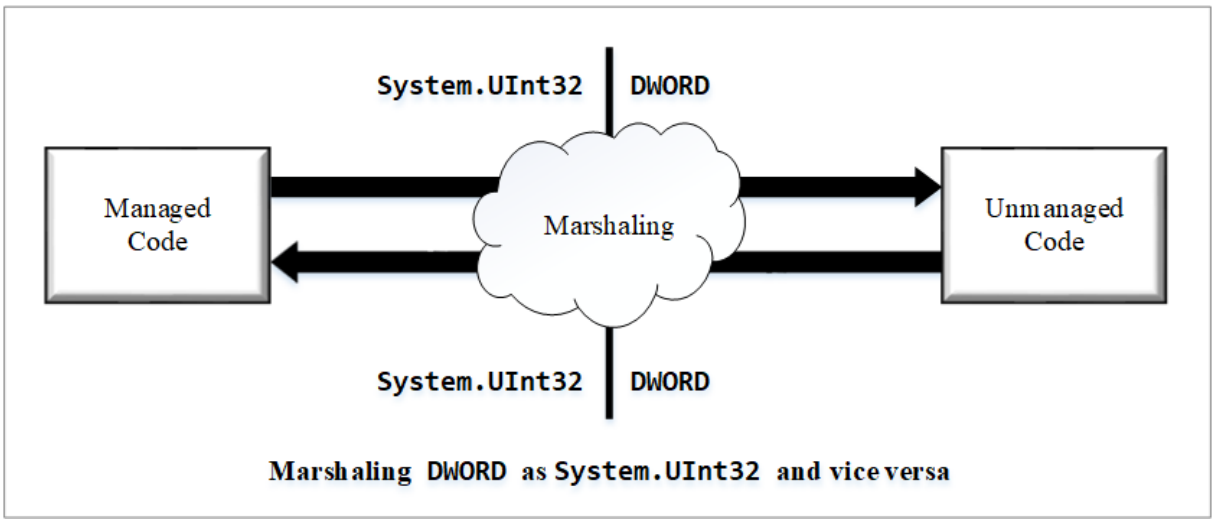
Marshaling je proces vytvoření mostu mezi *managed* a *unmanaged* kódem. Dochází k obousměrnému přenášení zpráv mezi oběma typy kódu. Jedná se o jednu z hlavních služeb nabízených CLR.

Mezi managed a unmanaged prostředím neexistuje kompatabilita. Jinými slovy .NET neobsahuje typy jako `HRESULT`, `DWORD` a `HANDLE`, které existují v oblasti unmanaged kódu. Proto je nutné najít náhradu v .NET nebo vytvořit vlastní, pokud bude potřeba. K tomu slouží právě marshaling.

Marshaling je užitečný, když se pracuje s unmanaged kódem, ať už se jedná o komponenty Windows API nebo COM. Pomáhá správně spolupracovat a sdílet data mezi oběma prostředími. Obrázek 3 ukazuje příklad funkce procesu marshaling.

Marshaling rozděluje datové typy do dvou kategorií:

- jednoduché
- a složené. [3]



Obrázek 3 - Proces marshaling

Zdroj: přepracováno dle (ELSHEIMY, 2010)

4.1 Jednoduché typy

Jednoduché typy (např. celá čísla, *booleany* atd.) jsou ty, které nejsou odvozeny neboli definovány z jiných datových typů. Tyto typy se považují za základ pro všechny ostatní typy. Příkladem managed primitivních datových typů se pokládají:

- *System.Byte,*
- *System.Int32,*
- *System.UInt32,*
- *System.Double,*
- *System.Char,*
- *System.String*
- a *System.IntPtr*. [3]

4.1.1 Datový typ Blittable

Většina datových typů má společnou reprezentaci v managed i unmanaged paměti a nevyžaduje speciální zpracování. Tyto typy se nazývají *blittable,* protože při předávání mezi managed a unmanaged kódem nevyžadují zvláštní manipulaci. Jiné typy, které vyžadují speciální manipulaci, se nazývají *Non-Blittable*. Dalo by se říci, že většina jednoduchých typů jsou Blittable a všechny složené Non-Blittable.

Následující Tabulka 2 uvádí seznam typů Blittable, které se vyskytují v rozhraní .NET.

Tabulka 2 - Typy Blittable

Managed Type	Description
`System.SByte`	8-bit signed integer
`System.Byte`	8-bit unsigned integer
`System.Int16`	16-bit signed integer
`System.UInt16`	16-bit unsigned integer
`System.Int32`	32-bit signed integer

System.UInt32	32-bit unsigned integer
System.Int64	64-bit signed integer
System.UInt64	64-bit unsigned integer
System.IntPtr	Signed pointer
System.UIntPtr	Unsigned pointer

Zdroj: přepracováno dle (ELSHEIMY, 2010)

V další Tabulce 3 jsou uvedeny některé unmanaged numerické datové typy ve Windows, jejich klíčová slova v C/C++ a jejich protějšky (marshaling typy) v .NET. [3]

Tabulka 3 - Numerické datové typy

Description	Windows Type	C/C++ Keyword	Managed Type	C# Keyword
8-bit signed integer	CHAR	char	System.SByte	sbyte
8-bit unsigned integer	BYTE	unsigned char	System.Byte	byte
16-bit signed integer	SHORT	Short	System.Int16	short
16-bit unsigned integer	WORD, USHORT	unsigned short	System.UInt16	ushort
32-bit signed integer	INT, INT32, LONG, LONG32	int, long	System.Int32	int
32-bit unsigned integer	DWORD, DWORD32, UINT, UINT32	unsigned int, unsigned long	System.UInt32	uint

64-bit signed integer	INT64, INT32, LONG, LONG32	_int64, long long	System.Int64	long
64-bit unsigned integer	DWORDLONG, DWORD64, ULONGLONG, UINT64	unsigned _int64, unsigned long long	System.UInt64	ulong
Floating-point integer	FLOAT	float	System.Double	double

Zdroj: přepracováno dle (ELSHEIMY, 2010)

4.1.2 Datový typ Non-Blittable

Vedle numerických datových typů je potřeba vědět, jak nakládat s unmanaged textovými datovými typy. Avšak dané typy jsou Non-Blittable a vyžadují tak zvláštní manipulaci. Následující Tabulka 4 uvádí stručně unmanaged textové datové typy. [3]

Tabulka 4 - Textové datové typy

Description	Unmanaged Type(s)	Managed Type
8-bit ANSI character	CHAR	System.Char
16-bit Unicode character	WCHAR	System.Char
8-bit ANSI string of characters	LPSTR, LPCSTR, PCSTR, PSTR	System.String
16-bit Unicode string of characters	LPCWSTR, LPWSTR, PCWSTR, PWSTR	System.String

Zdroj: přepracováno dle (ELSHEIMY, 2010)

4.2 Složené typy

Složené typy (struktury a třídy) představují takové typy, které vyžadují speciální manipulaci a jsou odvozeny z jiných typů. Vytvářejí jiné datové typy tím, že zapouzdřují jednoduché typy a jiné složené typy.

Složené typy zapouzdřují do sebe související data, poskytuje organizovaný a uspořádaný kontejner pro přenos skupin proměnných mezi klientskou aplikací a unmanaged serverem. Obvykle obsahuje proměnné jednoduchých typů a volitelně jiných složených typů. Navíc mohou uvnitř definovat i jiné složené typy.

Složené typy se představují ve dvou kategoriích:

- unmanaged struktury
- a unmanaged *unions.* [3]

4.2.1 Unmanaged struktury

Pomocí marshalingu mohou vystupovat unmanaged struktury jako managed struktury nebo dokonce i třídy. Rozhodnutí, zda použít managed strukturu nebo třídu, závisí na řešené problematice.

Při mashalingu struktur v managed prostředí je třeba vzít v potaz, že zatímco se přistupuje k proměnné pomocí jejího názvu, Windows přistupuje k proměnné prostřednictvím adresy (tj. pozice) uvnitř paměti. Nezáleží mu na názvu, ale zaobírá se umístěním a velikostí. Na základě přístupu je rozložení paměti a velikosti typu nesmírně důležité.

V následujících několika krocích lze marshalovat unmanaged strukturu:

- Vytvořit mashaling typ managed struktury nebo třídy.
- Přidat definované proměnné. Uspořádání a velikosti datového typu proměnné jsou velmi důležité. Proměnné musí být upořádány tak, jak jsou definovány v unmanaged struktuře, aby k nim Windows mohl správně přistupovat.
- Výslednou strukturu nebo třídu o anotovat o atribut `StructLayoutAttribute`. [3]

4.2.2 Unmanaged unions

Union představuje paměťové místo, které sdílí dva a více různých typů proměnných. Poskytuje způsob interpretace stejného bitového vzoru dvěma nebo více různými způsoby (nebo formami).

Ve skutečnosti, unie sdílejí se strukturami plno vlastností, jako je způsob definování a marshalovaní. Stejně jako struktury, mohou být unie definovány uvnitř struktury nebo dokonce jako jedna entita. Kromě toho mohou unie uvnitř definovat složené typy stejně jako struktury.

Použití unie je velmi efektivní ve chvíli, když přetypování mezi datovými typy vyžaduje mnoho režie.

Marshalování unie používá stejný způsob jako při marshalovaní struktury. Kvůli způsobu ukládání unie do paměti, bude navíc nutné uvnitř explicitně nastavit pozici proměnné.

Následující pokyny představují postup marshalování unie:

- Vytvořit mashaling typ managed struktury nebo třídy.
- Anotace typu o atribut `StructLayoutAttribute` a `LayoutKind.Explicit`, který specifikuje explicitní druh rozvržení.
- Přidat pouze definované proměnné. Protože rozvržení se specifikuje explicitně, pořadí jednotlivých proměnných není důležité.
- O anotovat každou proměnnou o atribut `FieldOffsetAttribute`, který určuje absolutní polohu v bytech od začátku managed struktury nebo třídy. [3]

5 SQLITE

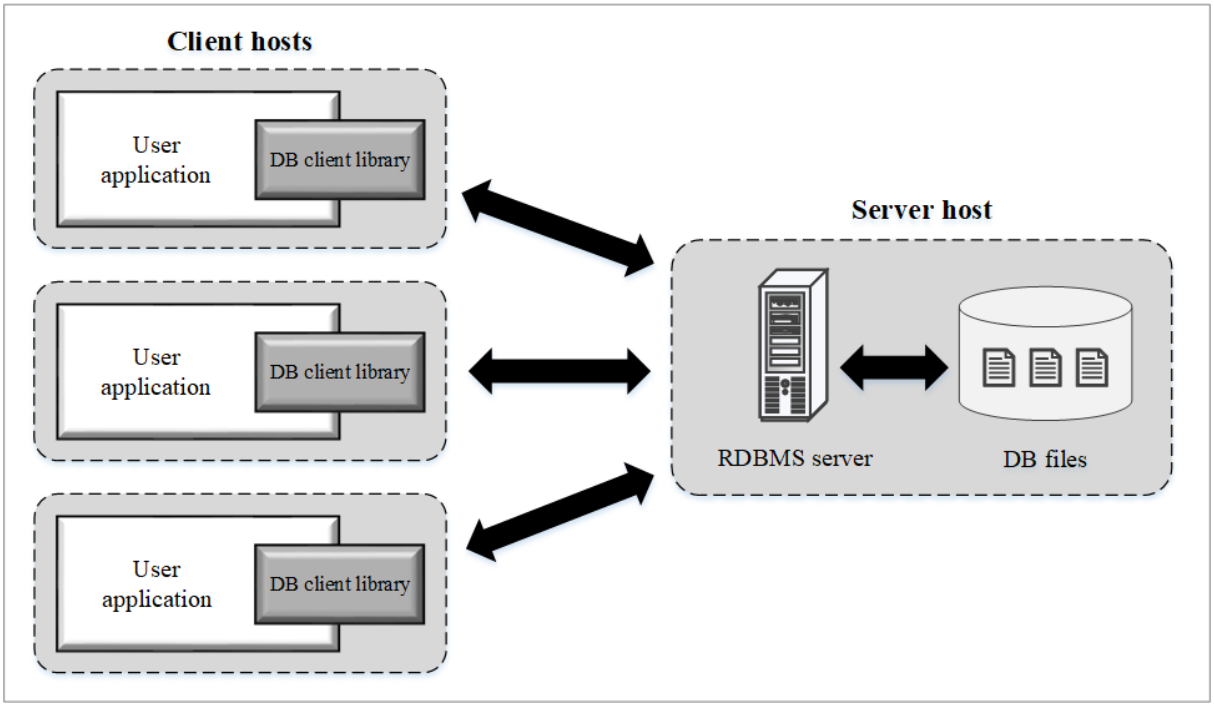
SQLite je softwarový balík pro veřejnou doménu, který poskytuje systém pro správu relačních databází (RDBMS). Relační databázové systémy slouží k ukládání uživatelsky definovaných záznamů do velkých tabulek. Vedle ukládání a správy dat může databázový stroj zpracovávat složité dotazy, které kombinují data z více tabulek ke generování reportů a sumarizačních dat. „*Lite*“ v názvu SQLite neodkazuje na jeho schopnosti, nýbrž ukazuje na jednoduchost jeho nastavení, administrativní režii a využití zdrojů. Poskytuje velmi funkční a flexibilní relační databázové prostředí, které spotřebovává minimální zdroje a vytváří minimální potíže pro vývojáře a uživatele.

