

Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

DIPLOMOVÁ PRÁCE

PLZEŇ, 2020/2021

FILIP POLÁK

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2020/2021

Studijní program: Aplikované vědy a informatika
Forma studia: Prezenční
Obor/kombinace: Kybernetika a řídicí technika (KŘT)

Podklad pro zadání DIPLOMOVÉ práce studenta

Jméno a příjmení: **Bc. Filip POLÁK**
Osobní číslo: **A19N0106P**
Adresa: **Litohlavy 110, Litohlavy, 33701 Rokycany 1, Česká republika**
Téma práce: **Reprezentace stavu hlasových dialogových systémů**
Téma práce anglicky: **State representation for spoken dialog systems**
Vedoucí práce: **Ing. Jan Švec, Ph.D.**
Katedra kybernetiky

Zásady pro vypracování:

1. Nastudujte problematiku reprezentace stavu hlasových dialogových systémů a v práci uveďte přehled různých metod a přístupů k reprezentaci stavu
2. Obdobně nastudujte způsoby reprezentace stavu v moderních webových technologiích, např. React.
3. Analyzujte a navrhnete softwarovou knihovnu jazyka Python sloužící k vhodné reprezentaci stavu hlasového dialogového systému.
4. Knihovnu navrhnete s ohledem na následné řízení dialogu, univerzálnost knihovny demonstujete na jejím užití ve více hlasových dialogových systémech.
5. Zdrojové kódy s dokumentací přiložte jako součást diplomové práce a zpřístupněte jako open-source.

Seznam doporučené literatury:

Rastogi, A., Hakkani-Tur, D., & Heck, L. (2018). Scalable multi-domain dialogue state tracking. 2017 IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2017 – Proceedings, 2018-Janua, 561–568. <https://doi.org/10.1109/ASRU.2017.8268986>

Metallinou, A., Bohus, D., & Williams, J. D. (2013). Discriminative state tracking for spoken dialog systems. ACL 2013 – 51st Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 1, 466–475.

Raux, A., & Eskenazi, M. (2009). A finite-state turn-taking model for spoken dialog systems. NAACL HLT 2009 – Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Proceedings of the Conference, (June), 629–637. <https://doi.org/10.3115/1620754.1620846>

Podpis studenta:

Datum:

Podpis vedoucího práce:

Datum:

PROHLÁŠENÍ

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne

.....

Chtěl bych poděkovat vedoucímu této práce Ing. Janu Švecovi, Ph.D. za jeho vedení a lidský přístup a v neposlední řadě za to, že mě naučil, že nejsložitější řešení někdy nejsou těmi nejlepšími. Dále bych také chtěl poděkovat své rodině, přítelkyni a přátelům za to, že mě v tomto těžkém období podporovali.

Abstrakt

Hlasové dialogové systémy slouží k zprostředkování komunikace mezi člověkem a počítačem. Tato komunikace může ulehčit život nejenom handicapovaným lidem (slepota, tělesné postižení), ale usnadňuje i aktivity běžného života, jako například zarezervování místa v restauraci, chytrá domácnost nebo zakoupení letenky. Pro správnou funkčnost těchto systémů je klíčové správně reprezentovat jejich stav, což je hlavním cílem této diplomové práce. Reprezentaci stavu lze popsat jako představu systému o tom, co uživatel řekl a co je jeho cílem. V práci jsou uvedeny různé metody a přístupy k reprezentaci stavu. Poté následuje analýza a návrh softwarové knihovny jazyka Python sloužící k vhodné reprezentaci hlasového dialogového systému, jejíž funkčnost je pak demonstrována na několika hlasových systémech.

Klíčová slova

rozpoznání řeči, porozumění řeči, dialogový manažer, generování odpovědi, text do řeči, neuronové sítě, reprezentace stavu

Abstract

Spoken dialogue systems offer a possibility of communication between a person and a computer. The communication can make a life easier not only for handicapped people (with blindness, physical disability) but also for people in their everyday lives, e.g. making a reservation in a restaurant, smart homes or buying a plane ticket. If the systems have to function correctly, a proper state representation is required, which is the main goal of this thesis. A state representation can be described as an idea of the system of what the user said and what his goal is. A thesis states several current methods used for state representation. A Python state representation library is then analyzed, implemented and its functionality is then demonstrated on a number of voice dialogue systems.

Keywords

speech recognition, spoken language understanding, dialog manager, natural language generation, text-to-speech, neural networks, state representation

Obsah

1	Úvod	1
2	Hlasový dialogový systém (HDS)	3
2.1	Rozpoznání řeči - Automatic Speech Recognition (ASR)	5
2.2	Porozumění řeči - Spoken Language Understanding (SLU)	7
2.2.1	Statistický přístup	8
2.2.2	Znalostní přístup	9
2.3	Dialogový manažer - Dialog Manager (DM)	9
2.3.1	Podle iniciativy	9
2.3.2	Podle struktury dialogu	10
2.3.3	Řízení dialogu s konečným počtem stavů	10
2.3.4	Řízení dialogu založené na rámcových strukturách	11
2.3.5	Promluvy v hlasových dialogových systémech	14
2.3.6	Potvrzování v hlasovém dialogovém systému	15
2.3.7	Uzavřené výzvy	15
2.3.8	Otevřené výzvy	15
2.3.9	Explicitní potvrzování	16
2.3.10	Implicitní potvrzování	16
2.4	Generování odpovědi - Natural Language Generation (NLG) a Převod textu na řeč - Text-to-speech (TTS)	16
3	Metody a přístupy k reprezentaci stavu	17
3.1	Discriminative state tracking for spoken dialog systems	17
3.1.1	Úvod	17
3.1.2	Reprezentace stavu	17
3.1.3	Učení	19
3.1.4	Poznátky k přístupu	20
3.2	Scalable multi-domain dialogue state tracking	21
3.2.1	Úvod	21

3.2.2	Reprezentace stavu	21
3.2.3	Učení	22
3.2.4	Poznatky k přístupu	23
4	Reprezentace stavu v moderních webových technologiích	24
4.1	React	24
4.2	Persistent	24
5	Platforma SpeechCloud	25
5.1	Architektura platformy SpeechCloud	25
5.1.1	Hlasový klient	25
5.1.2	Vizuální klient	25
5.1.3	DM	26
5.1.4	SpeechCloud API Server	26
5.1.5	SIP ústředna	26
5.1.6	Worker	26
6	Platforma Mycroft	28
6.1	Skills	29
6.2	Demo Skill	29
6.2.1	Intent	30
6.2.2	Dialog	31
6.2.3	Testování	32
6.3	Architektura platformy Mycroft	33
6.3.1	Wake Word	33
6.3.2	ASR	33
6.3.3	SLU	34
6.3.4	NLG, TTS	34
7	Návrh knihovny pro reprezentaci stavu pro Mycroft	35
7.1	Mycroft a reprezentace stavu	35

7.1.1	Inicializace stavové reprezentace	35
7.1.2	Přidání nových nepotvrzených hodnot	36
7.1.3	Získání stavu dialogu	37
7.1.4	Potvrzení hodnot slotů	37
7.1.5	Smazání reprezentace stavu	37
7.1.6	Změna stavu uživatele	38
7.1.7	Změna stavu úlohy	38
7.1.8	Získání (a řešení nekonzistentních) slotů	38
7.2	Výsledný kód a jeho zhodnocení	39
8	Analýza a návrh obecné knihovny pro reprezentaci stavu	41
8.1	Základní myšlenka knihovny	41
8.2	Strukturizace knihovny	41
8.2.1	Slot - třída Slot	41
8.2.2	Hodnota slotu - třída Value	42
8.2.3	Kompozitní stav - třída State	43
8.2.4	Historie - třída History	43
8.3	Úpravy knihovny během vytváření	44
8.3.1	Využití callbacků a ukládání historie	45
8.4	Výsledný kód	46
9	Demonstrace užití knihovny na různých dialogových systémech	48
9.1	Karty - Mycroft	49
9.1.1	Příklad 1	51
9.1.2	Příklad 2	53
9.1.3	Příklad 3	55
9.1.4	Příklad 4	58
9.2	Karty - SpeechCloud	60
9.2.1	Příklad 1	61

9.2.2	Příklad 2	63
9.2.3	Příklad 3	66
9.2.4	Příklad 4	68
9.3	Vlakové spojení - SpeechCloud	70
9.3.1	Příklad 1	71
9.3.2	Příklad 2	73
9.3.3	Příklad 3	77
9.3.4	Příklad 4	80
9.4	Možná vylepšení knihovny	84
10	Závěr	86
11	Literatura	89

Seznam obrázků

1	Schéma hlasového dialogového systému	3
2	Spojení uživatele, hlasového agenta a řešené úlohy	4
3	Rozpoznání řeči	5
4	Porozumění řeči	7
5	Řízení dialogu s konečným počtem stavů	11
6	Algoritmus řízení dialogu	13
7	Přehled reprezentace stavu[11]	19
8	Struktura platformy SpeechCloud[19]	27
9	Mark II[23]	28
10	Zakomponování Skills do Mycroftu	29
11	Ukázka zdrojového kódu knihovny	40
12	Ukázka zdrojového kódu knihovny	47
13	Ukázka zdrojového kódu dialogu	50
14	Ukázka zdrojového kódu dialogu	60
15	Ukázka zdrojového kódu dialogu	70

1 Úvod

Komunikace (z latinského *communicare* - sdílet) je proces, kdy si dvě entity předávají informace za účelem splnění určitého cíle. Komunikace, a především ústní komunikace, je základním pilířem každé vyspělé společnosti. O tom, kdy spolu lidé začali komunikovat slovy a ne jenom gesty, se stále vedou debaty. První předpoklady [4] zasazovaly začátek ústní komunikace do období cca 50 000 let před naším letopočtem, kdy se na světě už několik tisíc let vyskytoval moderní člověk homo sapiens. Novější výzkumy s pomocí skenu mozků [1] ovšem ukazují, že existuje korelace mezi výrobou nástrojů a vytvářením slov. Lidé proto spolu mohli komunikovat pomocí slov již před 2 miliony let, kdy se po afrických savanách procházeli zástupci druhu homo erectus.

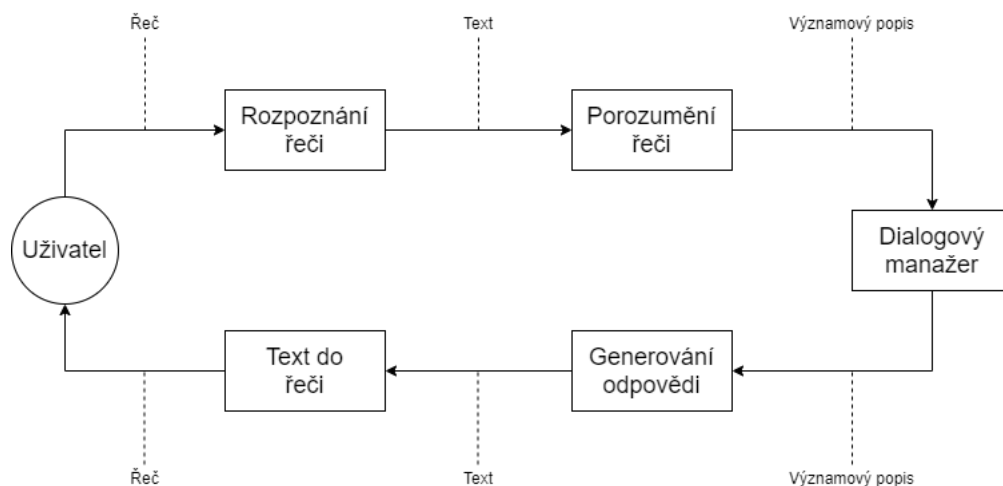
Hlasové dialogové systémy přesouvají verbální komunikaci z domény člověk-člověk do domény člověk-počítač. Od nastavování budíku či zjištění informací o počasí po asistované učení matematiky nebo zakoupení letenky, hlasové dialogové systémy jsou omezené pouze výpočetní náročností, která se v dnešní době pojí s jejich implementací. Za několik desetiletí bude možné realizovat autopiloty jako ve Star Treku nebo operační systémy jako Jarvis v obleku Iron Mana. Nezávisle na doméně jsou hlasové dialogové systémy užívány z jednoduchého důvodu a to v každodenních operacích mít možnost mnohonásobně rychlejší alternativy. Místo toho, aby člověk zašel stáhnout žaluzie, stačí, aby to řekl chytré domácnosti. Místo toho, aby člověk musel zdlouhavě prohlížet výkony svých akcí, stačí, aby se na jejich stav zeptal hlasového dialogového systému, kterému by pak mohl přikázat, aby je případně prodal, nebo ještě podržel. Hlasové dialogové systémy mohou být také využívány jako pomoc lidem s tělesným postižením, například slepotou nebo ochrnutými končetinami.

Ačkoliv je komunikace pomocí řeči součástí každodenního života všech lidí, je důležité si uvědomit, že s sebou přináší i řadu úskalí. Jedním z nich je například jazyková bariéra, která na rozdíl od gest brání komunikaci dvou lidí z opačných konců zeměkoule, a není pak překvapivé, že většina hlasových dia-

logových systémů nabízí možnosti alespoň těch nejrozšířenějších světových jazyků. Dalším může být tzv. sériová povaha předávaných informací, kdy se kvůli dávkové povaze komunikace mohou ztratit potřebné informace. V neposlední řadě je dalším neduhem ústní komunikace omezení možnosti při přeskokování nedůležitých informací, které obírají o drahocenný čas jak lidi samotné, tak i hlasový dialogový systém. Jedním z řešení těchto problémů je využití tzv. multimodálního přístupu k návrhu dialogu, který tyto problémy kompenzuje a nebo je plně odstraní, kdy uživatel či hlasový agent mohou například využít ke komunikaci kromě hlasu i text.

Úkolem této práce je vytvoření softwarové knihovny, která by zprostředkovávala reprezentaci stavu hlasového dialogového systému bez ohledu na to, kde by byl systém použit. Reprezentace stavu představuje to, co hlasový agent porozuměl uživateli. Toto porozumění je klíčové z pohledu řízení dialogu, protože se od něj odvíjí samotná konverzace. Proto je reprezentace stavu pro architekty jedním z hlavních problémů k řešení při návrhu hlasových dialogových systémů.

2 Hlasový dialogový systém (HDS)



Obrázek 1: Schéma hlasového dialogového systému

Obrázek 1 představuje schéma hlasového dialogového systému. **Uživatel** promluví, může jít o odpověď nebo otázku, provede se **rozpoznání řeči**, kde se řeč převede na text, ten je poté vstupem bloku **porozumění řeči**, jehož výstupem je sémantický tvar textu neboli strojová reprezentace textu. Podle té řídí **dialogový manažer** samotný dialog a popřípadě řízenou úlohu a generuje významový popis, který je **generátorem odpovědi** převeden do textové odpovědi, která je poté **převedená do řeči** a prezentována uživateli, který na ni může podle potřeby reagovat.

Důležitým faktem, který je třeba zdůraznit, je, že hlasový dialogový systém se skládá ze "spojení" uživatele, hlasového agenta a řešené úlohy. Toto spojení lze pozorovat na schématu 2.

Jak uživatel, tak i agent a řešená úloha, mají svůj vnitřní stav, jejichž kompozice tvoří **stav hlasového dialogového systému**.

Stav HDS se zapisuje jako:

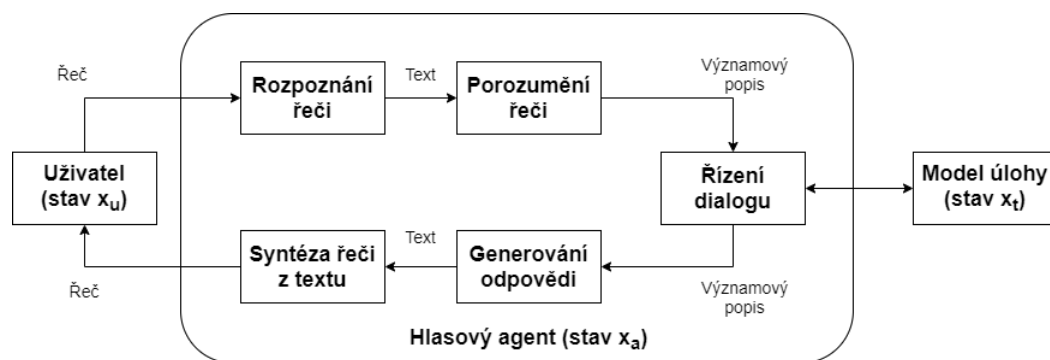
$$\text{Stav HDS} = [\text{stav uživatele, stav agenta, stav úlohy}] \quad (1)$$

Stav uživatele představuje cíl a motivaci, se kterými uživatel interakci s agentem začal. Obsahuje také ale i mentální rozpoložení a zkušenosti uživatele s HDS, jelikož tyto informace mohou pomáhat s přizpůsobením uživateli tzv. na míru.

Stav agenta reprezentuje veškerou komunikaci od začátku do konce hlasového dialogu, tj. historii dialogu, která se s každou odpovědí uživatele nebo agenta aktualizuje, čímž se aktualizuje samotný stav agenta. Díky této historii je u sofistikovanějších HDS možné parametrizovat jeho části (rozpoznání řeči - ASR, porozumění řeči - SLU, generování řeči - NLG, převod textu na řeči - TTS).

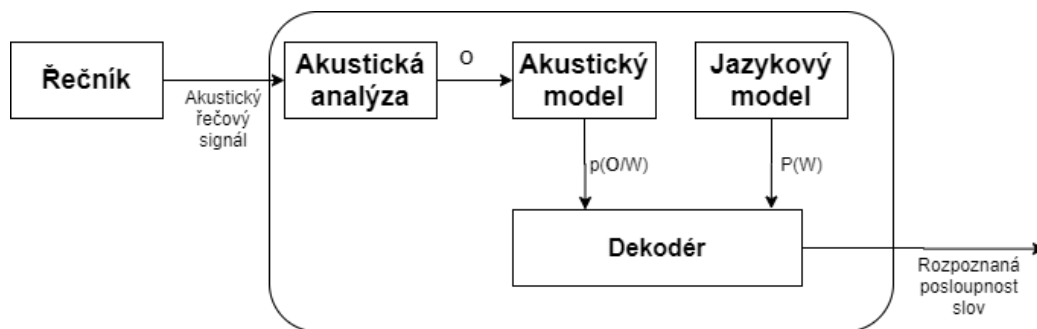
Stav úlohy říká, v jaké fázi se aktuálně úloha nachází. Jelikož jsou úlohy v reálném světě rozsáhlé a složité, nejsou modelovány přímo v řízení dialogu, ale jsou pouze pozorovány nepřímou pomocí řízení úlohy (např. databázový dotaz) a výstupu úlohy (odpověď z databáze).

