VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

EXTRAKCE DAT Z DOKUMENTŮ NA ZÁKLADĚ ANALÝZY ROZLOŽENÍ

LAYOUT-BASED DATA EXTRACTION FROM DOCUMENTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MARTIN SEDLÁČEK**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

146958

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Sedláček Martin, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Informační systémy a databáze
Název:	**Extrakce dat z dokumentů na základě analýzy rozložení**
Kategorie:	Informační systémy
Akademický rok:	2022/23

Zadání:

1. Prostudujte dokumentový formát PDF a dostupné nástroje pro čtení a zpracování dokumentů v tomto formátu. Zaměřte se na platformu Java a knihovnu FitLayout pro analýzu dokumentů. Prostudujte i možnosti převodu obrázků na text v tomto kontextu.
2. Na základě analýzy požadavků a po konzultaci s vedoucím navrhněte architekturu rozšiřitelného nástroje pro identifikaci a extrakci dat z PDF dokumentů na základě analýzy rozložení obsahu na stránkách.
3. Implementujte navržený nástroj pomocí vhodných technologií.
4. Proveďte testování na dostupných reálných dokumentech.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Adobe Systems, Inc.: PDF Reference (sixth edition), Version 1.7, November 2006
- Dvořáček, Libor. Využití získávání znalostí pro data z PDF souborů. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- Oltmanová, Kristína. Statistická analýza dat z PDF souborů. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Burget Radek, doc. Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	21.10.2022

Abstrakt

Diplomová práce se zabývá automatizovanou extrakcí dat z lékařských zpráv ve formátu PDF na základě analýzy rozložení dokumentu. Hlavním obsahem práce je uvedení čtenáře do problematiky extrakce dat, srovnávaní existujících nástrojů a představení návrhu a požadavků vyvíjeného nástroje, který bude založen nad aplikačním rámcem FitLayout. Práce dále popisuje samotnou implementaci nástroje v jazyce Java a komentuje výsledky, kterých nástroj dosáhl na reálných datech.

Abstract

This thesis deals with automated data extraction from medical reports in PDF format based on document layout analysis. The main content of the thesis is an introduction to data extraction, a comparison of existing tools and a presentation of the design and requirements of the developed tool, which will be based on the FitLayout application framework. The thesis then describes the actual implementation of the tool in Java and comments on the results achieved by the tool on real data.

Klíčová slova

PDF, extrakce, data, FitLayout, Java, Swing, FN Bnro, rozložení dokumentu, získávání dat

Keywords

PDF, extraction, data, FitLayout, Java, Swing, FN Brno, document layout, layout-based, data extraction

Citace

SEDLÁČEK, Martin. *Extrakce dat z dokumentů na základě analýzy rozložení.* Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

Extrakce dat z dokumentů na základě analýzy rozložení

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod odborným vedením pana doc. Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Sedláček
16. května 2023

Poděkování

Na tomto místě bych rád poděkoval doc. Ing. Radkovi Burgetovi, Ph.D. za vstřícnost, ochotu a cenné rady při vedení mé práce. Dále bych chtěl poděkovat mým kolegům, rodině a přátelům za podporu a shovívavost při mém studiu.

Obsah

Kapitola 1

Úvod

Dokumentový formát PDF se postupem času stal téměř standardem v oblasti sdílení a ukládání elektronických dat. Téměř každý uživatel počítače se s tímto formátem setkává de facto na denní bázi. Avšak s takovou popularitou přicházejí i různá úskalí. Ne vždy je formát PDF tím nejvhodnějším způsobem jak ukládat nějaká data a to platí především v oblastech jako je věda a medicína. Velké množství takovýchto odborných dat je ukládáno v podobě PDF reportů někde na pevných discích a jediným možným způsobem, jak s takovými daty následně pracovat je jejich ruční procházení.

Takovýto způsob uložení dat však může v některých oblastech činit přinejmenším potíže. V dnešní době, kdy znalost dat představuje obrovskou výhodu pro jakékoli odvětví je tento způsob nakládání s daty velmi nevyhovující. A jelikož ne vždy jsme schopni měnit formát dat, která dostáváme na výstupu různým zařízení či měřících přístrojů, uchylují se organizace či společnosti k řešením, které pracují na základě extrakce dat. To má výhodu mimo jiné také v tom, že tímto způsobem lze převést již existující data, která by za jiných okolností byla nepoužitelná.

Právě takovým extrakčním nástroje pro získávání dat z lékařských zpráv se zabývá tato diplomová práce. Práce je psaná ve spolupráci s Interní kardiologickou klinikou při FN Brno a pojednává o možnostech a realizaci extrakčního nástroje, který bude schopen získávat důležitá data z kontrolních zpráv implantovaných srdečních kardiostimulátorů a defibrilátorů pacientů.

Celkově je tato diplomová práce členěna do osmi kapitol. V kapitole 2 je popsána samotná motivace pro extrakci dat, také je nastíněn způsob uložení dat v PDF dokumentech a jsou přiblíženy základní přístupy, které jsou uplatňovány při extrakci dat. Následující kapitola 3 pak popisuje a srovnává jednotlivé dostupné nástroje pro získávání dat z PDF dokumentů a současně také popisuje aplikační rámec FitLayout, který je požit jako základ navrhovaného nástroje. Dále jsou v kapitole 4 popsány technologie zvolené pro tuto práci. Kapitoly s čísly 5 a 6 pak popisují návrh a implementaci nástroje. Kapitoly definují jednotlivé požadavky, které jsou na nástroj kladeny, popisují vstupní data a komentuje zdrojový kód nástroje. V posledních dvou kapitolách 7 a 8 je pak nástroj otestován na reálných data a jsou prezentovány výsledky. V závěru jsou pak shrnuty celkové dosažené výsledky celé této diplomové práce.

Kapitola 2

Proces a motivace získávání dat ze souborů PDF

Text této kapitoly slouží zejména jako úvod do celkové problematiky práce s PDF dokumenty. Popisuje hlavní důvody a motivace ukládání dat v této podobě, zabývá se samotným formátem PDF dokumentů a porovnává některé fundamentálně odlišné způsoby uložení dat. V neposlední řadě pak dále nastiňuje základní možné principy extrakce dat z takovýchto dokumentů.

2.1 Motivace a náročnost extrakce dat

Se soubory ve formátu PDF se setkal téměř každý běžný uživatel počítače. To je způsobeno zejména tím, že míra využití tohoto formátu je ve světe digitálních dokumentů velmi velká. Využití PDF souborů pro ukládání dat je velmi rozličné – od běžných textových souborů, přes skenované dokumenty až po velmi strukturované výstupy různých měření a pokusů. Tyto výstupy mohou být tvořeny jak manuálně uživatelem, tak automaticky generovány.

Nejčastěji jsou PDF dokumenty používány a zamýšleny pro účely uživatelského čtení a sdílení. Avšak v praxi nám může vyvstat požadavek, že je nutné takovéto *lidsky čitelné* dokumenty zpracovat nějakým způsobem strojově a automaticky.

Takovýto požadavek nejčastěji pramení z toho, že některé společnosti používají formát PDF jako výstupní formát svých přístrojů a zařízení, avšak to je pro další práci s těmito daty velmi nevhodné. Při velkém počtu takovýchto výstupních souborů vyvstávají problémy jako obtížné vyhledávání informací, obtížná analýza vývoje dat v čase a nebo nemožnost provádění jakýchkoli výpočtů nad takovými daty. Data v tomto formátu jsou tak předurčena pouze pro lidské čtení a manuální extrakci informací.

Právě kvůli výše zmíněným důvodům se začaly vyvíjet nástroje a systémy pro extrakci dat právě z PDF dokumentů, díky kterým je možné získat data ve strojově čitelné podobě a s takovými daty pak dále pracovat.

Před samotným získáváním dat je však nutné si uvědomit, jaké jsou problémy právě pro extrakci takových dat z dokumentů. Nejjednodušším případem pro extrakci dat jsou soubory obsahující pouze holý text členěný například do odstavců (příkladem může být například tato diplomová práce v elektronické podobě). Takový text je pak velmi lehce extrahovatelný a je možné ho využít téměř bez dalších změn.

Na druhé straně spektra pak ale stojí nejsložitější případ – to jsou data, která jsou uložena v určité hierarchické struktuře, kde i pozice na samotné stránce udává datům určitý

význam. Příkladem takových strukturovaných mohou být různé tabulky, diagramy či grafy. Extrakce z takovýchto souborů je poté o poznání komplexnější, ale v praxi mnohem více užitečná. Právě extrakcí takovýchto hierarchicky uspořádaných dat se proto zabývá tato práce.

Potřeba pro extrakci dat v praxi je velmi citelná a přínos, který takto získaná data přináší je nezanedbatelný. Je tedy zřejmé, že existuje nespočet nástrojů, které se této problematice věnují. Jelikož se však jedná o úkol, který je velmi rozličný, je velmi často potřeba nástroje nějakým způsobem modifikovat či vytvářet nové, aby sloužili přesně daným účelům.

2.2 PDF soubory a jejich formát

Formát souborů PDF vyvinula v roce 1991 společnost Adobe[1]. PDF je zkratka anglického slovního spojení *Portable Document Format* (česky přenosný formát dokumentů) a jedná se o souborový formát, který slouží pro ukládání digitálních dokumentů a to nezávisle na tom, na jakém zařízení či v jakém nástroji byly pořízeny či prohlíženy. Základní myšlenkou tohoto formátu totiž bylo, že dokumenty by měly vypadat na všech zařízení stejně – přesně tak jak byly vytvořeny.[1]

Formát dokumentů PDF umožňuje do dokumentů uložit nejenom běžný text a grafiku (tabulky, obrázky), ale dovoluje uložení i interaktivního obsahu jako jsou formulářové prvky, video, 3D grafika či zvuk.

Formát PDF je také od roku 2008 uznáván jako otevřený standard pod záštitou Mezinárodní organizace pro normalizaci (ISO 32000-1:2008), což zaručuje jeho jasný a přesně daný význam a strukturu [6].

Soubory PDF je možné otevřít a editovat v různých, volně dostupných programech, které jsou dostupné pro téměř všechny používané platformy. Dále pak i většina webových prohlížečů je schopná tyto soubory otevřít a dále s nimi v omezené míře pracovat. Nejznámější a nejpoužívanějším externím nástrojem pro prohlížení PDF souborů je pak zdarma dostupný program Acrobat Reader[2] od právě zmiňované společnosti Adobe.

2.2.1 Základní struktura a vnitřní uložení souboru PDF

Následující odstavce týkající se technických detailů PDF dokumentů jsou částečně přejaty z [3]. Základní vnitřní struktura samotného PDF souboru je rozdělaná na 4 části:

- **hlavička** obsahuje verzi PDF, která je pro dokument použita
- v **těle** dokumentu jsou uloženy samotná data – tedy texty, fonty, obrázky, formulářové prvky a jiné
- **tabulka odkazů** obsahuje odkazy na všechny prvky do těla souboru – slouží pak zejména k náhodnému přístupu k objektům stránky a také dovoluje provádění malých inkrementálních změn
- **závěrečná sekce** obsahuje lokaci tabulky odkazů v celém souboru

V samotném těle jsou pak data uloženy jako objekty různých druhů, kdy každý jeden typ z celkových osmi typů objektů má přesně danou strukturu zápisu a dat. Jednotlivé objekty je možné do sebe i navzájem zanořovat.

[1] Stránky společnosti dostupné na https://www.adobe.com/cz/

[2] Informace o programu k dispozici na https://www.adobe.com/acrobat/pdf-reader.html

2.2.2 Souřadnicový systém

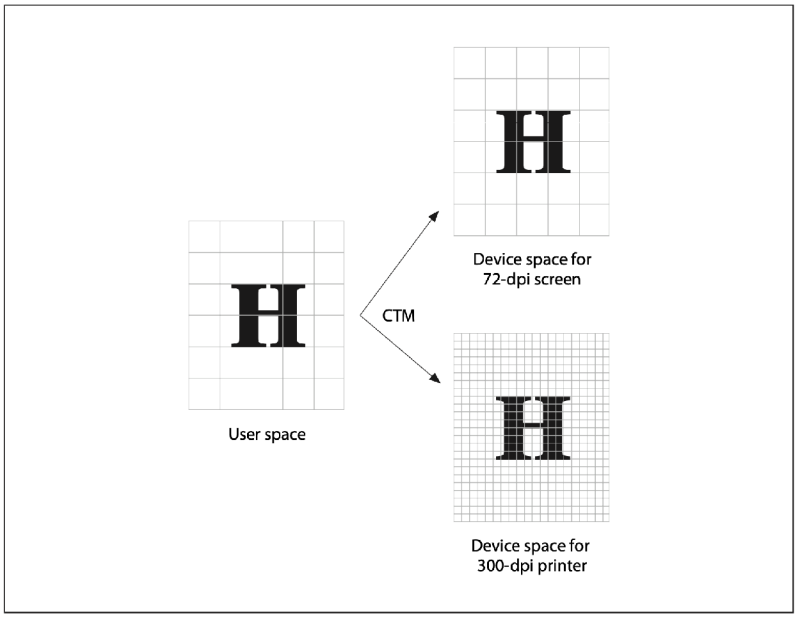
Umístění jednotlivých elementů (*objektů*) na stránkách zajišťuje v případě PDF souborů souřadnicový systém. Jedná se o dvojrozměrnou kartézskou soustavu souřadnic, která prakticky vystihuje výsledný fyzický papír, na něž může být dokument vytištěn.

V PDF dokumentech rozlišujeme kvůli zachování jednotného vzhledu výstupu dva typy souřadnicových systému.

První z nich se nazývá *User space* (česky volně přeloženo jako Uživatelský prostor), který definuje samotné umístění dat na stránce. Tento prostor je pevně definován v samotné struktuře PDF dokumentu. Jeho počátek se zpravidla nachází v levém dolním rohu a jednotkou tohoto prostoru jsou body. Tyto body mají ve výchozím stavu definovanou hustotu 72 bodů/palec. Avšak veškeré tyto výchozí hodnoty parametrů mohou být v dokumentu uživatelsky předefinovány.

Druhým souřadnicovým systémem je pak takzvaný *Device space* (česky volně přeloženo jako Prostor zařízení), který definuje umístění jednotlivých objektů vzhledem k výstupnímu zařízení, které dokument reprezentuje. Takovým výstupním zařízením může být například monitor, tiskárna či plotr. Tento souřadnicový systém je důležitý, protože každé výstupní zařízení má jiné parametry a specifika (například rozlišení), a je tedy nutné, aby každé takové zařízení pro správné vykreslení mělo odpovídající prostor.

Prostory mezi sebou definují takzvané *Transformation matrices* (česky přeloženo jako matice transformací), které určují převod, mezi jednotným *User space* do různých výstupních *Device spaces* jak je zobrazeno na obrázku 2.1 [3].



Obrázek 2.1: Převod *User space* do různých *Device spaces*. Zkratka CTM znamená *Current transformation matrix* a definuje používanou matici transformace [3]

2.3 Základní přístupy extrakce dat z PDF dokumentů

Jak již bylo zmíněno, extrakce informací z PDF dokumentů je velice důležitá úloha. Avšak tato úloha není tak triviální, jak se na první pohled může zdát.

Na rozdíl od jiných, běžně používaných, formátů má PDF jeden zásadní rozdíl. Není primárně určen k editaci což je jeden z důvodů, proč jsou data v něm uložena tak, aby výstup byl stejný, ale není zaručeno, že si data drží svůj původní tvar.

Může se tedy stát, že dokument, obsahující například pouze text bude mít své věty *rozkouskované* na několik částí, které budou napozicované v dokumentu tak, aby vypadaly jako souvislé věty, avšak jejich textová hodnota bude uvnitř v dokumentu rozdělená.

Tento problém může různým extrakčním metodám způsobit více či méně potíží a podle toho bude potřeba přiměřené množství lidské kontroly.

V této části bych rád popsal jednotlivé možné přístupy na extrakci dat z dokumentů a určitým způsobem provedl jejich vzájemné srovnání.

2.3.1 Manuální extrakce

Prvním ze způsobů získávání dat z PDF dokumentů je manuální extrakce. Jedná se o manuální úkon prováděný člověkem, při kterém z otevřeného dokumentu opisuje nebo kopíruje jeho jednotlivé části do jiného výstupního dokumentu.

Tato metoda je nejjednodušší z hlediska použitých technologií a uživatelských znalostí, jelikož při ní není potřeba nic jiného než program pro zobrazení PDF souborů a uživatel.

Výsledky této metody jsou velmi dobré, protože každý text prochází lidskou kontrolou. To na druhou stranu ale může vést k uživatelským chybám. Hlavní nevýhodou této metody avšak je její časová náročnost – celý proces manuální extrakce je složitý a zabere uživateli netriviální množství času. Je proto zcela nevhodný pro velké množství dat.

2.3.2 Konverze do jiného formátu

Další variantou, kterou lze pro získání dat použít je konverze PDF dokumentu do nějakého editovatelného formátu. Nejčastěji převod probíhá do běžně používaných textových formátů – například `.txt`, `.docx` a jiné.

