Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Generovanie UML modelu z prípadov použitia

Diplomová práca

2024
Bc. Filip Zaikner

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Generovanie UML modelu z prípadov použitia

Diplomová práca

Študijný program:	Aplikovaná Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. Lukáš Radoský

Bratislava, 2024
Bc. Filip Zaikner

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Filip Zaikner
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Generovanie UML modelu z prípadov použitia
Generating UML model from use cases

Anotácia: Prípady použitia predstavujú skvelý nástroj pre komunikáciu medzi zákazníkom, analytikom, a v konečnom dôsledku aj architektom či programátorom. Umožňujú porozumieť procesom, ktoré bude daný informačný systém či softvér podporovať. Než sú prípady použitia pretavené do finálneho produktu v podobe zdrojového kódu, prechádzajú mnohými fázami, najmä návrhom a implementáciou, často vo viacerých iteráciách. Vývoj softvéru by bol značne rýchlejší a lacnejší, ak by bolo možné z prípadov použitia automatizovane odvodiť štrukturálne alebo behaviorálne modely, ktoré vizuálne reprezentujeme napríklad diagramom tried či diagramom sekvencií.

Analyzujte existujúce prístupy pre konverziu prípadov použitia na modely bližšie zdrojovému kódu. Navrhnite a implementujte metódu pre konverziu prípadov použitia na niektorý z týchto modelov. Umožnite vizualizáciu vytvorených modelov. Svoju metódu a jej implementáciu overte na testovacej množine dát.

Cieľ: Vytvorte prototyp využívajúci novú alebo zdokonalenú existujúcu metódu pre konverziu prípadov použitia na UML model. Vytvorený prototyp bude poskytovať vizualizáciu vytvorených modelov vo forme diagramov. Overte svoje riešenie pomocou množiny testovacích dát.

Literatúra:
Cockburn, A.: Writing Effective Use Cases. Boston, MA, USA: AddisonWesley Longman Publishing Co., Inc., first edition, 2000, ISBN 0201702258.

Thakur, J.S. and Gupta, A., 2016, September. AnModeler: a tool for generating domain models from textual specifications. In 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 828-833). IEEE.

Deeptimahanti, D. K.; Babar, M. A.: An Automated Tool for Generating UML Models from Natural Language Requirements. In 2009 IEEE/ACM International Conference on Automated Software Engineering, Nov 2009, ISSN 1938-4300, s. 680–682, doi:10.1109/ASE.2009.48.

Vedúci: Ing. Lukáš Radoský
Katedra: FMFI.KAI - Katedra aplikovanej informatiky

Poďakovanie:

Abstrakt

Kľúčové slová:

Abstract

Keywords:

Obsah

Kapitola 1

Teoretické východiská

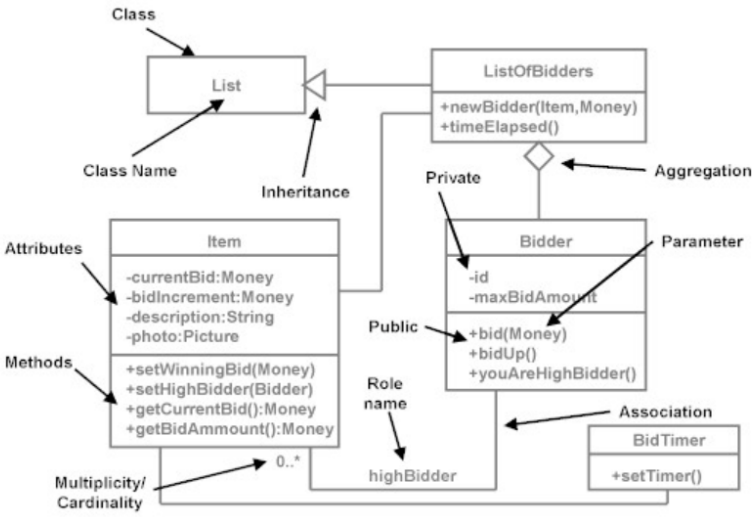
V tejto kapitole si rozoberieme to, z čoho budeme vychádzať. Sústredíme sa na nástroje, ktoré sa využívajú na prácu s prirodzeným jazykom a analyzujeme už existujúce práce a riešenia, ktoré sa zaoberajú podobnou témou. Na základe analýzy týchto zdrojov, určíme smerovanie našej práce.

1.1 Unified Modeling Language

Unified Modeling Language, skrátene UML, je grafický jazyk a štandard v softvérovom inžinierstve, určený na reprezentáciu, dokumentáciu a vizualizáciu návrhu softvérového riešenia[1]. Jazyk UML ,vytvorený firmou Object Management Group (OMG) koncom 90. rokov 20. storočia, sa stal v roku 2005 ISO štandardom, a od vtedy sa stal prvou a jasnou voľbou každého softvérového návrhára alebo architekta. Vo svete grafických jazykov, určených na návrh systémov má v tomto čase malú konkurenciu. Jediný porovnateľný jazyk, ktorý má dostatočne vyjadrovacie schopnosti sa nazýva Archimate, a ten je využívaný skôr na biznis stránku. Od svojho vzniku vznikli už 3 veľké revízie a aktualizácie jazyka UML. Aktuálny štandard je 3.0.UML modely vieme rozdeliť na dve skupiny: štrukturálne diagramy a behaviorálne diagramy.

1.1.1 Štrukturálne diagramy

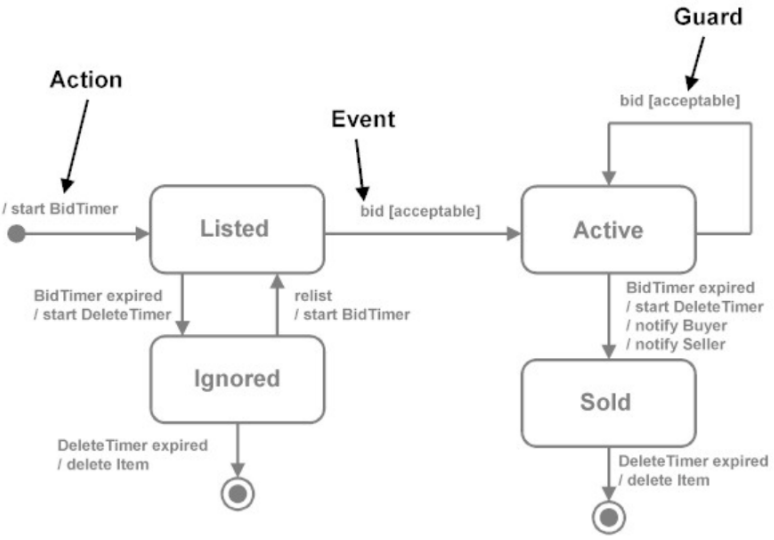
Štrukturálne diagramy reprezentujú statickú stránku systému. Tieto diagramy znázorňujú stavbu systému a organizáciu elementov systému. Zaoberajú sa elementami ako sú komponenty, triedy, vzťahy medzi objektami. Medzi prominentné diagramy patria diagram tried, diagram komponentov a diagram nasadenia. Štrukturálny diagram typicky pozostáva z artefaktov a pomenovaných hrán.