SQLite je definován následujícími rysy:

- *Serverless* – SQLite nevyžaduje k práci samostatný serverový proces nebo systém. Přistupuje přímo k uloženým souborům.
- *Zero Configuration* – Žádný server znamená žádné nastavení. Vytvoření instance SQLite databáze je stejně jednoduché jako otevření souboru.
- *Cross-Platform* – Celá instance databáze je umístěna v jediném cross-platform souboru bez nutnosti správy.
- *Self-Contained* – Jediná knihovna obsahuje celý databázový systém, který se integruje přímo do hostitelské aplikace.
- *Small Runtime Footprint* – Výchozí *build* je menší než megabajt kódu a vyžaduje pouze několik megabytů paměti. Při některých úpravách lze značně snížit velikost i využití paměti knihovny.
- *Transactional* - Transakce SQLite jsou plně kompatibilní s ACID, což umožňuje bezpečný přístup z více procesů nebo vláken.
- *Full-Featured* – SQLite podporuje většinu funkcí dotazovacího jazyka, které se nacházejí ve standardu SQL92 (SQL2).
- *Highly Reliable* – Vývojový tým SQLite provádí velmi pečlivé ověřování kódu. [2], [5]

5.1 Architektura

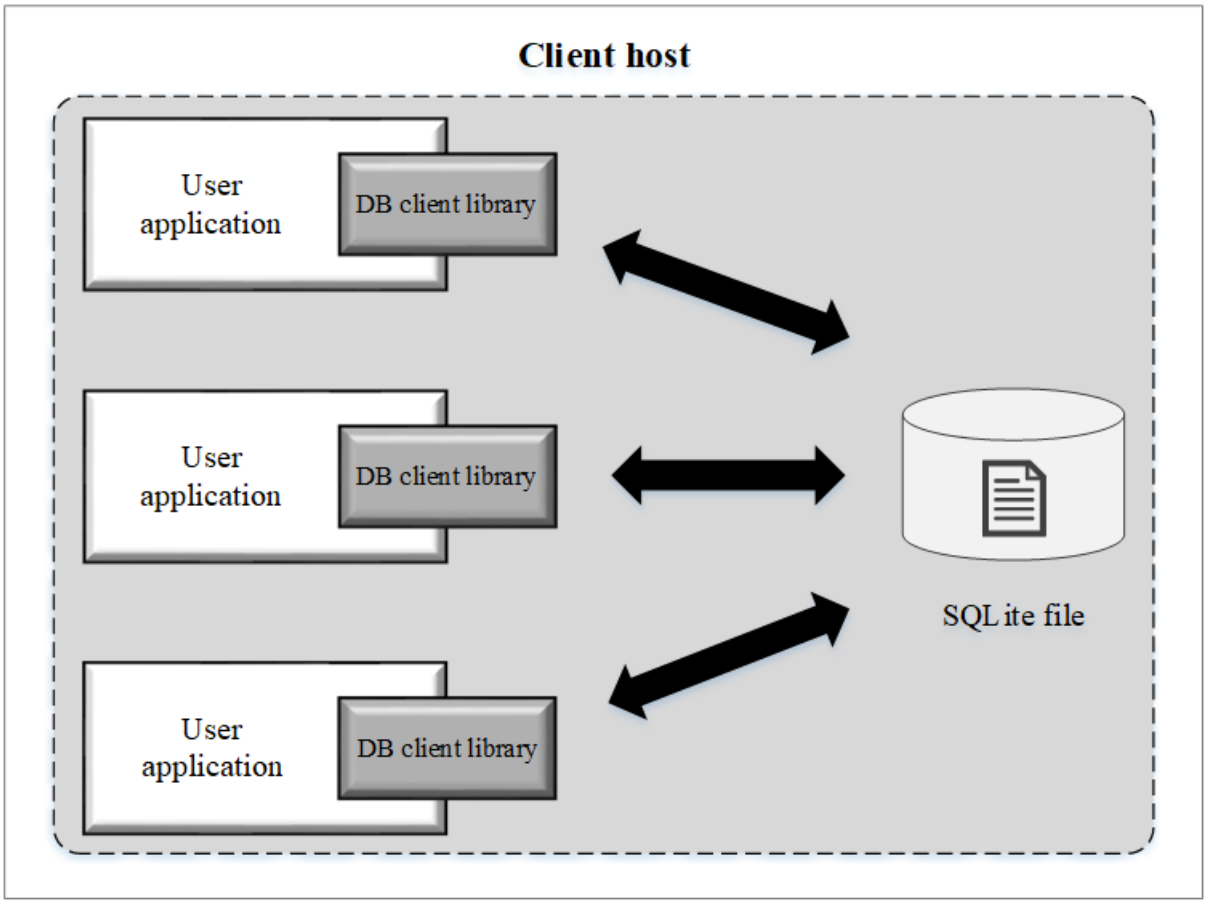
Většina rozsáhlých databázových systémů (RDBMS) obsahuje velký serverový balíček, který tvoří databázový stroj. Pro přístup k databázi jsou knihovny klientského softwaru zpravidla poskytovány dodavatelem databáze. Dané knihovny musí být integrovány do klientské aplikace, která si přeje získat přístup k databázovému serveru. Poskytují rozhraní API pro nalezení a připojení k databázovému serveru, stejně jako nastavení a spouštění databázových dotazů a příkazů. Obrázek 4 nastiňuje typickou databázi RDBMS klient/server.



Obrázek 4 - Architektura RDBMS

Zdroj: přepracováno dle (JAY A. KREIBICH, 2010)

Na rozdíl od RDBMS nemá SQLite architekturu klient/server, jelikož nedisponuje žádným samostatným serverem. Celý databázový stroj je integrován do jakékoli aplikace, která potřebuje pracovat s databází. Jediný způsob sdílení mezi aplikacemi představuje soubor s jednou databází, který je uložen na fyzickém disku. Pro přesun nebo zálohování databáze stačí jednoduše zkopírovat soubor. Obrázek 5 ukazuje infrastrukturu SQLite.



Obrázek 5 - Architektura SQLite

Zdroj: přepracováno dle (JAY A. KREIBICH, 2010)

Eliminací serveru se odstraní značné množství složitosti. Zjednodušují se softwarové komponenty a nastává téměř nezávislost od podpory operačního systému. Na rozdíl od RDBMS, který potřebuje pokročilý multitasking[8] a vysoce výkonnou meziprocesovou komunikaci, vyžaduje SQLite o něco více než schopnost číst a zapisovat do nějakého typu úložiště. Na základě této jednoduchosti je poměrně snadné přenášet SQLite na téměř libovolné prostředí (např. mobilní telefony, herní konzole apod.). SQLite je integrován přímo do spustitelného souboru, aby eliminoval potřebu externí knihovny a zjednodušoval distribuci a instalaci.

Odstraněním serveru však vznikají některá omezení. SQLite je určen k potřebám lokálního ukládání. Znamená to, že není vhodný pro situace, kdy více klientských aplikací potřebuje přistupovat k jedné centralizované databázi. [2], [5]

[8] schopnost vykonávat více procesů současně

5.1.1 Uložiště

SQLite databáze vystupuje jako jediný soubor, který obsahuje rozložení databáze a uložená aktuální data v tabulkách. Formát souboru poskytuje kompatibilitu napříč platformami a dostupnost na libovolném počítači bez ohledu na pořadí bytů a velikosti. Pokud databáze představuje pouze jediný soubor, je triviální k vytvoření, ke kopírování nebo k zálohování.

Celé databáze tím mohou být jednoduše přesunuty, modifikovány a sdíleny. Umožňuje aplikacím používat instance databáze jako dokumenty, úložiště dat nebo preferenční data. Neexistuje šance, aby se databáze stala poškozenou nebo nedostupnou, protože jeden z tuctu souborů byl náhodou přesunut nebo přejmenován. [2], [5]

5.1.2 Jedinečné funkce

SQLite nabízí několik funkcí, které se nenacházejí v mnoha jiných databázových systémech. Nejvíce pozoruhodný rozdíl spočívá v tom, že SQLite používá pro tabulky systém dynamického typu. Umožní vkládat libovolnou hodnotu do téměř libovolného sloupce bez ohledu na typ. Jedná se o zásadní odklon od tradičních databázových systémů, které mají tendenci být staticky definovány. V mnoha ohledech je dynamický typ podobný tomu, který se nachází v populárních skriptovacích jazycích. Mají často jediný skalární typ, který může přijmout cokoli od celých čísel až k textovým řetězcům.

Další užitečnou funkcí je schopnost manipulovat s více než jednou databází v jednom okamžiku. Umožňuje jedno databázové připojení k více databázovým souborům současně. Dělá to trivialitu spojení tabulek z různých databází s jediným dotazem, nebo hromadné kopírování dat s jediným příkazem.

SQLite má také možnost vytvářet databázi v paměti zařízen. Jedná se v podstatě o „soubory“, které nemají zálohovací uložiště a nachází se po celou životnost v mezipaměti. Zatímco databáze v paměti nedisponují trvanlivostí a neposkytují plnou podporu transakcí, nabízejí velkou rychlost (za předpokladu dostatku paměti RAM) a jsou využitelné zejména pro ukládání dočasných tabulek a dalších přechodných dat. [2], [5]

5.2 Použití

SQLite je pozoruhodně flexibilní, ať už se jedná o použitelnost nebo prostředí, kde může být spuštěn. Některé vlastnosti, které by měl SQLite plnit, jsou podobné jako u tradičních RDBMS. Velikost databáze SQLite a jednoduchost použití někdo nepovažuje za úplnou databázi. [2], [5]

5.2.1 Jednoduchá databáze

Velké platformy RDBMS jsou výkonným nástrojem pro bezpečné ukládání, uspořádání a manipulaci s daty. Většina velkých produktů RDBMS se prezentují náročností na prostředky a vyžadují spoustu údržby. Zvyšuje to jejich výkonnost a kapacitu, ale také omezení, jak a kde mohou být prakticky nasazeny.

Návrh SQLite vyplňuje tyto mezery a zároveň poskytují stejně mocné a známe nástroje pro bezpečné ukládání, organizování a manipulaci s daty v omezeném prostředí. Implementace SQLite má za úkol doplnit platformy RDBMS v situacích, kdy je jednoduchost a snadnost použití důležitější než kapacita a souběžnost. Umožnuje aplikacím a nástrojům využívat relační správu dat i na menších platformách bez administrativního dohledu. [2], [5]

5.2.2 Souborové aplikace

Moderní desktopové aplikace se obvykle vyznačují velkým počtem souborů. Většina aplikací a nástrojů využívá jeden nebo více souborů. Mohou existovat také uživatelské konfigurační soubory, mezipaměti a další data, která musí být sledována a uložena.

Použití knihovny SQLite jako abstraktní úložné vrstvy má mnoho výhod (např. metadata aplikací, stavové údaje, konfigurační údaje apod.) Díky tomu je relativně snadné vytvořit vhodný design databáze, který snadno a jednoduše mapuje vnitřní strukturu dat aplikace.

Spravedlivé množství metadat aplikací, jako jsou mezipaměti, stavové údaje a konfigurační data, se dobře hodí k relačnímu datovému modelu. I když aplikace nemá specificky náročné relační požadavky, existují značné výhody při použití knihovny SQLite jako kontejneru pro ukládání dat. Poskytuje přírůstkové aktualizace, které umožňují rychlé a snadné ukládání malých změn. Transakční systém chrání všechny vstupní výstupní soubory před ukončením práce a narušením napájení, což téměř eliminuje možnost poškození souborů. SQLite poskytuje vlastní vrstvu pro ukládání do mezipaměti, takže lze otevřít a zpracovat velmi velké soubory v omezené paměti bez další interakce na straně aplikace. [2], [5]

5.2.3 Cache aplikace

SQLite disponuje možností vytvářet databáze, které jsou uchovávány zcela v paměti. To je extrémně užitečné pro vytváření malých, dočasných databází, kde není potřeba permanentního uchovávání dat.

Databáze v paměti se často používají k ukládání výsledků stažených z tradičních RDBMS. Aplikace může zpřístupnit podmnožinu dat ze vzdálené databáze, umístit ji do dočasné databáze a potom na ni provádět různé databázové operace. Postup umísťování do dočasné databáze se jeví velmi užitečné v případech potřeby rychlé interakce s určitou množinou dat. Dočasné databáze mohou být také použity k indexování a ukládání téměř jakéhokoli typu interně provázaných dat.