O moderních přístupech k reprezentaci stavu se píše v kapitole Metody a přístupy k reprezentaci stavu.



Obrázek 2: Spojení uživatele, hlasového agenta a řešené úlohy

2.1 Rozpoznání řeči - Automatic Speech Recognition (ASR)



Obrázek 3: Rozpoznání řeči

Rozpoznání řeči je převod řeči, respektive řečového signálu, na text a pracuje podle následující rovnice 2:

$$W^* = \arg \max_W P(W|A) = \arg \max_W \frac{P(W, A)}{P(A)} = \arg \max_W \frac{P(A|W) \cdot P(W)}{P(A)} \quad (2)$$

kde A vyjadřuje posloupnosti vektorů pozorování neboli posloupnosti akustických příznaků, W vyjadřuje posloupnost slov, $P(A|W)$ vyjadřuje akustický model, $P(W)$ vyjadřuje jazykový model, $\arg \max_W$ vyjadřuje prohledávací strategii přes možné posloupnosti slov a W^* vyjadřuje posloupnost slov, která řečovému signálu nejlépe odpovídala.

Výstupem rozpoznání řeči je posloupnost slov, která měla podle analýzy největší shodu s původním řečovým signálem a nazývá se **1-best hypotéza**. Ostatní posloupnosti slov, které se s původním řečovým signálem neshodovaly úplně, se ovšem nevyřazují, ale je možné je také využít. Z rozpoznávání řeči se pak vybírá **n-best hypotéz**, kde n odpovídá počtu posloupností slov, které postupují do další analýzy.

Akustický model (Acoustic Model - AM) modeluje pravděpodobnost pozorování A při hypotéze W , $P(A|W)$. Zjednodušeně - jaká je

pravděpodobnost akustického příznaku či posloupnosti akustických příznaků A , pokud bylo řečeno určité slovo či posloupnost slov W . Model je trénován z dvojic *akustický signál - posloupnost akustických jednotek*, kde jsou akustické jednotky fonémy, grafémy, slabiky atd. a mění se podle typu modelu. Historicky byly akustické modely modelovány jako Gaussovské směsi představující pravděpodobnost pozorování, následovány skrytými Markovskými procesy (Hidden Markov Model). Aktuálně jsou nejčastěji využívány hluboké neuronové sítě ve spojení s HMM.

Jazykový model (Language Model - LM) modeluje apriorní pravděpodobnost posloupnosti slov W , k čemuž využívá n -gramové jazykové modely, kde **n-gram** představuje n po sobě jdoucích slov.

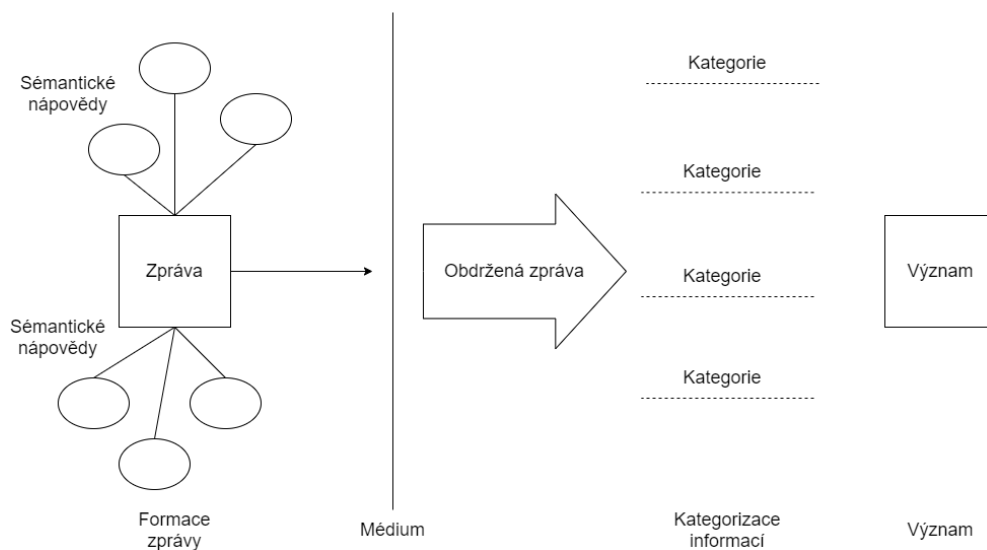
N -gramové jazykové modely jsou vyjádřeny pomocí rovnice 3:

$$\begin{aligned}
 P(W) = P(\omega_i, \omega_{i-1}, \omega_{i-2} \dots \omega_0) &= \prod_{i=0}^n P(\omega_i | \omega_{i-1}, \omega_{i-2} \dots \omega_0) \approx \\
 &\approx \prod_{i=0}^n P(\omega_i | \omega_{i-1}, \omega_{i-2} \dots \omega_{i-N+1})
 \end{aligned}
 \tag{3}$$

Rovnice 3 vyjadřuje, jaká je pravděpodobnost slova ω_i , pokud mu předchází posloupnost slov $\omega_{i-1}, \omega_{i-2}, \dots$, což lze aproximovat pomocí pouze N předchozích slov.

Jazykové modely jsou v dnešní době taktéž založeny na neuronových sítích a jsou trénovány z rozsáhlých textových korpusů.

2.2 Porozumění řeči - Spoken Language Understanding (SLU)



Obrázek 4: Porozumění řeči

V bloku porozumění řeči se řeč v podobě textu, která přišla z ASR bloku, převádí na tzv. významový popis řeči. Jde o převod lexikální podoby řeči do strojové reprezentace, jež je srozumitelná pro počítač.

Porozumění rozpoznává globální význam promluvy, tzv. o sémantické koncepty, a lokální význam, tzv. sémantické entity. Sémantické koncepty vyjadřují, zdali je celá promluva nebo její část souhlasem/nesouhlasem, otázkou/odpovědí atd. Sémantické entity se zaměřují na jednotlivá slova či několik slov, například jedná li se o čas, datum, místo atd. Stejně jako v ASR je možné využít n-best hypotéz, díky čemuž se může zvýšit přesnost porozumění, jelikož se generuje několik alternativních hypotéz.

K modelování porozumění řeči se klasicky přistupuje dvěma způsoby, statisticky a znalostně. Tomu, než se ale jeden z těchto přístupů zvolí, předchází

tzv. 0. krok, kde musí dojít k definici domény, ve které bude dialog probíhat, a doménově závislých konceptů a entit.

K definici domény patří pravděpodobně ještě důležitější krok, a tím je definice uživatele, což může znít po definici domény jako nadbytečná a nepotřebná věc. Pokud přeci pracuje hlasový agent v určité doméně a je tím pádem omezen na velmi specifický obor otázek a odpovědí, není pak tolik podstatné, kdo je uživatelem a jak na hlasového agenta mluví. Naprostý opak je ovšem pravdou. Uživatel je nejdůležitější částí hlasového dialogového systému už jen z toho důvodu, že celá konverzace závisí na jeho otázkách a odpovědích. Proto je podstatné, aby byl v hlasovém dialogovém systému správně nadefinován uživatel a jeho možný způsob mluvy. Tato definice může být provedena i pouze mentálně v hlavě konstruktéra systému, který se musí zamyslet, pro jaké uživatele svůj hlasový systém konstruuje. Jaký obor otázek bude systém "ochotný" zpracovávat, jestli bude zpracovávat pouze spisovnou, nebo i hovorovou řeč, jestli bude chtít po každé operaci potvrzení uživatele a tak dále. Iron Manův oblek bude mít jistě jiného uživatele než uživatele zajímajícího se pouze o hlasové ovládání hlasitosti rádia.

S tím souvisí, že se musí nadefinovat možný tvar vstupu od uživatele a tvar výstupu agenta, jež jsou dále použity v dialogovém manažeru.

2.2.1 Statistický přístup

Největší výhodou statistického přístupu je možnost modelovat a postihnout většinu promluv včetně neviděných. Stačí k tomu pouze velké množství anotovaných dat, která jsou povětšinou nasbírána z reálného světa a jimiž je tím pádem jazyk definován. S těmito daty se provádí trénování statistického přístupu. Trénování je prováděno buď na generativních, kde se modeluje $\arg \max_A P(A|W) = \arg \max_A P(W|A)P(A)$ pomocí dekompozice na model $P(A)$ a $P(W|A)$, nebo diskriminativních modelech, kde se provádí přímé modelování $\arg \max_A P(A|W)$.

2.2.2 Znalostní přístup

Hlavním rozdílem oproti statistickému přístupu je to, že jazyk promluv je omezen gramatikou. Pokud jazyk promluvy neodpovídá předem definované gramatice, pak v něm nelze určit sémantické objekty. Znalostní přístup se pak může využívat buď k popisu celé promluvy, nebo pouze popisu sémantických entit.

2.3 Dialogový manažer - Dialog Manager (DM)

Dialogový manažer (v angličtině Dialog Manager/Management) je ve své podstatě mozkiem celého hlasového dialogové systému, jelikož řídí konverzaci z pohledu hlasového agenta. Dialogový manažer pozoruje stav uživatele prostřednictvím významového popisu řeči a na základě tohoto pozorování a aktuálního stavu agenta generuje významovou reprezentaci odpovědi, která je poté v dalších krocích převedena na řeč. Dialogový manažer ale negeneruje jenom samotnou odpověď, ale může ovlivňovat řešenou úlohu (zapne/vypne světlo, zvýší/sníží hlasitost, posune robota, stáhne si data z databáze) a také stav agenta. Typy DM se dělí podle iniciativy, struktury a způsobu řízení.

2.3.1 Podle iniciativy

Iniciativa hlasového agenta je takový způsob řízení dialogu, kdy agent pokládá otázky a nabízí uživateli způsob řešení úlohy. **Iniciativa uživatele** je řízení založené na příkazech od uživatele, který jimi přímo řídí agenta a řešenou úlohu. Kombinací obou předešlých přístupů je **smíšená iniciativa**, kdy může v jakékoliv fázi dialogu převzít iniciativu jedna či druhá strana. Uživatel například položí otázku (iniciativa uživatele), hlasový agent se může doptat na chybějící informace a prezentuje výsledek uživateli (iniciativa agenta), uživatel se může ptát dále či s výsledkem jinak pracovat (iniciativa uživatele).

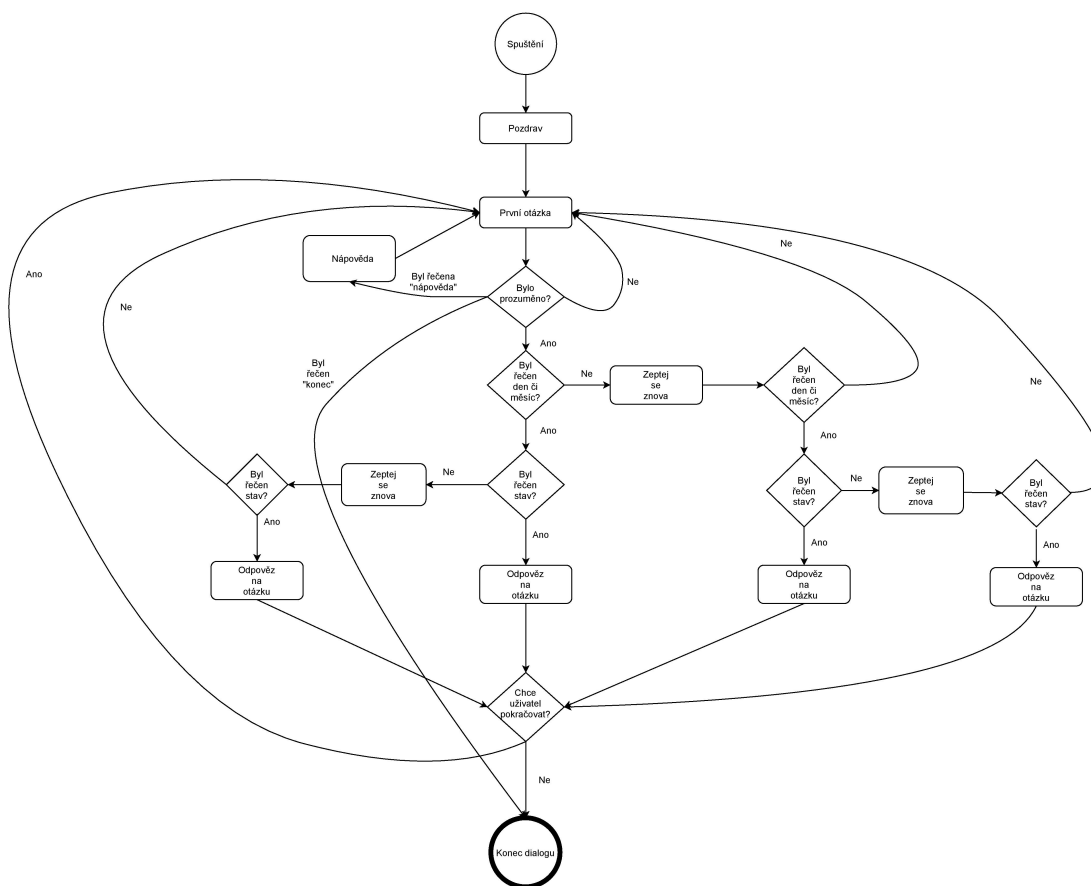
2.3.2 Podle struktury dialogu

Turn based dialog je, jak název napovídá, řízení dialogu prováděné po krocích či obrátkách (turn). Agent vždy čeká na úplný vstup uživatele (nemá možnost tzv. barge-in - skoku do řeči) a až poté vygeneruje odpověď a odprezentuje ji uživateli. Jednu dialogovou obrátku v tomto přístupu představuje dvojice otázka uživatele + odpověď agenta. Stav hlasového dialogového systému se mění pouze v diskrétních krocích.

Přístup podobnější reálným konverzacím mezi lidmi je **inkrementální dialog**, kde agent průběžně zpracovává vstup uživatele a může na něj v případě nejednoznačnosti reagovat neboli má možnost provést barge-in, jež má samozřejmě možnost provést i uživatel. V teorii hlasových dialogových systémů se stav, kdy uživatel nebo agent mluví, nazývá, že uživatel nebo agent drží *floor*.

2.3.3 Řízení dialogu s konečným počtem stavů

Dialog je v tomto přístupu reprezentován jako orientovaný graf. **Uzly** představují stavy dialogu a **přechody** mezi uzly představují změnu stavu dialogu podmíněnou vstupem uživatele.



Obrázek 5: Řízení dialogu s konečným počtem stavů

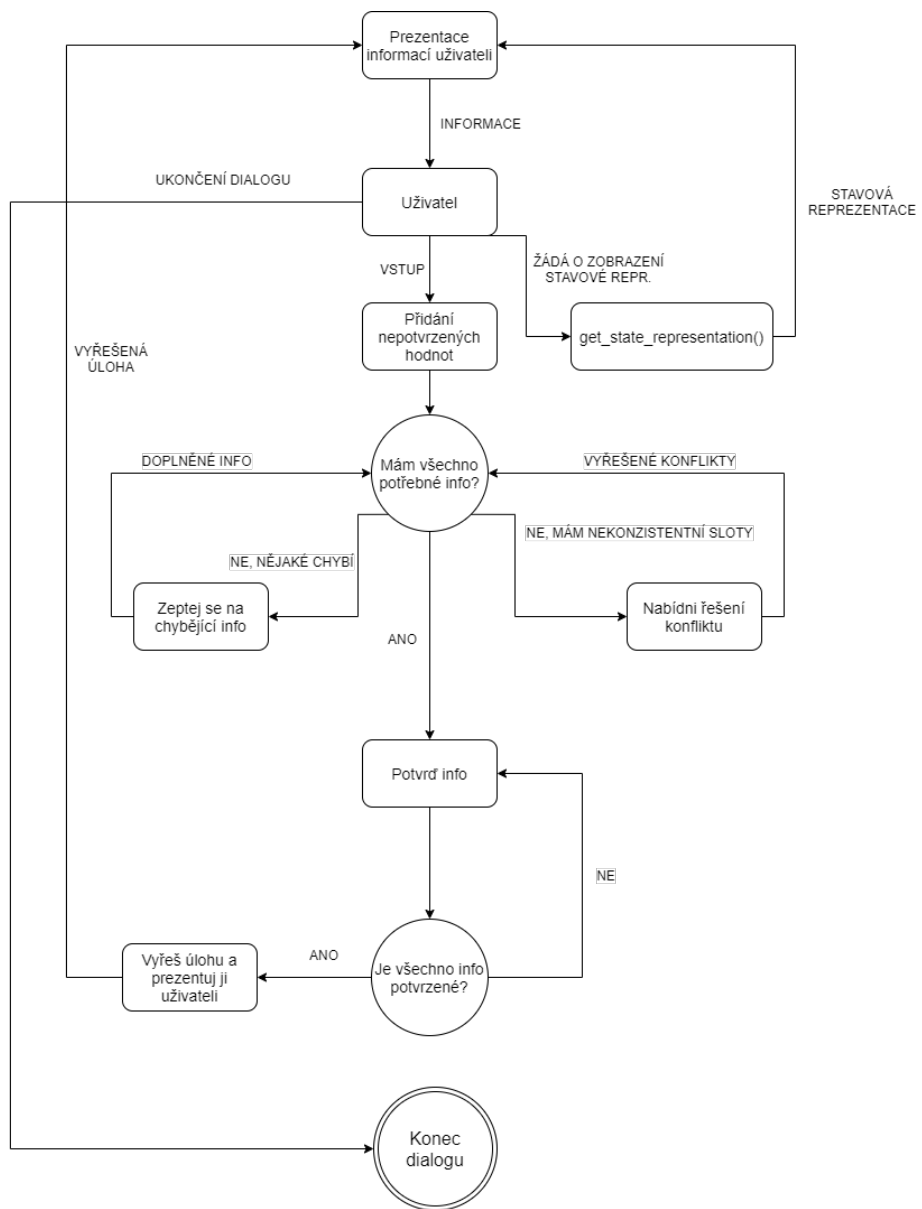
2.3.4 Řízení dialogu založené na rámcových strukturách

Stav dialogu se v tomto přístupu nazývá **rámec** a lze ho v programovacích jazycích reprezentovat např. jako slovník nebo pole, kde každá položka pole/slovníku představuje jeden **slot**. Slot je vyplněn jednou nebo více hodnotami a ke každému slotu může ještě příslušet info o potvrzení a důvěra (confidence) systému v to, jestli bylo řečeno to, co si systém uložil. Slot je na základě přiřazených hodnot:

- **nevyplněný** - nemá hodnotu
- **nepotvrzený** - hodnota nebyla potvrzena uživatelem
- **potvrzený** - hodnota byla potvrzena uživatelem
- **nekonzistentní** - více možných hodnot

Každý slot prochází následujícím procesem řízení dialogu:

1. Vyber povinné **nevyplněné sloty** a dotaž se na jejich hodnoty.
2. Vyber **nepotvrzené sloty** a vyžádej si potvrzení.
3. Vyber **nekonzistentní sloty** a nabídni uživateli řešení nekonzistence.
4. Z hodnot **potvrzených slotů** sestav řízení úlohy a na základě výstupu úlohy vyplň sloty.
5. Prezentuj vybrané sloty uživateli.



