UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Agent Asistent Inteligent
Generarea Răspunsurilor în Limbaj Natural

Alexandra Mariana Cîrstian

Coordonator științific:

Conf. dr. Andrei Olaru

BUCUREȘTI

2018

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

Intelligent Virtual Assistant
Natural Language Generation of Replies

Alexandra Mariana Cîrstian

Thesis advisor:

Conf. dr. Andrei Olaru

BUCHAREST

2018

CUPRINS

1 Introducere **1**

2 Motivație **3**

3 Metode Existente **7**

4 Soluția Propusă **11**

4.1 Scenarii de utilizare 11

4.2 Descrierea soluției 13

4.3 API 16

5 Detalii de implementare **17**

5.1 Componenta de susținere a dialogului 17

5.1.1 Moderarea interacțiunii 17

5.1.2 Tratarea precizărilor 21

5.2 Componenta de translatare 22

5.2.1 Eliminarea nodurilor reziduale 22

5.2.2 Remodelarea grafului 23

5.2.3 Adăugarea articolelor 25

6 Evaluare **27**

7 Concluzii **30**

8 Anexa A: Rezultate produse de componenta de translatare **35**

SINOPSIS

Lucrarea propune un sistem pentru generarea limbajului natural și pentru moderarea interacțiunii între utilizator și întreaga aplicație. Generarea limbajului natural are la bază informații reprezentate prin grafuri. Sistemul tratează anumite tipuri de scenarii, în care cererile utilizatorilor trebuie să se încadreze. Rezultatul este acela că utilizatorul poate comunica o cerere în limbaj natural iar sistemul îi va putea răspunde într-o manieră similară.

ABSTRACT

The paper proposes a system for natural language generation and for mediating the interaction between the users and the whole application. The language generation is based on a graph representation of information. The system handles specific types of scenarios and the users' request must fit into one of these scenarios. As a result, the user can perform a request in natural language and the system will reply in the same manner.

MULȚUMIRI

Îi mulțumesc lui Andrei Olaru pentru îndrumare și pentru răbdarea de a-mi citi e-mailurile lungi, și Gabrielei Vlădescu pentru că mi-a răspuns la întrebările legate de proiectul ei și cum poate fi folosit.

1 INTRODUCERE

Tema proiectului o reprezintă generarea de limbaj natural pentru un agent asistent inteligent, pornind de la o reprezentare a cunoștințelor bazată pe grafuri și șabloane. Principalele atuuri ale proiectului sunt proprietățile sale de a fi open-source și a funcționa local, precum și modalitatea de stocare a informațiilor în cadrul sistemului: graful se dovedește a fi o structură de date suficient de flexibilă astfel încât să poată cuprinde toate informațiile necesare și să fie ușor interpretabilă atât din punctul de vedere al mașinii cât și din cel uman.

Proiectul își propune să contribuie la crearea unui agent asistent inteligent prin dezvoltarea unei componente de generare a limbajului natural, astfel încât agentul să poată comunica cu utilizatorul uman în limba engleză, atât pentru a oferi informațiile cerute de acesta cât și pentru a cere eventuale lămuriri, a confirma execuția sau a nara rezultatul unei anumite acțiuni realizate la cerere sau automat.

Obiectivele proiectului se referă la rezolvarea problemei fără a utiliza componente externe, care să necesite o conexiune la Internet. Se dorește acest lucru deoarece, urmând a fi folosit în cadrul unui agent asistent inteligent, sistemul de generare de limbaj natural va intra în contact cu informații private ale utilizatorului, iar lipsa transferului de date prin rețea va duce la o mai bună protecție a vieții sale private.

Pentru moment se dorește tratarea a două tipuri de scenarii în care cererile utilizatorului trebuie să se încadreze: scenarii cu cereri condiționale și scenarii cu cereri de informații. Scenariile cu cereri condiționale se referă la propoziții condiționale cu particula „if", cum ar fi „If Emily is at home, she will answer the door.". Scenariile cu cereri de informații pot fi la rândul lor de două tipuri: cu pronume interogativ, de tipul „What is a blue flower?" sau cu verb la modul imperativ, cum ar fi „Recommend me a good book.".

Soluția propusă oferă suport pentru cele două tipuri de scenarii propuse. În plus, există posibilitatea adăugării de precizări la cererea anterioară, fără a fi necesară repetarea întregii cereri. Sistemul este capabil să primească o cerere în limbaj natural de la utilizator și să ofere un răspuns bazat pe cunoștințele sale interne. De asemenea, utilizatorul poate introduce

informații în sistem în mod dinamic, tot prin intermediul unei cereri la care răspunsul va fi o simplă confirmare.

Componenta de interacțiune, recunoaștere a scenariilor și agregare a precizărilor la cereri funcționează conform cerințelor propuse. Componenta care se ocupă cu generarea limbajului natural pornind de la grafuri reușește să construiască propoziții corecte, cu sens, cuvinte de legătură, particule și articole pentru cereri ce nu formează grafuri cu cicluri și care se înscriu în cadrul unuia dintre scenariile tratate.

În capitolul 2 va fi prezentată motivația acestui proiect și motivele pentru care se dorește dezvoltarea unui agent asistent inteligent local și open-source. În capitolul 3 vor fi discutate diferite arhitecturi existente în cadrul sistemelor de generare de limbaj natural, precum și câteva exemple de abordări în rezolvarea problemelor specifice acestor sisteme. Capitolele 4 și 5 descriu soluția propusă de această lucrare, respectiv detaliile de implementare ale acesteia. În capitolul 6 se au în vedere rezultatele produse de soluția implementată, iar capitoul 7 discută concluziile și viitoarele dezvoltări posibile.

2 MOTIVAȚIE

Prezenta lucrare se înscrie în cadrul unui proiect mai amplu, al cărui scop este crearea unui asistent inteligent cu care se poate dialoga într-o formă simplificată a limbajului natural. Rolul asistentului este acela de a asista utilizatorul la îndeplinirea diferitelor sarcini în mediul digital sau de a realiza diverse acțiuni specificate de utilizator, cum ar fi, de exemplu, setarea unui memento sau a unei alarme. Pentru reprezentarea internă a cunoștințelor dobândite, asistentul va utiliza grafuri de context și șabloane, care pot fi prezentate utilizatorului la cerere, sub formă grafică sau textuală. Pentru moment se intenționează realizarea aplicației pentru sisteme desktop, însă pe viitor se are în vedere și o variantă pentru Android.

În lucrarea de față este adresată problema generării de text în limbaj natural de către asistent, ca răspuns la cererile utilizatorului, pornind de la un graf de cuvinte.

Motivația prezentei lucrări, cât și a întregului proiect și totodată principalele atuuri ale acestuia sunt reprezentate de faptul că este local și open-source, caracteristici neîntâlnite la alți asistenți inteligenți populari în prezent și a căror importanță urmează a fi discutată în următoarele paragrafe. Alexa [1], asistentul dezvoltat de Amazon, nu este open-source și utilizează Alexa Voice Services (AVS), servicii disponibile în cloud pentru recunoașterea și înțelegerea limbajului natural, așadar nu este nici local. În aceleași categorii se află și Google Assistant (Google), Cortana [2] (Microsoft) și Siri [3] (Apple), niciunul oferind acces la codul sursă și făcând procesare în cloud. Mycroft [4] este un asistent inteligent dezvoltat de Mycroft AI, Inc. ce este promovat ca o alternativă open-source pentru asistenții menționați anterior [6]. Îi lipsește însă proprietatea de a fi local din cauza componentei de recunoaștere și înțelegere a limbajului natural, care funcționează în cloud. Inițial această componentă era Cloud Speech-to-Text, dezvoltată de Google și utilizată și de Google Assistant, însă de la sfârșitul lui martie 2018 s-a trecut la utilizarea DeepSpeech [1], un proiect open-source dezvoltat de Mozilla.

Stocarea și procesarea locală a tuturor datelor și proceselor unei aplicații poate însemna că aceasta funcționează fără a avea nevoie de conexiune la internet sau, ca aspect negativ, că

[1] https://github.com/mozilla/DeepSpeech Ultima accesare: iunie 2018

este limitată de resursele fizice deținute de utilizator. Însă, din punct de vedere al proiectului aflat în discuție, cel mai notabil aspect al rulării locale îl reprezintă plusul de securitate și protecție a datelor oferit. Nu numai datele cu caracter personal, ci și alte informații ce țin de propria persoană (obiceiuri, zi de naștere, orașe în care a locuit, email-uri, etc.), deși pot părea inofensive pentru utilizatorul neexperimentat, atunci când sunt culese de la un eșantion de populație suficient de mare, pot fi folosite în scopuri negative sau imorale împotriva victimelor „donatoare" și nu numai [5].

Un prim exemplu de situație ce poate fi generată de lipsa de securitate a datelor aferente profilului personal și care susține importanța protecției datelor este mediatizatul scandal Facebook–Cambridge Analytica[2]. Compania britanică Cambridge Analytica a folosit datele a peste 50 de milioane de utilizatori Facebook, majoritatea cetățeni ai Statelor Unite ale Americii, cu scopul imoral de a influența votul în cadrul alegerilor prezidențiale din SUA în 2016.

Un alt exemplu este cel am companiei Equifax[3], care a anunțat în septembrie 2017 că a pierdut date personale (printre care date de naștere, adrese, numere de siguranță socială, numere de carnete de conducere) de la peste 143 de milioane de clienți din SUA. Pentru a obține informațiile criminalii au exploatat o vulnerabilitate a unei aplicații web, ceea ce susține faptul că o aplicație ce rulează doar local va fi mai sigură decât una conectată la rețea, deoarece vor exista mereu vulnerabilități și indivizi care să le exploateze.