Spíše než navrhovat sadu komplexních datových struktur runtime (např. hashovací tabulky, stromy apod.), může vývojář jednoduše navrhnout vhodné schéma databáze a uložit jednoduše do ní data. Tento způsob ukládání představuje značnou efektivitu a snížení složitosti při vývoji. Pokud databáze v paměti nemůže dále růst (nedostatek paměti RAM) nebo znemožňuje běh aplikace za běhu, lze databázi uložit na disk. [2], [5]

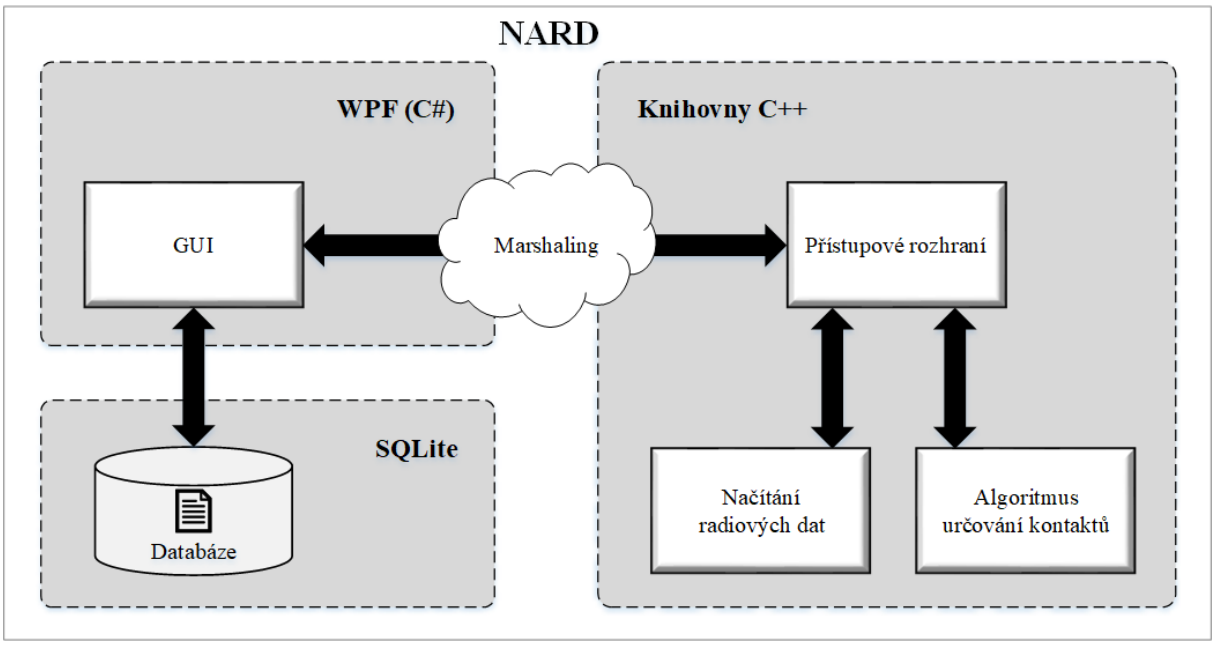
5.2.4 Archivy a datové uložiště

SQLite umožnuje jednoduše zabalit komplexní datové množiny do jediného souboru, který je snadno přístupný a plně přenositelný. Veškerá data v jediném souboru představují jednodušší distribuování a stahování dat z mnoha tabulek (např. velké slovníky, odkazy na geolokaci atd.). Na rozdíl od mnoha produktů RDBMS může SQLite přistupovat k databázím v režimu pouze pro čtení. Umožňuje to ukládat data přímo z optického disku nebo jiného souborového systému (pouze v režimu čtení). Tato možnost je obzvláště užitečná pro systémy s omezeným prostorem na pevném disku, jako jsou konzole na videohry. [2], [5]

6 NÁSTROJ PRO ANALÝZU RADIOVÝCH DAT

Praktická část závěrečné práce se věnuje tvorbě softwarového nástroje pro samočinnou analýzu rádiových dat. Softwarový nástroj pojmenovaný NARD (Nástroj Analýzy Radiových Dat) bude umožňovat načítat libovolný počet rádiových dat z binárního souboru, do kterého se zachycená rádiová data ukládají z pasivního sledovacího systému. Dále tyto data analyzovat (zpracovat) a uložit do lokální databáze. Nad databází se následně budou moci provádět selekce, jejímž výsledkem bude zobrazení v grafické podobě.

Hlavním cílem praktické části je navrhnout algoritmus, jenž bude zkoumat schodu mezi impulsy. Impulsy představují abstrakci důležitých hodnot radiových dat. Shoda mezi impulsy se bude dále nazývat *kontakt*. Kontakt lze definovat jako sekvence impulsů od jednoho rádiového zdroje, který má v měřítku desítek milisekund rozpoznatelný začátek a konec sekvence. Vedlejším cílem bude vytvoření grafické aplikace, jenž poslouží k otestování správnosti algoritmu. Aplikace bude umožňovat vybírání zdrojových radiových dat, použití implementovaného algoritmu pro určování kontaktů a následné uložení zpracovaných impulsů do lokální databáze. Ty mohou být staženy a vyobrazeny, aby operátor mohl provádět kontrolu a jednoduchou interakci s daty v podobě grafu.



Obrázek 6 - NARD – Architektura

Zdroj: Vlastní

K dosažením cílů bude využita veškerá zmíněná technologie v teoretické části závěrečné práce. Obrázek 6 vyobrazuje použití jednotlivých technologiích pro specifické části budované aplikace. Grafická aplikace bude postavěna na technologii WPF (C#), která bude zastřešovat procesy načítání, analýzy, ukládání a zobrazení.

Načítání a analýza budou vystupovat jako samostatné knihovny, který budou spojeny jednou rozhraní knihovnou pro jednotnou správu těchto knihoven. Z důvodu rychlosti a potřeby práce s velkým množstvím dat na nižší úrovni, kde se bere velký ohled na zacházení a využívání paměti RAM, byla pro tyto knihovny zvolena technologie C++.

Aby bylo možné určit, který binární soubor s rádiovými daty se má zpracovat, a následně se k načteným a analyzovaným impulsům dostat v GUI aplikaci, bude využit technologický proces marshaling. Jednoduše lze říci, že knihovna, která funguje jako přístupové rozhraní, poskytne GUI aplikaci ukazatel na paměť, kde jsou impulsy dočasně uloženy, a celkový počet všech načtených impulsů. Ta si je následně namapuje do svých definovaných struktur, která jsou totožná jako struktury v C++, a dále se s nimi již pracuje v technologii C#.

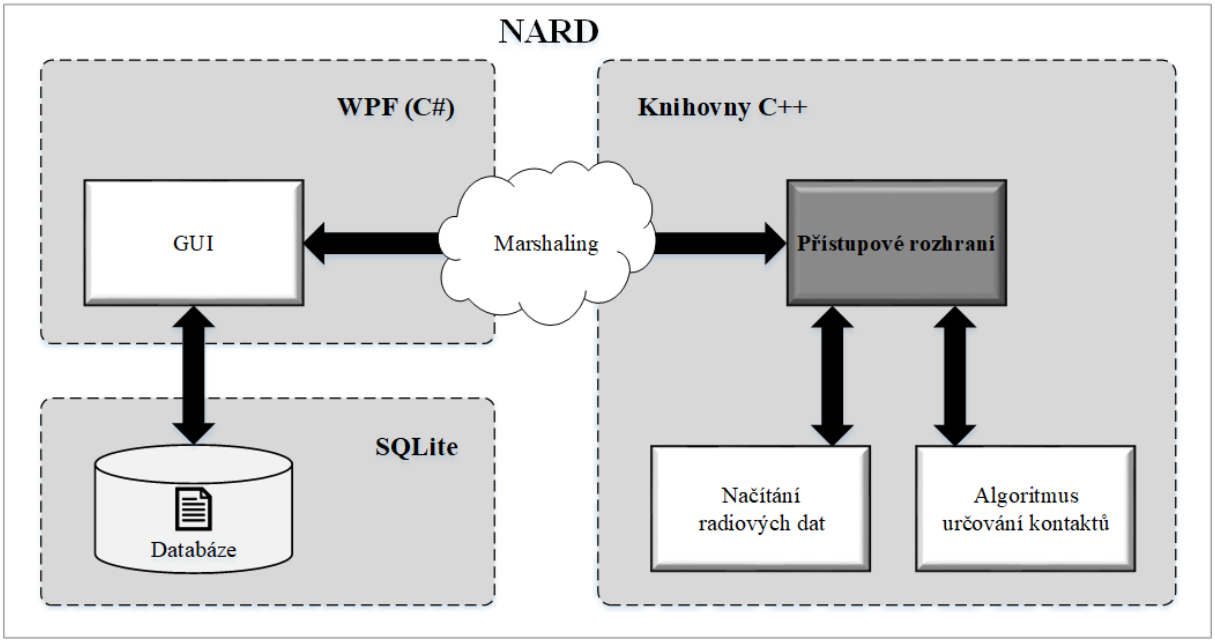
Pokud jsou data již k dispozici na úrovni GUI aplikace, mohou být následně uložena do lokální databáze, kde se uchovají pro případné další zkoumání. Jedním z důvodu volby technologie SQLite pro uchovávání dat byl fakt, že je někdy potřeba jednotlivé zpracované soubory ukládat do rozdílných databází. Dalším důvodem volby této technologie je jednoduchá manipulace, přenositelnost souborů SQLite a citlivý charakter zpracovaných impulsů.

Celá NARD aplikace se skládá z pěti částí, kde každá část se zaobírá svoji specifickou problematikou. Tyto části tvoří:

- GUI – grafické rozhraní aplikace
- Databáze – uchovává zpracované impulsy
- Přístupové rozhraní – zpřístupňuje GUI aplikaci impulsy
- Načítání rádiových dat – stará se o načítání rádiových dat z binárního souboru, jejich abstrakci na impulsy a uložení do paměti
- Algoritmus určování kontaktů – implementuje navrhnutý algoritmus pro určování kontaktů dočasně uložené impulsy v paměti

6.1 Přístupové rozhraní

Cílem knihovny *AccessIterface.dll* je zprostředkovat GUI aplikaci načtená a zpracovaná data. Mimo jiné má na starosti správu připojených knihoven pro načítání a zpracování impulsů. Následující Obrázek 7 ukazuje, jaké místo v aplikaci NARD zastupuje.



Obrázek 7 - NARD – Přístupové rozhraní

Zdroj: Vlastní

Knihovna disponuje následujícími metodami, které jsou přístupné v rámci GUI aplikace:

- `InitDataLoader`
- a `ExportImpulses`.

6.1.1 Inicializace a export vektoru impulsů

První zmíněná metoda má na starosti inicializaci *vektoru* („skladu“) impulsů. Vstupem do této metody je cesta k umístění binárního souboru s rádiovými daty, kterou zadá operátor prostřednictvím GUI aplikace. Cesta je následně předána knihovně, která má na starosti načítání rádiových dat. Pokud načtení abstrakce rádiových dat do vektoru impulsů proběhlo v pořádku, je předána reference na vektor načtených impulsů knihovně, jenž aplikuje na načtené impulsy navrhnutý a implementovaný algoritmus pro určování kontaktů. Výstupem inicializační metody je logická hodnota pro nadřazenou vrstvu GUI, zda byly úspěšně načteny a zpracovány impulsy.

Impuls je reprezentován následující strukturou, která je popsána v Tabulce 5. V tabulce jsou uvedeny pouze atributy, které jsou hlavními složkami pro budoucí analýzu. Jednotlivé hodnoty atributů jsou vyplňovány dynamicky tzn., že nemusí být nastaveny najednou, ale mohou být doplňovány postupně.

Tabulka 5 - Datová struktura impulsu

Datový typ	Název atributu	Popis	Jednotky
`uint32_t`	`Id`	Unikátní identifikátor impulsu pro DB	-
`uint32_t`	`MeasuringSection`	Měřící sekce	ms
`uint32_t`	`Idx`	Index výskytu od začátku měřícího intervalu	
`uint64_t`	`TimePart`	Celá část unixového času	s
`uint64_t`	`DecimalTimePart`	Desetinná část unixového času	ps
`uint8_t`	`Channel`	Vstupní kanál	-
`uint32_t`	`Frequency`	Frekvence	kHz
`uint8_t`	`PrefixFrequency`	Prefix frekvence pro tvoření *hash* hodnoty (První dvě čísla frekvence)	-
`uint32_t`	`Pw`	Délka impulsu	ns
`uint64_t`	`Pri`	Čas, uplynulý od příchodu předchozího impulsu v kontaktu	ps
`uint32_t`	`Mi`	Měřící interval	-
`uint8_t`	`Amplitude`	Amplituda	-
`uint32_t`	`ContactId`	Id kontaktu, ke kterému impuls patří	-

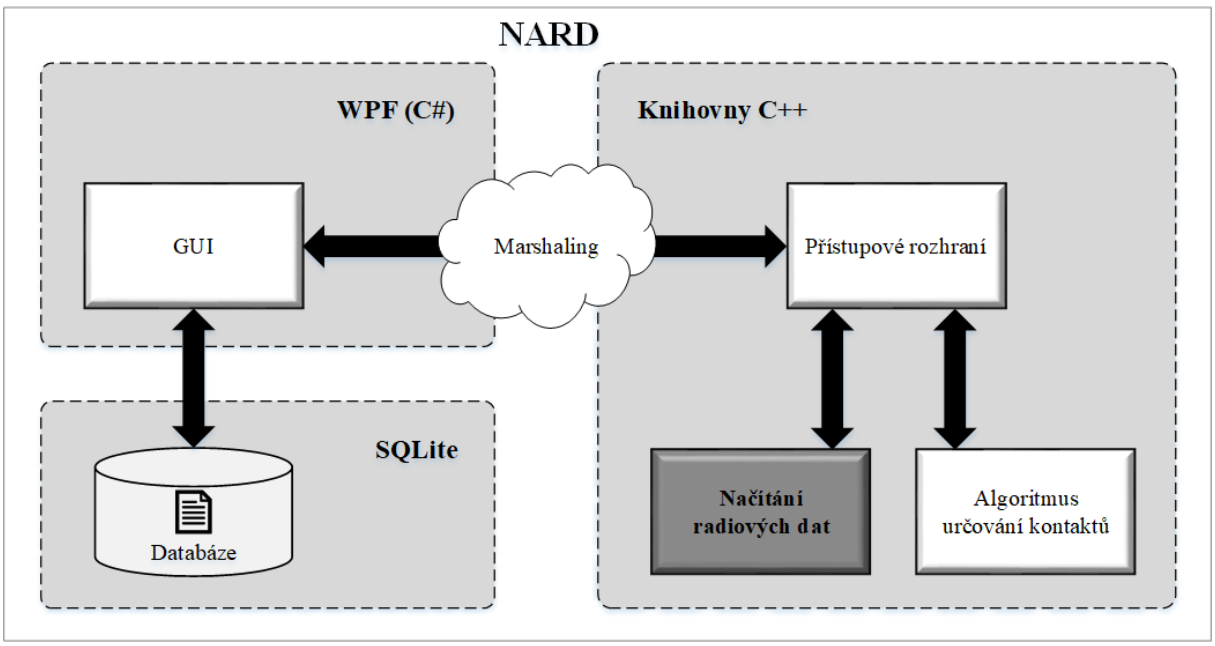
Zdroj: Vlastní

Jestliže proběhla korektní inicializace vektoru impulsů, požádá GUI aplikace o export těchto impulsů. Exportní metoda pracuje s již inicializovanými impulsy v paměti. Má za úkoly poskytnou ukazatel na paměť začátku vektoru impulsů a hodnotu počtu všech impulsů obsažených ve vektoru. Ukazatel na paměť a celkový počet je následně předána zpět do GUI aplikace, kde s nimi aplikace dále pracuje.

