Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

PLZEŇ, 2022 **TOMÁŠ LEBEDA**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Tomáš LEBEDA**
Osobní číslo:	**A19B0303P**
Studijní program:	**B0714A150005 Kybernetika a řídicí technika**
Specializace:	**Umělá inteligence a automatizace**
Téma práce:	**Osobní hlasová asistentka**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Prostudujte problematiku hlasových dialogových systémů a seznamte se s hlasovou platformou SpeechCloud.
2. Navrhněte hlasový dialog „osobní asistentky“, který bude umožňovat základní obsluhu vybrané komunikační platformy (mail, messenger), reagovat na vnější podněty (domácnost, web, časovač). Použijte vhodnou reprezentaci stavu dialogu.
3. Dialog realizujte a otestujte.

Rozsah bakalářské práce: **30 – 40 stránek A4**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

Dodá vedoucí práce.

Vedoucí bakalářské práce: **Ing. Luboš Šmídl, Ph.D.**
Katedra kybernetiky

Datum zadání bakalářské práce: **15. října 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 15. října 2021

Prohlášení

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

Prohlašuji, že jsem bakalářskou 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. Luboši Šmídlovi, Ph.D. za jeho vstřícnost, lidský přístup a za to, že pod jeho vedením jsem měl při práci volnost, ale zároveň byl schopný a ochotný poradit v případě potřeby. Dále bych chtěl poděkovat své rodině a přátelům za jejich podporu a také pomoc při testování.

Abstrakt

Hlasoví asistenti jsou systémy, jejichž hlavním účelem je zjednodušit nebo automatizovat běžné každodenní činnosti. Uživatelé s nimi interagují pomocí hlasu, což umožňuje jejich aplikace v situacích, kdy má člověk obě ruce a/nebo oči plně zaměstnané, nebo má omezené zrakové či pohybové schopnosti. Tato práce se zabývá návrhem a implementací systému osobní hlasové asistentky s důrazem na snadnou modifikovatelnost a rozšiřitelnost, schopnou základního ovládání vybrané komunikační platformy a reakcí na vnější podněty. V práci je popsán návrh architektury celého systému i jednotlivých částí a zdůvodněné některé designové volby. Poté je popsána implementace a řešení problémů, které se během práce vyskytly.

Klíčová slova

hlasové dialogové systémy, osobní asistentka, zpracování přirozeného jazyka, porozumění řeči, řídicí pravidla, dialogový manažer, stavová reprezentace, bezkontextové gramatiky

Abstract

Voice assistants are systems with main purpose to simplify or automatize common everyday tasks. Users interact with them through voice, allowing them to be used in situations where user has both hands and/or eyes fully occupied, or has limited visual or physical abilities. The topic of this thesis is design and implementation of personal voice assistant with emphasis on easy modifiability and extensibility, capable of basic management of the selected communication platform and reacting to external events. The thesis describes the design of the whole system as well as individual parts and reasons some design choices. The implementation is described along with problems that occurred during the work and their solutions.

Keywords

Spoken dialog systems, personal assistant, natural language processing, spoken language understanding, dialog manager, state representation, context-free grammars

Obsah

1 Úvod

Použití hlasových asistentů je dnes poměrně běžnou věcí, zejména pak v anglicky mluvících zemích, kde jsou používané známé systémy jako Alexa, Google Assistant nebo Siri. Hlavním účelem těchto systémů je zjednodušení každodenních činností a nebo jejich plná či částečná automatizace. Jak již název napovídá, jedním z hlavních rysů těchto systémů je schopnost interagovat s uživateli pomocí hlasu a přirozené řeči. Tato vlastnost umožňuje jejich aplikace v situacích, kdy by použití jiných způsobů interakce (typicky grafické nebo textové) bylo nemožné nebo nepraktické, většinou z důvodů zaměstnání uživatele jinou činností, například při řízení automobilu.

Cílem této práce je navrhnout a vytvořit systém osobní hlasové asistentky, která bude schopna s uživatelem komunikovat v českém jazyce. Ačkoli existují hlasoví asistenti schopní komunikace v českém jazyce [1], tak jejich použití nedosahuje takové úrovně přirozenosti a otevřenosti jako anglicky mluvící alternativy. Důvodem je převážně vysoká složitost a variabilita českého jazyka, která komplikuje počítačové zpracování. Pro zpracování a analýzu přirozené češtiny je využita platforma SpeechCloud [2] vyvinutá na Katedře kyberneticky Západočeské univerzity v Plzni.

Při návrhu a realizaci bude kladen důraz na univerzálnost a snadnou rozšiřitelnost, aby nebyl výsledek fixován pouze na potřeby této práce, ale aby bylo možné výsledný systém snadno modifikovat a rozšířit i pro jiná použití.

Příkladem možného použití by mohlo být sledování předpovědi počasí a kalendáře, takže by systém dokázal například ráno uživatele varovat, že má dle předpovědi pršet a měl by se na to vybavit. Jiným příkladem by mohlo být hlídání změn na zvolených webových stránkách, například sledování ceny a dostupnosti v internetových obchodech u vybraných produktů. Asistentka by tak uživatele mohla například upozornit, že nějaké zboží je ve slevě, nebo že například bylo právě naskladněno a je opět dostupné. V kombinaci s prvky chytré domácnosti by systém mohl také například automaticky regulovat topení či klimatizaci tak, aby byla v místnosti efektivně udržována příjemná teplota. Při integraci chytrého kávovaru by mohla na uživatele ráno čekat i čerstvě uvařená káva.

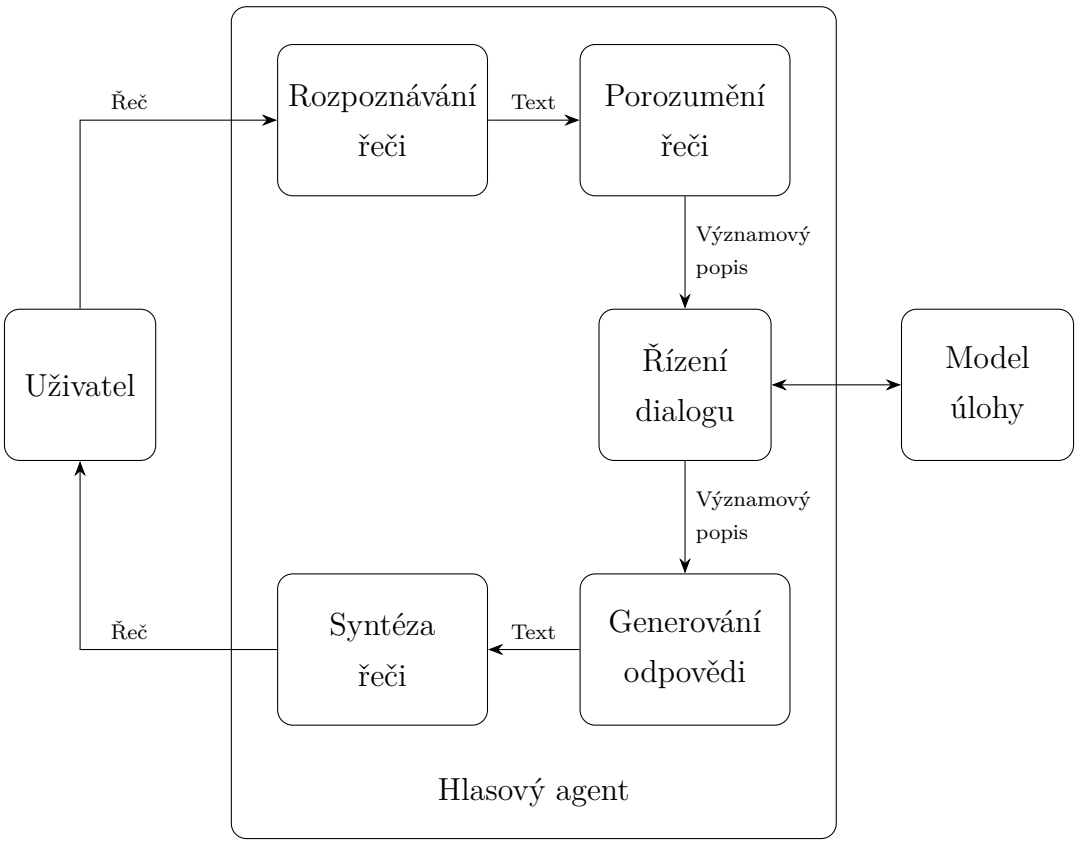
Jako základní funkce výsledné hlasové asistentky byly v této práci vybrány následující úlohy, které reprezentují různé způsoby interakce systému s vnějším světem. Jsou jimi obsluha emailové schránky, která zahrnuje kontrolu nepřečtených zpráv, výběr emailů a přečtení jejich vybraných částí a možnost nadiktování odpovědi. Dále bude systém sledovat zadané RSS zdroje a reagovat na hlasové příkazy uživatele. Upozornění od osobní asistentky uživatel dostane v podobě hlasových promluv nebo vizuálních notifikací, kdy systém rozhodne, který typ je vhodný v závislosti na aktivitě uživatele a denní době. Navíc bude systém schopný automaticky vykonávat některé vybrané akce na uživatelském počítači.

V práci bude popsán návrh a architektura celého systému i jeho jednotlivých částí, zdůvodnění designových voleb. Poté bude popsána realizace včetně problémů, které se během práce vyskytli, spolu s jejich řešením.

2 Problematika hlasových dialogů

Hlasový dialogový systém je počítačový systém, který umožňuje komunikaci mezi člověkem a strojem pomocí lidské řeči. Využití těchto systémů je značně rozsáhlé, může se jednat o různé infolinky, telefonní spojovatelky nebo třeba virtuální asistenty. Pomocí hlasových dialogových systémů je ale možné studovat také komunikace mezi lidmi nebo dokonce i mezi dvěma stroji (když je uživatel simulován) [3].

Schéma hlasového dialogového systému je znázorněno na Obrázku 1, kde je možné vidět, že se jedná o netriviální problematiku zahrnující množství dílčích úloh.



Obrázek 1: Schéma hlasového dialogového systému [3]

Uživatelem pronesená řeč je zachycena jako akustický signál, který je v bloku „Rozpoznávání řeči“ převeden na text. Tento text by měl být v ideálním případě přepisem toho, co uživatel řekl. Rozpoznaný text je poté zpracován v bloku „Porozumění řeči“, kde je ze vstupního textu extrahována informace o jeho významu v daném kontextu

úlohy. Tato informace je předána do bloku „Řízení dialogu“, kde je spolu se stavem z bloku „Model úlohy“ použita k vyhodnocení dalších akcí a odpovědí systému. Následně je toto rozhodnutí převedeno do textové podoby srozumitelné člověku, což reprezentuje blok „Generování odpovědi“. Odpověď je poté nutno převést opět na akustický signál, který je přehrán nahlas uživateli na výstupu. Tuto část zajišťuje blok „Syntéza řeči z textu“. Uživatel si poslechne odpověď, v případě potřeby odpoví a cyklus se opakuje, dokud není splněn cíl dialogu.

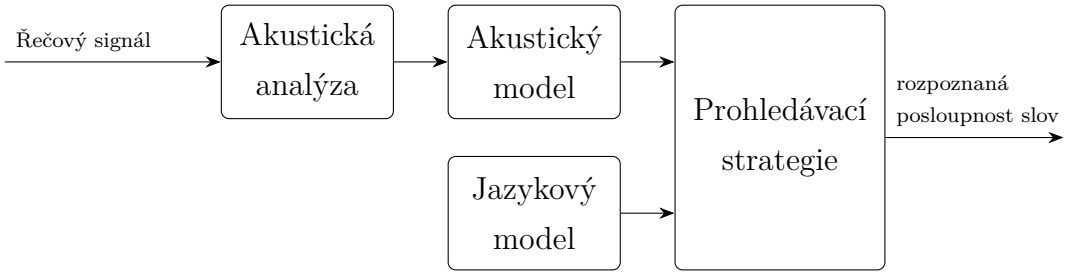
2.1 Rozpoznání řeči

Schopnost systému rozpoznat, co uživatel řekl, je jedním ze základních požadavků na funkční hlasový dialogový systém. Vzhledem k tomu, že rozpoznávání řeči nebylo hlavním předmětem této práce, budou zde popsány jen základní principy.

Jedná se o proces převodu zachyceného akustického signálu (řečníkovy promluvy) do textové podoby, která by měla v ideálním případě odpovídat přepisu toho, co bylo řečeno. Podle učebního textu [4] je možné kategorizovat úlohy rozpoznávání řeči podle následujících faktorů:

- způsob řeči → rozpoznávání izolovaných slov nebo plynulé (souvislé) řeči
- velikost slovníku → rozlišují se systémy s malým slovníkem (pouze hlasové povely), se středně velkým slovníkem (úzce vymezené promluvy, například předpověď počasí) a nebo systémy s velkým slovníkem (lze hovořit přirozeně na libovolné téma)
- počet uživatelů → systémy mohou být závislé na řečníkovi (systém používá pouze jeden uživatel) nebo mohou být nezávislé (systém používá více uživatelů)

Na Obrázku 2 je znázorněno schéma procesu rozpoznávání řeči. Jak je na této vizualizaci vidět, proces rozpoznání řeči začíná akustickou analýzou vstupního řečového signálu. Zachycený vstupní akustický signál je zpracován metodami frekvenční a časové analýzy a převeden na posloupnost příznakových vektorů. Ty jsou poté předány do akustického modelu, kde dojde k jejich převedení na posloupnost akustických jednotek. Tato posloupnost akustických jednotek je použita v kombinaci s jazykovým modelem jako vstup pro prohledávací strategii, jejímž cílem je najít nejpravděpodobnější posloupnost slov odpovídající vstupní promluvě [4].



Obrázek 2: Schéma rozpoznávání řeči

Výstupy mají textovou podobu, která je čitelná člověkem a je vhodná pro zobrazení. Vedle nejpravděpodobnější posloupnosti slov mohou být ve výstupu ještě také další alternativní hypotézy, které je možné využít při vyhodnocování významů v procesu porozumění řeči [3].

Jelikož je rozpoznaný text výchozím bodem pro porozumění textu, které je následně vstupem pro rozhodování o dalším chování celého dialogového systému, tak je pochopitelné, že přesnost a kvalita rozpoznávání řeči je kritickým faktorem pro správnou funkčnost hlasových dialogových systémů. Chyby při rozpoznávání se mohou projevit jako chybné hodnoty aktuálního stavu hlasového dialogového systému a korekce je časově náročná [3].

2.2 Porozumění řeči

Porozumění řeči (anglicky Spoken Language Understanding, zkráceně SLU) je další z klíčových součástí hlasových dialogových systémů. Jak již bylo zmíněno výše, tato část zajišťuje extrakci významu z rozpoznaného textu. Získané významy jsou použité pro rozhodování o dalších akcích systému. Podle učebního textu [3] je možné rozlišovat globální významové koncepty (například (ne)souhlas, požadavek, odpověď) nebo lokální významové entity (například datum, čas, pozdrav nebo jméno).

Velká variabilita přirozeného jazyka umožňuje vyjádřit stejný požadavek nebo informaci mnoha způsoby, což je hlavním problémem, který je při porozumění řeči potřeba řešit. Kromě přirozené variability řeči je potřeba uvažovat s možností chybného výsledku rozpoznávání řeči. Existuje množství metod a technologií, které jsou používané pro získávání významového popisu ze vstupního textu [5], v této práci bylo použito bezkontextových gramatik, které jsou popsány v sekci 2.2.1.

Některé metody používané pro analýzu významu vstupního textu informací:

1. **Tokenizace**

   Typicky se jedná o jeden z prvních kroků, kdy dojde k normalizaci vstupního textu. Jedná se především o rozdělení vstupního textu do diskrétních částí - tokenů - se kterými je možné dále pracovat snadněji. Tokenem může být slovo, interpunkční znaménko nebo třeba číslice [5].