Publicarea codului sursă al unei aplicații poate avea multiple beneficii, inclusiv în domeniul securității discutat anterior. Bruce Schneier [7] afirmă că orice are legătură cu securitatea trebuie să fie open-source. David A. Wheeler [8] vorbește despre motivele pentru care un software open-source poate deveni mai sigur decât unul closed-source, punând accentul pe faptul că un program ale cărui surse nu sunt făcute publice nu este nu nimic mai protejat de atacatori comparativ cu un program cu codul sursă public. Atacatorii folosesc în general tehnici pentru descoperirea vulnerabilităților ce nu au nevoie de codul sursă. De asemenea, există software care poate dezasambla sau decompila codul mașină pentru a reproduce codul sursă la un nivel suficient de avansat pentru ca atacatorii să poată căuta șabloane specifice vulnerabilităților, ceea ce face ascunderea codului sursă o măsură de securitate foarte ineficientă, chiar inutilă.

[2]https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election Ultima accesare: iunie 2018

[3]https://investor.equifax.com/news-and-events/news/2017/09-07-2017-213000628 Ultima accesare: iunie 2018

Pe de altă parte, un cod făcut public are mai multe șanse să devină, în timp, mult mai sigur datorită expunerii sale către revizii făcute de utilizatori și către îmbunătățirile ce pot fi făcute de aceștia într-un timp mult mai scurt de la descoperirea unei vulnerabilități.

Un alt mare avantaj al software-ului open-source este comunitatea ce se formează în jurul acestuia, comunitate care poate ajunge să contribuie permanent, la o dezvoltare de lungă durată a software-ului în jurul căruia s-a format. Printre cele mai notabile exemple în acest sens se află sistemul de operare Linux, un clasic exemplu de software open-source de succes. De asemenea, nucleul open-source al Linux este utilizat de și a dus la dezvoltarea Android, care, conform Net Market Share, este cel mai popular sistem de operare pentru telefoane mobile, fiind utilizat de 70% dintre acestea, și este al doilea cel mai popular pentru tablete, deținând 46 de procente, doar 9 mai puțin decât iOS. Aceste reușite au fost posibile datorită comunității care, începând cu anul 1991, când Linux a devenit open-source, a contribuit constant la dezvoltarea nucleului acestuia, ajungând în prezent la peste 1000 de contribuții săptămânale [4].

Un alt exemplu de proiect software a cărui comunitate a contribuit la succesul său este chiar Mycroft, menționat la începutul acestei secțiuni ca fiind o alternativă open-source pentru asistenții inteligenți dezvoltați de Apple, Microsoft, Google și Amazon. Principalul atu în acest sens îl reprezintă posibilitatea oricărui dezvoltator de a adăuga propriile abilități la nucleul Mycroft, putându-și proiecta astfel propriul asistent personalizat nevoilor sale, dincolo de abilitățile implicite ale acestuia, dar și posibilitatea de a împărtăși aceste noi abilități și eventuale îmbunătățiri cu restul comunității.

Comunitatea de dezvoltatori ce se formează în jurul proiectelor open-source poate avea o importanță majoră pentru succesul și evoluția produsului, ceea ce a determinat inclusiv companii precum Sun, Netscape Communications și Microsoft să includă produse ca Java [5], Netscape [6], respectiv .NET [7] pe lista software-ului open-source.

Un ultim punct de menționat ca avantaj pentru un asistent inteligent cu codul sursă deschis și

[4] https://github.com/torvalds/linux/pulse Ultima accesare: iunie 2018

[5] https://www.theregister.co.uk/2011/11/13/open_sourcing_java_five_year_anniversary Ultima accesare: iunie 2018

[6] https://www.cnet.com/news/netscape-sets-source-code-free Ultima accesare: iunie 2018

[7] https://www.cio.com/article/3026664/open-source-tools/the-real-reason-microsoft-open-sourced-net.html Ultima accesare: iunie 2018

pentru software-ul de acest tip în general este încrederea sporită a utilizatorilor în produs și în probabilitatea tot mai mică (cu cât trece mai mult timp de la publicarea codului) de a exista componente malițioase sau vulnerabilități care pot fi folosite în scopuri negative. Acest aspect este cu atât mai important cu cât un asistent va ajunge în timp să dețină o cuprinzătoare bază de date cu informații despre utilizator, informații ce trebuie să fie cât mai bine protejate. Borland InterBase [8] este un exemplu de software cu cod sursă închis care a funcționat cu un backdoor plasat intenționat timp de cel puțin 7 ani. Baza de date putea fi accesată și modificată de orice utilizator local sau prin rețea prin autentificarea folosind un nume și o parolă hardcodate în software. Această problemă a fost rezolvată târziu și numai după ce software-ul a devenit open-source și backdoor-ul a fost descoperit de utilizatori, la scurt timp după publicarea codului sursă.

[8] https://www.theregister.co.uk/2001/01/12/borland_interbase_backdoor_exposed Ultima accesare: iunie 2018

3 METODE EXISTENTE

Generarea limbajului natural (NLG - Natural Language Generation) formează, împreună cu înțelegerea limbajului natural (NLU - Natural Language Understanding) domeniul prelucrării limbajului natural, subdomeniu al inteligenței artificiale. NLG se referă la generarea de text în limbaj natural, inteligibil din punct de vedere uman, pornind de la formatul de reprezentare al cunoștințelor specific mașinii sau aplicației.

Perera și Nand [9] analizează patru tipuri de arhitecturi, specifice sistemelor de generare a limbajului natural dezvoltate în prezent:

- arhitecturi tip bandă de asamblare (pipeline)
- arhitecturi cu revizie (revision architectures)
- arhitecturi uniforme (uniform architectures)
- arhitecturi care aderă la principiile de design software

Arhitectura de tip bandă de asamblare este cea mai clasică abordare, fiind propusă de Dale și Reiter [10] în anul 2000. Cele trei etape specifice acestei arhitecturi pot fi observate în Figura 1.

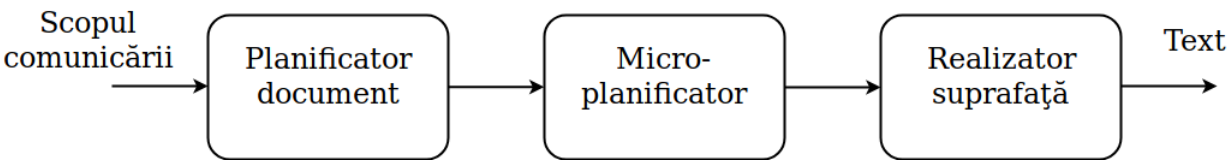

Figura 1: Cele 3 stagii ale arhitecturii pipeline [10]

Cele trei etape ilustrate în Figura 1 cuprind șase subetape ce sunt identificate și descrise de Dale și Reiter [10] ca fiind șase categorii principale de probleme care trebuie rezolvate într-un sistem de generare a limbajului natural, după cum urmează:

1. **Etapa de planificare a documentului** cuprinde două subetape: **determinarea conținutului** și **structurarea documentului**. Prima dintre acestea se referă la decizia cu privire la

ce informații vor fi incluse în text și ce informații vor fi omise, pe când structurarea documentului vizează organizarea datelor în text.

2. **Microplanificator** cuprinde 3 subetape: **lexicalizarea**, **agregarea** și **generarea expresiilor referențiale**. Lexicalizarea presupune alegerea efectivă a cuvintelor și expresiilor ce vor fi folosite pentru a comunica informația dorită. Agregarea se referă la împărțirea informației în propoziții iar generarea expresiilor referențiale implică determinarea proprietăților care pot fi folosite pentru a referi o anume entitate.
3. **Etapa de realizare a suprafeței** se referă atât la realizarea structurală cât și lingvistică și presupune determinarea modului în care conținutul de informații al textului este mapat în propoziții corecte din punct de vedere gramatical.

Dale, Geldof și Prost [11] au dezvoltat CORAL, un sistem care generează descrieri în limbaj natural ale rutelor obținute de la un sistem de calculare a traseelor. CORAL a fost creat urmând stagiile arhitecturii de tip bandă de asamblare și are ca date de intrare o serie de arce, echivalentul unor străzi, ce reprezintă drumul dintre două noduri, poziții de pe hartă. Rolul componentei de NLG este de a decide ce informații despre traseul generat sunt necesare pentru ca un utilizator uman să poată urma efectiv ruta și de a reda aceste informații într-o secvență de propoziții corecte și inteligibile, urmând stagiile descrise anterior.

Cu toate că arhitectura de tip bandă de asamblare are avantajul de a avea o execuție secvențială simplă, ușor de urmărit și de implementat și promovează modularitea, fapt benefic în special sistemelor NLG la scară largă, Perera și Nand [9] vorbesc și despre principalul dezavantaj al acestei arhitecturi, și anume faptul că fluxul datelor este unidirecțional, neoferind posibilitatea unei corectări sau îmbunătățiri a rezultatului.

Neajunsul arhitecturii tip bandă de asamblare este ceea ce a dus la evoluția ei spre cel de-al doilea tip menționat: arhitectura cu revizie. Acest tip de arhitecturi presupune o structură asemănătoare celei tip bandă de asamblare, însă rezolvă problema fluxului de date unidirecțional prin adăugarea componentelor de revizie, care pot fi de două tipuri: globale, la nivel de sistem NLG, sau per componentă/stagiu. Perera și Nand [9] afirmă despre a doua variantă că este mai larg utilizată atât datorită simplității sale cât și faptului că feedback-ul local oferit la fiecare stagiu redă mai multe informații utile fiecărui modul decât o face un feedback global.