6.2 Načítání rádiových dat

Knihovna *DataReader.dll* má na starosti načítání radiových dat z binárního souboru a jejich abstrakci do impulsů. Knihovna je podřazeným prvkem přístupového rozhraní tzn., že čeká na povel z této knihovny. Povel je myšleno, že přístupové rozhraní předá načítací knihovně referenci na prázdný vektor impulsů a cestu k umístění binárního souboru pro načtění. Následující Obrázek 8 vyobrazuje podřazenost dané knihovny vůči přístupovému rozhraní.

Knihovna obsahuje jednu hlavní metodu `InitDataReader`, ke které přistupuje přístupové rozhraní:



Obrázek 8 – NARD – Načítání rádiových dat

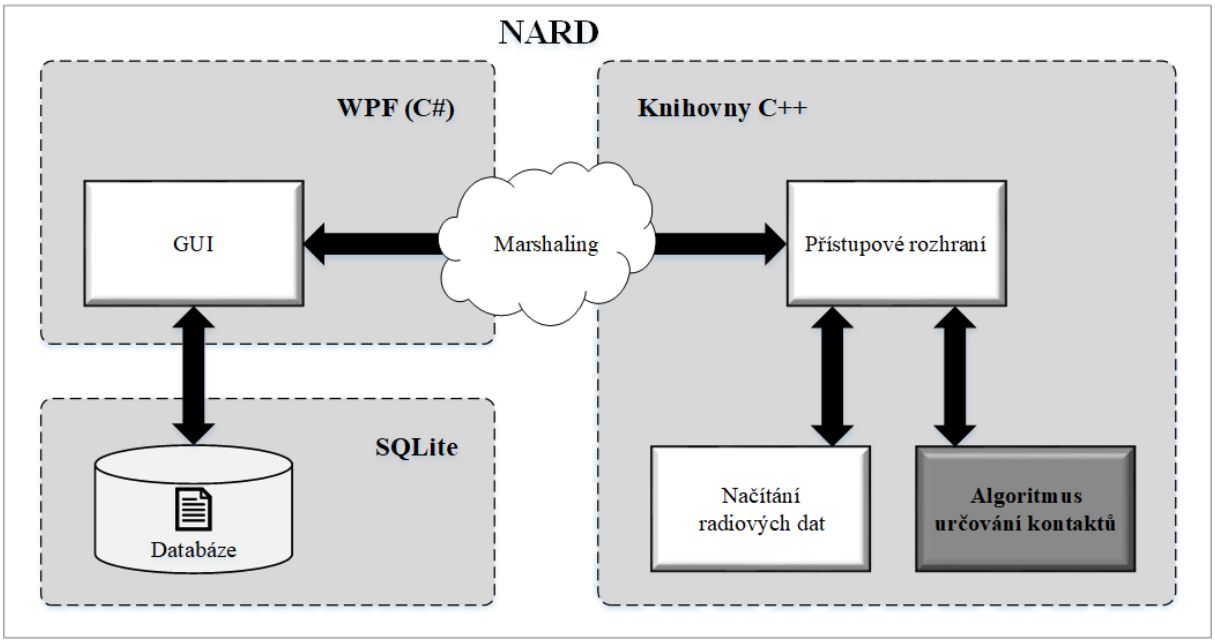
Zdroj: Vlastní

6.2.1 Postup načítání

Hlavní metoda `InitDataReader` obdrží jako vstup referenci na prázdný vektor impulsů a cestu k umístění binárního souboru rádiových dat. Následně dojde k otevření, načtení a uložení binárního obsahu souboru do paměti. Binární obsah je převeden na rádiová data a následně dochází k abstrakci těchto dat na impulsy, které se uloží do zpřístupněného prázdného vektoru. Po dokončení načítání a naplnění vektoru jsou impulsy ve vektoru seřazeny podle času (`TimePart` a `DecimalTimePart`). Výstupem je logická hodnota, která informuje přístupné rozhraní o výsledku postupu načítání. Přístupové rozhraní na základě hodnoty výsledku postupuje podle definovaného scénáře nebo dochází k informování GUI aplikace o neúspěchu naplnění vektorů impulsů.

6.3 Algoritmus určování kontaktů

Stěžejní knihovna *DataAnalysis.dll* aplikace NARD má za primární úkol určování kontaktů neboli hledání shod mezi impulsy. Účelem knihovny je zapsat do atributu `ContactId` impulsu číselnou hodnotu. Pomocí `ContactI` dochází ke spojení souvisejících impulsů a tím vzniká kontakt. Hodnota atributu `ContactId` se mění pouze u impulzů, které spadají do některého z kontaktů, jinak zůstává hodnota atributu ve výchozím stavu.



Obrázek 9 - NARD – Algoritmus určování kontaktů

Zdroj: Vlastní

Knihovna disponuje jednou vstupní metodou `InitDataAnalysis`, jenž odstartuje proces analýzy vektoru impulsů:

Definovanou metodu využívá knihovna přístupového rozhraní. Přístupové rozhraní poskytne metodě referenci na naplněný vektor impulsů. Výstupem je logická hodnota, která upozorní přístupové rozhraní o ukončení analýzy dostupných impulsů. Hodnota je posléze předána GUI aplikaci, která na základě jejího obsahu požádá přístupové rozhraní o export těchto impulsů.

Funkce knihovny lze rozdělit na fáze:

- počáteční inicializace,
- proces identifikace adeptů na kontakt měřící sekce (MS),
- proces vyhodnocení adeptů na kontakt MS,
- proces filtrování SCM podle minimální hodnoty PRI,
- proces filtrování SCM podle histogramu PRI,
- a konečné zpracování.

6.3.1 Počáteční inicializace

Impulsy budou zpracovávány postupně MS. MS se zavádějí, aby došlo k rozdělení impulsů do menších celků. Impulsy si hodnotu MS uchovávají v atributu `MeasuringSection`. Výhody definování MS spočívají v tom, že na konci MS dochází k vyhodnocení obsažených impulsů v dané MS. Vyhodnocení spočívá v odstranění impulsů, které nesplňují podmínku pro označení jako adepta na kontakt. Odstranění nevhodných impulsů (např.: přeslechy, odrazy apod.) již v průběhu zpracování umožní pracovat pouze s impulsy, které mohou tvořit adepta na kontakt a zrychlit průběh zpracovávání. Na základě empirických zkoumání bylo stanoveno časové kvantum MS na 15 ms. Definovaná hodnota představuje optimální časové kvantum pro náběr dostatečného množství impulsů, aby čekání na vyhodnocení na konci MS bylo přijatelné.

První inicializace MS probíhá na základě času prvního impulsu obsaženého ve vektoru, který již byl utříděn podle času vzestupně v knihovně pro načítání rádiových dat. Od daného času začíná první MS.

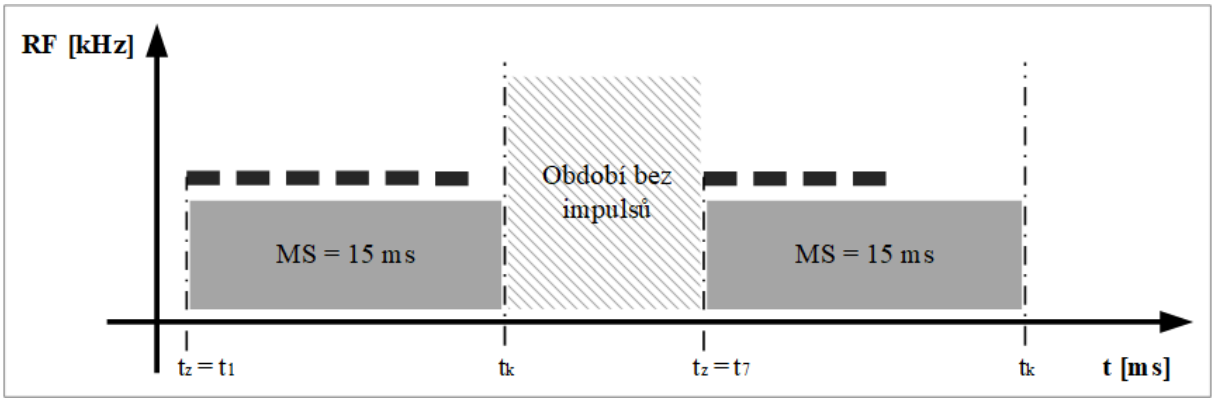
Obecně bude platit pro každou MS, že se stanoví čas začátku MS t_z, který bude odpovídat času příchodu i-tého impulsu t_i v utříděném vektoru, kde $i = 1 \ldots n$, a koncový čas MS t_k, jenž odpovídá vztahu:

$$t_k = t_z + \check{c}asov\acute{e}\ kvantum\ MS.$$

MS se určují dynamicky (viz. Obrázek 10). Dynamičnost umožňuje přeskakovat časové okamžiky, kdy nedochází k výskytu žádného impulsu. Nová MS se vytváří, jak již bylo definováno, s příchodem impulsu tzn., že MS bude obsahovat nejméně jeden impuls. Cílem dynamického přístupu je:

- shromáždit maximální počet impulsů od okamžiku vytvoření MS, jelikož se přepokládá sled dalších impulsů,
- a zvýšit šanci na nalezení dostatečného počtu impulsů pro vytvoření adepta na kontakt.

Stávající adepti z minulé MS nejsou ovlivňovány dynamickým způsobem určování MS.

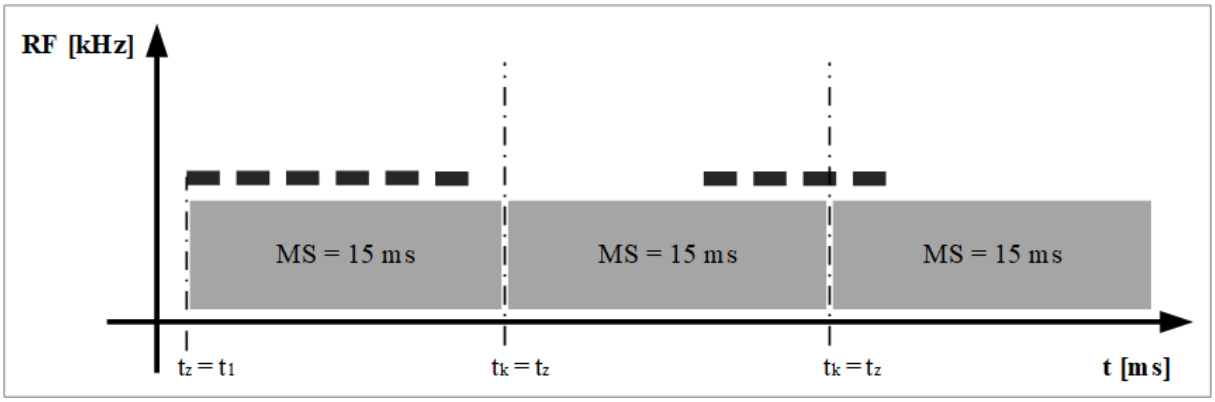


Obrázek 10 - Dynamické určování MS

Zdroj: Vlastní

Další možnou variantou určování MS bylo statické určování (viz. Obrázek 11) tzn., že první impuls v utříděném vektoru určil první MS a další MS začínají hned po vypršení časového kvanta MS, měl za následek, že se impulsy mohou objevovat až na konci jedné MS a mohou pokračovat v následující MS. Pokud vznikl nový adept na kontakt v MS a nebude obsahovat dostatečný počet impulsů v dané MS, bude odstraněn jako statisticky nevýznamný a odstraněné impulsy mohou následně chybět v následující MS. Jestliže počet impulsů v následující MS, které jsou stejného charakteru jako v předchozí MS, není dostatečný, jsou taktéž odstraněny.

Poznatek spočívá v tom, že pokud se budou nový adepti na kontakt vyskytovat na konci MS, mohou být zbytečně odstraněny.