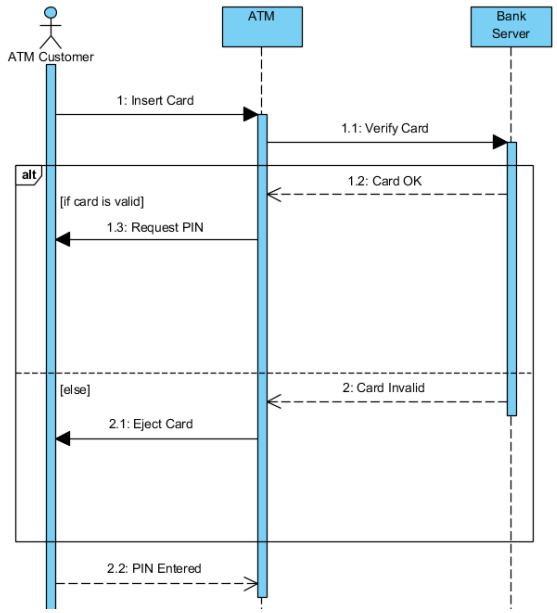
Obr. 1.1: Príklad štrukturálneho diagramu - diagram tried

1.1.2 Behaviorálne diagramy

Behaviorálne diagramy zaznamenávajú dynamiku systému. Zaoberajú sa najme správaním systému, stavmi jednotlivých elementov a interakciami medzi elementami vo vnútri systému ale aj interakciami externých aktérov. Taktiež je to jediný spôsob ako zaznamenať komunikáciu a nadväznosť správ. Medzi behaviorálne diagramy patria najmä diagram prípadov použitia, diagram aktivít a sekvenčný diagram. Podľa štruktúry sa dajú rozdeliť na dve kategórie. Behaviorálny diagram buď pozostáva z procesov/aktivít reprezentovaných objektami(v obdĺžnikoch s oblými hranami) a hrán reprezentujúcich ich postupnosť (diagram aktivít, stavový diagram), alebo pozostáva z artefaktov a hrán reprezentujúcich aktivity (sekvenčný diagram).



Obr. 1.2: Príklad behaviorálneho diagramu - stavový diagram



Obr. 1.3: Príklad behaviorálneho diagramu - sekvenčný diagram

1.1.3 Prípady použitia

Prípady použitia vytvoril Ivarom Jacobssonom[2] počas práce pre firmu Ericsson. Prípady použitia zaznamenávajú v prirodzenom jazyku sekvenciu interakcií medzi systémom a používateľom systému, za cieľom splniť nejaký cieľ[3]. Slúžia ako hlavný popis formálnych požiadaviek systému. Prípady použitia nie sú nevyhnutne spojené s UML, a ich existencia je staršia ako existencia UML. UML je primárne grafický jazyk, kým prípady použitia sú textové opisy napísané v prirodzenom jazyku. Pre grafické znázornenie prípadov použitia bol navrhnutý UML diagram prípadov použitia. Existuje množstvo notacií pri písaní prípadov použitia. Najrozšírenejšia notácia po tej Jacobsonnovej je notácia Alistaira Cockburna[3]. Prípady použitia pozostávajú typicky z týchto častí:

- **Názov**: Názov prípadu použitia.
- **Popis**: Stručný popis prípadu použitia.
- **Aktéri**: Aktéri sú vnútorné alebo externé entity, ktoré interagujú so systémom. Medzi aktérov môže patriť nie len človek, ale aj iný systém alebo subsystému.

Aktérov poznáme primárnych a sekundárnych. Jeden aktér môže účinkovať vo viacerých prípadoch použitia.

- **Prekondície**: Prekondície sú podmienky, ktoré musia byť splnená pred začiatkom vykonávania hlavného toku.
- **Postkondície**: Postkondície sú podmienky, ktoré budu platiť po úspešnom uskutočnení hlavného toku.
- **Hlavný tok**: Hlavný tok symbolizuje najkratšiu, optimistickú sekvenciu akcií uskutočnených v prípadoch použitia. V hlavnom toku sa nenachádzajú žiadne odchylky od plánovej funkcionality systému, musí byť jednoduchý a priamy. Kroky zvyknú byť očíslované. Ivar Jacobson, autor prípadov použitia, označil krok toku ako jednotnú tranzakciu.[3]
- **Alternatívne toky**: Ak príde ku odchyľke od hlavnej funkcionality systému, tak takéto okrajové prípady rieši alternatívny tok. Príkladom je výber financií z bankomatu klienta ktorý nedisponuje dostatočným finančným krytím.

Vzorový prípad použitia vyzerá nasledovne:

- **Názov**: Nákup zľavneného tovaru.
- **Popis**: Užívateľ si prezrie ponuku zľavneného tovaru na stránke, vyberie si tovar a vloží do košíku.
- **Aktéri**: zákazník, systém
- **Prekondície**: Zákazník má prístup na internet.
- **Hlavný tok**

  1. Systém zobrazí zoznam zľavnených tovarov.
  2. Zákazník si vyberie špecifický zľavnený tovar.
  3. Systém zobrazí detaily zľavneného tovaru.
  4. Zákazník klikne na tlačidlo "Pridať do košíka".
  5. Systém skontroluje počet dostupných zľavnených tovarov.
  6. Systém vloží zľavnený tovar do košíka.

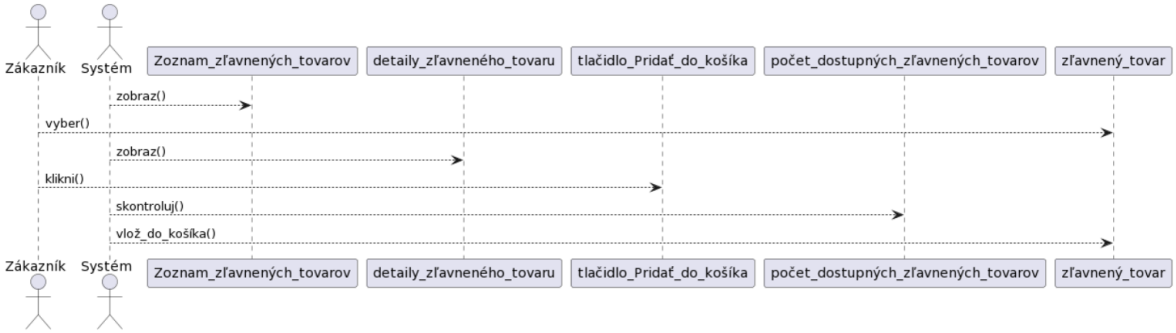
1.1.4 Sekvenčný diagram

Sekvenčný diagram je behaviorálny UML diagram, ktorý je určený na modelovanie interakcií medzi objektami alebo procesmi v čase. Sekvenčný diagram je navrhnutý tak, aby v ňom bolo znázornené plynutie času. Plynutie času je umerné vzdialenosti na čiare života od začiatku diagramu. Diagram sa skladá z niekoľkých častí:

- **Aktéri**: Aktéri sú externé entity ktoré interagujú so systémom.
- **Objekty**: Objekty alebo komponenty, sú časti systému, ktoré medzi sebou komunikujú, aby splnili požiadavku na funkcionalitu.
- **Čiary života**: Čiara života symbolizuje existenciu objektu v čase. Značí sa čiarkovanou vertikálnou čiarou. V bode začiatku tejto čiary daný objekt vznikol, a v bode konca tejto čiary daná objekt zanikol.
- **Aktivačné bloky**: Aktivačné bloky vznikajú pri tom, ako objekt obdrží správu. Slúžia na to, aby znázornili, že je daný objekt aktívny, a je ním spracovávaná metóda, ktorá spracuváva dopyt. Po odoslaní odpovede aktivačné bloky zanikajú. Keďže jeden objekt môže v jednom momente spracovávať viac správ, je možné aby na jednej čiare života naraz existovalo viacej aktivačných blokov.
- **Správy**: Správy sú nástrojom komunikácie medzi objektami. Existuje množstvo druhov správ a delení správ. Najdôležitejšie delenie správ je medzi synchrónne a asynchrónne. Správy taktiež môžu obsahovať parametre.
- **Fragmenty**: Fragmenty reprezentujú štruktúry, typicky znázornené v obdĺžnikoch, s názvom fragmentu v pravom rohu, ktoré ovplyvňujú tok správ. Fragmenty sa dajú skladať z ďalších fragmentov. Takéto poskladané fragmenty nazývame kombinované fragmenty. Medzi najpoužívanejšie fragmenty patria:
  - **Fragment voliteľnosti**: Fragment voliteľnosti ,označený v diagrame ako opt, ponúka možnosť vykonania správ v tom prípade, ak splnená podmienka.
  - **Fragment alternatívy**: Fragment alternatívy, označený ako alt, je fragment, ktorý ponúka viacej podmienok v jednom fragmente. Analogicky sa tento fragment dá prirovnať ku viacerým fragmentov možnosti v jednom fragmente.
  - **Iteračný fragment**: Iteračný fragment, označený ako loop. Správy ktoré sa nachádzajú v tomto fragmente sa uskutočňujú niekoľkokrát, dovtedy, pokým sa neuskutočnia taký počet krát, ako je zapísané vo fragmente, alebo nie je splnená podmienka, ktorú je taktiež možné do fragmentu pridať.

