Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

GATE plugin pro rozdělování textu na věty

Plzeň, 2012 Pavel Slavíček

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 10. května 2012

Pavel Slavíček

Poděkování

Na tomto místě bych především rád poděkoval vedoucímu této bakalářské práce Ing. Michalovi Konkolovi za jeho ochotu, rady a připomínky při tvorbě této práce. Mimo vedoucímu této práce bych dále chtěl poděkovat mé rodině a přátelům za jejich podporu během studia.

Abstract

This work is concentrated on problems of text segmentation in Czech language. It begins with a brief introduction to the point at issue of segmentation and with the summary of possible conditions, which they could come on within its implementation. Further, the work contains the description of GATE system whose part of are the preface, the characterization of fundamental attributes, the procedure of creation plug-in incl. its integration and the potentiality of GATE Developer in terms of segmentation extent. These follows with the characterization of qualities and the content of created plug-in for GATE Developer that is carrying out text segmentation. Consequently my work is getting on with description of implementation plug-in in program Java. The latest section of this work is attended to the data evaluation and the summing up results of segmentation performed by created plug-in.

Obsah

1 Úvod

Tato práce se zabývá problematikou segmentace textu, přesněji segmentace textu v českém jazyce. Segmentace je způsob zpracování textu, který vykoná proces, při kterém text rozdělí do vět. V této problematice jsou zkoumány pravidla a aspekty podle, kterých je prováděna segmentace. Segmentace úzce souvisí s problematikou tokenizace, kde tokenizací se rozumí především rozdělení textu do slov a v některých literaturách jsou označovány jednotně. Většina aplikací jako jsou POS taggers, parsers, stemmers, vyhledávače atd. zpracovávají text na základě slov a vět. Je to tedy proces, jenž dále využívají tzv. „inteligentní“ aplikace a nepřesnosti vzniklé provedením segmentace mají nepříznivé následky v pozdější fázi zpracování tj. proces segmentace je předzpracováním textu pro inteligentní aplikace .

Cílem této práce je implementace algoritmů v programovém jazyce Java. Algoritmy budou vyvinuty na základě získaných poznatků o problematice segmentace. Navržené algoritmy budou vytvořeny tak, aby zohledňovaly danou problematiku a bude se experimentovat s variacemi pravidel a kombinacemi příznaků. Dále navržené algoritmy se budou implementovat pro systém GATE Developer ve formě pluginu. Součástí tohoto pluginu má být několik tzv. zdrojů, které obsahují algoritmy pro segmentaci.

V úvodní části práce jsou popsány základní problémy, které vyvstanou při segmentaci českého jazyka. Potom navazuje popis systému GATE, postup vytvoření pluginu a dále popis implementace navrženého pluginu.

2 Definice problému segmentace a tokenizace

Přirozený jazyk není možné kompletně popsat, jelikož se vždy najde výjimka. U segmentace nastává problém při určování hranic věty, neboť nejsou vždy jednoznačně určeny. Interpunkce patří mezi nejednoznačné znaky, protože pokaždé neukončují větu. Interpunkce může být součástí předcházející posloupnosti znaků např. tečka může být ukazatelem konce věty anebo součástí předcházející posloupnosti tzn. nezakončuje větu.

Dalším aspektem jsou slova začínající velkým písmenem, mezi tato slova zpravidla patří vlastní jména, názvy společností atd. Ovšem slova nemusejí začínat na velké písmeno jen v případech uvedené v předchozí větě, ale počáteční písmeno je ovlivňováno také pozicí ve větě např. pozicí v textu, jenž si také vyžadují velké písmeno a těmi jsou např.: titulek, položky v tabulce, v seznamech a další [4].

Existují tedy nejednoznačné situace, u kterých je potřeba rozhodnout o jejich významu. Nicméně pokud jsou použita jednoduchá pravidla je možné docílit celkem dobrého výsledku rozdělení do vět. Například odborný článek [6] uvádí 93,2% úspěšnost rozeznaných vět za předpokladu, že každá tečka je považována za konec věty.

S pojmem segmentace úzce souvisí i tokenizace. Tokenizace textu je proces, kdy jsou vyhledávaná v největší míře jednotlivá slova a pak celky, které mohou být tvořena více slovy najednou a výsledkem tokenizace jsou pak primitiva nazývané tokeny. Jednotlivá slova se rozeznávají mezerami, což jsou velmi dobrými ukazateli hranic tokenů a patří do rodiny tzv. bílých mezer, které také mohou posloužit ke stejnému účelu jako mezery. Bílé znaky mohou poukazovat na uspořádání textu kolem nich a vztah mezi slov. V konečném důsledku to znamená, že se slovem odděleném mezerou může být naloženo jinak, než kdyby bylo oddělené tabulátorem. Celky slov, které mají být tokeny se mohou dělit do několika typů jako například do celku tvořící jméno z více něž jednoho slova anebo dříve zmíněné názvy společností atd. Rozeznání tokenů, které jsou tvořeny více jak jedním slovem je tedy náročnější, protože si vyžaduje analýzu okolních slov nebo dokonce celé věty [5].

Pojem korpus

Korpus je sbírka linguistických dat s velkým množství přirozeného textu. Používají se pro testování a trénování algoritmů. Pro přirovnání první moderní elektronický korpus (1961 W.N.Francis a H.Kučera) Brown Corpus of Standard American English má cca milión slov, z toho 500 textů s průměrnou délkou kolem 2 300 znaků. Obsahuje různé typy textů s odvětví odborných textů, novinových článků atd.

Existují také anotované korpusy, které obsahují přidané linguistické nebo jiné informace o obsahu.

2.1 Interpunkční znaky

Znak vykřičníku „!“ nebo otazníku „?“ ve většině případech spolehlivě ukončuje větu. Existuje několik výjimek, kdy vykřičník a otazník může být součástí názvů společností, jako je „Yahoo!“.

Středník je znak, který odděluje části jedné věty a je používán také pro výčet prvků, které bývají jinak odděleny čárkou seznam.

Dvojtečka uvozuje přímou řeč (citace), uvozuje také výčet, pak se používá mezi předtiskem a daným údajem, pro časové údaje, poznamenání sportovního skóre, pro zápis poměru a pro vysvětlení nebo odůvodnění (ve smyslu totiž, neboť...). Zpravidla se píše ihned za slovem bez mezery a za ní je mezera, ale v některých situacích je oddělena z obou stran anebo vůbec. V matematických výrazech dělení (obecně, když vyjadřuje poměr) je oddělena mezerou z obou stran. Při zápisu časových údajů jsou mezery vynechány a stejně tak i u skóre. Po dvojtečce je možné napsat malé i velké písmeno a to v případě, kdy po ní následuje větný celek, pokud následuje nevětný je pravidlem malé písmeno. Ovšem při výpisu výčtů lze napsat velké i malé písmeno za dvojtečkou. Jednotlivé body ve výčtu se oddělují čárkami, tečkami (každý bod začíná velkým písmenem) nebo je možné tečku vynechat. Pokud jsou jednotlivé body dostatečně graficky odděleny (každý bod začíná velkým písmenem). Podle normy ČSN 01 0197 Bibliografická citace je dvojtečka oddělena mezerami mezi místem vydání a nakladatelstvím [2].
Příklady použití dvojtečky:

- uvození přímé řeči: Petr řekl: „Pojd’me ven.“,
- předtisk: Jméno a příjmení: Karel Čapek,
- poměr: Obdélník má strany v poměru je 2 : 3.,
- časový údaj: Oběd je ve 12:00.,
- výčet: Dnešní povinnosti: schůzka, trénink, nakoupit.,
- dělení: $4 : 2 = 1$,
- skóre: Jejich skóre bylo 3:1.,
- signalizace vysvětlení nebo odůvodnění: Dnes to je už jasné: Bude/bude nádherný den.

Tečka většinou ukončuje větu, ale někdy je součástí zkratky, pořadového čísla, data (je možné za tečkou napsat mezeru podle určitých norem) nebo časového údaje (tečka odděluje hodiny a minuty s absencí mezer). Problémem je tedy správné rozeznání konce věty, když může tečka být jak koncem věty, tak i součástí předchozí sekvence znaků. V horším případě má tečka dvojí význam a to takový, že je koncem věty a zároveň je součástí zkratky či pořadového čísla [6].

2.2 Zkratky

Se zkratkami souvisí problém určení, jestli je na konci věty nebo nikoliv. Pokud je za zkratkou (poslední tečkou zkratky) mezera a velké písmeno je možné, že tečka zakončuje větu a velké písmeno začíná větu novou. Tento případ není, ale pravidlem, protože velké písmeno může být součástí jména, názvu firmy apod. a neuvozovat novou větu. Pro takové situace a jim podobné je nutná analýza celé věty nebo alespoň její části.