Obrázek 11 - Statické určování MS

Zdroj: Vlastní

Dynamické určování MS přináší hlavní výhodu v tom, že přepokládá výskyt sekvence impulsů hned na začátku MS a tím zvýší šanci, aby nový adept na kontakt mohl nabrat dostatečný počet impulsů a nebyl vyhodnocen jako nevýznamný. Proto bylo zvoleno přístup dynamické určování MS.

6.3.2 Proces identifikace adeptů na kontakt MS

Po skončení počáteční inicializace začíná proces identifikace případných adeptů na kontakt v určené MS. Z vektoru impulsů jsou postupně vybírány impulsy a proces se je snaží přiřadit k odpovídajícímu adeptu na kontakt nebo se sám vybraný impuls stane novým adeptem na kontakt.

Adepti na kontakt v dané MS tvoří jednotlivé položky *aktuální mapy kontaktů (CCM)*. Adepti na kontakt ze všech MS se uchovávají v *uložené mapě kontaktů (SCM).* Přičemž jedna položka v CCM a SCM se skládá z:

- klíče
- a hodnoty.

Načež klíč CCM a SCM tvoří předpis (viz. Tabulka 6) pro adepta na kontakt.

Tabulka 6 - Předpis klíče CCM a SCM

Datový typ	Název atributu	Popis	Jednotky
`uint32_t`	`Id`	Unikátní identifikátor adepta na kontakt	-
`uint32_t`	`Frequency`	Frekvence	kHz
`uint8_t`	`PrefixFrequency`	Prefix frekvence pro tvoření *hash* hodnoty (První dvě čísla frekvence)	-
`uint32_t`	`Pw`	Délka impulsu	ns
`uint8_t`	`Channel`	Vstupní kanál	-
`bool`	`State`	Stav adepta na kontakt (otevřeny, zavřený)	
`uint64_t`	`EndTimePart`	Celá část času příchodu posledního impulsu	s
`uint64_t`	`EndDecimalTimePart`	Desetinná část času příchodu posledního impulsu	ps
`uint32_t`	`ToleranceFrequency`	Tolerance frekvence	kHz
`uint32_t`	`TolerancePw`	Tolerance délky impulsu	ns

Zdroj: Vlastní

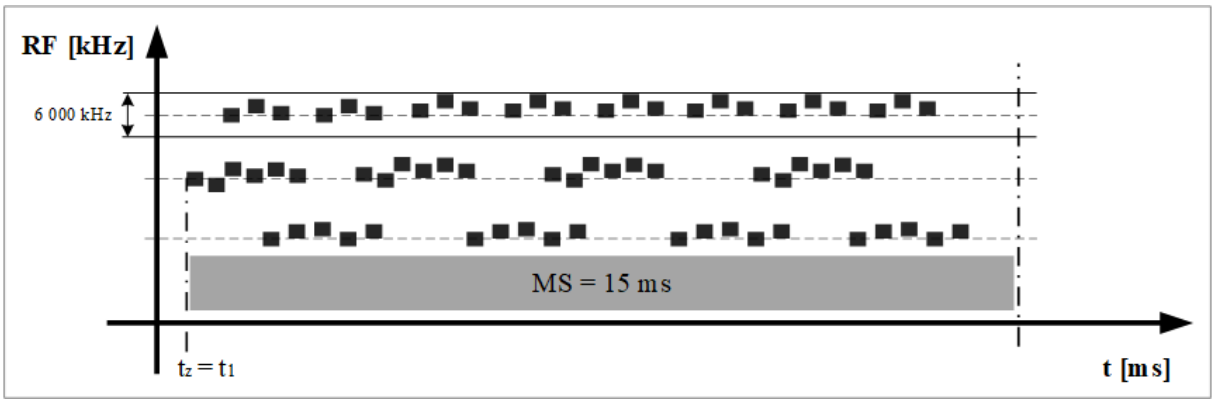
Hodnotou jednoho adepta na kontakt v CCM a SCM představuje vektor ukazatelů na impulsy, jejichž primární vlastnosti se shodují s primárními atributy klíče CCM.

Primárními atributy pro definování, zda daný impuls patří k danému adeptu na kontakt v CCM, jsou:

- stav adepta na kontakt (otevřený),
- vstupní kanál (stejný),
- frekvence (s uvažováním tolerance)
- a délka impulsu (s uvažováním tolerance).

Pokud jsou splněny tyto podmínky, impuls patří k danému adeptu na kontakt. Primární atributy frekvence a délka impulsu pracují s uvažováním tolerance. Hodnota tolerance byla stanovena na základě empirických zkoumání a znalostí z praxe. Na základě těchto podkladů byla tolerance frekvence definována na 3000 kHz a tolerance délky impulsu na 200 ns.

Obrázek 12 zobrazuje tři potencionální adepty na kontakt, kdyby se pracovalo pouze za podmínky frekvence. Pro názornou ukázku hledaní adeptů v MS tento obrázek postačí. Dále vyznačuje aplikovanou frekvenční toleranci pro každého adepta. Obdobný způsob tolerance se použije i na délku impulsu.



Obrázek 12 – Použití frekvenční tolerance

Zdroj: Vlastní

Na začátku procesu je zpřístupněn impuls z vektoru impulsů. Proběhne vyhodnocení, zda aktuálně zpřístupněný impuls náleží do aktuálně definované MS. Jestliže patří:

- pokračuje se dále v procesu identifikace,
- nebo přijde na řadu proces vyhodnocování.

Potencionálně každý zpracovávaný impuls představuje nového adepta na kontakt. Proto je z aktuálně zpřístupněného impulsu nejdříve vytvořen nový adept, kde klíčové hodnoty atributů adepta budou odpovídat atributům impulsu.

Nově vytvořený adept na kontakt, který nyní obsahuje jeden impuls, je porovnán proti již definovaným adeptům na kontakt v CCM. Jestliže existuje shoda, nově vytvořený adept o jednom impulsu zaniká a impuls, který ho tvořil, je přidán k ostatním impulsům shodujícího se adepta. Pokud shoda neexistuje, nově definovaný adept s jedním impulsem je vložen jako nová položka do CCM.

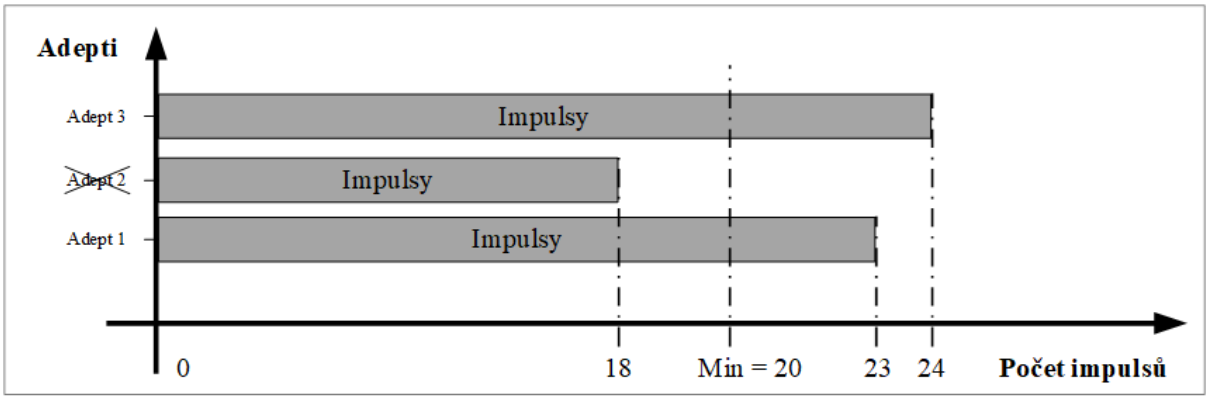
Daný proces se replikuje postupně na všechny impulsy ve vektoru impulsů, přičemž dojde k vyhodnocení právě definované MS, pokud přijde na řadu impuls, který již do této aktuální MS nepatří.

6.3.3 Proces vyhodnocení adeptů na kontakt MS

Jestliže při procesu identifikace adeptů na kontakt přijde na řadu impuls, který již nepatří do právě definované MS, je proces identifikace dočasně pozastaven a nastává proces vyhodnocení adeptů této MS.

V procesu vyhodnocení se projdou všichni adepti na kontakt v CCM. Pokud procházený adept obsahuje méně než 20 impulsů, je adept zahozen. Hodnota minimálního počtu impulsu pro validního adepta na kontakt byla zvolena na základě empirických zkoumání a znalostí z praxe. Časové kvantum MS bylo nastaveno s přihlédnutím na toto zkoumání na dostatečně velikou hodnotu, aby všichni validní adepti hranici minimálního počtu impulsů s přehledem splnily. Menší počet impulsů než definovaná hranice muže znamenat různé přeslechy, odrazy apod. Následně by mohly tyto impulsy negativním způsobem ovlivňovat další průběh zpracování.

Pro ilustraci následující Obrázek 13 zobrazuje tři adepty na kontakt v jedné MS a počet impulsů, které tvoří daného adepta. Z nichž dva splňují a jeden nesplňuje minimální hranici počtů impulsů.



Obrázek 13 - Vyhodnocení adeptů na kontakt v MS

Zdroj: Vlastní

Pokud procházený adept splňuje minimální hranici počtu impulsů adepta na kontakt je s ním naloženo následujícími způsoby:

- Adept se nenachází se v SCM, je do SCM vložen.
- Adept se nachází v SCM a zároveň počet impulsů adepta byl navýšen v CCM, je odpovídající adept v SCM aktualizován.
- Adept se nachází v SCM a zároveň počet impulsů nebyl navýšen v CCM, vyhodnocuje se rozdíl mezi časem konce dané MS t_k a časem příchodu posledního impulsu daného adepta na kontakt. Jestliže rozdíl je větší než 1 sekunda, je adept v CCM uzavřen a nastaví se mu čas konce dané MS. Na základě empirických zkoumání a znalostí byla definována tato prodleva čekání na další impuls.

Jestliže nastane situace, že se objeví nový adept na kontakt v CCM v budoucích MS, který disponuje stejnými parametry jako uzavřený adept v SCM, je vyhodnocen jako nový záznam SCM. Výhoda v uzavírání adeptů v SCM spočívá v tom, že je zaznamenán násobný výskyt v různých časových okamžicích a nebudou sloučeny do jednoho výskytu pod jednou časovou značkou.

Po vyhodnocení adeptů na kontaktů v dané MS dojde k vymazání CCM a překopírují se do ní takový adepti na kontakt, který nebyly uzavřeny během vyhodnocování. Následně se dynamicky nastaví nová MS podle času příchodu impulsu, který byl na řadě před tím, než došlo ke spuštění procesu vyhodnocení MS.

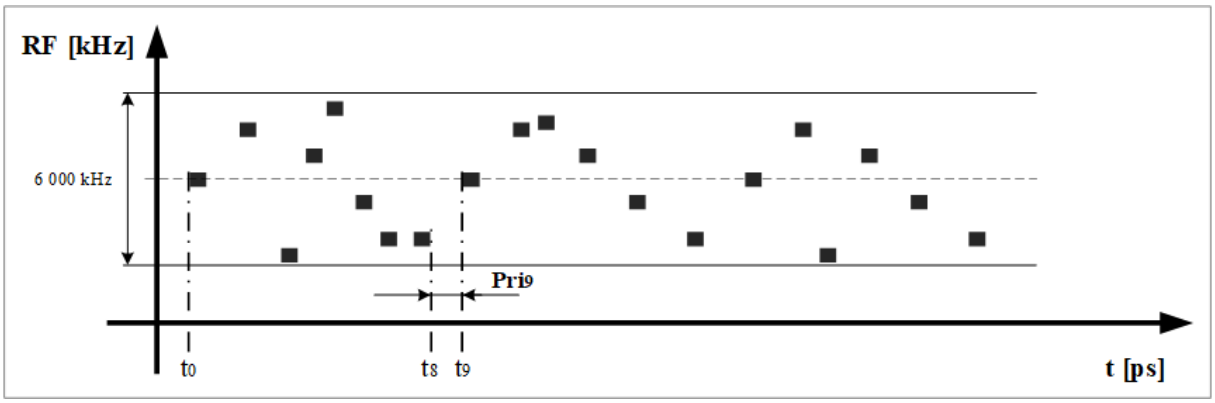
Závěrem lze shrnout, že procesy identifikace a vyhodnocení adeptů na kontakt MS se postupně střídají, dokud se neprojdou všechny impulsy ve vektoru. Výstupem těchto operací je naplnění SCM validními adepty na kontakt.