Cu toate că majoritatea sistemelor NLG urmează una dintre puținele arhitecturi menționate, există o multitudine de abordări diferite la nivelul fiecărei componente NLG, variind de la învățare automată și recunoaștere de șabloane la metode bazate pe reguli, scheme sau structuri retorice.

Pentru etapa de determinare a conținutului, Kutlak, Mellish și Deemter [12] propun utilizarea unei euristici care să aproximeze cunoștințele comune, cunoștințe ce vor fi incluse în documentul final, pornind de la premisa că textul produs trebuie să conțină informații cunoscute utilizatorului, potrivite unei introduceri de articol.

Cimiano, Lüker, Nagel și Unger [13] au dezvoltat un sistem care urmează arhitectural clasică de tip bandă de asamblare, însă se diferențiază de alte abordări prin faptul că folosește un lexicon ontologic utilizat împreună cu structuri sintactice bazate pe informații statistice obținute din documente specifice domeniului vizat, pentru o lexicalizare cât mai potrivită contextului.

Wen, Gasic, Mrkšić, Su, Vandyke și Young [14] propun un model bazat pe rețele neuronale recurente cu unități de tipul LSTM (Long Short-Term Memory), propuse de Hochreiter și Schmidhuber [15] pentru prima dată în 1997. La fiecare pas intrarea celulei este reprezentată de codificarea unui cuvânt, iar ieșirea este o distribuție de probabilitate pentru următorul cuvânt. Acest proces este utilizat pentru stagiul de realizare a suprafeței. Pentru etapele de planificare se utilizează un alt tip de celule, cu funcție de tip sigmoid.

Lucrarea de față își propune să contribuie la realizarea unui sistem de generare de limbaj natural și dialog având la bază reprezentarea cunoștințelor și a contextului utilizator prin grafuri respectiv șabloane, așa cum a fost propus de Andrei Olaru [16]. Se va utiliza o arhitectură simplă, tip bandă de asamblare. Alte modalități de reprezentare a cunoștințelor includ reguli sau ontoligii, însă acestea trebuie predefinite manual. Avantajul grafurilor este atât acela că sunt ușor de implementat și vizualizat, cât și faptul că oferă mai multă flexibilitate față de alternativele menționate. De asemenea, acest tip de reprezentare a cunoștințelor oferă un suport sporit pentru producerea de limbaj natural, punând la dispoziție nu doar informațiile ce trebuie transmise utilizatorului uman, ci și informații prețioase pentru planificarea frazelor, cum ar fi cuvintele propriu-zise, indexul lor în propoziția originală, partea de vorbire și relațiile dintre concepte.

Componenta de NLU (înțelegerea limbajului natural), care face translatarea de la fraze în limba

engleză la reprezentarea prin șabloane de context este descrisă în lucrarea Gabrielei Vlădescu [17]. Șabloanele de context sunt colecții de noduri și muchii, asemănătoare grafurilor, ce prezintă lumea sub formă de entități a căror sens este înțeles prin perspectiva relațiilor cu alte entități.

Informațiile despre tipul părților de vorbire, indecșii cuvintelor și relațiile dintre cuvinte sunt obținute utilizând Stanford CoreNLP [18].

Având grafurile, ce reprezintă cunșotințele deja dobândite, și șabloanele, ce reprezintă cererea lansată de către utilizator, acestea pot fi folosite pentru generarea unui alt graf care să cuprindă o grupare formată din cerere și informațiile ce sunt regăsite ca răspund în baza de cunoștințe. Acest proces este realizat conform algoritmului de potrivire (matching) propus de Andrei Olaru [19] [20] și are ca rezultat grafuri ce vor reprezenta date de intrare pentru implementarea discutată în capitolele următoare.

4 SOLUȚIA PROPUSĂ

În Figura 2 este prezentată schema întregii aplicații. Aceasta oferă posibilitatea utilizatorului de a da comenzi vocale în limbaj natural, la care se va răspunde în același fel. Lucrarea de față tratează componentele reprezentate în schemă cu roz. Componenta verde este discutată de Matei Mihalea [21], cea mov de Gabriela Vlădescu [17], iar cea albastră de Andrei Olaru [19] [20].

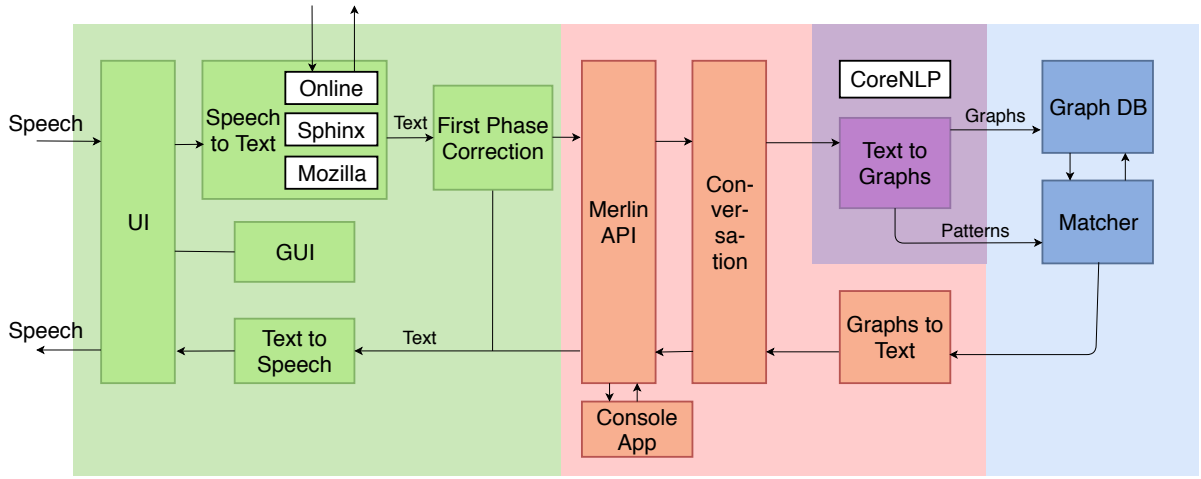

Figura 2: Schema aplicației

4.1 Scenarii de utilizare

Din cauza componentei utilizate pentru potrivirea cererilor de la utilizator cu informațiile deja deținute dar și din cauza complexității care ar fi survenit în cazul abordării oricărei cereri posibile în limbaj natural, aplicația a fost dezvoltată astfel încât să poată interacționa cu utilizatorul în cadrul a două tipuri de scenrii (cereri):

- condiționale (if)
- cerere de informații

Scenarii cu cereri condiționale

Scenariile cu cereri condiționale se referă la cererile din partea utilizator care au forma unor propoziții condiționale cu particula „if".

Un exemplu de o astfel de cerere este următoarea: „If Emily is at home, she will answer the front door." În cazul în care în baza de cunoștințe există un graf care să afirme „Emily is at home", răspunsul generat va fi „She will answer the front door.".

Scenarii cu cereri de informații

Scenariile cu cereri de informații se referă la cererile din partea utilizatorilor care folosesc pronume interogative (what, where, who, how, which) sau care folosesc un verb la modul imperativ pentru a cere o informație (de exemplu recommend, tell, say, etc).

Un exemplu de cerere cu pronume interogativ este: „What is a yellow fruit?". Răspunsul va fi de tipul „A pineapple is a strange yellow fruit.".

Un exemplu de cerere cu verb la modul imperativ este: „Recommend me an Italian restaurant", iar răspunsul va fi de tipul „Pinsere is an Italian restaurant".

Fiecare tip de scenariu acceptă și precizări adiționale. O precizare se referă la faptul că s-a realizat o cerere, de exemplu cea menționată anterior: „What is a yellow fruit?", la care s-a răspuns cu „A pineapple is a strange yellow fruit." În acest moment utilizatorul poate preciza: „A sour fruit.". Efectul acestei precizări este acela că șablonul rezultat va fi alipit la șablonul cererii anterioare, rezultând un șablon echivalent cu cel ce ar fi fost creat dacă s-ar fi făcut din prima cererea „What is a sour yellow fruit?". Această nouă cerere va primi un răspuns de tipul „A lemon is a sour yellow fruit.".

O ultimă situație posibilă ce a fost tratată este cea în care utilizatorul face o afirmație. Aceasta va fi primită tot sub forma unei cereri, însă în loc să i se caute un răspuns, aceasta va fi introdusă în baza de cunoștințe interne.

4.2 Descrierea soluției

Soluția propusă reunește două componente:

1. componenta ce translatează un graf într-un șir de caractere reprezentând o propoziție în limbaj natural
2. componenta ce susține dialogul

Pentru început va fi discutată componenta de translatare.