2. **Metoda Bag of Words**

   Tento přístup nepoužívá lingvistické znalosti, pouze počítá výskyty jednotlivých slov a jeho variant, ze kterých poté vytvoří vektorovou reprezentaci daného textu. Používá se především pro vyhledávání mezi dokumenty podle zadaného požadavku, kdy se porovnávají vektorové reprezentace obou stran. Výhodou je jednoduchost, ale za cenu nižší informační hodnoty této reprezentace (ztrácí se například informace o pořadí a umístění slov) [5].

3. **Latentní sémantická analýza**

   Stejně jako předchozí metoda, tak ani tato nevyžaduje lingvistické znalosti a vstupní text je opět reprezentován pomocí „Bag of Words“ reprezentace, ovšem s tím rozdílem, že jednotlivé prvky nejsou přímo daná slova, nýbrž jejich významy či koncepty. Tato reprezentace umožňuje pracovat se vzory slov, které se například často vyskytují blízko sebe [5].

4. **Regulární výrazy**

   Metoda využívající regulární výrazy se opírá o pattern-matching algoritmy vstupních textových řetězců. Jednotlivé vzory, oproti kterým se testuje vstupní text, jsou ve formě regulárních výrazů. Tento přístup je velmi široce využívaný v případech, kde mají vstupní texty předvídatelnou strukturu [5].

5. **Metoda Part-of-Speech Tagging**

   Tato metoda značkuje slova vstupního textu podle jejich syntaktických a gramatických významů (slovní druhy, větné členy, ...). K tomu je možné použít regulárních výrazů a pattern-matching, ale i jiné, složitější metody. Při značkování je možné využívat heuristických pravidel a nebo statistického přístupu [5].

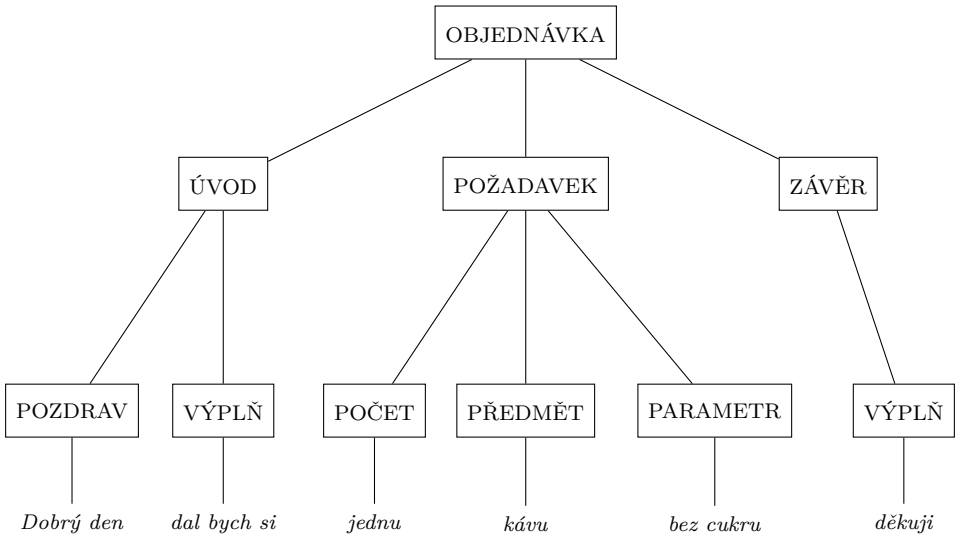
6. **Gramatiky**

   Použití gramatik je detailněji popsáno v další kapitole 2.2.1.

2.2.1 Použití sémantických gramatik

Vstupní text je možné rekurzivně rozkládat na menší části, takže vznikne stromová struktura. Tato stromová struktura lze poté analyzovat a reprezentovat pomocí syntaxe formálních jazyků. Nejmenší části, které jsou dále nedělitelné, se označují jako *primitiva* a slouží jako *terminální symboly*, kterými jsou stromové struktury zakončené (jsou to listy). Množina těchto terminálních symbolů se pak označuje jako *abeceda formálního jazyka*. Kromě terminálních symbolů jsou ještě zavedené pomocné *neterminální symboly*, které v grafu reprezentují nelistové uzly a lze je dále rozložit [6].

Na Obrázku 3 je znázorněn rozklad věty „*Dobrý den, dal bych si jednu kávu bez cukru, děkuji.*“.



Obrázek 3: Ukázka sémantického rozkladu věty

Stejně jako v běžném přirozeném jazyce, posloupnost terminálních symbolů (prvků abecedy) se označuje jako *slovo* (formálního jazyka). Všechny předměty, které lze pomocí formálního jazyka popsat, jsou popsané pomocí takzvaných *substitučních pravidel*. Tato substituční pravidla říkají, jak je možné jednotlivé prvky skládat. Substituční pravidla lze také použít rekurzivně, což umožňuje popisovat velmi bohaté množiny všech možných vyjádření, která lze v daném formálním jazyce poskládat [6].

Gramatiku je možné popsat jako čtveřici

$$G = (V_N, V_T, \rho, S\ )$$

kde V_N je množina neterminálních symbolů a V_T je množina terminálních symbolů (tyto množiny musí být disjunktní). S je počáteční (kořenový) symbol gramatiky a ρ je množina substitučních pravidel [6].

Gramatiky je možné rozdělit do několika skupin podle typu pravidel a nebo jejich komplexnosti. Nejjednodušším typem gramatiky je regulární gramatika, složitější je poté bezkontextová gramatika a ještě složitější jsou kontextové gramatiky a nebo gramatiky typu 0 [6].

Jak již bylo několikrát zmíněno, přirozená lidská řeč je velmi složitá a je proto obtížné ji nějak formálně reprezentovat. Regulární gramatikou nelze zachytit vztah mezi částmi, které nejsou bezprostředně vedle sebe, což pro zpracování řeči není vhodné. Například základní skladební dvojice ve větě má značnou vzájemnou významovou vazbu, ale velmi často se vyskytuje mezi podmětem a přísudkem ještě množství dalších slov. Pomocí složitějších typů gramatik, jako jsou kontextové gramatiky a nebo gramatiky typu 0, by bylo možné lépe zachytit informace v textu, ale komplexnost těchto gramatik je natolik vysoká, že práce s nimi je nepraktická a výpočetně velmi náročná. Z těchto důvodů se nejčastěji pro porozumění řeči používá bezkontextových gramatik, které poskytují kompromis mezi složitostí a schopností zachytit informace v přirozeném jazyce [6].

Jednou z nevýhod použití bezkontextových gramatik je (jak je z názvu zřejmé) absence kontextu, který může být v některých případech podstatný. Dalším problémem je pak fakt, že pomocí gramatik složených ze substitučních pravidel, terminálních a neterminálních symbolů je možné sestavit strukturu, která sice dané gramatice odpovídá, ale v přirozeném jazyce nedává žádný smysl. Částečným řešením tohoto problému je aplikace statistických metod a přiřazení pravděpodobností výskytu pro jednotlivé struktury [6].

Po analýze vstupního textu za použití bezkontextových gramatik tedy dostaneme na výstupu informaci o tom, zda vstupní text odpovídá dané gramatice a případně strukturu jednotlivých rozpoznaných entit, která reprezentuje významový popis vstupního textu.

2.3 Řízení dialogu

Řízení dialogu je další velmi důležitou součástí hlasových dialogových systémů. Jak je možné z názvu odhadnout, tak se jedná o proces rozhodování o tom, jak dále postupovat, jaké odpovědi vygenerovat a případně jaké akce vykonat.

Stejně jako všechny úlohy řízení, i zde je potřeba mít k dispozici nějakou informaci o současném stavu úlohy. V případě hlasových dialogových systémů ale není možné, aby uživatel mohl přímo pozorovat vnitřní stav systému a stejně tak systém nemůže přímo pozorovat stav uživatele (musel by umět „číst myšlenky“). Systém tedy musí získávat informace pouze prostřednictvím významového popisu řeči, který obdrží z předchozího porozumění řeči.

Na základě tohoto významového popisu v kombinaci se svým aktuálním vnitřním stavem může hlasový agent rozhodovat o dalších akcích. Těmito akcemi může být aktualizace svého vnitřního stavu nebo vygenerování odpovědi pro uživatele. Aktualizace stavu pak může pochopitelně ovlivnit právě řešenou úlohu, což může rekurzivně vést opět ke změně dalších stavů [3].

Řízení dialogu lze dělit podle iniciativy:

1. **Iniciativa hlasového agenta**
   V tomto typu iniciativy funguje systém tak, že pokládá uživateli otázky a uživatel na ně pouze odpovídá. Může se jednat o odpovědi typu ano/ne, nebo nějaké odpovědi s předem danou strukturou, či obecné odpovědi. Samozřejmě s rostoucí variabilitou odpovědi roste i náročnost na správné zpracování, jak plyne z předchozího popisu rozpoznávání a porozumění řeči [3].

2. **Iniciativa uživatele**
   Tento typ dialogového systému pracuje typicky tak, že uživatel zadává příkazy, které agent zpracovává. Příkazy, které je agent schopen zpracovat se opět mohou lišit komplexností, může se jednat například pouze o předem definované fráze a nebo se může jednat o věty přirozeného jazyka. Při tomto stylu řízení dialogu má uživatel větší kontrolu nad úlohou, ale je potřeba, aby dopředu věděl, jaké příkazy systém dokáže zpracovat a jak s agentem komunikovat [3].

3. **Smíšená iniciativa**
   Toto je styl řízení dialogu, který je nejbližší přirozenému hovoru mezi dvěma

lidmi. Jedná se o dialog, kde se strany mohou střídat v přebírání iniciativy. Typicky se jedná o pokládání doplňujících otázek, kdy se hlasový agent může například doptat na nějakou informaci, která je kritická pro rozhodování o dalším postupu a uživatel ji ještě neřekl [3].

Dále lze ale řízení dialogu také dělit podle toho, jakou má strukturu:

1. **Tahový dialog**
   Jedná se o takový typ dialogu, kde hlasový agent čeká na nějaký vstup od uživatele (promluvu), kterou zpracuje a vygeneruje odpověď. Tato sekvence je považována za jeden tah a tyto tahy jsou brány jako diskrétní události. Z toho poté plyne, že v tomto typu dialogu se stav hlasového agenta mění diskrétně, vždy během daného tahu [3].

2. **Inkrementální dialog**
   Tento typ dialogu je bližší běžné lidské komunikaci, protože hlasový agent zpracovává vstup od uživatele průběžně (ideálně v reálném čase). Hlavní výhodou této struktury je to, že hlasový agent může ihned reagovat na případné nejasnosti. Například může uživatele ihned zastavit a požádat jej, aby něco zopakoval nebo doplnil. Z popisu by mělo být zřejmé, že v tomto případě se bude stav hlasového agenta měnit průběžně [3].

V přirozené komunikaci mezi lidmi je běžné, že jeden řečník může druhého náhle přerušit, například mu může „skočit do řeči“. To samé se může stát i v případě, kdy komunikuje člověk se strojem. V terminologii hlasových dialogových systémů se tato událost označuje jako *vboření* (anglicky barge-in) [3]. Dobře navržený hlasový dialogový systém by měl s možností vboření počítat a nebo být vůči ní dostatečně robustní.

Další přirozenou vlastností přirozené komunikace mezi lidmi je vzájemné potvrzování informací. Typicky se jedná o situaci, kdy jedna strana zopakuje to, co zaznamenala a čeká na potvrzení od druhé strany, že skutečně zachytila a zpracovala promluvu správně. Protože uživatel a agent nepozorují přímo navzájem své stavy, je schopnost vzájemného potvrzování užitečná i pro úlohy hlasových dialogových systémů. V případě, že hlasový agent zaregistruje nějakou nezvyklost v promluvě, nebo třeba detekuje nesrovnalost ve svém vnitřním stavu, může se uživatele zeptat, zda sku-

tečně řekl danou věc a nebo si vyžádat potvrzení nějaké dřívější informace. Podle odpovědi uživatele pak přirozeně hlasový agent upraví svůj stav. Toto potvrzování je možné rozlišit podle způsobu, jakým je potvrzení předáno [3]:

- Explicitní - jedná se standardně o otázky typu ano/ne a k nim příslušející odpovědi.
- Implicitní - v tomto případě je potvrzení přímo jako součást následující výzvy a podle odpovědi uživatele na výzvu je vyhodnoceno samotné potvrzení.

2.4 Stav úlohy

Stav je další klíčovou součástí úlohy, protože jak již bylo řečeno v předchozích sekcích, je nezbytný pro správnou funkčnost řízení dialogu. Podle učebního textu [3] je možné stav hlasového dialogu rozdělit na následující části:

1. **Stav uživatele**
   Tento stav je reprezentací uživatele, kterou má hlasový agent uloženou v paměti. Obsahem by měl být cíl a motivace interakce, tedy nějaká představa systému o tom, čeho chce uživatel dosáhnout a proč toho chce dosáhnout. Poté v tomto stavu může být uložena informace o mentálním rozpoložení uživatele a případně znalosti a zkušenosti uživatele, podle čehož mohou být například různě formulovány otázky [3].
2. **Stav agenta**
   Jedná se o reprezentaci veškeré komunikace, která proběhla mezi uživatelem a hlasovým agentem od začátku dialogu. Pomocí stavu agenta je možné parametrizovat ostatní části (rozpoznávání, porozumění, syntézu řeči) [3].
3. **Stav úlohy**
   Jedná se o samotnou reprezentaci dané řešené úlohy. Často je v reálných situacích natolik komplexní, že je uchovávána mimo řízení dialogu. Stav úlohy je možné ovlivňovat pouze nepřímo a to řízením a výstupem dané úlohy [3].

2.5 Syntéza řeči z textu

Syntéza řeči z textu (anglicky text-to-speech, zkráceně TTS) je proces, kdy dochází k vygenerování audiosignálu, který by měl nahrazovat lidské předčítání textu. Výsledek syntézy je ovlivněn mnoha parametry, nejvýraznějšími jsou vlastnosti hlasu, který je použit pro syntézu (pohlaví, intonace, hloubka, rychlost hlasu). Jedním z hlavních problémů syntézy řeči je snaha dosáhnout přirozenosti, aby člověk nedokázal rozeznat, zda text přečetl jiný člověk a nebo stroj. Samozřejmě ne ve všech úlohách je tato vlastnost potřebná [3].

Syntéza řeči z textu se v hlasových dialogových systémech využívá k předání informace uživateli pouze pomocí zvuku. Výhodou tohoto výstupu je, že uživatel může informaci přijmout i ve chvíli, kdy má plně zaměstnané oči a nemůže si informaci přečíst nebo prohlédnout. Navíc může být pro některé uživatele a situace tato forma komunikace přirozenější a pohodlnější. Dalším využitím pak může být komunikace stroje s uživateli, kteří mají omezené zrakové nebo pohybové schopnosti a jiné formy interakce by pro ně byly nepraktické.