6.3.4 Proces filtrování SCM podle minimální hodnoty PRI

Jednotlivým impulsům v adeptech na kontakt v SCM se vypočítá hodnota PRI. Jelikož impulsy v daných adeptech jsou seřazeny podle času, je hodnota PRI počítána vzorcem:

$$PRI = t_n - t_{n-1} \ [ps],$$

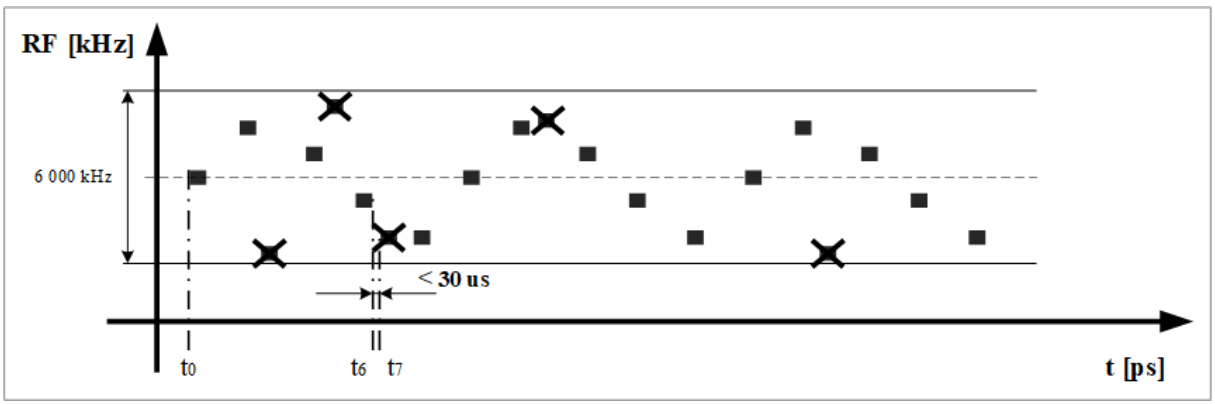
kde t_n je čas příchodu aktuálního impulsu a t_{n-1} je čas příchodu impulsu předešlého před aktuálním (viz. Obrázek 14).



Obrázek 14 - Aplikování výpočtu hodnoty PRI

Zdroj: Vlastní

Následuje proces, který odmaže z adeptů na kontakt impulsy, kde hodnota PRI je menší než 30 us (viz. Obrázek 15). Z empirických znalostí bylo určeno, že následující impuls v adeptu na kontakt, nemůže přijít do 30 us. Impulsy, které přicházejí do 30 us, mohou adepta zkreslit. Tyto impulsy jsou definovány v praxi jako přeslechy, odrazy apod.



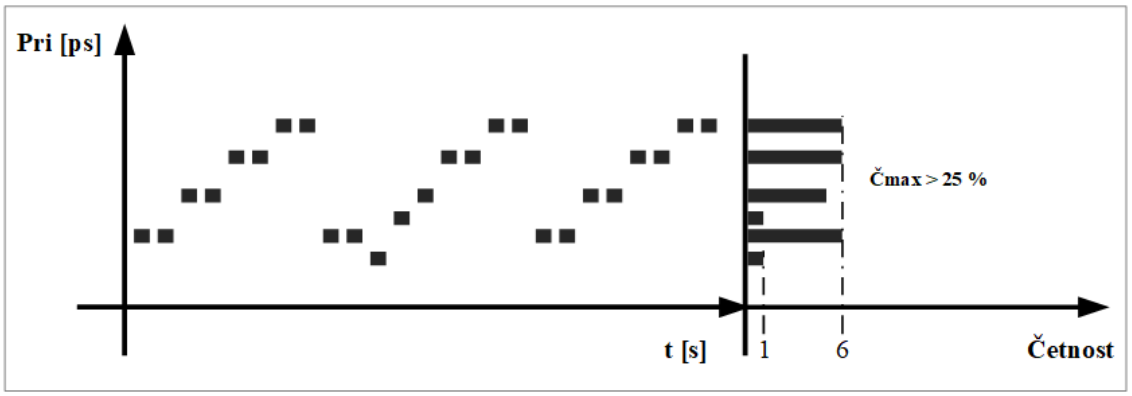
Obrázek 15 - Odstranění impulsů pomocí minimální hodnoty PRI

Zdroj: Vlastní

Po odstranění nežádoucích impulsů z adeptů na kontakt v SCM, se přepočítají hodnoty PRI, jelikož po odebrání impulsů vzniknou *„díry“* a časy PRI již neodpovídají skutečnosti. Po z aktualizování hodnot PRI adeptů na kontakt v SCM dojde znovu ke kontrole, zda aktualizovaní adepti obsahují více než 20 impulsů. Pokud tato skutečnost nastane, je nevyhovující adept odstraněn ze SCM.

6.3.5 Proces filtrování SCM podle histogramu PRI

Následný proces projde znovu všechny adepty na kontakt v SCM, ale tentokrát se použije jiný systém filtrování. K aktuálnímu procházenému adeptu v SCM se vytvoří histogram četností hodnot PRI impulsů (viz. Obrázek 16).

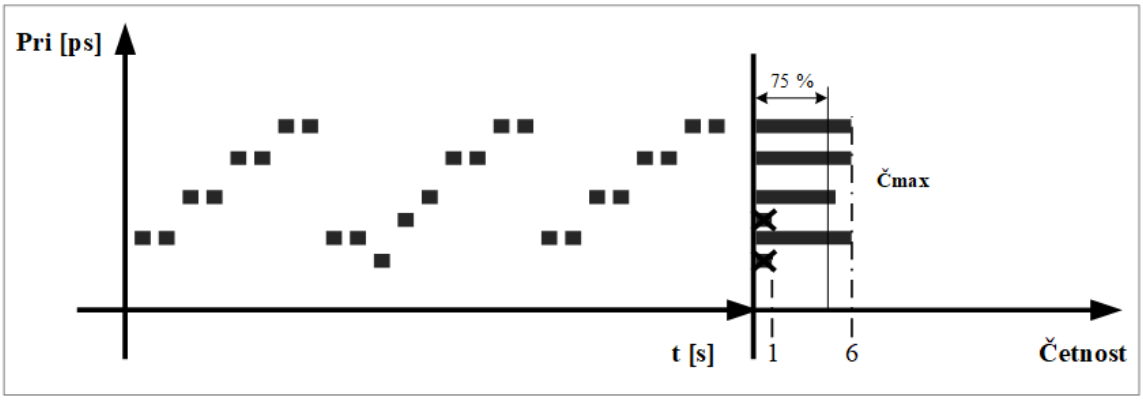


Obrázek 16 - Histogram četností hodnot PRI impulsů

Zdroj: Vlastní

Ve vytvořeném histogramu se nalezne nejvyšší sloupec, který představuje největší četnost $č_{max}$, a vyhodnotí se, zda hodnota $č_{max}$ konkrétního PRI tvoří minimálně 25 procent všech četností v histogramu. Ze zkoumání se přepokládá, že $č_{max}$ by měla pokrývat alespoň tuto stanovenou procentuální hranici. Pokud netvoří, je odebrán z SCM.

Je-li podmínka 25 procent splněna, projdou se všechny četnosti PRI v histogramu a odeberou se ty četnosti, kde četnost nedosahuje minimálně 75 procent hodnoty $č_{max}$ (viz. Obrázek 17).



Obrázek 17 - Odstranění nevyhovujících čestností z histogramu PRI

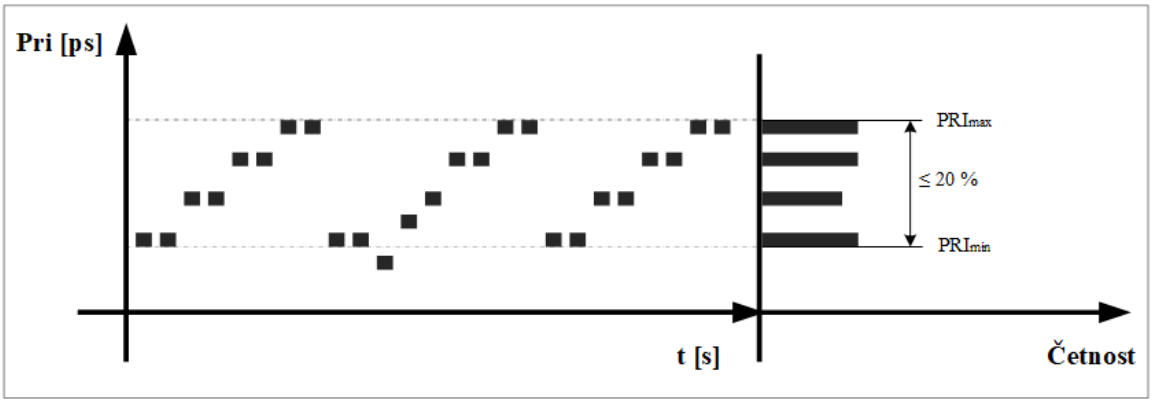
Zdroj: Vlastní

Přepokládá se ze zkoumání, že hodnoty PRI by měli utvářet opakující se vzor. Očekává se, že impulsy budou přicházet v predikované posloupnosti. Minimální četnost $č_{min}$ by se měla pohybovat v minimální vzdálenosti od hodnoty $č_{max}$.

Ve vyfiltrovaném PRI histogramu se nalezne nejmenší hodnota PRI_{min} a PRI_{max}. Vypočte se procentuální velikost intervalu $PRI_{interval}$ výskytu hodnot PRI:

$$PRI_{interval} = \left(\frac{PRI_{max} - PRI_{min}}{PRI_{min}}\right) \times 100$$

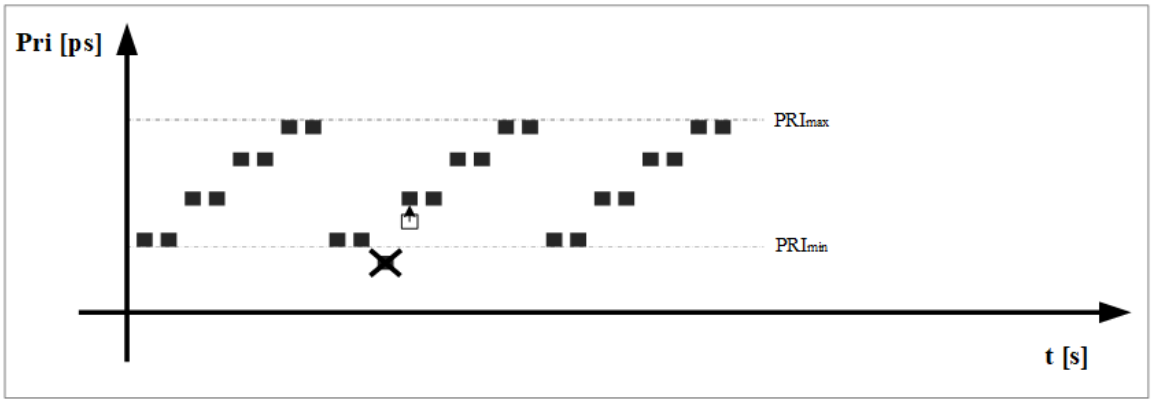
a porovná se s maximální možnou hranicí velikosti intervalu 20 procent. Pokud hodnota $PRI_{interval}$, přesahuje tuto stanovenou hranici, je odebrán procházený adept na kontakt ze SCM. (viz Obrázek 18). Z praktických poznatků a zkoumání vyplynulo, že hodnoty PRI v jednom validním kontaktu se pohybují pospolu.



Obrázek 18 - Rozsah intervalu hodnot PRI

Zdroj: Vlastní

Jestliže hodnota $PRI_{interval}$ nepřesahuje definovanou hranici, odstraní se z procházeného adepta na kontakt v SCM impulsy, které mají menší PRI než nalezené PRI_{min}. Následně dojde k aktualizaci PRI impulsu, jenž se nachází za odebíraným impulsem (viz. Obrázek 19). Odebírány impuls může být cizí nebo se jedná odraz, proto je nutné aktualizovat hodnotu PRI impulsu bezprostředně za ním. Odebíraný impuls může svým zahrnutí do adepta na kontakt zkreslovat tento následující impuls.



Obrázek 19 - Aktualizace hodnot PRI po využití histogramu

Zdroj: Vlastní

V návaznosti se odstraní impulsy s hodnotami PRI vetší než PRI_{max}, kde nedochází k aktualizaci hodnoty PRI následujícího impulsu. Tento druh odebíraných impulsů může představovat chvilkové časové odmlčení (nezachycená rádiová data). Proto není nutné aktualizovat hodnotu PRI následujícího impulsu.

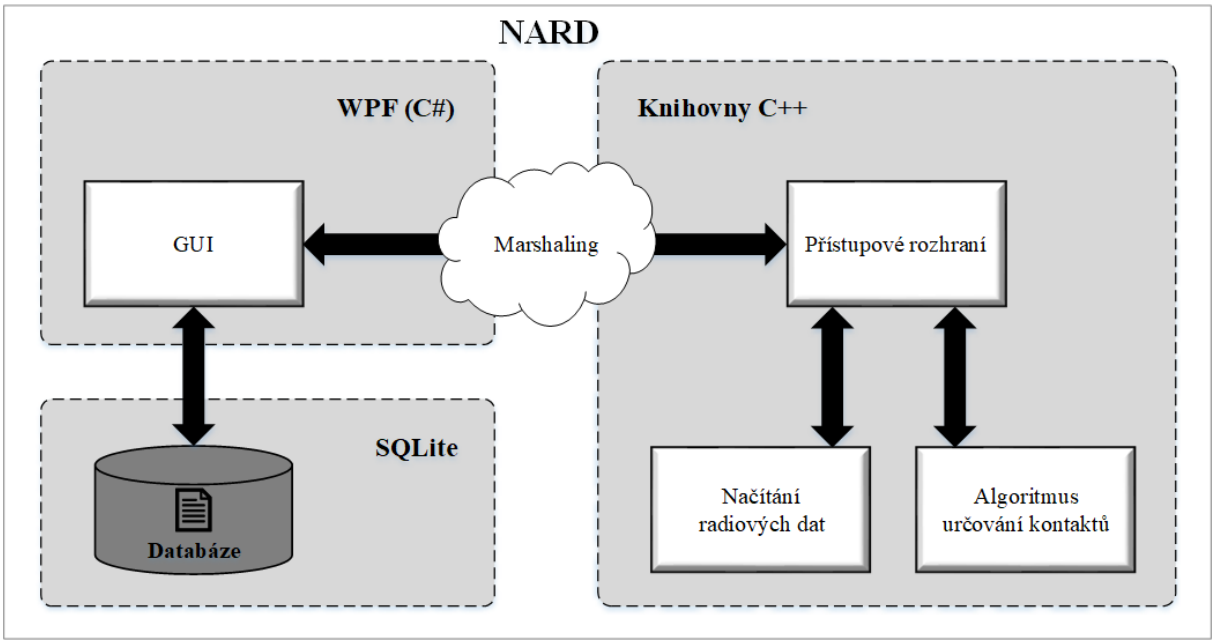
6.3.6 Konečné zpracování

Po skončení procesů analýzy dochází ke konečnému zpracování validních kontaktů, který zůstaly v SCM. Závěrečné zpracování zahrnuje aktualizaci atributu `ContactId` u impulsů, který jsou součástí nějakého validního kontaktu v SCM. Hodnota `ContactId` odpovídá unikátnímu identifikátoru klíče příslušného kontaktu v SCM.