– **Paralelný fragment**: Paralelný fragment, označený ako par, znázorňuje vykonávanie správ naraz v jeden moment.

Typicky priebeh scenára v sekvenčnom diagrame začína aktér, ktorý spustí daný scenár. Objekty od najvyššieho levelu si posielajú správy, až po najnižší level potrebný na spracovanie požiadavky. V tom momente ako objekt spracováva požiadavku, je na tomto objekte vytvorený aplikačný blok. Objekty si výsledky svojej činnosti typicky, ale nie nevyhnutne, vracajú správou typu odpoveď. Po tom ako objekt odošle správu typu odpoveď, aplikačný blok zaniká.



Obr. 1.4: Príklad behaviorálneho diagramu - sekvenčný diagram

1.2 Spracovanie prirodzeného jazyka

Spracovanie prirodzeného jazyka(Natural language processing - NLP) je oblasť výskumu a aplikácie, ktorá skúma ako počítače možno použiť na pochopenie textu alebo reči v prirodzenom jazyku a manipuláciu s nimi.[4] Využíva pri tom rôzne nástroje lingvistiky,informatiky,rôzne pravidlové systémy, štatistické metódy ale aj strojové učenie a hlboké učenie. Medzi využitia NLP patrí množstvo študijných odborov, ako je napríklad strojový preklad, spracovanie textu v prirodzenom jazyku, rozpoznávanie reči, umelá inteligencia,expertné systémy a pod.[4] Cieľom NLP je naučiť počítač rozumieť ľudskej reči tak, ako jej rozumie človek.

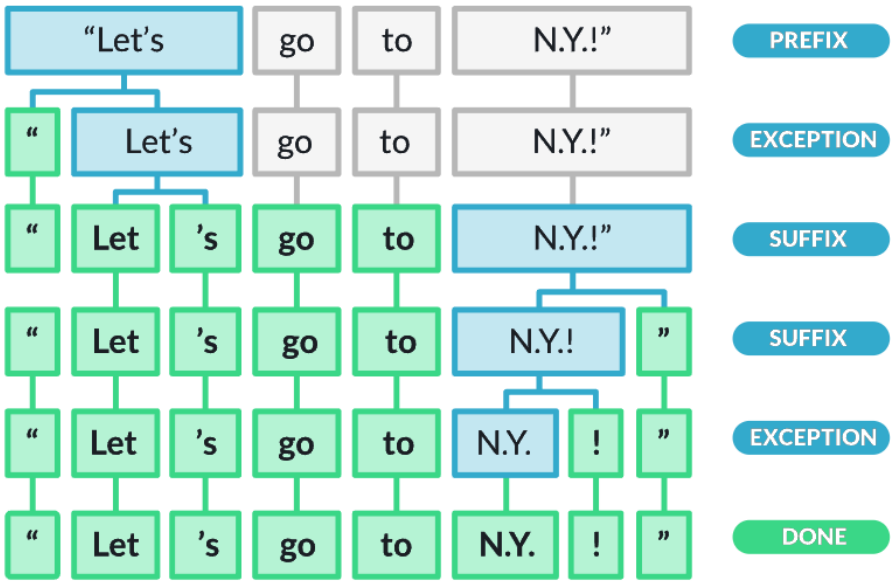
1.2.1 Nástroje spracovania prirodzenej reči

Medzi nástroje spracovania prirodzenej reči patrí:

- **Stemmovanie**: Stemmovanie je technika, typicky zameraná na transformáciu slova na základný tvar, pri ktorom sa používa napr. odstraňovanie prípon v slove.

Slovo spracované touto technikou, na rozdiel od lematizácie nemusí dávať zmysel. Príkladom je anglické slovo motory: engines - engin. Stemmovanie výrazne urýchluje proces indexácie.[5]

- **Tokenizácia**: Tokenizácia je proces identifikácie každej atomickej zmysluplné časti, teda súvetia, frázy ale najtypickejšie slova vo vete.[6] Zo všetkých NLP techník je táto navyužívanejšie. Využíva sa často pri vyhľadávacích systémoch, napr. Google alebo Apache Lucene.



Obr. 1.5: Proces tokenizácie

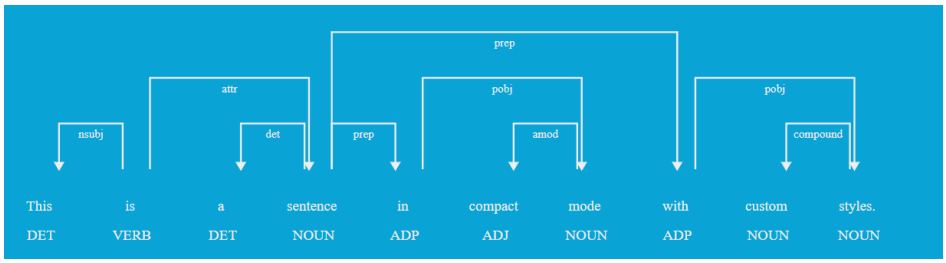
- **Lematizácia**: Pomocou lematizácie hľadáme slovotvorný základ slova. Výsledok však nie je deterministický. Skvelým príkladom je anglické slová „born“, ktorého lemma je aj slovo „born“, teda narodiť sa, ale aj slovo „bear“, teda slovo zniesť. Napriek nejednoznačnosti lematizácie patrí táto technika medzi kľúčové a najpoužívanejšie techniky NLP.[5]

- **Odstraňovanie stop slov**: Odstraňuje stop slova. Najčastejšie sa odstraňujú anglické členy ako „a“/„an“/„the“. Takýmto odstránením pre počítač nepotrebných znakov vieme oveľa lepšie optimalizovať výkon.

- **Part-of-speech-tagging**: Part-of-speech-tagging alebo inak gramatické tagovanie, priraďuje vo vete jednotlivým tokenom(slovám, ktoré prešli tokenizáciou) gramatické kategórie. Medzi tieto kategórie patrí rod, osoba, číslo a mnoho ďalších kategórií daného jazyka. Najvyužívanejšou gramatickou kategóriu je však slovný druh. Slovných druhov je v angličtine na rozdiel od slovenčiny len 9, a sú nimi: podstatné mená, slovesá, prídavné mená, príslovky, zámená, predložky,

spojky, citoslovce a častice. Rovnako však ako v slovenčine tieto druhy obsahujú poddruhy.

- **Rozoznávanie závislostí medzi slovami**: Rozoznávanie závislostí medzi slovami pomenováva vzťahy medzi súvetiami, frázami a jednotlivými slovami vo vete a priraďuje im jednotlivé vetné členy. Každému slovu vo vete priradí jemu prislúchajúci vetný člen. Takto vytvorený vzťah medzi jednotlivými časťami vety sa nazýva dependencia, teda závislosť. Všetky závislosti sa odvíjajú od jedného centrálneho slova vety, ktorý sa nazýva koreň. Z takto určených závislostí vieme skonštruovať takzvaný syntaktický strom.



Obr. 1.6: Rozpoznávanie závislostí medzi slovami

- **Open Information Extraction**: Open Information Extraction rozdeľuje vetu na trojicu: predmet, podmet a reláciu medzi nimi.
- **Rozpoznávanie pomenovaných entít**: Rozpoznávanie pomenovaných entít je proces, ktorým v texte rozpoznávame existujúce entity, typicky pomenované vlastnými podstatnými menami, napríklad mestá, pohoria, štáty atď. Príkladom je slovo Londýn.
- **CorefAnnotator**: CorefAnnotator je nástroj, ktorým vieme rozpoznať slová, ktoré sa referencujú na rovnakú entitu. Príkladom sú slová vták a papagáj, ktoré sa v jednom súvetí môžu referencovať na jedného konkrétneho jedinca.
- **Sémantická podobnosť**: Sémantická podobnosť je metrika, ktorou porovnávame, či dve vety majú rovnaký sémantický význam.

1.3 Existujúce prístupy

Nasledujúca časť sa zaoberá existujúcim vedeckým prácami zaoberajúcimi sa podobnou tématikou, akú riešime v našej práci.