Pro zlepšení rozeznání zkratek by se mohl udělat jejich seznam. Nicméně úplný seznam není jednoduché sestavit, protože je jich velké množství a stálé vznikají např. zkratky pro instituce apod. Dále skutečnost zhoršuje fakt, že některé zkratky mohou být i slovy např. „obr“ může být zkratka „obrázek“.

2.3 Čísla

Čísla jsou v češtině psána ve formátu tak, že desetinným oddělovačem je čárka a oddělovačem tisíců mezera. Nejednoznačné znaky v čísle je mezera a čárka. Čárka může oddělovat věty a nebo je desetinným oddělovačem. Další nejednoznačný znak je mezera, která může být oddělovačem tisíců. Mohou se objevit i taková čísla, která nemají oddělovač tisíců.

U pořadových čísel je podobná situace jako u zkratek. Pokud po číslu následuje tečka přicházejí v úvahu možnosti, že to je pořadové číslo, číslovka nebo pořadové číslo na konci věty.

3 GATE

Zdroje potřebné k napsání této kapitoly jsem čerpal s dokumentace „GATE User Guide“ dostupné na webových stránkách GATE NLP [1] tj. `http://gate.ac.uk/`. Pro podrobnější informace o GATE odkazuji na onu dokumentaci.

Tato kapitola zahrnuje popis některých frekventovaných termínů, úvod do základních možností GATE, popis vývoje tzv. zdrojů a jejich následnou integraci do GATE Developer, též manipulaci s anotacemi v GATE Developer a nakonec popis pluginu, jež vznikl na základě této práce.

3.1 Úvod

Označení GATE je zkratka „General Architecture for Text Engineering“. GATE je možné použít v mnoha oblastech analýzy textu, ale v této práci bude uvedeno nejčastější použití a tím je proces anotace textu v sadě dokumentů. Pro identifikování předem definovaných částí tj. anotace textu GATE poskytuje vhodné rozhraní. Dokumenty určené k anotaci se mohou seskupovat do korpusů. Anotace dokumentu se může provádět automaticky použitím z některých dostupných pluginů, jako je například Gazetteers. Nebo lze anotovat text ručně v GATE Developer.

GATE je architektura, pomocí které lze zakomponovat a kombinovat Natural Language Processing (NLP)[1] software. Například: oddělovač vět (sentece splitters), určování pojmenovaných entit (Named Entity recognizers) atd.

Prvky softwarových systémů lze rozdělit do několika komponent známé jako zdroje. GATE komponenty jsou specializované typy JavaBeans rozděleny do tří druhů zdrojů a jsou to: Language Resources (LRs, LR); Processing Resources (PRs, PR) a Visual Resources (VRs, VR).

- **Language Resources**: jsou entity, jejichž obsahem jsou linguistické data. Data mohou poskytovat textové dokumenty, korpusy, lexikony a jiné.
- **Processing Resources**: jsou entity zpracovávající linguistické data. Tyto zdroje realizují různé algoritmy pro zpracování linguistických dat. Mohou anotovat text podle určitých pravidel například rozezná věty, slova atd.
- **Visual Resources**: jsou komponenty, které vytvářejí grafické rozhraní (V této práci nejsou zahrnuty).

[1] Natural Language Processing ve volném překladu znamená „Počítačové zpracování přirozeného jazyka“ a zabývá se, jak napovídá samotný název, problematikou počítačového zpracování lidské řeči

3.2 Instalace softwaru

Verze GATE 6.1 použitá při vývoji pluginu je v současné době dostupná především pro operační systémy Windows, Linux, Mac OS X a Solaris. Na ostatních platformách není zaručena kompatibilita. Pro správnou funkčnost GATE 6.1 je zapotřebí mít na stroji nainstalovanou Java 6.0 update 10 nebo vyšší. Stažení příslušného souboru pro instalaci podle OS lze na adrese:

`http://gate.ac.uk/download/index.html`

Jedna z možností je stáhnout binární balík, který po rozbalení lze spustit souborem `gate.exe` ve Windows nebo v Linuxu spuštěním `gate.sh` nacházející se ve složce `bin`. Nebo také příkazem `ant run` v kořenovém adresáři GATE Developer, když je na daném stroji dostupný nástroj odpovídající verze Apache Ant[2] k dané verzi GATE Developer.

3.2.1 Dokumentace

Součástí instalace je také dokumentace, která zahrnuje všechny důležité informace. Umístění dokumentace je v kořenovém adresáři pod názvem „tao.pdf“. Samozřejmě je také dostupná online na adrese `http://gate.ac.uk/sale/tao`.

3.3 GATE Developer

GATE Developer je grafické rozhraní, které umožňuje výzkum a vývoj softwaru pro zpracování jazyka. Základní pojmy k orientaci jsou:

- **documents** - dokumenty, nad kterými se mají provést anotace,
- **corpora** - korpus, který sdružuje několik dokumentů,
- **annotations** - anotace vytvořeny v dokumentu,
- **annotation types** - typ anotace, jako je například datum,
- **annotation sets** - seskupují anotace pod jednu skupinu (volně přeloženo - „sada anotací“),
- **processing resources** - zdroj, který vykoná algoritmus nad dokumentem například vytvoří nové anotace,
- **applications** - aplikace obsahující pipeline mezi PR a dokumenty.

[2] http://ant.apache.org

GATE Developer umožňuje uložení relace při ukončení programu. Takže při příštím spuštění se automaticky načtou všechna nastavení např. typ písma. Nastavení této funkčnosti se provede zvolením volby „Options“ a poté „Configuration“ v hlavním menu. Následně v záložce „Advanced“ v části pojmenované „Session persistence“.

3.3.1 Zdroje „Language Resources“

V GATE Developer je možné pracovat s dokumenty. V této části bude popsáno jak načíst dokument do rozhraní, jak nastavit parametry a základní manipulace.

Typ zdroje Language Resources načteme kliknutím pravým tlačítkem myši v levém postranním okně na položku „Language Resources“ a výběrem „New“[3]. Po tomto kroku je možné vybrat vždy minimálně ze dvou možností a těmi jsou: „GATE Document“ a „GATE Corpus“.

Dokument

Dokument načteme volbou „GATE Document“, po které se objeví okno pro nastavení parametrů dokumentu a jeho názvu. Pokud se nezadá název dokumentu tak je vygenerován. Povinné parametry jsou `collectReositioningInfo`, `markupAware`, `preserveOriginalContent` a `sourceUrl/stringContent`. URL cesta k dokumentu je určena parametrem `sourceUrl/stringContent`, typ kódování je zadán v parametru `encoding` (obvykle UTF-8). Parametr `markupAware` umožňuje zapnutí funkce hodnotami `true` nebo `false` pro načtení podporovaných tagů do skupiny anotací „Annotation Sets“ a nazvané jako „Original markups“. Rozeznané tagy v načteném dokumentu nejsou zobrazeny (v textové podobě) během jeho prohlížení v hlavním okně programu, ale je možné zobrazit oblast, kterou tagy ohraničují viz sekce 3.3.4.

Pokud bude vyžadována opětovná práce s dokumentem v „GATE Developer“ je tu možnost uložit dokument v „Datastores“ (popsáno v sekci 3.3.5) a nebo další možností je dokument exportovat ve formátu XML.

Korpus

Korpus vytvoříme volbou „GATE Corpus“ a v následujícím okně zvolením názvu korpusu (parametr „Name“) a skrze parametr „dokumetnList“ se mohou vkládat již načtené dokumenty v prostředí GATE Developer. Editovat korpus lze i po jeho vytvoření například ho otevřeme v hlavním okně přes dvojklik levým tlačítkem myši a můžeme postupně přidávat nebo odstraňovat z korpusu dokumenty. Nebo po kliknutí pravým tlačítkem myši na korpus v levém okně zvolíme možnost „Populate“, což nám umožňuje načtení složky se

[3] „New“ znamená načtení již existujícího dokumentu a nikoli jeho vytvoření. Obsah načteného dokumentu je možné kdykoli měnit.

všemi dokumenty, zvolení přípony dokumentů a jejich kódování. Dokumenty mohou být obsaženy v několika korpusech zároveň.

3.3.2 Zdroje „Processing resources“

Aby bylo možné manipulovat se zdroji Processing resource a využívat jejich prostředky, musejí se nejprve zpřístupnit v grafickém rozhraní GATE Developer. Zpřístupnění zdrojů docílíme pomocí prostředku pro správu pluginů viz níže.