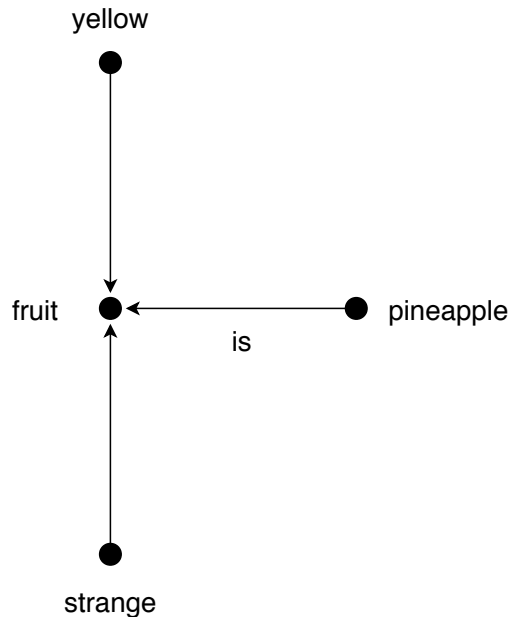

Figura 3: Exemplu de graf de intrare

Graful de intrare este o structură de tipul celei prezentate în Figura 3., fiecare nod reprezentând un cuvânt din enunțul sau enunțurile din care a fost obținut, iar fiecare arc stabilind o relație între cele două cuvinte de la capetele sale, uneori conținând și cuvinte de legătură, ca etichete ale arcului. Graful din Figura 3 a fost obținut prin parsarea propoziției „A pineapple is a strange yellow fruit.". Prin translatarea grafului se va obține un enunț identic cu cel inițial.

Soluția implementată urmărește o arhitectură clasică tip bandă de asamblare, datorată ordinii naturale de aplicare a operațiilor pe datele de intrare pentru obținerea ieșirilor, cu toate că, din punct de vedere al dezvoltării, operațiile nu au fost implementate în ordinea exactă a aplicării finale.

Datorită faptului că datele de intrare conțin deja cuvintele necesare pentru exprimarea informației

dorite, inclusiv gruparea în propoziții sau enunțuri, procesul a fost simplificat, după cum urmează să fie prezentat.

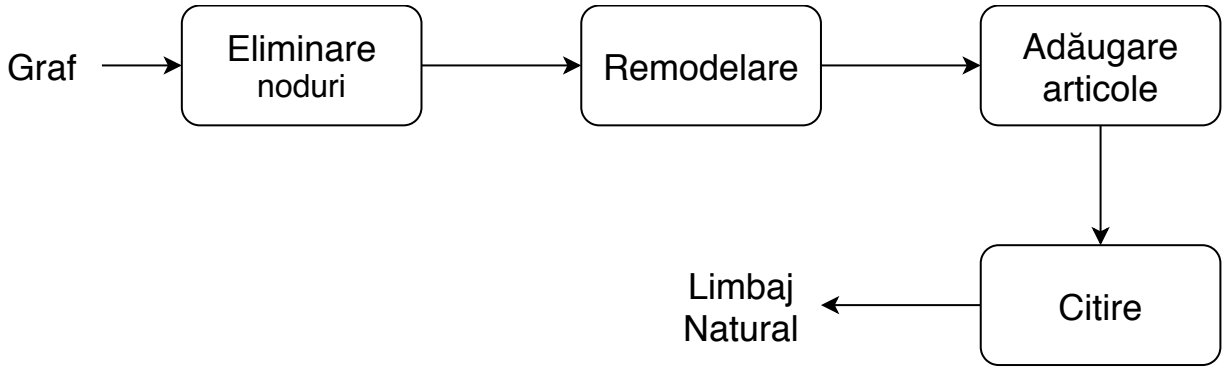

Figura 4: Fluxul de execuție al aplicației

Deoarece graful de intrare reprezintă deja o selecție de informații, am decis să încep dezvoltarea cu stagiul a doilea, prezentat în Figura 4 ca „remodelarea" grafului. Acest stagiu este cel mai laborios din punct de vedere al implementării și totodată cel mai important și cu cel mai mare efect asupra rezultatului final.

Remodelarea grafului se referă la modificarea structurii grafului fără a modifica mulțimea de elemente (noduri și arce) ale acestuia. Remodelarea cuprinde două etape:

1. Reorientarea arcelor
2. Modificarea topologiei

Reorientarea uni arc, fără a schimba cele două noduri de la capetele sale, se referă la redirecționarea săgeții astfel încât aceasta să se îndrepte dinspre nodul al cărui cuvânt va apărea în propoziție la un index mai mic, către nodul al cărui cuvânt va apărea la un index mai mare.

Modificarea topologiei se referă la reatașarea arcelor din graf astfel încât să se păstreze ordinea cuvintelor stabilită la pasul anterior și în același timp să se realizeze o ordonare la nivelul întregului graf, păstrându-se relațiile dintre noduri sub forma arcelor. Rezultatul acestui proces va fi un graf ce va avea forma unei liste simplu înlănțuite. O parcurgere începând din rădăcină (primul nod al listei) va însemna o parcurgere a listei de cuvinte în ordinea lor naturală în care vor apărea în rezultatul final.

În Figura 5 se poate observa efectul remodelării asupra grafului prezentat anterior, în Figura 3.

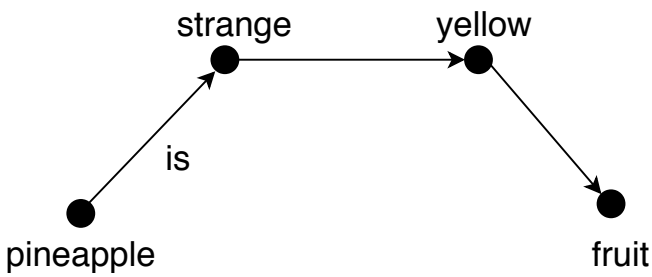

Figura 5: Efectul etapei de remodelare asupra grafului din Figura 3

Următorul stagiu dezvoltat a fost cel de citire a grafului, care permite vizualizarea rezultatelor finale, precum și a rezultatelor intermediare pe parcursul dezvoltării diferitelor stagii. Citirea grafului se referă efectiv la parcurgerea grafului-listă și stocarea tuturor etichetelor din noduri și arce, în ordinea parcurgerii, într-un singur șir de caractere ce reprezintă propoziția în limbaj natural. Aplicarea citirii pe graful prezentat în Figura 5 va rezulta în următorul șir: pineapple is strange yellow fruit.

Un aspect care nu se observă în grafurile din Figura 3 sau Figura 5 dar care va fi discutat și exemplificat în capitolul următor este acela că grafurile de intrare pot conține și noduri reziduale, care vin din șablonul creat din cererea utilizatorului și care nu sunt dorite în răspuns. Rolul primului stagiu prezentat în Figura 4, eliminarea nodurilor, se referă la aceste noduri reziduale, care vor fi șterse din graful de intrare înainte de a aplica orice altă operație pe acesta.

Cuvintele de legătură precum prepozițiile sau verbul copulativ „to be" nu apar ca noduri în graf, ele nereprezentând concepte, însă sunt păstrate ca etichete ale arcelor, de unde sunt și restaurate în stagiul de citire. Ceea ce nu se păstrează în graf sunt articolele, după cum se poate observa și în Figura 5. Acestea sunt păstrate însă pe arcele șabloanelor, proprietate utiliztă în cadrul stagiului de adăugare a articolelor.

Toate stagiile discutate aparțin componentei de translatare a unui graf în limbaj natural, care este utilizată în cadrul componentei de susținere a dialogului.

În soluția propusă, componenta de susținere a dialogului este una simplă, care nu poate servi unei conversații. Rolul acesteia este de a modera interacțiunea între utilizator și componenta de translatare a grafurilor, oferind și posibilitatea de a adăuga precizări la cererea făcută anterior sau de a introduce informații în sistem în mod dinamic. Mai multe detalii vor fi oferite în cadrul capitolului următor.

4.3 API

Ambele componente prezentate sunt înglobate în clasa Java numită Merlin. Aceasta oferă o interfață simplă și intuitivă care cuprinde un constructor și o metodă pentru introducerea cererii utilizatorului în sistem. Ambele antete sunt prezentate în Listarea 4.1.

```
1 public Merlin(String theFileOfWisdom);
2 public String askMerlin(String query) throws Exception;
```

Listarea 4.1: API-ul Merlin

Constructorul primește ca argument calea către un fișier ce conține informațiile în limbaj natural ce vor fi introduse în baza de cunoștințe. Metoda pentru lansarea unei cereri poartă numele askMerlin și primește ca argument cererea în limbaj natural sub formă de șir de caractere. Această metodă returnează răspunsul cererii în limbaj natural, tot sub formă de șir de caractere. Un exemplu de utilizare se poate vedea în Listarea 4.2.

```
1 Merlin m = new Merlin("/home/alexandra/sertaras/licenta/graphs.txt");
2 String query = "Recommend me a good book.";
3 String reply = m.askMerlin(query);
```

Listarea 4.2: Exemplu de utilizare al API-ului

5 DETALII DE IMPLEMENTARE

Sistemul dezvoltat cuprinde două componente: componenta de susținere a dialogului și componenta de translatare a grafurilor.

5.1 Componenta de susținere a dialogului

5.1.1 Moderarea interacțiunii

Rolul principal al componentei de susținere a dialogului este acela de a modera interacțiunea dintre utilizator, componenta de înțelegere a limbajului natural (NLU) și componenta de translatare a grafurilor (NLG).

Primul pas realizat în cadrul acestei componente se execută chiar la instanțiere și îl reprezintă extragerea informațiilor interne dintr-un fișier dat ca parametru și traducerea lor în grafuri. Translatarea informațiilor, date în limbaj natural, în grafuri se face prin intermediul componentei de NLU, descrisă în lucrarea Gabrielei Vlădescu [17].

```
1  SirCaractere askMerlin(SirCaractere cerere) {
2      userReq = nou UserRequest(cerere)
3
4      daca userReq este de tip precizare:
5          alipeste la userRequest cererea anterioara
6      raspuns = userReq.calculeazaRaspuns()
7      daca raspuns este vid:
8          returneaza replica predefinita
9
10     returneaza raspuns
11 }
```

Listarea 5.1: Funcția principală a componentei de dialog

În Listarea 5.1 este prezentat pseudocodul funcției principale a componentei de dialog.