1.3.1 An Automated Approach to Transform Use Cases into Activity Diagrams

tbd[7]

1.3.2 Generating UML Use Case and Activity Diagrams Using NLP Techniques and Heuristics Rules

tbd[8]

1.3.3 Automatic Generation of Sequence Diagram from Use Case Specification

tbd[9]

1.3.4 An Automated Tool for Generating UML Models from Natural Language Requirements

tbd[10]

1.3.5 Towards Automatically Extracting UML Class Diagrams from Natural Language Specifications

tbd[11]

1.3.6 Porovnanie existujúcich prístupov

1.3.7 RBeIPo: Advanced methods of analysis and design using multidimensional UML

V práci[12] sa autori zaoberajú tvorbou softvéru na modelovanie 3D diagramov aktivít z textového opisu v prirodzenom jazyku. Autori vytvorili grafické prostredie na prácu s takýmito diagramami.

Autori vytvorili vlastný jazyk RBeIPo, ktorý využívajú v ich práci. Obsahuje iba pár slov a slúži na tvorbu fragmentov. Slová z jazyka sa zadajú najskôr do prípadu použitia, a potom sa spracujú. Jazyk bol vytvorený v 4. iteráciach, a každá priniesla nové kľúčové slová do tohto jazyka. Tieto iterácie sú:

1. **If/Then/Else/ElseIf**: Tieto slová sú analógiou podmienok, a sú využívané na tvorbu rozhodovacích blokov, OPT fragmentov.

2. **For/EndFor**: Sú použité na opakované uskutočňovanie správ, teda fragment LOOP.

3. **Opt/When/EndWhile/EndWhen**: Slúžia pre tvorbu podmienené cyklu, teda cyklu, ktorý iteruje pokým nie je splnené podmienka. Vytvára fragment LOOP s podmienkou.

4. **Alt/Par/EndAlt/EndPar**: V poslednej iterácií boli pridané fragmenty alternatívy a paralelného spracovania.

Na spracovanie textu autori využívajú NLP techniky. Výsledný text sa spracuje následovne

1. **Tokenizácia**: Text sa pomocou tokenizácie pretransformuje na tokeny.

2. **Lematizácia alebo Stemmovanie**: Tokeny sa upravia na slovotvorný základ.

3. **Odstránenie stop slov**: Zadefinované stop slová sa odstránia.

4. **Part of speech a RBeIPo**: Tokenom sa určí slovný druh.

5. **Identifikácia aktivít**: Aktivity symbolizujú funkcionalitu, akciu systému. Preto ako aktivitu autori identifikovali všetky slovesá. Nesprávne idetifikované vetné členy sú opravované tzv. WhiteListom/BlackListom. Pre rozvinutie aktivít autor používa nástroje ako Stanford Universal Dependencies a Stanford Enhanced++ Universal Dependencies, ktoré identifikujú ďalšie slová, ktoré sú pridané do aktivít.

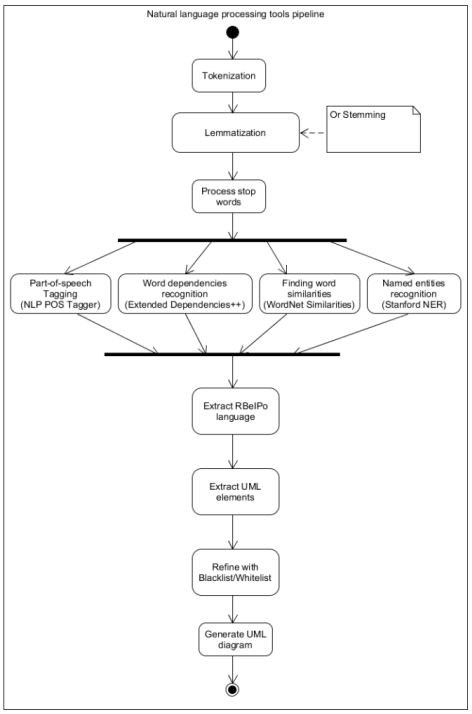
6. **Identifikácia plávacích dráh**: Spočiatku sa za plávajúce dráhy považovalo všetky podstatné mená, tých však v texte bolo príliš veľké množstvo. Neskôr sa miesto nich použili špecifické interakcie, a najme také, ktoré sa vyskytovali v NER.

7. **Identifikácia fragmentov**: Fragmenty sa určia pomocou autormi navrhnutého jazyka RBeIPo

8. **Identifikácia relácií medzi aktivitami, plávacími dráhami a fragmentmi**: Relácie medzi týmito elementami sú určované pomocou sémantickej podobnosti textu sekvencií slov, ignorujúc členy, keďže tie v kontexte nič nemenia. Pre určenie tejto podobnosti používajú, The Semantic Similarity Service, službu, ktorá používa WordNet a inú vlastnú databázu slov.

Tieto dáta sú neskôr vizualizované ako 3D model. Chyby urobené touto pipeline si užívateľ vie sám upraviť v grafickom prostredí.

Autori v tejto práci ako nástroj pre prácu s textom a technikou "Part of speech tagovanie"používajú knižnicu Stanford POS Tagger. Tento nástroj nie je dokonalý a mýli sa aj v niektorých často používaných slovách, preto autori museli zaviesť tzv. WhiteList/BlackList, pomocou ktorého ďalej spracuvávajú a upravujú výsledky.



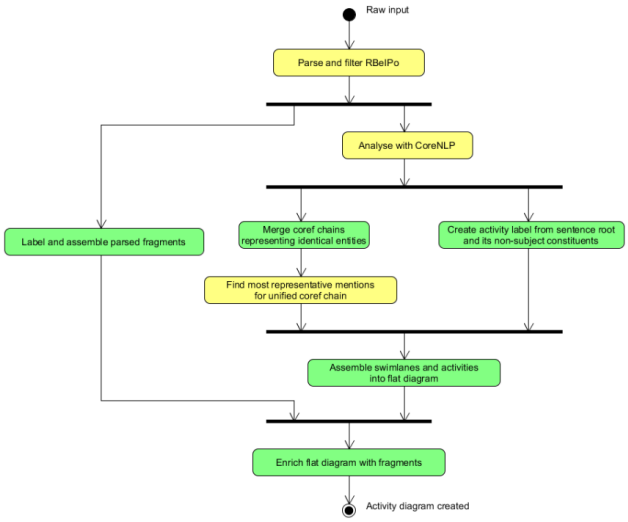
Obr. 1.7: RBeIPo NLP pipeline[12]

1.3.8 SMeIPo: Transformation of textual use cases into behavior UML diagram

Ďaľšiou prácou bol prístup SMeIPo[13]. S využitím jazyka RBeIPo vytvoreným predchádzajúcim prístupom vytvorili autori pipeline, ktorá produkuje jednu aktivitu za každú vetu. Spracovanie textu pozostávalo z nasledovných krokov:

1. **Part of speech**: Part of speech tokenom priradí jednotlivé slovné druhy. Na prácu s POS používa rovnako ako v predchádzajúcej práci Stanford CoreNLP.
2. **Rozpoznanie závislostí**: Rozpoznanie závislostí určí závislosti medzi slovami, teda im pridá vetné členy.

3. **Rozpoznanie koreferencií**: Koreferencie sa v tomto kroku iba vyhľadajú.
4. **Lematizácia**: Slová sa upravia lematizáciou na lemmy, teda na slovotvorný základ.
5. **Rozpoznanie fragmentov**: Pomocou nástroj ANTLR sa vytvorí hierarchický strom fragmentov. Tento strom sa vytvorí za pomoci jazyka RBeIPo.
6. **Zjednotenie koreferencií**: Vyhľadané koreferencie sa zjednotia. Na hľadanie referencií bol použitý nástroj WordNet.
7. **Blacklist**: Blacklist odstráni neželané slová, ktoré nenesú význam, a členy ako napr. ä, änä the.
8. **Konštruovanie aktivít**: Aktivity sú konštruované v pomere 1 aktivita za každú vetu. Za aktivity boli vybrané slovesá. Tieto slovesá však musia byť do vhodnej formy spracované v neskoršom spracovaní.
9. **Konštruovanie plávajúcich dráh**: Ako plávajúce dráhy boli vybrané podmety vety.