K takovému účelu jde najít nespočet nástrojů, které pracují jak on-line tak offline, některé placené, některé zdarma. Každý z takových nástrojů má jiné přednosti a je schopný převést dokumenty s různou přesností.

Takto převedené dokumenty jde pak dále zpracovat a data z nich už poté získat mnohem jednodušeji. Je však ale důležité myslet na to, že tento převod nemusí být zcela bezchybný a může tak vyprodukovat nepřesné výsledky.

V neposlední řadě je při využívání on-line nástrojů také velmi důležité myslet na bezpečnost, jelikož data, která převádíme nahráváme na server provozovatele služby. A v případě například citlivých dat by tento způsob nebylo možné zvolit.

2.3.3 Pokročilé extrakční nástroje

Poslední variantou, jak získat informace z PDF souborů je použití nějakého pokročilého extrakčního nástroje. Většina těchto nástrojů funguje na principu analýzy vstupního souboru a vytvoření výstupu, který obsahuje všechny data společně s informací o struktuře těchto dat.

Nejčastěji jsou tak data extrahována do struktury připomínající hierarchický strom a jednotlivé uzly stromu reprezentují jednotlivé objekty v dokumentu. Tyto uzly pak nesou mimo samotných dat i další informace jako pozici objektu, velikost fontu, druh objektu a další.

Tyto nástroje jsou k dispozici buď jako samostatné aplikace nebo jako knihovny, které slouží pro implementaci do vlastních řešení. V případě samostatných aplikací pak většinou bývá k dispozici i určitá forma uživatelského grafického rozhraní, kde uživatel vidí jednotlivé extrahované části přímo ve zdrojovém dokumentu.

Výstupy této metody je poté možné dále upravovat a filtrovat pro získání co nejpřesnějších výsledků.

Velkou výhodou tohoto přístupu je, že výstupy jsou velmi přesné a rychlé. Takovýmto způsobem je možné zpracovat velké množství dokumentů podle předem nadefinovaných pravidlem.

Nevýhodou pak je, že se většinou jedná o komplexní a komerční nástroje a je tedy třeba počítat i s pořizovací cenou licence, která není vždy zcela zanedbatelná a může se vyplatit až v případě většího počtu dokumentů.

Kapitola 3

Nástroje pro extrakci dat z PDF dokumentů

Jak již bylo zmíněno, automatická extrakce dat z PDF dokumentů není jednoduchý úkol. Z toho důvodu není vždy zcela výhodné vytvářet svůj vlastní software pro extrakci dat zcela od začátku, ale je výhodnější využít nějaké již existující řešení.

Řešení a nástrojů, jež se věnují právě extrakci informací z PDF dokumentů, existuje na trhu nepřeberné množství. Cílem této kapitoly tedy je představit a srovnat některá z nich a dále pak představit aplikační rámec FitLayout, který bude použit pro implementaci řešení v rámci této diplomové práce.

3.1 Základní nástroje s uživatelským grafickým rozhraním

První skupinou nástrojů pro extrakci dat jsou grafické nástroje. Ty jako hlavní způsob ovládání používají grafické uživatelské rozhraní (zkratka *GUI* z anglického *Graphic User Interface*), přes které uživatel prohlíží dokumenty a definuje jednotlivé akce nad nimi.

3.1.1 Adobe Acrobat

Nástroj Adobe Acrobat[1] je nejpoužívanějším a nejrozšířenějším nástrojem na zobrazení a editaci PDF dokumentů. Je vyvíjen společností Adobe. Již ze základu je tento program placený, disponuje však dvěmi úrovněmi licence – Standard a Pro.

Slouží zejména na zobrazování, anotaci a úpravu dokumentů ve formátu PDF.

Při načtení dokumentu detekuje text a umožní jeho editaci i změnu pozice či smazání. K dispozici je také možnost konverze dokumentu do různých editovatelných formátů, včetně rozpoznání a konverze tabulek.

Velkou výhodou je, že tento software je vyvíjen společností, jež stojí za samotným vznikem formátu PDF. Je tedy očekávané, že podpora tohoto formátu bude nejlepší možná. Na druhé straně nevýhodou je, že se jedná pouze o placený software, jehož cena není zrovna nízká.

[1] Program ke stažení na adrese https://www.adobe.com/cz/acrobat.html

3.1.2 PDFelement

Nástroj PDFelement[2] vyvinutý společností Wondershare slouží k zobrazení a editaci dat. Jeho základní verze je k dispozici zdarma, avšak většina pokročilých funkcí je dostupná ve zpoplatněné PRO verzi – lze koupit jak doživotní licence, tak program používat na bázi předplatného.

Hlavní funkcí tohoto softwaru (*pouze v PRO verzi*) je editace a úprava dokumentů. Při načtení dokumentu program vyhodnotí jeho obsah a označí jednotlivé vizuální oblasti. Ty jde následně editovat, přeskupovat a extrahovat.

Nástroj také nabízí možnost extrakce veškerých dat, avšak ta funguje pouze pro extrakci dat z formulářových prvků. Výrobce sice uvádí, že extrakce je možná i z obyčejných textových polí, avšak v aktuální verzi jsem nebyl schopný toho docílit [16].

Dále pak nástroj nabízí konverzi dat do jiných formátů a to včetně rozpoznání a konverze tabulkových dat.

Jako nevýhodu bych opět uvedl cenu tohoto řešení, což může být pro některé uživatele či organizace zábranou.

3.1.3 PDFTables

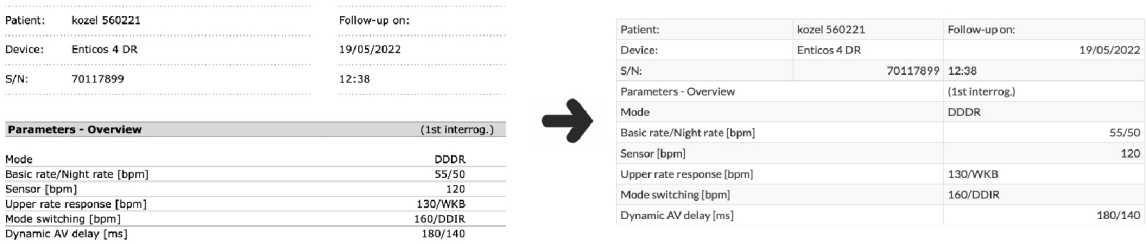
Další nástroj sice nedisponuje plnohodnotným uživatelským rozhraním v pravém slova smyslu, avšak i přesto si v této kategorii zaslouží zmínku.

Jedná se o jednoduchý webový nástroj[3], který po nahrátí PDF dokumentu tento dokument konvertuje do jeho tabulkového ekvivalentu a ten zobrazí uživateli s možností stáhnutí.

To v praxi znamená, že celý dokument je rozdělen na jednotlivé vizuální oblasti a z těch je vytvořena tabulka, která se snaží co nejlépe kopírovat strukturu a umístění dat v původním dokumentu.

Největší výhodou je, že výstupy, jež tento nástroj produkuje jsou velmi kvalitní a konzistentní. (Ukázka výstupu konverze na obrázku 3.1)

Nevýhodou je, že se jedná o placený nástroj (prvních několik stránek je zdarma, poté se platí za každou stránku), ale také to, že se jedná o on-line nástroj. To tedy vylučuje jakékoli použití pro práci s tajnými či citlivými daty.



Obrázek 3.1: Příklad konverze části PDF dokumentu na tabulku za použití nástroje PDFTables

[2]Webové stránky programu na https://pdf.wondershare.com

[3]Nástroj dostupná na adrese https://pdftables.com

3.2 Knihovny pro programovací jazyk Java

Další skupinou nástrojů, pomocí kterých lze získávat data z PDF dokumentů jsou knihovny. Ty nabízejí uživateli určitou funkcionalitu, avšak ta je použitelná pouze pokud je začleněna do nějakého již existujícího softwarového řešení.

Z důvodu, že hlavním programovacím jazykem, jež bude použit při implementaci nástroje pro tuto diplomovou práci, je Java, tak následující sekce bude pojednávat o knihovnách a nástrojích právě pro tento programovací jazyk.

3.2.1 Adobe PDF Extract API

Adobe PDF Extract API[4] je provozováno společností Adobe a jedná se o komerční API (zkratka z anglického *Application Programming Interface* označuje aplikační rozhraní) sloužící pro extrakci dat z PDF dokumentů. Tato služba je zahrnuta v této kategorii nástrojů, jelikož nástroj nabízí i oficiální sadu SDK (zkratka z anglického Software development kit označuje sadu vývojových nástrojů) pro jazyk Java (a mnohé další).

Jedná se o službu, jež extrahuje veškerá data ze vstupního PDF dokumentu a výstupem je soubor ve formátu JSON. Ten obsahuje informace o všech logicky oddělených elementech společně s jejich metadaty jakou jsou pozice, text, font, velikost a další. Služba umí extrahovat nejenom text, ale i obrázky nebo tabulky.

Výhodou tohoto řešení je opět vývoj samotnou společností Adobe a s tím spojená podpora a spolehlivost. Jako další výhoda je, že Adobe nabízí velkou rodinu API pro veškeré možné účely, takže je velmi jednoduché do systému implementovat i jinou další funkcionalitu [2].

Jako nevýhodu lze zmínit cenu, kdy je účtován každý požadavek [5]. Dále pak také to, že se jedná o API, z čehož plyne, že veškeré výpočty se dějí mimo výpočetní stanici uživatele na cizích výpočetních serverech a je tedy vyloučeno použití pro tajná a citlivá data.

3.2.2 PDFTron

PDFTron[6] je knihovna pro jazyky Java, která nabízí téměř veškeré nástroje pro práci s PDF dokumenty a to včetně pokročilých extrakčních nástrojů.

Nabízí nejenom klasické extrakční nástroje pro získání veškerých dat, ale disponuje i pokročilými extraktory, které se specializují na různé typy dat jako jsou texty, tabulky či obrázky. Například při extrakci textů shlukuje jednotlivé slova a věty do skupin na základě jejich vizuální podoby a blízkosti umístění.

Dále pak disponuje třídou, jež pro extrakci dat používá strojové učení s pomocí kterého je schopná ještě lépe oddělovat a extrahovat jednotlivé logické a vizuální bloky. Těmto blokům je také schopná přiřadit význam, jež v původním dokumentu měly. [7]

Nevýhodou může být, že pro komerční účely je nutná speciální placená licence.

[4] Postup instalace a dokumentace je k dispozici na https://developer.adobe.com/document-services/apis/pdf-extract/

[5] Služba je provozovaná jako *Pay-as-you-go*, což znamená, že se platí pouze služby, které uživatel opravdu využije

[6] Návod na instalaci a dokumentace je k dispozici na https://www.pdftron.com/documentation/java/

3.3 Aplikační rámec FitLayout

Tato sekce se blíže věnuje aplikačnímu rámci (anglicky *framework*) FitLayout, který je použit jako základ řešení vyvíjeného nástroje pro tuto diplomovou práci. Celá tato sekce je částečně přejata z oficiální dokumentace frameworku [4].

Jedná se o rozšiřitelný framework napsaný v jazyce Java určený na renderování webových stránek a PDF dokumentů, jejich modelování a analýzu.

Framework nabízí několik možností, jak jej lze použít či implementovat do vlastních systémů. Těmito rozhraními jsou:

- **Aplikační rozhraní v jazyce Java**, pomocí kterého lze využít FitLayout jako knihovnu při tvorbě vlastních Java aplikací.
- **Rozhraní příkazové řádky** (anglická zkratka CLI), které lze využít buď samostatně nebo jinou aplikací, zejména takovou, která není napsaná v jazyce Java.
- **Aplikační rozhraní REST**, které je využitelné, chceme-li využít FitLayout jako webovou službu.

3.3.1 Artefakty

Základem celého aplikačního rámce jsou artefakty (anglicky *artifacts*), což jsou výstupy všech fázi analýzy dokumentů. Artefakty vznikají již při renderování stránky a dále se s nimi pracuje v každém dalším kroku. Nejčastěji jsou artefakty seřazeny do hierarchické struktury – stromu. Všechny artefakty kromě kořenového jsou potomkem jiného artefaktu.

Artefakty vznikají výhradně prostřednictvím tzv. *artifact service* (volně přeloženo jako *služeb artefaktů*) a to buď zcela nově nebo vytvořením z již stávajících artefaktů.

Artefakty ve frameworku FitLayout mohou být následujícího typu:

- **Stránka** (anglicky *page*, také ale *box tree*) přímo reprezentující vykreslenou stránku.
- **Strom oblastí** (anglicky *area tree*) reprezentující výsledky segmentace a obsahující hierarchicky uspořádané vizuální oblasti stránky.
- **Strom logických oblastí** (anglicky *logical area tree*) je tvořen logickými oblastmi, které sestávají z množiny vizuálních oblastí.
- **Množina bloků textu** (anglicky *text chunk set*) reprezentující jednotlivé obdélníkové oblasti obsahující textové celky.

3.3.2 Služby artefaktů

Jak již bylo zmíněno výše, služby artefaktů jsou hlavní funkční jednotkou celého frameworku. Následující odstavce tedy popisující některé tyto služby, které jsou fundamentální pro řešení této diplomové práce.

Vykreslovací jádro PDF Slouží pro samotné vykreslení PDF dokumentů. Pro jeho implementaci byla použita knihovna *Apache PDFBox*. Výstupem této služby je pak artefakt *Stránka*.

Segmentační algoritmus Basic Page Jedná se o algoritmu, který je založen na segmentaci zdola nahoru. Detekuje tak jednotlivé větší vizuální oblasti, které seskupuje do uzlů. Na svém vstupu přijímá *Visual box tree* (upravený artefakt *Stránka*) a výstupem pak je artefakt *Strom oblastí*.

Box tree post-procesor VisualBoxTree Jedná se o službu k následnému zpracovaní artefaktu *Stránka* a vytvoření tzv. *Visual box tree*, který dále přijímá segmentační algoritmus.

3.3.3 Operátory

Framework FitLayout dále také definuje několik operátory, které lze využít pro následnou práci se *Stromy oblastí*.

Operátor řazení SortByPosition Slouží k seřazení jednotlivých oblastí ve *Stromu oblastí* dle jejich pozice výskytu na stránce. Tento operátor řadí oblasti zleva doprava, shora dolů.

Kapitola 4

Další použité technologie pro vývoj nástroje

Tato kapitola pojednává o technologiích použitých v této práci. V podkapitolách níže se nachází stručný popis programovacího jazyku Java a použitých knihoven. Dále pak také popis knihovny použité pro vytvoření grafického rozhraní nástroje a sestavovacího nástroje Maven.

U každé ze zmíněných technologií je pak stručně zmíněn důvod jejího výběru a také, zda existují případné alternativy.

4.1 Programovací jazyk Java

Programovací jazyk Java[1] vyvinula koncem minulého století společnost *Sun Mycrosystems*. Následně byla celá platforma Java SE převedena pod společnost *Oracle*, která platformu spravuje doposud. I přesto, že v posledních letech zájem o tento programovací jazyk upadá (hlavně na úkor nových, rychle se vyvíjejících technologií), stále se řadí mezi nejpopulárněji a nejžádanější jazyky na světě [13].

Jak již bylo zmíněno výše, Java není pouze programovací jazyk, ale jedná se o celou platformu jež umožňuje vývoj a spuštění programů napsaných v jazyce Java.

Mezi hlavní výhody celé platformy pak zajisté patří samotná portabilita, jelikož prostředí Java je používáno v širokém spektru zařízení od osobních počítačů, přes mobilní zařízení, servery či dokonce superpočítače. Dle zařízení, na kterém výsledný kód bude běžet se pak Java dělí na několik platforem [5].

Java ME (z anglického *Micro Edition*) je platforma pro aplikace provozované na mobilních zařízení. Důraz je dbán především na omezený paměťový a výpočetní výkon.

Java SE (z anglického *Standard Edition*) je platforma pro vývoj obyčejných desktopových a serverových aplikací. Většina nástrojů – včetně toho vyvíjeného v rámci mé práce – je napsána právě nad touto platformou.

Java EE (někdy také Jakarta EE, z anglického *Enterprise Edition*) je platforma, která rozšiřuje Javu SE o různé technologie a nástroje určené pro vývoj rozsáhlých podnikových informačních systému.

[1] Oficiální stránky jazyku Java jsou dostupné zde https://www.oracle.com/java/

JavaCard je platforma, která umožňuje tvořit malé aplikace (tzv. *applety*), které bezpečně běží na malých čipových kartách.

Samotný programovací jazyk se pak řadí do kategorie interpretovaných, silně typovaných jazyků. Jde o objektově orientovaný jazyk, který řadu konstruktů a syntax převzal od jazyků jako C nebo C++, kde však oproti nim odpadla řada nízkoúrovňových konstrukcí.

Při překladu se jazyk převádí do takzvaného *bajtkódu*, který je poté vykonáván interpretrem – virtuálním strojem Javy (anglická zkratka JVM z *Java Virtual Machine*). Samotný interpretr až poté překládá bajtkód na strojový kód a je tak zajištěna kompatibilita se všemi zařízeními a architekturami, na kterých běží JVM.