Správa pluginů

Všechny zdroje nejsou ihned dostupné v GATE Developer a některé se musejí dodatečně přidat pomocí Manage CREOLE plugins (volně přeloženo jako „správa CREOLE pluginů“). Otevření Manage CREOLE plugins provedeme zvolením „File“ v hlavním menu a následně „Manage CREOLE plugins“. Stejného výsledku dosáhneme také přes ikonu na hlavním panelu připomínající tvar otazníku.

V následně otevřeném okně se zobrazí všechny dostupné pluginy. Pro každý plugin jsou možné tři volby a jsou to:

- **Load now** - načtení zdroje pro aktuálně spuštěnou relaci GATE Developer,
- **Load always** - načtení zdroje při každém spuštění GATE Developer,
- **Delete** - odstranění pluginu ze současné nabídky, nikoli trvalé odstranění.

Pokud není hledaný zdroj v nabídce, protože se nachází mimo standardní uložiště pro pluginy anebo byl plugin přidán do uložiště až po spuštění GATE Developer, lze využít tlačítka „Add a CREOLE repository“ (umístěné v dolní části okna) k dodatečnému přidání do seznamu.

V podokně s popisem „CREOLE resources in directory“ na pravé straně jsou uvedeny zdroje obsažené v označeném pluginu.

Načtení zdrojů k dalšímu použití

V levém postranním okně načteme zpracovatelné zdroje kliknutím pravým tlačítkem na „Processing resources“ a poté výběrem „New“, po tomto kroku je možné vybrat požadovaný zdroj podobně jako u „Language Resources“. Tímto způsobem můžeme postupně vybrat několik zdrojů, přičemž na pořadí vkládání nezáleží.

Po vybrání zdroje se objeví okno obsahující kolonku pro zadání názvu. U některých zdrojů se zadávají také inicializační parametry, které obvykle není možné měnit v pozdější

fázi tj. po vytvoření zdroje. Ovšem pokud zvolený zdroj povoluje měnit inicializační parametry lze toho docílit přes dvojklik levým tlačítkem myši na daný zdroj, což následně zobrazí položky v hlavním okně. Další a častější parametry jsou tzv. „run-time“ parametry. Tyto „run-time“ parametry lze měnit podle potřeby až po načtení zdroje.

3.3.3 Aplikace

GATE Developer nabízí vytvoření aplikace, které jsou v grafickém rozhraní nazvané „Aplications“. Tyto aplikace umožňují zakombinovat, nastavit vlastnosti a pořadí vykonávání Processing resources.

Vytvoření applications

První kroky k vytvoření jedné položky jsou podobné jako u výše popsaných zdrojů viz sekce 3.3.2.

Applications nabízí vytvoření pěti druhů položek: „Conditional Corpus Pipeline“, „Conditional Pipeline“, „Corpus Pipeline“, „Pipeline“, „Real-Time Corpus Pipeline“. „Corpus Pipeline“ se odlišuje od samotné „Pipeline“ tím, že se vztahuje ke korpusu s již načtenými dokumenty.

Natavení

Po vytvoření položky „Corpus Pipeline“ dvojklikem levým tlačítkem myši zobrazíme její nastavení v hlavním okně. Do části nazvané „Selected Processing resources“ se přidávají „Processing resources“, které se mají vykonat. Jednotlivé zdroje mohou na sobě záviset jako například zdroje „The Gazetteer“ je závislý na „Tokeniser“, které jsou součástí instalace. A proto je potřeba dbát na správné pořadí zdrojů. Podle vybraného zdroje lze nastavovat „run-time“ parametry v dolní části hlavního okna.

Aby bylo možné aplikaci spustit je nutné zvolit korpus u položky nazvané „Corpus“. Následné spuštění aplikace se provede pomocí tlačítka „Run this application“ v dolní části podokna.

3.3.4 Anotace

Anotace je označení, v tomto případě, části textu. Jsou dva obecné postupy jak vytvořit anotace. Jedním je tzv. automatická anotace a druhým je manuální anotace, která je zejména užitečná při dodatečné opravě anotací.

Automatické anotace

Automatické anotace v dokumentu jsou vytvářeny při jeho načtení (parametr `markupAware` nastaven na `true`) nebo spuštěním aplikace v GATE Developer, která obsahuje zdroj/e Processing Resources, které jsou k tomu uzpůsobený viz sekce 3.3.2.

Manuální anotace

Nutným předpokladem pro manuální anotaci je mít požadovaný dokument načten viz sekce 3.3.1. Zobrazení dokumentu v hlavním okně GATE se provede přes dvojklik levým tlačítkem myši v levém podokně na dokument. Manuální anotace nebo editace již existující se provede označením odpovídající části textu v editoru a následně podržením kurzoru nad označenou oblastí nebo pravým tlačítkem myši je vyvoláno menu pro editaci a vytváření anotace. Přes toto menu se nastaví název anotace, vlastnosti a hodnoty.

Popis možností poskytované skrze menu pro anotaci:

- Některé druhy anotace lze vybírat z nabídky a nebo si vytvořit vlastní, totéž platí pro vlastnosti tzn. pokud některá z anotací nabízí výběr vlastnosti je možné ji použít nebo vytvořit vlastní. Toto menu také dovoluje smazat anotace ve vyznačené části textu.

Pro více informací o manuální anotaci odkazuji na GATE User Guide sekce 3.4.5 (Creating and Editing Annotations)

Zobrazení anotací

Aby bylo možné prohlížet anotace musíme nejprve zobrazit dokument v hlavním okně. Následně v horní části hlavního okna je dostupných několik tlačítek: „Annotation Sets“, „Annotation List“, „Annotation Stack“, „Co-reference Editor“, „Text“ a tlačítko pro vyhledávání v textu s ikonou lupy. Popis funkcí některých tlačítek:

- Skrze tlačítko **Text** se přepíná mezi zobrazením a skrytí textu.
- Skrze tlačítko **Annotation Sets** se zobrazí v pravé části podokno se seznamem skupin anotací (Annotation Set), kde je možné zvolit jaké anotace se mají aktuálně zobrazovat v textu. Zaškrtnuté anotace v seznamu jsou v textu barevně zvýrazněny.
- Tlačítkem **Annotation List** se zobrazí v dolní části podokno. Toto podokno má formu tabulky s podrobnými informacemi o jednotlivých anotacích. Anotace jsou uspořádaná po řádcích a sloupce v tabulce jsou: Typ („Type“), Sada („Set“), začátek („Start“) a konec („End“) pozice anotace, Id a vlastnost („Features“). Skrze tuto tabulku se dají detailně prohlížet jednotlivé anotace a jejich parametry.

- Tlačítko **Annotation Stack** zobrazí v dolní části podokno, které poskytuje užitečný a podrobný náhled na zvýrazněné anotace v textu. Přehledně zobrazuje hranice aktuálně zvolených anotací v okolí označeného textu. V horní části tohoto podokna je také možné využít tlačítko pro posunutí ukazatele na další hranici („Next boundary“) a nebo na předchozí hranici („Previous boundary“) anotace.

3.3.5 Ukládání dat

Pokud je vyžadováno, aby při opětovném spuštění GATE Developer byly Language Resources zdroje ve stavu jako byly naposled je možné využít služeb „Datastores“. Datastores umožňuje uložení více LR najednou.

Postup je následující:

1. Vytvoření prázdné složky v počítači ještě před vytvořením Datastores.
2. Vytvoření Datastores - klikněte pravým tlačítkem myši na „Datastores“, potom vybrat „Create Datastores“. Objeví se nabídka typů Datastores - zvolte nabídku začínající „SerialDataStore:...“[4] a nakonec zvolte složku vytvořenou v předchozím kroku.
3. Uložení požadovaného dokumentu/korpusu se provede kliknutím na dokument/korpus pravým tlačítkem myši a následně zvolením „Save to Datastore...“. Posledním krokem je výběr konkrétního uložiště (je nutné, aby uložiště bylo otevřeno/načteno v GATE Developer před ukládáním zdrojů).

3.3.6 Export do formátu XML

Vyberte „Save as XML...“ z nabídky vyvolanou kliknutím pravým tlačítkem myši na dokument nebo korpus a poté zvolte místo uložení a název ukládaného souboru.

3.3.7 Ukládání aplikací

Společně s Applications jsou ukládané i související Processing Resources. Konkrétní Applications uložíme výběrem volby „Save Application State“ vyvolanou pravým tlačítkem myši (standardní přípona ukládaného souboru je .gapp nebo .xgapp, ale není podmínkou).

3.3.8 Obnovení relace

Data se načtou přes „Datestores“ v levém podokně. Uložiště se otevře kliknutím pravým tlačítkem myši na „Datestores“, volbou „Open Datestores“ a na konec výběrem odpoví-

[4] „SerialDataStore:file-based storage using java serialisation“ je „Serial Datastores“ založené na nástroji serializace z programového jazyku Java.

dající složky. Po úspěšném načtení uložiště dat ho zobrazíme v hlavním okně (dvojklikem levým tlačítkem myši) a konkrétní dokument/korpus načteme volbou „Load“, vyvolanou pravým tlačítkem myši.

