TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für angewandte Informatik

Dokumentation Komplexpraktikum Mensch-Computer Interaktion

Individualisierbare Sprachinteraktion: Deep-Learning-Ansatz zur Konfiguration von Feature-Modellen

Manuel Jorde
Matrikelnummer: 4801360

Clara-Sophie Kasassov
Matrikelnummer: 4763578

Eric Lischinski
Matrikelnummer: 4763559

20. September 2022

Betreuer
David Gollasch

Betreuender Hochschullehrer
Prof. Dr.-Ing. Gerhard Weber

Inhaltsverzeichnis

1 Ablauf

1.1 Aufgabenstellung

Das Ziel unserer Projektarbeit im Zuge des Komplexpraktikums bestand aus der Anwendung der Softwarevariabilität zur Umsetzung individualisierbarer Sprachinteraktion, wobei ein Deep-Learning-Ansatz gewählt werden sollte, um Feature-Modelle konfigurieren zu können. Somit soll für die Sprachinteraktion ein Feature-Modell erstellt werden, welches alle nutzerspezifischen Individualisierbarkeiten des Gesprächs enthält und später auch ein Deep-Learning-Algorithmus implementiert, welcher die richtige Konfiguration für einzelne User zurückgibt. Das Endresultat sollte ein Konfigurationsmechanismus sein, welcher die Sprachinteraktion auf verschiedene Besonderheiten wie beispielsweise Behinderungen, Interessen oder Sprachgebrauch der User anpassen kann und somit den Dialog individuell personalisiert.

1.2 Arbeitsschritte/-Fortschritte

Unsere Arbeitsschritte lassen sich im 2-Wochen Rhytmus darstellen. In den ersten beiden Wochen bestand unsere Arbeit darin, sich in die Thematik Deep Learning, Recommender Systeme und Softwarevariabilität einzulesen. Mithilfe mehrerer Paper wie z.B. Neural Collaborative Filtering[2]. Daraufhin haben wir uns folgende Teilziele gesetzt.

1. Vorbereitung der Deep-Learning-Thematik, sowie variabler Sprachinteraktionen: Wie lässt sich der Konfigurationsprozess als Deep-Learning-Problem abbilden? Welchen individualisierbaren Möglichkeiten sind in einem Dialog wichtig?
2. Entwicklung eines Feature-Modells zur Darstellung der Interaktionsmöglichkeiten.
3. Identifikation geeigneter Deep-Learning-Modelle für Tensorflow bzw. Keras.
4. Entwicklung eines Sprachskills, mit passenden Anpassungsmöglichkeiten auf Basis des Feature Modells.

Die 2. Etappe bestand darin, erste Entwicklungen am Feature-, sowie User-Model zu tätigen und bereits Vorüberlegungen für Teilziel 3 zu machen. Wir arbeiteten an den ersten Entwürfen des User- und Feature-Modells und erweiterten diese mehrmals nach Rücksprache mit unserem Betreuer. In Abbildung 2 und 3 ist unser Endprodukt zu erkennen. Unsere Hauptaktoren sind Interessen, Eloquenz, Sprachniveau, Information, Artikulation und Stimme, welche jeweils noch weiter aufgesplittet und verfeinert wurden. Daraus entwickelte sich die Nutzerkonfiguration, welche in Abbildung 4 erkennbar ist. Weiterhin beschäftigten wir uns bereits mit dem Deep Learning Problem, damit wir reibungslos in die Entwicklung überspringen konnten.
Die darauffolgenden 2 Wochen bestanden daraus, das Feature, User-Modell zu finalisieren, um die Basis der Sprachindividualisierung zu haben. Weiterhin begonnen wir damit unser Recommender System in Python zu entwickeln, da nun die theoretischen Überlegungen feststanden.
In Etappe 4 bauten wir das Recommender System aus und finalisierten es. Nun hatten wir eine Datenbank mit 100 Usern, welche jeweils ein Profil und eine Konfiguration besitzen. Darauf

basierend kann nun mittels KNN und einem Profil, jedem Nutzer eine passende Konfiguration gegeben werden, um möglichst barrierefrei mit dem Sprachassistenten zu kommunizieren. Danach begonnen wir damit einen Sprachskill mittels Alexa zu entwickeln. Dies startete damit einen lokalen Server in Python zu entwickeln, über welche. die Konfiguration als JSON an Alexa gesendet wird. Danach konzentrierten wir uns darauf ein Dialogbeispiel zu schreiben und die Kommunikation zwischen Recommender System und Alexa fertigzustellen.
In Etappe 6 fiel der Fokus darauf den Sprach-Skill weiterzuentwickeln. Durch URL Aufrufe mittels Alexa, konnten wir nun direkt, mithilfe des Namens, die Konfiguration bereitstellen und an Alexa zurücksenden, um die Interaktion dementsprechend anzupassen. Zum Schluss kümmerten wir uns darum ein Webtemplate zu erstellen, damit man einen neuen User hinzufügen kann, welcher in die Datenbank übernommen wird und über den Alexa Dialog simuliert werden kann.