Componenta de dialog este implementată de clasa numită Merlin și utilizează un alt obiect, de tip UserRequest, pentru a îngloba procesul de construire a grafului de intrare pentru componenta de translatare. O instanță de UserRequest reține **șirul de caractere al cererii**, **tipul** acesteia și **șablonul** corespunzător ei.

Șabloanele sunt un tip de graf obținut prin parsarea unei cereri de la utilizator primită sub formă de șir de caractere. În Figura 6 se poate observa un exemplu de șablon extras din propoziția „Recommend me an Italian restaurant.".

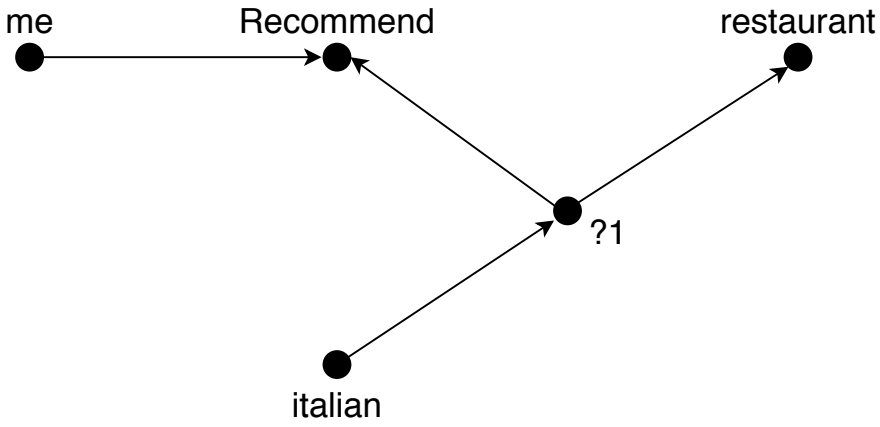

Figura 6: Exemplu de șablon

Șabloanele se aseamănă grafurilor din punct de vedere al structurii. Diferența constă în faptul că șabloanele permit două tipuri de noduri:

- **noduri concept**, care pot reprezenta un tip de obiect, o acțiune, o proprietate, etc.
- **noduri de instanță**, care ajung să fie create în cazul cuvintelor cu determinanți (cum ar fi articolul „the" în limba engleză) datorită faptului că proprietățile acestora trebuie atribuite unei instanțe anume a unui concept, și nu nodului concept în sine

Nodurile concept sunt cele întâlnite și în grafurile ce formează baza de cunoștințe, însă nodurile de instanță, sau generice, sunt cele specifice șabloanelor. Nodurile de instanță nu sunt identificare printr-o etichetă ce conține un cuvânt, precum nodurile concept, ci printr-un index. Din punct de vedere grafic, ele sunt marcate print-un semn de întrebare alăturat indexului. Așa cum se poate observa, în șablonul din Figura 6 există un singur nod generic, marcat cu „?1".

Calcularea tipului cererii presupune încadrarea acesteia într-unul dintre tipurile de scenarii discutate în capitolul anterior: scenarii cu cereri condiționale (if), scenarii cu cerere de informații sau precizări.

Pentru identificarea scenariilor cu cereri condiționale se va încerca identificarea particulei „if" în cererea utilizatorului. În caz de succes, cererea aparține acestui tip de scenariu.

Distincția între scenariile cu cerere de informații și precizări se face pe baza identificării unui pronume interogativ sau a unui verb la modul imperativ în conținutul cererii, ceea ce relevă un scenariu cu cerere de informații. Pentru identificarea părților de vorbire se utilizează informațiile generate de Stanford CoreNLP [18] și păstrate în nodurile grafurilor și șabloanelor. Un pronume interogativ este marcat cu WP (what), WRB (where, how) sau WDT (which), pe când un verb la forma de bază (ceea ce include atât modul imperativ, cât și infinitiv și conjunctiv) este notat cu VB.

În momentul cererii unui răspuns de la un obiect de tip UserRequest, altfel spus în momentul în care utilizatorul dorește răspunsul la cererea tocmai lansată, se va executa o secvență de operații descrisă în pseudocod în Listarea 5.2.

```
1  SirCaractere calculeazaRaspuns(Sablon s) { // s este sablonul corespunzator
2                                             // cererii utilizatorului
3      Graf k = graful din baza de cunostinte care
4               contine informatia ceruta de s
5      daca k este null:
6          returneaza sir vid
7
8      Graf g = agregare intre s si k
9
10     eliminareNoduriReziduale(g)
11     remodelare(g)
12     adaugareArticole(g)
13
14     returneaza citire(g)
15 }
```

Listarea 5.2: Secvența de operații executată la cererea unui răspuns

Se observă faptul că secvența de operații aplicată pe graful g în Listarea 5.2 coincide celor partu stagii ale componentei de translatare a grafului, ale căror funcționalități au fost descrise în capitolul precedent. Astfel, la acest nivel se realizează îmbinarea între cele două componente discutate în cadrul lucrării.

Graful de intrare al componentei de translatare reprezintă o agregare între un graf și un șablon, mai exact o combinație între o cerere formulată de utilizator în limbaj natural și translatată într-un șablon și o informație primită tot în limbaj natural și stocată intern sub formă de graf.

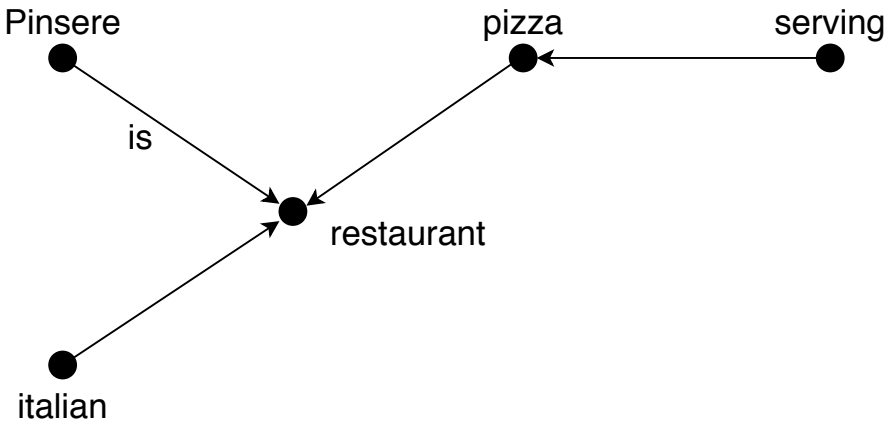

Figura 7: Graf din baza de cunoștințe

În Figura 8 se poate observa un exemplu de agregare între un graf și un șablon ce poate reprezenta o intrare pentru componenta de translatare. Graful din Figura 8 a fost obținut prin agregarea șablonului corespunzător cererii „Recommend me an Italian restaurant", exemplificat în Figura 6, și a grafului din baza de cunoștințe din Figura 7.

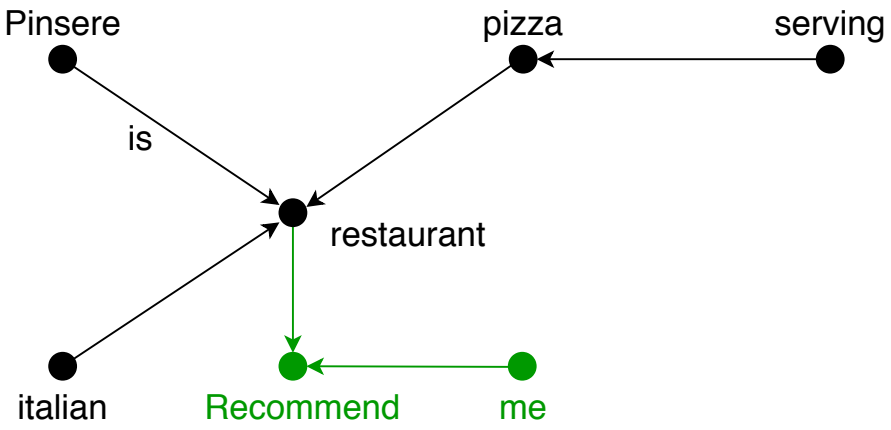

Figura 8: Agregare de graf și șablon

5.1.2 Tratarea precizărilor

Precizările se referă la intrări din partea utilizatorului care nu sunt cereri efective, ci vin în completarea unei cereri anterioare. Un exemplu este cel în care utilizatorul introduce cererea „Recommend me an Italian restaurant.", la care primește ca răspuns „Pinsere is an Italian restaurant serving pizza.". Nemulțumit de răspunsul primit, utilizatorul poate preciza: „A restaurant serving pasta.". În acest caz, șablonul precizării va fi combinat cu cel al cererii anterioare, pentru a obține o cerere mai detaliată, echivalentă cu „Recommend me an Italian restaurant serving pasta."

Algoritmul pentru agregarea celor două șabloane, al precizării și al cererii, este descris de pseudocodul din Listarea 5.3.

```
1  Sablon agregaCerere(Sablon p) {          // p este sablonul
2                                           // corespunzator precizarii
3      Sablon s = sablonul cererii precedente
4      daca s nu exista sau s si p nu au noduri comune:
5          returneaza null
6
7      pentru_fiecare arc(src, dst) din s:
8          daca src nu exista in p:
9              adauga nodul src in p
10         daca dst nu exista in p:
11             adauga nodul dst in p
12         daca nu exista arc(src, dst) in p:
13             adauga arc(src, dst) in p
14
15     returneaza p
16 }
```

Listarea 5.3: Algoritmul de agregare a două șabloane

5.2 Componenta de translatare

În această secțiune vor fi abordate, pe rând, fiecare dintre primele 3 stagii ale componentei de translatare. Ultimul stagiu, citirea, nu este unul suficient de complex, fiind deja descris în totalitate în cadrul capitolului 4.

