Classifier-Basierter Conversation-Service

Kontext zum Teilprojekt

Ziel dieses Projektes ist die Anbindung der IWIBot Chat-Anwendung an den Classifier-Rest-Service und steuerung der Konversationslogik. In dieser Dokumentation wird der grundlegende Aufbau und die Funktionsweise des Conversation-Services beschrieben. Dies Rest-Schnittstelle zur Klassifizierung von Sätzen und deren Dokumentation, auf der dieser Conversation-Service basiert, ist im Teilprojekt **classification** zu finden.

Zusätzlich dazu bietet dieser Conversation Service die selben Schnittstellen, die selben Parameter und das selbe Ergebnisobjekt an wie der Conversation Service, den die IBM-Cloud bereitstellt. Das macht es besonders einfach, zwischen den beiden Conversation-Services zu wechseln. Wie der Wechsel genau funktioniert, wird später besprochen.

Anforderungen und Funktionsweise

Bestandteile einer Konversation

Eine Anfrage über die Oberfläche durch den Nutzer, welche die Ausführung einer Action zur Folge hat, wird im Conversation-Service wie folgt aufgeteilt:

- `intent:` Gibt den Namen der Action an, die ausgeführt werden soll
- `entity:` Eine für die Action erforderliche Verfeinerung der Anfrage

Beispiel: (*Zeige mir den Studenplan am Freitag*) -> `intent = 'timetables'`, `entity = 'friday'`

Alle ausführbaren Actions werden dabei in one-stage und two-stage Actions eingeteilt:

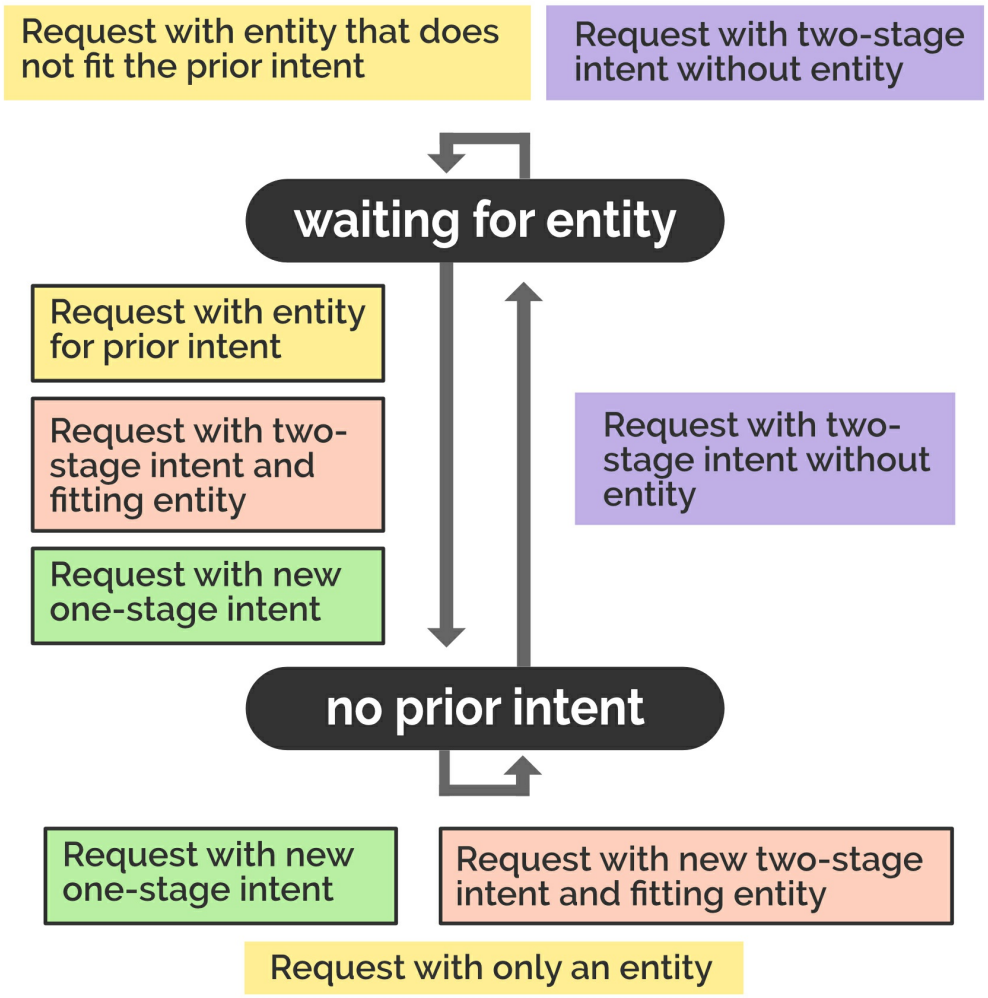
- `one-stage:` Benötigt nur einen Intent und keine Entity, **Beispiel:** Action Joke.
- `two-stage:` Benötigt nur sowohl Intent und als auch Entity, **Beispiel:** Action Timetables.