Applications a Processing Resources se načtou následovně: Zvolte „Restore application from file“ (pod nabídkou „File“ v hlavním menu GATE) a vyberte požadovaný soubor.

3.4 Vytvoření nového pluginu

Pro GATE je možné implementovat mimo jiné zdroje PR. Pluginy v GATE Developer jsou dostupné přes URL adresu. Popis načtení zdrojů viz sekce 3.3.2. GATE Developer vyhledá konfigurační soubor `creole.xml` vzhledem k danému pluginu a podle tohoto souboru GATE Developer rozezná jaké zdroje plugin poskytuje, jejich vlastnosti a kde se nacházejí třídy, které implementují dané typy zdrojů (třídy jsou obvykle uloženy v JAR souboru ve složce pluginu). Konfigurační data mohou být uložena jak v souboru `creole.xml`, tak i jako Java anotace ve zdrojovém kódu.

3.4.1 Processing resource

Třída vykonávající proces je uvedena v `creole.xml` souboru jako zdroj, aby mohla být GATE Developer rozeznána. Tato třída obsahuje kromě jiných metod hlavní metodu `execute()` ke spuštění zdroje, tzv. setter a getter (nastavení a získání hodnoty parametru) pro všechny parametry zdroje. Metody `set...` a `get...` musejí odpovídat názvu parametru například pro parametr `annotationSetName` existují metody `setAnnotationSetName(type param)` a `getAnnotationSetName()`. Další používané metody pro Processing Resources jsou:

- `Resource init()` - inicializuje zdroj a vrátí ho.
- `void reInit()` - znovu inicializuje zdroj a měl by být ve stejném stavu jako po zavolání `Resource init()`. Využívá-li zdroj externí zdroje pak jsou znovu načteny.
- `void interrupt()` - oznámí, že zdroj má být zastaven ihned jak jen to bude možné.
- `boolean isInterrupted()` - zjistí, jestli byl zdroj přerušen po jeho posledním spuštění.

Třída `AbstractLanguageAnalyser` poskytuje metody `setDocument()`, `getDocument()`, `getCorpus()` a `getCorpus()` pro práci s dokumentem a korpusem. Popis dalších metod pro práci s LR:

- `DocumentContent getContent()` - vrátí obsah dokumentu.

- `void edit(Long start, Long end, DocumentContent replacement)` - úprava obsahu dokumentu.
- `void setContent(DocumentContent newContent)` - nahradí celý obsah dokumentu.
- `public AnnotationSet getAnnotations()` - navrátí výchozí sadu anotací.
- `public AnnotationSet getAnnotations(String name)` - navrátí sadu anotací podle parametru (názvu sady).
- `public Map getNamedAnnotationSets()` - navrátí všechny pojmenované sady anotací.
- `void removeAnnotationSet(String name)` - odstraní sadu anotací podle parametru (názvu sady).

3.4.2 Soubor `creole.xml`

Jak již bylo řečeno `creole.xml` je konfigurační soubor, který definuje plugin. Kořenový element ve struktuře souboru `creole.xml` je `<CREOLE-DIRECTORY>`. Obvykle parametry v kořenovém elementu nejsou nutné, nicméně může obsahovat například `ID`, `VERSION`, `DESCRIPTION` atd. viz GATE User Guide sekce 4.7. V kořenovém elementu by měly být jen elementy `<RESOURCE>` pro každý typ zdrojů v pluginu. Element `<RESOURCE>` může být obsažen v elementu `<CREOLE>`. Příklad jedné z možných struktur:

```
<CREOLE-DIRECTORY>
   <CREOLE>
      <RESOURCE>
         <NAME>Regex Segmentation</NAME>
         <JAR>segmentation.jar</JAR>
         <CLASS>cz.zcu.segmentation.RegexSplitter</CLASS>
         <COMMENT>Any comment about the resource.</COMMENT>
         <PARAMETER NAME="annotationSetName"
            RUNTIME="true">java.lang.String</PARAMETER>
         <PARAMETER NAME="regex"
            RUNTIME="true">java.lang.String</PARAMETER>
      </RESOURCE>
      <RESOURCE>
         <NAME>Origin Sentences</NAME>
         <JAR>segmentation.jar</JAR>
         <CLASS>cz.zcu.segmentation.OriginalAnnotation</CLASS>
         <COMMENT>Any comment about the resource.</COMMENT>
         <PARAMETER NAME="annotationSetName"
            RUNTIME="true">java.lang.String</PARAMETER>
      </RESOURCE>
   </CREOLE>
</CREOLE-DIRECTORY>
```

Výše uvedený příklad definuje plugin mající dva zdroje, kde první obsahuje dva „run-time“ parametry a komentář ke zdroji. Druhý zdroj má jeden „run-time“ parametr a komentář.

Každý element `<RESOURCE>` musí obsahovat elementy `<NAME>`, `<JAR>` a `<CLASS>`. Přehled možných elementů vnořených do `<RESOURCE>`:

- **NAME** je název zdroje, pod kterým je reprezentován v GATE Developer. V případě vynechání tohoto elementu je vygenerován podle názvu třídy zdroje.
- **CLASS** je celý název Java třídy, která implementuje tento zdroj.
- **JAR** obsahuje název JAR souboru, který obsahuje zdroje. Těchto elementů může být více kromě předchozího použití například URL cesta k pomocným knihovnám. Uvedená cesta k JAR souboru je relativní k vzhledem k souboru `creole.xml`.
- **COMMENT** je popis zdroje, který se objevuje jako popisek v GATE Developer.
- **HELPURL** poskytuje URL cestu k nápovědě na internetu pro daný zdroj.
- **INTERFACE** je typ rozhraní implementované prostřednictvím tohoto zdroje. Může to být například nový typ dokumentu.
- **ICON** je ikona zobrazovaná v GATE Developer. Cesta k ikoně je uvedena relativně vzhledem k umístění v JAR souboru. Pokud cesta nezačíná lomítkem tak se předpokládá, že se ikona nachází v knihovně `gate.jar` (v `gate/resources/img`).
- **PRIVATE** je použit pro zdroje, které mají být skryty před uživatelem v GATE Developer. Takové zdroje jsou využívané jinými zdroji.
- **AUTOINSTANCE (and HIDDEN-AUTOINSTANCE)** předá informaci GATE, které instance zdrojů mají být vytvořeny při načtení pluginu.
- **TOOL** představuje typ zdrojů umístěné jako součást „Tools menu in GATE Developer“.
- **RESOURCE_DISPLAYED** a **MAIN_VIEWER** jsou typy zdroje Visual Resources.

Zdroje mohou mít také různé parametry definované v elementu `<PARAMETR>`. Parametry mohou obsahovat atributy, které definují mimo jiné tzv. výchozí hodnoty, komentáře zobrazující v GATE Developer atd. Atributy v elementu `<PARAMETR>` mohou být:

- **NAME** je název parametru odkazující na parametry ve zdroji. Například pro parametr nazvaný „annotationSetName“ (ve zdrojovém kódu třídy implementující Processing resource) musejí existovat metody `setAnnotationSetName` a `getAnnotationSetName`.
- **DEFAULT** je výchozí hodnota parametru.
- **RUNTIME** určuje, jestli je nutné nastavit parametr při spouštění Processing Resources, v opačném případě je parametr vyžadován při inicializaci.
- **COMMENT** komentář popisující účel parametru.

- **OPTIONAL** není vyžadován.
- **ITEM_CLASS_NAME** platí pouze pro typ parametru `java.util.Collection` nebo od něj odvozené. Specifikuje typ kolekce, které zdroj navíc akceptuje. Pokud není nastaveno, tak všechny elementy jsou akceptovány jako řetězec.
- **SUFFIXES** platí pouze pro typ parametru `java.net.URL` a uvedené hodnoty (oddělené středníkem) vymezují jaké typy souborů mohou být vybrány.

Parametry jsou uzavřeny v elementu `<OR>`, pokud se mají vzájemné vylučovat. Typ parametru je určen významem řetězce v elementu `<PARAMETR>`, kterému musí odpovídat návratový typ „get“ metody. Typem parametru může být třída, rozhraní nebo výčtový typ nebo také jiný typ zdroje. Ovšem v případě primitivních typů (char, int, ...), které nejsou podporovány musí se použít místo nich odpovídající typ tj. (`java.lang.Character`, `java.lang.Integer`, ...). Pokud je nutné, aby metoda vracela hodnotu jako je např. kolekce `List<Integer>` je místo kolekce zadána hodnota `java.util.List`. Typ `java.net.URL` je zpracován tak, že pokud není URL adresa absolutní je použita relativní adresa s kořenovým adresářem odpovídající umístění `creole.xml` souboru. Více detailů ohledně parametrů se nachází v GATE User Guide.