Správa paměti je řešena pomocí automatického *garbage collectoru*, který hlídá a vyhledává nepoužité části paměti a uvolňuje je pro další použití.

Programovací jazyk Java byl pro tuto práci zvolen zejména pro jeho přenositelnost a kompatibilitu s různými platformami a není tak třeba řešit, na jakém zařízení bude nástroj spouštěn. Jako další důvod bych pak uvedl existenci knihovny FitLayout, která je napsaná v jazyce Java a je tedy velmi snadné její použití.

4.1.1 Knihovna Gson

Jazyk Java ve své standardní knihovně nenabízí žádné funkce pro práci s daty ve formátu JSON[2]. Pro práci s těmito daty je tedy třeba použít externí knihovnu.

V této práci jsem zvolil knihovnu Gson[3]. Jedná se o knihovnu vyvíjenou společností *Google*, která slouží pro převod dat mezi formáty JSON a objekty jazyku Java.

Nabízí jednoduché aplikační rozhraní pro převod mezi jednotlivými formáty a její hlavní výhodou je, že pro převod není potřeba modifikovat samotné třídy, které jsou pro převod použity. Je tedy možné tuto knihovny použít i pro objekty, které jsou z cizího zdrojového kódu a není tak možná jejich modifikace.

Tuto knihovnu jsem vybral pro svoji práci hlavně pro její jednoduché použití, ale také proto, že společnost jež knihovnu vyvíjí je velice známá a její reputace je dobrá.

Jako další možné alternativy by šlo použít například knihovny *org.json*[4] nebo *Jackson*[5].

4.1.2 Knihovna opencsv

Další nutnou funkcí vyvíjeného nástroje bylo zpracování souborů ve formátu CSV[6]. To jde v jazyce Java zajistit pouze za pomocí standardní knihovny a CSV soubory si generovat sám, avšak pro jednodušší použitelnost je lepší použít externí knihovnu.

Pro tuto diplomovou práci byla zvolena knihovna opencsv vyvíjená společností *Maven*. Nabízí základní funkcionalitu pro čtení a zápis souborů ve formátu CSV a to včetně ošetření veškerých výjimek jako například přítomnost čárky v textovém řetězci nebo hodnoty, jež obsahují znaky konce řádků.

Knihovna byla opět vybrána pro její jednoduché použití a jako možná alternativa se nabízí například knihovna *Apache Commons CSV*[7].

[2] Zkratka z anglického *JavaScript Object Notation* označuje textový datový formát určený pro přenos dat. Jeho základními prvky jsou pole, objekty typu klíč-hodnota a samostatné hodnoty.

[3] Zdrojový kód a dokumentace dostupná na https://github.com/google/gson

[4] Dostupné na https://mvnrepository.com/artifact/org.json/json

[5] Dostupné na https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind

[6] Zkratka pro Comma-Seperated Values označuje jednoduchý, textový, obecně uznávaný formát pro tabulková data.

[7] Dostupné na https://commons.apache.org/proper/commons-csv/

4.1.3 Knihovna FitLayout

Pro práci s PDF soubory byla zvolena knihovna FitLayout. Byla zvolena pro její jednoduché použití a množství funkcí. Detailněji popsaná byla v kapitole 3.3.

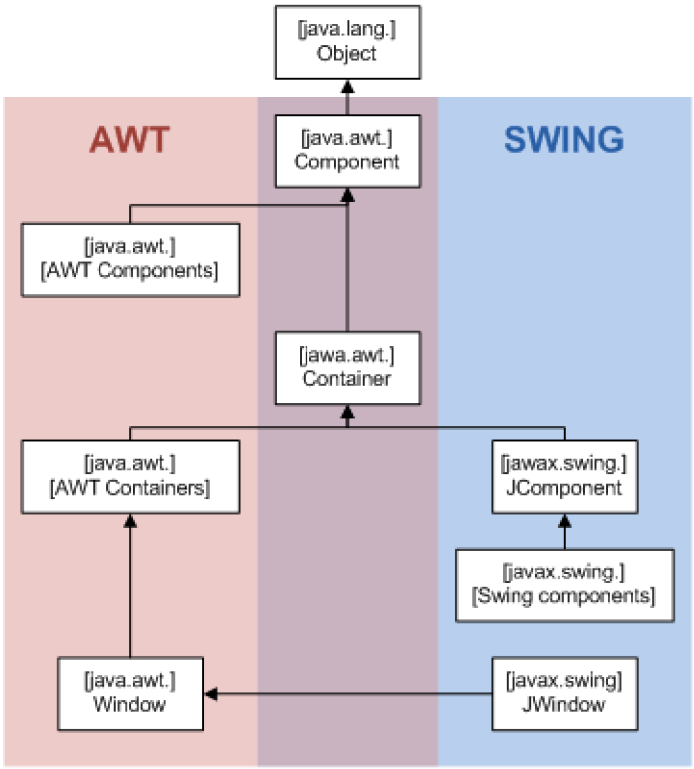
4.2 Knihovna uživatelského rozhraní Swing

Swing[8] je knihovna prvků pro grafické uživatelské rozhraní pro jazyk Java. Samotná knihovna patří do JFC (zkratka z anglického *Java Foundation Classses*), což je programové aplikační rozhraní (angl. *API*), jež poskytuje grafické uživatelské rozhraní pro javovské aplikace a programy.

Swing je postaven nad původním nástrojem AWT (zkratka z anglického *Abstract Window Toolkit*), jež byl vyvíjen společností Sun Microsystems. Avšak jelikož se v nástroji objevili chyby, bylo od jeho vývoje opuštěno a na jeho základě vznikl Swing.

Z toho důvodu veškeré objekty a třídy Swingu jsou založeny na třídách z původního AWT. To jde jde vidět například na schématu hierarchie tříd 4.1.

Základní nadtřídou všech objektů ve Swingu je třída `Component`. Všechny instance třídy `Component` mají grafickou reprezentaci a je tedy možné jejich vykreselení na obrazovku. Dalším důležitým stavebním kamenem je třída `Container`, jež je schopná v sobě sdružovat více komponent a vykreslovat je tak dohromady. Pořadí a pozice jednotlivých komponent je poté spravováno pomocí `LayoutManager`ů (možný volný český překlad je *Správci rozložení*), jejíchž použití však není povinné [17] [8].



Obrázek 4.1: Diagram hierarchie tříd komponent AWT a Swingu v Javě. Převzato z [15].

[8]Oficiální dokumentace knihovny dostupná na https://docs.oracle.com/javase/tutorial/uiswing

4.3 Sestavovací nástroj Apache Maven

Aplikace v jazyce Java musí být pro svůj běh sestaveny (angl. *build*). K tomu lze použít nástroje, které nám tuto úlohu usnadní a zautomatizují. Takovým nástrojem je Apache Maven[9], který je vyvíjen neziskovou organizací *Apache Software Foundation*.

Jedná se o nástroj, který slouží k automatizovanému sestavování projektů napsaných v jazyce Java. Mimo samotného sestavování aplikace usnadňuje také správu závislostí, jejich stahování a provázání.

Nástroj pro svoje fungování používá konfigurační soubor (`pom.xml`) ve formátu XML, který se nachází v kořenovém adresáři celého projektu. Tento soubor slouží k tomu, že celý projekt popisuje jako *Project Object Model*. Jsou v něm specifikovány veškeré základní informace o projektu jako jsou název, verze, autor nebo popis. Dále pak obsahuje popis toho, jak je cílový program sestaven a také informace o všech potřebných externích knihovnách. Tyto závislosti jsou pak při sestavení automaticky staženy a uloženy do lokálního úložiště pro další použití [12].

Maven je postaven na modulární architektuře a je tedy možné ho rozšířit za použití různých zásuvných modulů (anglicky *pluginů*). Samotný Maven se pak stará o dodání a spouštění takto definovaných rozšíření.

Samotný Project Object Model také dále definuje doporučenou adresářovou strukturu projektu, což vytváří jakýsi standard, jak jsou veškeré projekty využívající Maven členěny. Přestože tato struktura není vynucována a lze v konfiguračním souboru změnit, je její použití silně doporučené [14].

Jako alternativa k Apache Maven lze k sestavení aplikací v jazyce Java použít i nástroj *Gradle*[10].

[9] Oficiální stránky nástroje dostupné na `https://maven.apache.org`
[10] Dostupný na `https://docs.gradle.org/current`

Kapitola 5

Návrh nástroje pro zpracování lékařských dat

Hlavní náplní této práce je vytvoření nástroje, který pomůže s automatickou extrakcí dat z lékařských zpráv. Jedná se o výstupy, které jsou extrahovány z implantovaných kardiostimulátorů či defibrilátorů (*pro jejich označení se používá jednotná zkratka CIEDs*) pacientů při pravidelných kontrolách. Tato data jsou uloženy ve formátu PDF a cílem je z těchto zpráv data extrahovat.

Tato kapitola tedy popisuje jednotlivé kladené požadavky na finální nástroj a dále pak náležitosti vstupních dat a možný návrh funkcionality nástroje.

5.1 Zadavatel

Zadavatelem práce je Fakultní nemocnice Brno, Interní kardiologická klinika[1]. Na této klinice vznikla myšlenka, že data – PDF zprávy z CIEDs pacientů – by bylo dobré převést do strojově čitelné podoby a to jak pro účely samotné léčby a sledování pacientů, ale také pro účely výzkumu a sledování trendů.

Jelikož dat je velké množství, má být tento převod obstarán automaticky pomocí počítačového nástroje, který má být spustitelný lokálně – na počítačích FN Brno – a to z důvodu práce s citlivými informacemi.

5.2 Požadavky na funkcionalitu

Jak již bylo zmíněno, cílem této práce je navrhnout nástroj pro automatickou extrakci určitých dat z PDF dokumentů do strojově čitelné podoby. Při návrhu takového nástroje se vycházelo z požadavků, jež na systém jsou kladeny. Tato podkapitola tyto jednotlivé požadavky blíže specifikuje.

Jednotlivé funkční (FP) a nefunkční (NP) požadavky vycházejí ze základních předpokladů, které by systém měl splňovat ale také z očekávání, které na systém klade jeho zadavatel.

[1] Stránky kliniky k dispozici na https://www.fnbrno.cz/interni-kardiologicka-klinika/k1451

5.2.1 Funkční požadavky

FP1 Nástroj extrahuje z PDF dokumentů data, která byla specifikovaná vstupním konfiguračním řetězcem/souborem.

FP2 Nástroj vyhledává informace dle textových kotev (angl. *anchors*) zvolených v textu.

FP3 Nástroj podporuje zpracování více podobných dokumentů zároveň.

FP4 Nástroj jako svůj výstup vytváří soubor, kde jednotlivé extrahované hodnoty jsou ve tvaru klíč-hodnota. Pro více vstupních souborů pak takovým výstupem může být například soubor ve formátu CSV.

FP5 Vstupní konfigurační řetězec musí mít dostatečně jednoduchý formát, aby nástroj byl použitelný i pro běžného uživatele PC. Například formát JSON.

FP6 Nástroj extrahuje požadovaná data přesně a s minimální chybovostí.

5.2.2 Nefunkční požadavky

NP1 Nástroj je možné spustit lokálně na počítači uvnitř FN Bnro kvůli ochraně dat.

NP2 Nástroj používá již existující aplikační rámec FITLayout.

NP3 Nástroj musí být dostatečně rychlý, aby bylo možné zpětně zpracovat velké množství dat v přijatelném čase.

NP4 Nástroj je ovládán pomocí jednoduchého grafického rozhraní.

5.3 Formát vstupní konfigurace

Hlavní úlohou vyvíjeného nástroje je cílená extrakce dat z dokumentů na základě jejich pozice a rozložení dokumentu. Tuto informaci o tom, jaká data se mají z dokumentů extrahovat je třeba předat jako vstup pro každý typ souborů. Proto nástroj přímá na svém vstupu při zpracování dokumentů vstupní konfiguraci, která definuje, která data budou z dokumentu získána.

Za účelem co největší jednoduchosti a případné rozšířitelnosti nástroje byl jako formát vstupní konfigurace zvolen JSON.

5.3.1 Popis struktury objektu vstupní konfigurace

Vstupní konfigurace se skládá z pole objektů, kdy každý objekt reprezentuje jednu extrahovanou hodnotu. Samotné objekty poté mají definované povinné a volitelné klíče, pod kterými uživatel vyplňuje hodnoty a definuje tak, jaká data budou získána.

V tabulce 5.1 je vypsán seznam možných klíčů, které lze ve vstupní konfiguraci použít. Příklad možné vstupní konfigurace, jež extrahuje jméno pacienta a datum další kontroly je pak možné vidět na ukázce kódu 5.1

Tabulka 5.1: Definice klíčů objektu vstupní konfigurace

Jméno klíče	Typ hodnoty	Popis	Výchozí hodnota	Povinný
`anchor`	`string`	Definuje textový řetězec, který je vyhledáván v dokumentu a lokalizuje extrahovaná data.	-	ano
`position`	`string`	Definuje pozici extrahovaných dat vhledem ke kotvě (`anchor`). Povolené hodnoty jsou `right` a `down`.	-	ano
`key`	`string`	Definuje klíč, pod kterým bude hodnota uložena ve výstupním souboru	-	ano
`skip`	`number`	Udává počet vizuálních bloků dat, které budou přeskočeny než budou extrahována data	`0`	ne
`length`	`number`	Udává počet souvislých vizuálních bloků, které budou extrahovány	`1`	ne

```
[
  {
    "anchor": "Patient:",
    "position": "right",
    "key": "Patient name"
  },
  {
    "anchor": "Follow-up on:",
    "position": "down",
    "key": "Follow-up date",
    "length": 2
  }
]
```

Kód 5.1: Příklad vstupní konfigurace

5.4 Popis a analýza vstupních lékařských zpráv

Pro správnou implementaci výsledného nástroje je třeba vědět s jakými vstupními daty bude nástroj pracovat. Od zadavatele jsem tedy obdržel soubor lékařských zpráv, jež je třeba nástrojem strojově zpracovat a extrahovat z nich informace.

Jelikož samotné implantovaná měřící zařízení vyrábí několik různých výrobců, mají i výstupní lékařské zprávy každá jiný formát. Dále se pak druh dokumentu liší i dle dat které obsahuje a jaký účel samotný report má (například zpráva o chybách, pravidelný report, report o nastavení přístroje a jiné).

Všechny dokumenty však spojuje jedna věc – typ dat, jež samotné zprávy obsahují. Ve většině případů se jedná o naměřené hodnoty jednotlivých implantovaných měřičů, které jsou prezentovány v dokumentech nejčastěji ve formě tabulek, doplněných grafickými prvky jako grafy či diagramy. Většina textů v tabulkách je uložena jako text (je tedy možné text přímo extrahovat). Na druhou stranu, grafy a grafické prvky jsou ve většině dokumentů uloženy v rastrovém formátu a jejich extrakce je tedy náročná.

Dodané dokumenty ze základu dělíme na 4 kategorie dle výrobce zařízení. V následujících podkapitolách jsou popsány náležitosti jednotlivých skupin dokumentů. Příklady některých typů dokumentů jsou pak k dispozici v příloze A.

5.4.1 Biotronic (BIO)

Dle dodaných dokumentů je patrné, že dokumenty společnosti Biotronic mají tři formáty, kde podle názvu souborů lze usuzovat, že rozdíl v těchto formátech je dle jejich data vzniku. Označujme tedy formáty jako *starý*, *nový* a *nový-verze2*.

Všechny tyto formáty mají hodně společných znaků a většinou se liší pouze formátováním. Dokumenty se skládají z velké části z tabulek, jež obsahují různé naměřené hodnoty. Obsahují také grafy, jež jsou všechny v rastrovém formátu. Jednotlivé verze mají vždy jednotnou hlavičku obsahující základní informace o pacientovi.

Samotný obsah a data, které dokumenty obsahují se velmi liší dle modelu implantovaného přístroje. To znamená, že každý model přístroje má v reportu obsažená jiná data. Jednotlivé dokumenty jsou tedy rozděleny i dle typu přístroje.

Starý formát je specifický svým úzkým provedením. Hlavička v tomto formátu je rastrová, zbytek dokumentu je však již uložen jako text.

Nový formát je nejčetněji zastoupen. Obsahuje tabulku tvořící hlavičku, která se opakuje na každé straně dokumentu. Hlavní tabulky s daty mají vizuálně označené záhlaví.

Nová formát – verze 2 je velmi podobný předešlému formátu. Na rozdíl od něj má však jednodušší a moderněji vypadající hlavičku, ve které nejsou data uvozeny slovem co znamenají – to znamená, že například jméno pacienta je pouze napsáno v reportu velkým písmem ale není uvozeno nadpisem *"Patient:"*. Další rozdíly jsou pak spíše zanedbatelné.