Po dokončení nastavení hodnot `ContactId` přichází na řadu resetování hodnot atributu `Pri` u impulsů, který nejsou přiřazeny k žádnému validnímu kontaktu v SCM. Po výsledných úpravách dojde k výpisu validních kontaktů do textového souboru s příslušnými informace. V daném bodě algoritmus určování kontaktů končí a upozorní se přístupové rozhraní o dokončení analýzy s výsledkem, zda byla úspěšně či neúspěšně provedena.

6.4 Databáze

Přístup k databázi se provádí prostřednictvím GUI aplikace (viz. Obrázek 20). Pokud proběhlo úspěšné načtení radiových dat, jejich zpracování v podobě impulsů a export z přístupového rozhraní do části GUI aplikace, snaží se GUI aplikace tyto impulsy uložit do lokání databáze SQLite. Struktura tabulky pro impulsy je obdobná struktuře impulsu.



Obrázek 20 - NARD – Databáze

Zdroj: Vlastní

Jednotné připojení k databázi a manipulaci s ní je zaručeno pomocí návrhového vzoru *Singleton*. Existují tři možnosti, kterými disponuje manažer pro manipulaci s databází:

- nahrání impulsů,
- stažení impulsů
- a stažení kontaktů.

6.4.1 Nahrání impulsů

Pro nahrání impulsů do databáze se předají zpracované impulsy manažeru a následně nastane navázání spojení s databází. Po úspěšném navázání dochází k vytvoření nové transakce, aby impulsy mohly být nahrány najednou. Nahrání všech impulsů během jedné transakce poskytuje značné rychlostní vylepšení. Po dokončení se transakce uzavře a potvrdí se provedené změny v databázi.

6.4.2 Stažení impulsů

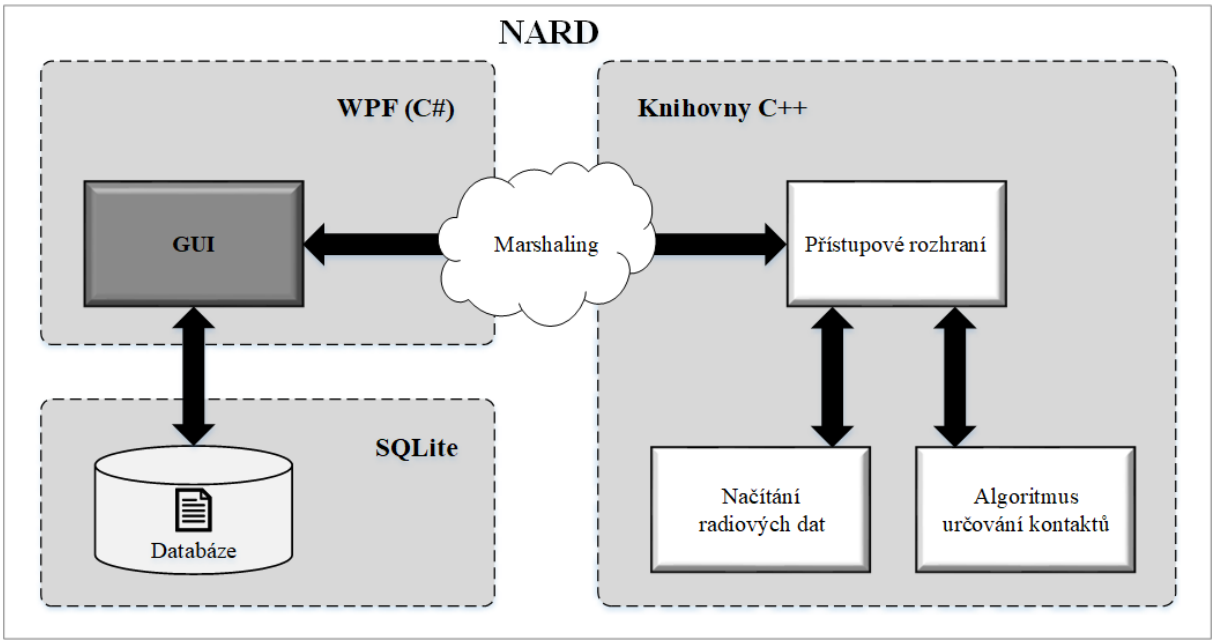
Manažeru, který má na starosti práci s databází, je dána iniciativa ke stažení všech impulsů, jenž jsou obsaženy v připojené databázi. Stažení všech impulsů probíhá najednou, aby rychlost odezvy čekání na impulsy od počáteční iniciativy byla minimální. Následně jsou stažené impulsy zpřístupněny GUI aplikaci.

6.4.3 Stažení kontaktů

Poslední funkcí, kterou manažer pro práci s databází umí, je možnost stažení impulsů, které jsou součástí nějakého kontaktu. Tato funkce umožňuje přistupovat pouze k impulsům, který jsou svým charakterem významným. Stažení kontaktů probíhá na stejné bázi jako stažení všech impulsů z databáze.

6.5 GUI

Součástí NARD je GUI aplikace, které slouží zejména k interakci mezi operátorem a implementovanými funkcemi softwaru. Grafické rozhraní slouží jako diagnostický nástroj, aby operátorovi umožnil shlédnout zpracované impulsy. Disponuje jednoduchou možností manipulace nad načtenými impulsy. Obrázek 21 vyobrazuje GUI jako poslední část, která má na starosti celou správu NARD.

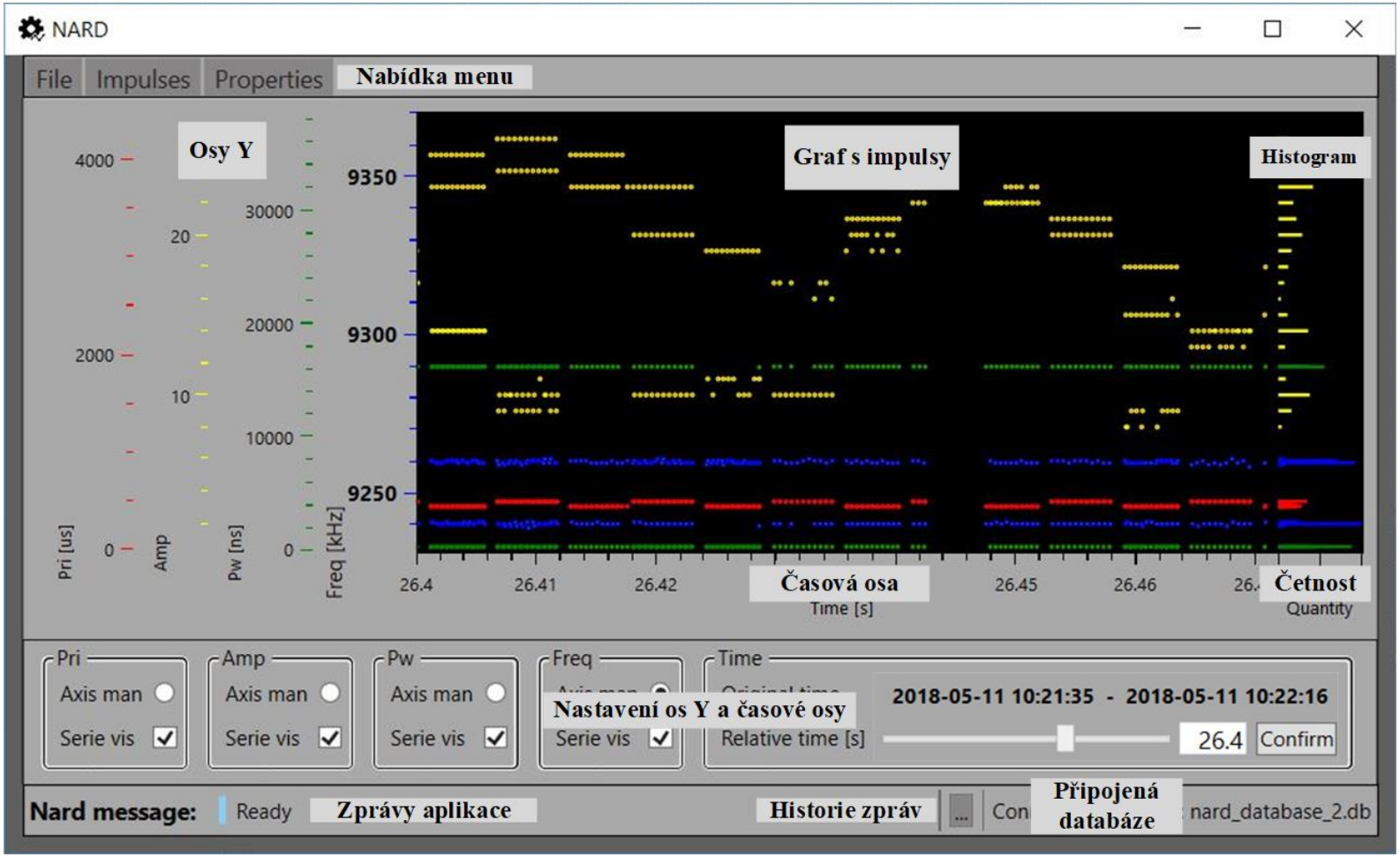


Obrázek 21 - NARD – GUI

Zdroj: Vlastní

Grafické rozhraní je postaveno na jednoduchém principu ovládání. Pro lepší pochopení, jak s grafickým softwarem pracovat, lze aplikaci rozdělit do několika logických celků (viz. Obrázek 21):

- panel nabídka menu,
- panel zpráv aplikace,
- grafy
- a osy X a Y.



Obrázek 22 - Rozložení GUI aplikace

Zdroj: Vlastní

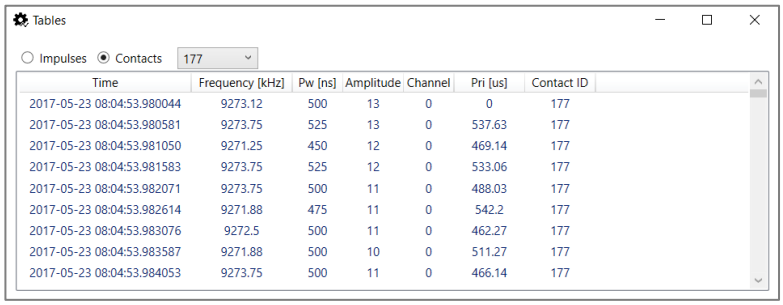
6.5.1 Panel nabídky menu

Nabídka menu slouží jako prvotní interakce, kterou operátor provede. Nabízí tři hlavní možnosti, jak s aplikací pracovat:

- menu *File*,
- menu *Impulses*
- a menu *Properties*.

Menu *File* nabízí hlavní funkci GUI položku *Open* a běžnou funkci grafických aplikací položkou *Exit*. Položka *Open* umožnuje operátorovi vyvolat dialogové okno pro výběr binárního souboru *.dat*. Tento soubor obsahuje zdrojová radiová data. Operátorovi se po výběru souboru otevře další dialogové okno, kde zadá název databáze, do které se budou vytvořené a zpracované impulsy ukládat. Databáze se vytvoří jako soubor SQLite s příponou *.db*. Následuje celý proces od předání cesty k umístění souboru s rádiovými daty až po samotné uložení impulsů do vytvořené databáze. Je-li tento proces úspěšně dokončen objeví se v panelů zpráv aplikace systémové hlášení o vytvořené a připojené databázi s následnou zprávou, která obsahuje kolik impulsů bylo nahráno do databáze. Pokud nastane chyba během celého procesu, je tato chyba zobrazena taktéž v panelu zpráv aplikace.

Menu *Impulses* nabízí položky *Impulses*, *Contacts* a *Impulses – Table*. První dvě položky umožňují operátorovi výběr, zda chce stáhnout a zobrazit všechny impulsy v databázi nebo pouze kontakty s příslušnými impulsy v grafu. Informace o úspěšnosti provedení výběru, se zobrazí operátorovi v panelu zpráv. Poslední položka *Impulses – Table* umožní operátorovi shlédnout stažené impulsy v tabulce (viz. Obrázek 23) s důležitými informacemi, kde si muže vybírat mezi zobrazením všech impulsů nebo impulsů vybraného kontaktu.