Următoarele secțiuni descriu stagiile în ordinea aplicării lor, rezultatul unuia reprezentând datele de intrare pentru următorul.

5.2.1 Eliminarea nodurilor reziduale

Procesul de eliminare a nodurilor reziduale depinde de tipul cererii utilizatorului, tip ce determină categoriile de noduri ce vor fi excluse din graful rezultant. Intrarea pentru acest proces este reprezentată de agregarea dintre un graf și un șablon, așa cum a fost prezentat la finalul secțiunii 5.1.1.

În cazul scenariilor cu cereri de informații, mulțimea de noduri ce vor fi eliminate este formată din nodurile care se află în șablon dar nu și în graf. În Figura 8 se pot observa cu negru nodurile provenind din graful din baza de cunoștințe, iar colorate cu verde sunt nodurile ce provin exclusiv din șablon. Se poate observa cu ușurință faptul că nodurile colorate în verde sunt cele ce nu sunt dorite în răspunsul la cererea „Recommend me an Italian restaurant."

În cazul scenariilor cu cereri condiționale situația este diferită deoarece răspunsul așteptat nu se află în grafurile din baza de cunoștințe, ci chiar în șablonul generat din cererea utilizatorului. Un exemplu este cererea „If Emily is at home, she will answer the front door.". La primirea unei astfel de cereri se caută un graf care să afirme „Emily is at home". O dată stabilit că această afirmație este adevărată, răspunsul dorit este „She will answer the front door.". De aici rezultă faptul că nodurile redundante sunt cele care se regăsesc atât în graf, cât și în șablon.

În Figura 9 se poate observa graful agregat al exemplului discutat anterior. Contrar scenariilor cu cereri de informații, în acest caz nodurile ce trebuie eliminate sunt cele negre, iar cele păstrate vor fi cele ce aparțin exclusiv șablonului, adică cele verzi.

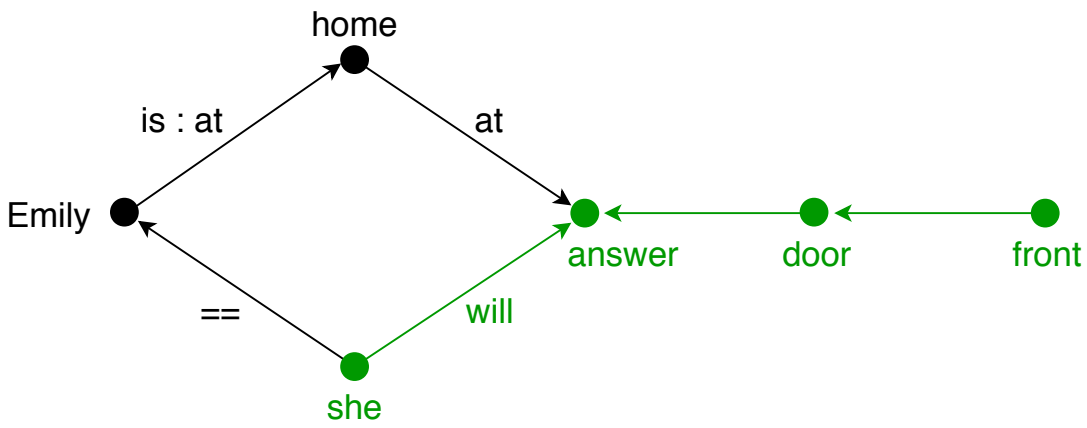

Figura 9: Exemplu de agregare pentru un scenariu cu cerere condiţională

Algoritmul de eliminare a nodurilor alcătuieşte în primul rând lista nodurlor reziduale, după care face o parcurgere a acesteia pentru a şterge atât nodurile, cât şi arcele asociate.

5.2.2 Remodelarea grafului

Remodelarea grafului presupune două operaţii ce vor fi descrise pe rând în cele ce urmează. Rezultatul executării celor două procese este un graf de forma unei liste simplu înlănţuite, dar care conţine acelaşi număr de elemente şi aceleaşi etichete precum graful primit ca dată de intrare.

Reorientarea arcelor

Această operaţie se referă la reorientarea fiecărui arc astfel încât să fie direcţionat de la nodul al cărui cuvânt va apărea în propoziţia finală la un index mai mic, către nodul al cărui cuvânt va avea un index mai mare.

Iniţial am dorit stabilirea ordinii cuvintelor pe baza arcelor grafului şi a tipului părţii de vorbire de care aparţin. Am dorit această abordare datorită flexibilităţii sale şi independenţei faţă de propoziţia sau propoziţiile efective din care a fost creat graful. Totuşi, cu excepţia câtorva reguli gramaticale simple, cum ar fi aceea că în limba engleză adjectivul va fi mereu înaintea substantivului pe care îl determină, nu am putut observa şi crea un set de reguli care să stabilească ordinea corectă a tuturor cuvintelor, astfel că am căutat o altă abordare.

Soluţia utilizată pentru ordonarea cuvintelor se foloseşte de indecşii cuvintelor în propoziţia

originală, aceștia fiind păstrați ca informații atât în grafuri cât și în șabloane. Cu toate că nu este o soluție la fel de flexibilă precum cea descrisă în paragraful anterior, totuși este una care a putut fi implementată și care, în acest stadiu al proiectului, este suficientă. Soluția nu poate funcționa corect pe grafuri agregate care cuprind cuvinte din mai multe propoziții, însă în stadiul curent nu există un astfel de scenariu.

Modificarea topologiei

Această operație se referă la repoziționarea arcelor, fără a modifica numărul acestora (decât în cazul în care graful de intrare are cicluri) astfel încât fiecare nod să aibă atât gradul interior cât și cel exterior cel mult egal cu 1.

Înainte de a continua procesul, trebuie asigurat faptul că graful nu conține cicluri, deoarece algoritmul de modificare a topologiei nu va funcționa pe un astfel de graf.

Modificarea topologiei grafului se realizează în două etape, ambele plecând de la o sortare topologică. Se pornește de la nodurile rădăcină, care au gradul interior egal cu 0 și la fiecare pas se adaugă în lista sortată toate nodurile încă nesortate care au toți părinții (nodurile de care sunt legate printr-un arc interior) deja sortați. În Figura 10 se poate observa un graf cu două sortări topologice posibile: Pinsere, Italian, restaurant, serving, pizza, respectiv Italian, Pinsere, restaurant, serving, pizza.

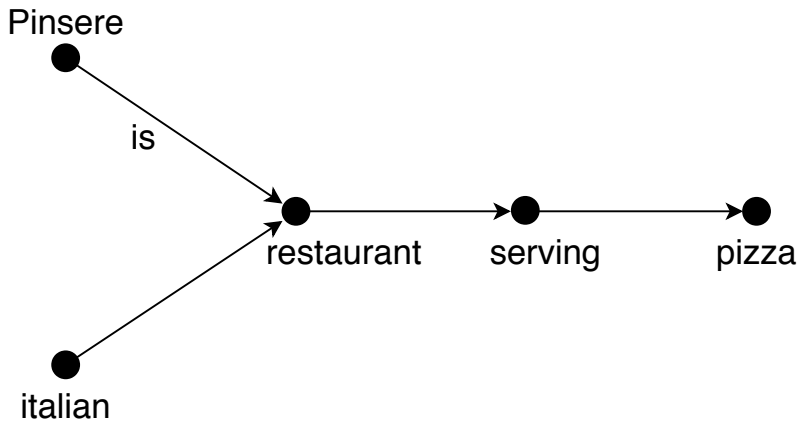

Figura 10: Exemplu de graf cu două sortări topologice posibile

Având lista nodurilor sortate topologic, acestea sunt parcurse pe rând și "aplatizate". Acest procedeu se aplică în două etape, după cum urmează a fi descris.

În prima etapă, sunt repoziționate arcele interioare ale tuturor nodurilor care au gradul interior mai mare decât 1. În exemplul din Figura 10 „restaurant" este un astfel de nod, având două arce interioare (doi părinți): cel dintre „Pinsere" și cel dinspre „Italian". În cazul în care nodurile părinte au la rândul lor părinți, se parcurge lanțul până se întâlnește o rădăcină, care va înlocui în operațiile ce urmează părintele direct, cu scopul de a păstra lanțurile de cuvinte deja create corect și a nu intercala cuvintele în mod eronat. Toate nodurile părinte sunt sortate după indecșii cuvintelor. Pentru n noduri părinte sortate, arcele primilor n-1 părinți direcți sunt redirecționate către următorul părinte-rădăcină din lista sortată. În Figura 11 se poate observa efectul acestei operații pe graful din Figura 10.