5.4.2 Boston scientific (BSCI)

Dodané dokumenty mají jednotný formát, avšak typ dokumentů se liší zřejmě dle typu generovaného reportu. Každý typ dokumentu má svoje specifika a náležitosti.

Některé typy dokumentů jsou zastoupeny v dodaných datech hojně, jiné spíše sporadicky. Z toho a také z jejich názvu lze vyvozovat, že nejčastěji zastoupené dokumenty obsahují nejdůležitější informace a méně zastoupené obsahují data spíše doplňková.

V práci tedy budu pracovat primárně se čtyřmi typy dokumentů, jež jsou zastoupeny v největším počtu. Jsou to *Arrhythmia logbook report*, *Combined Follow-up report*, *Quick notes report* a *Device Settings Report*.

Všechny dokumenty obsahují hlavičku, která se napříč typy dokumentů mírně liší, avšak obsahuje vždy název reportu. Hlavička je vždy obsažena pouze na první straně dokumentu.

Arrhythmia logbook report obsahuje tabulku zobrazující srdeční arytmie. Tato tabulka je kompaktní (tj. jeden záznam na řádek) a obsahuje základní informace jako čas, typ a dobu trvání arytmie.

Combined Follow-up report obsahuje úvodní stranu s jednotnou hlavičkou a tabulku se základními informacemi. Na dalších stranách jsou různé tabulky, grafy a grafické prvky. Grafy i grafické prvky jsou rastrové.

Quick notes report je jednostránkový dokument obsahující tabulku se základními daty a jednotnou hlavičku.

Device Settings Report obsahuje jednotnou hlavičku a tabulky s daty.

5.4.3 Medtronic (MDT)

U tohoto výrobce mají jednotlivé dokumenty nejvyšší variabilitu v rámci obsahu i v rámci formátování. Společné znaky napříč všemi dokumenty se u tohoto výrobce hledají velmi špatně a obsah informací záleží jak na typu zařízení tak i na typu generovaného reportu.

Podobnosti formátování jde nalézt, rozdělíme-li dokumenty dle typu zařízení. Pro nalezení podobnosti v datech je pak nutné rozdělit dokumenty i dle typu reportu.

Většina dokumentů má nějakou společnou hlavičku obsahující základní data o pacientovi. Dále pak reporty obsahují data v tabulkách a různé grafy. Všechny grafy v dokumentech nejsou vloženy jako obrázky ale text a linie obsahují jako objekty PDF dokumentu (text, obrazce, křivky a jiné).

5.4.4 St. Jude Medical (SJM)

Dokumenty tohoto výrobce mají jednotný grafický formát. Data obsažený v dokumentech se liší dle typu reportu. Data jsou uloženy většinou v tabulkách a grafech avšak dokumenty obsahují i jiné grafické prvky. Obsahují různé sloupcové indikátory, či i data reprezentují ikonami či obrázky.

Veškerý obsah dokumentu je uložen v rastrové podobě a před zpracováním by bylo nutné nejprve celý obsah převést na textovou reprezentaci.

5.4.5 Extrahovaná data

Specifikace, jaká data mají být z dokumentů extrahována bohužel nebyla zadavatelem dodána. Na úvodní schůzce se mluvilo o základních datech o pacientovi a stavu jeho měřícího přístroje.

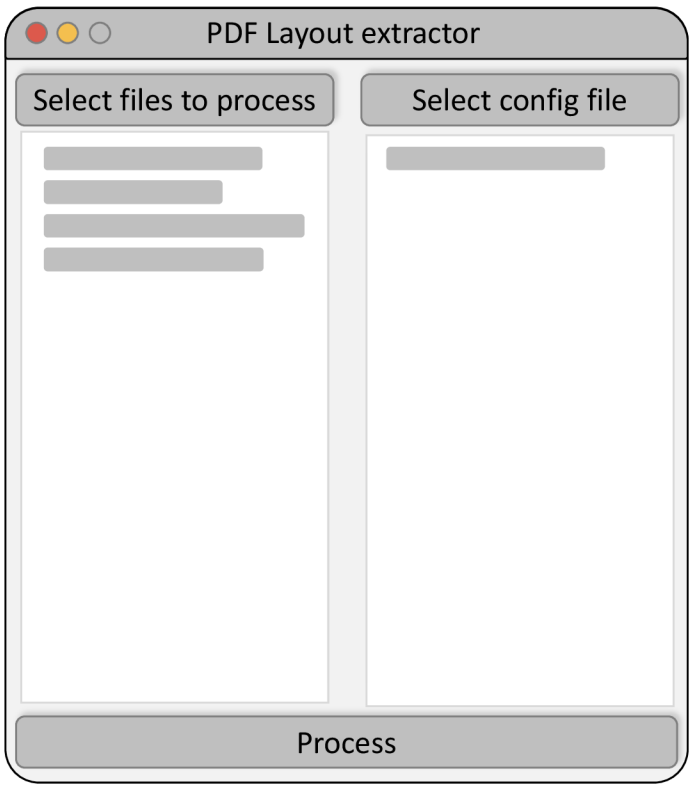
Pro účely této diplomové práce a jako prostředek demonstrace funkčnosti tedy budou při experimentech používány smyšlené úlohy, které nepochází z praxe.

5.5 Návrh grafického uživatelského rozhraní

Z povahy jednoduchosti aplikace bude samotné grafické uživatelské rozhraní také spíše jednoduché. Grafické rozhraní musí uživateli umožnit zvolit několik souborů pro zpracování, dále pak zvolit konfigurační soubor a v neposlední řadě zvolit umístění výstupního souboru. Dále pak grafické rozhraní nabídne ukazatel průběhu samotného zpracovávání dokumentů, aby měl při déle trvajících úlohách uživatel možnost tento postup sledovat.

Většina grafických prvků bude použita ze zvolené knihovny *Swing*. Jelikož je tato zmíněná knihovna velmi populární a rozšířená, nabízí již ze základu mnoho grafických elementů, které lze pro vývoj grafického rozhraní použít. Další výhodou pak je i to, že kvůli popularitě jsou prvky grafického rozhraní už mezi uživateli zažité a pomáhají tak ke snadnějšímu použití samotného nástroje.

Uživatelský pohled tedy bude obsahovat dva sloupce. První sloupec bude sloužit pro volbu souborů pro zpracování, druhý sloupec pak pro volbu konfiguračního souboru. Oba sloupce budou po zvolení souborů tyto soubory zobrazovat. Po vyplnění obou vstupů pak bude uživatel moci spustit proces extrakce pomocí hlavního tlačítka ve spodní části okna. Zobrazení postupu pak bude řešeno dialogovým oknem obsahujícím ukazatel celkového postupu (angl. *progress bar*). Drátěný model (angl. *wireframe*) grafického rozhraní je zobrazen na obrázku 5.1.

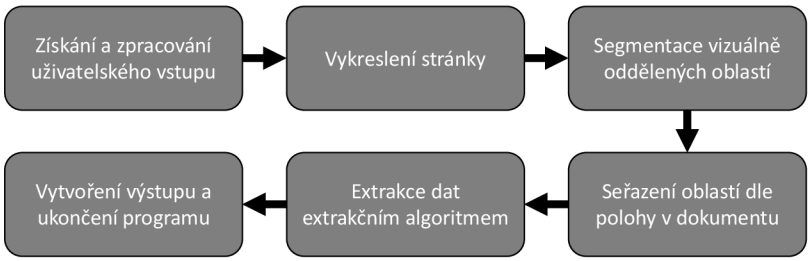


Obrázek 5.1: Návrh drátěného modelu uživatelského rozhraní aplikace

5.6 Architektura nástroje

Zpracování vstupních souborů lze rozdělit do několik kroků, které dohromady tvoří celý proces zpracování. V následujících odstavcích je popsáno, jaké kroky jsou pro zpracování potřeba provést. U jednotlivých kroků je pak nastíněn návrh jejich funkcionality. Kroky jsou také graficky vyjádřeny diagramem 5.2.

Pro implementaci samotného nástroje bude použit aplikační rámec FITLayout (blíže popsán v kapitole 3.3). Tento aplikační rámec nám již ze základu nabízí několik funkcí a nástrojů, které se stanou základními kameny námi vytvořené aplikace.



Obrázek 5.2: Diagram jednotlivých kroků zpracování dokumentu

1. **Získání uživatelského vstupu** Prvním krokem je obdržení uživatelského vstupu – tedy vytvoření grafického rozhraní pro zadání těchto informací.
2. **Zpracování vstupu** Následuje zpracování uživatelského vstupu. To znamená zpracování informace o vstupních souborech (validace jejich formátu, získání ze souborového systému). Tento krok zahrnuje také validaci vstupní konfigurace z níž jsou vytvořeny pravidla (a kotvy) pro hledání jednotlivých informací.
3. **Vykreslení stránek** Dalším krokem je vykreslení stránek PDF souborů pomocí vhodného vykreslovacího jádra. Tímto postupem získáme reprezentaci jednotlivých stránek.
4. **Segmentace vizuálně oddělených oblastí** Následně za použití vhodného algoritmu segmentuje jednotlivé výrazně oddělené vizuální oblasti. Získáme tím reprezentaci hierarchického stromu jednotlivých oblastí.
5. **Seřazení oblastí dle pozice v dokumentu** Pro následující krok je nutné jednotlivé vizuální oblasti seřadit dle jejich pozice v originálním dokumentu. Seřadíme tedy strom jednotlivých oblastí zleva doprava, shora dolů.
6. **Nalezení a extrakce dat dle vstupní konfigurace** V tomto kroku probíhá samotná extrakce dat ze vstupních dokumentů. Prochází se jednotlivé dokumenty (resp. jejich stromy oblastí) a hledají se shody vůči definovaným pravidlům. Popis návrhu samotného algoritmu pro vyhledávání shod a extrakce dat na základě polohy vůči kotvám je popsán v podkapitole 5.7
7. **Vytvoření výstupu** Výstupní data z extrakčního algoritmu jsou vyčištěna a uložena do výstupního souboru. Nejsou-li doposud zpracovány všechny soubory, pokračuje se dále bodem 3 pro zpracování dalšího dokumentu.
8. **Výstup a ukončení programu** Po dokončení zpracování všech vstupních dokumentů je proces ukončen a extrahovaná data jsou uložena do výstupního souboru.

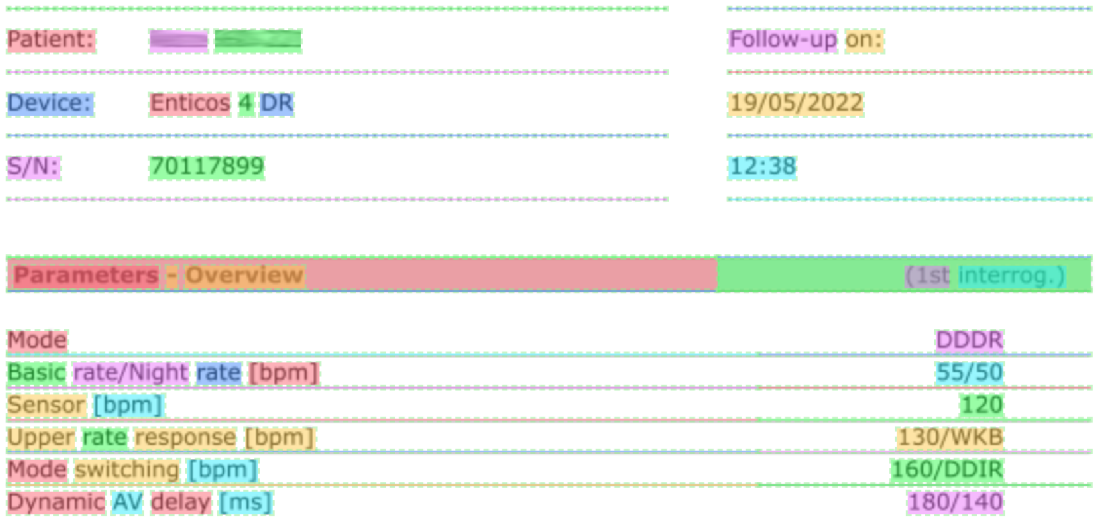
5.7 Návrh algoritmu pro extrakci informací na základě polohy

Hlavní podstatou nového extrakčního nástroje je samotná extrakce dat dle uživatelské konfigurace (bod 6 v sekci 5.6). Následující odstavce proto popisují způsob, jakým budou relevantní data v dokumentu vyhledávána a následně získávána.

Na vstupu samotného algoritmu jsou již zpracovaná vstupní konfigurace a zpracovávaný dokument reprezentovaný stromem oblastí.

Vstupní konfigurace je na vstup dodána již zpracovaná ve formě vnitřní reprezentace. Obsahuje všechny potřebné informace k extrakci jako kotvu, délku výstupu ale také vygenerované pořadové číslo. Toto pořadí poté bude použito k seřazení nalezených dat do výstupního souboru.

Strom oblastí je zpracovaná reprezentace dokumentu, jež umožňuje jeho procházení na základě polohy jednotlivých prvků. Samotný strom se skládá z uzlů, jež reprezentují jednotlivé logické prvky na stránce – prvky, jež jsou logicky odděleny bílým místem. Jednotlivé uzly stromu jsou na obrázku 5.3 překryty přes původní dokument a jsou reprezentovány barevnými obdélníky. Jedná se tedy o jednotlivá slova či slovní spojení. Tyto prvky jsou již z předchozího kroku seřazeny – shora dolů, zleva doprava – tak, aby procházení samotného stromu bylo co nejjednodušší. Jednotlivé oblasti obsahují informaci o pozici v kartézském systém souřadnic (tj. pozici na ose x a y) a také samotný textový obsah oblasti či průměrnou velikost písma.



Obrázek 5.3: Příklad stromu oblastí jednoho z dokumentů. Barvy pozadí jsou pouze ilustrační pro lepší vizuální oddělení jednotlivých oblastí. *Z důvodu ochrany citlivých osobních informací jsou některé údaje rozmazány.*

5.7.1 Slovní popis algoritmu pro extrakci dat

1. Algoritmus prochází přes všechny oblasti stromu. Při průchodu si ukládá polohy posledních zpracovaných oblastí a počítá změny poloh vůči aktuálně zpracovávané oblasti.

2. Při zpracování každé oblasti vyhodnotí, zda se jedná o signifikantní změnu polohy.

   - Signifikantní změna v **souřadnici x** značí bílé místo v řádku – tedy začátek nového vizuálního bloku dat na jednom řádku.
   - Signifikantní změna v **souřadnici y** značí začátek řádku nového – opět tedy začátek nového vizuálního bloku dat.

3. Při průchodu jsou shlukována data, která nemají signifikantní změnu ani v jednom směru. Tím je docíleno, že slovní uskupení či spojení jsou vnímána jako jeden logický blok dat.

4. Nad sdruženými daty jsou vyhledávány kotvy ze vstupní konfigurace. Při nalezení kotvy je do paměti uložen objekt (tzv. hledací pole), který drží informaci o tom za kolik bloků a kterým směrem se nachází daná data.

5. Při konci každého bloku jsou vyhodnoceny všechny možné nalezená data dle hledacích polí. Splňují-li data v aktuálním zpracovávaném bloku požadavky na pozici, kde se dle hledacího pole má informace nacházet, je tato informace přidána do hledacího pole.

6. Naplní-li hledací pole svoji délku nebo počet přeskočení, je takové hledací pole uloženo do kolekce hotových hledacích polí.

7. Takto se zpracují všechny oblasti ze stromu oblastí a na výstup je vrácena kolekce hotových hledacích polí.

Kapitola 6

Implementace

6.1 Struktura zdrojového kódu

Zdrojové kódy aplikace se nachází v balíku `cz.vutbr.fit`, který je dále členěn na jednotlivé adresáře. Jména adresářů ve stromové struktuře jsou popisné a usnadňují orientaci ve zdrojovém kódu.

Soubor `Main` a `FileProcessor` se nachází mimo adresářovou strukturu, jelikož se jedná o hlavní vstupní body vyvíjeného programu. Dále se pak ve stromové struktuře nachází následující adresáře:

- `data` – obsahuje třídy vlastních datových typů, jež jsou použité v rámci extrakce dat
- `extractor` – obsahuje rozhraní pro implementaci jednotlivých extraktorů a také implementaci extraktoru na základě rozložení
- `ui` – obsahuje třídy sloužící pro vytvoření grafického uživatelského rozhraní aplikace
- `utils` – obsahuje třídy sdružující pomocné funkce
- `workers` – obsahuje asynchronní workery (česky volně přeloženo jako *zpracovávače*) událostí uživatelského rozhraní, které slouží pro zpracování dlouho běžících úloh

6.2 Běh aplikace s frameworkem Swing

Každá Java aplikace má sestavu vláken (anglicky *Initial Threads*), které používá pro svůj běh. Standardní aplikace bez uživatelského rozehraní používají pro svůj běh většinou pouze jedno vlákno, které se stará a spuštění programu `main` a jeho následné vykonání.