2.6 Platforma SpeechCloud

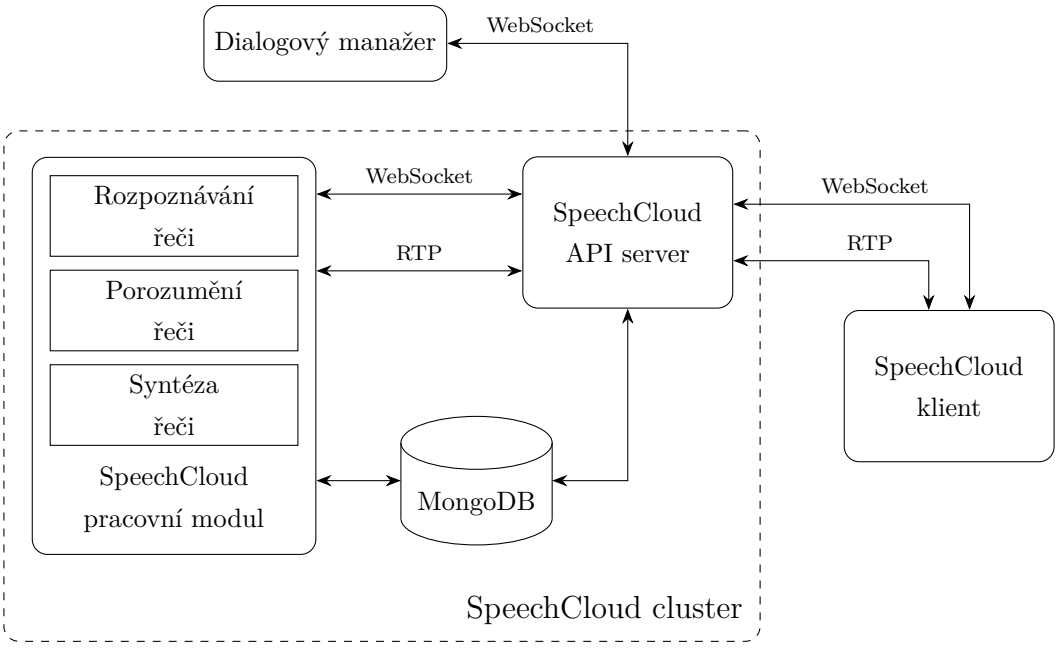
SpeechCloud je systém, který poskytuje nástroje pro řešení jednotlivých úloh spojených s hlasovými dialogovými systémy, které byly popsány v předchozích částech. Hlavními poskytovanými nástroji jsou systémy pro rozpoznávání, porozumění a syntézu řeči. Umožňuje komunikaci mezi klientem a serverovým clusterem zajišťujícím zmíněné systémy. Systém byl vytvořený na Západočeské univerzitě v Plzni [7].

Pro přenos dat mezi klientem a serverem je využit protokol WebRTC. Je to prohlížečové API, které umožňuje získávání audiosignálu z mikrofonu a přehrávání audiosignálu na reproduktorech klientského zařízení. Vedle WebRTC protokolu využívá platforma SpeechCloud také JavaScript a Session Description Protokol (SDP). Na straně serveru je s využitím Session Initiation Protocol (SIP) připojen prohlížeč a jednotlivé řečové procesory (pro rozpoznání, porozumění, syntézu řeči, ...). Samotný přenos paketů obsahujících audiosignál je zajištěn pomocí Realtime Transport Protocol (RTP). Propojení prohlížeče, jednotlivých alokovaných řečových procesorů a zajištění potřebných komunikačních úloh je řešeno s využitím softwaru FreeSwitch. Protože pomocí webového prohlížeče není možné komunikovat po libovolném TCP/IP

portu s libovolným protokolem, je spojení zabaleno pomocí softwaru FreeSwitch do SIP-over-Websocket rozhraní. Po inicializaci SIP relace jsou SDP a RTP pakety odesílány přímo prohlížečem jako součást WebRTC komunikace [7].

Využití webového prohlížeče umožňuje kromě přenosu audiosignálu a ovládání klientského hardwaru také interakci pomocí běžných periferií jako jsou myš a klávesnice, což zvyšuje versatilitu a uživatelskou přívětivost. Celkově je tady možné, aby uživatel ovládal systém pomocí svého hlasu a zároveň klasického grafického rozhraní s využitím myši a klávesnice [7].

Kromě webových prohlížečů využívajících WebRTC umožňuje SpeechCloud připojení jiných typů klientů, kteří mohou mít například vlastní hardware, příkladem může být Google VoiceKit. Pro tyto ostatní klienty je přístupný klasický SIP protokol [7].



Obrázek 4: Schéma systému SpeechCloud [7]

Architektura SpeechCloudu je postavená tak, aby umožňovala plně využít výpočetních schopností clusterového řešení. Klient interaguje se SpeechCloud API serverem a zahájí relaci na dané URL adrese, která musí mít požadované specifikace jednotlivých řečových procesorů. API server poté alokuje potřebné pracovní moduly pro

danou relaci a začne směrovat audiosignál pomocí softwaru FreeSwitch přímo mezi klientem a přiřazeným pracovním modulem. Architektura je vizualizována na Obrázku 4. Kromě audia jsou mezi klientem a přiděleným pracovním modulem posílané ještě kontrolní zprávy, které mají JSON formát. Server tyto zprávy validuje a ukládá si je pro případné ladění a generování statistik [7].

Dialogový manažer je implementovaný jako knihovna v jazyce Python. Samotná knihovna poskytuje obecnou „prázdnou“ implementaci, která je připravena na to, aby bylo chování definováno vývojářem konkrétního systému. Základem pro implementaci chování dialogového manažeru je odesílání požadavků na SpeechCloud API server a příjem událostí. Význačným rysem je schopnost asynchronního běhu, což je ideální pro situace, kdy počítač komunikuje s relativně pomalými nebo vzdálenými zařízeními a musí čekat delší dobu na odezvu. Dialogový manažer většinu času čeká na nějaké události, kterými může být například vypršení časovače nebo výsledek rozpoznávání či konec syntézy. Z tohoto důvodu je použití asynchronního běhu velmi výhodné [2].

Pracovní modul je proces, který zajišťuje základní úlohy potřebné pro chod celého hlasového dialogového systému. Typicky se jedná o rozpoznávání řeči, porozumění řeči a nebo syntézu řeči. Tyto pracovní moduly využívají ke své činnosti vlastní slovníky a modely [7].

Na každý pracovní modul jsou standardně přidělena dvě výpočetní CPU jádra. Protože ale za dobu řešení úlohy hlasového dialogového systému není potřeba neustále rozpoznávat a zpracovávat řeč (jsou chvíle, když uživatel nic neříká), tak by permanentní přidělení výpočetních jader bylo plýtváním výpočetního výkonu. Z tohoto důvodu je architektura SpeechCloudu navržena tak, aby bylo možné nečinná jádra zapůjčit i ostatním pracovním modulům a tím tak lépe využít výpočetní potenciál [7].

Pracovní moduly mají ještě své post-procesory, pomocí kterých jsou prováděné různé dokončovací operace, například hledání alternativních nejlepších hypotéz a hledání sémantických entit pro porozumění řeči. K těmto činnostem jsou požívané gramatiky splňující Speech Recognition Grammar Specification, která je veřejně dostupná na adrese `https://www.w3.org/TR/speech-grammar/` [7].

Kromě API serveru a pracovních modulů má SpeechCloud také nástroje pro správu modelů, například pro interpolaci malých specifických jazykových modelů s využitím velkého obecného jazykového modelu. To umožňuje navrhovat a konstruovat modely, které mohou být úzce specifikované pro nějakou oborovou doménu, zejména je pak užitečná možnost používat specifické příkazy a slova, která se nevyskytují příliš často v běžném jazyce, ale jsou často používaná v dané oblasti [7].

Platforma SpeechCloud je použita na mnoha existujících systémech [7]. Příkladem je *An Automatic Training Tool for Air Traffic Control Training* [8], *Multimodal Dialog with MALACH audiovisual Control Trainees* [9] a nebo backendové řešení HTTP rozpoznávač UWebASR [10].

3 Návrh a architektura

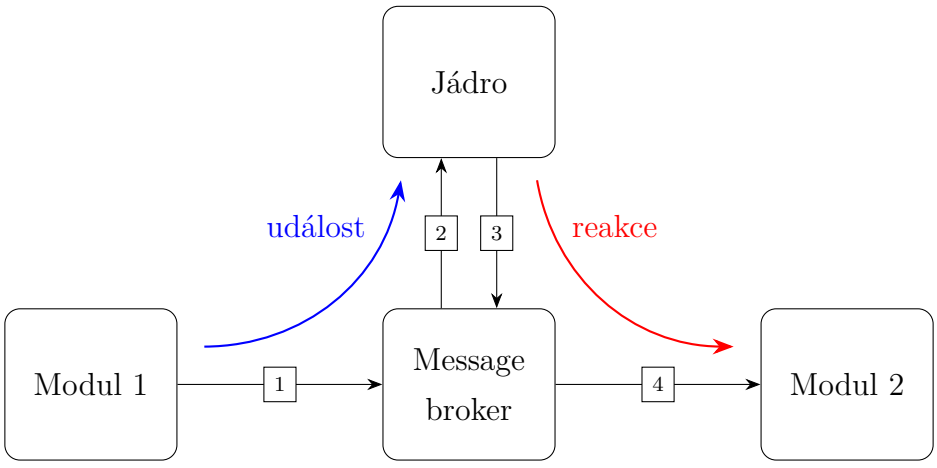
V této kapitole je popsán obecný návrh a architektura celého systému. Budou zde popsány hlavní myšlenky, ze kterých návrh vychází a odůvodněné některé designové volby. Následně bude popsáno chování a účel jednotlivých části celého systému.

3.1 Základní koncept a hlavní principy

Při navrhování architektury celého systému byl kladen důraz na to, aby byl systém konceptuálně jednoduchý, ale zároveň modulární s dostatečně univerzálním rozhraním. Kombinace těchto vlastností by měla umožnit snadnou rozšiřitelnost a modifikovatelnost, což je pro systém hlasové asistentky žádoucí, jelikož vyžadovaná funkcionalita se může značně lišit podle toho, kde a kým bude používána.

S ohledem na tyto dva základní požadavky byla architektura navržena jako síť modulů, které mezi sebou komunikují pomocí zpráv. Každý modul je samostatný program zajišťující nějakou úzce specifikovanou funkcionalitu, například obsluhu emailu nebo zobrazování notifikací na počítači uživatele. Všechny moduly jsou schopny samostatného a nezávislého běhu. Speciálním modulem je centrální blok označovaný jako *jádro*, který slouží jako logické centrum celého systému.

Celý koncept je vizualizován na Obrázku 5, kde šipky mezi bloky označují předávání zpráv a čísla jejich pořadí.



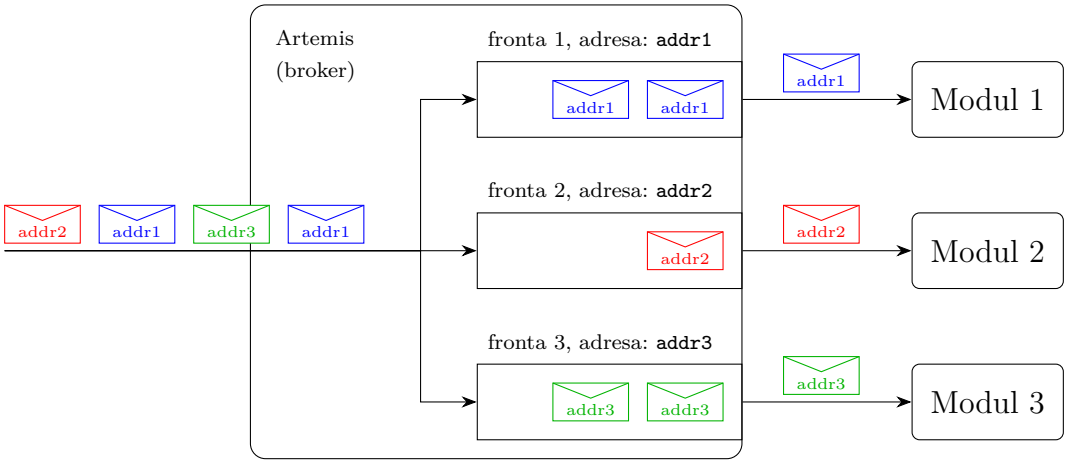
Obrázek 5: Základní architektura celého systému

Jádro přijímá zprávy od ostatních modulů, zpracovává je podle své vnitřní logiky a posílá příkazy (opět v podobě zpráv) ostatním modulům. Právě v jádře je uložena téměř všechna vyšší logika a pravidla chování celého systému, ostatní moduly by měly mít co nejjednodušší chování. Tímto způsobem je hlavní řídící logika centralizována na jednom místě, což je jednodušší pro její vývoj, modifikace a testování. Detailnější popis modulů a jejich chování je v kapitole 3.8.

3.2 Komunikace mezi moduly

Základem pro fungování celého systému je schopnost modulů předávat si mezi sebou informace v podobě zpráv. Systém osobní asistentky je řízený událostmi a proto musí být každý modul schopen vyslat i přijmout zprávu asynchronně a nezávisle na ostatních.

Pro tyto potřeby se hodí použití infrastruktury MOM (Message Oriented Middleware), zde byl zvolen Apache ActiveMQ Artemis (dále označovaný pouze Artemis). Jedná se o software poskytující message broker podporující mnoho protokolů a konfigurací, jak je popsáno v dokumentaci [11].

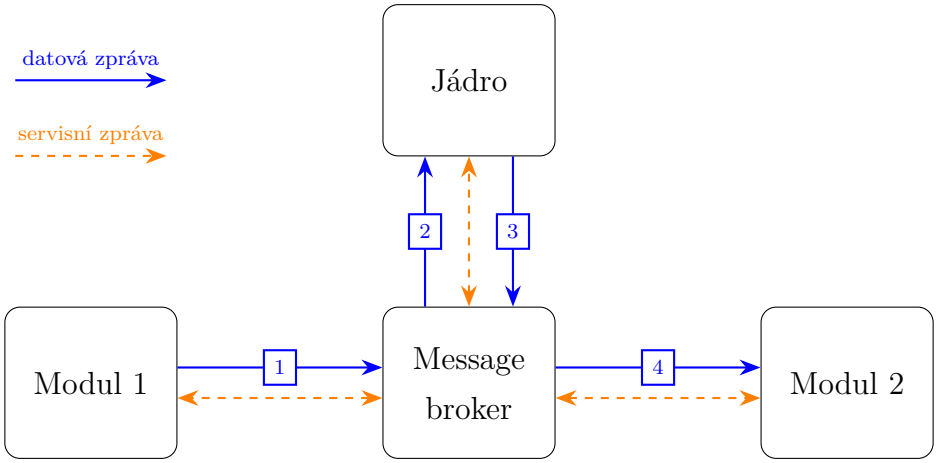


Obrázek 6: Schéma posílání zpráv přes broker Artemis

Posílání zpráv přes broker funguje tak, že každý z modulů obsahuje klienta, který se připojí k brokeru a začne sledovat svoji frontu zpráv na dané adrese (může i více zároveň). Zároveň pomocí připojeného klienta může modul odeslat novou zprávu s cílovou adresou fronty zpráv. Broker zprávu přijme a vloží ji do požadované fronty,

odkud si ji jiný klient vyzvedne. Kromě soukromých front zpráv pro každý modul byla ještě vytvořena veřejná fronta s konfigurací broadcast, kam může libovolný modul poslat zprávu a ta je rozeslána všem ostatním modulům. Tato fronta je používána pro hromadné rozesílání zpráv. Grafické znázornění právě popsaného systému je na Obrázku 6.

Ačkoli jsou moduly pomocí brokeru schopné posílat zprávy libovolnému jinému modulu (za předpokladu, že vědí, jakou frontu odebírá), tak bylo rozhodnuto, že moduly budou posílat zprávy pouze jádru a nikdy ne přímo mezi sebou. Jádro pak může posílat zprávy libovolnému modulu. Toto rozhodnutí se doplňuje s myšlenkou toho, že jádro má fungovat jako logické centrum obsahující veškerou logiku celého systému. Pokud by si moduly posílaly zprávy libovolně, jednalo by se o více decentralizované řešení, které by bylo sice více robustní, ale za cenu výrazně obtížnějšího vývoje a nižší modularity.



Obrázek 7: Schéma komunikace mezi moduly

Zprávy byly rozděleny do dvou kategorií - servisní a datové. Servisní zprávy jsou používané pro navazování a udržování spojení, předávání meta-informací a případně pro žádosti o provedení nějaké činnosti nebo vyžádání dat. Datové zprávy jsou pak používané pro reprezentaci událostí nebo objektů, se kterými moduly pracují a které si mezi sebou předávají.