2 Anleitung

2.1 Installationsanweisung

Zu Beginn der Installation wird der Port 8080 freigegeben, um später Alexa den Zugriff auf den Server über URL Anfragen auf die globale IP Adresse zu gewährleisten.
In der IDE werden die Bibliotheken „sklearn“, „flask“ und „pandas“ benötigt und als Python-Interpreter wird Python 3.9 gewählt. Der Code wird von GitHub gedownloadet und die IP-Adresse in der Klasse ”Skill_Handler.py” auf die eigene, lokale Adresse anpasst. Anschließend wird die Klasse ”Recommender_System.py” ausgeführt, um den Server zu starten, auf dem die Informationen über die jeweiligen Namen abrufbar sind. Der Code ”Alexa Skill” wird in der Maven Configuration mit ”assembly:assembly -DdescriptorId=jar-with-dependencies package” zu einer jar-Datei gepackt.
Bei Amazon Web Services AWS wird nun eine Lambda Funktion erstellt. Die Einstellungen dafür sind ”Author from Scratch”, ”Java 11 Corretto” und „x86_64“. In dem Reiter Code Source wird die jar-Datei hochgeladen und bei Runtime Settings wird als Handler ”alexa_skill.SpeechStreamHandler” eingegeben. Anschließend wird Alexa als Trigger gewählt und die ARN kopiert, da diese später noch gebraucht wird.
In der Developer Console wird ein Skill in der Default Region EN erstellt. Die Einstellungen für die Konsole sind „Custom und Provision your own“ und „Start from Scratch“. Danach wird im JSON Editor die JSON Datei aus dem Code kopiert und der Skill Invocation Name auf „recommender system“ gesetzt. Bei dem Endpoint AWS Lambda ARN wird bei Default Region die kopierte ARN eingefügt. Anschließend wird das Model gespeichert und gebaut. Damit wurde das Proramm vollständig installiert und kann nun nach der Bedienungsanleitung verwendet werden.

2.2 Bedienungsanleitung

Zu Beginn startet man den Server, indem man die Klasse ”Recommender_System.py” ausführt. Beim Start wird im Hintergrund das Model trainiert, sodass unsere KI die bestmöglichen Konfigurationen empfehlen kann. Nachdem der Server gestartet ist, wechseln wir nun zur Alexa Developer Console. Dort können wir im Bereich Test mit Alexa sprechen. Der Dialog wird initial durch die Wörter „hello recsys“ gestartet. Nun kann der Beispiel Dialog mit Alexa durchgelaufen werden und danach erneut durch den Befehl ”hello recommender system” gestartet werden. Währenddessen können durch ein Web-Template weitere User erstellt und der Datenbank hinzugefügt werden. Hierzu muss man lediglich seine IP-Adresse in die Suchleiste eintragen und /MCI/newUser dahinter schreiben, sodass es zB. So aussieht ”https://127.0.0.0/MCI/newUser”. Hier findet man ein Formular vor, wo man seinen Namen, sowie weitere wichtige Daten für unseren Algorithmus eintragen muss, falls man einen neuen User anlegen möchte. Sobald man die Daten eingetragen hat, muss man den „Submit“ Button drücken, um das Formular abzusenden. Falls dies korrekt erfolgt, taucht unter dem Formular eine Grüne Box auf, wo drinsteht, dass der User erfolgreich mit der jeweiligen ID angelegt wurde. Sobald Daten nicht ausgefüllt wurden und man trotzdem auf den „Submit“ Button drückt, taucht nun anstatt der grünen Box eine rote auf, welche die fehlende Information enthält.

3 Code - Recommender System Server

3.1 Recommender System

3.1.1 Aufbau

Der Server basiert auf folgenden Python-Dateien:

- Recommender_System.py
- Data_Fitting.py
- Config_KNN.py
- Skill_Handler.py

Des Weiteren werden folgende Datensätze zum Trainieren des KNN als auch für User-Abfragen und User-Speicherung verwendet:

- user.csv
- configs.csv
- users_created.csv (neu angelegte User ohne prebaked configs)

Dazu findet man außerdem noch eine "Profile_Creator.py" Datei, welche lediglich dafür erstellt wurde, um schneller zufällige Userprofile zu generieren und nicht essenziell für den Server ist.