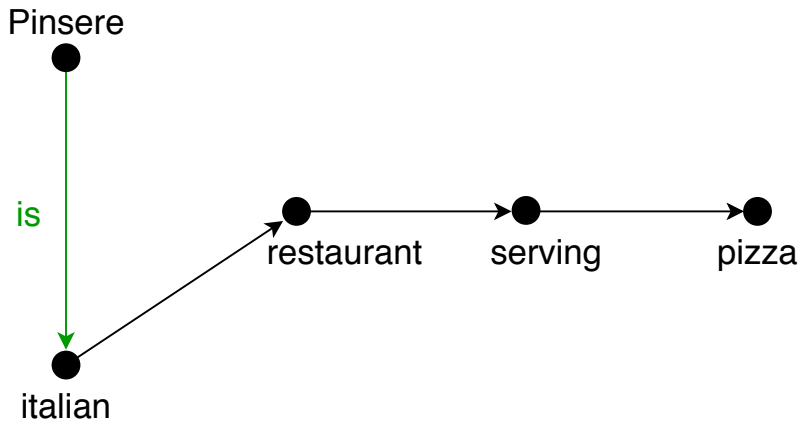

Figura 11: Efectul etapei de modificare a topologiei asupra grafului din Figura 10

A doua etapă presupune același algoritm descris anterior, aplicat însă pe sortarea topologică inversă, pentru nodurile cu gradul exterior mai mare decât 1.

5.2.3 Adăugarea articolelor

Procesul de adăugare a articolelor presupune două etape: extragerea din șablon a articolului asociat fiecărui substantiv și plasarea lui în graful-listă rezultat în urma stagiului de remodelare.

Pentru extragerea articolului din șablon se caută pur și simplu în șablon nodul a cărui etichetă corespunde substantivului dorit și se observă dacă există un arc între acesta și un nod generic. Dacă există, articolul căutat se va afla ca etichetă pe acel arc. În caz contrar, substantivul apare fără articol.

Pentru plasarea articolului în poziția corectă, se parcurge înapoi lanțul de adjective al substantivului în cauză și articolul este asociat muchiei interioare a ultimului adjectiv din lanț, sau

a substantivului, dacă lanțul de adjective nu există. În această etapă se realizează și acordul articolului, înainte de plasarea sa în cadrul muchiei.

În Figura 12 se poate observa, evidențiat cu verde, efectul adăugării articolelor pentru graful din Figura 11. În acest caz substantivul „restaurant" este singurul articulat, având asociat articolul nehotărât „a". Pe lanțul său de adjective este găsit doar nodul „Italian". Articolul este acordat cu adjectivul „Italian" și devine „an" înainte de a fi plasat pe muchia interioară a nodului. Privind acum graful putem observa deja propoziția formată cu toate elementele ei.

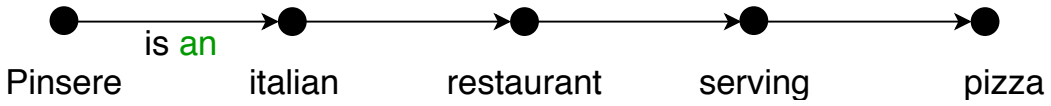

Figura 12: Efectul adăugării articolelor la graful din Figura 12

6 EVALUARE

Componenta de susținere a dialogului funcționează corect. În cazul tratării precizărilor reușește să construiască șablonul corect, corespunzător celor două șabloane ce reprezintă datele de intrare. Șablonul rezultant se dovedește a fi identic cu cel care este generat atunci când cele două cereri ce intră în scenariul cu precizare sunt date ca sub forma unei singure cereri.

Din punct de vedere al componentei de translatare, a fost implementată tratarea tuturor tipurilor de scenarii propuse, însă nu cu rezultate perfecte.

În Anexa A este prezentată o serie de scurte conversații ce cuprind rezultate ale componentei de translatare. Diferențele față de referințe sunt marcate cu text îngroșat. După cum se poate observa, ordinea cuvintelor este stabilită corect în toate cazurile, la fel și poziția prepozițiilor și particulelor. Probleme apar însă în cadrul poziționării articolelor. Acestea sunt corect determinate, însă există cazuri în care nu sunt plasate pe poziția potrivită în cadrul propoziției.

Înainte de toate, în exemplul 22 se poate observa faptul că, deși există în sistem mai multe date, este ales răspunsul corect din punct de vedere al conținutului. În acest exemplu în baza de cunoștințe există date despre 3 restaurante de 3 tipuri diferite. La întrebarea „Recommend me an american restaurant." este aleasă informația corectă și dat răspunsul „McDonalds is an american restaurant.".

Exemplul 14 din Anexa A poate fi consierat un exemplu clasic de scenariu simplu cu cerere condițională.

Un caz special de scenariu cu cerere condițională îl reprezintă exemplul 15 din Anexa A. Cererea ce a determinat răspunsul „Leaving the house." este „If Emily is at the front door, Emily must be leaving the house.". În acest caz problema apare în stagiul de eliminare a nodurilor reziduale, unde nodul „Emily" (din răspunsul așteptat „Emily must be leaving the house.") este eliminat datorită modului în care funcționează acest stagiu în cazul scenariilor cu cereri condiționale. Cazurile de tipul acestui exemplu încă nu a fost tratate și se înscriu pe lista dezvoltărilor viitoare.

Din punctul de vedere al scenariilor cu cerere de informații cu propoziții cu pronume interogativ, exemplul 1 poate fi considerat unul de referință.

Asemănător este și exemplul 6, în care se răspunde la întrebarea „Who is the upstairs neighbour?". Aici se poate observa însă faptul că poziționarea articolului în răspunsul „Mary is upstairs the neighbour." nu este tocmai corectă. Articolul este corect asociat substantivului neighbour. Totuși, din cauza faptului că „upstairs" a fost etichetat de către Stanford CoreNLP [18] ca fiind substantiv, algoritmul de poziționare a articolelor discutat în capitolul 5 secțiunea 12 nu a putut da rezultatul corect.

Exemplul 7, în care se răspunde la întrebarea „Tell me what a sunflower looks like?" reprezintă un scenariu ce poate fi atribuit oricărui subtip al scenariilor cu cereri de informații. Întrebarea conține atât un verb la imperativ, „tell", cât și pronumele interogativ „what". După cum se poate observa, răspunsul oferit este o propoziție corectă: „A sunflower looks like a yellow sun.".

După cum se poate observa din exemplele 8, 9 și 10 din Anexa A, sistemul poate trata inclusiv propoziții mai complexe. Toate aceste trei conversații răspund la aceeași întrebare, și anume „Recommend me a good book.", crescându-se însă complexitatea răspunsului de la un exemplu la altul. În cazul primului răspuns, „A good book is one about girl a living at Green-Gables." nu se reușește plasarea corectă a articolului nehotărât „a" ce determină substantivul „girl". Totuși, atunci când, în exemplul următor, substantivul primește atributele „named Anne" răspunsul generat este identic cu referința: „A good book is one about a girl named Anne living at Green-Gables.". În final, în exemplul 10 propoziția este îmbogățită cu sintagma „As far as I am concerned". Acest fapt determină plasarea greșită a articolului nehotărât „a" ce determină substantivul „book" în răspunsul „As far as I am a concerned good book is one about a girl named Anne living at Green-Gables". Această plasare greșită a articolului apare ca urmare a faptului că adjectivul „concerned" este imediat urmat de sintagma „good book". Algoritmul de plasare a articolelor asociază „a" cu substantivul „book", însă ceea ce nu realizează este faptul că lanțul invers de adjective ce determină „book" se oprește la „good", chiar dacă și următorul cuvânt de pe lanț, „concerned", este de asemenea un adjectiv. Algoritmul de plasare a articolelor nu are capacitatea de a testa dacă un adjectiv determină un anume substantiv sau nu.

Exemplul 21 este unul în care baza de informații este inițial goală. După cum se poate observa,

la prima întrebare a utilizatorului, „What is a red fruit?", este redat un răspuns automat care semnifică lipsa informațiilor pe subiectul cererii. Utilizatorul face apoi o afirmație „A cherry is a sweet red fruit.", care este memorată în sistem. Când este pusă încă o dată prima întrebare informația este găsită și răspunsul este oferit.

Un ultim exemplu discutat va fi exemplul 24 din Anexa A, care este unul ce cuprinde o precizare. La prima întrebare a utilizatorului „What is a yellow fruit?" este ales primul dintre cele două posibile răspunsuri: „A pineapple is a strange yellow fruit.". Utilizatorul adaugă apoi „A sour fruit". Cu toate să, în acest moment, se construiește corect șablonul pentru cererea „What is a sour yellow fruit?", totuși răspunsul găsit este în continuare „A pineapple is a strange yellow fruit.". Acest fapt se datorează modului în care funcționează componenta de potrivire a grafurilor și șabloanelor, însă se încearcă găsirea unei soluții.

7 CONCLUZII

În cadrul lucrării prezentate a fost dezvoltat un sistem care primește o cerere în limbaj natural de la utilizator sub forma unui șir de caractere, utilizează sistemul discutat în lucrarea Gabrielei Vlădescu [17] pentru a produce grafuri și șabloane [16] și a le potrivi conform celor descrise de Andrei Olaru [19] [20] pentru a obține un răspuns sub formă de graf. Răspunsul este apoi translatat înapoi în limbaj natural și oferit utilizatorului, moment în care procesul se poate relua.

Sistemul poate trata două tipuri de scenarii: scenarii cu cereri condiționale, de tipul „If X, then Y", respectiv scenarii cu cereri de informații, care pot lua forma unor întrebări cu pronume interogativ sau a unor comenzi, recunoscute după verbul la modul imperativ de tipul „Recommend me X" sau „Tell me Y". În plus, există posibilitatea adăugării unei precizări la cererea anterioară, fără a mai fi nevoie să se repete întreaga cerere, ci doar informația nou menționată.