U aplikací využívajíce pro své ovládání grafické uživatelské rozhraní Swing je však situace jiná. Většina událostí (interakce s GUI) a výpočtů je řízena takzvaným *Event Dispatch thread* (česky přeloženo jako *Vlákno událostí*). To je z důvodu, že většina Swing objektů není vláknově bezpečná a jejich spouštění v různých vláknech by vedlo k chybám či k neočekávaným chováním [9].

Hlavní vlákno programu, jež spouští program `main` tedy není příliš vytížené a jeho hlavním úkolem je naplánovat vytvoření a spuštění GUI právě na již zmíněném *Event Dispatch thread*.

Toho je docíleno použitím pomocné funkce `SwingUtilities.invokeLater`, které je předám objekt implementující rozhraní `Runnable`. Příklad takové třídy `Main` je naznačen v ukázce kódu 6.1.

```
1  public class Main {
2
3      private static void initNimbusLookAndFeel() {
4          // initialize look and feel for the app
5      }
6
7      public static void main(String[] args) {
8          initNimbusLookAndFeel();
9          SwingUtilities.invokeLater(MainWindow::run);
10     }
11 }
```

Kód 6.1: Příklad třídy Main pro spuštění programu využívajícího Swing GUI

6.3 Vytvoření uživatelského rozhraní

Jak již bylo zmíněno, pro vytvoření uživatelského rozhraní byla použita knihovna Swing. Ta ve svém základu nabízí již připravené základní ovládací prvky, ale nabízí také možnost definovat prvky vlastní.

Většina prvků byla s modifikacemi použita ze základní knihovny, některé však byly vytvořeny zcela nové. V následujících odstavcích je popsáno, jaké prvky byly použity pro docílení finálního vzhledu uživatelského rozhraní, jehož návrh byl popsán v sekci 5.5.

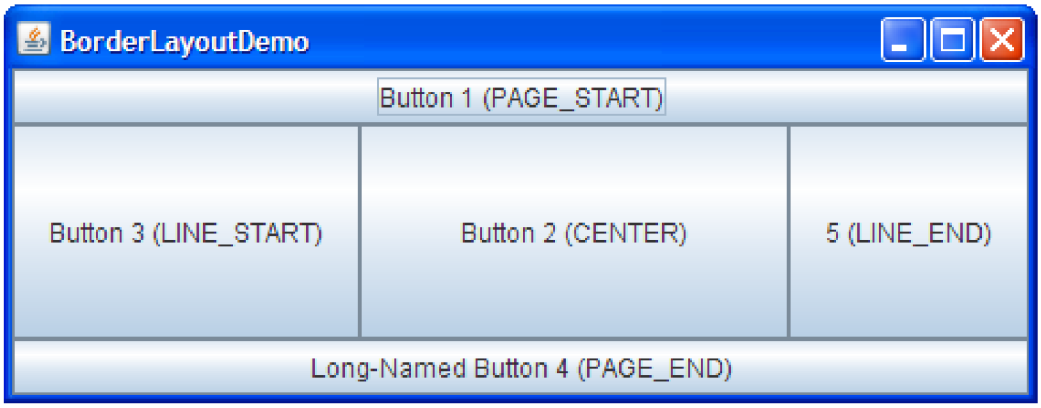
6.3.1 Základní grafické rozložení

Základní okno aplikace je tvořeno komponentou `JFrame`. Jedná se o základní *top-level container* (česky přeloženo jako *kontejner nejvyšší úrovně*), který slouží k vykreslení okna aplikace na obrazovku. Oknu specifikujeme jeho velikost a text v záhlaví (angl. *title*). Sdružuje v sobě všechny ostatní komponenty, a to jak ty obsahové uvnitř okna, tak například i kontextové menu.

Uvnitř okna se pak nacházejí ostatní obsahové komponenty obstarávající funkčnost uživatelského rozhraní. Není však příliš praktické tyto komponenty umísťovat do okna aplikace samostatně, ale pro jejich lepší organizaci a pozicování je výhodnější je sdružovat do logických celků. K tomuto účelu se nejčastěji používá komponenta `JPanel`. Jedná se o nejobyčejnější kontejnerovou třídu, která nabízí místo, ve kterém se nacházejí ostatní komponenty. Tato komponenta nemá žádný vzhled.

Pro vkládání komponent do `JPanelu` je často žádoucí tyto komponenty nějakým způsobem pozicovat tak, aby jejich rozložení odpovídalo našemu návrhu. K pozicování elementů uvnitř komponent se používají takzvané *LayourManagery*. Těch je v základní knihovně k dispozici několik[1], avšak je možné definovat i své vlastní. V případě naší aplikace byl pro všechny panely využit `BorderLayout`, který je mimo jiné zvolen i jako výchozí layout manger pro všechny top-level kontejnery. Ten umožňuje definovat pozici jednotlivých komponent do 5 oblastí – horní (`PAGE_START`) a dolní (`PAGE_END`) okraj panelu, levý okraj (`LINE_START`), pravý okraj (`LINE_END`) a střed panelu (`CENTER`) jak je vidět na obrázku 6.1. Všechny elementy jsou tímto rozložením *tlačeny* co nejvíce ke krajům a volné místo je umístěno vždy ke středu.

[1] Seznam všech layout managerů k dispozici zde https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html



Obrázek 6.1: Schéma rozložení komponent při použití BorderLayout. Převzato z [11].

O vytvoření a vykreslení obsahu hlavního okna se v kódu stará třída `OptionsPanel`. Ta pomocí metody `getPanel` vytváří jednotlivé panely a umísťuje je do hlavního okna. Jedná se o tři panely a jedno dialogové okno.

Panel pro výběr zpracovaných souborů obsahuje v horní části tlačítko (`JButton`) pro vyvolání dialogového okna pro výběru souborů a pod ním oblast, kde se vybrané soubory vypisují (`FileList` – tato komponenta je blíže popsána v sekci 6.3.3). Celý panel má nastavenou velikost, která odpovídá polovině velikosti hlavního okna aplikace a je umístěn k levému okraji.

Panel pro výběr zpracovaných souborů obsahuje stejně jako předchozí panel tlačítko a seznam vybraných souborů. Velikost panelu je také totožná s předchozím panelem a je umístěn k pravému okraji okna.

Panel pro tlačítko zpracování se nachází na spodním okraji okna a obsahuje tlačítko pro spuštění zpracování vybraných dokumentů. Před samotným zpracováním však zobrazí dialogové okno pro výběr cesty k výstupnímu souboru.

Dialogové okno zobrazující proces zpracování je při prvotním vykreslení skryto. Obsahuje text s popisem akce (`JLable`) a indikátor procesu (`JProgressBar`). Toto dialogové okno je zobrazeno při začátku zpracování souborů a hodnota v indikátoru procesu je aktualizována dle toho, kolik již souborů bylo zpracováno.

6.3.2 Výběr souborů

Výběr souborů pro zpracování je prováděn pomocí dialogového okna určeného pro výběr souborů (`JFileChooser`), které je zobrazeno po kliknutí na odpovídající tlačítko. Jedná se o vestavěnou komponentu Swing, ta však umožňuje velmi specifickou modifikaci jejího chování.

Můžeme nastavit například, zda je možné provést výběr více souborů (pomocí metody `setMultiSelectionEnabled`). Dále pak nastavujeme, chceme-li povolit výběr pouze souborů, pouze adresářů či obojího pomocí metody `setFileSelectionMode`.

Dalším důležitým nastavením může být nastavení, které soubory je možné pomocí dialogového okna vybrat. K tomu slouží metoda `setFileFilter`, která přijímá objekt, jež implementuje rozhraní `FileFilter`. Můžeme použít již existující filtry nebo si můžeme vytvořit vlastní, kde implementujeme metodu `accepts(File f)` pomocí které máme plnou kontrolu nad přijímanými soubory. V našem případě jsme použili již existující filtr, který filtruje dle přípon jednotlivých souboru – `FileNameExtensionFilter`.

Po zobrazení a potvrzení uživatelem získáme přístup k jeho vybraným souborům pomocí metody `getSelectedFiles`. Tato metoda vrací pole objektů typu `File` (což můžou být i adresáře).

Jelikož je však u výběru zpracovaných souborů povolený výběr celých adresářů, musíme nejprve data předzpracovat a z adresářů soubory získat. K tomuto účelu byla vytvořená metoda `FileUtils::getFilesWithDirectoryContent`, která přijímá právě pole zvolených souborů a nepovinně i filtr souborů. Metoda pak rekurzivně prochází všechny složky a přiřazuje do výstupního pole všechny soubory, které se ve složkách nacházejí a případně splňují dodaný filtr souborů.

Příklad vytvoření a specifikace chování komponenty pro výběr souborů je vidět v ukázce kódu 6.2.

```
1  private void handleSelectProcessFiles() {
2      JFileChooser fileChooser = new JFileChooser();
3      fileChooser.setDialogTitle("Choose files to process");
4
5      // Set file chooser properties to allow selecting directories and multiple files
6      fileChooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
7      fileChooser.setMultiSelectionEnabled(true);
8
9      // Add filter to only allow PDF files
10     FileNameExtensionFilter pdfFilter = new FileNameExtensionFilter("PDF", "pdf");
11     fileChooser.setFileFilter(pdfFileFilter);
12
13     int userSelection = fileChooser.showOpenDialog(this.mainPanel);
14
15     if (userSelection == JFileChooser.APPROVE_OPTION) {
16         // Recursively obtain all files from selected directories
17         processFiles = FileUtils.getFilesWithDirectoryContent(
18             fileChooser.getSelectedFiles(), pdfFileFilter
19         );
20         // Add files to GUI display list
21         processFileList.setItems(processFiles);
22     }
23 }
```

Kód 6.2: Příklad metody pro vytvoření a zobrazení dialogu pro volbu souborů

6.3.3 Zobrazení vybraných souborů

Zobrazení vybraných souborů je realizováno pomocí vlastní komponenty `FileList`. Tato komponenta je potomkem základní `Component` a stará se tak o celé vykreslení všech prvků.

Jejím základním stavebním kamenem je komponenta `JScrollPane`, která slouží k zobrazení většího počtu dat, než by se normálně vlezlo do jednoho okna. Vytváří tak okno, jež povoluje *scrollování* (česky vertikální či horizontální pohyb obsahu v okně) a je tak možné do tohoto okna vložit více obsahu.

Data, jež tato komponenta zobrazuje jsou dodána jako součást takzvaného list modelu (tj. třídy, implementující `AbstractListModel`). Jedná se o obálku, jež v sobě dokáže držet

seznam hodnot. Její výhodou oproti například obyčejnému poli je, že při změně těchto hodnot oznámí tuto událost komponentám výše, jež na tuto událost čekají. Toho pak využívá již zmíněný `JScrollPane`, aby při změně dat provedl překreslení svého obsahu.

Komponenta `FileList` pak nabízí jednoduché rozhraní, kde pomocí metod `setItems` a `addItem` přidáváme data, která má komponenta zobrazovat.

Zobrazení souborů pak ovlivňuje takzvaný *list cell renderer* (česky volně přeloženo jako *vykreslovač položek seznamu*), který vykresluje jednotlivé soubory. Zobrazuje tak jejich systémovou ikonu a název.

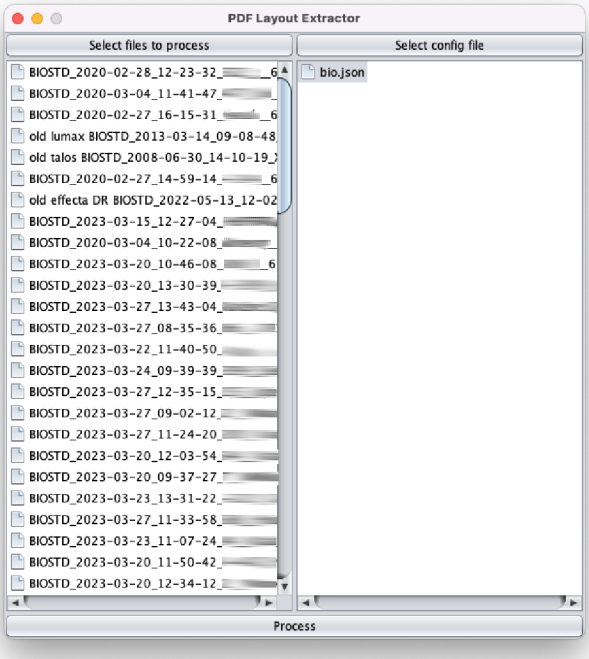
Jedná se o implementaci třídy, která rozšiřuje třídu `DefaultListCellRenderer` a přetěžuje její metodu `getListCellRendererComponent`, která ovlivňuje právě vhled, jakým budou jednotlivé buňky seznamu vykresleny.

6.3.4 Zobrazení průběhu zpracování

Průběžný průběh zpracování dokumentů je reprezentován za pomocí indikátoru postupu (angl. *progress bar*), pro který v knihovně Swing existuje komponenta `JProgressBar`. Tato komponenta má určitý rozsah hodnot a aktuální hodnotu, dle které pak zobrazuje výplň indikátoru.

Jelikož se zpracování souborů provádí v separátním vlákně (jak je popsáno v sekci 6.6), tak aktualizace aktuální hodnoty průběhu musí být realizována za pomocí *event listeneru* (volně česky přeloženo jako naslouchač událostí). Ten reaguje na změny v jiném vlákně a hodnotu aktualizuje.

Progress bar se nachází v dialogovém okně (`JDialog`), které je zobrazeno při začátku zpracování dokumentů a skryto při dokončení všech souborů.



Obrázek 6.2: Snímek obrazovky uživatelského rozhraní nástroje

6.4 Implementace `LayoutExtractoru` pro extrakci dat

Extrahovat data z dokumentů jde různým způsobem, proto při návrhu extraktoru bylo myšleno na rozšířitelnost a vznikla tak dvojice rozhraní (angl. *interface*) a abstraktní třídy.

Abstraktní třída nabízí metodu `extract`, která postupně projde celý strom a pro každý listový uzel zavolá metodu `extractFromArea`, která je předepsaná intefacem a každý extraktor si ji implementuje dle vlastní logiky. Extraktor při zpracování jednotlivých dokumentů drží v jednotlivých svých instancích data o právě zpracovávaném dokumentu a je tedy pro každý dokument třeba vytvořit instanci novou.

Pro extrakci dat na základě polohy a rozložení dokumentu tedy vznikla implementace tohoto rozhraní pojemenovaná jako `LayoutExtractor`.

6.4.1 Použité datové struktury

V průběhu extrakce dat je nutné si držet průběžný stav zpracovávaného dokumentu a pozic různých elementů. Za tímto účelem využívá algoritmus dvě datové struktury, jež jsou popsány v následujících odstavcích.

Datová struktura `ParseField`

Třída `ParseField` je javovská reprezentace objektů z JSON vstupní konfigurace popsané v sekci 5.3. Obsahuje tedy veškeré atributy jako vstupní konfigurace (to jsou `anchor`, `position`, `key`, `skip`, `lenght`). Navíc ještě obsahuje atribut `outputOrder`, jež drží informaci o tom, na jaké pozici bude extrahovaná informace obsažena ve výstupním souboru.

Třída navíc obsahuje ještě dvě pomocné metody (ukázka zdrojového kódu obou metod je vidět na kódu 6.3):

- `getParseFieldsFromJson` - Statická metoda, která převádí vstupní konfiguraci (tedy textový řetězec formátu JSON) na list objektů `ParseField`.
- `matchesAnchor` - Metoda pro porovnání, zda se text shoduje s hledaným textem kotvy. Tato metoda pouze neporovnává textové řetězce na jejich shodu, ale před porovnáním provádí jejich normalizaci, aby drobné odchylky (například v bílých znacích) nehráli při vyhledávání roli.

Datová struktura `SearchField`

Třída `SearchField` slouží k udržování informace o právě zpracovávaných datech v dokumentu. Plní funkci mezi-paměti, která je vytvořena v moment nalezení kotvy (z objektu `ParseField`) v dokumentu. Od této chvíle si drží dosavadní textovou hodnotu nalezených dat, počet zpracovaných oblastí a v případě hledaní ve sloupcích i souřadnici, ve kterém sloupci data hledat.

Třída poskytuje pak metodu `isComplete`, která vyhodnotí, je-li již nalezena kompletní informace dle předpisu. Instance je vyhodnocena jako dokončená, je-li součet počtu přeskočení a délky roven počtu zpracovaných oblastí.