3.1.2 Funktionsweise

Die Datei „Recommender_System.py“ ist das Hauptscript welches zum Beginn ausgeführt wird. Am Anfang dieses Scriptes kann man verschiedene Variablen für das aktivieren und deaktivieren von Einstellungen festlegen. Dort kann man mit „do_test“ festlegen, ob nach dem Trainieren des Models noch ein Testset durchlaufen werden sollen, um die Accuracy des Models zu bestimmen. Dabei gibt „print_probability“ die Möglichkeit, die einzelnen Confidences zu jeder Entscheidung ausgeben zu lassen. Des Weiteren kann man mit „do_recommendation“ Tests mit einzelnen Usern durchführen lassen und sich die jeweilige Treffergenauigkeit ausgeben lassen. Mit der Variable „config_translate“ werden hierbei die Konfigurationen in lesbare Worte konvertiert, da der KNN ausschließlich mit Integern arbeitet. Außerdem lassen sich auch die Parameter „test_size“, „number_of_neighbours“ für den KNN einstellen. Für verschiedene Testgründe kann man mit „best_of_n“ das Model auch wiederholt trainieren lassen, bis das resultierende Modell eine bestimmte Accuracy erreicht.

Die main-Funktion beginnt damit die getroffenen Einstellungen an die „Config_KNN.py“ zu übergeben und führt dann die Funktionen „fit_data()“ von „Data_Fitting.py“ aus, welche die einzelnen Datensätze aus den CSV-Dateien ausliest und für das Trainieren des Models vorbereitet. Schließlich wird mit der Funktion „trainmodel()“ unser KNN Modell trainiert und es gibt uns Testsätze zurück, mit denen auf Wunsch die Accuracy des Modells bestimmt werden

kann. Wurde die Einstellung „do_test“ aktiviert, wird das Model mit den Testsätzen getestet, wie gut es die Konfiguration voraussagen kann. Bei aktivertem „do_recommendation“ wird nun der spezifizierte Beispiel-User, sowie die dazu erstellte Config, genommen und in die Funktion „config_rating()“ gegeben. In dieser Funktion wird nun jeder Wert von der bestehenden Config mit der erstellten des KNN verglichen und dadurch eine Erfolgsquote berechnet. Nach durchlaufen dieser Funktionen wird die alte Config, die neue Config, der User und die Erfolgsquote in der Ausgabe angezeigt.

```
# SETTINGS
do_test = False
print_probability = False
number_of_neighbors = 10
test_size = 0.1

do_recommendation = True
recommendation_ids = [2, 7, 8]
config_translate = False

best_of_n = True  # only for test purpose
min_score = 0.88


Vanjrd *
def main():

    if not best_of_n:
        KNN.apply_settings(print_probability, config_translate, number_of_neighbors, test_size)
        x, y, user_mappings, config_mappings = Fitting.fit_data()
        x_test, y_test = KNN.trainModel(x, y)

        # debug purpose
        if do_test:
            KNN.testModel(x_test, y_test, 0)
        if do_recommendation:
            user, user_config = Skill_Handler.getUser(recommendation_ids)
            for x in range(len(user)):
                KNN.config_rating(user[x], user_config[x])
```

Abbildung 1: Settings und Main-Funktion

Zuletzt wird mit „Server()“ von der „Skill_Handler.py“ der Flask Server gestartet.

Die Datei „Config_KNN.py“ enthält die Funktionen „apply_settings()“, „trainModel()“, „testModel()“, „predictConfig()“, „config_rating()“. Dabei wendet „apply_settings()“ die in „Recommender_System.py“ ausgewählten Einstellungen an. „predictConfig()“ sagt die Konfiguration für einen hereingegebenen Nutzer voraus. Dabei wird die Funktion „predict()“ des SKLearn Package genutzt, die auf Basis des KNN aus „trainModel()“ entscheidet. Die restlichen Funktionen wurden bereits in der Datei „Recommender_System.py“ erklärt.
In der Datei „Data_Fitting.py“ gibt es 3 Übersetzungsfunktionen. „translate_userMapping()“, „retranslate_configMapping()“, „translate_configMapping()“ sind dafür zuständig die Config-Daten lesbar für den Benutzer in Strings zu übersetzen und wiederum lesbar für den KNN in Integer zu ändern. Dabei werden Mappings verwendet, welche beim Datafitting erstellt wurden und einen Bezug zwischen den Integer Werten des KNNs und den zugehörigen Strings herstellt.

```
def trainModel(X, y):
    x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(X, y, test_size=test_size)
    x_train = list(x_train)
    y_train = list(y_train)
    for obj in range(len(x_train)):
        x_train[obj] = x_train[obj][2:]
        y_train[obj] = y_train[obj][2:]
    model.fit(x_train, y_train)
    return x_test, y_test
```