3.4.3 Integrace pluginu do GATE Developer

Již vytvořený plugin doporučuji před jeho použitím v GATE Developer umístit do složky určenou pro pluginy tj. `<root>/plugins`, kde `root` je domovský adresář GATE Developer. Takto bude plugin automaticky načten do Manage CREOLE plugins při spuštění GATE Developer. V opačném případě je nutné ručně přidat URL adresu pluginu do Manage CREOLE plugins viz sekce 3.3.2.

4 Plugin „Segmentation“

V rámci této práce vznikl plugin „Segmentation“ archivován v souboru:
`gate_plugin_segmentation.zip`, který obsahuje následující PR zdroje:

Evaluator

Tento zdroj porovná vybrané anotace vytvořené nad dokumentem a poté vypíše přesnost a jiné statistiky o výsledku porovnání. Vzorová anotace pro porovnání je `original_sentences` generována zdrojem `OriginalAnnotation` (název sady anotace obsahující originální anotaci se zadává jako run-time parametr `annotationOriginalName`). Anotace k porovnání jsou zvolena zadáním názvu skupiny anotací do run-time parametru `annotationSetName` (jsou porovnávané všechny přidružené anotace pod zvolenou skupinou anotací), v případě nezadání názvu tj. zadání prázdného řetězce jsou porovnávaná všechna anotace vytvořena nad dokumentem se vzorovou anotací.

Original Sentences

Tento zdroj vygeneruje anotace označující jednotlivé věty tzv. originální věty. Vstupem je předpokládaný dokument, který obsahuje jednotlivé věty rozděleny do řádků. Název skupiny anotace je dána run-time parametrem `annotationOriginalName` a výsledná anotace má název `original_sentences`. Tato anotace je následně použitelná zdrojem `Evaluator`.

Chars Segmentation

Tento zdroj vygeneruje anotaci `sentences_chars` označující v textu věty. Text je rozdělen na věty na základě předem dané posloupnosti znaků definované v algoritmu. Obsahuje run-time parametr pro volbu názvu skupiny anotace `annotationSetName`.

Regex Segmentation

Tento zdroj vygeneruje anotaci `sentences_regex` označující v textu věty. Věty jsou rozděleny na základě regulárního výrazu definovaném v algoritmu nebo zvolením jiného výrazu v run-time parametru `regex`. Obsahuje run-time parametr pro volbu názvu skupiny anotace `annotationSetName`.

ClassifierTrainer

Tento zdroj vygeneruje soubor (pomocí Java nástroje pro serializaci) s výchozím názvem `trained_classifier.ser`, který obsahuje natrénovaný klasifikátor a je využívaný zdrojem `ClassifierMaxEnt`. Pro nastavení jiného názvu souboru slouží run-time parametr `nameFileClassifier`. Vstupní dokument je vyžadován takový, který má jednotlivé věty rozdělené do řádků, stejně jako vyžaduje zdroj `OriginalAnnotation`.

Classifier

Tento zdroj provede anotaci na základě již natrénovaného klasifikátoru viz výše zdroj **ClassifierTrainer**. Výsledná anotace je pojmenovaná `sentences_max_ent`. Výsledné anotace označující v textu věty. Inicializačním parametrem je soubor obsahující natrénovaný klasifikátor.

Dále obsahuje soubor `creole.xml`, zdrojové kódy, binární soubory, vygenerovanou programátorskou dokumentaci nástrojem JavaDoc, tuto dokumentaci se zdrojovými kódy, pomocné soubory ve složce `zk`, potřebnou knihovnu, již natrénovaný klasifikátor rozdělené do kategorií viz sekce 5.4 a JAR soubor.

GATE Embedded

Pokud jsou vyvinuté zdroje, které mají být součástí jiné aplikace použije se GATE Embedded. A potom výsledné zdroje jsou dodány jako několik JAR souborů.

Spuštění zdrojů GATE Embedded

Následující text popisuje spuštění zdrojů z příkazové řádky pomocí vytvořeného JAR souboru na základě GATE Embedded.

Soubor `segmentation_app.zip` obsahuje JAR soubor `segmentation_app.jar`, který obsahuje třídy `ClassifierMaxEntTrainerApp`, `ClassifierMaxEntApp` a `SegmentationApp`.

- `ClassifierMaxEntTrainerApp` - využívá služeb zdroje ClassifierTrainer z GATE pluginu Segmentation,
- `ClassifierMaxEntApp` - využívá služeb zdroje Classifier z GATE pluginu Segmentation,
- `SegmentationApp` - využívá služeb zdrojů Regex Segmentation, Original Sentences a Chars Segmentation z GATE pluginu Segmentation.

Dále obsahuje složku `corpus`, ve které je dokument `data.txt` využívaný výše uvedenými třídami.

Spuštění JAR souboru `segmentation_app.jar` (umístěný v adresáři, v němž je také GATE Developer a složka `corpus`) pomocí příkazové řádky lze například příkazem `java -cp './segmentation_app.jar' cz.zcu.segmentation.app.ClassifierMaxEntApp`. Pro spuštění `ClassifierMaxEntApp` musí existovat natrénovaný klasifikátor pojmenovaný `trained_classifier.ser` a uložen ve stejném adresáři jako spouštěný JAR soubor.

5 Implementace

GATE podporuje technologii JavaBeans, což jsou znovu použitelný komponenty v jazyce Java. Komponenty v GATE jsou programovány v jazyce Java. Takže vyvinutý plugin „Segmentation“ je implementován ve stejném programovém jazyce.

V kapitole 4 je popsán plugin z hlediska jeho rozsahu funkčnosti tj. jaké zdroje jsou jeho součástí a jejich účel.

5.1 Struktura programu

Následující popis implementace je rozdělen podle následujících balíků.

5.1.1 cz.zcu.segmentation

Tento balík obsahuje třídy `Evaluator`, `CharsSplitter`, `RegexSplitter`, `OriginalAnnotation`, `ClassifierMaxEnt` a `ClassifierMaxEntTrainer`. Tyto třídy implementují algoritmy pro segmentaci textu využívanou v GATE a jsou tedy zdroji výsledného pluginu „Segmentation“.

Evaluator

Tato třída rozšiřuje třídu `AbstractLanguageAnalyser`, která poskytuje metody pro práci s dokumentem a korpusem. Dále třída `Evaluator` obsahuje metodu `execute()`, která vykonává daný algoritmus, jejímž výsledkem je vyhodnocení přesnosti vytvořených anotací v dokumentu. Také obsahuje metody:

- `public String getAnnotationSetOriginalName()` - slouží k získání hodnoty privátní proměnné `annotationSetOriginalName`. Tato proměnná obsahuje název skupiny anotace vzhledem, které se má provádět vyhodnocení.
- `public void setAnnotationSetOriginalName(String aSetOriginalName)` - slouží k nastavení hodnoty privátní proměnné `annotationSetOriginalName`. Tato proměnná obsahuje název skupiny anotace vzhledem, ke které se má vykonat vyhodnocení.
- `public String getAnnotationSetComparingName()` - slouží k získání hodnoty privátní proměnné `annotationSetComparingName`. Tato proměnná obsahuje název skupiny anotace, která se bude vyhodnocovat.

- `public void setAnnotationSetComparingName(String aSetComparingName)` - slouží k nastavení hodnoty privátní proměnné `annotationSetComparingName`. Tato proměnná obsahuje název skupiny anotace, která se bude vyhodnocovat.

CharsSplitter

Tato třída rozšiřuje třídu `AbstractLanguageAnalyser`, která poskytuje metody pro práci s dokumentem a korpusem. Dále třída `CharsSplitter` obsahuje metodu `execute()`, která vykonává daný algoritmus, jejímž výsledkem je segmentace textu v dokumentu prostřednictvím anotací. Také obsahuje metody:

- `private int extractSentence(String text)` - tato metoda je pomocná hlavní metody této třídy tj. `execute()`. Vrací index hranice věty.
- `public String getAnnotationSetName()` - slouží k získání hodnoty privátní proměnné `annotationSetName`. Tato proměnná obsahuje název skupiny anotace, která označuje jednotlivé věty rozeznané algoritmem.
- `public void setAnnotationSetName(String annotationSetName)` - slouží k nastavení hodnoty privátní proměnné `annotationSetName`. Tato proměnná obsahuje název skupiny anotace, která označuje jednotlivé věty rozeznané algoritmem.