Kromě významového rozdílu se oba druhy zpráv liší v tom, jak je s nimi zacházeno. Všechny zprávy následují schéma na Obrázku 7. Jsou vytvořené v nějakém z mo-

dulů, přes broker jsou odeslané do jádra, kde jsou zpracovány. Zpracování zpráv může v případě potřeby vygenerovat nové zprávy, které jsou buď odpovědí a nebo nějakým novým příkazem. Je vhodné poznamenat, že ačkoli je na Obrázku 7 znázorněn Modul 1 a Modul 2 jako dva různé bloky, může se jednat o stejný modul a nově vygenerovaná zpráva může být poslána zpět do stejného modulu, odkud původní zpráva přišla.

Jednoduchým příkladem může být situace, kdy modul hlídající emailovou schránku zaregistruje příchod nového emailu, vygeneruje datovou zprávu nesoucí informace o tom, že uživatel má nepřečtený email. V jádře je tato zpráva přijata a zpracována. Po vyhodnocení může být vygenerována nová zpráva nesoucí informaci o vizuální notifikaci, která se má uživateli zobrazit na počítači. Tato nová zpráva je odeslána do jiného modulu, který zajišťuje zobrazování notifikací na počítači. Tento modul zprávu přijme a notifikace zobrazí. Alternativně může být v jádře rozhodnuto, že je vhodnější uživatele upozornit pomocí hlasové notifikace. V takovém případě je vygenerována zpráva obsahující požadovanou promluvu a je odeslána modulu s dialogovým manažerem, který text syntetizuje a přečte nahlas uživateli.

Tento způsob předávání informací v podobě zpráv umožňuje velmi snadnou integraci nových modulů do systému a robustní řešení havárií modulů. To odpovídá výše zmíněným požadavkům na robustnost, modularitu a univerzálnost celé architektury. Odpojení nebo selhání nějakého modulu nezpůsobí kolaps celého systému, pouze ztrátu jedné dané funkčnosti, kterou tento modul obstarával. Tento způsob předávání informací také nativně řeší situace, kdy se nějaký modul připojí za běhu, protože ostatní části systému nejsou vůbec závislé na sobě navzájem, ale pracují pouze s příchozími zprávami, jejichž původ není podstatný.

3.3 Formát zpráv

Formát a struktura zpráv byla zvolena jako textová podoba ve formátu JSON. Tento formát je velmi univerzální, takže snadno umožňuje dostatečnou variabilitu k tomu, aby jej bylo možné použít obecně pro potřeby všech modulů. Zároveň je ale jednoduchý a čitelný člověkem, což je výhodné pro vývoj a testování jak celého systému, tak jednotlivých modulů.

Každá zpráva je složená z hlaviček, ke kterým jsou přiřazené nějaké hodnoty. Všechny zprávy mají čtyři povinné hlavičky, které musí být přítomné vždy u všech typů zpráv. Těmito povinnými hlavičkami jsou unikátní identifikační kód zprávy, název modulu odkud zpráva pochází, typ zprávy (datová nebo servisní) a kód zprávy, který určuje její primární účel nebo význam (například zda zpráva reprezentuje email, rozpoznanou promluvu, nebo třeba příkaz k nějaké akci).

Kromě těchto povinných hlaviček může zpráva obsahovat libovolné množství dalších hlaviček, které obsahují konkrétní informace o dané události, objektu nebo příkazu. Obsah těchto dalších volitelných hlaviček je používán při zpracování zprávy všemi moduly. U každého modulu je předem dáno, které zprávy odesílá a jakou budou mít strukturu (hlavičky a jejich významy). Stejně tak je dáno jaké zprávy modul přijímá, které hlavičky v těchto zprávách hledá a jak s nimi pracuje. Detailní popis chování jednotlivých modulů je rozebrán později.

Názvy hlaviček jsou volené tak, aby bylo možné stejný název použít pro více zpráv s různými účely. Zároveň je dbáno na to, aby hlavička alespoň částečně vypovídala o tom, co se v ní nachází. Příkladem může být hlavička `title`, která ve zprávě reprezentující email obsahuje text předmětu emailu, ale ve zprávě reprezentující grafickou notifikaci na počítači stejně pojmenovaná hlavička obsahuje titulek notifikace, která bude zobrazena. Formát JSON nicméně umožňuje pojmenovat hlavičky libovolně, je tedy technicky možné pojmenovat každou specifickou hlavičku svým unikátním názvem. V takovém případě by ale vzniklo velké množství velmi specifických názvů, které by bylo potřeba při implementaci modulů a zpracování zpráv zohledňovat. Všechny použité hlavičky, jejich názvy, datové typy a účely je možné blíže prozkoumat ve zdrojových kódech, pro ilustraci je zde uvedeno ještě několik příkladů:

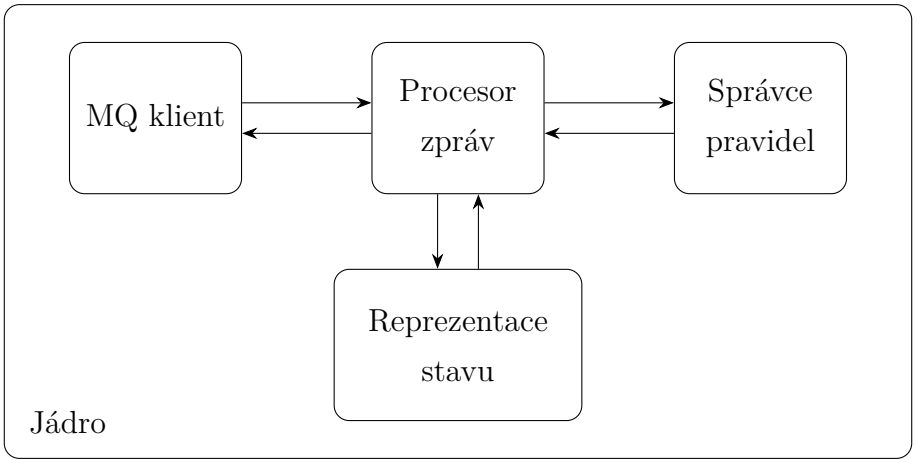
- `description` → ve zprávě oznamující ukončení modulu obsahuje důvod ukončení, ve zprávě reprezentující notifikaci obsahuje text (nikoli titulek) notifikace
- `link` → ve zprávě reprezentující email obsahuje adresu odkazující na webového klienta, ve zprávě nesoucí RSS článek obsahuje URL adresu daného článku
- `content` → ve zprávě reprezentující email obsahuje text emailu, ve zprávě oznamující ukončení modulu obsahuje identifikační kód ukončeného modulu, ve zprávě nesoucí rozpoznanou promluvu obsahuje rozpoznaný text

3.4 Jádro

Jádro je možné chápat jako speciální modul, který je zodpovědný za fungování systému jako celku. Jeho základní účel je přijímat zprávy o událostech od ostatních modulů, vyhodnocovat je a generovat odpovídající reakce (opět v podobě zpráv) a posílat je příslušným modulům. Tyto reakce jsou většinou příkazy k provedení nějaké akce, ale není to striktní pravidlo, může se jednat třeba také o žádost na zaslání nějakých dat nebo jen změnu stavu.

Aby bylo možné příchozí zprávy korektně zpracovat, obsahuje jádro také stavovou reprezentaci systému a některých vnějších veličin. Tato stavová reprezentace byla navržena tak, aby byla univerzální, modulární a konceptuálně jednoduchá. Díky těmto vlastnostem by mělo být snadné modifikovat chování jádra a zpracování různých zpráv bez nutnosti měnit stavovou reprezentaci. Detailní popis stavu, jeho struktura a vlastnosti jsou rozebrány v sekci 3.5.

Jádro bylo navržené jako skupina dílčích bloků, které spolu úzce spolupracují, jak je znázorněno na Obrázku 8. Tato struktura byla zvolena z důvodu lepší přehlednosti, ze které plyne rychlejší vývoj a jednodušší testování.



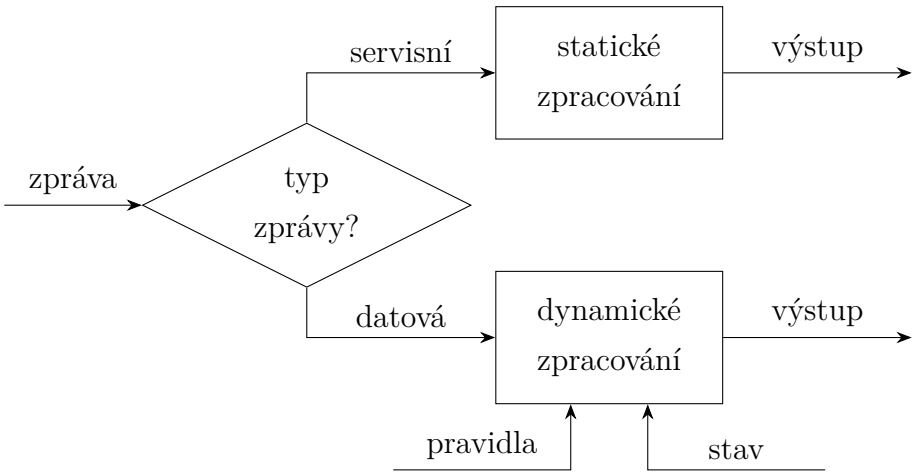
Obrázek 8: Schéma vnitřní struktury jádra

Blok označený jako „MQ klient“ je část zajišťující komunikaci s brokerem, stará se o příjem zpráv od ostatních modulů a odesílání nově vygenerovaných zpráv. Po spuštění jádra je tento klient vytvořen a ihned je proveden pokus o připojení k brokeru. Pokud je připojení úspěšné, začne klient odebírat fronty zpráv „core“ a „broad-

cast“. Pokud se připojení nepovede, je periodicky opakováno, dokud není úspěšné a nebo dokud program není ukončen. Stejně tak pokud připojení z libovolného důvodu selže za běhu, tak je periodicky opakován pokus o opětovné připojení. Toto chování zajišťuje robustnost a stabilitu, takže případné výpadky internetu nepředstavují dlouhodobý problém.

Blok „Procesor zpráv“ zajišťuje zpracování všech příchozích zpráv. Všechny zprávy přijaté MQ klientem jsou předané tomuto bloku ke zpracování. Pokud je obdržená zpráva servisní, tak je zpracována předem definovaným statickým způsobem, detailní popis bude v kapitole 3.4. V případě, že je příchozí zpráva datová, tak je zpracována pomocí pravidel, které jsou uložené uvnitř bloku „Správce pravidel“. Detailní popis pravidel a jejich použití je v kapitole 3.6.

Blok „Správce pravidel“ obsahuje seznam aktivních pravidel, které jsou používané pro zpracování datových zpráv. Jedná se o obalovací třídu, která poskytuje interface pro „Procesor zpráv“, přes který je s pravidly zacházeno. Podobně blok „Stavová reprezentace“ představuje část jádra obsahující množinu stavových proměnných, ke kterým je přistupováno přes implementované rozhraní.



Obrázek 9: Schéma zpracování zpráv v jádře

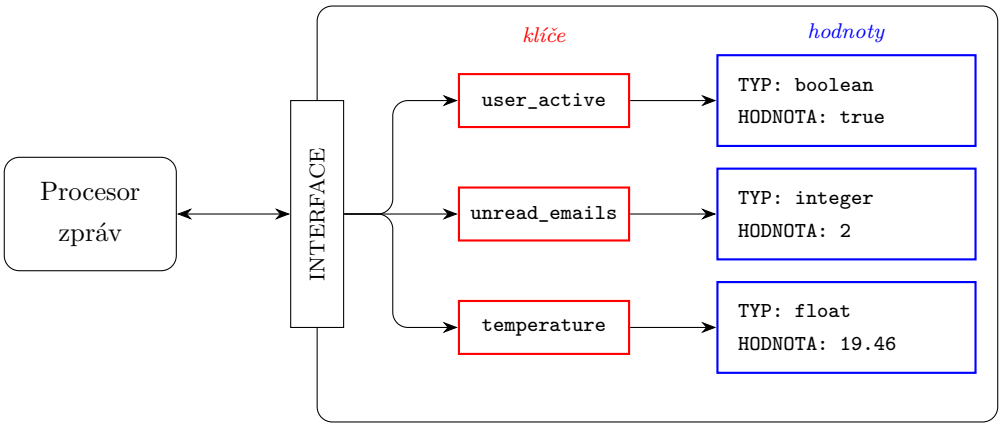
Celkově lze tedy popsat hlavní funkci jádra tak, že po inicializaci všech potřebných částí a připojení klienta k brokeru čeká jádro na příchozí zprávy. Když přijde servisní zpráva, je zpracována předem daným způsobem pouze v bloku „Procesor zpráv“. Pokud přijde datová zpráva, tak je také zpracována ve stejném bloku „Procesor zpráv“,

ale dynamickým způsobem pomocí pravidel získaných z bloku „Správce pravidel“ s využitím stavů získaných z bloku „Stavová reprezentace“. Detailní využití pravidel a zpracování zpráv je popsáno v části 3.6. Znázornění tohoto procesu je na Obrázku 9.

3.5 Reprezentace stavu

Jak již bylo řečeno v předchozí části, reprezentace stavu je potřebná pro správnou funkci jádra a tím pádem celého systému. Stav se skládá ze stavových proměnných, které jsou používané při zpracování datových zpráv. Celý stav je možné reprezentovat jako dynamicky se měnící JSON dokument, avšak skutečná implementace je jiná, detailněji rozebrána v kapitole 4.2.1.

Každá stavová proměnná je uložena ve stavu jako dvojice `klíč:hodnota`. Klíčem je zpravidla textový řetězec a hodnotou může být číslo, řetězec, boolean a nebo pole jednoho z těchto typů. Architektura stavu je ovšem navržena obecně a není vázána na tyto datové typy, v případě potřeby by bylo možné snadno implementovat libovolnou datovou strukturu jako další datový typ stavové proměnné. Pro potřeby této práce stačily pouze výše zmíněné typy. Vizualizace je na Obrázku 10.



Obrázek 10: Schéma reprezentace stavu uvnitř jádra

Stavové proměnné mohou být za běhu vytvářeny, mazány a měněny. Pro vytvoření nové stavové proměnné je potřeba definovat její název, který bude použit jako klíč při jejím hledání a počáteční hodnotu, ze které bude odvozen datový typ. Jakmile je jednou stavová proměnná vytvořena, je možné měnit pouze její hodnotu, nikoliv

její datový typ. Tato vlastnost je designovou volbou, která má předejít nepředvídatelným chováním za běhu systému a tím tak výrazně zjednodušit testování. Pro změnu stavové proměnné je potřeba zadat její klíč a novou hodnotu s odpovídajícím datovým typem. K odstranění stavové proměnné stačí znát její klíč.

Protože celá tato práce klade důraz na modulárnost, univerzálnost a snadnou modifikovatelnost, byla stavová reprezentace navržena právě takto dynamickým způsobem. Nevýhodou tohoto přístupu je vyšší komplexnost stavu a potřeba znalosti o tom, které stavové proměnné jsou uložené pod kterým klíčem, jaký mají datový typ a k čemu jsou využívány. Alternativním řešením by mohla být statická stavová reprezentace, která by obsahovala pouze předem dané stavové proměnné se stálým názvem, konstantním významem a neměnným datovým typem. Ačkoli by byl takto fixní návrh jednodušší na implementaci a orientaci, jakákoli modifikace systému nebo přidání dalších modulů a funkčností by byla výrazně komplikovanější.

Každé stavové proměnné, jejíž datový typ není pole, může být při vytvoření nastaven časovač. Pokud je stavová proměnná vytvářena s časovačem, tak při její inicializaci je potřeba uvést časový limit a výchozí hodnotu. Stavová proměnná s časovačem poté funguje tak, že při změně hodnoty začne časovač odpočítávat zadaný limit a po jeho uplynutí je hodnota automaticky nastavena na výchozí hodnotu. Pokud je hodnota obnovena (na libovolnou hodnotu, i stejnou jako je v daný moment), tak je časovač resetován. Tato vlastnost byla přidána do návrhu pro snadnou obsluhu takových proměnných, které mají časově omezenou platnost a je potřeba je periodicky obnovovat.