Obr. 1.8: SMoIPo NLP pipeline[13]

Autori sa v tejto práci venujú viacej okrajovými prípadom, ako je trpný rod alebo súvetia. Trpný rod autori v práci pomenovali ako výzvu, a vstupný prípad použitia s trpným rodom nevedia spracovať. Na rozdiel od predchádzajúceho prístupu však vedia spracovať súvetia rozdelené spojkou tak, že tieto vety osamostatnia. Na správne fungovanie riešenie taktiež vyžaduje blacklist a whitelist. Riešenie trpí tým, že dokáže spracovať len také vety, ktoré obsahujú jedno sloveso v činnom rode.

1.3.9 FBeIpo:Supporting software design by sequence diagram generation and visualization from use cases

Táto práca[14] sa na rozdiel od predchádzajúcich zaoberala tvorbou sekvenčných diagramov. Ako obe predchádzajúce práce, použili autori ako primárny nástroj na spracovanie prirodzeného jazyka Stanford NLP. Autori identifikovali 4 rôzne postupy, ktoré následne analyzovali. Vybrali si však následovný:

1. **Tokenizácia a lematizácia**: Vytvoria sa príslušné tokeny a lemmy, potrebne pre vstup do Open Information Extraction.

2. **Open Information Exctraction**: Open Information Exctraction je najdôležitejší proces v určení trojíc. Zlematizované tokeny sa vložia do Open Information Exctraction, ten potom vyprodukuje trojicu podmet, sloveso a predmet. Takýchto trojíc sa vygeneruje viac.

3. **Projektové slovníky**: Vďaka tomu že Open Information Exctraction generuje viacej trojíc naraz, autori museli nájsť algoritmy na to, aby z týchto trojíc boli schopný určiť tie najopisnejšie predmety, relácie a podmety. Boli navrhnuté nasledovné slovníky:

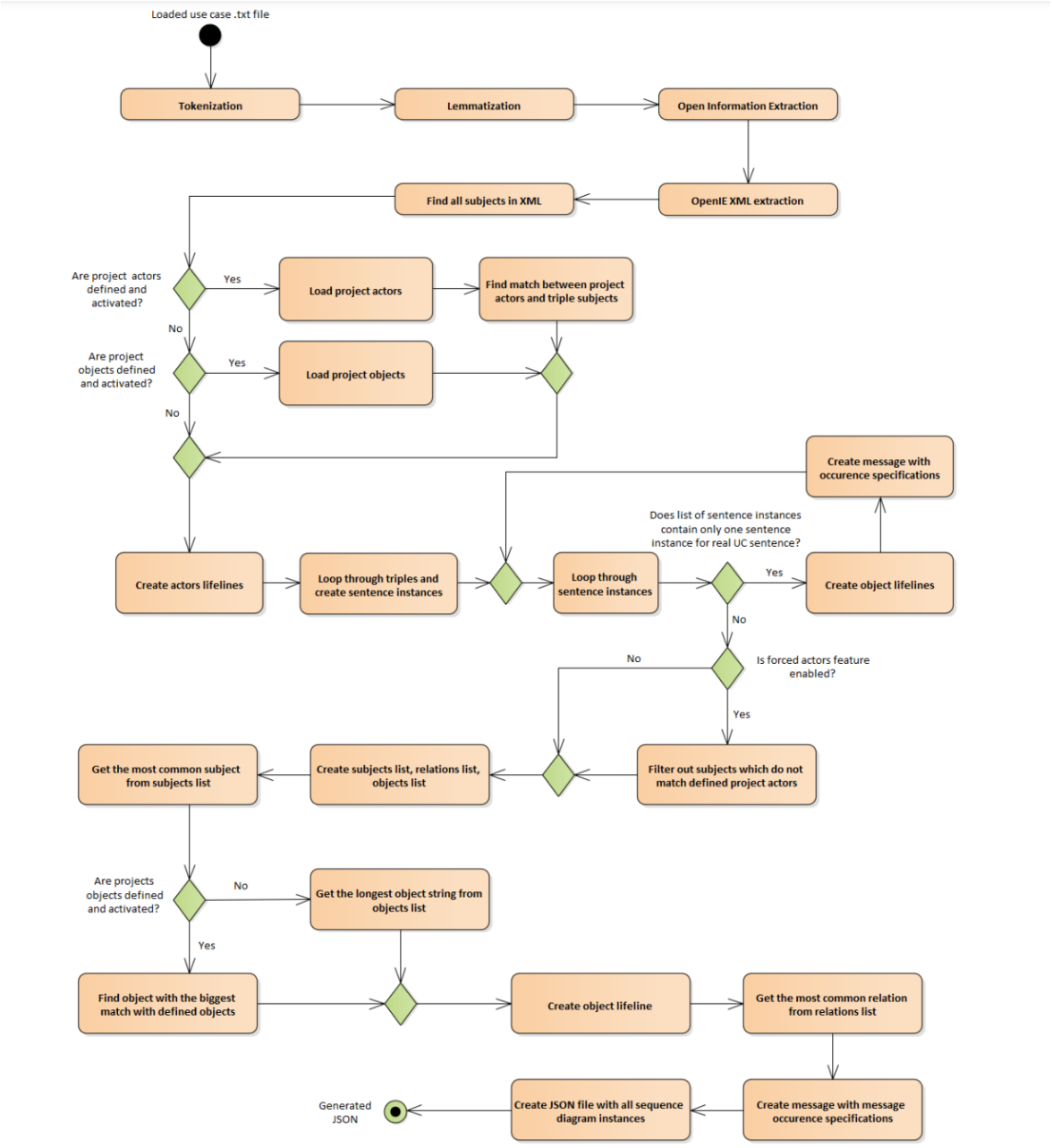
   (a) **Podmetový slovník**: Zaznamenáva sa v ňom podmet. Algoritmus postupne prejde všetky trojice, a porovnáva dĺžku uloženého podmetu s podmetom spracovávanej trojice. Ak je spracovávaný podmet dlhší ako ten už uložený v slovníku, uloží sa do slovníku tento dlhší podmet. Týmto sa zamedzí ukladaniu duplicít.

   (b) **Predmetový slovník**: Algoritmus vyzerá rovnako ako v prípade podmetového slovníka, len sa určí najdlhší predmet.

   (c) **Relačný slovník**: Algoritmus s vyberaním najdlhšieho elementu ako v prechádzajúcich prípadoch v tomto prípade generovalo chyby. Autori sa teda v tomto prípade rozhodli ukladať do projektového slovníka relácie, ktoré sa v trojiciach vyskytovali najčastejšie.

4. **Projektový aktéri a predmety**: Autori vytvorili ďalšie slovníky. Na rozdiel od predchádzajúcich slovníkov, ktoré sa generovali algoritmicky pri spracovávaní textu, tieto si vie užívateľ nastaviť v grafickom prostredí. Užívateľ si vie ku každej vete priradiť aktéra a objekt. Ak je aktér aspoň na 75% zhodný s akýmkoľvek aktérom, ktorý bol uložený v podmetovom slovníku, bude ako aktér použitý tento aktér. Taktiež ak je predmet zhodný v určitej autormi nešpecifikovanej miere s akýmkoľvek objektov predmetovom slovníku, bude použitý tento predmet.

5. **Vynútené použite definovaných projektových aktérov**: Pri niektorých komplexných vetách, vďaka nepresnosti NLP nástroja, sa generovali nesprávne trojice. Autori identifikovali ako problém nesprávne určenie aktérov. Pre tento dôvod bolo do riešenia pridané vynútené použite definovaných projektových aktérov.



Obr. 1.9: FBeIPo NLP pipeline[14]

1.3.10 Porovnanie existujúcich diplomových prác

Všetky spomínané práce majú ústrednú technológiu, ktorú používajú na prácu s prirodzeným jazykom. Táto technológia sa volá Standford NLP. Kým prvé práce využívajú

najmä prvky Part of Speech a rozlišovanie závislostí, najnovšia práca využila pokroku v tejto knižnici, a využila Öpen information exctraction", nástroj, ktorý autori predchádzajúcich prác nemal. V skutočnosti sa za Open Information Exctraction taktiež skrýva Part of Speech a rozlišovanie závislostí, toto však autori riešiť nemuseli.