Obrázek 6: Algoritmus řízení dialogu

2.3.5 Promluvy v hlasových dialogových systémech

Při komunikaci s hlasovým dialogovým systémem se odpovědi a otázky uživatele či agenta rozdělují na:

- **vyzvání** druhé strany k předání informace a může být i doplněno, jaký druh informace se od druhé strany očekává:
 - "Pro **kolik** lidí chcete zarezervovat stůl?"
 - "Do jaké **destinace** chcete koupit letenku?"
 - "Na **kdy** chcete nastavit budík?"
- **potvrzení** umožňující zjištění správnosti předané informace:
 - "Chcete rezervovat stůl pro **dva**, v **čínské** restauraci, v **20:00**?"
 - "Chcete objednat letenku z **Praha** do **Berlín** v **pondělí**?"
 - "Chcete nastavit budík na **7:00**?"
- **zotavení z chyby** pro opravení nebo opakování vstupu z důvodu nesprávně předané informace:
 - "Omlouvám se, nerozuměl jsem Vám."
 - "Prosím, zopakujte Vaši odpověď."
 - "Neslyšel jsem, do jaké **destinace** chcete koupit letenku."
- **opětovná pobídka**, pokud byla očekávána promluva, ale ta nebyla předána:
 - "Řekněte mi, prosím, pro **kolik** lidí chcete zarezervovat stůl."
 - "Neřekl jste, do jaké **destinace** chcete koupit letenku."
 - "Na **kdy** chcete nastavit budík?"
- **dokončení** pro předání požadované informace:

- "Děkuji za potvrzení. Hledám volný stůl pro **dva**, v **čínské** restauraci, v **20:00**."
- "Letenka z **Praha** do **Berlín** v **pondělí** byla zakoupena."
- "Budík byl nastaven na **7:30**."

2.3.6 Potvrzování v hlasovém dialogovém systému

Jelikož uživatel ani hlasový dialogový systém nejsou schopni pozorovat své stavy přímo, ale pouze prostřednictvím převodu stavu na řeč a prostřednictvím přenosu řečového kanálu, je důležité, aby hlasový agent udržoval uživatele "v obraze" ohledně svého vnitřního stavu, díky čemuž si může uživatel vytvořit mentální obraz toho, jak hlasový agent funguje a co od uživatele potřebuje. Toho je dosaženo pomocí výzev a potvrzování.

2.3.7 Uzavřené výzvy

Uživatel může volit pouze z nabízených možností.

- "Pro zrušení účtu řekněte ano, nebo ne."
- "Pro potvrzení rezervace řekněte ano, nebo ne."

2.3.8 Otevřené výzvy

Uživatel může odpovědět libovolně, ačkoliv je jeho možná odpověď omezena seznamem příkladů, které zná i hlasový agent. Agent může uživateli seznam příkladů sdělit několika způsoby.

- **okamžitě** - "Dobrý den, můžete se zeptat na faktury nebo stav účtu."
- **po prodlevě** - "Dobrý den,..." - 3s - "Můžete se zeptat na faktury nebo stav účtu."
- **na vyžádání uživatele** - "Na co se můžu zeptat?"

2.3.9 Explicitní potvrzování

Hlasový agent očekává od uživatele odpověď **ano**, či **ne**.

- "Chcete odstranit seznam skladeb?"
- "Chcete zapnout světlo?"

2.3.10 Implicitní potvrzování

Agentem požadovaná informace je představena jako součást výzvy.

- "Řekněte jméno seznamu skladeb, který chcete odstranit."
- "Řekněte, do jakého typu restaurace chcete udělat rezervaci."
- "Řekněte, na kdy chcete nastavit budík."

2.4 Generování odpovědi - Natural Language Generation (NLG) a Převod textu na řeč - Text-to-speech (TTS)

Generování odpovědi a převod textu na řeč úzce navazují na výstup dialogového manažera, jímž je významový popis řeči. Ten může např. pro potvrzení (confirm) výchozí vlakové stanice a dotaz (request) na cílovou stanici vypadat jako (*confirm, departure_station, "Praha"*) + (*request, arrival_station*). Generování odpovědi tento popis převede např. do textu "Kam chcete jet z Prahy?". Uživatelsky přívětivé jsou také různé styly otázek a jejich průběžné změny, tedy že se agent nebude pouze ptát "Kam chcete jet z Prahy?", ale třeba "Chcete jet z Prahy, kam to bude?", "Chcete jet z Prahy? A kam to dnes bude?". Především v českém jazyce je potřeba dávat velký důraz na správné skloňování, časování, plural a předložky. Takto vygenerovaný text poté vstupuje do bloku převádějící text do řeči, který z něj vygeneruje řeč, jež je pak prezentována uživateli.

3 Metody a přístupy k reprezentaci stavu

3.1 Discriminative state tracking for spoken dialog systems

3.1.1 Úvod

Reprezentace stavu představená v článku [11] přistupuje k problému s využitím statistického přístupu, jmenovitě diskriminativního, představeného v článku [2]. Tento přístup využívá natrénované podmíněné modely přímo odhadující distribuci pravděpodobnosti nad množinou hypotéz, čímž jim přiřazují "confidence score" (skóre důvěryhodnosti). Hlavním problémem tohoto postupu je omezení množství sledovaných hypotéz, což vyústí v nenulovou pravděpodobnost, že bude správný výsledek vyřazen. Cílem článku je představit řešení, kdy bude možné sledovat a ohodnocovat libovolný počet hypotéz a stále vypočítávat správnou distribuci pravděpodobnosti.

3.1.2 Reprezentace stavu

Jak již bylo nastíněno v úvodu této kapitoly, sledování stavu a ohodnocování hypotéz je zprostředkováno diskriminativním způsobem. Ten, jak již bylo řečeno v kapitole Porozumění řeči - Spoken Language Understanding (SLU), obsluhuje primárně část porozumění řeči hlasového dialogového systému, a proto je samotná reprezentace stavu úzce provázána přímo s touto částí.

Každá dialogová obrátka představuje jeden datapoint (jednu množinu dat). Pro každý datapoint se vstup skládá ze tří částí:

- množina K příznaků vzešlých z ASR a SLU popisující aktuální kontext dialogu
- počet G hypotéz stavu dialogu, který se mění v krocích dialogu

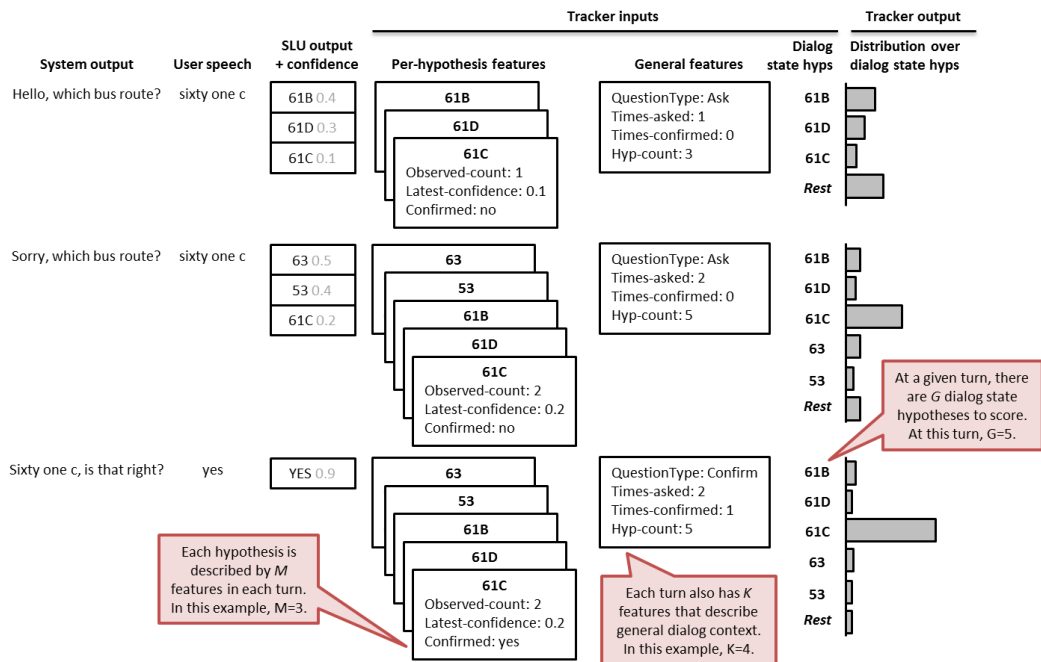
- počet M příznaků popisující každou hypotézu

Jedním z požadavků na G hypotézy je vzájemná disjunkce neboli nemůže dojít k případu, že bude zároveň pravdivá část hypotézy G_1 a část hypotézy G_2 . Cílem je pak přiřadit G hypotézám a meta-hypotéze, která vyjadřuje, že ani jedna z G hypotéz není pravdivá, pravděpodobnost. Na základě nejpravděpodobnější hypotézy poté odpovídá dialog uživateli a celý cyklus se opakuje až do konce dialogu. Vhodnou analogií k tomuto procesu jsou N-best hypotézy z části Rozpoznání řeči - Automatic Speech Recognition (ASR).

Nalezení správné neboli nejpravděpodobnější hypotézy probíhá s použitím upravených **maximum entropy modelů**. Ty ve svém výpočtu používají *hypothesis-specific* (vztahující se k hypotéze) příznaky, které se dělí na tři kategorie:

- *base* - informace o aktuální dialogové obrátce
- *history* - informace o předešlých dialogových obrátkách
- *confusion* - informace o pravděpodobnosti chyby ASR, jako je například hluk pozadí, stejné názvy atd.

a *general* (obecné) příznaky, které zahrnují souhrnné informace o historii dialogu, SLU výsledcích atd.



Obrázek 7: Přehled reprezentace stavu[11]

3.1.3 Učení

Prvním možným způsobem reprezentace stavu je **Fixed-length discriminative state tracking** (sledování stavu s pevnou délkou), který problém definuje jako jednoduchou vícetřídovou klasifikaci. Ze všech G hypotéz je vybrána podmnožina \tilde{G} nejlepších hypotéz ke skórování. \tilde{G} nebere ovšem pouze jenom nejlepší hypotézy z aktuálního kroku, ale dívá se i do minulosti. Nejlepších výsledků bylo dosaženo s rovnicí:

$$\tilde{G} + 1 = G_1 + G_2 + G_3 + 1 \quad (4)$$

a hodnotami:

- $G_1 = 1$ - nejlepší hypotéza v aktuálním kroku t

- $G_2 = 2$ - dvě nejlepší hypotézy v kroku $t - 1$
- $G_3 = 3$ - tři nejlepší hypotézy v kroku $t - 2$
- $\tilde{G} = 6$
- přičtené jedničky na obou stranách rovnice představují metahypotézu

Hlavní výhodou tohoto přístupu je to, že příznaky z jedné třídy (jedné hypotézy) mohou při výpočtu ovlivňovat i ostatní třídy. Nevýhodou, jež převažuje nad výhodami, je pak fakt, že počet vah potřebných k natrénování je $(\tilde{G} + 1) \cdot (\tilde{G}M + K)$, který se kvadraticky zvětšuje v závislosti na zvoleném \tilde{G} . To koresponduje s myšlenkou představenou v úvodu této kapitoly, že kvůli omezení počtu hypotéz, které jsou ohodnocovány, může dojít ke ztrátě té správné.

V článku je proto představen způsob **Dynamic discriminative state tracking** (diskriminativní sledování stavu s dynamickou délkou). Využívá příznakových funkcí, které linkují příznaky specifické pro hypotézu s korespondující hypotézou, čímž je možné se zbavit závislosti množství vah na učení na počtu hypotéz. Dochází zde ke konkatenaci M příznaků a K general příznaků, které jsou stejné pro všechny hypotézy. Tato konkatenace je provedena i pro metahypotézu, kde M má nedefinované hodnoty. Příznaky jsou označeny x_i^g , představující i -tý příznak hypotézy g , kde $i = 1, \dots, M + K$ a $g = 1, \dots, G + 1$. Příznaky mohou být poté dynamicky měněné v každém kroku, v maximum entropy modelech se ovšem trénuje jenom $M + K$ vah, označené jako λ_i .

3.1.4 Poznatky k přístupu

Celý přístup **Discriminative state tracking for spoken dialog systems** jde za hranice této diplomové práce, jelikož kromě sledování a reprezentace stavu hlasového dialogového systému také zprostředkovává část porozumění řeči. Proto z něj bude pro knihovnu na reprezentaci stavu využita

část týkající se zprostředkování dynamického přidávání a smazání libovolného slotu a část použití immutable stavu, jež lze použít pro extrakci příznaků z logů dialogů a natrénování modelu.

3.2 Scalable multi-domain dialogue state tracking

3.2.1 Úvod

State of the art metody reprezentace stavu využívají hluboké neuronové sítě naučené na odhad a reprezentaci jak uživatele a dialogového agenta, tak inicializaci slotů, jejich pojmenování a hodnoty v nich obsažené. Hlavní překážkou tohoto přístupu je nedostatek dostupných dat, z čehož vyplývá limitace báze znalostí dialogového systému. To vyúsťuje v ohraničenou množinu hodnot využitelných pro jednotlivé sloty. Dalším neduhem je nemožnost zachycení entit, které nebyly přítomné během trénování. Cílem představeným v článku [15] je pomocí hlubokých neuronových sítí vytvořit metodu, která by umožňovala mít ve stavové reprezentaci sloty s velkou nebo dokonce neomezenou množinou hodnot.

3.2.2 Reprezentace stavu

Problém omezené množiny hodnot pro sloty je viditelný především u slotů s dynamickou databází (filmy, písničky, atd.) nebo u slotů reprezentujících například čas nebo datum, kde je velmi obtížné dostat seznam hodnot rozumné velikosti. Řešením je použití **candidate set** (množina kandidátů) pro slot. Množina kandidátů se značí C_s^t , kde t odpovídá indexu dialogové obrátky a s představuje slot a je naplněn dvojicemi *hodnota:skóre*. Na začátku dialogu je každá množina kandidátů prázdná a jeho plnění probíhá následovně:

1. Po promluvě uživatele se naplní hodnotami z SLU části.
2. Po promluvě agenta se naplní hodnotami získanými z akcí systému.

3. Kandidáti z předešlé dialogové obrátky se seřadí sestupně podle skóre, množina kandidátů se rozšíří o nové kandidáty až $|C_s^t| \leq K$.

Z experimentů představených v článku vyšlo, že množiny kandidátů málokdy dosáhly maximální kapacity $K = 7$.

Namísto použití množiny V_s (všechny hodnoty pro slot s) je využita množina kandidátů a **samotná reprezentace stavu slotu** je poté definována jako $V_s^t = C_s^t \cup \{\delta_s, \phi_s\}$, kde je C_s^t množina kandidátů pro slot s v dialogové obrátce t , δ_s představuje *dontcare* hodnotu (uživatel nemá pro slot s žádnou preferenci), ϕ_s představuje *null* hodnotu (slot ještě nebyl specifikován).

Aby byla velikost množiny kandidátů konstatní pro všechny dialogové obrátky, přidává se, pokud je to nutné, $K - |C_s^t|$ dummy (prázdných) hodnot (tzv. padding - vyplnění). Z toho vychází $|V_s^t| = K + 2 \ll |V_s^t|$, což znamená, že velikost množiny kandidátů V_s^t se rovná maximální kapacitě $K + 2$. Tato velikost je řádově menší než množina všech hodnot V_s . Připočtená dvojka představuje *dontcare* a *null* hodnoty.

3.2.3 Učení

Učí se diskriminativní model, do kterého vstupuje množina kandidátů pro každý slot a který aktualizuje skóre pro každého kandidáta v množině kandidátů. Je také schopný identifikovat a pracovat s *dontcare* a *null* hodnotami. Model se učí pomocí příznaků, které jsou extrahovány separátní neuronovou sítí. Příznaky jsou rozděleny do tří kategorií:

- *utterance related* (vztahující se k celé promluvě) - relevantní pro všechny sloty, a proto sdíleny přes všechny kandidátní skóre
- *slot related* (vztahující se ke slotu) - relevantní pro určitý slot, a proto sdíleny přes všechny kandidátní skóre obsažené ve slotu
- *candidate related* (vztahující se ke kandidátovi) - relevantní pro určitou hodnotu a nejsou sdíleny s žádným jiným kandidátem

Největší výhodou tohoto přístupu je zjednodušení následujícího problému. Pokud mají být neuronové sítě využívány pro několik různých domén, pak jednou z hlavních nevýhod je potřeba trénovat je pro každou doménu znovu a zvlášť, k čemuž je potřeba obrovské množství označených dat. Tento problém je zde částečně vyřešen, jelikož natrénované váhy pro sloty jsou nezávislé na doméně a mohou být jednoduše přeneseny do domény nové, čímž je omezena nutnost trénovat novou neuronovou síť.