3.6 Pravidla a zpracování datových zpráv

Datové zprávy jsou zpracovávané v jádře pomocí pravidel. Základní koncept tohoto zpracování spočívá v tom, že na příchozí datovou zprávu jsou postupně aplikovaná jednotlivá pravidla. Aplikace pravidla může způsobit změnu stavových proměnných nebo vygenerování libovolného počtu nových zpráv či pravidel. Právě pomocí pravidel je tedy možné nastavit chování celého systému.

Pravidla jsou aplikována na vstupní datovou zprávu pokaždé ve stejném pořadí a všechny akce jsou prováděné okamžitě. Nově vygenerované zprávy jsou odeslané ihned poté, co je aplikace pravidla dokončena (ještě před tím, než je proveden pokus

aplikaci následujícího pravidla). Tímto postupem jsou na vstupní zprávu postupně aplikována všechna pravidla, dokud nenastane jedna z následujících dvou situací:

- **Všechna pravidla byla vyzkoušena**
  V tomto případě již není žádné pravidlo, které by bylo platné a zároveň nebylo vyzkoušeno. Vstupní datová zpráva byla kompletně zpracována a již není potřebná, může být tedy odstraněna z paměti.

- **Nějaké z pravidel terminovalo**
  Při navrhování pravidel bylo rozhodnuto, že by bylo výhodné, kdyby mohla být aplikace pravidel ukončena dříve, než jsou vyzkoušena všechna pravidla. Z toho důvodu byla pravidlům přidána možnost „terminace“. Jedná se o označení situace, kdy pravidlo po svém dokončení žádá, aby bylo zpracování aktuální datové zprávy zastaveno a žádná další pravidla již nebyla ani zkoušena. Když pravidlo terminuje, zpracování vstupní datové zprávy je ukončeno a zpráva je odstraněna z paměti.

Každé pravidlo je složené z podmínek a akcí, jeho vstupem je datová zpráva a stavová reprezentace, výstupem je pak libovolný počet nových zpráv. Nejjednodušší pravidlo může být tedy složené z jedné podmínky, při jejímž splnění bude provedena jedna akce. Realizace složitějších chování pomocí takto jednoduchých pravidel by ovšem vyžadovala jejich velké množství, což by komplikovalo vývoj a testování celého systému. Z tohoto důvodu byl koncept pravidel navržen velmi volně a pravidlo je možné složit jako libovolnou kombinaci podmínek a akcí. Pravidla lze připodobnit psaní `if` struktur (a jejich variací) v běžných programovacích jazycích.

V podmínkových částech se může vyskytovat libovolná kombinace logických a matematických operátorů. Tyto operátory mohou pracovat s proměnnými, které je možné získat buďto ze vstupní zprávy a nebo ze stavu systému. Výhodou je zde podobnost struktury pravidel a stavové reprezentace, protože díky tomu lze v pravidlech snadno kombinovat oba zdroje. Pravidlo tedy může pracovat se všemi hlavičkami příchozí zprávy (pouze číst) a se všemi proměnnými stavové reprezentace (čtení i zápis).

Výstupem pravidla (to jsou akce, které jsou provedené při splnění podmínek) pak může být libovolná kombinace následujících věcí:

- změna existující stavové proměnné

- vytvoření nové stavové proměnné
- odstranění existující stavové proměnné
- vytvoření nové zprávy a její odeslání
- vytvoření nového pravidla

Všechny tyto body mohou být v každém pravidle v libovolných počtech, takže jedno pravidlo může vygenerovat více zpráv nebo třeba více nových pravidel.

Příkladem použití pravidel může být následující situace: Do jádra přijde nová datová zpráva reprezentující upozornění, že uživatel má nějaké nepřečtené emaily. Tato zpráva je předána ke zpracování pomocí pravidel. Každé pravidlo postupně kontroluje své podmínky a porovnává je se zprávou a stavem, zda jsou splněné. Jedno z těchto pravidel pak bude obsahovat podmínku, že pokud:

- má zpráva kód, který reprezentuje upozornění na nepřečtené emaily
- a zároveň denní doba je v zadaném intervalu 9:00 - 18:00
- a zároveň stav reprezentující aktivitu uživatele u počítače je `TRUE`,

tak má být vygenerována zpráva reprezentující hlasové upozornění, odeslána do hlasového modulu a přečtena uživateli.

Pravidla je možné rozdělit do dvou základních typů podle jejich doby platnosti - permanentní a dočasná. Permanentní pravidla jsou platná po celou dobu běhu systému od okamžiku jejich vytvoření. Dočasná pravidla mohou za běhu systému vznikat i zanikat. Při vytváření dočasného pravidla je potřeba definovat, čím bude určena jeho platnost. Platnost pravidla může být dána libovolnou kombinací (v logickém součinu) těchto elementárních vazeb:

1. **Vazba na časový limit**
   Pravidlu je při vytvoření stanovena doba, po kterou je platné. Po uplynutí tohoto času pravidlo přestává být platné.
2. **Vazba na booleovskou stavovou proměnnou**
   Pravidlu je přidělena stavová proměnná s datovým typem `boolean`, jejíž logická hodnota určuje, zda je pravidlo platné (hodnota `TRUE`) nebo nikoli (hod-

nota `FALSE`). Pokud z nějakého důvodu zadaná stavová proměnná přestane existovat, je interpretována pravidlem jako `FALSE`.

3. **Vazba na vnitřní počítadlo aktivací**
   Pravidlu je při vytvoření přiřazeno počítadlo s nějakou kladnou počáteční hodnotou. Uvnitř pravidla je pak možné (v libovolném místě jeho vykonávání) hodnotu počítadla upravit, typicky po dokončení nějaké akce je počítadlo dekrementováno o jedničku dolů. Pravidlo je platné, dokud je hodnota počítadla větší než nula.

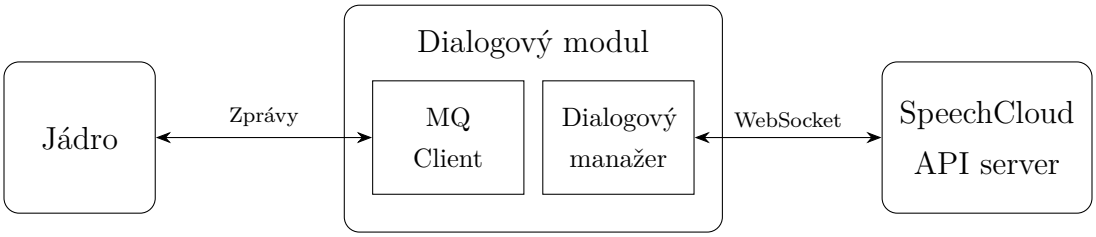
Platnost pravidel je kontrolována vždy těsně před tím, než jsou aplikována na vstupní datovou zprávu a všechna pravidla, která jsou v okamžik kontroly shledána neplatnými, jsou odstraněna.

Možnost dynamicky přidávat a odstraňovat pravidla byla další designovou volbou s cílem udělat systém pravidel více variabilní a univerzální. Pokud by nebylo možné pravidla za běhu systému přidávat a ubírat, bylo by nutné předem zmapovat všechny možné situace pro dané použití systému a pravidla na ně sepsat dopředu. To by vyžadovalo výrazně větší množství stavů a zprávy s více hlavičkami, aby bylo možné rozlišit, která pravidla mají být zrovna aplikována. Další výhodou popsaného dynamického návrhu je možnost jej jednoduše nevyužít, pokud to není potřeba.

3.7 Dialogový modul

Účelem dialogového modulu je zajistit řešení dílčích úloh spojených s hlasovými dialogovými systémy, konkrétně pak rozpoznání, porozumění a syntézu řeči. Tyto úkony jsou poskytované platformou SpeechCloud, která byla popsána v kapitole 2.6.

Aby mohl modul využívat služby, které SpeechCloud poskytuje, je jeho součástí dialogový manažer. Kromě komunikace se SpeechCloud API serverem je samozřejmě potřeba, aby byl modul schopný komunikovat s jádrem, k čemuž slouží vnitřní message-queue (MQ) klient. Protože jsou MQ klient i dialogový manažer součástí stejného programu, je komunikace mezi těmito částmi velmi snadná a je realizována přímo v kódu. Schéma této části systému je zobrazeno na Obrázku 11.



Obrázek 11: Schéma dialogového modulu

Chování dialogového manažeru bylo navrženo a implementováno tak, aby bylo v souladu s celkovým konceptem modulárního systému osobní hlasové asistentky. V kapitole 3 bylo řečeno, že moduly by měly mít pokud možno jednoduché chování a většina složitého řízení by měla být koncentrována v jádře. Z tohoto důvodu je chování vnitřního dialogového manažeru poměrně jednoduché a dalo by se popsat tak, že modul zprostředkovává komunikaci mezi jádrem a SpeechCloud API serverem. Jádro může poslat zprávu s příkazem a nebo může modul poslat zprávu jádru s informací o nějaké proběhlé události. Aby byl modul soběstačný, tak disponuje základním rozhodovacím algoritmem, který obsluhuje okrajové situace (třeba ztrátu spojení), inicializaci a ukončení.

Po spuštění je uvnitř dialogového modulu vytvořen MQ klient, který se připojí k brokeru na zadané adrese a čeká na příchozí zprávy, nebo příkazy na odeslání nové zprávy. Zároveň je vytvořen a spuštěn dialogový manažer, který se připojí k API serveru a po inicializaci začne poslouchat na dané URL adrese, kde očekává interakci s klientem. V moment, kdy uživatel otevře prohlížeč a načte požadovanou stránku, tak je inicializace dokončena a modul poté pošle zprávu jádru s oznámením, že je připraven k provozu. Od chvíle, kdy je modul připravený, tak vyčkává na zprávy od jádra a nebo na události z API serveru.

Jediný typ datové zprávy, kterou modul dokáže zpracovat, je zpráva nesoucí příkaz k syntéze textu. Když modul takovouto zprávu přijme, tak z ní extrahuje požadovaný text a přes SpeechCloud API server jej nechá syntetizovat a následně přečíst. Po dokončení syntézy a čtení je jádru o této události posláno oznámení.

Kromě servisních zpráv pro zajištění komunikace může modul také obdržet servisní zprávu, která neponese přímo žádná data, ale bude obsahovat příkaz ke změně stavu modulu. Konkrétně se pak jedná o příkazy pro spuštění a zastavení rozpoznávání

řeči (při kterém zároveň běží odposlech na mikrofonu) a porozumění řeči. Servisní zprávy nesou tyto příkazy z toho důvodu, že není potřeba žádného přenosu dat.

Když modul dostane zprávu s příkazem pro zahájení rozpoznávání řeči, tak zavolá příslušnou metodu, která zapne odposlech mikrofonu na uživatelském zařízení. Současně zobrazí informaci pro uživatele na webovém rozhraní, že systém právě poslouchá. Když pak uživatel začne mluvit, tak je tento signál zachycen a zpracován. V moment, kdy uživatel přestane hovořit na dostatečně dlouhou dobu, tak je audiosignál zakončen a je na něj aplikováno rozpoznání a porozumění řeči. Výsledek tohoto porozumění je poté zachycen manažerem jako signál z API serveru, zabalen do zprávy a poslán jádru pro zpracování.

3.7.1 Použití gramatik

Po připojení webového klienta je součástí inicializace kompilace gramatik, které mají být použité pro porozumění řeči. Tyto gramatiky jsou textové soubory ve formátu Augmented Backus-Naur Form (ABNF) [12], které splňují standard Speech Recognition Grammar Specification zmíněný v kapitole 2.6. Psaní těchto gramatik prošlo v průběhu zpracovávání této práce dvěma iteracemi.

První verze byla inspirována ukázkovými příklady, které byly poskytnuty vedoucím práce. Tento styl použití spočíval v tom, že bylo předem zmapováno vše, co bude systém schopen dělat a podle toho byly vytvořené odhady všech příkazů a interakcí, které by mohl uživatel se systémem mít. Následně bylo pro každou takovou interakci vytvořeno co největší množství formulací, které jsou v přirozeném jazyce běžně používané. Poté byly tyto formulace (ručně) zapsané do podoby ABNF gramatiky s vhodným využitím syntaktických nástrojů pro optimalizaci zápisu.

Pro ilustraci následuje ukázka části původní gramatiky, kde jsou zapsány fráze, které reprezentují:

- žádost o to, aby bylo něco označeno jako přečtené (mail, zpráva, ...)
- oznámení, že uživatel bere informaci na vědomí
- oznámení, že uživatel si přeje nadiktovat odpověď

```
public $response = (
    ((označit|označ)[to|ho] jako
    (přečtené|přečtený|vyřízený|vyřízené)) {mark_as_read} |

    (ok|jasně|jasný|dobře|rozumím|jo|dobrý) {ok} |

    ([chci]odpovědět|odpověz|
    (nadiktovat|nadiktuji|nadiktuju|řeknu) odpověď) {respond} |
)
```

Tímto zápisem je možné zachytit například promluvy „Označ to jako přečtený.“, „Označit jako vyřízené.“, „Chci odpovědět.“, „Nadiktuji odpověď.“. Takto vypadající gramatika byla následně použita pro porozumění řeči. Když byla detekována nějaká z definovaných frází, tak byla jádru tato informace zaslána datová ve formě zprávy, kde jednou z položek byl kód rozpoznané promluvy, například `mark_as_read`. Jádro poté tuto zprávu zpracovalo příslušným způsobem.

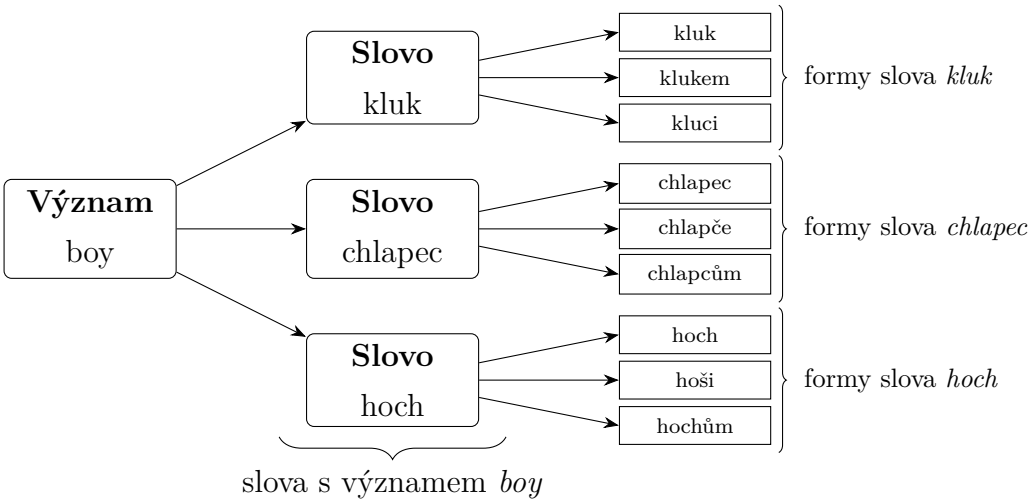
Při testování se ale rychle ukázalo, že fráze, které uživatelé běžně říkají, jsou velmi rozmanité a bylo potřeba gramatiku neustále rozšiřovat, protože se vždy našla nějaká formulace, kterou gramatika nebyla schopna popsat. Alternativní možností by bylo poučení uživatelů o tom, jaké promluvy systém dokáže zpracovat. Ani jedna z těchto možností ale není příliš vhodná a proto byl způsob práce s gramatikami modifikován.

Druhá verze gramatik přistupuje k problematice trochu jiným způsobem a byla inspirována tím, jak uvažuje člověk při porozumění textu. Namísto toho, aby systém hledal přesně definované fráze, tak se zaměřuje na vyhledávání obecných významů nebo konceptů. Výstupem je pak posloupnost těchto významů, ze kterých je již poměrně snadné odhadnout, jaký byl cíl promluvy. Například když je detekována posloupnost významů `chtít - značit - přečtené`, tak by mělo být zřejmé, že uživatel chce označit daný objekt jako přečtený. Výhodou tohoto přístupu je, že v promluvě se mohou vyskytovat výplňová slova a nebo třeba jiné (i gramaticky nesprávné) tvary slov a systém stejně dospěje ke stejnému výsledku.