Abbildung 2: trainmodel()-Funktion

In „Skill_Handler.py“ findet man die Funktionen, die für den Flask Server verantwortlich sind. Durch „Server()“ und „app.run(ip,port)“ wird der Flask-Server initialisiert und auf derjeweiligen IP-Adresse mit dem angegebenen Port gestartet. Mithilfe von „@app.route“ lassen sich URL-Aufrufe seitens Alexa abfangen und dadurch bestimmte Funktionen auf Basis des URL-Aufrufs starten. „return_config()“ liefert, für die per URL-Request angeforderte User-ID, eine vom KNN erstellte Konfiguration. Dafür wird innerhalb dieser Funktion eine andere Funktion namens „get_config()“ aufgerufen, welche die passenden Userdaten zur angegeben User-ID raussucht und mittels predictConfig() eine vom KNN vorausgesagte Konfiguration zurückgibt. Diese Konfiguration wird in eine JSON Datei gepackt und auf den Flask-Server zurückgegeben. „get_new_user()“ empfängt die eingegebenen Werte des Web-Templates zu einem neuen User als JSON und speichert diese im Datensatz von „users_created.csv“. Dies funktioniert mit „storeUser()“. Diese Funktion erstellt einen neuen User und ließt die Daten aus der JSON-Datei aus, welche in die Datenbank übernommen werden und schließlich eine neue ID für den User vergeben, welche dann zum Schluss in „get_new_user()“ an das Webtemplate zurückübergeben wird. Die Funktion „newUser_form()“ stellt lediglich das Webtemplate „newUser.html“ auf der angegeben URL zur Verfügung. „get_csvuser()“ und „get_csvconfig()“ ließt die Daten für die angegebene UID aus und gibt diese als Liste zurück. Diese Funktionen sind aber lediglich für das Testen und Auswerten des KNNs zuständig.

```python
@app.route('/MCI/<int:uid>/')
def return_config(uid):
    data = get_config(uid)
    userName = data[1]
    data = data[0]

    if data is not None:
        return jsonify({'interests': data[1],
                        'language_usage': data[2],
                        'foreign_words': data[3],
                        'slang_words': str(data[4]),
                        'formality': data[5],
                        'conciseness': str(data[6]),
                        'information_density': data[7],
                        'repeat': data[8],
                        'language': data[0],
                        'speed': data[9],
                        'emphasis': data[10],
                        'speech_pauses': str(data[11]),
                        'volume': data[12],
                        'voice_gender': data[13],
                        'voice_age': data[14],
                        'userName': userName})
    else:
        return "No User"
```

Abbildung 3: return_config()-Funktion

```python
@app.route('/MCI/newUser', methods=['GET', 'POST'])
def newUser_form():
    if request.method == 'POST':
        return redirect(url_for('index'))
    return render_template('newUser.html')
```

Abbildung 4: newUser_form()-Funktion

3.2 Web Template

In der „newUser.html“ wurde ein Web-Template entwickelt, mit welchem man neue Nutzer anlegen kann. Weiterhin erstellten wir noch eine ”new_user.css” um das Design der HTML-Datei anzupassen. Das Template besteht aus 2 Inputs für den Namen und das Alter. Der Name kann nur aus Buchstaben bestehen und das Alter geht von 1-150, andere Angaben sind nicht zulässig. Danach kommen 6 Selects für die Werte:

- Interessen
- Behinderungen
- Bildung
- Formalität
- Sprache
- Bevorzugter Redepartner