Práca RBeIPo udala cestu a vytvorila prvú všeobecnú pipeline, ktorá využíva proces spracovania vetných kategórií, nájdenie koreferencií, a následne záverečné spracovanie výsledku. Autori viac či menej dokázali vytvoriť dané diagramy v pomere 1 veta z prípadu použitia ku 1 aktivite alebo správe. Líšili sa v možnosti ako spracovania súvetí, tak fragmentov. Možnosti, ktoré ponúkajú tieto práce, a my sme si ich vydedukovali z týchto prác sú v nasledovnom porovnaní. V porovnaní uvádzame len tie parametre, v ktorých sa aspoň jedna práca odlišuje:

[h]

	RBeIPo	SMoIPo	FBeIpo
Part of Speech	✓	✓	×
Vyhladávanie závislostí	✓	✓	×
Open Information Exctraction	×	×	✓
Fragmenty	×	×	✓
Vynútenie elementov v diagrame	×	×	✓
Projektový slovník	×	×	✓
Spracovanie súvetí	×	✓	×
WhiteList/BlackList	✓	✓	×

1.4 Technológie

V tejto podkapitole pomenujeme použité technológie v našej práci.

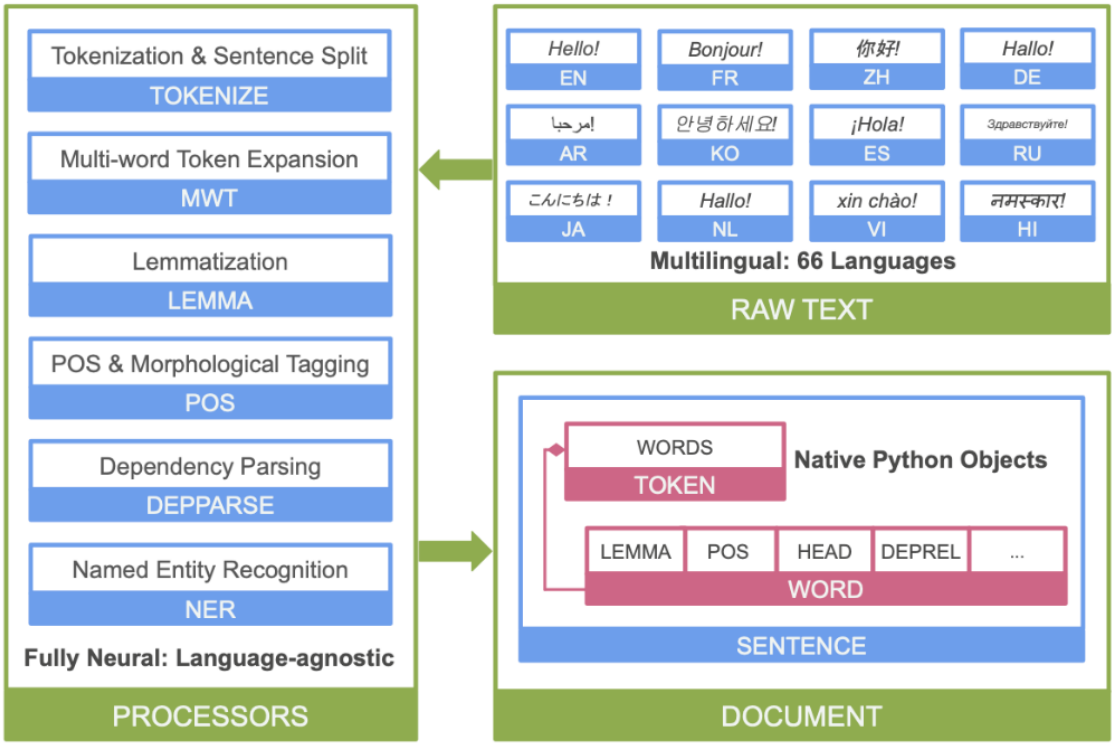
1.4.1 WordNet

Je rozsiahla databáza anglických slov vytvorená a udržiavaná na Princeton University. V tejto databáze sa slová zgrupujú do množín kognitívnych synoným, označujúcímy viacerými slovami jeden koncept. Medzi slovami v takýchto množinách existujú sémantické alebo lexikálne vzťahy. Takéto množiny sa nazývajú. tzv.synsety.[15] Takýchto množín obsahuje WordNet až cez 117 000.

WordNet využívame v spolupráci s NER, aby sme v nami spracovávaných vetách vedeli nájsť jednu entitu, pomenovávanú 2 rôznymi slovami, napr: vlčiak a pes. .

1.5 Stanza

Stanza je softvér napísaní v jazyku Python, ktorý poskytuje množstvo nástrojov na analýzu prirodzeného jazyka. Stanza umožnuje zostrojiť tzv."pipeline"na spracovanie textu. Takáto "pipeline"mení text na vektor slov, následne ich mení na tokeny a spracováva ich ďalej. Medzi pridateľné časti "pipeline"patria napr.: tokenizácia, viacslovná tokenizácia, lematizácia, "part of speech",rozoznávanie morfologických čŕt, parsovanie závislostí, a roz ˇpoznavanie pomenovaných entít. Stanza v súčastnosti dokáže pracovať s vyše 70 jazykmi. Stanza taktiež ponúka možnosti využívania Stanford CoreNLP klient skrz ním vystavený interface.



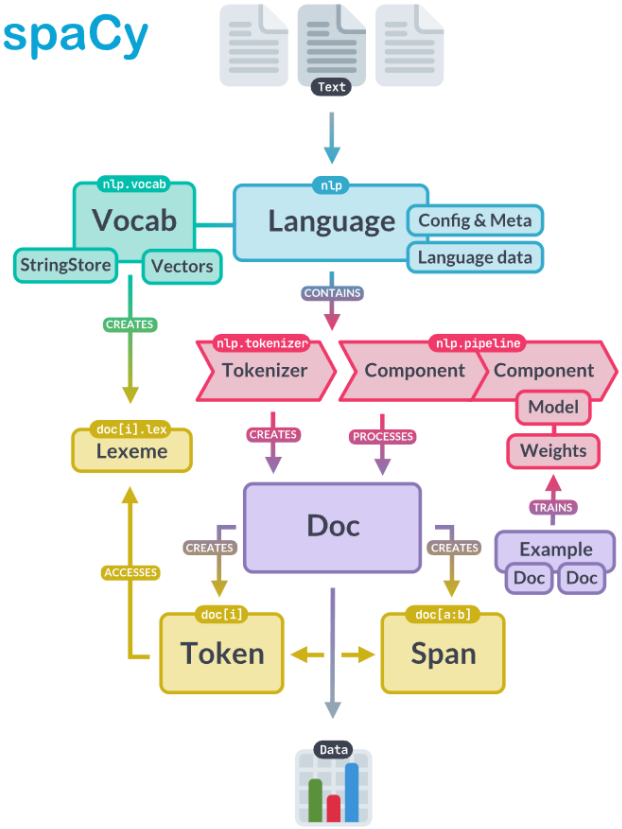
Obr. 1.10: Proces spracovanie textu softvérom Stanza

1.6 Spacy

Spacy je NLP knižnica pre Python, vytvorená v jazyku Cython pre lepší výkon. Knižnica je navrhnú tak, nech jej je použitie čo najjednoduchšie a najrýchlejšie. Spacy

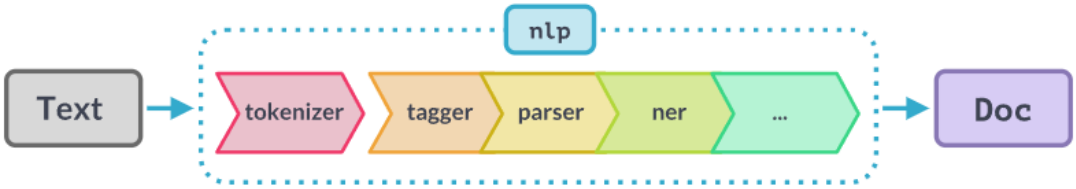
dokáže porozumieť veľkému množstvu textu za krátky čas. Podporuje cez 70 jazykov alebo transformery ako BERT. Ponúka rozpoznávanie pomenovaných entít, "part-of-speech"tagovanie, parsovanie závislostí, segmentáciu viet, klasifikáciu textu, lematizáciu, tokenizáciu, odstraňovanie stopových slov, morfologickú analýzu, prepájanie entít a mnoho ďalších nástrojov. Umožňuje vytváranie si vlastných komponentov do "pipelineä využívanie jazykových modelov tretích strán.