RegexSplitter

Tato třída rozšiřuje třídu `AbstractLanguageAnalyser`, která poskytuje metody pro práci s dokumentem a korpusem. Dále třída `RegexSplitter` obsahuje metodu `execute()`, která vykonává daný algoritmus, jejímž výsledkem je segmentace textu v dokumentu prostřednictvím anotací. Také obsahuje metody:

- `private int extractSentence(String text)` - tato metoda je pomocná hlavní metody této třídy tj. `execute()`. Vrací index hranice věty.
- `private String nextWord(String text, int index)` - tato metoda je pomocná pro hlavní metodu této třídy tj. `execute()`. Vrací hranici dalšího slova.
- `public String getRegex()` - slouží k získání hodnoty privátní proměnné `regex`. Tato proměnná obsahuje regulární výraz podle, kterého se provádí segmentace.
- `public void setRegex(String regex)` - slouží k nastavení hodnoty privátní proměnné `regex`. Tato proměnná obsahuje regulární výraz podle, kterého se provádí segmentace.
- `public String getAnnotationSetName()` - slouží k získání hodnoty privátní proměnné `annotationSetName`. Tato proměnná obsahuje název skupiny anotace, která označuje jednotlivé věty rozeznané algoritmem.

- `public void setAnnotationSetName(String annotationSetName)` - slouží k nastavení hodnoty privátní proměnné `annotationSetName`. Tato proměnná obsahuje název skupiny anotace, která označuje jednotlivé věty rozeznané algoritmem.

OriginalAnnotation

Tato třída rozšiřuje třídu `AbstractLanguageAnalyser`, která poskytuje metody pro práci s dokumentem a korpusem. Dále třída `OriginalAnnotation` obsahuje metodu `execute()`, která vykonává daný algoritmus, jejímž výsledkem je segmentace textu v dokumentu prostřednictvím anotací. Také obsahuje metody:

- `public String getAnnotationSetName()` - slouží k získání hodnoty privátní proměnné `annotationSetName`. Tato proměnná obsahuje název skupiny anotace, která označuje jednotlivé věty rozeznané algoritmem.
- `public void setAnnotationSetName(String annotationSetName)` - slouží k nastavení hodnoty privátní proměnné `annotationSetName`. Tato proměnná obsahuje název skupiny anotace, která označuje jednotlivé věty rozeznané algoritmem.

ClassifierMaxEntTrainer

Tato třída využívá model maximální entropie, což je softwarová struktura tzv. "framework", která slouží pro začlenění informací z různých zdrojů za účelem klasifikace. Třída rozšiřuje třídu `AbstractLanguageAnalyser`, která poskytuje metody pro práci s dokumentem a korpusem. Dále třída `ClassifierMaxEntTrainer` obsahuje metodu `execute()`, která pomocí metody (obsahující příznaky) `private FeatureExtractorManager<Character> getFeatureSet()` natrénuje klasifikátor. Výsledkem této třídy je výsledný soubor s natrénovaným klasifikátorem [3]. Také obsahuje metody:

- `private ArrayList<Character> getSentences(Corpus corpus)` - slouží k získání anotací, které jsou součástí daného korpusu a v případě neexistence inicializuje požadovanou anotaci.
- `private ArrayList<Integer> getLabels(Corpus corpus)` - viz `private ArrayList<Character> getSentences(Corpus corpus)`
- `private void serialize(Classifier<Character> classifier) throws Exception` - tato metoda vytvoří soubor s natrénovaným klasifikátorem.
- `public URL getSerializedClassifier()` - slouží k získání hodnoty privátní proměnné `serializedClassifier`, která obsahuje URL cestu k adresáři. V tomto koncovém adresáři se vytvoří výstupní soubor.
- `public void setSerializedClassifier(URL serializedClassifier)` - slouží k nastavení hodnoty privátní proměnné `serializedClassifier`, která obsahuje URL cestu k adresáři. V tomto koncovém adresáři se vytvoří výstupní soubor.

- `public String getNameFileClassifier()` - slouží k získání hodnoty privátní proměnné `nameFileClassifier`, která obsahuje název výsledného souboru.
- `public void setNameFileClassifier(String nameFileClassifier)` - slouží k nastavení hodnoty privátní proměnné `nameFileClassifier`, která obsahuje název výsledného souboru.

ClassifierMaxEnt

Tato třída využívá model maximální entropie, což je softwarová struktura tzv. "framework", která slouží pro začlenění informací z různých zdrojů za účelem klasifikace. Třída rozšiřuje třídu `AbstractLanguageAnalyser`, která poskytuje metody pro práci s dokumentem a korpusem. Dále třída `ClassifierMaxEnt` obsahuje metodu `execute()`, která vykonává daný algoritmus, jejímž výsledkem je segmentace textu v dokumentu prostřednictvím anotací [3]. Také obsahuje metody:

- `public Resource init() throws ResourceInstantiationException` - slouží k nastavení proměnných nutné k vykonání algoritmu tj. načtení natrénovaného klasifikátoru.
- `private Classifier<Character> deserialize() throws Exception` - tato metoda načte soubor s natrénovaným klasifikátorem.
- `public URL getSerializedClassifier()` - slouží k získání hodnoty privátní proměnné `serializedClassifier`, která obsahuje URL cestu k souboru s natrénovaným klasifikátorem.
- `public void setSerializedClassifier(URL serializedClassifier)` - slouží k nastavení hodnoty privátní proměnné `serializedClassifier`, která obsahuje URL cestu k souboru s natrénovaným klasifikátorem.
- `public String getAnnotationSetName()` - slouží k získání hodnoty privátní proměnné `setAnnotationSetName`. Tato proměnná obsahuje název skupiny anotace, která označuje jednotlivé věty rozeznané algoritmem.
- `public void setAnnotationSetName(String nameFileClassifier)` - slouží k nastavení hodnoty privátní proměnné `setAnnotationSetName`. Tato proměnná obsahuje název skupiny anotace, která označuje jednotlivé věty rozeznané algoritmem.

5.1.2 cz.zcu.segmentation.app

Tento balík obsahuje třídy, které inicializují požadované zdroje a vynutí provedení jejich algoritmů. Všechny třídy v tomto balíku obsahují následující metody:

- `public static void main(String[] args) throws Exception` - hlavní metoda, která nastaví parametry pro spuštění zdroje/ů tj. vyvolá inicializaci dokumentu a parametrů pomocí níže uvedených metod, creole souboru a ostatních.
- `private static SerialAnalyserController initApp() throws Exception` - inicializuje parametry nutné pro spuštění odpovídajících zdrojů.
- `private static Corpus initCorpus() throws Exception` - inicializuje dokument pro který se mají vykonat příslušné algoritmy.

ClassifierMaxEntApp

Tato třída vyvolá spuštění zdroje `cz.zcu.segmentation.ClassifierMaxEnt`

ClassifierMaxEntTrainerApp

Tato třída vyvolá spuštění zdroje `cz.zcu.segmentation.ClassifierMaxEntTrainer`

SegmentationApp

Tato třída vyvolá spuštění zdrojů v tomto pořadí `cz.zcu.segmentation.CharsSplitter`, `cz.zcu.segmentation.RegexSplitter` a `cz.zcu.segmentation.OriginalAnnotation`.

5.1.3 cz.zcu.segmentation.feature

Tento balík obsahuje třídy, které implementují rozhraní `FeatureExtractor` a jsou příznaky pro natrénování klasifikátoru.

AbbreviationsD1

Tento příznak identifikuje řetězec délky dva (včetně tečky zakončující zkratku) nastaví označení neboli kategorii na hodnotu 1. Implementuje tyto metody:

- `public void train(TrainingInstanceList<Character> instances)` - načte ze souboru seznam zkratek délky řetězce o dvou znacích včetně tečky a uloží do datové struktury `ArrayList<String>`,

- `public int getNumberOfFeatures()` - vrátí počet vlastností příznaku tj. velikost seznamu se zkratkami,
- `public void extractFeature(InstanceList<Character> instances, FeatureVectorGenerator generator)` - provede extrakci informace na základě seznamu zkratek. V případě nalezení zkratky v posloupnosti znaků nastaví příznak na hodnotu 1 odpovídající kategorii.

AbbreviationsD2

Tento příznak identifikuje řetězec délky tři (včetně tečky zakončující zkratku) nastaví označení neboli kategorii na hodnotu 1. Implementuje tyto metody:

- `public void train(TrainingInstanceList<Character> instances)` - načte ze souboru seznam zkratek délky řetězce o třech znacích včetně tečky a uloží do datové struktury `ArrayList<String>`,
- `public int getNumberOfFeatures()` - vrátí počet vlastností příznaku tj. velikost seznamu se zkratkami,
- `public void extractFeature(InstanceList<Character> instances, FeatureVectorGenerator generator)` - provede extrakci informace na základě seznamu zkratek. V případě nalezení zkratky v posloupnosti znaků nastaví příznak na hodnotu 1 odpovídající kategorii.