3.2.4 Poznatky k přístupu

Hlavním přínosem metody **Scalable multi-domain dialogue state tracking** pro tuto diplomovou práci je potřeba modularity knihovny na reprezentaci stavu dialogového systému. Tzn., že knihovna by měla být jednoduše využitelná v řadě dialogových systémů bez ohledu na jejich doménu.

4 Re prezentace stavu v moderních webových technologiích

4.1 React

React je JavaScript knihovna vyvíjená Facebookem a primárně slouží jako základ mobilních aplikací, jako je například Instagram. V knihovně využívají jednotlivé komponenty *states* (stavy) pro data měnící se v čase. Stavy jsou využívány pouze v případě, že aplikace přijme vstup od uživatele nebo se změní hodnota na interním čítači, když například uživatel nereaguje. Namísto toho, aby měla každá komponenta aplikace přístup ke všem stavům, existuje **jeden stavový komponent**, který zapouzdřuje veškerou práci se stavy a poté je předává ostatním komponentám. Díky minulému a současnému stavu je stavový komponent schopný odhadnout nový stav a to, co se bude dít v dalším kroku aplikace či dialogu.

4.2 Pyrsistent

Pyrsistent je Python knihovna vytvořená k doplnění základních objektů jako seznam (list) nebo slovník (dictionary). Název knihovny je kombinací slov **Python** a **persistent** (trvalý) a vypovídá dokonale o funkci této knihovny, jež je pouze vytvoření nových objektů, které mají stejné funkce jako ty původní, ale jsou trvalé neboli immutable. Objekt proto zůstane stejný od začátku svého života (inicializace) až po svůj konec (vymazání), nelze z něj žádné prvky odebrat nebo do něj nějaké přidat. Nové immutable objekty vznikají ze stávajících immutable objektů pouze pomocí operací definovaných v knihovně. Při použití immutable objektů je designér donucen udržovat ve stavu všechny důležité hodnoty a data. Immutable objekty jde jednoduše ukládat na disk a využít je i například k trénování neuronových sítí. Díky těmto vlastnostem je Pyrsistent ideální k reprezentaci stavu v prostředí Python.

5 Platforma SpeechCloud

Pro seznámení se s prostředím dialogů a reprezentace stavu byla použita platforma SpeechCloud a platforma Mycroft představená v následující kapitole Platforma Mycroft.

SpeechCloud je řečová platforma vytvořena ve spolupráci katedry kybernetiky Fakulty aplikovaných věd Západočeské univerzity a společnosti Speech-Tech. Platforma byla vyvinuta pro nasazení v cloudu, díky čemuž je možné výsledný dialogový systém škálovat podle potřeby (podle počtu uživatelů).

Platforma byla původně vytvořena jako prostředek k zpracování a ukládání výpovědí přeživších holokaustu. Její využití je ale díky použité architektuře jednoduše přenositelné i na jiná odvětví a domény.

5.1 Architektura platformy SpeechCloud

SpeechCloud integruje řečové technologie neboli hlasové modely (ASR, SLU, DM, NLG, TTS) pomocí předpřipravených modelů a zprostředkovává je uživateli prostřednictvím WebSocket URI, přes kterou se uživatel připojí na dialogový manažer. [18] [19]

5.1.1 Hlasový klient

Hlasový klient je klientská aplikace zprostředkovávající analogově-digitální konverzi hlasu od uživatele a digitálně-analogovou konverzi odpovědi systému. Hlasový klient je propojen s ostatními moduly kombinací protokolů SIP (Session Initiation Protocol) pro přenos signalizací a RTP (Real-time Transport Protocol) pro přenos samotných audio dat.

5.1.2 Vizualní klient

Vizualní klient je klientská aplikace mající k dispozici zpravidla vizuální výstup (display) pro multi-modální interakci s uživatelem. Zatímco hlasový

klient komunikuje pouze prostřednictvím hlasu, vizuální klient může používat např. webový prohlížeč pro bohatou oboustrannou komunikaci - zobrazování dat, dialogových oken, vizuální zpětné vazby, stejně jako vstup z klávesnice, myši nebo dotykového rozhraní.

5.1.3 DM

Dialogový manažer ve SpeechCloudu je obecný systém, do něhož lze implementovat řízení dialogu podle preferencí uživatele. Není přímo součástí platformy, získává pouze události typu "byl získán výsledek rozpoznání" apod., na jejichž základě vyvolává metody typu "syntetizuj text" apod. Komunikace mezi DM a platformou probíhá prostřednictvím již zmíněných WebSockets.

5.1.4 SpeechCloud API Server

API server slouží jako klíčový prvek spravující SpeechCloud aplikace a propojuje jednotlivé komunikační kanály. Při připojení hlasového klienta vytvoří sezení a v rámci sezení propojuje zprávy zasílané DM, vizuálním klientem a řečovými moduly běžící v rámci SpeechCloud Worker procesu.

5.1.5 SIP ústředna

Ústředna předává audio signál v podobě SIP/RTP od hlasového klienta k Worker procesu a zpět.

5.1.6 Worker

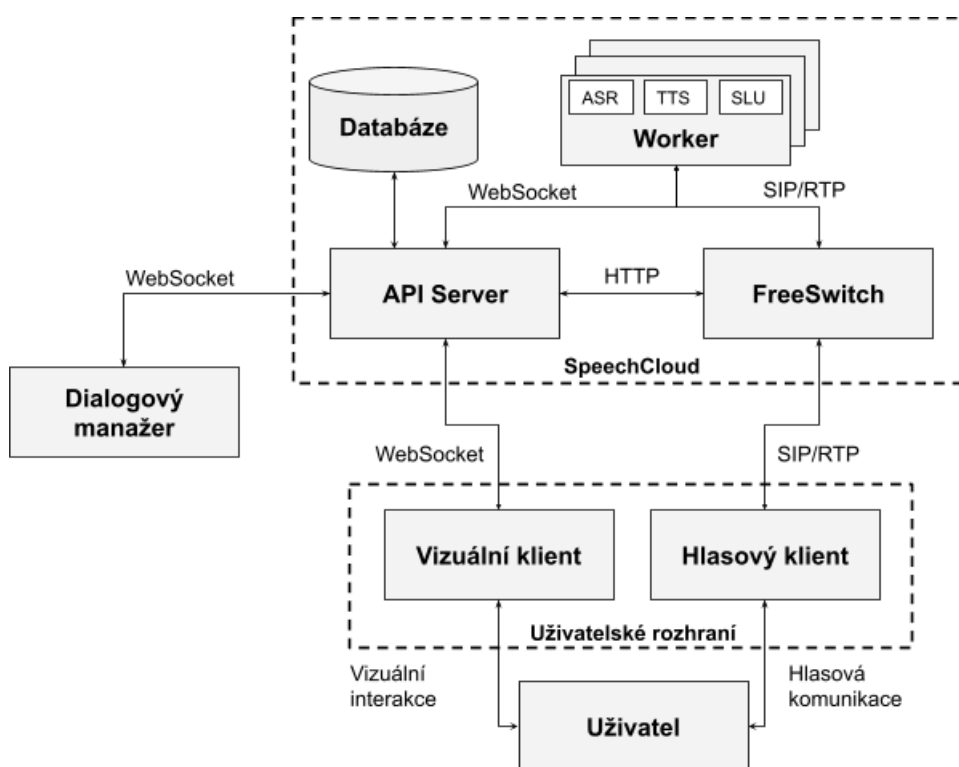
Proces Worker je spuštěn i ukončen podle začátku a konce konkrétního sezení. Na začátku sezení inicializuje hlasové moduly (ASR, SLU, TTS) a zpracovává jejich vzájemná propojení.

Porozumění řeči (*ASR*) je trénováno v *KALDI*, nástroji pro rozpoznávání řeči. *KALDI* provádí trénování pomocí neuronových sítí využívajících GMM

(Gaussian Mixture Models). Předpřipravený jazykový model (LM) a rozpoznávací slovník je možné rozšířit o uživatelem poskytnuté fráze a slova, díky čemuž není užití SpeechCloudu omezeno na jednu nebo malé množství domén a lze jej použít i pro jiné jazyky.

Modul porozumění řeči (*SLU*) funguje ve SpeechCloud podle SRGS gramatik.

Syntéza řeči (*TTS*) je zprostředkována přímo v DM, čímž je odstíněna od uživatele. Je ale možné ji podle představ modifikovat pomocí dostupných modulů.



Obrázek 8: Struktura platformy SpeechCloud[19]

6 Platforma Mycroft

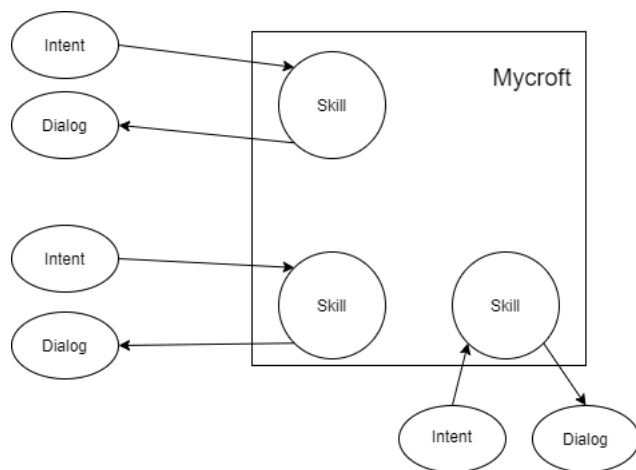
Mycroft, pojmenovaný podle počítače ze sci-fi románu *Měsíc je drsná milenka*, je open source hlasový asistent napsaný v jazyce Python společností Mycroft AI. Je možné jej spustit na počítači či notebooku s operačním systémem Linux, v autě nebo na mobilu s operačním systémem Android. Hlavním tahákem jsou ovšem hardwarové hlasové asistenti Mark I a Mark II, kteří představují alternativu k Google Home či Amazon Echo a slouží k ovládání domácnosti. Ve svých počátcích byl projekt podpořen kampaní na Kickstarтеру, kde na něj uživatelé přispěli přes 120 000\$. Jedním z hlavních důvodů úspěchu této kampaně byly skandály velkých hráčů v oboru hlasových asistentů se sdílením dat uživatelů třetím stranám, které otevřely cestu menším společnostem, jež si zakládají na ochraně uživatelských dat a uživatelích jako takových.



Obrázek 9: Mark II[23]

6.1 Skills

Každý hlasový dialogový systém musí mít definovanou doménu, ve které se bude konverzace provádět. Tato definice je v Mycroft elegantně provedena pomocí tzv. Skills (dovedností). Skills jsou vytvářeny techniky a programátory přímo v Mycroft AI, ale možnost přispět k rozšíření novými či zlepšit již existující Skills mají i samotní uživatelé. Skills mohou zprostředkovávat jednoduché funkce jako hod kostkou, vyprávění vtipů nebo křišťálovou kouli (náhodně řekne ano/ne). Lze pomocí nich i nastavit budík, zjistit situaci na burze nebo pustit hudbu. Skills tedy zajišťují specializaci na různá odvětví, zpracování a generování řeči obstarává Mycroft jako takový.



Obrázek 10: Zakomponování Skills do Mycroftu

6.2 Demo Skill

Pro seznámení s prostředím Mycroft a s jednoduchou reprezentací stavu byl vytvořen Skill **Poker Hands**, který uživateli řekne, jestli jím řečená dvojice karet tvoří pár, či ne. Například eso a eso dává dohromady pár, sedmička a dvojka není pár. Skill v Mycroftu pracuje na turn-based řízení dialogu a Skill

Poker Hands byl vytvořen v souladu s tímto konceptem.

6.2.1 Intent

Každý Skill je rozdělen do dvou částí, kopírující klasickou definici vstupu a výstupu hlasového agenta. Intent (úmysl) představuje, jak název napovídá, vstup uživatele a cíl, se kterým konverzaci s Mycroftem začal.

Ve Skill Poker Hands jsou Intents rozděleny na několik částí podle cíle, který přes ně uživatel sleduje. V prvním a základním se uživatel ptá, jestli jím předložena dvojice karet tvoří pár, nebo ne.

What is {first_card} and {second_card}

Mycroft Intent rozeznává tak, že zpracuje část *What is*, projede všechny nainstalované Skills a snaží se najít ten, který promluvě nejlépe odpovídá, a pokud jich najde více, hledá Skill vyhovující kombinaci *What is* + {first_card} + {second_card}.

Uživatel má také možnost karty předkládat Mycroftu postupně, což zajišťuje Intent skládající se ze dvou menších Intents, kde jeden řeší první kartu a druhý řeší druhou kartu.

1. The first card is {first_card}
2. The first card is {second_card}

Další Intent, který obsahuje čtyři menší Intents, je potvrzování. Jelikož byl Skill Poker Hands vytvořen hlavně za účelem seznámení se s možnostmi reprezentace stavu v prostředí Mycroft, byl též implementován jednoduchý potvrzovací systém, jelikož obě karty jsou nezbytné k vyřešení úlohy, jíž je rozpoznání páru. Uživatel může po vyzvání Mycroftu potvrdit obě karty, první či druhou, nebo obě zamítnout a vybrat si nové karty. Příklady použití v pořadí - 1) potvrdit obě karty, 2) potvrdit první a změnit druhou, 3) potvrdit druhou a změnit první, 4) zamítnout obě karty:

1. I confirm both cards
2. I confirm the first card but second card is {second_card}
3. I confirm the second card but first card is {first_card}
4. I want to change my cards

Poslední Intent je opět spojení menších Intents, kde jeden se využívá na prezentaci reprezentace stavu a druhý slouží k jejímu vymazání.

1. Show me your state representation
2. Delete your state representation

6.2.2 Dialog

Dialog definuje, jakým způsobem bude hlasový asistent odpovídat, představuje tedy výstup dialogového agenta. Hlasoví asistenti by měli mít možnost odpovídat vícero způsoby, proto je dobrou praxí pro Skill předepsat několik možných variant, aby konverzace zněla více "life-like" (živěji, méně roboticky). Toho bylo dosaženo využitím kombinací odpovědí generovaných v samotném kódu, které mají charakter samotné odpovědi pár/nepár a které byly poté poslány do dialogu:

{result}

kde je hlavní odpovědí *text*, a do dialogů, z nichž se každý stará o zajištění odpovědi na jeden z Intents. Pořadí - 1) potvrzení obou karet, 2) neporozumění ani jedné z karet, 3) porozumění první a neporozumění druhé kartě, 4) neporozumění první a porozumění druhé kartě, 5) opětovné neporozumění jedné z karet po opravě uživatele, 6) prezentace stavové reprezentace, 7) vymazání stavové reprezentace:

1. Please, confirm your cards. The first card is {first_card}, the second card is {second_card}

2. I am sorry, I did not understand anything. Please, try again.
3. I am sorry, I did not understand the second card, but I understood the first which was {first_card}. Please, say the second card again.
4. I am sorry, I did not understand the first card, but I understood the second which was {second_card}. Please, say the first card again.
5. I am sorry, I don't understand the {which_card}. Please, try again.
6. I am showing you my state representation. {result}
7. I am deleting my state representation, master. {result}

6.2.3 Testování

V Mycroft AI jsou velmi pečliví ve vybírání Skills, které zveřejní na svém Skills Marketplace (trhu), ze kterého si mohou uživatelé stahovat libovolné Skills podle své potřeby. V samotném Mycroftu je proto zakomponován testovací algoritmus, pomocí kterého může uživatel jednoduše ověřit funkčnost svého vytvořeného Skill. Testy mohou být nadefinované v *.json* formátu, kde návrhář nadefinuje, kterému *Intent* má určitá *utterance* (promluva) odpovídat a jaký *dialog* by na ni měl reagovat. Může se zde také definovat maximální *timeout*, jestli má Mycroft žádat uživatele o *confirmation* (potvrzení) atd., což bylo z důvodu pouhého seznámení s funkčností Mycroftu vynecháno.

Návrhář má také možnost nadefinovat testy pomocí tzv. *test steps* (testovacích kroků). V Mycroftu existují čtyři základní steps, které obsáhnou naprostou většinu možných Intents a Dialogs, i tak má ale technik možnost vytvořit si své vlastní steps, pokud mu pomohou se zachycením kýžené funkčnosti Skill. Prvním ze základních steps je *Given* (předpokládající), který Mycroftu předkládá skutečnosti, které platí pro určitého uživatele. Pokud se předpokládá, že uživatel hovoří anglicky a žije v Londýně, lze tuto podmínku

napsat jako: **Given** an english speaking user **And** user is located in London. Dalšími důležitými steps jsou *When* (pokud, když) a *Then* (pak), jež představují jednu dialogovou obrátku a jsou napsané ve stylu: **When** the user says "how hot will it be today", **Then** "mycroft-weather" should reply with dialog from "current.high.temperature.dialog", kde "mycroft-weather" odpovídá Skill, který má zpracovat "how hot will it be today" pomocí dialogu "current.high.temperature.dialog". Posledním základním step je *And*, *But*, který pouze upravuje čitelnost více po sobě jdoucích *Then* steps.

Pro Skill Poker Hands by měl jeden test vypadat jako:

Feature: poker-hands

Scenario: flush

Given an english speaking user

When the user says "what is a flush"

Then "poker-hands" should reply with dialog from "poker.hands.dialog"

6.3 Architektura platformy Mycroft

6.3.1 Wake Word

Wake Word ("probouzečí" slovo) je fráze používaná u všech hlasových asistentů na světě. Jednoduché vysvětlení je takové, že hlasoví asistenti pořád poslouchají a průběžně zpracovávají zvuky, které zachytili, ale žádné z těchto zvuků si neukládají nebo s nimi dále nepracují. Skutečná konverzace začíná až potom, co uživatel pronese *Wake Word*. "Hey Siri" nebo "Hey Google" jsou těmi nejznámějšími, v Mycroftu si může uživatel vybrat z několika variant, nejpoužívanější ovšem je pro zachování jednotnosti "Hey Mycroft".