Die Verwendung von Selects begründet sich daher, dass hier keine fehlerhaften Angaben gemacht werden dürfen, da wir für die einzelnen Informationen feste Werte besitzen. Das Alter, sowie der Name müssen als String Input eingegeben da diese stark variieren können. Unter den Eingaben befindet sich ein Submit-Input, welcher beim Drücken die Funktion „submit“ auslöst. Diese Funktion schreibt die eingegeben Werte auf 8 Variablen und schickt diese mittels POST-Request als JSON zurück an den Server. Des Weiteren befindet sich ein verstecktes Ausgabefeld unter der gesamten Eingabe, welches erst auftaucht falls falsche Eingaben gemacht werden oder ein Feld nicht ausgefüllt wurde, was beim Betätigen des Submit-Buttons geprüft wird. Falls dies auftritt, wird das Ausgabefeld sichtbar gemacht und die Farbe auf rot geändert. Im Ausgabefeld wird ein Text erzeugt welcher das fehlende Element oder den Fehler enthält. Im Falle einer korrekten Eingabe wird bei Klick auf den Submit-Button das Ausgabefeld sichtbar gemacht und seine Farbe auf grün geändert. Der erzeugte Text enthält den Namen des Nutzers und die jeweilige ID an der er angelegt wurde. Die ”new_user.css” enthält die jeweiligen Design Einstellungen für jedes Element. Der HTML-Body wird von einem div umfasst, welches die Klasse Background hat. Dieses div erzeugt den milchig weißen Hintergrund vom Template. Das innere div ”Container” beinhaltet die gesamte Eingabe. Dies wurde aus Positionierungsgründen gewählt, da sich so alles einfacher verschieben lässt. Für die Eingaben wurden die Farben Royalblue und Midnightblue gewählt. Weiterhin hat die Überschrift die Schriftart ”Rockwell”. Die Informationen haben die Schriftart ”Imperial” und die Eingabefenster ”Open Sans”. Der Submit-Input ist auch in einem einzelnen div, um ihn bezüglich der Eingabe mittiger zu positionieren.

```
var user = {
    'uID': '',
    'userName': userName,
    'age': age,
    'interests': interests,
    'disability': disability,
    'education': education,
    'formality': formality,
    'language': language,
    'preferred_speaker': preferred_speaker
}

$.ajax({
    type: "POST",
    url: "/get_new_user",
    contentType: "application/json",
    data: JSON.stringify(user),
    dataType: "json",
}).done(function(result){
    console.log(result);
    document.getElementById("button").style.display = "block";
    document.getElementById("button").style.backgroundColor = "forestgreen";
    document.getElementById("demo").innerHTML = "Successfully submitted '" + result.userName + "' on ID: " + result.uID;
});
```

Abbildung 5:

4 Code - Alexa Skill

4.1 Aufbau

Für den Alexa-Skill werden folgende Handler verändert oder hinzugefügt:

- LaunchRequestHandler.java
- EndIntentHandler.java
- NameIntentHandler.java
- WellBeingIntentHandler.java
- InterestIntentHandler.java
- RepeatIntentHandler.java

Des Weiteren werden folgende Klassen zur Charakterisierung und Verfeinerung der Alexa-Ausgabe verwendet:

- SpeechConfig.java
- VoiceConfig.java

Zu Beginn der Konversation wird die User-ID des aktuellen Benutzers abgefragt. Die ID wird mit Hilfe eines Slots des Typen „AMAZON.NUMBER“ an die Klasse „NameIntentHandler.java“ übergeben. Diese wird ausgelesen und mit Hilfe von "http://"+ publicIP + /MCI/"+ ID+ /"wird ein URL Aufruf gegen den Server gestartet, damit der Alexa Handler auf die entsprechende Website zugreifen kann. Die auf der Website enthaltenen Informationen sind:

- Sprache (englisch, deutsch)
- Name
- Interessen (Sport, Wetter, Politik, Klatsch)
- Betonung (stark, normal, keine)
- Geschwindigkeit (schnell, mittel, langsam)
- Sprachlevel (hoch, mittel, niedrig)
- Wiederholung (schneller, gleich, langsamer)
- Slang (ja, nein)
- Sprechergeschlecht (weiblich, männlich)
- Formalitäten (formal, informal)
- Informationsdichte (stark, mittel, wenig)

Die Interaktion startet durch Begrüßung des Nutzers und mit der Nachfrage nach dem Wohlbefinden. Der „WellBeingIntentHandler.java“ wird mit einer Aussage über die aktuelle Stimmung aktiviert.Es folgt die Frage, ob Interesse besteht, Neuigkeiten über das nutzerspezifische Hobby mitgeteilt zu bekommen. Aus grammatikalischen Gründen wird beim Deutschen die Funktion „getDecliInterest“ aus „SpeechConfig.java“ aufgerufen. In dieser wird das Hobby übersetzt und in den Kontext gebracht. Durch Bestätigung dieser Frage wird der „InterestIntentHandler.java“ aufgerufen, welcher je nach Informationsdichte und Sprachlevel der Person eine Antwort wählt. In der Klasse „SpeechConfig.java“ wird mit der Funktion „chooseLevel“ je nach Sprachniveau entschieden wie lang und detailliert die Ausführung über das Interesse sein sollte.
Im Anschluss gibt es die Möglichkeit, eine Wiederholung des Gesagten anzufordern. Der Skill wird je nach Nutzerkonfiguration entscheiden, welche Länge und Ausführlichkeit der Wiederholung für das Individuum von Interesse ist.