Möchte der Nutzer eine one-stage Action ausführen, so ist das möglich sobald im Satz, der klassifiziert werden soll, ein Intent erkannt wird, der zur entsprechenden one-stage Action führt. Möchte der Nutzer allerdings eine two-stage Action ausführen, so muss sowohl ein Intent, als auch eine Entity angegeben werden. Dies kann ein einem Satz (**Beispiel:** '*Zeige mir*

meinen Stundenplan am Mittwoch'), als auch in zwei Sätzen geschehen (**Beispiel:** '*Zeige mir meinen Studenplan*', '*An welchem Tag möchtest du deine Stunden sehen?*', '*Am Mittwoch*').

Zustände und mögliche Events

Um beide Fälle abdecken zu können notiert sich der Conversation-Service einen sog. **prior-intent**, der angibt, ob es noch unvollständige, two-stage Actions gibt, die noch nicht vollendet wurden. Je nachdem, ob dieser unvollendete Intent existiert, reagiert der Conversation-Service auf unterschiedliche Weise. Die Funktionsweise in Abhängigkeit vom Zustand der Conversation lässt sich wie folgt zusammenfassen:



Zustände und Events der Konversationslogik.
Alle umrandeten Events führen zur Ausführung einer Action.

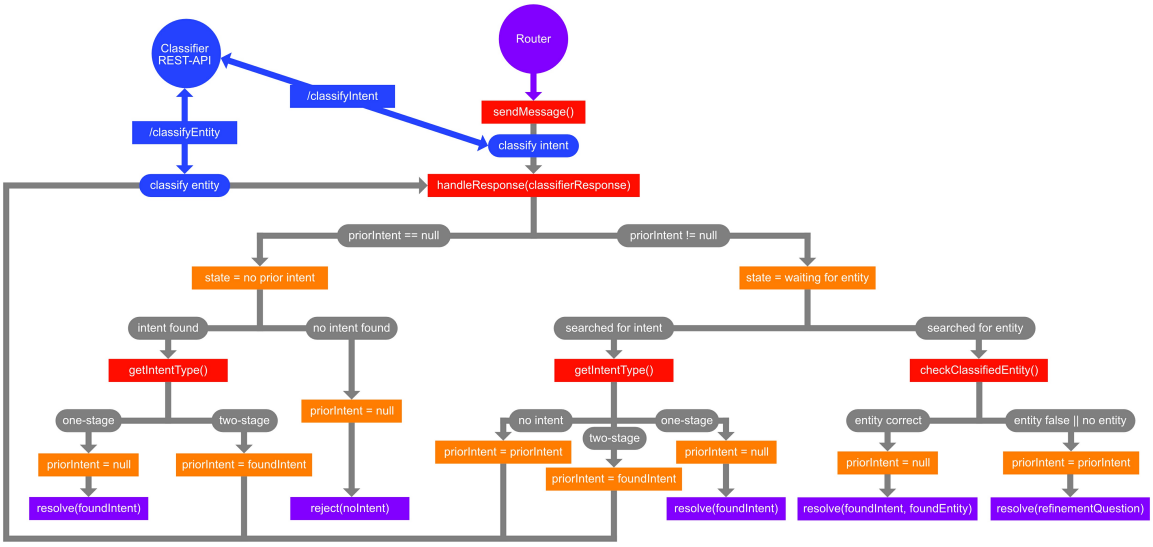
Die Information, in welchem Zustand sich die Konversation eines Nutzers befindet, wird dabei in einem **Kontext-Objekt** hinterlegt, mit dem der Konversationspartner eindeutig indetifiziert werden kann. Das Objekt enthält sowohl eine **conversation-id** für jeden Konversationspartner,