AbbreviationsD3

Tento příznak identifikuje řetězec délky čtyři (včetně tečky zakončující zkratku) nastaví označení neboli kategorii na hodnotu 1. Implementuje tyto metody:

- `public void train(TrainingInstanceList<Character> instances)` - načte ze souboru seznam zkratek délky řetězce o čtyřech znacích včetně tečky a uloží do datové struktury `ArrayList<String>`,
- `public int getNumberOfFeatures()` - vrátí počet vlastností příznaku tj. velikost seznamu se zkratkami,
- `public void extractFeature(InstanceList<Character> instances, FeatureVectorGenerator generator)` - provede extrakci informace na základě seznamu zkratek. V případě nalezení zkratky v posloupnosti znaků nastaví příznak na hodnotu 1 odpovídající kategorii.

CommonStartSen

Tento příznak identifikuje znak, který běžně ukončuje větu a nastaví označení neboli kategorii na hodnotu 1 v případě nalezení shody. Implementuje tyto metody:

- `public int getNumberOfFeatures()` - vrátí počet vlastností příznaku tj. v tomto případě 1,
- `public void extractFeature(InstanceList<Character> instances, FeatureVectorGenerator generator)` - provede extrakci informace na základě definovaných znaků, které běžně ukončují větu.

Date

Tento příznak identifikuje různé tvary kalendářních dat v textu podle souboru s daty a nastaví označení, neboli kategorii na hodnotu 1 v případě nalezení shody. Implementuje tyto metody:

- `public void train(TrainingInstanceList<Character> instances)` - načte ze souboru seznam různých tvarů kalendářních dat a uloží do datové struktury `ArrayList<String>`,
- `public int getNumberOfFeatures()` - vrátí počet vlastností příznaku tj. velikost seznamu s daty,
- `public void extractFeature(InstanceList<Character> instances, FeatureVectorGenerator generator)` - provede extrakci informace na základě seznamu zkratek. V případě nalezení data v posloupnosti znaků nastaví příznak na hodnotu 1 odpovídající kategorii.

Chars

Tento příznak ke každému znaku v textu, který zakončuje větu přiřadí označení neboli kategorii na hodnotu 1. Implementuje tyto metody:

- `public void train(TrainingInstanceList<Character> instances)` - načte všechny znaky v textu a uloží je do datové struktury `ArrayList<String>`,
- `public int getNumberOfFeatures()` - vrátí počet vlastností příznaku tj. velikost seznamu se znaky,
- `public void extractFeature(InstanceList<Character> instances, FeatureVectorGenerator generator)` - provede extrakci informace na základě seznamu zkratek. V případě nalezení znaku v posloupnosti zakončující větu nastaví příznak na hodnotu 1 odpovídající kategorii.

PotentialAbbreviation

Tento příznak identifikuje znak tečky a zkoumá její okolí. Podle podmínek, které splňují požadavky na konec věty je nastaveno označení neboli kategorie na hodnotu 1. Implementuje metody:

- `public int getNumberOfFeatures()` - vrátí počet vlastností příznaku tj. v tomto případě 1,
- `public void extractFeature(InstanceList<Character> instances, FeatureVectorGenerator generator)` - provede extrakci informace z textu na základě podmínek zda je znak tečky odklopován jinými znaky. V případě nalezení shody v posloupnosti znaků nastaví příznak na hodnotu 1 odpovídající kategorii.

5.1.4 cz.zcu.segmentation.input

Tento balík obsahuje rozhraní pro přístup k souboru, který obsahuje data potřebná v některých třídách v balíku `cz.zcu.segmentation.feature` a implementaci tohoto rozhraní.

Rozhraní - Input

Toto rozhraní `Input` definuje metody pro práci se vstupním souborem tj. nastavení název, kódování a čtení souboru. Obsahuje metody:

- `public void setFileName(String file)` - slouží k nastavení názvu načítaného souboru,
- `public void setEncoding(String encoding)` - slouží na nastavení kódování souboru,
- `public ArrayList<String> load()` - slouží k načtení obsahu souboru a vrátí obsah v datové struktuře `ArrayList<String>`.

InputData

Tato třída slouží k načtení souboru, který obsahuje pomocná data nutné pro některé algoritmy. Implementuje rozhraní `Input` viz výše sekce **Rozhraní - Input**.

Metoda `public ArrayList<String> load()` načte soubor a po řádkách ukládá pomocná data do datové struktury `ArrayList<String>`.

5.2 Soubor `creole.xml`

Tento soubor obsahuje definici všech zdrojů pluginu a jejich parametrů. Každý zdroj je uzavřen v elementu `<RESOURCE>`, dále obsahují elementy `<NAME>`, `<JAR>` (JAR soubor se zdroji tj. `segmentation.jar`), `<CLASS>`. Dále podrobnější popis zdrojů (celý obsah souboru `creole.xml` je v příloze A):

- Zdroj **Regex Segmentation** obsahuje dva run-time parametry „annotationSetName“ a „regex“ specifikované datovými typy `java.lang.String`.
- Zdroj **Original Sentences** obsahuje run-time parametr „annotationSetName“ specifikovaný datovým typem `java.lang.String`.
- Zdroj **Evaluator** obsahuje dva run-time parametry „annotationSetOriginalName“ a „annotationSetComparingName“ specifikované datovými typy `java.lang.String`.
- Zdroj **Chars Segmentation** obsahuje run-time parametr „annotationSetName“ specifikovaný datovým typem `java.lang.String`.
- Zdroj **ClassifierTrainer** obsahuje dva run-time parametry „nameFileClassifier“ a „serializedClassifier“, kde první je specifikovaný datovým typem `java.lang.String` a druhý `java.net.URL`. Další element `<JAR>` obsahuje relativní cestu k pomocné knihovně.
- Zdroj **Classifier** obsahuje parametr „serializedClassifier“ specifikovaný datovým typem `java.net.URL` a „annotationSetName“ specifikovaný datovým typem `java.lang.String`. Další element `<JAR>` obsahuje relativní cestu k pomocné knihovně.

5.3 Soubor `build.xml`

Tento build soubor slouží k vytvoření dvou ZIP souborů. Příkazy pro vytvoření výsledných souborů a odstranění nepotřebných souborů:

- `ant build` - vytvoří archiv pluginu viz kapitola 4 pro GATE Developer s touto a programátorskou dokumentací. Je také výchozím příkazem.
- `ant buildApp` - vytvoří archiv viz kapitola 4 část „Spuštění zdrojů GATE Embedded“.
- `ant clean` - odstraní binární soubory a ZIP soubory vygenerované příkazy viz předchozí body.

Pro podrobnější informace nahlédněte do zdrojových kódů.

5.4 Zhodnocení pluginu „Segmentation“

Plugin Segmentation a jeho zdroje byly spuštěny nad korpusem `PDT_sentence_per_line_-UTF-8.txt`, který byl dodán vedoucím této bakalářské práce, čítající přesně 90 828 vět rozdělené do řádků. Následující zdroje byla spuštěna na stroji s 4 jádrovým procesorem o frekvenci 1.6 GHz[1].

Zdrojem „Original Sentences“ se vytvořily anotace `original_sentences` a vyhodnocení segmentace pro následující zdroje bylo provedeno zdrojem „Evaluator“. Statistiky segmentace jsou uvedené v tabulkách 5.3 a 5.2 viz níže. Vytvoření těchto anotací přibližně trvalo několik sekund až desítek sekund.

ClassifierTrainer a Classifier

Zdroj „ClassifierTrainer“ vygeneroval samotný soubor s natrénovaným klasifikátorem v čase cca 3 minuty. K natrénování klasifikátoru byly použity různé kombinace příznaků. V následující tabulce 5.1 je znázorněn vztah mezi číslem kombinace a příznakem a tyto vztahy jsou použity i v tabulce 5.2.

Číslo kom.	**Příznak/y**
1	AbbreviationsD1; AbbreviationsD2; AbbreviationsD3; CommonStartSen; Date; Chars; PotentialAbbreviation
2	AbbreviationsD1; AbbreviationsD2; AbbreviationsD3; Chars
3	AbbreviationsD1; AbbreviationsD2; AbbreviationsD3; Chars; PotentialAbbreviation
4	Chars
5	Date; Chars
6	Chars; PotentialAbbreviation
7	CommonStartSen; Chars
8	CommonStartSen; Date; Chars

Tabulka 5.1: Vztah mezi č. kombinace a příznaky.