K vytvoření tohoto systému ovšem bylo potřeba vyřešit tyto problémy:

- Jak rozpoznat v promluvě jednotlivé významy?
- Jak přistupovat k synonymům?
- Jak řešit to, že jedno slovo může mít mnoho tvarů?

Řešení bylo navrženo tak, že se předpokládá, že *význam* je složen ze *slov* (synonym) a každé slovo je složeno ze svých různých *forem* (tvarů). Pod pojmem „složen“ je zde myšleno to, že může být reprezentováno jednou z podřazených částí, nejedná se přímo o spojování dohromady. Pro lepší rozlišení významů a samotných slov jsou významy pojmenované anglicky. Kromě obecných významů a konceptů také gramatika obsahuje základní reprezentace čísel a některá česká jména. Tento koncept je ilustrován na Obrázku 12.



Obrázek 12: Ilustrace druhého použitého konceptu gramatik

Část takto strukturované gramatiky vypadá poté následně:

```
public $response = ($MEANINGS);
$MEANINGS = ( $MEANING_good | $MEANING_author | $MEANING_all | ...);
$MEANING_good = ( (
    $WORD_dobrý |
    $WORD_fajn |
    $WORD_výborný |
```

```
    $WORD_pěkně | ... ) {MEANING_good} );
$MEANING_author = ( (
    $WORD_autor |
    $WORD_tvůrce |
    $WORD_původce | ...) {MEANING_author} );
...
$WORD_autor = ( ( autorům | autorem | autor | autoru |
    autore | autorech | autory | autora |
    autorů | autorovi | autoři ) {WORD_autor} );
$WORD_dobrý = ( ( dobrého | nejlepší | dobrým | dobrém |
    dobrou | dobrých | dobré | dobrému | lepší |
    dobří | dobrý | dobrými | dobrá ) {WORD_dobrý} );
...
```

Gramatika s právě popsanou strukturou byla použita pro porozumění řeči. Výsledek porozumění řeči je zabalen do datové zprávy a odeslán jádru ke zpracování. V této zprávě je kromě posloupnosti detekovaných významů uložena také informace o tom, podle kterého slova byly jednotlivé významy odvozené a zároveň zpráva obsahuje plný rozpoznaný text. Tyto dodatečné informace může jádro využít při zpracování zprávy pro nějaké specifičtější situace.

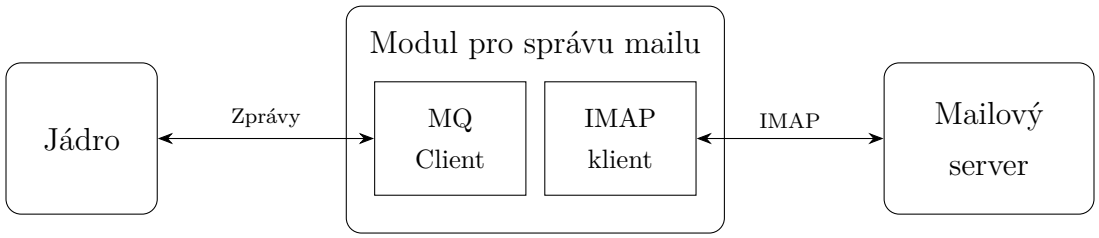
3.8 Ostatní moduly

V následujících podkapitolách jsou stručně popsány architektury a principy všech ostatních modulů, které byly během řešení této práce vytvořeny. Technicky jsou si všechny moduly rovné, ale vzhledem k tomu, že je tato práce zaměřena více na část hlasových dialogových systémů, tak byl dialogový modul rozepsán detailněji.

3.8.1 Modul pro správu mailové schránky

Účelem tohoto modulu je hlídat přidělenou emailovou schránku a zaslat upozornění jádru v případě, že je detekován nový nepřečtený email. K této činnosti modul využívá Internet Message Access Protocol (IMAP) [13]. Kromě hlídání příchodu nových zpráv dokáže modul pomocí IMAP protokolu také provádět základní operace nad uloženými zprávami, jako je vyhledávání, mazání nebo označování mailů jako

přečtených. Kromě IMAP klienta musí samozřejmě modul obsahovat také MQ klienta, aby byl schopný komunikovat s jádrem. Vnitřní schéma tohoto modulu je na Obrázku 13.



Obrázek 13: Schéma modulu pro správu mailové schránky

Stejně jako všechny ostatní moduly, samotné chování je navrženo tak, aby bylo jednoduché a především poskytovalo potřebné funkce pro jádro. Tento modul se po spuštění a inicializaci připojí na zadanou emailovou schránku a zkontroluje, zda obsahuje nějaké nepřečtené zprávy. Pokud ano, tak je tato informace zabalena do datové zprávy a poslána jádru ke zpracování. Následně je připojený IMAP klient uveden do „idle“ režimu, kdy pouze hlídá změny v dané schránce. Pokud je detekován příchod nového emailu, tak je opět počet nepřečtených emailů zabalen do datové zprávy, odeslán jádru a klient je opět uveden do idle režimu. Tento cyklus se opakuje po celou dobu běhu modulu.

Vedle sledování nepřečtených zpráv je modul schopný přijmout a zpracovat zprávy, které reprezentují následující příkazy:

1. **Vynucená kontrola schránky**
   Pokud přijde zpráva s kódem `force_check`, tak modul zkontroluje počet nepřečtených zpráv v připojené emailové schránce a pošle zpět odpověď ve formě datové zprávy. Na rozdíl od samovolné kontroly je zpět odeslána odpověď i ve chvíli, kdy ve schránce není detekován žádný nepřečtený mail.

2. **Vyžádání emailů**
   Modul dokáže také reagovat na zprávy, které reprezentují žádost o zaslání nějakých vybraných mailů. Aby mohl modul maily ze serveru získat, zabalit do datové zprávy a odeslat zpět jádru, musí být v příchozí zprávě uložena informace o tom, které maily má hledat. Tato informace může mít podobu

vyhledávacích parametrů, které modul použije jako filtr při vyhledávání požadovaných zpráv. Podporované parametry jsou přečtenost mailu, odesílatel, počet a řazení. Modul tedy dokáže například najít a odeslat `n` nejstarších nepřečtených emailů od vybraného uživatele.

3. **Označení emailů jako přečtené**
   Jednou z nejčastějších akcí, kterou uživatel dělá při práci s mailovou schránkou, je označování mailů jako přečtené. Toho je schopný i tento modul, kdy podobně jako v předchozím bodě, potřebuje informaci o tom, které maily označit jako přečtené. Je možné použít stejné parametry jako v předchozím bodě.

3.8.2 Modul pro vytváření nových mailů

Jak již název napovídá, toto je jednoduchý modul, který zajišťuje vytvoření a odeslání nového mailu. Základní funkcionalita tohoto modulu je taková, že po nastartování pouze čeká na příchozí zprávy z jádra, které nesou příkaz o tom, aby byl vytvořen a odeslán nový mail. K této činnosti využívá modul SMTP protokol [14].

Tyto zprávy musí pochopitelně obsahovat všechny informace potřebné k tomu, aby mohl být nový email vytvořen a odeslán. Těmito informace jsou adresa příjemce a adresa odesílatele. Volitelný obsah je pak text předmětu a samotný obsah emailu, alespoň jedna z těchto částí ale musí být vyplněna.

3.8.3 Modul pro sledování RSS

Účelem tohoto emailu je sledovat přidělené RSS zdroje a v případě, že je detekována publikace nového článku, tak upozornit jádro.

Chování tohoto modulu pak spočívá v tom, že po spuštění modul načte soubor se sledovanými adresami a u každé zkontroluje čas poslední aktualizace. Tento čas porovná se svou uloženou hodnotou. Pokud je detekován článek, který ještě nebyl zaznamenán, tak je stažen, rozparsován a zabalen do zprávy, která je odeslána jádru. Tato zpráva obsahuje titulek článku, jeho shrnutí a odkaz na web, kde byl publikován. Tímto způsobem jsou postupně zkontrolovány všechny adresy a kontrola je periodicky opakována každou hodinu.

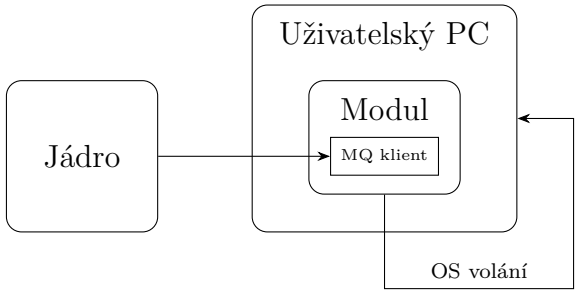
Zmíněný soubor se sledovanými adresami je textový soubor uložený ve stejném adresáři, odkud byl modul spouštěn. Na každém řádku tohoto souboru je vždy URL adresa jednoho RSS feedu následovaná středníkem, za kterým je uloženo číslo reprezentující poslední aktualizaci. Protože je před každou kontrolou znovu soubor otevřen a načten, tak je možné přidávat a ubírat sledované adresy za běhu modulu (za středník je možné vložit jednoduše nulu, nebo opsat nějakou předchozí hodnotu).

3.8.4 Modul pro správu grafických notifikací

Při běžném provozu osobní asistentky může nastat situace, kdy je preferované, aby byla informace uživateli předána pouze vizuální formou. Jedním z takovýchto případů může být třeba situace, kdy uživatel aktivně používá počítač a upozornění na nově příchozí email by tedy mohl spíše preferovat jako vizuální vyskakovací notifikaci než jako hlasovou promluvu.

Správu vizuálních notifikací na počítači uživatele právě zajišťuje tento modul. Jeho chování spočívá v tom, že po spuštění pouze čeká na příchozí zprávy od jádra, ve kterých jsou informace o tom, jakou notifikaci má zobrazit. V moment, kdy nová zpráva přijde, tak je vytvořena požadovaná notifikace, která je následně zobrazena formou vyskakovacího okna.

Kromě titulku a popisku dokáže modul také k notifikaci přidat rovnou tlačítka, na která může uživatel kliknout a tím tak zvolit nějakou akci. Informace o tom, co má zpráva obsahovat za tlačítka, musí být obsažena v příchozí zprávě. Po stisku nějakého tlačítka modul obdrží předem známý kód, který opět zabalí do nové zprávy a odešle jádru jako informaci o tom, že uživatel stiskl dané tlačítko na dané notifikaci.



Obrázek 14: Schéma běhu modulu zajišťujícího desktopové notifikace

Zobrazování notifikací je závislé na operačním systému a případně použitém správci notifikací Z pochopitelných důvodů musí tento modul běžet přímo na uživatelském počítači. Obecně je ale koncept i chování univerzální a přenositelné na libovolné zařízení s grafickým výstupem a dostatečným výpočetním výkonem. Vizualizace je na Obrázku 14.

3.8.5 Modul pro správu uživatelského počítače

Hlavním účelem tohoto modulu je sledovat stav uživatelského počítače a zajišťovat některé operace na tomto počítači.

Sledovaným stavem je doba od poslední interakce uživatele s myší nebo klávesnicí. Modul tedy sleduje dobu, po kterou uživatel na počítači neprovedl žádnou akci. Pokud tato doba přesáhne definovaný limit, tak je odeslána zpráva jádru, že uživatel již není aktivní po delší dobu. Když pak uživatel následně opět provede nějakou akci, tak je odeslána jádru zpráva, že uživatel začal být opět aktivní. Tato informace je užitečná zejména pro vyhodnocování toho, jaký typ notifikace (vizuální nebo hlasovou) bude uživatel preferovat.

Modul je také schopný přijímat zprávy od jádra a vykonávat podle nich vybrané základní operace na uživatelském počítači. Těmito akcemi jsou:

1. **Otevřít prohlížeč**
   Pokud přijde zpráva reprezentující příkaz k otevření prohlížeče, tak se modul pokusí spustit prohlížeč, obdobně jako kdyby uživatel poklikal myší na ikonu. V případě, že zpráva zároveň obsahuje URL adresu v příslušné hlavičce, tak je prohlížeč otevřen rovnou na této adrese, jinak je otevřen na domovské obrazovce. Jedná se o velmi univerzální příkaz, protože mnoho služeb je dnes poskytováno jako webová aplikace přístupná přes prohlížeč.
2. **Otevřít průzkumník souborů**
   Druhým příkazem, který modul dokáže vykonat, je otevření průzkumníka souborů. Opět je možné specifikovat přímo cestu, kterou má průzkumník otevřít, v případě absence této informace je otevřena výchozí lokace.

Výše vypsané příkazy jsou pouze ilustrativní pro potřeby této práce. Modul by mohl obdobným způsobem provádět libovolné akce, které mu budou implementovány, což

jej činí jedním z potenciálně nejmocnějších modulů. Typickým příkladem další funkcionality by mohlo být spouštění libovolných skriptů, přehrání nějakého multimediálního souboru nebo dokonce vypnutí samotného zařízení či odhlášení uživatele. Schéma tohoto modulu je možné vizualizovat obdobně, jako je na na Obrázku 14.

Z popisu by mělo být zřejmé, že tento modul je opět závislý na operačním systému a softwaru, který uživatel na svém počítači má. A z pochopitelných důvodů musí tento modul také běžet přímo na uživatelském zařízení. Konceptuálně je ale chování modulu univerzální a přenositelné, pouze konkrétní implementace jednotlivých příkazů by bylo nutné změnit.

3.8.6 Modul pro ukládání zpráv

Tento modul slouží jako odkládací prostor na zprávy, který je využíván jádrem. Koncept tohoto modulu vychází z toho, že při běhu jádra bylo potřeba poměrně často odložit příchozí zprávu někam stranou a vrátit se k ní až později. Typicky se jedná o situace, kde pro dokončení zpracování zprávy je potřeba počkat na nějakou další zprávu a nebo na odpověď od uživatele. Kromě toho se také jedná o situace, kdy jádro potřebuje, aby mu byla zaslána zpráva jako „připomenutí“ za nějakou dobu, nebo až na žádost při nějaké jiné události.

Příkladem může být příchod datové zprávy obsahující email, ale pro zpracování jádro vyžaduje nějakou odpověď od uživatele. Je tedy nutné tuto zprávu někam odložit, vyčkat na odpověď a poté dokončit její zpracování. Jiným příkladem může být situace, kdy přijde zpráva vyžadující zobrazení nějaké notifikace, ale denní doba není zrovna vhodná (třeba pozdě v noci) a tak je potřeba zprávu odložit na druhý den ráno.

K těmto účelům by bylo samozřejmě možné zprávy ukládat přímo v jádře, ovšem tím by se komplexnost již tak poměrně složitého programu ještě navýšila a navíc by tento odkládací prostor nebyl přístupný pro žádný jiný modul. Proto byl vytvořen samostatný modul, který tyto funkcionality zajišťuje externě.

Chování samotného modulu je opět poměrně jednoduché. V základu čeká po spuštění na příchozí zprávy, které musí obsahovat následující informace:

- Samotnou zprávu (což je text v JSON formátu), která se má uložit. Je vhodné zmínit, že příchozí zpráva není přímo tou zprávou, která je ukládána. Příchozí zpráva obaluje zprávu, která bude uložena a dodává informace o tom, jak má být uložena.
- Dobu, po kterou má být zpráva uložena. Každá zpráva při uložení dostane expirační čas, po jehož vypršení je provedena nějaká přidružená akce.
- Kód akce, která má být provedena, když zpráva expiruje. Může se jednat o odstranění zprávy z úložiště, odeslání zprávy jádru a její následné odstranění, nebo může být zpráva odeslána a časovač resetován.