als auch den Namen des **prior-intent**. Da dieser ausschlaggebend für das Ergebnis des Conversation-Services ist, muss der Context bei jedem Request von der Oberfläche mitgesendet werden.

Bearbeitung einer Anfrage

Beginnt der Nutzer durch die Oberfläche eine neue Konversation, so wird ein neuer Kontext für den Nutzer erstellt. Dieser wird, wie bereits beschrieben, verwendet, um den Zustand und eine Identifikationsmöglichkeit jeder Konversation mit jedem Nutzer zu speichern.

Auf Basis der beiden möglichen Zustände und der möglichen Events, lässt sich Ablauf einer Konversation etwas vereinfacht wie folgt zusammenfassen:



Schematische Darstellung der Konversationslogik

Anzumerken ist dabei, dass zuerst immer nach einem neuen Intent gesucht wird, um zu verhindern, dass neue Intents nicht verworfen werden, wenn es noch offene Intents gibt. Andernfalls könnte man eine begonnene Anfrage für einen Two-Stage Intent nicht mehr abbrechen. Die Konversationslogik kann nach Eingabe eines Satzes im Chatfenster auf drei Arten reagieren.

1. `resolve(foundIntent, foundEntity)` , `resolve(foundIntent)` : Es wurden genug Informationen für das Ausführen einer Action gesammelt. Der Router wird darüber informiert und Dieser führt mithilfe des Dispatchers die entsprechende Action aus. Der Nutzer erhält das Ergebnis seiner angeforderten Action.
2. `resolve(refinementQuestion)` : Es gibt einen two-stage Intent, für den es noch keine passende Entity im Satz gefunden wurde. Dem Router wird ein passender Antwortsatz mit einer Frage nach einer Entity übergeben. Der Nutzer hat nun die Möglichkeit, seine Frage

zu ergänzen.

3. `reject(noIntent)`: Der Classifier kann den vom Nutzer eingegebenen Satz nicht klassifizieren. Der Nutzer wird in der Oberfläche aufgefordert, seine Anfrage neu zu formulieren.

In jedem Fall wird eine der genannten Rückgabewerte für den Router so aufbereitet, dass diese in ihrer Form der Antwort des Bluemix Conversation-Services entspricht.

Weiterentwicklung und Ergänzung des Projektes

Aufbau und Bestandteile

Alle Teile des Projektes befinden sich im Ordner `/classifier-based-conversation` . Das Projekt ist in mehrere Dateien aufgeteilt. Im folgenden eine Auflistung der Dateien und deren Funktionen:

Datei	Funktion
conversation.js	Enthält die grundlegende Konversationslogik und steuert den Ablauf der Konversation
conversationContext.js	Funktionalitäten zur Erstellung eines neuen Kontextes für einen Nutzer
conversationDebugPrints.js	Funktionalitäten zur Erstellung von Logs, die als Unterstützung für die Fehlersuche nützlich sind
conversationObjects.js	Funktionalitäten zum Bauen von JSON-Objekten wie dem Request-Body für den Classifier
conversationUtils.js	Verschiedene Funktionen
intents.js	Sammlung von Intents, deren Entites und Refinement-Questions.
mockedClassifier.js	Funktionen die die in den Router-Tests verwendeten Fragen klassifizieren. Kann verwendet werden, wenn der Classifier Rest-Service nicht in erreichbar sein sollte