4.2 Sprachkonfiguration

Die JSON Datei von dem Server wird ausgelesen, gespeichert und mit Hilfe von ”getStrings()”dann an die Klasse ”SpeechConfig.java” übergeben. Zusätzlich werden gleich zu Beginn die Parameter ”level_formality” und ”level_slang” erstellt. Diese stellen sich aus den Werten des Sprachlevels, der Formalität und des Slangs zusammen. Definiert werden diese, um später im Code ohne redundandte Schleifendurchläufe die Auswahl der richtigen Sprachbausteine zu ermöglichen.
Infolgedessen wird die Person mit Namen angesprochen und nach dem Wohlbefinden gefragt. Unsere erste Lösung bestand daraus 12 Strings pro Sprache zur Verfügung zu stellen, welche im Sprachlevel, im Slang und in den Formalitäten variierten und einer davon im Durchlauf einer switch-case Anweisung ausgewählt wurde. Allerdings wurde dieser Ansatz verworfen und ersetzt.
In der neueren Variante erfolgt die Auswahl der Antwort nun über Wortbausteine, welche passend zusammengesetzt werden. Dies geschieht, indem auf einer HashMap die verfügbaren Bausteine gespeichert werden. Ein Beispiel aus dem Code ist in Abbildung

An dieser Stelle folgt der Einsatz der anfangs gespeicherten Strings ”level_formality” und ”level_slang”. Wie in der Abbildung erkennbar, bestehen die Keys der Map immer aus den zusammengesetzten Werten der Variablen oder dem Level selbst. In ”SpeechConfig.getAnswer” erfolgt das auslesen der Map. Dafür werden die beiden Strings und das Sprachlevel als Key verwendet. Die Antwort stellt sich nun also aus den drei Map-Values zusammen. Anschließend werden die Stimmeigenschaften noch hinzugefügt. 1 erkennbar.

In dem Handler ”WellBeingIntentHandler” wird zudem die Funktion ”getDecliInterest”aus ”SpeechConfig.java” benötigt. Diese stellt den grammatikalisch korrekten Ausdruck im Kontext der Aussage zur Verfügung. Im Englischen wird diese Funktion nicht benötigt, daher ist zudem auch die ”getInterest” Funktion von Interesse. Anhand derer ist es dem ”WellBeingIntentHandler” möglich, nutzerindividuell die richtige Interesse herauszufinden um die Ausgabe anzupassen.

4.3 Sprecherkonfiguration

Zur Erstellung der vollendeten Ausgabe, wird die Klasse ”VoiceConfig.java” aufgerufen, welche Informationen aus der Konfiguration erhält. Hierbei werden die Stimmeigenschaften an den Benutzer angepasst, indem SSML Tags bezüglich der Prosodie, dem Sprecher und der Betonung anhand der Nutzerinformationen passend ausgewählt und zu einem Rahmen rund um einen Platzhalter für künftige Aussagen zusammengesetzt werden. Die SSML Tags werden

anfangs vordefiniert auf Strings, welche die Eingabe erleichtern sollen. Es eine Funktions namens "getSpeak()äufgerufen. Diese durchläuft eine switch-case Anweisung, welche auf den Eingabeparametern ”language”, ”gender”, ”speed” und ”emphasis” basiert. Entsprechend der Auswahl wird der String ”speech” für die Klasse ”SpeechConfig” bereit gestellt.
Selbiges passiert mit dem String ”repetition”. Letztendlich sieht die Ausgabe wie folgt in Abbildung 1 aus:

```
public static String getAnswer(Map<String,String> map){
    String answer = map.get(level_slang) + map.get(level_formality) + map.get(level);
    speechText = speech.replace( target: "zero", answer);
    return speechText;
}
```

Abbildung 1: Codeausschnitt

In der Klasse ”WellBeingIntentHandler.java” wird der String ”speech” durch eine Emotion erweitert. Diese ist je nach Nutzereingabe entweder fröhlich oder traurig. Dies geschieht ebenfalls in der Klasse ”SpeechConfig”. Die Funktion ”addEmotion()” fügt dem String ”speech” die SSML Tags amazon:emotion hinzu. Damit diese außerhalb diesen Handlers nicht mehr verwendet werden, wird ”speech” gleich nach der Zusammenstellung des fertigen Strings durch ”endEmotion” wieder auf seine ursprüngliche Form zurückgesetzt.