Interní reprezentace počtu zpracovaných oblastí používá pro tuto informaci celé číslo. To je při vytvoření instance nastaveno na počet přeskočení (*skip*) pro jednotlivé pole. To znamená, že je-li toto číslo kladné, je žádoucí bloky pouze přeskakovat. Po snížení tohoto čísla pod nulu (tj. počet zpracovaných oblastí je záporný) se začínají oblasti zpracovávat a přidávat jejich obsah k výsledku. To se děje až do té doby, dokud je číslo větší než

požadovaná záporná hodnota délky výstupu. Tato reprezentace je zvolaná hlavně z důvodu, aby bylo snadné určit, zda se bloky mají přeskakovat, či zpracovávat.

```
public class ParseField {
    // ...

    // Method to normalize anchor text
    public String getAnchorNormalized() {
        return StringUtils.deleteWhitespace(this.getAnchor());
    }

    // Method to compare normalized anchor with given text
    public boolean matchesAnchor(String areaText) {
        return this.getAnchorNormalized().equals(StringUtils.deleteWhitespace(areaText)
    );
    }

    // Helper method to create list of ParseField from JSON string
    public static ArrayList<ParseField> getParseFieldsFromJson(String jsonConfigString)
     {
        // Parse the JSON string
        Type parseFieldsType = new TypeToken<ArrayList<ParseField>>(){}.getType();
        ArrayList<ParseField> parseFields = new Gson()
            .fromJson(jsonConfigString, parseFieldsType);

        // Assign the parse fields with outputOrder number
        int outputOrder = 0;
        for (ParseField parseField: parseFields) {
            parseField.setOutputOrder(outputOrder++);
        }

        return parseFields;
    }
}
```

Kód 6.3: Uklázka pomocných metod třídy ParseField

6.4.2 Popis extrakční funkce

Extrakce probíhá za pomocí extraktoru tak, že funkce `extractFromArea` je volána na každé oblasti stromu. Je důležité podotknout, že strom oblastí musí být před extrahováním seřazen, proto jej před samotným extrahováním abstraktní funkce `extract` seřadí a to dle pozice v dokumentu pomocí operátoru `SortByPositionOperator`.

Jelikož se jedná o extraktor založený na poloze a rozložení, jako hlavní metrika pro zpracovaní dat se používají jejich souřadnice v kartézské soustavě souřadnic. Ty získáme za pomocí funkce `Area::getBounds` a následném použití *getterů* na jednotlivé pozice na ose x a y.

Našim cílem je sloučit jednotlivé oblasti do celků logických oblastí – tedy oblastí, jež logicky patří k sobě. Proto pro každou oblast jsou z takto získaných dat o poloze spočítány rozdíly souřadnic oproti předchozí zpracovávané oblasti a tyto rozdíly jsou vyhodnoceny. Samotné vyhodnocení změn pozic je možné vidět ve zjednodušené ukázce kódu 6.4.

Nejprve se vyhodnocuje, zda se jedná o signifikantní změnu na ose y. Jako signifikantní změnu souřadnice y hodnotíme takový rozdíl, který je větší než $\frac{1}{4}$ výšky zpracovávané ob-

lasti. Pokud je taková změna nalezena, je vyhodnocena jako nový řádek a probíhá zpracování konce logické oblasti (popsáno v odstavci níže).

Dále se vyhodnocuje signifikantní změna na ose x. Jako signifikantní změnu souřadnice y hodnotíme takový rozdíl, který je větší než průměrná velikost písma v oblasti (v pixelech) – tedy mezera větší než jeden znak. Pokud je taková změna nalezena, je vyhodnocena jako nový logický blok a probíhá zpracování konce logické oblasti.

Pokud nebyla nalezena žádná signifikantní změna, je opět porovnána změna na pozici x. Je-li větší než $\frac{1}{3}$ průměrné velikosti písma, je vyhodnocena jako součást logické oblasti a její text je přiřetězen spolu se znakem mezery k aktuálnímu textu logické oblasti. Je-li změna menší, je text také přiřetězen, avšak bez mezery.

```
1  public void extractFromArea(Area a) {
2
3      // Skip areas not containing text
4      if (a.getText().isEmpty()) return;
5
6      var bounds = a.getBounds();
7      var em = a.getTextStyle().getFontSize();
8
9      if (lastY == -1) lastY = bounds.getY1();
10     var difY = bounds.getY1() - lastY;
11
12     // new LINE - significant difference on y
13     if (difY > 0.25f * bounds.getHeight()) {
14         this.handleEndOfLogicalArea(em, true);
15
16         areaStartX = bounds.getX1();
17         lastX = -1;
18         areaText = "";
19     }
20
21     if (lastX == -1) lastX = bounds.getX1();
22     var difX = bounds.getX1() - lastX;
23
24     // new LOGICAL AREA  - significant difference on x
25     if (difX > em) {
26         this.handleEndOfLogicalArea(em, false);
27
28         this.areaStartX = bounds.getX1();
29         this.areaText = a.getText();
30
31     // handle areas with small space between them
32     } else if (difX > 0.333f * em) {
33         this.areaText = this.areaText + " " + a.getText();
34
35     // handle areas right next to each other
36     } else {
37         if (this.areaStartX == -1) this.areaStartX = bounds.getX1();
38         this.areaText = this.areaText + a.getText();
39     }
40
41     lastX = bounds.getX2();
42     lastY = bounds.getY1();
43 }
```

Kód 6.4: Ukázka kódy metody pro detekci signifikatních změn polohy oblastí

Zpracování konce logické oblasti

Metoda `handleEndOfLogicalArea` pracuje již s předzpracovanými logickými oblastmi a ve velké míře k tomu využívá dočasné objekty třídy `SearchField`, jež vznikají a zanikají při průchodu souborem.

Pro každou logickou oblast nejprve projde všechny objekty `SearchField`. Podle směru, ve kterém jsou data hledaná se pak rozhoduje, zda je logická oblast hledanou oblastí či nikoli:

- **Jsou-li data hledaná v řádku,** zkontroluje, zda se dle počtu přeskočení a již zpracovaného počtu oblastí jedná o hledaná data. Pokud ano, přiřetězí data k výsledné hledané hodnotě. Následně, jedná-li se o konec řádku odstraní všechny nedokončená řádková hledání.
- **Jsou-li data hledaná ve sloupcích,** zkontroluje, zda se jedná o stejný sloupec. Pokud ano, zkontroluje zda se dle počtu přeskočení a již zpracovaného počtu oblastí jedná o hledaná data. Pokud ano, přiřetězí data k výsledné hledané hodnotě.

Následně je pro všechna vyhledávací pole zkontrolováno, zda jsou již dokončena (pomocí metody `SearchField::isComplete`) a pokud ano, uloží tato pole do seznamu dokončených a odstraní z aktuálního vyhledávání.

Jako poslední krok jsou zpracována všechna parsovací pravidla (`ParseField`) a pro všechny, jejichž kotva se shoduje s obsahem aktuálně zpracovávané logické oblasti (porovnání probíhá pomocí pomocné metody `ParseField::matchesAnchor`) se vytvoří nová vyhledávající pole (`SearchField`).

Zjednodušenou implementaci metody `handleEndOfLogicalArea` jde vidět na ukázce kódu 6.5.

```
protected void handleEndOfLogicalArea(float em, boolean endOfLine) {

    if (areaText.isEmpty()) return;

    Iterator<SearchField> i = searchFields.iterator();

    // Iterate over all possible search fields
    while (i.hasNext()) {
        SearchField searchField = i.next();

        // Handle search fields searching RIGHT
        if (searchField.getParseField().getPosition().equals("right")) {

            if (!searchField.shouldSkip()) {
                searchField.appendValue(areaText);
            }
            searchField.incrementProcessedAreaCount();
w
        // Handle search fields searching DOWN
        } else if (searchField.getParseField().getPosition().equals("down")) {

            // Check whether the column is same
            int columnDiff = areaStartX - searchField.getCoordinate();
            if (columnDiff >= 0 && columnDiff <= em) {
                searchField.incrementProcessedAreaCount();
                if (!searchField.shouldSkip()) searchField.appendValue(areaText);
            }
        }

        // Check if the search field is complete
         if (searchField.isComplete()) {
                doneSearchFields.putIfAbsent(
                    searchField.getParseField().getOutputOrder(),
                    searchField
                );
                i.remove();
        } else if (endOfLine && searchField.isSearchRight()) {
                i.remove();
        }
    }

    // Get all ParseFields, matching the anchor and current area text
    List<ParseField> currentParseFields = parseFields
            .stream()
            .filter(parseField -> parseField.matchesAnchor(areaText))
            .toList();

    // Create SearchFields for matching ParseFields
    for (ParseField currentParseField : currentParseFields) {
        if (currentParseField.getPosition().equals("down")) {
            this.searchFields.add(new SearchField(currentParseField, areaStartX));
         } else if (!endOfLine && currentParseField.getPosition().equals("right")) {
            this.searchFields.add(new SearchField(currentParseField));
        }
    }
}
```

Kód 6.5: Kód pro zpracování konce logické oblasti

6.5 Zpracování jednotlivých souborů

V předchozí kapitole jsme si popsali implementaci základního kamene získávání dat z dokumentu – `LayoutExtractor`. Avšak ten pro svoje správné fungování potřebuje pracovat nad stromem oblastí a ten doposud nemáme. Tato kapitola tedy popisuje kompletní zpracování jednoho dokumentu – od PDF souboru po výstup – za pomocí třídy `FileProcessor`.

Celé předzpracování dokumentu je řešeno za pomocí knihovny FitLayout. Na vstupu statické metody `LayoutExtractor::processFile` dostáváme soubor ke zpracování a seznam parsovacích polí (`ParseField`).

Prvním krokem je vykreslení všech stránek dokumentu do paměti – tím dostáváme objekt `Page`. Toho je docíleno pomocí renderovacího jádra knihovny FitLayout, konkrétně třídy `PDFBoxTreeProvider`.

Nyní potřebujeme na jednotlivé zpracované stránky použít segmentační algoritmus, který nám stránku převede na strom vizuálně oddělených oblastí. Problém ale je, že takovýto segmentační algoritmus pracuje pouze s již existujícím stromem oblastí. K tomuto účelu použijeme post-processor `VisualBoxTreeProvider`, který nám stránku převede do strumu oblastí (`AreaTree`), který následně zpracujeme již zmíněným segmentačním algoritmem `BasicSegmProvider`.

Tento strom segmentovaných oblastí již může být předán extraktoru, který z něj extrahuje data dle zadaných parsovacích pravidel. Tyto data jsou poté z extraktoru získány a předány na výstup funkce ve formě pole řetězců.

Ukázka celkového zpracování souboru je vidět v ukázce kódu 6.6.

```
1  public static ArrayList<String> processFile(
2      File processFile,
3      ArrayList<ParseField> parseFields
4  ) {
5
6      ArrayList<String> resultLine = new ArrayList<>();
7
8      var renderer = new PDFBoxTreeProvider(processFile.toURI().toURL(), true, false, 1.5
       f, 0, Integer.MAX_VALUE);
9
10     Page page = renderer.getPage();
11
12     var visualProvider = new VisualBoxTreeProvider();
13     Page visualPage = (Page) visualProvider.process(page);
14
15     var segmProvider = new BasicSegmProvider(true);
16     AreaTree areaTree = segmProvider.createAreaTree(visualPage);
17
18     var extractor = new LayoutExtractor(parseFields);
19     extractor.extract(areaTree);
20
21     var doneSearchFields = extractor.getDoneSearchFields();
22
23     // process doneSearchFields into results
24     // ...
25
26     return resultLine;
27 }
```

Kód 6.6: Zjednodušený kód demonstrující zpracování jednotlivých dokuemntů

6.6 Zpracování vstupu pomocí `SwingWorker`

`SwingWorker` je třída knihovny Swing, jež umožňuje vytvořit vlákno pro vykonávání dlouhotrvajících operací v grafickém rozhraní bez blokování hlavního vlákna, které se stará právě o vykreslování a obsluhování grafického rozhraní. Kdyby se tyto dlouhé operace vykonávali v hlavním vlákně, mělo by to za následek takzvané *zamrznutí* uživatelského rozhraní a možnou frustraci uživatele [10].

Zpracování souborů je kvůli své časové náročnosti tedy ideální kandidát na použití `SwingWorkeru`. Zpracování souborů tedy bude probíhat v třídě `ProcessFilesWorker`. Tato třída implementuje rozhraním předepsané funkce `doInBackground` a `done`.

Metoda `doInBackground` obsahuje samotný vykonávaný dlouhotrvající proces. Ten se skládá z následujících kroků:

1. Načtení obsahu konfigurační souboru a následná jeho konverze na javovské objekty (`ParseField`) pomocí funkce `ParseField::getParseFieldsFromJson`.

2. Vytvoření instance objektu pro zápis CSV souborů (`CSVWriter`) a následné vytvoření hlavičky tohoto souboru.

3. Zpracování jednotlivých souborů pomocí `FileProcesoru` (jak je popsáno v kapitole 6.5). Výsledky zpracování jsou zapsány do výstupního CSV souboru a je zvýšeno počítadlo hotových souborů.

4. Ukončení zápisu výstupního souboru a uložení souboru.

Po ukončení procesu je poté spuštěna metoda `done`, která skryje dialogové okno s ukazatelem aktuálního procesu a zobrazí hlášku o úspěchu.

Kapitola 7

Testování funkčnosti na dodaných datech

Jelikož zadavatel nedodal do data odevzdání práce specifikaci, které informace mají být z dokumentů extrahovány, probíhalo testování pomocí smyšlených úloh. To však neznamená, že by výsledky testování funkčnosti nebyly kompletní či vypovídající. Na všechny smyšlené úlohy bylo pohlíženo jako na úlohy dat, které by mohli být důležité a mít tak informační hodnotu.

7.1 Třídění dokumentů

Prvotně bylo testování prováděno na všech dokumentech jednoho výrobce najednou. To však ukázalo jedno velké úskalí – jednotlivé dokumenty se od sebe co do obsahu tak i do formátování velmi liší. Bylo tedy třeba dokumenty před samotným zpracováním nejprve roztřídit do skupin, které jsou si daty i formátem podobné a experimenty poté provádět na těchto podmnožinách.

Některé dokumenty byly tříděny dle typu reportu, jiné dle implantovaného zařízení a stáří dokumentu.

Třídění probíhalo ve velké částí manuálně, jelikož nebyl nalezen žádný efektivní způsob, jak tuto činnost zautomatizovat. Je však pravděpodobné, že byl-li by nástroj použit v reálu, jsou data nějak v systému tříděna a bylo by tedy snazší takto data rozdělit. Předpokládejme například, že jeden pacient má za celou dobu léčby implantovaný přístroj pouze jednoho výrobce a typu a bylo by tak prakticky možné data jednoho pacienta zpracovat bez předešlého třídění.

7.2 Tvorba testovacích scénářů

Jak již bylo zmíněno, pro testování bylo třeba si testovací scénáře vymyslet. A jelikož mi chybí znalost lékařských dat, přistupoval jsem k tomuto úkolu spíše laicky.

Scénáře byly tvořeny tak, že jsem se podíval na několik dokumentů ze skupiny, kterou jsem chtěl zpracovat, a snažil se hledat data, která se vyskytují ve všech (popřípadě většině) dokumentů. Takto jsem se snažil vybírat hlavně data, která mi přišla nějakým způsobem zajímavá a domníval jsem se o nich, že by mohli mít určitou informační hodnotu.

Pro takováto mnou vybraná data jsem pak vytvořil konfigurační soubory, za pomocí kterých bude program informace extrahovat.

7.3 Průběh testování

Jednotlivé konfigurační soubory spolu s relevantními daty byly postupně zpracovány programem a výsledky uloženy. V dalších kapitolách budou tyto výsledky prezentovány a komentovány. Jelikož však data mohou obsahovat citlivé osobní údaje, jsou některé informace ošetřeny, aby je nebylo možné přečíst.

Jednotlivé testovací scénáře jsou popsány zkratkou společnosti reportu a dále rozdělením souborů do kategorií nejčastěji dle typu zařízení a typu zpracovávaného reportu.

Zde v textu je pak vložen u každého scénáře výsledek extrakce. Vstupní data a konfigurační soubor jsou pak pod stejným jménem zařazeny v přiloženém adresáři.

7.4 Hodnocení úspěšnosti testování

Jednotlivé testovací scénáře jsem v rámci testování posuzoval na úspěšnost jejich extrakce. Toto posuzování bylo prováděno pouze manuálně a probíhalo porovnáním dat ze vstupních dokumentů s výstupem, který vytvořil implementovaný nástroj. Shodovali-li se výsledky a byly extrahovaný korektně, byl výstup považován za dobrý.

Jako možné problémy, na které byl při hodnocení testování brán zřetel jsou chybějící dat ve výstupu i přesto, že jsou obsaženy ve vstupu. Popřípadě špatné kódování výstupu nebo neúplné informace.

Všechny tyto neduhy byly řešeny v průběhu implementace. Povedlo se je však odladit a ve finálním testováním se již neprojevily.

Celkově tedy testování lze hodnotit jako úspěšné.

7.4.1 Testovací scénář – BIO | New | Rivacor

Tento testovací scénář se zaměřuje na extrakci základních dat o pacientovi a informací o baterii přístroje. Dokumenty tohoto typu jsou obsáhlé (řádově vyšší desítky stran) a proto samotná extrakce trvala spíše déle. Výstup extrakce je dobrý, data byla extrahovaná všechna a po manuální kontrole i korektně.