Wechsel zwischen dem Classifier-basierten und dem Bluemix Conversation-Service

Der Wechsel zwischen den beiden Conversation-Services ist ohne großen Aufwand in `Router.js` möglich. Im oberen Teil befinden sich folgende Zeilen, die eine der beiden Dateien `conversation.js` und `classifier-based-conversation/conversation.js` einbindet. Der Wechsel kann einfach durch das Einbinden der anderen Datei vorgenommen werden. Da alle angebotenen Funktionen, die entgegengenommenen Parameter und die Antworten gleich sind, müssen keine anderen Änderungen im Code vorgenommen werden.

```
var dispatcher = require('./dispatcher');
// Use this for Bluemix Conversation-Service
// var conversation = require('./conversation');
// Use this for own Python-Classifer Based Conversation
var conversation = require('./classifier-based-conversation/conversation');
```

Pfad zu intents.js

Testabdeckung

Um den korrekten Ablauf von Konversationen sicherzustellen, werden alle mögliche Eingabeabfolgen durch den Nutzer in `conversation_test.js` abgedeckt. Die Tests müssen mindestens folgende Fälle umfassen:

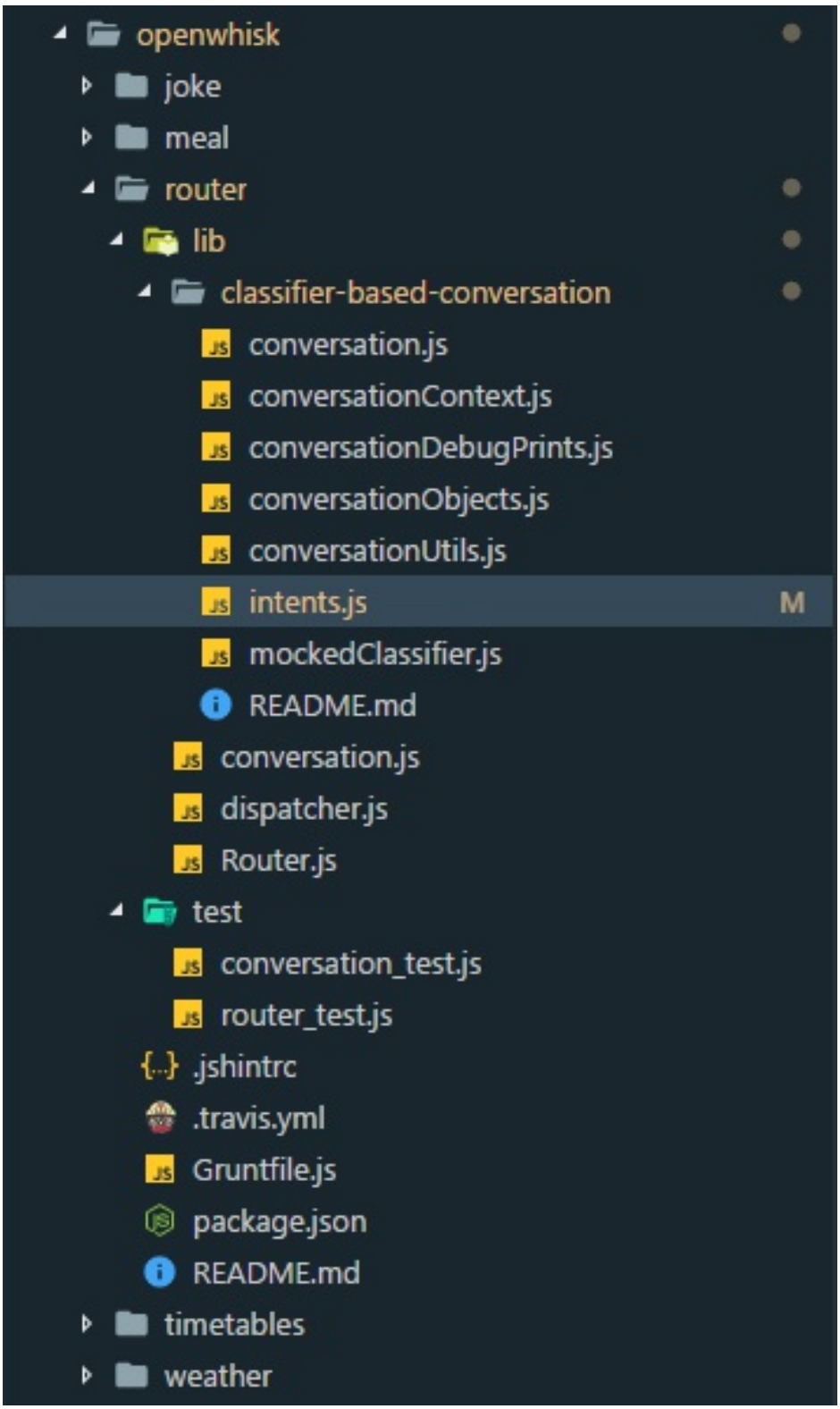
1. einstufige Intents
2. zweistufige Intents mit
    iii. Intent und Entity im selben Satz
    iv. in zwei getrennte Sätze mit Intent im Ersten und Entity im Zweiten
    v. einem Fehler in der Eingabe bei einer nachgereichten Entity, die dazu führen, dass der Satz nicht klassifiziert werden kann. Bei der dritten Anfrage wird dann die korrekte Entity gesendet
3. Unterbrechung eines zweistufigen Intents durch einen eintstufigen Intent. Der zweistufige, nicht vollendete Intent davor darf dann nicht mehr durch Nachreichen einer Entity ausführbar sein