5 Abbildungen

```
if(language.equals("german")){
    map.put("low_1", String.format("Yo %s. ", username));
    map.put("low_0", String.format("Alles klar %s. ", username));
    map.put("low_formally", "Wie geht es Ihnen?");
    map.put("low_informally","Wie geht es dir?");
    map.put("low", "");

    map.put("normal_1",String.format("Hi %s. ", username));
    map.put("normal_0", String.format("Hallo %s. ", username));
    map.put("normal_formally", "Wie geht es Ihnen ");
    map.put("normal_informally","Wie geht es dir ");
    map.put("normal", "heute?");

    map.put("pompous_1",String.format("Sei gegrüßt %s. ", username));
    map.put("pompous_0",String.format("Guten Tag %s. ",username));
    map.put("pompous_formally","Wie fühlen Sie sich ");
    map.put("pompous_informally","Wie fühlst du dich ");
    map.put("pompous","heute?");

    speechText = SpeechConfig.getAnswer(map);
      }
```

Abbildung 1: Codeausschnitt

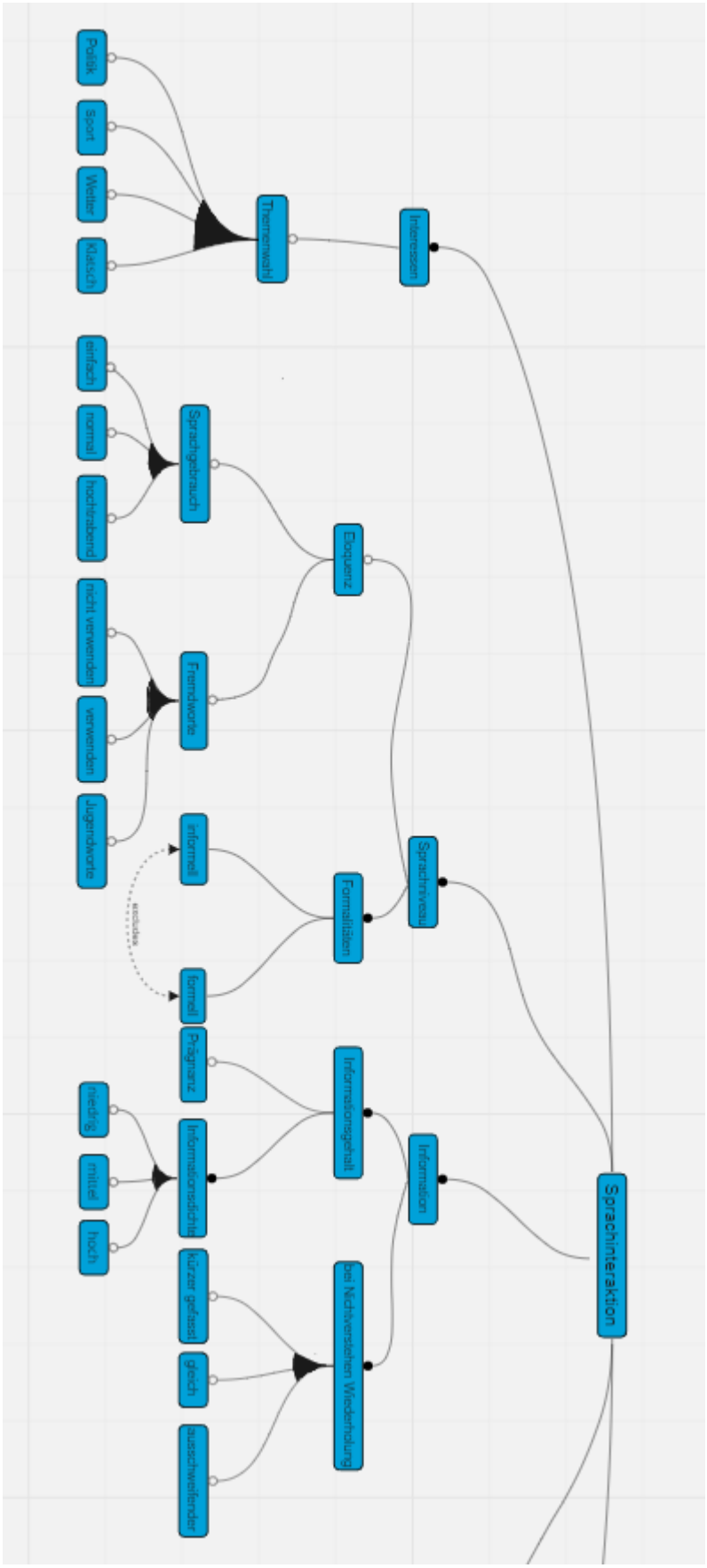

Abbildung 2: Feature-Modell 1

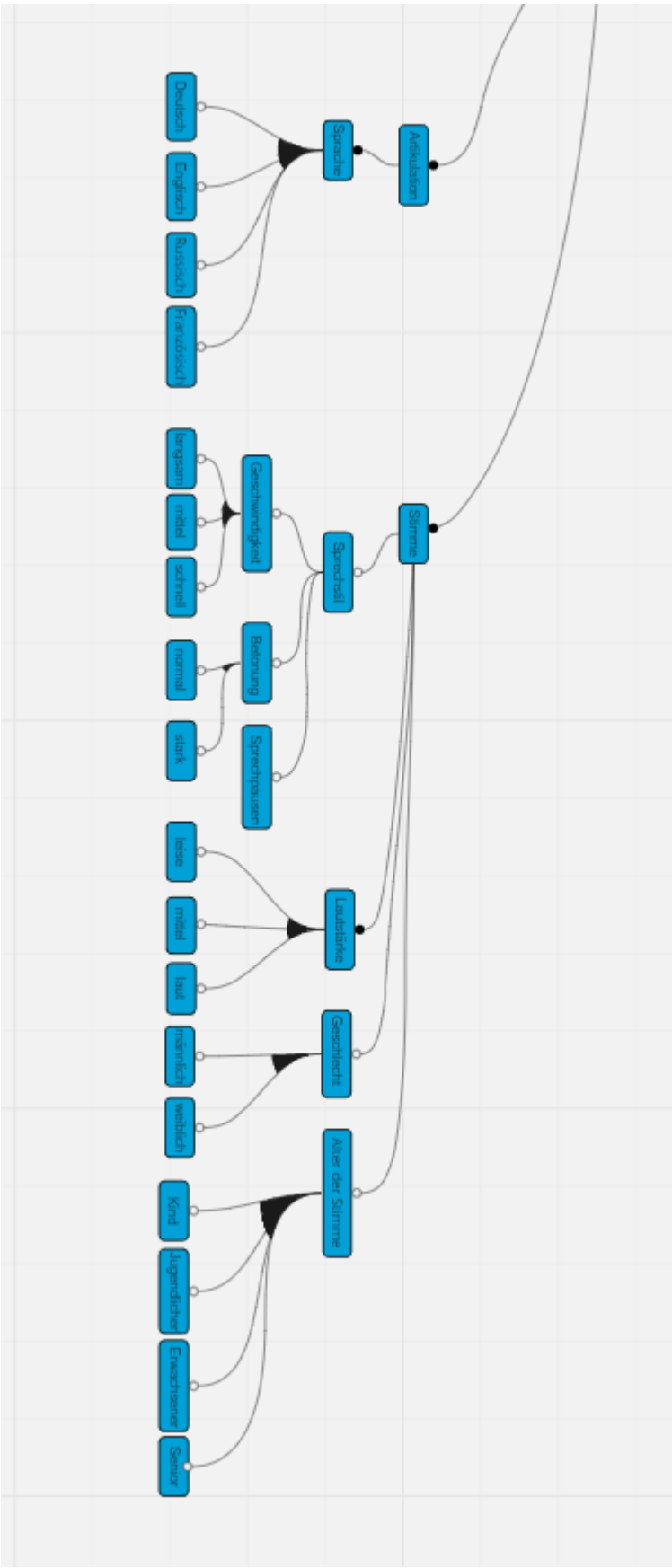

Abbildung 3: Feature-Modell 2

Features	Mr. Brown
Themenwahl	Politik
Sprachgebrauch	normale Sprache
Fremdworte	nicht verwenden
informell	nicht verwenden
formell	verwenden
Prägnanz	Aktiviert
Informationsdichte	gering
bei Nichtverstehen	gleich
Sprache	Deutsch
Geschwindigkeit	langsam
Betonung	stark
Sprechpausen	Aktiviert
Lautstärke	laut
Geschlecht	männlich
Alter der Stimme	Senior

Abbildung 4: Nutzerkonfiguration