Architektúra Spacy[16] je postavená na tom, že sa prvotne spracuje text uloží do triedy Language. Pomocou tokenizácie sa text premení na tokeny. Táto časť nie je voliteľná a je to predpoklad pre správne fungovanie. Spracovaní text sa uloží do inštancií triedy Document, a tieto inštancie podliehajú ďalšiemu spracovaniu.



Obr. 1.11: Architektúra Spacy

Keď je už text tokenizovaný a uložený do dokumentov, môžeme následne ku pipeline pridávať ďalšie komponenty na spracovanie. Medzi ďalšie výhody patria aj možnosť pridať trénované váhy alebo pristúpiť ku modifikáciám založených na pravidlách.



Obr. 1.12: Príklad Spacy "Pipeline"

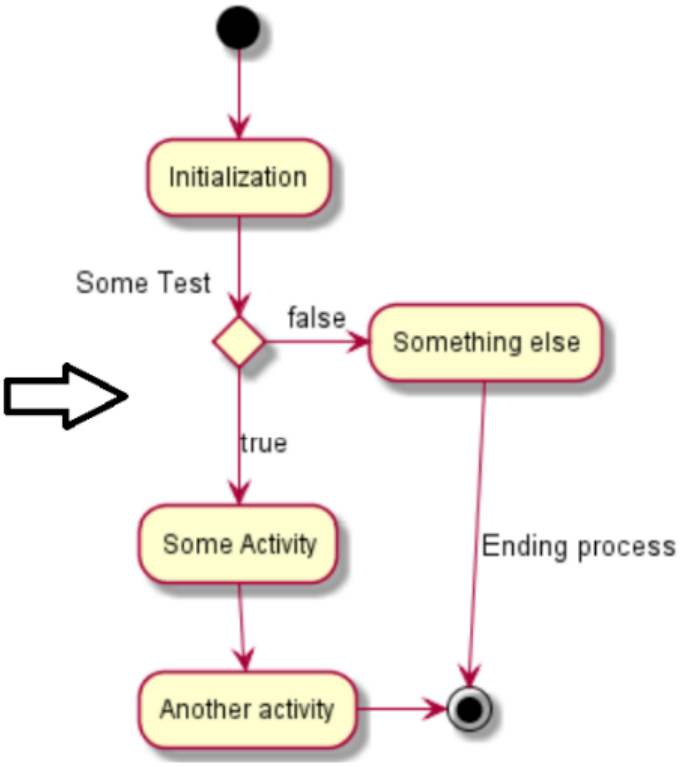
1.7 PlantUML

PlantUML je značkovací jazyk na tvorbu diagramov. Ponúka na výber širokú škálu diagramov. Umožňuje tvorbu nie len UML diagramov ale aj napr. Archimate diagramy a myšlienkove mapy. PlantUML ponúka vlastný značkovací jazyk, vďaka ktorému vieme vytvárať diagramy z textovej reprezentácie.

```
@startuml
(*) --> "Initialization"

if "Some Test" then
-->[true] "Some Activity"
--> "Another activity"
-right-> (*)
else
->[false] "Something else"
-->[Ending process] (*)
endif

@enduml
```



Obr. 1.13: Príklad tvorby PlantUML diagramu"

Kapitola 2

Výskum

V tejto kapitole sa zaoberáme výskumom a našim riešením problematiky tvorby sekvenčných diagramov z prípadov použitia.

2.1 Základne stanovenie problematiky

Zadefinovali sme si požiadavky, pojmy a stratégiu, ako pristúpiť ku riešenie problému.

2.1.1 Prekondície

Aby našej riešenie fungovalo, zadefinujeme si vstupne požiadavky prípadu použitia:

1. **Anglický jazyk**: Prípad použitia musí byť napísaný v anglickom jazyku, bez gramatických chýb.

2. **Očíslované riadky**: Jednotlive riadky musia byť očíslované a ukončené bodkou.

3. **Efektívny a štandardný prípad použitia**: Jednotlivé kroky v hlavnom toku musia byť napísane v prítomnom čase, byť stručné a smerovať k cieľu, obsahovať sloveso v činnom rode, veta sa musí začínať jedným z aktérov a prípad použitia by sa mal riadiť postupom ako písať efektívne prípady použitia.[3]. Jednotlivé kroky by mali byť tak atomické, aby sa považovali za samostatné transakcie.[3]

4. **SVO**: Vzor subject, verb, object(SVO), je vzor vety, ktorý budeme vyžadovať. Tento vzor vyžaduje aby veta začína predmetom, bola nasledovaná slovesom a podmetom. Medzi členmy sa taktiež môžu vyskytnúť rozvíjacie vetné členy. Ďalej v práci sa budeme na tento vzor referencovať ako na pomyselnú trojicu alebo triplet.

Príklad správnej vstupnej vety: 1. System displays a list of discount offers.

2.1.2 Transformácia prípadov použitia na sekvenčný diagram

Sekvenčný diagram sa dá pokladať za sekvenciu interakcií medzi dvomi komponentmi či už v systéme, alebo medzi externým aktérom a systémom. Jednotlivé interakcie pozostávajú z pomyselných trojíc:

1. Ten čo správu odosiela, budeme ho volať **podmet**.
2. Správy, teda interakcie medzi odosielateľom a prijmateľom, budeme ju volať **relácia**.
3. Ten čo správu príjma, budeme ho volať **predmet**.

Našou úlohou teda bolo spracovať text,zostaviť takéto trojice podmet-relácia-predmet, a tieto trojice pretransformovať na sekvenčný diagram, kde podmet sa ujme role čiary života, ktorá správu odosiela, relácia role správy a predmet role čiary života, ktorá správu obdrží. Ako príklad môžeme uviesť vetu :

System displays a list of discount offers. Z nej by sa mal vytvoriť triplet:

- **Podmet** - System,
- **Relácia** - displays,
- **Predmet** - list of discount offers.

2.1.3 Vzorový prípad použitia

Ako vzorový prípad použitia sme určili prípad použitia z práce FBeIPo[14], aby sme lepšie zvýraznili výsledky našej práce a mohli sme ich porovnať. Vzorový prípad použitia vyzerá:

1. System displays a list of discount offers.
2. Customer chooses a specific discount offer.
3. System displays the discount offer details.
4. Customer clicks on „Add to basket“ button.
5. System checks number of available discount offers.
6. System inserts discount offer to basket.

2.2 Prístup Stanza

Pre prvotné riešenie problematiky sme pristúpili

2.3 Prístup Spacy

2.4 Transformačná pipeline

Cieľom našej transformačnej pipeline nie je len čo najlepšia tranformácia, ale čo najlepšia konfigurovateľnosť. Naša transformačná pipeline pozostáva z typických aj menej typických častí transformačných pipeline.

2.4.1 Načítanie z textového súboru

Celý prípad použitia je načítaný z textového súboru, ošetrený na chybné vstupy a rozdelený na kroky.

2.4.2 Viacslovná tokenizácia

Viac slovná tokenizácia je proces, ktorý typicky nastupuje až po tokenizácií. Keď sa tokenizáciou vytvoria tokeny, viac slovná tokenizácia umožní spojiť skupiny tokenov tak, aby vystupovali ako jeden token.
V prípadoch použitia sa môže vyskytnúť scenár, keď je krok definovaný tak, že jeden z participantov je pomenovaný špecifickým názvom v úvodzovkách, napr. tlačidlo "Pridaj do košíka". Avšak NLP pipeline nedokáže z vety vyvodiť, že ide o špecifický názov nejakého objektu.
Vyvinuli sme teda algoritmus taký, že všetky takéto elementy s špecifickým názvom vieme nahradíme za iné podstatné meno, také ktoré nebolo použite v celom prípade použitia. Toto podstatné meno nezmení zmyseľ vety, lebo pipeline neskúma sémanticky zmysel vety, a gramatické kategórie a vlastnosti tohto slova zostanú nezmenené.