Název souboru se serializovaným klasifikátorem je odvozen od kombinace příznaků, které byly použity k jeho natrénování např. pro kombinaci č. 1 odpovídá soubor `1_trained_classifier.ser` atd.

[1]Přesná konfigurace: CPU 1.6 GHz (4 jádro), 6GB RAM, OS Windows 7

Dále zdroj „Classifier“ vyprodukoval anotace `sentences_max_ent` v čase cca 2 minut pro všechny natrénované klasifikátory viz výše tabulka 5.1. V následující tabulce 5.2 je shrnutí úspěšnosti tohoto zdroje.

Číslo kombinace	**Přesnost**	**Rozeznaných vět**
1	62,654%	56908 ze 90 828
2	62,654%	56908 ze 90 828
3	63,535%	57708 ze 90 828
4	64,645%	58716 ze 90 828
5	63,535%	57708 ze 90 828
6	63,535%	57708 ze 90 828
7	64,645%	58716 ze 90 828
8	62,654%	56908 ze 90 828

Tabulka 5.2: Úspěšnosti segmentace podle kombinace příznaků.

V tabulce 5.2 jsou znázorněny úspěšnosti segmentace zdrojem **Classifier**, kde číslo kombinace identifikuje příznaky použité k natrénování klasifikátoru, který byl zdrojem použit viz tabulka 5.1.

Chars Segmentation a Regex Segmentation

Zdroje pluginu „Chars Segmentation“ a „Regex Segmentation“ vytvořily anotace `sentences_chars` a `sentences_regex`. Vytvoření těchto anotací trvalo cca několik desítek sekund. V následující tabulce 5.3 je shrnuta úspěšnost segmentace.

Anotace	**Přesnost**	**Rozeznaných vět**
`sentences_chars`	62,623%	56 885 ze 90 828
`sentences_regex`	62,841%	57 078 ze 90 828

Tabulka 5.3: Úspěšnost - zdrojů **Chars Segmentation** a **Regex Segmentation**.

Shrnutí

Pro zdroje Chars Segmentation a Regex Segmentation vychází úspěšnost kolem 62%, kde zdroj Regex Segmentation vykazuje mírně lepší výsledky a to o 0,118%. Dále zdroj Classifier vykazuje nejlepší výsledky pro kombinaci č. 4 a 7 (64,645%) a nejnižší pro č. 1, 2 a 8 (62,654%). Ovšem v průměru segmentace zdrojem Classifier je o 0,859% přesnější než zdroj Chars Segmentation a o 0,641% než Regex Segmentation.

6 Závěr

V této práci byla diskutována problematika segmentace textu v českém jazyce. Bylo zkoumáno několik situací, ve kterých dochází k nejednoznačnosti u některých interpunkčních znaků. Tyto situace byly popsány a analyzovány. Na základě zjištěných poznatků z analýzy byly vytvořeny algoritmy k realizaci segmentace.

Bylo nutné seznámit se systémem GATE a popsat jeho základní vlastnosti a možnosti v rámci rozsahu segmentace. V kapitole o GATE byly nejprve popsány základní pojmy a k čemu je primárně určen. V další části o GATE byl popsán GATE Developer a potom následuje popis vytvoření pluginu, potřebné kroky k jeho integraci a všeho s tím souvisejícím.

Výsledný plugin byl implementován v jazyce Java a zahrnuje několik tzv. zdrojů pro vykonání segmentace. Dále plugin obsahuje zdroje potřebné pro vyhodnocení provedené segmentace nad textem. Zhodnocení výsledků provedených segmentací v rámci této práce se nachází v sekci 5.4. Výsledky se liší velmi málo a je mezi nimi rozdíl cca 1%, přesnější čísla se nacházejí v dříve zmíněné sekci. U jednoho zdroje byl využit model maximální entropie, pro který bylo navrhnuto několik tzv. příznaků a i pro tento zdroj se vyhodnocení segmentace odlišovalo cca 1%.

Zdrojové kódy pluginu je možné použít jako základ nebo jako vzor pro podobné projekty segmentace textu.

Seznam zkratek

PR, PRs Processing Resources

Jsou entity zpracovávající linguistické data. Tyto zdroje realizují různé algoritmy pro zpracování linguistických dat. Mohou anotovat text podle určitých pravidel například rozezná věty, slova atd.

LR, LRs Language Resources

Jsou entity, jejichž obsahem jsou linguistické data. Data mohou poskytovat textové dokumenty, korpusy, lexikony a jiné.

VR, VRs Visual Resources

Jsou komponenty, které vytvářejí grafické rozhraní.

Literatura

[1] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.

[2] Gregory Grefenstette and Pasi Tapanainen. What is a word, what is a sentence? problems of tokenization. pages 79–87, 1994.

[3] Christopher D. Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, Cambridge, MA, USA, 1999.

[4] Andrei Mikheev. Periods, capitalized words, etc. *Comput. Linguist.*, 28:289–318, September 2002.

[5] David D. Palmer and Marti A. Hearst. Adaptive sentence boundary disambiguation. In *Proceedings of the fourth conference on Applied natural language processing*, ANLC '94, pages 78–83, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics.

[6] Helmut Schmid. *Tokenizing*. 2007.

A XML schéma CREOLE souboru

```
<CREOLE-DIRECTORY>
        <CREOLE>
                <RESOURCE>
                        <NAME>Regex Segmentation</NAME>
                        <JAR>segmentation.jar</JAR>
                        <CLASS>cz.zcu.segmentation.RegexSplitter</CLASS>
                        <PARAMETER NAME="annotationSetName" RUNTIME="true">java.
                            lang.String</PARAMETER>
                        <PARAMETER NAME="regex" RUNTIME="true">java.lang.String</
                            PARAMETER>
                </RESOURCE>
                <RESOURCE>
                        <NAME>Original Sentences</NAME>
                        <JAR>segmentation.jar</JAR>
                        <CLASS>cz.zcu.segmentation.OriginalAnnotation</CLASS>
                        <PARAMETER NAME="annotationSetName" RUNTIME="true">java.
                            lang.String</PARAMETER>
                </RESOURCE>
                <RESOURCE>
                        <NAME>Evaluator</NAME>
                        <JAR>segmentation.jar</JAR>
                        <CLASS>cz.zcu.segmentation.Evaluator</CLASS>
                        <PARAMETER NAME="annotationSetOriginalName" RUNTIME="true">
                            java.lang.String</PARAMETER>
                        <PARAMETER NAME="annotationSetComparingName" RUNTIME="true"
                            >java.lang.String</PARAMETER>
                </RESOURCE>
                <RESOURCE>
                        <NAME>Chars Segmentation</NAME>
                        <JAR>segmentation.jar</JAR>
                        <CLASS>cz.zcu.segmentation.CharsSplitter</CLASS>
                        <PARAMETER NAME="annotationSetName" RUNTIME="true">java.
                            lang.String</PARAMETER>
                </RESOURCE>
                <RESOURCE>
                        <NAME>ClassifierTrainer</NAME>
                        <JAR>lib/ner-0.1.jar</JAR>
                        <JAR>segmentation.jar</JAR>
                        <CLASS>cz.zcu.segmentation.ClassifierMaxEntTrainer</CLASS>
                        <PARAMETER NAME="serializedClassifier"
                                RUNTIME="true">
                                java.net.URL
                        </PARAMETER>
                        <PARAMETER NAME="nameFileClassifier"
                                RUNTIME="true">
                                java.lang.String
                        </PARAMETER>
                </RESOURCE>
```

```
                <RESOURCE>
                        <NAME>Classifier</NAME>
                        <JAR>lib/ner-0.1.jar</JAR>
                        <JAR>segmentation.jar</JAR>
                        <CLASS>cz.zcu.segmentation.ClassifierMaxEnt</CLASS>
                        <PARAMETER NAME="serializedClassifier"
                                RUNTIME="false">
                                java.net.URL
                        </PARAMETER>
                        <PARAMETER NAME="annotationSetName" RUNTIME="true">java.
                            lang.String</PARAMETER>

                </RESOURCE>
        </CREOLE>
</CREOLE-DIRECTORY>
```

B Postup sestavení pluginu Segmentation

Sestavení pluginu Segmentation bylo testováno na operačních systémech GNU/Linux a Windows. Níže jsou uvedené nutné předpoklady k sestavení.

- **Java Development Kit (verze 1.6 nebo vyšší)** potřebné pro kompilaci zdrojových kódů v jazyce Java.
- Nástroj **Apache Ant (verze 1.8 nebo vyšší)** pomocí, kterého se provede proces sestavení.

Sestavení archivu s pluginem pro systém GATE Developer spustíte pomocí příkazu `ant build` z příkazové řádky z kořenového adresáře projektu.