6.3.2 ASR

Mycroft se datuje do roku 2016, kdy nebyly ještě hluboké neuronové sítě rozvinuté jako dnes, proto byly pro automatické rozpoznávání řeči použité

skryté Markovovy modely, které i v Mycroftu v následujících letech nahradily neuronové sítě. ASR je v Mycroftu zajištěna **Deep Speech** speech-to-text enginem, který je založen na vědeckém článku **Deep Speech: Scaling up end-to-end speech recognition**[9]. Deep Speech je obrovská rekurentní neuronová síť (RNN), která je natrénovaná na několika grafických kartách a využívá tisíce hodin nahrávek. Trénovací data se skládají jak z promluv pronesených lidmi, tak ze synteticky vytvořených, díky čemuž je Deep Speech schopen replikovat například nekvalitní mikrofon, který přenáší nedokonalou promluvu, a je velmi robustní vůči šumům pozadí a různým nekvalitám v datech.

6.3.3 SLU

Pro udržení konvencí zavedené Mycroftem je SLU nazýváno jako **Intent parsing** a je v Mycroftu obstaráno **Adaptem**, který byl vytvořen samotným Mycroftem AI [23][12]. Adapt je napsaný v jazyce Python a převádí promluvu od uživatele na **Intent** - cíl uživatele, **A match probability** - vyjádření důvěry systému v to, že se jedná o určitý intent, a seznam **Entit**, které pomáhají Mycroftu s dalšími funkcionalitami (například u skladby "Hey Jude" je Adapt schopný zjistit, že se jedná o skladbu skupiny "The Beatles").

6.3.4 NLG, TTS

Pro převod textu do řeči a generování hlasu využívá Mycroft **Mimic**, který je založen na **Festival Lite** systému syntézy řeči.[23][13]

7 Návrh knihovny pro reprezentaci stavu pro Mycroft

Před samotnou konstrukcí knihovny pro reprezentaci stavu bylo nutné provést analýzu, k čemuž vedla implementace dema v prostředí Mycroft. Díky této vizualizaci bylo jednodušší přesně určit, které základní i pokročilé funkce by tato knihovna měla nabízet.

7.1 Mycroft a reprezentace stavu

Jelikož Mycroft sám o sobě neobsahuje reprezentaci stavu hlasového dialogového systému, musela být vytvořena od základu. Reprezentace stavu byla v Skill Poker Hands implementována jako jednoduchý slovník, který obsahuje klíče: "user" k definování jeho cíle (v tomto případě nadbytečná z důvodu pouze jednoho možného cíle), "task", který nabývá hodnot *in_progress*, *pair* a *not_pair* (opět nadbytečné a užitečné spíše jenom k logování), a "agent", který obsahuje vnořený slovník představující první a druhou kartu s jejich hodnotami a stavem (None / confirmed / not_confirmed). Skill pracuje pouze s hodnotami karet a s jejich potvrzením. Nemůže ale říct uživateli, jestli je jeho dvojice karet pár, nebo ne, pokud se explicitně nezeptá na potvrzení.

7.1.1 Inicializace stavové reprezentace

Jak již bylo řečeno výše v kapitole Hlasový dialogový systém (HDS), stav hlasového dialogového systému se skládá z uživatele a jeho stavu, ze stavu řešené úlohy a z dialogového agenta a jeho stavu a jde si ho představit jako rámec se sloty, například jako slovník (= rámec) v jazyku Python s hodnotami (= sloty). Stav uživatele představuje cíl, za kterým uživatel s agentem dialog vede (v demu - zjištění, jestli je uživatelova handa pár, či ne), popřípadě by měl obsahovat také zkušenosti uživatele s hlasovými dialogovými systémy a pro další přizpůsobení může například obsahovat i rozpoložení uživatele.

Stav řešené úlohy vyjadřuje stav, ve kterém se úloha právě nachází (v demu - "zpracovává se", "pár", "nepár"), běžně jsou ale úlohy kvůli své rozsáhlosti pozorovány nepřímo, jako například dvojice databázový dotaz a odpověď z databáze. Stav agenta lze popsat jako obraz toho, co agent uživateli rozuměl (v demu - hodnota karty), a měl by zaznamenávat veškerou komunikaci od začátku hlasového dialogu. Důležitými prvky stavu agenta jsou také příznaky vypovídající o stavu hodnoty slotu. Tyto hodnoty mohou být **nevyplněné, nepotvrzené, potvrzené** a **nekonzistentní**. Způsob postupu při získávání a potvrzování hodnot je uveden v kapitole Řízení dialogu založené na rámcových strukturách.

V teorii programovacího jazyka Python je stavová reprezentace objekt (= třída) zastřešující objekty jednotlivých slotů, kde je slot myšlen jako stav uživatele, úlohy a agenta. Pro každý slot je vytvořena třída s funkcemi, které tento slot spravují. Důležitou poznámkou je, že slot pro uživatele a úlohu by mohla řešit pouze jedna třída, jelikož je třeba pouze přidávat a získávat hodnotu slotu, ale pro přehlednost kódu byla tato třída rozdělena do dvou, z nichž každá představuje jiný slot.

Pro inicializaci stavové reprezentace je důležité znát cíl uživatele a možné stavy úlohy. Sloty agenta mohou být dynamicky přidávány a odebírány.

7.1.2 Přidání nových nepotvrzených hodnot

Základním stavebním kamenem funkčnosti knihovny by měla být funkce umožňující přidání nových nepotvrzených hodnot do slotů. Opět příklad z dema: Pokud uživatel řekne "What is ace and ace?", hodnota slotů představující první a druhou kartu by se měla změnit na "ace", ale ani jedna z těchto karet stále není potvrzená, proto by jim měl být přiřazen příznak "not_confirmed".

Pokud obsahuje slot více než jednu hodnotu, měla by se/musí se zavést *confidence score* (skóre důveryhodnosti) pro všechny hodnoty, aby mohly být

seřazeny od nejpravděpodobnější po nejméně pravděpodobnou. Tyto confidence scores budou vycházet z ASR a SLU částí hlasového dialogového systému.

7.1.3 Získání stavu dialogu

V hlasových dialogových systémech je důležité, aby měl uživatel vždy přístup k jeho stavu a mohl si jej kdykoliv nechat prezentovat. Funkce by měla zajistit tuto prezentaci stavu uživateli. Pro tento účel je vytvořeno několik funkcí, z nichž každá zajišťuje získání určité části reprezentace stavu. Jedna získává stav uživatele, jedna stav úlohy a několik dalších získávají stav agenta. Funkce získávající stav agenta jsou rozděleny podle toho, jakou určitou informaci uživatel nebo hlasový agent požaduje. Existuje proto funkce na získání všech slotů, získání všech hodnot jednoho nekonzistentního slotu, všech potvrzených slotů, všech nepotvrzených slotů, všech nekonzistentních slotů, všech prázdných / nevyplněných slotů.

7.1.4 Potvrzení hodnot slotů

Jelikož hlasový agent může/by měl z definice hlasových dialogových systémů ovlivňovat řešenou úlohu pouze pomocí potvrzených hodnot slotů, musí existovat funkce, která k tomuto potvrzování slouží.

7.1.5 Smazání reprezentace stavu

Smazání reprezentace stavu slouží k tzv. resetování reprezentace stavu neboli vyprázdnění reprezentace. Tato situace je zcela v rukou uživatele a je pak na něm, kdy bude chtít reprezentaci smazat. Tato funkce také funguje dobře jako debugovací a testovací nástroj.

7.1.6 Změna stavu uživatele

V knihovně pro reprezentaci stavu je k dispozici funkce, která mění stav uživatele neboli jeho cíl. Na tento stav se může uživatel kdykoliv zeptat hlasového agenta.

7.1.7 Změna stavu úlohy

Stejně jako pro změnu stavu uživatele je také vytvořena funkce měnící stav úlohy. I tento stav si může uživatel podle libosti zobrazit.

7.1.8 Získání (a řešení nekonzistentních) slotů

Jedním z hlavních problémů reprezentace stavu jsou nekonzistentní sloty a řešení jejich nekonzistence. Je proto vytvořena funkce, která nalezne všechny nekonzistentní sloty a vrátí je hlasovému agentu jako list listů. Cílem je pak dostat z těchto nekonzistentních slotů jednu hodnotu, se kterou bude posléze vyřešena úloha.

Na následujícím řádku je vidět příklad nekonzistentního slotu se třemi hodnotami (ace, king, queen). Nekonzistence byla vyřešena vybráním jedné hodnoty (ace) a příznak se změnil na "nepotvrzený":

```
slot_1: [[ace, king, queen], "inconsistent"] -> slot_1: [ace, "not_confirmed"]
```

Jedním z možných vylepšení této metody získávání a řešení konfliktů by byla možnost ponechat více než jednu hodnotu z nekonzistentních slotů a úlohu pak provést nad více hodnotami. Například hlasový dialogový systém sčítající čísla, který by měl jeden slot s hodnotou 4 a jeden nekonzistentní slot s hodnotami 1, 3 a 5, z nichž by si uživatel vybral 1 a 5, by uživateli vrátil dva výsledky, 5 a 9.

7.2 Výsledný kód a jeho zhodnocení

Knihovna vzešlá z požadavků v kapitole Návrh knihovny pro reprezentaci stavu pro Mycroft fungovala uspokojivě, ale docházelo k problémům s jejím dalším rozšířením v kódu dialogu. Tyto problémy pramenily především z ne-správného počátečního rozhodnutí udržovat sloty v seznamu, kde byl jeden slot reprezentován jako slovník slovníků z důvodu udržení všech hodnot a jejich confidence scores.

Například vkládání dat do určitého slotu vyžadovalo, aby si uživatel (v tomto případě dialogový agent) pamatoval pozici slotu v seznamu slotů, do kterého chce tuto hodnotu vkládat. Stejný princip platil i pro získávání dat, kde se odehrávalo složité "prohlédávání" seznamu obsahujícího slovník slovníků apod.

Další chybou knihovny bylo opomenutí udržování historie, díky níž by bylo možné vrátit se k předešlému kroku dialogu, pokud by se dialog v určité části zasekl.

V knihovně taky neexistovalo rozdělení různých tříd pro sloty, hodnoty apod., což vycházelo z prvotního špatného rozhodnutí udržovat všechny hodnoty a sloty v seznamu či slovníku.

Na následujícím obrázku 11 je vidět podoba knihovny. Na obrázku 12 je pro srovnání vidět kód nové knihovny.

```

def get_all_empty_slots(self):
    """
    For getting all empty slots = to be used if there are no inconsistencies, thus no
    confidence score
    Returns: array of number of empty slots [1, 4, 7,...]
    """
    empty_slots = list()
    for i in range(len(self.STATE_REPRESENTATION["agent"])):
        if self.STATE_REPRESENTATION["agent"][f"slot_{i+1}"][0] is None:
            empty_slots.append(i+1)
    return empty_slots

def set_slot_value_not_cofirmed(self, slot_value, which_slot):
    """
    For setting an unconfirmed value to a slot = to be used if there are no inconsistencies,
    thus no confidence score
    Args: value to set, number of slot
    """
    self.STATE_REPRESENTATION["agent"][f"slot_{which_slot}"] = [slot_value, "not_confirmed"]

def set_slot_cofirmed(self, which_slot):
    """
    For "switching"/setting a slot to a confirmed state = to be used if there are no
    inconsistencies, thus no confidence score
    Args: which slot to confirm
    """
    self.STATE_REPRESENTATION["agent"][f"slot_{which_slot}"][1] = "confirmed"

```

Obrázek 11: Ukázka zdrojového kódu knihovny

8 Analýza a návrh obecné knihovny pro reprezentaci stavu

Ačkoliv knihovna sestavená pro reprezentaci stavu založená na demu pro Mycroft fungovala po několika iteracích a testech správně, bylo usouzeno, že tímto směrem cesta nepovede. Kód byl zbytečně rozsáhlý, rigidní, místy až nepřehledný, a hlavně vyžadoval velké úpravy a přizpůsobení kódu samotného dialogu, a jelikož hlavní výhodou softwarové knihovny by měla být její vysoká modularita, začalo sestavování nové knihovny od začátku.

8.1 Základní myšlenka knihovny

Jak již bylo zmíněno výše, jedním z hlavních aspektů softwarové knihovny je její modularita neboli přenositelnost. Uživatel, v tomto smyslu slova je uživatel knihovny hlasový agent, popřípadě řízení dialogu, by měl minimálně, v nejlepším případě vůbec, měnit část svého kódu, aby do něj mohla být knihovna zakomponovatelná. Dalším důležitým bodem této knihovny by měla být dynamičnost a univerzálnost, aby uživatel nemusel být omezen staráním se například o inicializaci správného počtu slotů apod. Těchto požadavků bylo dosaženo pomocí následující strukturizace knihovny.

8.2 Strukturizace knihovny

Knihovna obsahuje čtyři entity, z nichž má každá svůj účel.

8.2.1 Slot - třída Slot

Tato třída představuje jednotlivé sloty stavové reprezentace (uživatele, agenta, řešenou úlohu). Nejdříve byly implementovány tři typy tříd (UserSlot, AgetSlot, TaskSlot), ale během iterování kódu bylo toto řešení zefektivněno

a byla vytvořena jedna třída pro všechny typy slotů. Na třídu **Slot** byly kladeny požadavky:

- **Slot** může být buď prázdný, nekonzistentní, nepotvrzený nebo potvrzený
- **Slot** může mít žádnou, jednu nebo více hodnot
- potvrzený **Slot** má právě jednu hodnotu
- nekonzistentní **Slot** má více než jednu hodnotu
- hodnoty **Slotu** jsou reprezentovány třídou **Value**
- hodnota s nejvyšší confidence je první
- hodnota s nejnižší confidence je poslední
- přechod mezi stavy řešen podle diagramu 6
- použití `getattr`, `setattr`, `delattr` pro jednoduchý přístup k hodnotám a stavům **Slotu**[8]
- přístup k libovolné hodnotě
- kontrola přijatelnosti hodnot
- při přiřazení nové hodnoty **Slotu** se **Slot** se starý hodnotami odsune do **History** a v rámci **State** jej nahradí **Slot** s novými hodnotami

8.2.2 Hodnota slotu - třída **Value**

Třída **Value** představuje hodnotu určitého slotu a byly na ni kladeny nároky:

- hodnota slotu má svoji confidence

- výchozí hodnota confidence je 1.0
- hodnota slotu je immutable s využitím Pyrsistent

8.2.3 Kompozitní stav - třída State

Třída **State** představuje samotnou stavovou reprezentaci a obaluje sloty uživatele, agenta a úlohy. Byly na ni kladeny nároky:

- ke **Slotu** se přistupuje pomocí atributu instance **State**
- immutable object s využitím Pyrsistent
- umožňuje vrátit globální stav stavové reprezentace
- množinu definovaných **Slotů** lze dynamicky měnit
- při odstranění **Slotu** se **Slot** odsune do **Historie**, v nové instanci **State** již nebude jako atribut
- instance třídy **State** odkazují na svoji **Historii**

8.2.4 Historie - třída History

Hlavní změnou oproti knihovně vytvořené pro Mycroft demo je implementace historie dialogu, reprezentována třídou **History**. Stavová reprezentace totiž sestává nejen z aktuálních stavů a hodnot jednotlivých slotů, ale také z minulých. Díky historii se může dialog kdykoliv beztréstně vrátit k jakémukoliv stavu, což je velmi výhodné například při tom, když se dialog dostane do neřešitelné situace. Z této filozofie vycházely pak požadavky:

- immutable object s využitím Pyrsistent
- prvky Historie jsou instance třídy State

- historie musí být součástí stavové reprezentace z důvodu možnosti vrátit se do konzistentního stavu, pokud se dialog “ztratí”
- historie se bude aktualizovat při jakékoliv změně kompozitního slotu
- musí obsahovat celou historii dialogu

8.3 Úpravy knihovny během vytváření

Po několika iteracích této podoby knihovny se dospělo k tomu, že bude jednodušší obejít se bez třídy **History**, která pouze obaluje immutable object sloužící k ukládání průběhu dialogu, a proto byla historie přímo zavedena do třídy **State** jako samostatný immutable object typu *pvector* (seznam) knihovny ??.

Jelikož jsou hlasové dialogové systémy, jmenovitě části ASR a SLU - ”pair” (pár) je porozuměno jako ”bear” (medvěd), ”z Varů” je porozuměno jako ”svárů”, často zatíženy chybami a může se i stát, že uživatel řekne hodnotu, která není pro určitý slot podporována - ”Najdi mi vlak z Plzně do Prahy ve **25:00**”, je důležité tyto hodnoty kontrolovat a případně uživatele upozornit, že mu nebylo porozuměno nebo že jím řečená hodnota není v dialogu podporována. To bylo v knihovně dosaženo vytvořením tříd obstarávajících kontrolu hodnot pro specifické sloty. Tyto třídy jsou odděleny od hlavní třídy **Value**. Oddělené třídy jsou rozšířeny o příznak *valid*, který nabývá hodnot *True* pro použitelné hodnoty a *False* pro nesprávné hodnoty.

Další úpravou byla implementace funkce pro přidávání slotů, která slouží primárně k specifikování oboru hodnot, které do slotu mohou přicházet. Pokud se jedná například o hlasový dialogový systém poskytující informace o vlakových spojeních, který potřebuje pro splnění úkolu stanici odkud, stanici kam, čas odjezdu a typ vlaku, pak se na začátku dialogu inicializuje instance třídy **State**, do které se pomocí funkce *new_slots* vloží tyto potřebné sloty, u nichž bude specifikováno, jakou oddělenou třídu od třídy **Value** budou používat.

Pro tento příklad tedy **Station**, **TimeValue** a **TraintypeEnum**. Tyto třídy musí být samozřejmě připraveny před spuštěním samotného dialogu.