Time	Frequency [kHz]	Pw [ns]	Amplitude	Channel	Pri [us]	Contact ID
2017-05-23 08:04:53.980044	9273.12	500	13	0	0	177
2017-05-23 08:04:53.980581	9273.75	525	13	0	537.63	177
2017-05-23 08:04:53.981050	9271.25	450	12	0	469.14	177
2017-05-23 08:04:53.981583	9273.75	525	12	0	533.06	177
2017-05-23 08:04:53.982071	9273.75	500	11	0	488.03	177
2017-05-23 08:04:53.982614	9271.88	475	11	0	542.2	177
2017-05-23 08:04:53.983076	9272.5	500	11	0	462.27	177
2017-05-23 08:04:53.983587	9271.88	500	10	0	511.27	177
2017-05-23 08:04:53.984053	9273.75	500	11	0	466.14	177

Obrázek 23 - Tabulka impulsů

Zdroj: Vlastní

Menu *Properties* nabízí operátorovi možnost *Change db* pro připojení jiné databáze (soubor SQLite s příponou *.db*). Umožnuje to připojit již existující databáze se zpracovanými impulsy. Operátorovi v daném případě stačí pouze jednou zpracovat soubor s radiovými daty.

6.5.2 Panel zpráv aplikace

Panel zpráv je rozdělen na tři části jako:

- zprávy aplikace,
- historie zpráv
- a připojená databáze.

Zprávy aplikace zobrazují systémová hlášení o dění v grafickém rozhraní. Zobrazená zpráva je zobrazována po určitou dobu a následně uložena do historie zpráv.

Historie zpráv představuje tabulku (viz. Obrázek 24), kde se uchovávají všechna systémová hlášení.

Obrázek 24 - Historie zpráv aplikace

Zdroj: Vlastní

Zprávy systémového hlášení jsou tvořeny z:

- časové značky zprávy,
- typu zprávy (úspěch, chyba apod.)
- a ze samostatného textového hlášení.

Poslední část panelu zpráv se týká připojené databáze. Informuje operátora, se kterou databází pracuje popř., že není připojená žádná databáze.

6.5.3 Grafy

Hlavní část GUI představují dva grafy. První graf se zabývá zobrazením impulsů. Jeden impuls je na grafu zobrazen čtyři krát, ale pokaždé závislý na jiné ose Y. U impulsu se zobrazují v čase (osa X) následující hodnoty:

- frekvence (modrá barva),
- délka pulsu (zelená barva),
- amplituda (žlutá barva)
- a hodnota PRI (červená barva).

S grafem může operátor manipulovat pomocí myši. Kliknutím levým tlačítkem na impuls v grafu si muže zobrazit podrobnosti o daném impulsu. Jedním kliknutím na střední tlačítko (kolečko) lze vybrat obdélníkovou oblast pro přiblížení. Dvojklikem na střední tlačítko se resetuje zobrazení do původního stavu. Rolováním kolečkem dochází k přibližování či oddalování grafu (mění se osa X a aktivní osa Y). Kliknutí a podržení pravého tlačítka umožní pohybovat s časovou osou (osa X) a aktivní osou Y.

Druhý graf je označován jako histogram, který je na prvním grafu zcela závislí. Obsahuje sloupcový diagram jednotlivých četností aktuálně zobrazených hodnot (frekvence atd.) v grafu s impulsy. Histogram se aktualizuje průběžně se změnami os Y a X v závislém grafu. Levým klikem myši na jednotlivý sloupec se zobrazí podrobnosti o daném sloupci.

6.5.4 Osy X a Y

Grafy s impulsy a s ním kooperující histogram disponují následujícími osami:

- osa *Pri [us]* (Y – hodnota PRI),
- osa *Amp* (Y – amplituda),
- osa *Pw [ns]* (Y – délka pulsu),
- osa *Freq [kHz]* (Y – frekvence),
- osa *Time [s]* (X – relativní čas impulsu)
- a osa *Quantity* (X – počet hodnot histogramu)

Kromě osy *Quantity* mají ostatní osy společné tyto vlastnosti:

- podržení pravého tlačítka myši na ose lze hodnoty os posouvat,
- dvojklikem středního tlačítka myši lze osy resetovat do původního stavu
- a použití rolování kolečkem lze přibližovat či oddalovat.

Barevné označení os Y je asociováno s barvami v grafech a disponují navíc dodatečným nastavením, kde operátorovi se nabízejí další možnosti manipulace. Pro z každou os Y může operátor provést jednu těchto dvou natavení:

- zobrazení osy Y a příslušných hodnot v grafech (s impulsy a histogram)
- a nastavení osy Y jako aktivní osy při manipulování s grafy.

Pokud operátor zobrazení osy Y vypne, osa zmizí a s ní i hodnoty v grafech. V opačném případě se osa i hodnoty znovu objeví. Vypínání a zapínání viditelnosti pomáhá operátorovi zkoumat pouze hodnoty (jako frekvence apod.), které jsou pro operátora aktuálně užitečné.

Přepínání mezi osami Y, která z nich bude v aktuální okamžik aktivní, umožňuje operátorovi primárně manipulovat (přiblížení, posouvání apod.) s hodnotami impulsů v grafech, jenž jsou závislé na této aktivní ose Y.

Na časové ose X závisí všechny osy Y. Hodnoty časové osy představují relativní čas, kde nulová hodnota představuje čas příchodu prvního impulsu ve stažených impulsech. Tento způsob zobrazení času umožňuje detailněji pracovat se zobrazenými impulsy za pomoci dostupného posuvníku. Rozsah posuvníku je určen z rozdílu skutečných časů mezi prvním a posledním impulsem ve stažených impulsech. Posuvník slouží k nastavení hodnoty přechodu k danému relativnímu času na časové ose X. Nastavenou hodnotu přechodu je nutné potvrdit tlačítkem *Confirm*. Kolečko myši usnadňuje nastavování hodnoty na posuvníku. Informativně je pro operátora poskytnuta informace o skutečném času příchodu prvního a posledního impulsu. Je-li potřeba vidět skutečný čas příchodu každého z impulsů, může operátor tyto časy najít v tabulce impulsů.

Četnostní osa X slouží pouze k zobrazení kvanta jednotlivých sloupců v histogramu. Nedisponuje žádnou možností manipulace.

ZÁVĚR

Na základě stanovených cílů diplomové práce bylo vytvořeno funkční softwarové řešení, které umožní operátorovi analyzovat zdrojová rádiová data pomocí navrženého a zkonstruovaného algoritmu a dostupného grafického rozhraní.

Aplikace byla rozdělena do několika částí s využitím různých technologií nebo procesů tak, aby výsledné řešení bylo efektivní a modulární. Na základě modularity vystupují funkční části aplikace, které mají na starosti načítání rádiových dat a jejich zpracování, jako samostatné knihovny. Knihovny jsou propojeny další knihovnou, která spravuje jejich funkce, a přistupuje k ní grafické rozhraní. Toto rozdělení umožňuje nahrazovat knihovny, aniž by tato substituce nějakým způsobem ovlivnila další knihovny nebo jiné části aplikace. Při následné záměně knihovny za jinou postačí pouze zachování stejného rozhraní původní knihovny. Tento způsob je efektivní v okamžiku, kdy se mění struktura binárních dat nebo při zkvalitnění algoritmu pro analýzu. Z důvodu rychlosti a potřeby práce s velkým množstvím dat na nižší úrovni, kdy je potřeba brát velký ohled na zacházení a využívání paměti, byla pro tyto knihovny zvolena technologie C++.

Grafické rozhraní aplikace, se kterým bude operátor pracovat, je postavena na moderní technologii WPF, a to z toho důvodu, že tato technologie umožňuje více oddělit grafickou vizualizaci od aplikační logiky. Na základě daného přístupu lze vytvářet moderní prostředí pro operátora s využitím moderních prvků a možností, které nejsou k dispozici ve starších formulářových aplikací C#. Ke grafickému rozhraní je připojena databáze SQLite. Technologie SQLite byla zvolena s přihlédnutím na důvěrný charakter zpracovaných dat, možností jednoduché manipulace s uloženými daty a bezproblémové přenositelnosti. Jako další je ke grafickému rozhraní připojena knihovna, která má na starostí správu připojených knihoven pro načítání a zpracování impulsů. Komunikaci mezi grafickým rozhraním (WPF) a knihovnami (C++) obstarává proces marshaling. Proces marshaling je jednoduchý způsob, jak namapovat datové typy v C++ do datových typů C#.

Jedním z cílů závěrečné práce bylo vytvořit algoritmus, jenž bude zkoumat shodu mezi impulsy. Na základě daného cíle bylo v knihovně, která má na starosti analýzu, implementován hrubý algoritmus, který dokáže nacházet shody mezi impulsy. Algoritmus byl vytvořen na základě výsledků analýzy odborné konzultace s osobou z praxe, která se zabývá problematikou zpracování rádiových dat. Na základě jeho znalostí byly v algoritmu definovány postupy, hra-

nice a hodnoty, které umožní hledat shodu mezi impulsy. Implementovaný algoritmus představuje hrubý nástroj, který má operátorovi pomoci při analyzování impulsů. Počítá do jisté míry s vnějšími vlivy, jako jsou přeslechy a odrazy. Naopak nepočítá se situacemi, pokud impulsy mají více charakterů, ale přesto k sobě tyto impulsy patří. Závěrem lze shrnout, že kvalita správnosti určování shod závisí na zdrojových datech. Pokud zdrojová data obsahují příliš mnoho vnějších vlivů nebo impulsů v jeden časový okamžik, věrohodnost výsledků klesá, ale přesto může velmi pomoci operátorovi při vlastní analýze.

Dalším z cílů závěrečné práce bylo vytvořit grafické uživatelské prostředí pro operátora, který bude pomocí implementovaného algoritmu a vlastní analýzy, zkoumat zdrojová data. Grafická aplikace disponuje grafy pro zobrazení zpracovaných impulsů. Nabízí možnost sledovat výsledky aplikovaného algoritmu. Dále je operátorovi poskytnuta sada užitečných nástrojů pro práci s grafy. Aplikace využívá pro zobrazení grafů volně dostupnou knihovnu, která přináší různá omezení. Omezení spočívá v počtu možných zobrazených bodů v grafu. Je-li překročena určitá hranice počtu zobrazených bodů, aplikace se začne zpomalovat. Na základě testování aplikace bylo definováno přirozené maximum pro počet zobrazených bodů v grafu pro plynulost aplikace.

Závěrečná práce přinesla nové možnosti k ulehčení analyzování rádiových dat v oboru analýzy a zpracování rádiových dat. Dále přinesla nové povědomí, jak vytvářet moderní grafické prostředí ve WPF (C#) s využitím knihoven, které jsou vytvořeny v technologii C++. Tato nová znalost umožní, aby na jednotlivé části vyvíjené aplikace mohly být použity vhodné technologie.

POUŽITÁ LITERATURA

[1] ALLAIN, Alex. *Jumping into C++*. San Francisco: Cprogramming.com, 2012. ISBN 978-0-988-92780-3.

[2] ALLEN, Grant a Mike OWENS. *The definitive guide to SQLite [take control of this compact but powerful tool to embed sophisticated SQL databases within your applications!]*. 2nd ed. New York: Apress, 2010. ISBN 978-1-430-23226-1.

[3] ELSHEIMY, Mohammad. *Marshaling with C# – Chapter 1: Introducing Marshaling* [online]. 2010 [cit. 2018-04-07]. Dostupné z: https://www.codeproject.com/Articles/66245/Marshaling-with-Csharp-Chapter-1-Introducing-Marsh.aspx

[4] JAMES, Buddy. *Pro XAML with C#: from design to deployment on WPF, Windows Store, and Windows Phone*. Berkeley, Calif.: Apress, 2015. ISBN 978-1-4302-6775-1.

[5] JAY A. KREIBICH. *Using SQLite*. Sebastopol, CA: O'Reilly, 2010. ISBN 978-0-596-52118-9.

[6] LIPPMAN, Stanley B., Josée. LAJOIE a Barbara E. MOO. *C++ primer*. 5th ed. Upper Saddle River, NJ: Addison-Wesley, c2013. ISBN 978-0-321-71411-4.

[7] MACDONALD, Matthew. *Pro WPF 4.5 in C#: Windows Presentation Foundation with .NET 4.5*. Fourth edition. Berkeley: Apress, 2012. ISBN 978-1-430-24365-6.

[8] MEYERS, Scott. *Effective modern C++: 42 specific ways to improve your use of C++11 and C++14*. Sebastopol, CA: O'Reilly Media, 2014. ISBN 978-1-491-90399-5.

[9] NATHAN, Adam. *WPF 4.5 unleashed*. Indianapolis, Ind.: Sams, 2014. Unleashed. ISBN 978-0-672-33697-3.

[10] PRATA, Stephen. *Mistrovství v C++*. 4., aktualiz. vyd. Přeložil Boris SOKOL. Brno: Computer Press, 2013. Bestseller (Computer Press). ISBN 978-80-251-3828-1.

[11] STROUSTRUP, Bjarne. *Programming: principles and practice using C++*. Second edition. Upper Saddle River, NJ: Addison-Wesley, 2014. ISBN 978-0-321-99278-9.

[12] YOSIFOVICH, Pavel. *Windows Presentation Foundation 4. 5 Cookbook*. New Edition. Birmingham: Packt Publishing, Limited, 2012. ISBN 978-1-849-68622-8.