2.4.3 Tokenizácia

2.4.4 Lematizácia

2.4.5 Rozdelenie na trojice

Part of speech

Určovanie závislostí

Open information extraction

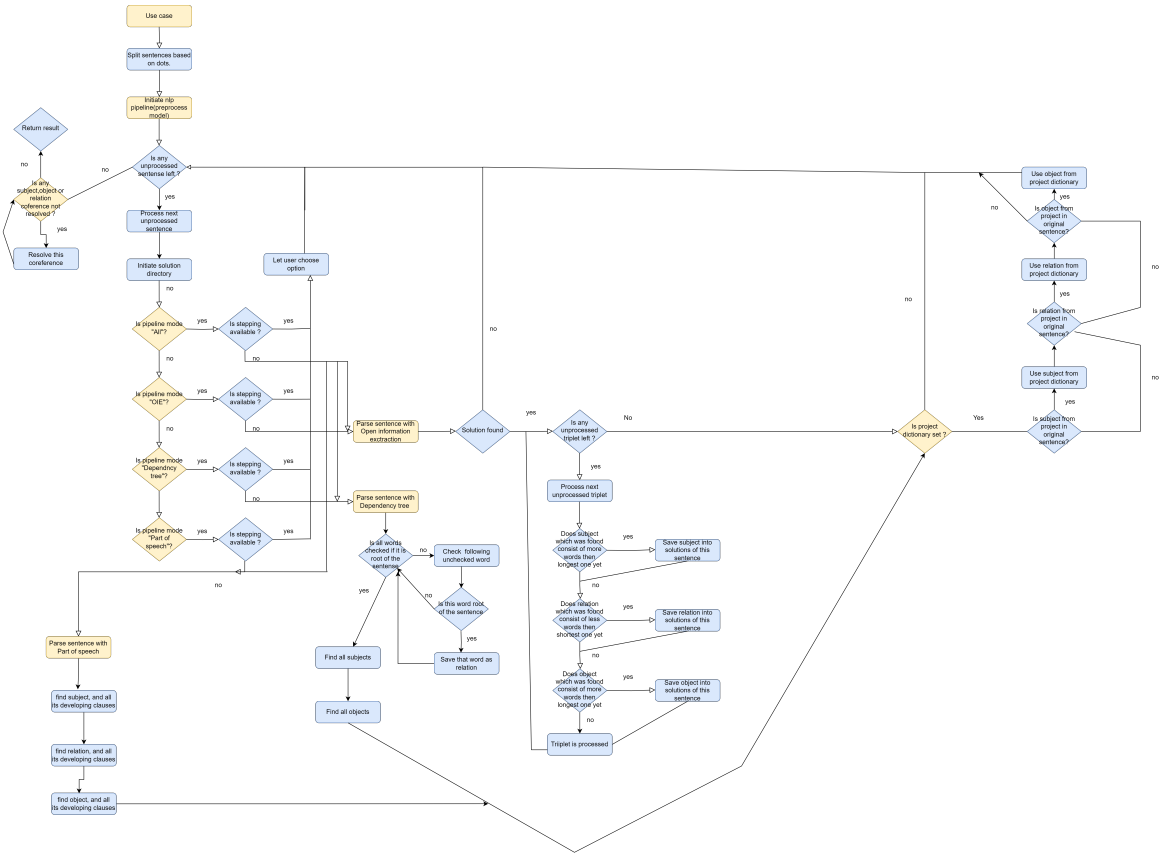
2.4.6 Určovanie koreferencií

2.4.7 Preddefinovaný projektový slovník

2.4.8 Krokovanie pipeline

2.5 Konfigurovateľnosť pipeline

2.6 Vizualizácia diagramov



Obr. 2.1: FZaLRa NLP pipeline

Kapitola 3

Evaluácia

V tejto kapitole vysvetlíme, ako sme evaluovali naše výstupy a porovnáme ich s evaluáciou existujúcich riešení.

3.1 Spôsob evaluácie

V existujúcich riešeniach na evaluácií využívali F1 skóre.

F1 skóre je spôsob štatistickej analytickej klasifikácie. Slúži na meranie presnosti evaluácie. F1 skóre sa počíta ako pomer počtu správnych odpovedí ku počtu všetkých odpovedí.

Zvolili sme si dve merítka, podľa ktorých budeme evaluovať naše výsledky.

3.2 Existujúce riešenia a ich evaluácia

3.3 Evaluácia nášho riešenia

3.3.1 Dataset

Pre náš dataset sme využili niekoľko rôznych zdrojov.

- ChatGPT: ChatGPT sme využili na vygenerovanie 10 prípadov použitia a ich transformáciu na sekvenčný diagram vo forme PlantUML.
- Autori práce: 5 prípadov použitia manuálne vytvorili autori práce a pretransformovali ich na sekvenčné diagramy.
- Študenti so zameraním na softvérové inžinierstvo: 5 prípadov použitia manuálne vytvorili oslovení študenti softvérového inžinierstva na Fakulte matematiky,fyziky

a informatiky Univerzity Komenského v Bratislave. Dané prípady použitia autori pretransformovali na sekvenčné diagramy.

- Autori predchádzajúcich prác - využili sme všetky uvádzané prípady použitia, ktoré využili autori predchádzajúcich diplomových prác. Tento prístup sme zvolili, aby sme dosiahnute výsledky mohli porovnať.

3.3.2 Evaluačny skript

3.3.3 Dosiahnuté výsledky

3.4 Porovnanie nášho riešenia s existujúcimi riešeniami

3.5 Budúca práca

Bibliografia

[1] Jim Arlow a Ila Neustadt. *UML 2 and the unified process: practical object-oriented analysis and design*. Pearson Education, 2005.

[2] Ivar Jacobson. *The unified software development process*. Pearson Education India, 1999.

[3] Alistair Cockburn. *Writing effective use cases*. Pearson Education India, 2001.

[4] KR1442 Chowdhary a KR Chowdhary. „Natural language processing“. In: *Fundamentals of artificial intelligence* (2020), s. 603–649.

[5] Vimala Balakrishnan a Ethel Lloyd-Yemoh. „Stemming and lemmatization: A comparison of retrieval performances“. In: (2014).

[6] Benoit Habert et al. „Towards tokenization evaluation.“ In: *Lrec*. 1998, s. 427–432.

[7] Tao Yue, Lionel C Briand a Yvan Labiche. „An Automated Approach to Transform Use Cases into Activity Diagrams.“ In: *ECMFA* 10 (2010), s. 337–353.

[8] Abdelsalam M. Maatuk a Esra A. Abdelnabi. „Generating uml use case and activity diagrams using nlp techniques and heuristics rules“. In: *International Conference on Data Science, E-learning and Information Systems 2021*. 2021, s. 271–277.

[9] Jitendra Singh Thakur a Atul Gupta. „Automatic generation of sequence diagram from use case specification“. In: *Proceedings of the 7th India Software Engineering Conference*. 2014, s. 1–6.

[10] Deva Kumar Deeptimahanti a Muhammad Ali Babar. „An automated tool for generating UML models from natural language requirements“. In: *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE. 2009, s. 680–682.

[11] Song Yang a Houari Sahraoui. „Towards automatically extracting UML class diagrams from natural language specifications“. In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 2022, s. 396–403.

[12] Richard Belan. *dvanced methods of analysis and design using multidimensional UML*. 2017.

[13] Štefan Motko. *Transformation of textual use cases into behavior UML diagram*. 2018.

[14] František Bekeš. *Supporting software design by sequence diagram generation and visualization from use cases*. 2020.

[15] George A Miller et al. „Introduction to WordNet: An on-line lexical database“. In: *International journal of lexicography* 3.4 (1990), s. 235–244.

[16] *Architektúra spacy, 2023,Dostupné online na: https://spacy.io/api[online].*