8.3.1 Využití callbacků a ukládání historie

Jednou z hlavních překážek při vytváření knihovny byl požadavek na ukládání průběhu dialogu. Jakým způsobem, v jakých metodách a kdy během dialogu by se měly jednotlivé informace ukládat. Řešení bylo dosaženo využitím **callbacků**, které byly vytvořeny nad rámec původní myšlenky knihovny a které jsou při svém zavolání ukládány do historie. Callbacky jsou kontextově závislý kód pro obsluhu budoucích vstupů od uživatele. Běžně se využívají v asynchronním kódu, kde odpověď není obecně instantní a čeká se na ni. Dialogy jsou právě tímto asynchronním kódem a proto byly do knihovny callbacky implemetovány. S jejich využitím se jednoduše kontroluje průběh dialogového systému, jelikož každý callback je jednou částí průběhu existence slotu jako v kapitole Řízení dialogu založené na rámcových strukturách. Idea je taková, že ve funkci *new_slots* se do historie vloží vytvoření nových slotů a poté se budou do historie ukládat callbacky, díky čemuž bude jednoduché sledovat průběh stavu jednotlivých slotů. V historii je pak vidět, co se s určitými sloty dělo a jaké hodnoty se do nich případně vkládaly.

Callbacků bylo vytvořeno šest:

- *complete_empty* - vyplnění **nevyplněných slotů**
- *add_new_value* - přidání nové hodnoty do **neprázdného slotu**
- *confirm_unconfirmed* - potvrzení **nepotvrzených slotů**
- *disambig* - nabídka vyřešení **nekonzistentních slotů**
- *present* - vyřešení úlohy z **potvrzených slotů** a prezentace výsledku

- *emptying_slots* - vyprázdnění slotů pro další kolo dialogu

Callbacks v knihovně řeší funkce *expect*, která je ukládá do immutable objektu *expectations*, z něhož jsou v následující obrátce dialogu vybrány a řešeny funkcí *push*. Funkce *push* provádí všechny hlavní úkoly v knihovně, jež jsou přidání nové hodnoty do slotu, vyřešení inkonzistence slotu, potvrzení slotu a vyprázdnění slotu.

8.4 Výsledný kód

Oproti původní knihovně, která vzešla z demo dialogu v Mycroftu, došlo k celkovému zjednodušení kódu, ačkoliv je kód delší.

Jak bylo v analýze požadováno, knihovna obsahuje pouze tři hlavní třídy (**Value**, **Slot** a **State**) a třídy pro reprezentaci speciálních hodnot a jejich kontrolu, jako například vlakové stanice, čas atd., jsou pouze oddělené od hlavní třídy **Value**.

Do třídy **Value** jsou vkládány hodnoty a jejich confidence score a u specifických hodnot se také provádí jejich kontrola, reprezentována příznakem *valid*, který nabývá hodnot True, nebo False.

Třída **Slot** reprezentuje sloty stavové reprezentace a obsahují instance třídy **Value**. Při vložení nové hodnoty do slotu se provede sestupné seřazení podle confidence score. Díky tomu je možné ze slotu získat *first_value* (nejpravděpodobnější hodnota), *last_value* (nejméně pravděpodobná hodnota) a *all_values* (všechny hodnoty). Třída **Slot** má také příznak *state* (stav), který nabývá hodnot podle části analýzy Slot - třída Slot.

Třída **State** obsahuje funkci *new_slots*, která pro sloty "připraví místo" a uloží jejich jména do seznamu a zároveň uloží, jakou oddělenou třídu hodnot bude slot používat. Dále obsahuje funkci *push*, která zařizuje uspokojení callbacků, a pokud jsou dostupné nové hodnoty, vloží nové hodnoty do slotů. V této funkci také dochází k naplňování historie dialogu callbacky, což umožňuje jednoduché trasování průběhu dialogu. Funkce call-

backů je popsána a vysvětlena v kapitole Využití callbacků a ukládání historie. Dále třída **State** obsahuje funkce na získávání všech prázdných, potvrzených, nepotvrzených a inkonzistentních slotů a funkce na smazání, respektive vyprázdnění, buď jednoho, či všech slotů.

Knihovna po těchto úpravách splňuje všechny body analýzy a k jejímu použití bylo potřeba minimální zásahu do samotného kódu dialogu.

Na obrázku 12 je vidět finální vzhled knihovny, jmenovitě funkce *push*.



```
def push(self, *args, **kwargs):
    expectations, self.expectations = self.expectations, v()
    if len(expectations) != 0: # condition so as not to have an empty spaces in
    history
        self.History = self.History.append(expectations)
        for cb, slot_names in expectations:

            cbkwargs = {key: kwargs.get(key) for key in slot_names}

            # Call the expectation callback and pass myself as first argument
            handled = cb(self, **cbkwargs)
            if handled:
                # The arguments were handled by the callback
                for key in slot_names:
                    del kwargs[key]
            if kwargs:
                # Only change state if there is something to push
                self.extend(**kwargs)

            if args and cb.__name__ == "confirm_unconfirmed":
                self.confirm_slots(*args)
            if args and cb.__name__ == "emptying_slots":
                self.empty_slots_fcn(*args)
```

Obrázek 12: Ukázka zdrojového kódu knihovny

9 Demontrace užití knihovny na různých dialogových systémech

Pro demonstraci funkčnosti knihovny na reprezentaci stavu byla zakomponována jak v dialogích ve spojení s platformou SpeechCloud, tak v rámci Mycroft agenta.

Všechny dialogy probíhaly podle tohoto schématu:

1. vytvoření nových slotů (*new_slots*)
2. očekávání vyplnění slotů (*complete_empty*)
3. vložení nové hodnoty do neprázdného slotu (*add_new_value*)
4. vyřešení případných inkonzistencí (*disambig*)
5. potvrzení nepotvrzených slotů (*confirm_unconfirmed*)
6. prezentace slotů (*present*)
7. vyprázdnění slotů pro další dialog (*emptying_slots*)

Příklady a jejich výstupy jsou vidět na následujících stránkách. U výstupů číslování představuje dialogovou obrátku, část **Promluva** představuje konverzaci uživatele a agenta, kde **U** je promluva uživatele a **A** promluva agenta, a část **Aktualizace stavu dialogu** představuje vývoj dialogu ¹.

Poznámka k historii - jelikož historie dialogu je pro člověka nečitelná, je v příkladech uveden její zjednodušený a zkrácený tvar. Použité zkratky jsou:

- *Create* pro vytvoření slotů
- *CompEmpty* pro *complete_empty*

¹pro konverzaci příkladu v Mycroftu chybí vždy 1. a poslední obrátka, jelikož Mycroft nijak nezahajuje a neukončuje dialog

- *AddNew* pro *add_new_value*
- *ConfUnconf* pro *confirm_unconfirmed*
- *Disam* pro *disambig*
- *Pres* pro *present*
- *Empty* pro *emptying_slots*

Po zkratkách vždy následuje, jakých slotů a případně jakých hodnot se tato operace týká.

Jak bylo zmíněno v kapitole Porozumění řeči - Spoken Language Understanding (SLU), důležitým krokem při navrhování hlasových dialogových systémů je tzv. "model uživatele". Pro všechny následující příklady byl uživatel modelován jako člověk s minimálními zkušenostmi s hlasovými dialogovými systémy. Proto se agent opakovaně ptá, jestli uživatel souhlasí se zadanými hodnotami, a stále mi připomíná, které hodnoty již řekl. Uživatelské rozpoložení bylo vybráno jako neutrální, agent proto mluví celými větami a spisovnou češtinou či angličtinou.

9.1 Karty - Mycroft

Struktura a funkčnost tohoto dialogu byla oproti původnímu dialogu v kapitole Výsledný kód a jeho zhodnocení změněna, aby byla lépe demonstrována funkčnost vytvořené knihovny.

Uživatel se stále ptá, jestli spolu dvě karty tvoří pár. Může říct obě karty najednou, může říct nejdříve první a pak až druhou, nebo nejdříve druhou a pak až první. Pokud uživatel řekne pro jednu kartu více hodnot, je agentem vyzván, aby si vybral a vyřešil tím inkonzistenci. Uživatel je také vyzván k případnému doplnění chybějící karty a potvrzení hodnot karet, jež může uživatel také zamítnout, čímž se stavová reprezentace vyprázdní a uživatel

se může zeptat znovu. Po splnění úlohy je uživateli prezentován výsledek a může se ptát dále.

```
@intent_file_handler('first.card.hands.poker.intent')
# handle "The first card is {first_card}"
def handle_first_card_hands_poker(self, message):
    first_card = message.data.get('first_card')
    self.state.push(first_card=first_card)
    if len(self.state.first_card.all_values) > 1:
        result = f"You said {first_card} but you already said the first card. It
        was {self.state.first_card.first_value}. From the cards
        {self.state.first_card.all_values} which do you want to choose?"
        self.state.expect(self.state.disambig, self.state.first_card)
        self.speak_dialog('hands.poker', data={"result": result})
    elif (not self.state.second_card.first_value) and
        (self.state.first_card.first_value):
        result = f"The first card is {first_card}. What is the second card?"
        self.state.expect(self.state.complete_empty, self.state.second_card)
        self.speak_dialog('hands.poker', data={"result": result})
    else:
        if not self.state.first_card.first_value:
            self.state.expect(self.state.complete_empty, self.state.first_card,
            self.state.second_card)
            self.speak_dialog('notunderstood.hands.poker', data={'which_card':
            "first card"})
        else:
            self.state.expect(self.state.confirm_unconfirmed,
            self.state.first_card, self.state.second_card)
            self.speak_dialog('confirm.cards', data={'first_card': first_card,
            'second_card': self.state.second_card.first_value})
```

Obrázek 13: Ukázka zdrojového kódu dialogu

9.1.1 Příklad 1

V tomto příkladu uživatel pouze předloží dvě karty, potvrdí je a je mu prezentován výsledek.

Turn 1:

Aktualizace stavu dialogu:

New slot added: first_card

New slot added: second_card

Slot first_card must be filled

Slot second_card must be filled

Turn 2:

A: *začátek*

U: Hey Mycroft. What is ace and ace?

Aktualizace stavu dialogu:

Extended values: first_card = ace

Extended values: second_card = ace

Turn 3:

A: Please, confirm your cards. The first card is ace, the second card is ace.

U: Hey Mycroft. I confirm both cards.

Aktualizace stavu dialogu:

Slot first_card must be confirmed

Slot second_card must be confirmed

Slot first_card is confirmed

Slot second_card is confirmed

Turn 4:

A: It is a pair of aces.

U: *konec*

Aktualizace stavu dialogu:

Value was presented ace for slot first_card

Value was presented ace for slot second_card

Turn 5:

Aktualizace stavu dialogu:

Slot first_card will be emptied

Slot second_card will be emptied

Slot first_card is emptied

Slot second_card is emptied

Historie dialogu:

Turn 1: Create['first_card', 'second_card']

Turn 2: CompEmpty['first_card'='ace', 'second_card'='ace']

Turn 3: ConfUnconf['first_card', 'second_card']

Turn 4: Present['first_card', 'second_card']

Turn 5: Empty['first_card', 'second_card']

9.1.2 Příklad 2

V tomto příkladu uživatel nejdříve předloží druhou kartu, po vyzvání doplní i první kartu, ale pak si druhou kartu rozmyslí a změní ji, načež je vyzván agentem, aby vyřešil tuto inkonzistenci. Po vyřešení inkonzistence potvrdí uživatel karty a je mu prezentován výsledek.

Turn 1:

Aktualizace stavu dialogu:

New slot added: first_card

New slot added: second_card

Slot first_card must be filled

Slot second_card must be filled

Turn 2:

A: *začátek*

U: Hey Mycroft. The second card is king.

Aktualizace stavu dialogu:

Extended values: second_card = king

Slot first_card must be filled

Turn 3:

A: The second card is king. What is the first card?

U: Hey Mycroft. The first card is ace.

Aktualizace stavu dialogu:

Extended values: first_card = ace

Turn 4:

A: Please confirm your cards. The first card is ace, the second card is king.

U: Hey Mycroft. The second card is two.

Aktualizace stavu dialogu:

A new value will be added to slot second_card

Extended values: second_card = two

Turn 5:

A: You said two but you already said the second card. It was king. From the cards king, two which do you want to choose?

U: Hey Mycroft. For the second card I choose two.

Aktualizace stavu dialogu:

Value two was accepted in slot second_card

Assigned values: second_card = two

Turn 6:

A: Please confirm your cards. The first card is ace, the second card is two.

U: Hey Mycroft. I confirm both cards.

Aktualizace stavu dialogu:

Slot first_card must be confirmed

Slot second_card must be confirmed

Slot first_card is confirmed

Slot second_card is confirmed

Turn 7:

A: Ace and two is not a pair.

U: *konec*

Aktualizace stavu dialogu:

Value was presented ace for slot first_card

Value was presented two for slot second_card

Turn 8:**Aktualizace stavu dialogu:**

Slot first_card will be emptied

Slot second_card will be emptied

Slot first_card is emptied

Slot second_card is emptied

Historie dialogu:

Turn 1: Create['first_card', 'second_card']

Turn 2: CompEmpty['first_card'=None, 'second_card'='king']

Turn 3: CompEmpty['first_card'='ace']

Turn 4: AddNew['second_card'='two']

Turn 5: Disam['second_card'='two']

Turn 6: ConfUnconf['first_card', 'second_card']

Turn 7: Present['first_card', 'second_card']

Turn 8: Empty['first_card', 'second_card']

9.1.3 Příklad 3

V příkladu č. 3 uživatel nejdříve řekne první kartu, uživatel poté řekne nesprávnou hodnotu a je opět vyzván k doplnění druhé karty. Po doplnění druhé karty jsou obě karty potvrzeny a je prezentován výsledek.

Turn 1:

Aktualizace stavu dialogu:

New slot added: first_card

New slot added: second_card

Slot first_card must be filled

Slot second_card must be filled

Turn 2:

A: *začátek*

U: Hey Mycroft. The first card is ace.

Aktualizace stavu dialogu:

Extended values: first_card = ace

Slot second_card must be filled

Turn 3:

A: The first card is ace. What is the second card?

U: Hey Mycroft. The second card is eleven.

A: I am sorry, I don't understand the eleven. Please, try again.

U: Hey Mycroft. The second card is jack.

Aktualizace stavu dialogu:

Extended values: second_card = jack

Turn 4:

A: Please, confirm your cards. The first card is ace, the second card is jack.

U: Hey Mycroft. I confirm both cards.

Aktualizace stavu dialogu:

Slot first_card must be confirmed

Slot second_card must be confirmed

Slot first_card is confirmed

Slot second_card is confirmed

Turn 5:

A: Ace and jack is not a pair.

U: *konec*

Aktualizace stavu dialogu:

Value was presented ace for slot first_card

Value was presented jack for slot second_card

Turn 6:

Aktualizace stavu dialogu:

Slot first_card will be emptied

Slot second_card will be emptied

Slot first_card is emptied

Slot second_card is emptied

Historie dialogu:

Turn 1: Create['first_card', 'second_card']

Turn 2: CompEmpty['first_card'='ace', 'second_card'=None]

Turn 3: CompEmpty['second_card'='jack']

Turn 4: ConfUnconf['first_card', 'second_card']

Turn 5: Present['first_card', 'second_card']

Turn 6: Empty['first_card', 'second_card']

9.1.4 Příklad 4

V posledním příkladu uživatel doplní obě karty, ale nakonec je nepotvrdí a údaje jsou vymazány.

Turn 1:

Aktualizace stavu dialogu:

New slot added: first_card

New slot added: second_card

Slot first_card must be filled

Slot second_card must be filled

Turn 2:

A: *začátek*

U: Hey Mycroft. What is ace and king?

Aktualizace stavu dialogu:

Extended values: first_card = ace

Extended values: second_card = king

Turn 3:

A: Please, confirm your cards. The first card is ace, the second card is king.

U: Hey Mycroft. I confirm no cards.

Aktualizace stavu dialogu:

Slot first_card must be confirmed

Slot second_card must be confirmed

Turn 4:

A: I am deleting my state representation.

U: *konec*

Aktualizace stavu dialogu:

Slot first_card will be emptied

Slot second_card will be emptied

Slot first_card is emptied

Slot second_card is emptied

Historie dialogu:

Turn 1: Create['first_card', 'second_card']

Turn 2: CompEmpty['first_card'='ace', 'second_card'='king']

Turn 3: ConfUnconf['first_card', 'second_card']

Turn 4: Empty['first_card', 'second_card']

9.2 Karty - SpeechCloud

Dialog pro zjištění páru/nepáru byl také převeden do platformy SpeechCloud. Strukturou je téměř identický s dialogem vytvořeným pro platformu Mycroft, akorát je dialog veden v češtině a používá rozdílné aktivační promluvy.

```
if result_1.first_card.first and not result_1.second_card.first:
    # recena pouze prvni karta
    first = result_1.karta.first
    self.state.push(first_card=first)
    while True:
        result = await self.synthesize_and_wait_for_slu_result(text=f"První karta je
            {self.state.first_card.first_value}. Jaká je druhá karta?", timeout=4.)
        if result.first_card.first and result.karta.first:
            # znovu recena prvni karta
            first = result.karta.first
            self.state.push(first_card=first)
            self.state.expect(self.state.disambig, self.state.first_card)
            while True:
                result = await self.synthesize_and_wait_for_slu_result(text=f"Pro první
                    kartu jste již řekl {self.state.first_card.all_values}. Musíte z
                    nich vybrat jednu. Jaká je vaše odpověď?", timeout=4.)
                if result.karta.first:
                    # vyresena inkonzistence
                    first = result.karta.first
                    self.state.push(first_card=first)
                    break
                else:
                    # nebylo nic rozpoznano
                    await self.synthesize_and_wait(text=f"Neřekl jste, jakou hodnotu má
                        mít první karta.")
                    continue
```

Obrázek 14: Ukázka zdrojového kódu dialogu

9.2.1 Příklad 1

V tomto příkladu uživatel pouze předloží dvě karty, potvrdí je a je mu prezentován výsledek.

Turn 1:

A: Dobrý den. Můžete mi předložit dvě karty a já vám řeknu, jestli dohromady tvoří pár, nebo ne.

U: *začátek*

Aktualizace stavu dialogu:

New slot added: first_card

New slot added: second_card

Slot first_card must be filled

Slot second_card must be filled

Turn 2:

A: Jaké karty vás zajímají?

U: První karta je eso, druhá karta je eso.

Aktualizace stavu dialogu:

Extended values: first_card = eso

Extended values: second_card = eso

Turn 3:

A: První karta je eso a druhá karta je eso. Chcete potvrdit tyto karty? Ano, nebo ne.