Cu toate că rezultatele sunt mulțumitoare pentru grafuri simple ce se înscriu într-unul dintre scenariile menționate, ele pot fi îmbunătățite și chiar extinse la alte tipuri de scenarii și la grafuri mai complexe, inclusiv grafuri ce conțin cicluri sau scenarii în care același concept este referit de mai multe expresii. Astfel, proiectul prezentat poate reprezenta o bază pentru dezvoltarea unui sistem mai complex și mai cuprinzător.

Dezvoltări ulterioare

O primă direcție de dezvoltare care ar aduce mai multă flexibilitate componentei de translatare a grafurilor este crearea unui set de reguli care să poată stabili o ordine corectă a cuvintelor în propoziție și care să nu depindă de formatul propoziției din care a fost creat graful, ci de proprietăți ale cuvintelor cum ar fi tipul părții de vorbire.

O îmbunătățire semnificativă ar aduce-o și tratarea grafurilor cu cicluri. Acest lucru ar conduce la o mai mare flexibilitate în formularea cererilor de către utilizator, fapt înlesnit și de o altă

posibilă dezvoltare viitoare, și anume tratarea conceptelor referite de mai multe ori în cadrul aceluiași enunț. Este dorită și introducerea mai multor tipuri de scenarii, care să satisfacă nevoile utilizatorilor în diverse situații.

O dată cu tratarea scenariilor mai complexe vor apărea și posibile răspunsuri mai complexe. O funcționalitate care ar putea crește ușurința în citire și înțelegere a acestor răspunsuri este adăugarea punctuației.

Din punct de vedere al conținutului propoziției generate, se poate implementa o metodă de selecție mai riguroasă, care să elimine, pe lângă nodurile redundante discutate în cadrul lucrării, și informațiile nerelevante din punct de vedere al cererii care a generat răspunsul. De exemplu, în cazul unei informații interne de tipul „A pineapple is a strange yellow fruit", răspunsul actual la întrebarea „What is a yellow fruit?" este întreaga informație deținută. Aceasta ar putea fi totuși rezumată la „A pineapple is a yellow fruit." sau chiar la „A pineapple.".

BIBLIOGRAFIE

[1] R. Baguley, C. McDonald. *Appliance Science: Alexa, how does Alexa work? The science of the Amazon Echo*. https://www.cnet.com/news/appliance-science-alexa-how-does-alexa-work-the-science-of-amazons-echo. Ultima accesare: 22 iunie 2018

[2] *Cortana and privacy*. https://privacy.microsoft.com/en-us/windows-10-cortana-and-privacy. Ultima accesare: 22 iunie 2018

[3] *Speech*. https://developer.apple.com/documentation/speech. Ultima accesare: 22 iunie 2018

[4] S. Penrod. *Why We're moving to DeepSpeech on March 31 — Privacy, Speech to Text & Balance*. https://mycroft.ai/blog/mycroft-speech-to-text-and-balance. Ultima accesare: 22 iunie 2018

[5] J. L. Heimerl. *How Metadata Reveals More About You Than You Think*. https://www.securityweek.com/how-metadata-reveals-more-about-you-you-think. Ultima accesare: 22 iunie 2018

[6] M. Hunckler. T*his Open-Source AI Voice Assistant Is Challenging Siri and Alexa for Market Superiority*. Forbes. https://www.forbes.com/sites/matthunckler/2017/05/15/this-open-source-ai-voice-assistant-is-challenging-siri-and-alexa-for-market-superiority. Ultima accesare: 22 iunie 2018

[7] B. Schneier. *Open Source and Security*. https://www.schneier.com/crypto-gram/archives/1999/0915.html. Ultima accesare: 22 iunie 2018

[8] D. A. Wheeler. *Secure Programming HOWTO*. https://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO.pdf. Ultima accesare: 22 iunie 2018

[9] R. Perera, P. Nand. *Recent Advances In Natural Language Generation: A Survey And Classification Of The Empirical Literature*. *Computing and Informatics*, Vol. 36, 2017

[10] R. Dale, E. Reiter. *Building natural language generation systems*. Cambridge University Press, Cambridge, U.K., 2000

[11] R. Dale, S. Geldof, J. Prost. *CORAL: Using Natural Language Generation for Navigational Assistance*. *Australasian Computer Science Conference*, Darlinghurst, Australia, 2003

[12] R. Kutlak, C. Mellish, K. Deemter. *Content Selection Challenge*. *Proceedings of the Fourteenth European Workshop on Natural Language Generation*, Sofia, Bulgaria, 2013

[13] P. Cimiano, J. Lüker, D. Nagel, C. Unger. *Exploiting Ontology Lexica for Generating Natural Language Texts from RDF Data*. *Proceedings of the Fourteenth European Workshop on Natural Language Generation*, Sofia, Bulgaria, 2013

[14] T. H. Wen, M. Gasic, N. Mrkšić, P. H. Su, D. Vandyke, S. Young. *Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems* . *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisabona, Portugalia, 2015

[15] S. Hochreiter, J. Schmidhuber. *Long Short-Term Memory*. *Neural Computation* Vol. 9, 1997

[16] A. Olaru, A. M. Florea, A. El Fallah Seghrouchni. *Graphs and Patterns for Context-Awareness*. *Ambient Intelligence - Software and Applications. Advances in Intelligent and Soft Computing* Vol. 92, 2011

[17] G. Vladescu, A. Olaru. *User-Friendly Representation of Context Data*. Teză de disertație, urmează a fi publicată

[18] https://stanfordnlp.github.io/CoreNLP/ Ultima accesare: iunie 2018

[19] A. Olaru, A. M. Florea. *A Platform for Matching Context in Real Time. Hybrid Artificial Intelligent Systems. HAIS 2015. Lecture Notes in Computer Science Vol. 9121*, 2015

[20] A. Olaru. *Context Matching for Ambient Intelligence Applications. 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timișoara, România, 2013

[21] M. Mihalea. *Agent Asistent Inteligent - Prelucrarea și generarea vorbirii*. Lucrare de diplomă, urmează a fi publicată

8 ANEXA A: REZULTATE PRODUSE DE COMPONENTA DE TRANSLATARE

Legendă: I = Informație deținută de sistem U = Cerere adresată de utilizator R = Răspunsul sistemului

1. I: A pineapple is a strange yellow fruit.
   U: What is a strange fruit?
   R: A pineapple is a strange yellow fruit.
2. I: Emily is going to the beach in summer.
   U: Where is Emily going?
   R: Emily is going to the beach in summer.
3. I: Lily-of-the-valley are small white flowers.
   U: What are some white flowers?
   R: Lily-of-the-valley are small white flowers.
4. I: Emily is feeling fine.
   U: How is Emily feeling?
   R: Emily is feeling fine.
5. I: A bluebird is a nice bird.
   U: Tell me what is a nice bird?
   R: A bluebird is a nice bird.
6. I: Mary is the upstairs neighbour.
   U: Who is the upstairs neighbour?
   R: Mary is upstairs **the** neighbour.
7. I: A sunflower looks like a yellow sun.
   U: Tell me what a sunflower looks like?
   R: A sunflower looks like a yellow sun.
8. I: A good book is one about a girl living at Green-Gables.
   U: Recommend me a good book.

R: A good book is one about girl **a** living at Green-Gables.

9. I: A good book is one about a girl named Anne, living at Green-Gables.

   U: Recommend me a good book.

   R: A good book is one about a girl named Anne living at Green-Gables.

10. I: As far as I am concerned, a good book is one about a girl named Anne, living at Green-Gables.

    U: Recommend me a good book.

    R: As far as I am **a** concerned good book is one about a girl named Anne living at Green-Gables.

11. I: Pinsere is an Italian restaurant serving french fries.

    U: Recommend me an Italian restaurant.

    R: Pinsere is an Italian restaurant serving french fries.

12. I: The Aristocats is an animated movie.

    U: Recommend me an animated movie.

    R: The Aristocats is an animated movie.

13. I: Violets are small blue flowers.

    U: Recommend me some blue flowers.

    R: Violets are small blue flowers.

14. I: rain is forecast.

    U: If rain is forecast, bring an umbrella.

    R: Bring an umbrella.

15. I: Emily is at the front door.

    U: If Emily is at the front door, Emily must be leaving the house

    R: **Leaving the house.**

16. I: The season is summer.

    U: If the season is summer, the weather is hot.

    R: The weather is hot.

17. I: The day is Sunday.

    U: If the day is Sunday, I go to the cinema.

    R: I go to the cinema.

18. I: The weather is cold and cloudy.

    U: If the weather is cold and cloudy, it may start to snow.

R: It may start to snow.

19. I: Emily is right.

    U: If Emily is right, the schedule must have changed.

    R: The schedule must have changed.

20. I: Emily is at home.

    U: If Emily is at home, she will answer the green door.

    R: She will answer the green door.

21. U: What is a red fruit?

    R: I am not wise enough to know that.

    U: A cherry is a sweet red fruit.

    R: Ok.

    U: What is a red fruit?

    R: A cherry is a sweet red fruit.

22. I: Pinsere is an italian restaurant.

    I: McDonalds is an american restaurant.

    I: Chocolat is a french restaurant.

    U: Recommend me an american restaurant.

    R: McDonalds is an american restaurant.

23. U: Recommend me an animated movie.

    R: My wisdom is not infinite.

    U: The Aristocats is an animated movie.

    R: Ok.

    U: Recommend me an animated movie.

    R: The Aristocats is an animated movie.

24. I: A pineapple is a strange yellow fruit.

    I: A lemon is a sour yellow fruit.

    U: What is a yellow fruit?

    R: A pineapple is a strange yellow fruit.

    U: A sour fruit.

    R: A pineapple is a strange yellow fruit.