Kromě výše vypsaných akcí je také modul schopen provést nějaké akce na vyžádání. Když přijde zpráva reprezentující takovýto požadavek, tak modul dokáže najít, poslat a případně odstranit vybrané zprávy. Informace o tom, nad kterými zprávami má být provedena daná akce musí být obsažena v příchozí zprávě. Lze vybírat uložené zprávy jednotlivě podle jejich ID, nebo obecně podle stáří (vzít nejstarší nebo nejnovější). Dále je možné provést akci nad zprávami se specifikovaným kódem a opět modul dokáže zpracovat i informaci o tom, zda má vzít nejstarší nebo nejnovější zprávu s daným kódem, nebo případně všechny.

Jádro tedy může tomuto modulu posílat příkazy, které by se daly do přirozeného jazyka přepsat asi následovně (pouze ukázka nějakých možností):

- Pošli mi všechny zprávy, které mají kód `UNREAD_EMAILS`.
- Odstraň nejstarší uloženou zprávu.
- Ulož tuhle zprávu na deset minut a poté mi ji pošli zpět.

4 Realizace a testování

Tato kapitola je věnována popisu samotné realizace a s ní spojeného testování. Budou zde popsány postupy, jakými byly jednotlivé části implementované, jaké problémy se při realizaci vyskytly a jak byly vyřešené.

Zdrojové kódy jsou dostupné na `https://github.com/TomLebeda/bp2022`.

4.1 Volba brokeru

Jako první bylo potřeba zvolit nějaký message broker, který by zajišťoval přenos zpráv mezi všemi moduly. V úvahu přicházely RabbitMQ[1], Apache Kafka[2], Apache ActiveMQ Artemis [11] a Eclipse Mosquitto[3]. Nakonec byl zvolen broker Apache ActiveMQ Artemis, který disponuje všemi potřebnými funkcemi pro potřeby této práce, ale zároveň jeho použití není zbytečně složité.

Jak je možné se dočíst v oficiální dokumentaci [11], tak broker lze nakonfigurovat mnoha různými způsoby podle toho, co úloha vyžaduje. Dále broker podporuje množství různých protokolů a možností zabezpečení. Pro účely této práce bylo ponecháno výchozí nastavení a zabezpečení je řešeno formou přihlašovacích údajů pro jednotlivé moduly.

4.2 Jádro

Jádro bylo implementováno v jazyce Java, důvodem pro volbu tohoto jazyka byla jeho rozšířenost, výkonnost, přenositelnost a dobrá integrace s ostatními technologiemi, zejména pak brokerem.

Jednotlivé vnitřní části jádra popsané v kapitole 8 a zobrazené na Obrázku 8 jsou implementované jako samostatné třídy, které si mezi sebou udržují potřebné reference, aby si mohly předávat jiné objekty a volat vzájemně metody. Kromě těchto tříd je implementováno ještě několik pomocných tříd a také třídy reprezentující pravidla, zprávy a stavové proměnné.

[1] Oficiální stránky RabbitMQ: https://www.rabbitmq.com
[2] Oficiální stránky Apache Kafka: https://kafka.apache.org/
[3] Oficiální stránky Eclipse Mosquitto: https://mosquitto.org/

Samotný běh celého jádra začíná vytvořením instancí základních objektů, jejich inicializací a následným provázáním pomocí referencí. Při inicializaci MQ klienta je vytvořeno spojení na broker, kde je do fronty `broadcast` periodicky (s minutovým intervalem) posílána zpráva žádající o ohlášení všech dostupných modulů. Toto periodické volání probíhá ve vlastním vlákně a je nezávislé na běhu zbytku jádra. Zároveň je na adrese `core` vytvořen posluchač, který čeká na příchozí zprávy. Veškeré příchozí zprávy jsou validovány, převedeny na vnitřní reprezentaci zprávy a poslány na zpracování do třídy `MessageProcessor`. Všechny právě popsané úkony jsou implementovány tak, aby při ztrátě spojení nehavaroval celý systém, ale pouze počkal na obnovení připojení a opět se napojil na potřebné kanály.

4.2.1 Reprezentace stavu

Celková stavová reprezentace je množina stavových proměnných, které jsou uložené v datové struktuře `HashMap`. Díky tomu je možné dynamicky přidávat a ubírat nové stavové proměnné za běhu programu a pracovat i s větším množstvím stavových proměnných bez výpočetních obtíží. Z této implementace také plyne, že pro přístup k nějaké stavové veličině je potřeba znát její identifikační klíč (který je v této práci shodný s jejím názvem).

Pro lepší bezpečnost ale není ke stavovým proměnným možné přistupováno přímo, nýbrž přes definovaný interface `CoreStateMap`. Toto rozhraní poskytuje základní funkce pro získání hodnoty vybrané stavové proměnné a nebo její změnu či vytvoření. Rozdílem oproti přímému přístupu je hlavně kontrola datových typů, názvů a kolizí, skryté vytvoření instance a její inicializace. Zároveň je díky tomu možné vyměnit v případě potřeby implementaci na pozadí bez nutnosti přepisovat zbytek kódu, který se stavy pracuje.

Jednotlivé stavové proměnné jsou instance třídy `StateVariable`. Tato třída využívá generických datových typů a tak je možné pomocí ní reprezentovat libovolnou stavovou veličinu, bez ohledu na její datový typ. Pokud by to úloha vyžadovala, tak by bylo možné použít prakticky libovolnou datovou strukturu nebo třídu jako datový typ pro stavovou veličinu.

Obdobný přístup je použit pro „stav“ příchozích zpráv. Zpráva nemá přímo stav ve stejném smyslu jako je stavová reprezentace uložená v jádru, nicméně s jejím obsa-

hem lze pracovat obdobným způsobem. Aby byla tvorba pravidel co nejpohodlnější, je možné k jednotlivým hlavičkám přistupovat z vnitřku pravidla pomocí rozhraní `MessageStateMap`. Rozdíl mezi tímto rozhraním a `CoreStateMap` spočívá primárně v tom, že `MessageStateMap` poskytuje pouze metody pro čtení jednotlivých hodnot na pozadí.

4.2.2 Zpracování zpráv

Zpracování servisních zpráv je pro všechny příchozí servisní zprávy stejné a je závislé především na jejich kódu. Na zprávy reprezentující ohlášení nějakého modulu je odpovězeno v podobě potvrzovací servisní zprávy. Pokud má modul na starosti údržbu nějakého stavu (například modul pro správu uživatelského počítače udává stav aktivity uživatele), tak je tato informace také obsažena v ohlašovací zprávě a je podle ní vytvořena nová stavová proměnná. Zpracování zprávy oznamující smrt nějakého modulu pak tyto stavy zase odstraňuje.

Princip zpracování datových zpráv je popsán v kapitole 3.6. Samotná implementace tohoto dynamického zpracování je poměrně jednoduchá. Třída `MessageProcessor` získá pole pravidel a na vstupní zprávu postupně aplikuje jedno pravidlo po druhém. Ačkoli je pravidel více druhů, tak všechna mají společný interface, který poskytuje metodu `apply()`. Výstupem je pak pole objektů, které reprezentují výstup pravidla. V těchto výstupních objektech jsou uložené nové zprávy, které mají být rozeslané, spolu s dalšími nezbytnými informacemi (jako například destinace). Po aplikaci každého pravidla je zkontrolováno, zda pravidlo neterminovalo, a pokud ne, tak je aplikováno další pravidlo.

4.2.3 Implementace pravidel

Klíčovou částí jsou pravidla, která udávají chování celého systému. Tato pravidla jsou implementována v jádře jako vlastní objekty, které mají společný interface `Rule` poskytující metodu `apply()`. Kód uvnitř pravidla musí být z principu rozdílný, což je řešeno pomocí funkcionálního rozhraní. Jedná se o koncept, kdy třída má deklarovanou nějakou funkci, jejíž kód je ale definován až při vytváření instance. Tímto způsobem je tedy možné vytvořit instance jedné třídy, které budou stejnou metodou vykonávat jiné kódy.

Pravidlo dostane vždy na vstupu přístup ke stavové reprezentaci a aktuální zprávě (pomocí definovaných rozhraní). Uvnitř pravila pak může být libovolný kód, který pracuje s těmito rozhraními. Typicky se jedná o podmínku, po které následuje vygenerování nějaké nové zprávy nebo jiného pravidla. Počáteční sada pravidel je definována v metodě `loadAllRules()`, která je volána při inicializaci tohoto objektu. Pro ilustraci je zde ukázáno jedno z jednodušších pravidel pro odkládání upozornění na RSS články, když není vhodná denní doba:

```
rulebook.add(new RuleSimple(coreStateMap,
        "Rule for delaying NEW_RSS if daytime is bad",
        (core, msg, output, self) -> {
    if (msg.getCode() == NEW_RSS) {
        int hour = LocalDateTime.now().getHour();
        if (hour < 8 || hour > 22) {
            sendBackTomorrow(msg, output, 10);
            terminate(output, self.getDescription());
        }
    }
}));
```

Metoda `sendBackTomorrow()` je implementována v pomocné třídě a využívá modul pro ukládání zpráv zmíněný v části 3.8.

Pravidla byla implementována tímto způsobem z toho důvodu, aby bylo možné jejich implementaci snadno změnit a nebo případně doplnit další, úplně nové typy pravidel. Samotná pravidla jsou uložena v proměnné třídy, což umožňuje mít více sad pravidel a v případě potřeby mezi nimi přepínat. Tato možnost v práci využita nebyla, ale mohla by být výhodná například v situaci, kdy by jedno jádro sloužilo pro více uživatelů (každý by měl vlastní sadu pravidel) a nebo by jeden uživatel vyžadoval více režimů (každý režim by měl vlastní sadu pravidel).

4.3 Servisní protokoly

Jedná se o protokoly, které byly navržené pro jednotné chování všech modulů při připojování, odpojování a explicitním potvrzování zpráv. Protokoly specifikují chování jádra a modulu, typy a obsah zpráv, které protistrana očekává a způsob jejich zpracování. Názvy protokolů byly zvoleny v angličtině a jsou takto referované i v této práci.

4.3.1 Enroll Protocol

Tento protokol definuje chování modulu a jádra při navazování spojení. Enroll protokol tedy popisuje postup, jakým jsou nové moduly začleněny do systému.

Předpoklady pro funkčnost jsou:

- Modul musí znát své přístupové údaje k brokeru.
- Každý modul má svoje unikátní jméno, které slouží pro jeho identifikaci.
- Všechny moduly (včetně jádra) poslouchají na veřejné frontě `broadcast`.

Samotné kroky enroll protokolu jsou:

1. Modul pošle do fronty `core` servisní zprávu ohlašující, že má zájem se zapojit do systému. Tato zpráva musí obsahovat hlavičky:
   - `msgID` → identifikační kód zprávy ve formátu UUID
   - `msgOrigin` → obsahuje název modulu, který zprávu vytvořil
   - `msgType` → musí nabývat hodnoty „service“
   - `msgCode` → musí nabývat hodnoty „here“
   - `stateVars` → volitelná položka, může obsahovat seznam stavových proměnných, které modul bude přinášet.
   - `listeningOn` → obsahuje název fronty, na které bude modul poslouchat (v této práci se shoduje s názvem modulu).
2. Jádro zprávu zachytí, zpracuje, a pokud je v pořádku, tak pošle zpět servisní potvrzující zprávu, která vypadá následovně:
   - `msgID` → identifikační kód zprávy ve formátu UUID
   - `msgOrigin` → obsahuje název modulu, který zprávu vytvořil
   - `msgType` → musí nabývat hodnoty „service“
   - `msgCode` → musí nabývat hodnoty „confirm“
   - `respID` → musí obsahovat ID původní zprávy, kterou potvrzuje
3. Enroll sekvence je hotova, modul začne svou normální činnost.

4.3.2 Death Protocol

Tento protokol je používán ve chvíli, kdy nějaký modul ukončí svou činnost. Protokol popisuje postup pro případ záměrného i nečekaného ukončení modulu.

Předpoklady pro funkčnost jsou:

- Modul musí znát své přístupové údaje k brokeru.
- Modul již úspěšně dokončil enroll protokol a je k systému plně připojený.

Pro *záměrné* ukončení vypadá sekvence takto:

1. Modul ví, že bude ukončen, což se může stát buďto tím, že dostane zprávu s kódem `die` a nebo sám z nějakého důvodu chce provést ukončení.
2. Modul pošle do fronty `core` servisní zprávu s kódem `dying`. Tato zpráva může obsahovat (kromě povinných položek) ještě volitelnou hlavičku `description`, ve které může být popsán důvod ukončení.
3. Jádro zprávu přijme a zpracuje. Během zpracování dojde k odstranění všech stavových proměnných, které byly vázané na tento modul.
4. Jádro vyšle zprávu o ukončení modulu s kódem `dead` na adresu `broadcast`. Kromě povinných položek obsahuje tato zpráva hlavičku `content`, ve které je napsané ID modulu, který byl ukončen. Důvodem pro tento poslední krok je to, aby ukončovaný modul měl šanci zkontrolovat, že jeho ukončení bylo zaznamenáno a případně aby ostatní moduly měly přístup k této informaci, pokud by ji z nějakého důvodu potřebovali.

Pro případ, že modul skončí neočekávaně (třeba fatální chybou), tak nemá možnost odeslat jádru zprávu. Taková situace je řešena pomocí periodického ohlašování modulů, které bylo zmíněno výše. Toto periodické volání v každém z modulů způsobí počátek enroll protokolu a moduly provedou s jádrem opět celou sekvenci. Pokud ale na výzvu o ohlášení modul neodpoví, tak je jádrem považován za nefunkční nebo ukončený. V takovém případě je postup stejný jako při záměrném ukončení, pouze se začíná od třetího bodu.

4.4 Dialogový modul

Implementace dialogového modulu je v jazyce Python, důvodem volby je především fakt, že knihovna umožňující využití platformy SpeechCloud je implementována právě v Pythonu. Tento modul běží na virtuálním stroji ZČU, který má povolený přístup pro využití platformy SpeechCloud. Po spuštění skriptu je spuštěný dialogový manažer, jehož chování bylo definováno třídou `MyDialog` odvozenou od třídy `Dialog`. Následně je spuštěn WebSocket server, který poslouchá na všech IP adresách daného stroje na portu 8888 [2].

V modulu je také implementován MQ klient, který umožňuje komunikaci s jádrem. Tento klient je při startu spuštěn a připojen souběžně s dialogovým manažerem. Při inicializaci je dále vytvořena asynchronní smyčka, ve které probíhá obsluha samotného dialogu.

Za běhu pak modul vyčkává na příkazy od jádra, podle kterých ovládá dialog. Tyto příkazy byly již popsány v kapitole 3.7, pro připomenutí se jedná o start odposlechu a porozumění, ukončení odposlechu a porozumění a syntéza daného textu. Tyto tři příkazy jsou právě vykonávané uvnitř zmíněné asynchronní smyčky.

Původní implementace smyčku obsahovat vůbec neměla a MQ klient a dialogový manažer měly běžet paralelně v samostatných vláknech, přičemž by si předávali události a příkazy. Tato verze se bohužel nepodařila zprovoznit kvůli problémům s kombinací asynchronní implementace knihovny SpeechCloud.dialog a paralelním během. Řešením bylo použití asynchronní smyčky, která umožňuje simulovat paralelní běh dialogového manažeru a MQ klienta v jednom vlákně.

4.4.1 Poloautomatické generování gramatik

Součástí inicializace dialogového manažeru je nahrání bezkontextových gramatik pro porozumění řeči, viz kapitoly 2.2.1 a 3.7.1. Hlavním problémem s těmito gramatikami byla jejich tvorba.

Pro vytvoření takovéto gramatiky bylo potřeba předem zmapovat všechny možné typy interakcí, které by mohl uživatel se systémem mít. Poté byl podle nich vytvořen a sepsán seznam významů, které se budou v těchto interakcích pravděpodobně vyskytovat. Až do této části se jednalo o poměrně jednoduchý proces, který byl

udělán „ručně“ bez větších komplikací. Poté ale bylo potřeba najít ke všem významům všechna možná slova (synonyma), která mohou daný význam reprezentovat a následně ke každému tomuto slovu ještě sepsat všechny jeho tvary.