File name	Patient	Device	Follow up on	Tachycardia detection	Batrery voltage (V)	Remaining battery capacity (%)
BIOSTD_2020-03-04		Rivacor 7 DR-T	04/03/2020 09:44	Disabled	3.09	100
BIOSTD_2020-01-13		Rivacor 7 HF-T QP	13/01/2020 08:45	Disabled	3.09	100
BIOSTD_2020-03-30		Rivacor 7 VR-T	30/03/2020 10:14	Disabled	3.10	100
BIOSTD_2020-01-08		Rivacor 7 HF-T QP	08/01/2020 11:20	Disabled	3.09	100
BIOSTD_2020-02-10		Rivacor 7 HF-T QP	10/02/2020 09:25	Disabled	3.09	100
BIOSTD_2020-03-09		Rivacor 7 HF-T QP	09/03/2020 10:26	Disabled	3.09	100
BIOSTD_2020-02-19		Rivacor 7 VR-T DX	19/02/2020 12:48	Disabled	3.10	100
BIOSTD_2020-01-20		Rivacor 7 DR-T	20/01/2020 10:23	Disabled	3.08	100
BIOSTD_2020-03-02		Rivacor 7 HF-T QP	02/03/2020 11:51	Enabled	3.10	100
BIOSTD_2020-02-12		Rivacor 7 VR-T	12/02/2020 09:46	Disabled	3.08	100
BIOSTD_2020-01-20		Rivacor 7 HF-T QP	20/01/2020 09:29	Disabled	3.09	100
rivacor dr BIOSTD_2		Rivacor 7 VR-T DX	13/04/2022 12:00	Enabled	3.12	100
BIOSTD_2020-03-10		Rivacor 7 HF-T QP	10/03/2020 10:09	Disabled	3.10	100
BIOSTD_2020-03-05		Rivacor 7 VR-T	05/03/2020 10:33	Disabled	3.10	100
BIOSTD_2020-02-03		Rivacor 7 HF-T QP	03/02/2020 09:46	Disabled	3.08	100
BIOSTD_2020-03-17		Rivacor 7 HF-T QP	17/03/2020 10:09	Disabled	3.10	100
rivacor DR BIOSTD_		Rivacor 7 DR-T	01/04/2022 07:57	Enabled	3.11	100
BIOSTD_2020-02-24		Rivacor 7 HF-T QP	24/02/2020 10:07	Disabled	3.08	100

Obrázek 7.1: Extrahovaná data

7.4.2 Testovací scénář – BIO | New2 | Enitra

Tento testovací scénář se zaměřuje na extrakci dat o srdečním rytmu v noci a nastavení přístroje. Dokumenty jsou spíše obsáhlé, proto extrakce trvá déle. Některé dokumenty extrahované informace neobsahují, jelikož mají například danou funkcionalitu přistroje vypnutou. Tento testovací scénář tedy zobrazuje i takové případy. Výstup extrakce je dobrý a data byla extrahována korektně.

File name	Basic rate [bpm]	Night rate [bpm]	Night begins	Night ends	Pulse amplitude [V]
BIOSTD_2023-03-27_13	55	50	22:00	06:00	1.3
BIOSTD_2023-03-24_09	55	50	22:00	06:00	1.9
BIOSTD_2023-03-27_12	55	50	22:00	06:00	2.3
BIOSTD_2023-03-27_09	55	50	22:00	06:00	1.9
BIOSTD_2023-03-20_12	60	55	22:00	06:00	1.2
BIOSTD_2023-03-23_13	55	50	22:00	06:00	3.0
BIOSTD_2023-03-20_12	55	50	22:00	06:00	1.7
BIOSTD_2023-03-27_10	55	50	22:00	06:00	1.9
BIOSTD_2023-03-24_08	55	50	22:00	06:00	1.9
BIOSTD_2023-03-21_13	55	50	22:00	06:00	1.6
BIOSTD_2023-03-27_11	55	50	22:00	06:00	2.0
BIOSTD_2023-03-27_08	55	50	22:00	06:00	2.5
BIOSTD_2023-03-27_10	50	OFF			4.0
BIOSTD_2023-03-24_12	50	OFF			1.8
BIOSTD_2023-03-27_09	60	50	22:00	06:00	1.4
BIOSTD_2023-03-20_10	55	50	22:00	06:00	3.0
BIOSTD_2023-03-20_11	55	50	22:00	06:00	1.5

Obrázek 7.2: Extrahovaná data

7.4.3 Testovací scénář – BSCO | Quick Notes Report

Soubor dat typu *Quick Notes Report* obsahuje data z přístrojů více výrobců, avšak data jsou natolik podobná, že jdou zpracovávat současně. Pro extrakci v tomto testovacím scénáři byla vybraná základní data o pacientovi a nějaká lékařská data jako jsou *Přibližná doba do explantace* nebo *vnitřní amplituda*. Až na chybějící data v některých dokumentech se všechny informace vyextrahovali úspěšně a korektně. Výstup byl navíc díky obsáhlosti vstupních dokumentů (nižší jednotky stránek) vytvořen velice rychle.

File name	Device	Intrinsic Amplitude (Most recent)	Approximate time to explant	Date of Birth
QuickNotesReport-Sep-1	PROPONENT MRIELL231/938901	6.8mV	>11 years	N/RN/RN/R
QuickNotesReport-Sep-1	AUTOGENX4CRT-D G179/135761	1.3 mV	>5 years	N/R N/R N/R
QuickNotesReport-Feb-2	CHARISMA CRT-DG324/500899	N/RmV	> 8years	N/RN/RN/R
QuickNotesReport-Jul-1(	ALTRUA50 S508/576134	3.3 mV		
QuickNotesReport-Mar-1	VALITUDEX4 U128/705938	N/R mV	6.5 years	N/R N/R N/R
QuickNotesReport-Nov-2	INLIVEN W275/113732	3.1 mV	5.5 years	N/R N/R N/R
QuickNotesReport-Apr-2	ACCOLADEMRI EL L331/876849	8.7 mV	>11 years	N/R N/R N/R
QuickNotesReport-Aug-(	ACCOLADEMRI L311/269927	3.2 mV	6 years	N/R N/R N/R
QuickNotesReport-Jun-0	INCEPTA CRT-D P162/119874	N/R mV	5 years	7 Jan 1944
QuickNotesReport-Nov-1	CHARISMA ELICD D332/243151	5.6 mV@75 min [1]	>12 years	4 Apr 1951
QuickNotesReport-Jul-2:	AUTOGENICD D174/191431	15.2 mV@71 min	>8 years	N/R N/R N/R
QuickNotesReport-Apr-2	ACCOLADEMRI L311/269927	2.0 mV	9.5 years	N/R N/R N/R
QuickNotesReport-Dec-(	ALTRUA50 S504/573497	0.2 mV		
QuickNotesReport-May-(	AUTOGENX4CRT-D G179/136339	4.6 mV	7 years	30 Aug 1954
QuickNotesReport-Jul-0:	INVIVE W173/118030	N/R mV	>7 years	N/R N/R N/R
QuickNotesReport-Jan-0	VITALIO MRI J275/101893	20.5 mV@39 min	8.5 years	N/R N/R N/R
QuickNotesReport-Sep-1	VITALIO MRI J276/102818	6.8 mV	8.5 years	N/R N/R N/R
QuickNotesReport-Jun-0	VITALIO MRI J275/101893	24.9 mV@48 min	>8 years	N/R N/R N/R
QuickNotesReport-Feb-0	COGNIS100-D P107/009600			N/R N/R N/R
QuickNotesReport-Jun-2	AUTOGENX4CRT-D G179/136339	2.8 mV	8 years	30 Aug 1954
QuickNotesReport-Jan-1	VITALIO MRI J276/102818	2.4 mV	7 years	N/R N/R N/R
QuickNotesReport-Aug-2	ACCOLADEMRI L310/836133	14.1 mV@61 min	11 years	N/R N/R N/R
QuickNotesReport-Jan-2	ALTRUA50 S501/590330	7.4 mV		

Obrázek 7.3: Extrahovaná data

7.4.4 Testovací scénář – BSCO | Settings Report

Tento testovací scénář testuje extrakci dat, které popisují nastavení zařízení. Cílem tohoto scénáře je otestovat jak dokáže pracovat s textovými daty a znaky, které jsou více nestandardní jako jsou například horní indexy. I přes lehkou nesourodost dat a chybějící informace v dokumentech byla data extrahována správně a to včetně korektně extrahovaných speciálních znaků.

File name	Device	Last Programmed	Tachy EGM Storage	Detection Rate	Implant Date	Settings mode
SettingsReport-Mar-	ACCOLADEMRI L311/269927	27 Mar 2017		160 min ¹ (375 ms)	N/R	DDI
SettingsReport-Jan-	VITALIO MRI J276/102818	19 Sep2016		160 min ¹ (375 ms)	N/R	DDI
SettingsReport-Jun-	VITALIO MRI J275/101893	04 Jun2014	On	160 min ¹ (375 ms)	N/R	VVIR
SettingsReport-Sep-	VITALIO MRI J276/102818	19 Sep2016		160 min ¹ (375 ms)	N/R	DDI
SettingsReport-Jun-	AUTOGENX4CRT-D G179/136339	27 Jun2016			12Oct2015	DDI
SettingsReport-Jun-	INCEPTA CRT-D P162/119874	19 Jun2017			12Mar 2013	VVIR
SettingsReport-Oct-	INCEPTA CRT-D P162/119874	12 Oct2015			12Mar 2013	VVIR
SettingsReport-Sep-	INCEPTA CRT-D P162/119874	08 Sep2014			12Mar 2013	VVIR
SettingsReport-Jul-2	ACCOLADEMRIL310/ 839486	24Jul 2021	On	160min ¹(375ms)	N/R	VVIR
SettingsReport-Jun-	INCEPTA CRT-D P162/119874	02 May 2019			12Mar 2013	VVIR
SettingsReport-Dec-	ALTRUA50 S504/573497	25-MAY-2011			N.R. N.R. N.R.	VDD
SettingsReport-Jun-	INCEPTA CRT-D P162/119874	19 Jun2017			12Mar 2013	VVIR
SettingsReport-Jul-1	ALTRUA50 S508/576134	09-JUL-2013			N.R. N.R. N.R.	SSI
SettingsReport-Feb-	INCEPTA CRT-D P162/119874	13 Sep2021			12Mar 2013	VVIR
SettingsReport-Jan-	VITALIO MRI J275/101893	07 Jan2019	On	160 min ¹ (375 ms)	2Jun 2014	VVIR
SettingsReport-Jun-	VITALIO MRI J276/102818	02 Jun2014		160 min ¹ (375 ms)	N/R	DDI
SettingsReport-Oct-	VITALIO MRI J277/433364	02 Oct2015		160 min ¹ (375 ms)	N/R	DDI
SettingsReport-Jan-	VITALIO MRI J275/101893	11 Jan2016	On	160 min ¹ (375 ms)	N/R	VVIR
SettingsReport-Oct-	AUTOGENICD D174/101582	02 Oct2014			N/R	VVI
SettingsReport-Nov-	VITALIO MRI J276/104135	25 Nov 2016		160 min ¹ (375 ms)	N/R	DDI
SettingsReport-Sep-	INCEPTA CRT-D P162/119874	02 May 2019			12Mar 2013	VVIR
SettingsReport-Oct-	AUTOGENMINIICD D044/218955	08 Oct2018			5Oct 2018	VVI
SettingsReport-Jun-	INCEPTA CRT-D P162/119874	02 May 2019			12Mar 2013	VVIR

Obrázek 7.4: Extrahovaná data

Kapitola 8

Závěr

Cílem této diplomové práce bylo nastudovat způsob ukládání dat v dokumentovém formátu PDF a zmapovat existující nástroje, jež je možné požít pro extrakci dat a zpracování těchto dokumentů. Dále bylo třeba prostudovat zadání a vstupní dokumenty a na základě jich navrhnout řešení, které bude extrahovat data z lékařských zpráv na základě jejich rozložení. Práce dále popisuje samotný algoritmus pro extrakci dat, dále pak jeho implementaci a tvorbu uživatelského rozhraní. Praktickým výstupem práce je jednoduchý nástroj pro extrakci dat z dokumentů PDF dle definovaných pravidel založených na rozložení.

Nástroj je vyvinut v jazyce Java za použití knihovny FitLayout, která slouží pro práci s PDF dokumenty. Tato knihovna byla stručně popsána a na jejím základě vznikl návrh algoritmu, který na základě vstupních dat a zadané jednoduché konfigurace ve formátu JSON provede extrakci konfigurací definovaných dat do výstupního souboru. Extrakce probíhá na principu vyhledávání definovaných bodů – kotev – a následné extrakci dat na základě relativní pozice vůči těmto bodům.

Pro ovládání nástroje jsem navrhl a následně implementoval jednoduché uživatelské rozhraní, které umožňuje uživateli zvolit jednotlivé vstupy a sledovat proces zpracování jednotlivých dokumentů.

Na závěr byl nástroj otestován na reálných datech. Jelikož se však vstupní data velmi lišila, a jejich formát nebyl jednotný, bylo třeba před samotným testováním třeba provést jejich analýzu rozřazení do jednotlivých skupin. Následné pak bylo provedeno testování na několika testovacích scénářích, které ověřili funkčnost celého nástroje.

Na práci je možné navázat a pokračovat ve vývoji nástroje, bude-li ze strany zadavatele zájem. Jako možné budoucí práce se například nabízí možnost zpracování obrazových dat jako jsou grafy a různé grafické prvky nebo třeba rozšíření samotného jádra o strojové rozpoznávání a převod obrazu na text. Dále pak je možné implementovat nástroj na tvorbu a správu vstupních konfigurací.

Literatura

[1] Adobe. *Co je PDF?: Portable Document Format* [online]. 2021 [cit. 2023-01-1]. Dostupné z: https://www.adobe.com/cz/acrobat/about-adobe-pdf.html.

[2] Adobe Systems Inc.. *Adobe PDF Services API Documentation*. Dostupné z: https://developer.adobe.com/document-services/docs/overview/.

[3] Adobe Systems Inc.. *PDF Reference: Adobe® Portable Document Format*. 6. vyd. Adobe Systems Inc., 2006 [cit. 2023-01-10]. ISBN 0-321-30474-8. Dostupné z: https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/pdfreference1.7old.pdf.

[4] Burget, R. *FitLayout/2 Basic Concepts* [online]. 2022 [cit. 2023-01-12]. Dostupné z: https://github.com/FitLayout/FitLayout/wiki/Basic-Concepts.

[5] Das, R. *Core Java for Beginners, 3rd Edition*. Vika Publishing House, 2013. ISBN 9789325968509. Dostupné z: https://books.google.cz/books?id=BUZVAQAAQBAJ.

[6] International Organization for Standardization. *ISO 32000-1:2008: Document management — Portable document format — Part 1: PDF 1.7*. 1. vyd. 2008-07. Dostupné z: https://www.iso.org/standard/51502.html.

[7] Kumar, A. *Automated PDF Article Detection and Extraction* [online]. [cit. 2023-01-13]. Dostupné z: https://www.pdftron.com/blog/pdftron-ai/automated-pdf-article-detection-and-extraction/.

[8] Loy, M. a Eckstein, R. *Java Swing*. O'Reilly Media, Incorporated, 2002. Java Series. ISBN 9780596004088. Dostupné z: https://books.google.cz/books?id=W3HjIAduQfkC.

[9] Oracle. *Initial Threads: Concurrency in Swing*. Dostupné z: https://docs.oracle.com/javase/tutorial/uiswing/concurrency/initial.html.

[10] Oracle. *Simple Background Tasks: Worker Threads and SwingWorker*. Dostupné z: https://docs.oracle.com/javase/tutorial/uiswing/concurrency/simple.html.

[11] Oracle. *A Visual Guide to Layout Managers: Laying Out Components Within a Container*. Dostupné z: https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html.

[12] Siriwardena, P. *Mastering Apache Maven 3*. Packt Publishing, 2014. Community experience distilled. ISBN 9781783983872. Dostupné z: https://books.google.cz/books?id=_NAGBgAAQBAJ.

[13] Stack Exchange Inc.. *Stack Overflow Annual Developer Survey*. Dostupné z: https://insights.stackoverflow.com/survey.

[14] The Apache Software Foundation. *Introduction to the Standard Directory Layout*. Dostupné z: https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html.

[15] Wikipedia contributors. *Swing (Java) — Wikipedia, The Free Encyclopedia* [https://en.wikipedia.org/w/index.php?title=Swing_(Java)&oldid=1148203815]. 2023. [Online; accessed 14-May-2023].

[16] Williams, E. *How to Extract Data from PDF Quickly* [online]. 2022 [cit. 2023-01-13]. Dostupné z: https://pdf.wondershare.com/how-to/extract-data-from-pdf-form.html.

[17] Zukowski, J. *The Definitive Guide to Java Swing*. Apress, 2006. Definitive Guide. ISBN 9781430200338. Dostupné z: https://books.google.cz/books?id=YPjZNlEgAMcC.

Příloha A

Příklady vstupních dokumentů

Tato příloha obsahuje ukázky vstupních souborů. Některé informace jsou záměrně začerněny pro ochranu citlivých osobních údajů. U většiny vstupních dokumentů se jedná o první stranu souboru. Kompletní sada dokumentů je přiložena na paměťovém médiu.