Alle Tests verwenden dabei die Rest-Schnittstelle des Router, die in der selben Weise von der Nutzeroberfläche verwendet wird. Das ermöglicht es nicht nur zu Testen, ob ein Satz richtig behandelt wird, sondern auch, ob die Rückgabewerte an den Router korrekt sind und es ermöglichen die entsprechende Action auszuführen. Erweiterungen der Konversationslogik können hier getestet werden.

Erweiterungen der Logik und damit entstehende, neue Testfälle können hier durch die Simulation einer Konversation hier getestet werden.

Einbinden eines neuen Intents

Ein neuer Intent kann in der Datei `intents.js` angelegt werden. Als Beispiel dient hierzu der Intent **Navigation**.



Pfad zu intents.js

Für jeden Intent muss ein Objekt angelegt werden, das folgenden Inhalt hat:

- Name der Action, die ausgelöst werden soll
- Type (**one-stage** oder **two-stage**)
- Entities, die in der Action unterschieden werden
- Fragen, um den Nutzer aufzufordern, eine Entity anzugeben

```
var navigation = {
    name: 'Navigation',
    type: 'two-stage',
    entities: [
        'Platzhalter, Indizes beginnen in der Action bei 1.',
        'Aula',
        'building E',
        'building F',
        'building LI',
        'building M',
        'building P',
        'building R',
        'cafeteria',
        'main entrance',
        'Mensa'
    ],
    refinementQuestions: [
        'Wohin soll ich dich navigieren?',
        'Wo möchtest du hin?',
        'Wo soll ich dich hinbringen?',
        'Wohin möchtest du?'
    ]
}
```

Definition des neuen Navigations-Intent

Dieses Objekt muss dann noch in ein Array mit allen klassifizierbaren Intents aufgenommen werden. Damit ist der neue Intent und die damit verbundenen Entities für die Konversationslogik verwendbar, falls ein Satz vom Classifier mit dem gegebenen Intent-Namen klassifiziert wird.

```
82
83  var intents = [
84      joke,
85      meal,
86      timetables,
87      weather,
88      navigation
89  ]
90
```

Einbinden des neuen Intent-Objekts

Damit der Classifier den neuen Intent erkennen kann, muss dieser noch mit Trainingsdaten versorgt werden. Alle nötigen Informationen dazu sind in der Dokumentation der Classifier-Restschnittstelle zu finden.