U: Ano.

Aktualizace stavu dialogu:

Slot first_card must be confirmed

Slot second_card must be confirmed

Slot first_card is confirmed

Slot second_card is confirmed

Turn 4:

A: Eso a eso je pár.

U: Konec.

Aktualizace stavu dialogu:

Value was presented eso for slot first_card

Value was presented eso for slot second_card

Turn 5:

A: Konec dialogu, nashledanou.

U: *konec*

Aktualizace stavu dialogu:

Slot first_card will be emptied

Slot second_card will be emptied

Slot first_card is emptied

Slot second_card is emptied

Historie dialogu:

Turn 1: Create['first_card', 'second_card']

Turn 2: CompEmpty['first_card'='eso', 'second_card'='eso']

Turn 3: ConfUnconf['first_card', 'second_card']

Turn 4: Present['first_card', 'second_card']

Turn 5: Empty['first_card', 'second_card']

9.2.2 Příklad 2

V tomto příkladu uživatel nejdříve předloží druhou kartu, po vyzvání doplní i první kartu, ale pak si druhou kartu rozmyslí a změní ji, načež je vyzván agentem, aby vyřešil tuto inkonzistenci. Po vyřešení inkonzistence potvrdí uživatel karty a je mu prezentován výsledek.

Turn 1:

A: Dobrý den. Můžete mi předložit dvě karty a já vám řeknu, jestli dohromady tvoří pár, nebo ne.

U: *začátek*

Aktualizace stavu dialogu:

New slot added: first_card

New slot added: second_card

Slot first_card must be filled

Slot second_card must be filled

Turn 2:

A: Jaké karty vás zajímají?

U: Druhá karta je král.

Aktualizace stavu dialogu:

Extended values: second_card = král

Slot first_card must be filled

Turn 3:

A: Druhá karta je král. Jaká je první karta?

U: První karta je eso.

Aktualizace stavu dialogu:

Extended values: first_card = král

Turn 4:

A: První karta je eso a druhá karta je král. Chcete potvrdit tyto karty? Ano, nebo ne.

U: Druhá karta je vlastně dvojka.

Aktualizace stavu dialogu:

A new value will be added to slot second_card

Extended values: second_card = dvojka

Turn 5:

A: Pro druhou kartu jste již řekl král, dvojka. Musíte z nich vybrat jednu. Jaká je vaše odpověď?

U: Dvojka.

Aktualizace stavu dialogu:

Value dvojka was accepted in slot second_card

Assigned values: second_card = dvojka

Turn 6:

A: První karta je eso a druhá karta je dvojka. Chcete potvrdit tyto karty? Ano, nebo ne.

U: Ano.

Aktualizace stavu dialogu:

Slot first_card must be confirmed

Slot second_card must be confirmed

Slot first_card is confirmed

Slot second_card is confirmed

Turn 7:

A: Eso a dvojka není pár.

U: Konec.

Aktualizace stavu dialogu:

Value was presented eso for slot first_card

Value was presented dvojka for slot second_card

Turn 8:

A: Konec dialogu, nashledanou.

U: *konec*

Aktualizace stavu dialogu:

Slot first_card will be emptied

Slot second_card will be emptied

Slot first_card is emptied

Slot second_card is emptied

Historie dialogu:

Turn 1: Create['first_card', 'second_card']

Turn 2: CompEmpty['first_card'=None, 'second_card'='král']

Turn 3: CompEmpty['first_card'='eso']

Turn 4: AddNew['second_card'='dvojka']

Turn 5: Disam['second_card'='dvojka']

Turn 6: ConfUnconf['first_card', 'second_card']

Turn 7: Present['first_card', 'second_card']

Turn 8: Empty['first_card', 'second_card']

9.2.3 Příklad 3

V příkladu č. 3 uživatel nejdříve řekne první kartu, poté ovšem na výzvu na doplnění druhé karty nereaguje, a je proto znovu vyzván. Uživatel poté "omylem" řekne nesprávnou hodnotu a je opět vyzván k doplnění druhé karty. Po doplnění druhé karty jsou obě karty potvrzeny a je prezentován výsledek.

Turn 1:

A: Dobrý den. Můžete mi předložit dvě karty a já vám řeknu, jestli dohromady tvoří pár, nebo ne.

U: *začátek*

Aktualizace stavu dialogu:

New slot added: first_card

New slot added: second_card

Slot first_card must be filled

Slot second_card must be filled

Turn 2:

A: Jaké karty vás zajímají?

U: První karta je eso.

Aktualizace stavu dialogu:

Extended values: first_card = eso

Slot second_card must be filled

Turn 3:

A: První karta je eso. Jaká je druhá karta?

U: ...

A: Neřekl jste, jakou hodnotu má mít druhá karta.

První karta je eso. Jaká je druhá karta.

U: Druhá karta je jedenáctka.

A: Neřekl jste, jakou hodnotu má mít druhá karta. První karta je eso. Jaká je druhá karta?

U: Druhá karta je kluk.

Aktualizace stavu dialogu:

Extended values: second_card = kluk

Turn 4:

A: První karta je eso a druhá karta je kluk. Chcete potvrdit tyto karty? Ano, nebo ne.

U: Ano.

Aktualizace stavu dialogu:

Slot first_card must be confirmed

Slot second_card must be confirmed

Slot first_card is confirmed

Slot second_card is confirmed

Turn 5:

A: Eso a kluk není pár.

U: Konec.

Aktualizace stavu dialogu:

Value was presented eso for slot first_card

Value was presented kluk for slot second_card

Turn 6:

A: Konec dialogu, nashledanou.

U: *konec*

Aktualizace stavu dialogu:

Slot first_card will be emptied

Slot second_card will be emptied

Slot first_card is emptied
Slot second_card is emptied

Historie dialogu:

Turn 1: Create['first_card', 'second_card']
Turn 2: CompEmpty['first_card'='eso', 'second_card'=None]
Turn 3: CompEmpty['second_card'='kluk']
Turn 4: ConfUnconf['first_card', 'second_card']
Turn 5: Present['first_card', 'second_card']
Turn 6: Empty['first_card', 'second_card']

9.2.4 Příklad 4

V posledním příkladu uživatel řekne obě karty, ale nakonec je nepotvrdí, načež jsou údaje vymazány.

Turn 1:

A: Dobrý den. Můžete mi předložit dvě karty a já vám řeknu, jestli dohromady tvoří pár, nebo ne.

U: *začátek*

Vývoj stavu dialogu:

New slot added: first_card
New slot added: second_card
Slot first_card must be filled
Slot second_card must be filled

Turn 2:

A: Jaké karty vás zajímají?

U: První karta je eso, druhá karta král.

Aktualizace stavu dialogu:

Extended values: first_card = eso

Extended values: second_card = král

Turn 3:

A: První karta je eso a druhá karta je král. Chcete potvrdit tyto karty? Ano, nebo ne.

U: Nechci.

Aktualizace stavu dialogu:

Slot first_card must be confirmed

Slot second_card must be confirmed

Turn 4:

A: Nechcete potvrdit první kartu eso a druhou kartu král. Budou smazány.

U: *konec*

Aktualizace stavu dialogu:

Slot first_card will be emptied

Slot second_card will be emptied

Slot first_card is emptied

Slot second_card is emptied

Historie dialogu:

Turn 1: Create['first_card', 'second_card']

Turn 2: CompEmpty['first_card'='eso', 'second_card'='král']

Turn 3: ConfUnconf['first_card', 'second_card']

Turn 4: Empty['first_card', 'second_card']

9.3 Vlakové spojení - SpeechCloud

V platformě SpeechCloud byl vytvořen jednoduchý dialogový systém pro nalezení vlakového spojení mezi českými krajskými městy. Uživatel je vyzván ke specifikaci z jakého města, do jakého města a jakým typem vlaku chce jet. Pokud uživatel explicitně neřekne, jakým typem vlaku chce jet, pak agent předpokládá, že je mu to jedno, a snaží se najít jakékoliv spojení. Pokud není rozpoznáno, odkud nebo kam chce uživatel jet, agent ho vyzve k doplnění informace. Pokud uživatel řekne dvě hodnoty pro výchozí nebo cílovou stanici, je vyzván k tomu, aby si jednu hodnotu vybral. Model úlohy (informace o vlakových spojeních) byla jednoduchá syntetická databáze. Z každého města jel do každého města vlak po dvouhodinových intervalech od 4:00 do 24:00 a typ vlaku byl každému spojení přiřazen náhodně. Čas, podle kterého dialog vyhledával vlakové spojení, byl nastaven jako čas začátku konverzace.

```
if (result_1.from_station.first and result_1.to_station.first):
    # bylo receno z a kam
    from_station = result_1.from_station.first
    to_station = result_1.to_station.first

    self.state.push(from_station=from_station, to_station=to_station)
    if result_1.train_type.first:
        # kontrola, jestli byl recen typ vlaku, pokud ne -> je to uzivateli jedno
        train_type = result_1.train_type.first
    else:
        train_type = "ANY"
    self.state.push(train_type=train_type)
    while True:
        if train_type == "ANY":
            result = await self.synthesize_and_wait_for_slu_result(text=f"Chcete jet z města {self.state.from_station.first_value} do města {self.state.to_station.first_value}. Chcete potvrdit tyto údaje? Ano, nebo ne.", timeout=4.)
        else:
            result = await self.synthesize_and_wait_for_slu_result(text=f"Chcete jet z města {self.state.from_station.first_value} do města {self.state.to_station.first_value} vlakem typu {train_types[self.state.train_type.first_value]}. Chcete potvrdit tyto údaje? Ano, nebo ne.", timeout=4.)
```

Obrázek 15: Ukázka zdrojového kódu dialogu

9.3.1 Příklad 1

V prvním příkladu uživatel řekne výchozí i cílovou zastávku a doplní i typ vlaku. Poté je vyzván agentem k potvrzení údajů, jež uživatel potvrdí, a je mu prezentován výsledek.

Turn 1:

A: Dobrý den. Vyhledám pro vás vlakové spojení.

U: *začátek*

Aktualizace stavu dialogu:

New slot added: to_station

New slot added: from_station

New slot added: time

New slot added: train_type

Slot to_station must be filled

Slot from_station must be filled

Slot time must be filled

Slot train_type must be filled

Turn 2:

A: Řekněte prosím, odkud a kam chcete jet, případně jakým typem vlaku.

U: Chci jet z Plzně do Brna rychlíkem.

Aktualizace stavu dialogu:

Extended values: time = 13:36:06

Extended values: from_station = Plzeň

Extended values: to_station = Brno

Extended values: train_type = R

Turn 3:

A: Chcete jet z města Plzeň do města Brno vlakem typu rychlík. Chcete potvrdit tyto údaje? Ano, nebo ne?

U: Ano.

Aktualizace stavu dialogu:

Slot to_station must be confirmed

Slot from_station must be confirmed

Slot time must be confirmed

Slot train_type must be confirmed

Slot to_station is confirmed

Slot from_station is confirmed

Slot time is confirmed

Slot train_type is confirmed

Turn 4:

A: Z města Plzeň do města Brno jede rychlík v 16:00.

U: Konec.

Aktualizace stavu dialogu:

Value was presented Brno for slot to_station

Value was presented Plzeň for slot from_station

Value was presented 13:36:06 for slot time

Value was presented R for slot train_type

Turn 5:

A: Konec dialogu, nashledanou.

U: *konec*

Aktualizace stavu dialogu:

Slot to_station will be emptied

Slot from_station will be emptied

Slot time will be emptied
Slot train_type will be emptied
Slot to_station is emptied
Slot from_station is emptied
Slot time is emptied
Slot train_type is emptied

Historie dialogu:

Turn 1: Create['to_station', 'from_station', 'time', 'train_type']

Turn 2: CompEmpty['to_station'='Brno', 'from_station'='Plzeň',
'time'=13:36:06, 'train_type'='R']

Turn 3: ConfUnconf['to_station', 'from_station', 'time', 'train_type']

Turn 4: Present['to_station', 'from_station', 'time', 'train_type']

Turn 5: Empty['to_station', 'from_station', 'time', 'train_type']

9.3.2 Příklad 2

V tomto příkladu řekne uživatel pouze výchozí stanici, načež je vyzván, aby doplnil i cílovou. Uživatel pak doplní cílovou stanici a potvrdí oba údaje a pak je mu prezentován výsledek.

Turn 1:

A: Dobrý den. Vyhledám pro vás vlakové spojení.

U: *začátek*

Aktualizace stavu dialogu:

New slot added: to_station

New slot added: from_station

New slot added: time

New slot added: train_type

Slot to_station must be filled

Slot from_station must be filled

Slot time must be filled

Slot train_type must be filled

Turn 2:

A: Řekněte prosím, odkud a kam chcete jet, případně jakým typem vlaku.

U: Chci jet z Ústí.

Aktualizace stavu dialogu:

Extended values: time = 13:37:47

Extended values: from_station = Ústí nad Labem

Extended values: train_type = ANY

Slot to_station must be filled

Turn 3:

A: Chcete jet z města Ústí nad Labem. Do jakého města chcete jet?

U: Do Liberce.

Aktualizace stavu dialogu:

Extended values: to_station = Liberec

Turn 4:

A: Chcete jet z města Ústí nad Labem do města Liberec. Chcete potvrdit tyto údaje? Ano, nebo ne?

U: Ano.

Aktualizace stavu dialogu:

Slot to_station must be confirmed

Slot from_station must be confirmed

Slot time must be confirmed

Slot train_type must be confirmed

Slot to_station is confirmed

Slot from_station is confirmed

Slot time is confirmed

Slot train_type is confirmed

Turn 5:

A: Z města Ústí nad Labem do města Liberec jede vlak v 14:00.

U: Konec.

Aktualizace stavu dialogu:

Value was presented Liberec for slot to_station

Value was presented Ústí nad Labem for slot from_station

Value was presented 13:37:47 for slot time

Value was presented ANY for slot train_type

Turn 6:

A: Konec dialogu, nashledanou.

U: *konec*

Aktualizace stavu dialogu:

Slot to_station will be emptied

Slot from_station will be emptied

Slot time will be emptied

Slot train_type will be emptied

Slot to_station is emptied

Slot from_station is emptied

Slot time is emptied

Slot train_type is emptied

Historie dialogu:

Turn 1: Create['to_station', 'from_station', 'time', 'train_type']

Turn 2: CompEmpty['to_station'=None, 'from_station'='Plzeň',
'time'=13:37:47, 'train_type'='ANY']

Turn 3: CompEmpty[to_station='Liberec']

Turn 4: ConfUnconf['to_station', 'from_station', 'time', 'train_type']

Turn 5: Present['to_station', 'from_station', 'time', 'train_type']

Turn 6: Empty['to_station', 'from_station', 'time', 'train_type']

9.3.3 Příklad 3

Ve třetím příkladu uživatel řekne nejdříve výchozí stanici, poté si ale zvolí jinou výchozí stanici. Je vyzván agentem k vyřešení inkonzistence, již uživatel vyřeší. Poté je po výzvě doplněna i cílová stanice, nakonec ale místo potvrzení hodnot si svoji volbu uživatel rozmyslí, a hodnoty jsou proto smazány.

Turn 1:

A: Dobrý den. Vyhledám pro vás vlakové spojení.

U: *začátek*

Aktualizace stavu dialogu:

New slot added: to_station

New slot added: from_station

New slot added: time

New slot added: train_type

Slot to_station must be filled

Slot from_station must be filled

Slot time must be filled

Slot train_type must be filled

Turn 2:

A: Řekněte prosím, odkud a kam chcete jet, případně jakým typem vlaku.

U: Dobrý den, chci jet z Budějovic.

Aktualizace stavu dialogu:

Extended values: time = 13:39:14

Extended values: from_station = České Budějovice

Extended values: train_type = ANY

Slot to_station must be filled

Turn 3:

A: Chcete jet z města České Budějovice. Do jakého města chcete jet?

U: Vlastně chci jet z Plzně.

Aktualizace stavu dialogu:

A new value will be added to slot from_station

Extended values: from_station = Plzeň

Turn 4:

A: Pro výchozí stanici jste již řekl České Budějovice, Plzeň. Musíte z nich vybrat jednu. Jaká je vaše odpověď?

U: Chci jet z Plzně.

Aktualizace stavu dialogu:

Value Plzeň was accepted in slot from_station

Assigned values: from_station = Plzeň

Slot to_station must be filled

Turn 5:

A: Chcete jet z města Plzeň. Do jakého města chcete jet?

U: Do Ostravy.

Aktualizace stavu dialogu:

Extended values: to_station = Ostrava

Turn 6:

A: Chcete jet z města Plzeň do města Ostrava. Chcete potvrdit tyto údaje? Ano, nebo ne?

U: Rozmyslel jsem si to, nechci.

Aktualizace stavu dialogu:

Slot to_station must be confirmed

Slot from_station must be confirmed

Slot time must be confirmed

Slot train_type must be confirmed

Turn 7:

A: Nechcete potvrdit odjezd z města Plzeň do města Ostrava. Údaje budou smazány.

U: *konec*

Aktualizace stavu dialogu:

Slot to_station will be emptied

Slot from_station will be emptied

Slot time will be emptied

Slot train_type will be emptied

Slot to_station is emptied

Slot from_station is emptied

Slot time is emptied

Slot train_type is emptied

Historie dialogu:

Turn 1: Create['to_station', 'from_station', 'time', 'train_type']

Turn 2: CompEmpty['to_station'=None, 'from_station'='České Budějovice', 'time'=13:39:14, 'train_type'='ANY']

Turn 3: AddNew['from_station'='Plzeň']

Turn 4: Disambig['from_station'='Plzeň']

Turn 5: CompEmpty['to_station'='Ostrava']

Turn 6: ConfUnconf['to_station', 'from_station', 'time', 'train_type']

Turn 7: Empty['to_station', 'from_station', 'time', 'train_type']

9.3.4 Příklad 4

V posledním příkladu uživatel řekne výchozí i cílovou stanici, ale ještě před potvrzením hodnot upraví výchozí stanici a vyřeší inkonzistenci. Pak už jenom potvrdí hodnoty a je mu prezentován výsledek.