Příloha obsahuje tyto příklady těchto dokumentů:

- **BIO** – Formát **New** – Typ zařízení **Enitra** – A.1
- **BIO** – Formát **New** – Typ zařízení **Rivacor** – A.2
- **BIO** – Formát **New v2** – Typ zařízení **Enticos** – A.3
- **BIO** – Formát **Old** – Typ zařízení **Evia** – A.4
- **BSCI** – Typ dokumentu **Combined follow-up report** – A.5
- **BSCI** – Typ dokumentu **Device settings report** – A.6
- **BSCI** – Typ dokumentu **Quick Notes Report** – A.7
- **MDT** – Typ dokumentu **Final: Session Summary** – Typ zařízení **Advisa** – A.8
- **MDT** – Typ dokumentu **Quick Look Report** – Typ zařízení **Micra** – A.9

Patient:	[redacted]	Follow-up on:	
Device:	Enitra 8 DR-T	18/02/2020	
S/N:	[redacted]	10:58	

Parameters - Bradycardia	(1st interrog.)
Mode	DDD
Basic rate/Night rate [bpm]	60/OFF
Basic rate [bpm]	60
Night rate [bpm]	OFF
Night begins	
Night ends	
Hysteresis [bpm]	OFF
Repetitive/Scan cycles	
Atrial overdrive	OFF
Magnet response	AUTO
Sensor/Rate fading [bpm]	120/OFF
Max. activity rate [bpm]	120
Sensor gain	AUTO
Sensor threshold	Medium
Rate fading	OFF
Rate increase [bpm/cycle]	2
Rate decrease [bpm/cycle]	0.5
Upper rate response [bpm]	130/WKB
Upper tracking rate [bpm]	130
Wenckebach response of [bpm]	130-188
Atrial upper rate [bpm]	240
Mode switching [bpm]	160/DDIR
Mode switching	ON
Intervention rate [bpm]	160
Switch to	DDIR
Onset criterion [out of 8]	5
Resolution criterion [out of 8]	5
Change of basic rate [bpm]	+10
Rate stabilization during mode switching	OFF
2:1 Lock-in protection	ON
Vp suppression	OFF

Obrázek A.1: BIO | New | Enitra

Patient:	[redacted]	Follow-up on:	
Device:	Rivacor 7 HF-T QP	13/01/2020	
S/N:	[redacted]	08:45	

Follow-up	(1st interrog.)
Tachycardia detection	Disabled

Patient

Name	
Last follow-up	XX/XX/XXXX
Implantation	XX/XX/XXXX

Device status

Mode/Ven. pacing	OFF/-----
Basic rate/UTR [bpm]	---- / ----

	A	RV	LV	2nd LV
Pulse amplitude [V]	----	----	----	----
Pulse width [ms]	----	----	----	----

VT1/VT2/VF [bpm]	OFF / OFF / OFF
Last charge time	----
Battery voltage [V]	3.09
Remaining battery capacity [%]	100
Battery status	BOS
Program number	0
Home Monitoring	OFF
MRI program	OFF

Test results	A	RV	LV	2nd LV
Sensing amplitude [mV]				
Mean rate [bpm]				
LV sensing polarity				
Pacing threshold [V]				
Pulse width [ms]				
Pacing impedance [Ω]				
Shock impedance [Ω]				
LV pacing polarity				
2nd LV pacing polarity				

Diagnostics

Pacing A/LV/BiV/CRT [%]	---- / ---- / ---- / ----
Atrial burden [%]	----

Episodes

New episodes VF/VT/others	---- / ---- / ----

BIOTRONIK excellence for life

Obrázek A.2: BIO | New | Rivacor

Parameters - Bradycardia	(1st interrog.)
Mode	DDDR
Basic rate/Night rate [bpm]	55/50
Basic rate [bpm]	55
Night rate [bpm]	50
Night begins	22:00
Night ends	06:00
Hysteresis [bpm]	OFF
Repetitive/Scan cycles	
Magnet response	AUTO
Sensor [bpm]	120
Max. activity rate [bpm]	120
Sensor gain	AUTO
Sensor threshold	Medium
Rate increase [bpm/cycle]	2
Rate decrease [bpm/cycle]	0.5
Upper rate response [bpm]	130/WKB
Upper tracking rate [bpm]	130
Wenckebach response of [bpm]	130-169
Atrial upper rate [bpm]	240
Mode switching [bpm]	160/DDIR
Mode switching	ON
Intervention rate [bpm]	160
Switch to	DDIR
Onset criterion [out of 8]	5
Resolution criterion [out of 8]	5
Change of basic rate [bpm]	+10
Rate stabilization during mode switching	OFF
2:1 Lock-in protection	ON

Obrázek A.3: BIO | New v2 | Enticos

BIOTRONIK

Patient: [redacted]
Device: **Evia DR-T**
S/N: [redacted]

PSW 2100.A
(PSW 1901.A/1)
RAM ID: 3.0

Parameters (perm.)

Date: **27/02/2020**
Time: **14:59**

Patient

ID	0
Last name	slezak
First name	
Date of birth	XX/XX/XXXX
Gender	XXX
Date of implant	23/06/2010
Hospital, City	-----
Physician	-----
LVEF [%]	20
NYHA	III
Symptom	XXX
ECG indication	XXX
Etiology	XXX

Leads

	A	**V**
Polarity	BIPL	BIPL
Type		
Manufacturer	XXX	XXX
Lead position	Lateral wall	RVOT

Status

Battery status	OK
Remaining battery capacity [%]	50
Calculated ERI	6 Y. 5 Mo.
Magnet response	AUTO
Last follow-up	06/11/2018
RAM ID	3.0
HM PID	90

Bradycardia

	Previous	**Current**
Mode		AAI
Basic rate/Night rate [bpm]		30/OFF
Night begins		-----
Night ends		-----
Hysteresis [bpm]		OFF
Repetitive cycles		-----
Scan cycles		-----
Sensor/Rate fading [bpm]		/OFF
Sensor gain		-----
Automatic gain		-----
Sensor threshold		-----
Rate fading		OFF
Rate increase [bpm/cycle]		4
Rate decrease [bpm/cycle]		0.5
Upper rate response [bpm]		-----
Mode switching [bpm]		-----
Vp suppression		OFF
AV delay [ms]		-----
Atrial overdrive		OFF

Atrium

Pulse amplitude [V]	0.2
Pulse width [ms]	0.1
Capture control	-----
Sensitivity [mV]	2.0
Pacing polarity	UNIP
Sensing polarity	UNIP

Obrázek A.4: BIO | Old | Evia

Boston Scientific

LATITUDE™ Programming System | **Report Created 23 Feb 2022**

Combined Follow-up Report

[redacted]

Date of Birth	N/R N/R N/R	Last Office Interrogation
Device	CHARISMA CRT-D G324/ 500899	**22 Feb 2022**
Tachy Mode	Monitor + Therapy	Implant Date
		22 Feb 2022

My Alerts

There are no alerts to display.

Events Since Last Reset (22 Feb 2022)

23 Feb 2022 07:27	SAM , Respiratory Sensor Vector Switched
22 Feb 2022 12:30	LV Auto
22 Feb 2022 12:28	RV Auto

Battery OK

Approximate time to explant:	> 8 years
Charge Time	N/R s
Last Capacitor Re-form	N/R

Leads Data	Implant 22 Feb 2022	Previous Session	Most Recent
Atrial			
Intrinsic Amplitude	N/R mV	0.5 mV	N/R mV
Pace Impedance	N/R Ω	>3000 Ω	>3000 Ω
Pace Threshold	N/R V @ N/R ms	N/R	Test Failed
Right Ventricular			
Intrinsic Amplitude	N/R mV	Paced mV	>25.0 mV @ 48 min^{-1}
Pace Impedance	N/R Ω	634 Ω	669 Ω
Pace Threshold	N/R V @ N/R ms	Auto 0.8 V @ 0.4 ms	Auto 0.8 V @ 0.4 ms
Left Ventricular			
Intrinsic Amplitude	N/R mV	Paced mV	4.9 mV
Pace Impedance	N/R Ω	654 Ω	706 Ω
Pace Threshold	N/R V @ N/R ms	Auto 2.5 V @ 1.0 ms	Auto 2.2 V @ 1.0 ms
Shock Vector			
Shock Impedance	N/R Ω	58 Ω	58 Ω

Settings

Ventricular Tachy Settings

VF	200 min^{-1}	ATP		41J, 41J, 41Jx6
VT	165 min^{-1}	Burst	Ramp	41J, 41J, 41Jx4

Brady/CRT Settings

Mode	VVIR	Pacing Output	
Lower Rate Limit	75 min^{-1}	Right Ventricular	Auto 2.0 V @ 0.4ms
Maximum Sensor Rate	130 min^{-1}	Left Ventricular	Auto 3.2 V @ 1.0ms
RV-Refractory (RVRP)	230 - 250 ms	Sensitivity	
LV-Refractory (LVRP)	250 ms	Atrial	AGC 0.25 mV
Ventricular Pacing Chamber	BiV	Right Ventricular	AGC 0.6 mV
LV Offset	-20 ms	Left Ventricular	AGC 1.0 mV
		Leads Configuration (Pace/Sense)	
		Atrial	Bipolar
		Right Ventricular	Bipolar
		Left Ventricular	LVtip>>Can / LVtip>>LVring
		Rate Adaptive Pacing	
		Accelerometer	On

3868 Software Version: 1.10
G324 Firmware Version: H_v1.00.00(1.01)

Clinician Signature:

Obrázek A.5: BIO | Combined follow-up report

Boston Scientific	ZOOM ® View™	Report Created 06 Jun 2016
	Device Settings Report	
	[redacted]	Last Office Interrogation
	Date of Birth [redacted]	**12 Oct 2015**
	Device INCEPTA CRT-D P162/119874	Implant Date
	Tachy Mode Monitor + Therapy	**12 Mar 2013**

Programming	
Last Programmed	12 Oct 2015
Ventricular Tachy Mode	12 Mar 2013 Changed to Monitor + Therapy
	12 Mar 2013 Changed to Off

Ventricular Tachy

VF 230 min⁻¹ (261 ms)

Detection/Redetection	
Initial Duration	1.0 s
Redetection Dur	1.0 s
Post-shock Dur	1.0 s

VT 205 min⁻¹ (293 ms)

Detection/Redetection	
Initial Duration	7.0 s
Redetection Dur	1.0 s
Post-shock Dur	1.0 s
Enhancements	Onset/Stability
VT Detection	On
Polymorphic VT Discrimination	
Initial Detection	
Shock if Unstable	30 ms

Therapy	
QUICK CONVERT™ ATP	On
Shock 1	31 J
Shock 2	41 J
Additional 41 J Shocks	6
ATP1	Scan
Number of Bursts	2
Pulses per Burst	
Initial	8
Increment	2
Maximum	10
Coupling Interval	88 %
Decrement	10 ms
Burst Cycle Length	88 %
Ramp Decrement	0 ms
Scan Decrement	10 ms
Minimum Interval	220 ms
ATP2	Ramp
Number of Bursts	2
Pulses per Burst	
Initial	8
Increment	2
Maximum	10
Coupling Interval	84 %
Decrement	0 ms
Burst Cycle Length	84 %
Ramp Decrement	10 ms
Scan Decrement	0 ms
Minimum Interval	220 ms
ATP Time-out	Off mm:ss
Shocks	
Shock 1	31 J
Shock 2	41 J
Shock 3 -6	41 J

2868 Software Version: 3.07
P162 Firmware Version: B_v1.02.00(4.01)

Clinician Signature:

Obrázek A.6: BIO | Device settings report

Boston Scientific	ZOOM ® View™	Report Created 22 Apr 2020
	QUICK NOTES ® Report	
	Date of Birth N/R N/R N/R Device ACCOLADE MRI EL L331/876849	Last Office Interrogation **N/R** Implant Date **N/R**

My Alerts

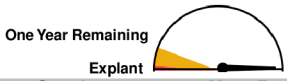
There are no alerts to display.

Events Since Last Reset (20 Apr 2020)

21 Apr 2020 15:10 RV Auto
21 Apr 2020 14:10 RA Auto

Battery OK

Approximate time to explant: > 11 years
Magnet Rate 100 min^{-1}



Leads Data	Implant N/R N/R N/R	Previous Session	Most Recent
Atrial			
Intrinsic Amplitude	N/R mV	N/R mV	8.7 mV
Pace Impedance	N/R Ω	N/R Ω	708 Ω
Pace Threshold	N/R V @ N/R ms	N/R	Auto 0.4 V @ 0.4 ms
Ventricular			
Intrinsic Amplitude	N/R mV	N/R mV	19.6 mV @ 155 min^{-1}
Pace Impedance	N/R Ω	N/R Ω	890 Ω
Pace Threshold	N/R V @ N/R ms	N/R	Auto 0.6 V @ 0.4 ms

POST completed. Test started at 20 Apr 2020 09:55

Brady Counters Since Last Reset (20 Apr 2020)

Counters

Atrial 4% Paced
Ventricular 30% Paced

Atrial Arrhythmia

% AT/AF 0

Settings

Ventricular Tachy Settings

Ventricular Tachy EGM Storage On
Detection Rate 160 min^{-1}

Brady Settings

Mode DDD
RYTHMIQ™ Off
Lower Rate Limit 55 min^{-1}
Maximum Tracking Rate 130 min^{-1}
Paced AV Delay 80 - 220 ms
Sensed AV Delay 75 - 200 ms
A-Refractory (PVARP) 240 - 280 ms
V-Refractory (VRP) 230 - 250 ms

Atrial Tachy Settings

ATR Mode Switch 170 min^{-1} DDI

Pacing Output
Atrial Trend 3.5 V @ 0.4 ms
Ventricular Trend 3.5 V @ 0.4 ms
Sensitivity
Atrial Fixed 0.75 mV
Ventricular Fixed 2.5 mV
Leads Configuration (Pace/Sense)
Atrial Bipolar
Ventricular Bipolar

2869 Software Version: 2.06
L331 Firmware Version: F_v1.00.00(1.10)

Clinician Signature:

Obrázek A.7: BIO | Quick Notes Report

Final: Session Summary

Device: **Advisa DR MRI A3DR01** Serial Number: [redacted] Date of Visit: **08-Mar-2023** [redacted]

Patient: [redacted] ID: Physician:

Device Information

Device	Medtronic	Advisa DR MRI A3DR01	[redacted]	Implanted: 20-Jan-2023

Device Status (Implanted: 20-Jan-2023)

Battery Voltage (RRT=2.83V)	3.06 V	(08-Mar-2023)
Remaining Longevity (based on initial interrogation)	10 years (9 - 11.5 years)	

	Atrial	**RV**
Lead Impedance	475 ohms	722 ohms
Capture Threshold	0.500 V @ 0.40 ms	0.250 V @ 0.40 ms
Measured On	08-Mar-2023	08-Mar-2023
In-Office Threshold	0.50 V @ 0.40 ms	0.50 V @ 0.40 ms
Programmed Amplitude/Pulse Width	3.50 V / 0.40 ms	3.50 V / 0.40 ms
Measured P/ R Wave	1.6 mV	9.1 mV
In-Office P/R Wave	1.6 mV	8.5 mV
Programmed Sensitivity	0.30 mV	0.90 mV

Parameter Summary

Mode	AAIR<=>DDDR	Lower Rate	55 bpm	Paced AV	180 ms
Mode Switch	171 bpm	Upper Track	130 bpm	Sensed AV	150 ms
		Upper Sensor	130 bpm		

Detection		**Rates**	**Therapies**
AT/AF	Monitor	>171 bpm	All Rx Off
VT	Monitor	>150 bpm	

Changes This Session **Session Start** **Current Value**

No parameters have been changed during the current session.

Obrázek A.8: MDT | Final: Session Summary | Advisa

Quick Look II Report

Device: **Micra VR TCP MC1VR01** Serial Number: [redacted] Date of Visit: **02-May-2022** [redacted]

Patient: [redacted] ID: [redacted] Physician: **Dr. Milan Sepsi** [redacted]

Device Status (Implanted: 22-Jun-2021)

Battery Voltage (RRT=2.56V)	3.07 V	(02-May-2022)
Remaining Longevity	>8.0 years	(>7.0 - >9.0 years)
	RV	
Electrode Impedance	530 ohms	
Capture Threshold	0.50 V @ 0.24 ms	
Measured On	02-May-2022	
Programmed Amplitude/Pulse Width	1.38 V / 0.24 ms	
Measured R-Wave	10.2 mV	
Programmed Sensitivity	2.00 mV	

Pacing (% of Time Since 02-Aug-2021)

VS	55.1 %
VP	44.9 %

Parameter Summary

Mode	VVIR	Lower Rate	60 bpm
		Upper Sensor	120 bpm

OBSERVATIONS (1)

· RV Capture Management determined that threshold increased on 06-Apr-2022. This increase was greater than Amplitude Safety Margin and may have compromised capture.

Obrázek A.9: MDT | Quick Look Report | Micra