Počet významů byl v této fázi vývoje přibližně 40 (během testování bylo ještě několik významů přidáno). Každý význam by mohl být (odhadem) reprezentován průměrně deseti slovy a každé slovo by mohlo mít (odhadem) průměrně deset až dvacet forem. Psaní této gramatiky by bylo pro člověka již výrazně náročnější, především časově. Z toho důvodu byl implementován program, jehož účelem je tuto práci usnadnit. Tomuto programu byl dán název *Gimir*.

Gimir byl implementován v programovacím jazyce Go, stejně jako všechny moduly kromě jádra a dialogového modulu. Jedná se o Command Line Interface (CLI) nástroj, který automatizuje hledání synonym k zadanému významu a následně hledání jejich forem. Jeho funkce spočívá v tom, že uživatel zadá typ entity, kterou chce vygenerovat a určí výchozí slova. Pro ilustraci může příkaz pro vytvoření významové entity vypadat takto:

```
generate meaning read from číst,četba
```

Program následně prohledává internet a snaží se najít co nejvíce synonym k zadaným slovům. Hledání synonym je rekurzivní (nalezená synonyma jsou také následně použita jako základ pro vyhledávání, ovšem pouze do jisté hloubky rekurze), což zvyšuje šanci na nalezení co největšího počtu synonym. Toto hledání je poměrně agresivní a slepé, takže často je nalezeno více slov, než by mělo být skutečně ponecháno. Z tohoto důvodu jsou slova vždy předkládána uživateli ke kontrole (i když je možné nastavit „důvěru“ a Gimir pak uloží vše, co najde). Po nalezení všech synonym jsou ke každému slovu hledány všechny jejich tvary. Po dokončení právě popsaného procesu je výsledek uložen v paměti programu a je možné pokračovat s generováním dalšího významu.

Weby použité pro získávání dat tímto softwarem:

- Wikislovník (https://cs.wiktionary.org/wiki/Wikislovník:Hlavní_strana)
- Dobrý slovník (http://www.dobryslovnik.cz/cestina)
- Skloňuj.cz (https://sklonuj.cz/)

Výstup pak může být textový přímo do konzole a nebo do souboru. Výstup do konzole je možné zvolit ve formátu ABNF a nebo formát, který je o něco lépe čitelný člověkem. Do souboru je pak výstup vždy ve formátu ABNF.

Kromě generování nových významů dokáže Gimir také načíst existující textový soubor v ABNF formátu, rozparsovat obsah do paměti a následně jej modifikovat. Vedle významů je také možné generovat pouze jednotlivá slova a byla také implementována funkce pro stažení vybraných českých jmen. Dále byla implementována jednoduchý optimalizační algoritmus, který hledá v datech duplicity a odstraňuje ne-významová slova (například zvratné „se“).

Je vhodné zmínit, že Gimir byl vytvořen pouze jako podpůrný program při tvorbě gramatiky pro tuto práci a pro nějaké širší využití by bylo potřeba jej ještě doplnit o další funkce, lépe ošetřit chybové stavy a zdokonalit implementované algoritmy. Použití tohoto nástroje však výrazně usnadnilo a urychlilo generování použité bezkontextové gramatiky. Výsledná gramatika má přibližně tisíc slov a významů a byla s pomocí tohoto nástroje vytvořena za několik desítek minut.

4.5 Ostatní moduly

Implementace všech ostatních modulů je ve svém principu stejná a s výjimkou dialogového modulu a jádra jsou všechny moduly implementované v jazyce Go. Jak již bylo v této práci několikrát řečeno, chování modulů je záměrně udržováno jednoduché a většina řízení je soustředěna v jádře.

Základní chování všech modulů funguje tak, že po spuštění se připojí pomocí MQ klienta k brokeru a pokusí se zahájit enroll protokol. Pokud se jim nedostane odpověď, tak čekají na výzvu k ohlášení na adrese `broadcast`. Po úspěšném vstoupení do systému vyčkávají moduly na příchod zpráv, podle kterých vykonávají příkazy. Co konkrétně jednotlivé moduly dělají bylo popsáno v kapitole 3.8.

V případě, že moduly sami aktivně vykonávají nějakou činnost bez nutnosti příchozího příkazu (například modul sledující aktivitu uživatele), tak je tato činnost spuštěna paralelně, takže není ovlivněna zbytkem běhu programu.

5 Vyhodnocení

V této kapitole je rozebrán a zhodnocen výsledek práce, především schopnost systémy interagovat s uživatelem a jednotlivé funkce, kterými systém disponuje.

5.1 Interakce s uživatelem

Výsledný systém hlasové asistentky nabízí uživateli možnost interakce pomocí hlasu, jak je typické pro hlasové dialogové systémy, ale také pomocí klasického grafického rozhraní. Pro využití hlasového vstupu a výstupu je prozatím potřeba, aby měl uživatel otevřený prohlížeč na stránce spravované dialogovým manažerem, grafické rozhraní je pak realizováno pomocí grafických notifikací s tlačítky.

Jako webové rozhraní bylo použito výchozí GUI implementované platformou SpeechCloud. V tomto rozhraní je několik tlačítek pro základní ovládání hlasového dialogu, pod kterými je textový výpis právě prováděných akcí a stavů. I když je použité webové rozhraní vhodné spíše pro vývoj a testování, než pro běžné „produkční“ použití, tak díky modulárnímu návrhu systému by ale nemělo být obtížné přidat nebo vyměnit moduly, které budou poskytovat uživatelsky přívětivější rozhraní. Jedním takovým příkladem může být vývojový kit Google VoiceKit zmíněný v kapitole 2.6, jehož integrace je plánovaným rozšířením.

Díky použití bezkontextových gramatik s upravenou strukturou (viz 3.7.1) může uživatel odpovídat systému a pronášet příkazy v přirozeném jazyce, aniž by bylo potřeba dodržovat nějaké předem dané fráze nebo hlídat klíčová slova. To činí komunikaci se systémem pro uživatele více přívětivou a přirozenou, což je jedním z důležitých faktorů pro všechny systémy, které mají jakýmkoli způsobem interagovat s lidmi.

Grafické notifikace s tlačítky zařizuje modul popsaný v sekci 3.8.4. Tento způsob interakce je jednoduchý a pro uživatele intuitivní, ale není příliš vhodný pro složité interakce nebo dlouhé texty. Hodí se tedy spíše pro krátká a jednoduchá upozornění, na které je očekávána žádná a nebo pouze jednoduchá reakce. Vzhled, chování a možnosti těchto notifikací jsou dány operačním systémem a použitým správcem notifikací, v této práci jimi jsou GNU/Linux s xfce4-notifyd.

5.2 Implementovaná funkčnost

Celkově byla v rámci této práce implementována do systému hlasové asistentky následující funkčnost:

1. **Správa emailové schránky, včetně možnosti odpovědi**
   Systém je schopen hlídat zadanou emailovou schránku a upozornit uživatele na nepřečtené zprávy v reálném čase. Upozornění může být hlasové nebo vizuální a při jeho výběru je brána v potaz aktivita uživatele u počítače a denní doba.

   V případě vizuální notifikace může uživatel upozornění odložit na později, nechat si zobrazit schránku v prohlížeči a nebo jen notifikaci „odkliknout“. V případě hlasového upozornění je systém schopen reagovat na žádost o přečtení odesílatelů, přečtení předmětu zpráv, přečtení obsahu zpráv. Dále uživatel může hlasem přikázat označení emailů jako přečtených, nechat si je zobrazit ve webovém klientovi a nebo přímo nadiktovat odpověď. Všechny odpovědi na hlasové upozornění je možné filtrovat jménem odesílatele, počtem a pořadím.

   Když tedy systém uživatele upozorní „Máte tři nepřečtené maily“, tak uživatel může odpovědět „A kdo mi píše?“, po přečtení odesílatelů pak může pokračovat třeba příkazem „Tak poslední od Tomáše označ jako přečtený“.

2. **Sledování RSS feedů**
   Systém sleduje definované RSS zdroje a při detekci nového článku zobrazí uživateli notifikaci, pomocí které může uživatel článek rovnou zobrazit, nebo upozornění odložit. Systém opět hlídá denní dobu a aktivitu uživatele a notifikace zobrazuje pouze ve vhodnou chvíli.

3. **Hlasové ovládání počítače**
   Systém umožňuje uživateli ovládat vybrané základní funkce na počítači, konkrétně se jedná o otevření prohlížeče na specifikované URL a nebo otevření průzkumníka souborů na zadané lokaci.

   Možnost otevřít prohlížeč na zadané adrese je široce využitelná činnost, protože značná část činností běženého uživatele se odehrává právě ve webovém prohlížeči. Tyto vybrané funkce jsou převážně ilustrační, modul zajišťující tuto činnost ovšem dokáže komunikovat přímo s operačním systémem, takže by bylo možné provádět na počítači i jiné operace.

6 Závěr

Cílem této práce bylo navrhnout a realizovat systém hlasové osobní asistentky, který by uživateli umožňoval základní obsluhu vybrané komunikační platformy a byl by schopný reagovat na vnější podněty. Zvolenou komunikační platformou byl email, jako reakce na vnější jevy bylo určeno sledování RSS zdrojů a toto celé ve spojení s ovládáním vybraných funkcí uživatelského počítače.

Při návrhu byl kladen důraz především na modularitu a univerzálnost, aby nebyl navržený a implementovaný systém fixovaný pouze na účely této práce, ale bylo možné jej snadno modifikovat pro případné jiné užití. Dále byl brán zřetel na otevřenost, modifikovatelnost a rozšiřitelnost systému, což souvisí s již zmíněnou univerzálností.

Tyto vlastnosti se následně promítly do architektury a konceptů, které byly v systému použité a jsou popsané v kapitole 3. Stručně by šlo celý systém popsat jako množinu programů, referovaných jako *moduly*, které mezi sebou komunikují. Centrální modul označený jako *jádro* slouží jako centrála řídící akce všech ostatních modulů, které zařizují jednotlivé funkčnosti systému.

Chování celé soustavy modulů je řízeno pomocí sady pravidel, která jsou uložena v jádře spolu se stavovou reprezentací. Hlavním rysem je univerzálnost a schopnost dynamických změn za běhu systému. Výhodou této implementace je možnost pravidla jednoduše kdykoli změnit, čímž se změní chování celého systému, aniž by bylo potřeba modifikovat jakýkoli kód jiných částí. Zároveň je možné jednoduše přidávat libovolná nová pravidla a tím tak zvyšovat komplexitu chování systému. Omezení tohoto řešení se mohou projevit při snaze o implementaci více složitých chování, protože je potřeba požadované chování předem zmapovat a převést do podoby pravidel. V takovém případě by ale bylo možné pravidlový systém kompletně vyměnit za jiný styl rozhodovacího algoritmu, například neuronové sítě. Pravidla jsou prozatím definována přímo ve zdrojovém kódu jádra, takže pro změnu pravidel by bylo potřeba modifikovat příslušnou část kódu. Jedním z plánovaných vylepšení do budoucna je načítání a správa pravidel pomocí nějakého lépe přístupného formátu, například textového souboru nebo dokonce grafického rozhraní.

Interakce s uživatelem pomocí hlasového dialogu zajišťuje jeden z modulů a k jeho funkci byla využita platforma SpeechCloud popsaná v kapitole 2.6. Ke správnému

fungování hlasového dialogu bylo potřeba vytvořit vhodnou sémantickou gramatiku, která je použita při porozumění řeči. Klasický návrh gramatiky popsaný v kapitole 3.7.1 se ale ukázal náročný a zdlouhavý, hlavním omezením byla nutnost předem definovat všechny možné fráze, které by uživatel mohl říci. I když by řešením této komplikace mohlo být poučení uživatelů o možnostech interakce se systémem, tak byl navržen mírně odlišný styl gramatiky, která se zaměřuje na extrakci významů a konceptů z promluvy, namísto hledání přesných frází. Tento návrh je detailně také popsán v kapitole 3.7.1.

Pro zjednodušení generování gramatik byl vytvořen pomocný nástroj (viz 4.4.1), který tvorbu potřebné gramatiky výrazně urychlil a zjednodušil. Po implementaci a testování tohoto řešení se ukázalo, že takto sestavené gramatiky umožňují uživateli přirozenější interakci a lepší robustnost porozumění řeči. Při testování byl ale také objeven problém tohoto řešení v podobě skloňování českých jmen, kdy systém nedokáže poznat rozdíl například mezi „Martina“ (mužské jméno ve 2. pádě) a „Martina“ (ženské jméno v 1. pádě). Potenciálním řešením by mohlo být sledování předchozích slov před jmény a odhad pádů podle předložek, ale tato možnost prozatím nebyla ozkoušena - jedná se o jednu z věcí, která je plánována jako budoucí rozšíření.

Při návrhu a implementaci ostatních modulů se nevyskytly žádné větší komplikace, pravděpodobně díky jejich jednoduchým konceptům. V současném stavu systému je možné, aby byl od každého typu modulu aktivní pouze jeden, připojení více modulů stejného typu by mohlo vést v některých případech k nepředvídatelnému chování systému. Umožnění připojení více modulů stejného typu je dalším potenciálním rozšířením do budoucna, realizace bude vyžadovat schopnost jádra zapůjčovat dynamické konfigurace pro jednotlivé moduly, což umožní jejich rozlišení.

Celkově je výsledkem této práce funkční prototyp hlasové asistentky splňující všechny požadované body, ovšem jak již bylo několikrát zmíněno, díky univerzální modulární architektuře je prakticky neomezeně rozšiřitelný, takže poskytuje bohaté možnosti pro další rozšíření a vylepšení.

Použitá literatura

1. *Seznam hlasový asistent*. 2022. Dostupné také z: `https://napoveda.seznam.cz/cz/prohlizec-seznam/seznam-hlasovy-asistent/`.
2. ŠVEC, J. *Knihovna SpeechCloud.dialog* [Interní dokument]. Katedra Kybernetiky ZČU, 2022.
3. ŠVEC, J. *Hlasové dialogové systémy* [Učební text KKY/HDS]. 2021.
4. IRCING, P. *Úvod do strojového vnímání prostředí: Rozpoznávání řeči, akustické a jazykové modelování* [Učební text USVP]. 2021.
5. MCTEAR, M.; CALLEJAS, Z.; GRIOL, D. *The Conversational Interface: Talking to Smart Devices*. Springer, 2016.
6. PSUTKA, J.; MÜLLER, L.; MATOUŠEK, J.; RADOVÁ, V. *Mluvíme s počítačem česky*. Prague: Academia, 2006. ISBN 80-200-1309-1.
7. ŠVEC, J.; NEDUCHAL, P.; HRŮZ, M. Multi-modal communication system for mobile robot. 2022.
8. STANISLAV, P.; ŠMÍDL, L.; ŠVEC, J. An automativ training tool for air traffic control training. *Interspeech*. 2016.
9. CHÝLEK, A.; ŠMÍDL, L.; ŠVEC, J. Multimodal Dialog with the MALACH audiovisual Archive. *Interspeech*. 2019.
10. ŠVEC, J.; BULÍN, M.; PRAŽÁK, A.; IRCING, P. *UWebASR - Web-based ASR engine for Czech and Slovak*. 2018.
11. *ActiveMQ Artemis Documentation*. 2022. Dostupné také z: `https://activemq.apache.org/components/artemis/documentation/`.
12. *ABNF Grammar Guide*. 2022. Dostupné také z: `https://speech-center.verbio.com/documentation/abnf`.
13. *Internet Message Access Protocol*. 2021. Dostupné také z: `https://datatracker.ietf.org/doc/html/rfc9051`.
14. *Simple Mail Transfer Protocol*. 2008. Dostupné také z: `https://datatracker.ietf.org/doc/html/rfc5321`.

Seznam obrázků