Turn 1:

A: Dobrý den. Vyhledám pro vás vlakové spojení.

U: *začátek*

Aktualizace stavu dialogu:

New slot added: to_station

New slot added: from_station

New slot added: time

New slot added: train_type

Slot to_station must be filled

Slot from_station must be filled

Slot time must be filled

Slot train_type must be filled

Turn 2:

A: Řekněte prosím, odkud a kam chcete jet, případně jakým typem vlaku.

U: Dobrý den, chci jet z Plzně do Olomouce.

Aktualizace stavu dialogu:

Extended values: time = 13:41:01

Extended values: from_station = Plzeň

Extended values: to_station = Olomouc

Extended values: train_type = ANY

Turn 3:

A: Chcete jet z města Plzeň do města Olomouc. Chcete potvrdit tyto údaje? Ano, nebo ne?

U: Vlastně jedu do Ostravy.

Aktualizace stavu dialogu:

A new value will be added to slot to_station

Extended values: to_station = Ostrava

Turn 4:

A: Pro cílovou stanici jste již řekl Olomouc, Ostrava.

Musíte z nich vybrat jednu. Jaká je vaše odpověď?

U: Do Ostravy.

Aktualizace stavu dialogu:

Value Ostrava was accepted in slot to_station

Assigned values: to_station = Ostrava

Turn 5:

A: Chcete jet z města Plzeň do města Ostrava. Chcete potvrdit tyto údaje? Ano, nebo ne?

U: Ano.

Aktualizace stavu dialogu:

Slot to_station must be confirmed

Slot from_station must be confirmed

Slot time must be confirmed

Slot train_type must be confirmed

Slot to_station is confirmed

Slot from_station is confirmed

Slot time is confirmed

Slot train_type is confirmed

Turn 6:

A: Z města Plzeň do města Ostrava jede vlak ve 14:00.

U: Konec.

Aktualizace stavu dialogu:

Value was presented Ostrava for slot to_station

Value was presented Plzeň for slot from_station

Value was presented 13:41:01 for slot time

Value was presented ANY for slot train_type

Turn 7:

A: Konec dialogu, nashledanou.

U: *konec*

Aktualizace stavu dialogu:

Slot to_station will be emptied

Slot from_station will be emptied

Slot time will be emptied

Slot train_type will be emptied

Slot to_station is emptied

Slot from_station is emptied

Slot time is emptied

Slot train_type is emptied

Historie dialogu:

Turn 1: Create['to_station', 'from_station', 'time', 'train_type']

Turn 2: CompEmpty['to_station'='Olomouc', 'from_station'='Plzeň', 'time'=13:41:01, 'train_type'='ANY']

Turn 3: AddNew['to_station'='Ostrava']

Turn 4: Disambig['to_station'='Ostrava']

Turn 5: ConfUnconf['to_station', 'from_station', 'time', 'train_type']

Turn 6: Present['to_station', 'from_station', 'time', 'train_type']

Turn 7: Empty['to_station', 'from_station', 'time', 'train_type']

9.4 Možná vylepšení knihovny

Funkčnost knihovny během všech experimentů byla dostatečná z pohledu obstarávání slotů, jejich hodnot a udržování historie dialogu.

Jedním z vylepšení by mělo být doplnění knihovny o další callbacky. Například v Mycroft příkladu č.3 by mohl být Turn 3 (neporozumění hodnotě) rozdělen na Turn 3 a Turn 4. V Turn 3 by se pak stav ve skutečnosti neaktualizoval o nové hodnoty, ale do historie by mohla být uložena informace o tom, že uživateli nebylo porozuměno. Do historie by tedy byl přibyl záznam např. jako:

```
NotUnderstood['second_card'='eleven']
```

Dalším vylepšením knihovny by mohlo být využití neuronových sítí. Jelikož knihovna obstarává základní operace nad sloty a jejich hodnotami, mohly by být neuronové sítě naučeny rozhodovat, jaká akce a např. jaký callback a syntaktizovaný prompt (pobídka, výzva) budou podle historických promluv následovat. Pomocí neuronových sítí by bylo například možné z promluvy extrahovat jazykové entity a jejich hodnoty a poté dynamicky vytvářet či mazat sloty.

Na předchozí možné vylepšení navazuje využití knihovny pro stavovou reprezentaci podobně jako v kapitole Discriminative state tracking for spoken dialog systems, kde knihovna nezprostředkovává pouze reprezentaci stavu, ale i samotnou SLU část hlasového dialogového systému. Díky tomu by měla knihovna a uživatel mnohem větší kontrolu nad daty, která se v hlasovém dialogovém systému objevují. Zároveň by díky tomuto vylepšení bylo možné od použitého callbacku odvodit, jaký prompt syntetizovat uživateli.

Dalším vylepšením by mohlo být navázání spolupráce knihovny s NLG částí platform, ve kterých bude knihovna použita. Hlasový dialog totiž pracuje podle rovnice:

$$\begin{aligned}x(t+1) &= f(x(t), u(t)) \\ y(t) &= g(x(t), u(t))\end{aligned}\tag{5}$$

kde $x(t)$ představuje stav (řešený knihovnou) v kroku t , $u(t)$ je vstup od uživatele, $y(t)$ je výstupní rovnice z DM a NLG a f a g představují řešení reprezentace stavu, respektive způsob generování odpovědi na základě stavu a vstupu od uživatele. Propojení knihovny a NLG části by mohlo pomoci s řešením výstupní rovnice, jelikož data, která do této rovnice vcházejí, by byla zpracována a kontrolována už v samotné knihovně.

10 Závěr

Cílem této diplomové práce bylo navrhnutí a vytvoření softwarové knihovny v jazyku Python pro reprezentaci stavu hlasových dialogových systémů.

V první části diplomové práce byl popsán hlasový dialogový systém a jeho jednotlivé moduly pro lepší orientaci v dalších kapitolách, kdy se často přístupy a metody odkazují na různé části hlasového dialogového systému. Byla popsána struktura **hlasového dialogového systému** jako celku. Nejdříve byl představen modul **ASR** (Rozpoznání řeči) a postup výpočtu k získání textu z řeči. Poté byla vysvětlena funkce **SLU** (Porozumění řeči) se znalostním a statistickým přístupem k získání významu slov. Následně byl popsán **DM** (Dialogový manažer) a různé způsoby řízení hlasového dialogového systému. Nakonec byly představeny moduly **NLG** (Generování odpovědi) a **TTS** (Převod textu na řeč) a jejich přístupy k převodu textu do řeči.

Na začátku bylo nutné nastudovat problematiku stavové reprezentace a popsat několik metod a přístupů k reprezentaci stavu. První metodou byl **Discriminative state tracking for spoken dialog systems** představené v článku [11]. Cílem přístupu bylo nalézt řešení, které by neomezovalo množství hypotéz z důvodu možné ztráty správného výsledku. Řešením bylo využití **Dynamic discriminative state tracking** používajícím příznakových funkcí spojujících příznaky specifické pro hypotézu s korespondující hypotézou, čímž se sníží počet vah nutných k natrénování. Hlavním přínosem pro diplomovou práci byla nutnost knihovny umožňovat dynamické přidávání a odebírání jednotlivých slotů. Druhou metodou, představenou v článku [15] byl **Scalable multi-domain dialogue state tracking**, kde hlavním cílem bylo mít možnost neomezené množiny možných hodnot pro všechny sloty stavové reprezentace. V přístupu byly využity množiny kandidátů, které obsahují všechny kandidáty pro určitý slot a ve kterých se kvůli omezení velikosti množiny nenacházejí hodnoty s nulovým skóre. Díky způsobu trénování neuronových sítí je tento přístup pro sloty jednoduše přenositelný mezi různými doménami bez potřeby

extenzivního trénování nových sítí. Poznatkem pro diplomovou práci byl důraz na modularitu knihovny.

V další části diplomové práce bylo úkolem nastudovat a popsat způsoby reprezentace v moderních webových technologiích. První popsanou knihovnou byl **React**, JavaScript knihovna, vyvíjená Facebookem a používaná například v aplikaci Instagram. V knihovně existuje jeden stavový komponent obhospodařující stavy a zastřešující ostatní komponenty. Další způsob byla knihovna **Persistent** navržená v jazyce Python. Persistent nabízí možnost vytvoření základních objektů Pythonu, jako například seznam nebo slovník, ale v immutable verzi, a díky této vlastnosti byl použit v návrzích softwarové knihovny na reprezentaci stavu.

Třetím úkolem byla analýza a samotný návrh softwarové knihovny. K analýze a návrhu sloužil demo dialog vytvořený pro platformu **Mycroft**, který uživateli řekl, jestli jím řečené karty tvoří dohromady pár. V této části byla popsána struktura a způsob fungování Mycroftu. Doménu zde tvoří **Skills**, vstup uživatele představuje **Intent** a odpověď systému definuje část **Dialog**. Knihovna prošla několika změnami a výsledný kód pracoval se složitými slovníkovými strukturami, které představovaly jak samotný stav, tak i jeho části. Tento kód je vidět v kapitole Výsledný kód a jeho zhodnocení.

Pro svou složitost a obtížnou přenositelnost byl kód knihovny od základu předělán. Byla vytvořena nová analýza, kde se kladl důraz na udržitelnou historii dialogu a kompozitní stav obalující ostatní části stavové reprezentace. Z toho důvodu vyšel návrh knihovny obsahující tři třídy - **Value** představující hodnoty určitého slotu, **Slot** představující jednotlivé sloty a obalující třídu **Value** a **State** neboli kompozitní stav obalující třídu **Slot**. V knihovně jsou využity **callbacks**, které nastavují očekávání dialogového systému a jimiž se naplňuje historie dialogu, kde je vidět průběh života jednotlivých slotů. Výsledný kód je nahlédnutí v kapitole Výsledný kód.

Funkčnost knihovny byla demonstrována na třech dialozích. Jeden dialog byl vytvořen pro platformu Mycroft, pracující v angličtině, a byl jím upra-

vený demo dialog rozeznání páru/nepáru a dva byly vyhotoveny pro platformu SpeechCloud, operující v češtině, kde jeden byl demo dialog z Mycroftu předělaný do češtiny a druhý sloužil jako vyhledávač vlakového spojení mezi českými krajskými městy. Knihovna byla ve všech dialozích úspěšná a správně zprostředkovávala reprezentaci stavu a historii dialogu.

Další kapitolou byla možná vylepšení knihovny na reprezentaci stavu, kde hlavním poznatkem bylo využití neuronových sítí, díky nimž by se stala knihovna naprosto dynamickou a přenositelnou do jakékoliv domény.

Hlavní úkol diplomové práce, analýza a návrh softwarové knihovny pro reprezentaci stavu hlasových dialogových systémů, byl splněn a knihovna byla vytvořena, aby všem požadavkům, které na ni byly v analýze kladeny, bylo vyhověno.

Výsledný kód knihovny je dostupný zde:

<https://github.com/Filipfill123/StateRepresentation>

Dialog v Mycroftu je dostupný zde:

<https://github.com/Filipfill123/poker-hands-skill/tree/final>

Dialogy pro SpeechCloud jsou dostupné zde:

<https://github.com/Filipfill123/SpeechCloud>

Seznam požadavků z kapitoly Strukturizace knihovny je dostupný zde:

<https://github.com/Filipfill123/LibraryRequirements>

11 Literatura

Reference

- [1] BALTER, Michael. Striking Patterns: Skill for Forming Tools and Words Evolved Together. Science [online]. Science, ©2021. [cit. 2.2.2021]. Dostupné z: <https://www.sciencemag.org/news/2013/08/striking-patterns-skill-forming-tools-and-words-evolved-together>
- [2] BOHUS, Dan - RUDNICKY, Alexander. A “K Hypotheses + Other” Belief Updating Model [online článek]. 2006 [cit. 11.12.2020]. Dostupné z: <https://www.aaii.org/Papers/Workshops/2006/WS-06-14/WS06-14-003.pdf>
- [3] BOHUS, Dan - RUDNICKY, Alexander. The RavenClaw dialog management framework: Architecture and systems. *Computer Speech and Language*. 2009, č. 23, s. 332-361 [cit. 11.12.2020]. ISSN 0885-2308. Dostupné z: <https://www.sciencedirect.com/journal/computer-speech-and-language/vol/23/issue/3>
- [4] BOTHA, Rudolf (ed.). The Cradle of Language. Oxford: Oxford Linguistics, 2009. ISBN 978-0-19-954585-8. Dostupné také z: <https://books.google.cz/books?id=IVRkzK1VX1oC&printsec=frontcover&hl=cs#v=onepage&q&f=false>
- [5] BRITZ, Denny et al. Efficient Attention using a Fixed-Size Memory Representation [online článek]. 2017 [cit. 5.1.2021]. Dostupné z: <https://arxiv.org/abs/1707.00110>
- [6] CHAO, Yi-Hsiang et al. Improving GMM–UBM speaker verification using discriminative feedback adaptation. *Computer Speech and Language*.

- 2009, č. 23, s. 376-388 [cit. 11.12.2020]. ISSN 0885-2308. Dostupné z: <https://www.sciencedirect.com/journal/computer-speech-and-language/vol/23/issue/3>
- [7] COSTA-JUSSÀ, Marta R. - FONOLLOSA, José A.R. An Ngram-based reordering model. *Computer Speech and Language*. 2009, č. 23, s. 362-375 [cit. 11.12.2020]. ISSN 0885-2308. Dostupné z: <https://www.sciencedirect.com/journal/computer-speech-and-language/vol/23/issue/3>
- [8] GORE, Pranay. Using getattr, setattr, delattr, hasattr in Python. *Medium.com* [online]. [cit. 8.3.2021]. Dostupné z: <https://medium.com/@pranaygore/using-getattr-setattr-delattr-hasattr-in-python-6d79c6f9fda3>
- [9] HANNUN, Awni et al. Deepspeech: Scaling up end-to-end speech recognition [online článek]. 2014 [cit. 20.2.2021]. Dostupné z: <https://arxiv.org/abs/1412.5567>
- [10] LI, Jinyu et al. A unified framework of HMM adaptation with joint compensation of additive and convolutive distortions. *Computer Speech and Language*. 2009, č. 23, s. 389-405 [cit. 11.12.2020]. ISSN 0885-2308. Dostupné z: <https://www.sciencedirect.com/journal/computer-speech-and-language/vol/23/issue/3>
- [11] METALLINO, Angeliki et al. Discriminative state tracking for spoken dialog systems. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 4.-9. srpna 2013: sborník příspěvků konference [online], 466-475. Sofia, Bulharsko: Association for Computational Linguistics, 2013 [cit. 13.12.2020]. Dostupné z: <https://www.aclweb.org/anthology/P13-1046.pdf>

- [12] Mycroft AI. Adapt. *GitHub repository* [online]. 2020 [cit. 2.1.2021]. Dostupné z: <https://github.com/MycroftAI/documentation/tree/master/docs/mycroft-technologies/adapt>
- [13] Mycroft AI. Mimic2. *GitHub repository* [online]. 2018 [cit. 2.1.2021]. Dostupné z: <https://github.com/MycroftAI/mimic2>
- [14] PIETQUIN, Olivier et al. Compact and Interpretable Dialogue State Representation with Genetic Sparse Distributed Memory [online článek]. 2017 [cit. 10.10.2020]. Dostupné z: https://www.researchgate.net/publication/311895162_Compact_and_Interpretable_Dialogue_State_Representation_with_Genetic_Sparse_Distributed_Memory
- [15] RASTOGI, Abhinav et al. Scalable Multi-Domain Dialogue State Tracking [online článek]. 2018 [cit. 10.10.2020]. Dostupné z: <https://arxiv.org/pdf/1712.10224.pdf>
- [16] RAUX, Antoine - ESKENAZI, Maxine. A finite-state turn-taking model for spoken dialog systems. In: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, 10.-12. května 2009: sborník příspěvků konference [online], 629-637. Stroudsburg, USA: Association for Computational Linguistics, 2009 [cit. 4.4.2021]. Dostupné z: <https://dl.acm.org/doi/10.5555/1620754.1620846>
- [17] SKANTZE, Gabriel. Error Handling in Spoken Dialogue Systems. Stockholm: KTH Computer Science and Communication, Department of Speech, Music and Hearing, 2007. Doctoral Thesis, ISBN 978-91-7178-788-0. Dostupné také z: <https://www.speech.kth.se/prod/publications/files/3101.pdf>

- [18] ŠVEC, Jan. Knihovna SpeechCloud.dialog. Katedra kybernetiky ZČU, 2021. Interní dokument.
- [19] ŠVEC, Jan et al. Prototyp hlasového interaktivního systému vybrané domény mluvené angličtiny. Katedra kybernetiky ZČU, 2021. Interní dokument.
- [20] TETREAULT, Joel R. - LITMAN, Diane. A Reinforcement Learning approach to evaluating state representations in spoken dialogue systems. *Speech Communication*. 2008, č. 50 (8-9), s. 683-696 [cit. 2.3.2021]. ISSN 0167-6393. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S0167639308000617>
- [21] ULTES, Stefan et al. PyDial: A Multi-domain Statistical Dialogue System Toolkit. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics-System Demonstrations*, 30. července - 4. srpna 2017: sborník příspěvků konference [online], 73-78. Vancouver, Kanada: Association for Computational Linguistics, 2017 [cit. 13.12.2020]. Dostupné z: <https://www.aclweb.org/anthology/P17-4013.pdf>
- [22] WEN, Haoyang et al. Sequence-to-Sequence Learning for Task-oriented Dialogue with Dialogue State Representation. In: *Proceedings of the 27th International Conference on Computational Linguistics*, 20.-26. srpna 2018: sborník příspěvků konference [online], 3781-3792. Santa Fe, USA: Association for Computational Linguistics, 2018 [cit. 13.12.2020]. Dostupné z: <https://www.aclweb.org/anthology/C18-1320/>
- [23] Wikipedia contributors. Mycroft (software). *Wikipedia, The Free Encyclopedia* [online]. 2021 [cit. 2.2.2021]. Dostupné z: [https://en.wikipedia.org/w/index.php?title=Mycroft_\(software\)&oldid=1007810723](https://en.wikipedia.org/w/index.php?title=Mycroft_(software)&oldid=1007810723)