Masarykova univerzita
Fakulta informatiky

Dialogová platforma s podporou komunikace pomocí protokolu XMPP

Bakalářská práce

Jakub Peschel

Brno, jaro 2015

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: Mgr. Luděk Bártek, Ph.D.

Poděkování

Rád bych poděkoval panu doktoru Bártkovi za jeho připomínky, které mi pomohly práci dovést do konce. Také bych chtěl poděkovat své rodině za to, že mi umožnili studovat, a své milé, že mě při tvorbě podporovala.

Shrnutí

Tato práce se zaobírá možností komunikace s dialogovými aplikacemi pomocí protokolu XMPP. Zkoumá technologie a tři možné přístupy k možnosti vytvořit fungující komunikaci přes XMPP na nekomerčním interpretu JVoiceXML. Jako hlavní část práce vznikl komunikační modul, jenž zajišťuje možnost komunikace s dialogovými aplikacemi přes XMPP. Práce také zahrnuje dvě testovací aplikace, které ukazují funkčnost modulu.

Klíčová slova

JVoiceXML, VoiceXML, OpenFire, XMPP, Java

Obsah

1 **Úvod** . . . 3
- 1.1 *Cíl práce* . . . 4
- 1.2 *Struktura práce* . . . 4

2 **Technologie** . . . 5
- 2.1 *eXtensible Markup Language* . . . 5
- 2.2 *VoiceXML* . . . 6
  - 2.2.1 Architektura VoiceXML . . . 6
  - 2.2.2 Syntaxe VoiceXML . . . 7
- 2.3 *Gramatiky a Speech Recognition Grammar Specification* 10
  - 2.3.1 Syntaxe SRGS . . . 10
- 2.4 *JVoiceXML* . . . 14
- 2.5 *XMPP* . . . 15
  - 2.5.1 XEP . . . 16
  - 2.5.2 Smack . . . 18
  - 2.5.3 OpenFire . . . 19
- 2.6 *Open Virtualization Format* . . . 19
  - 2.6.1 VirtualBox . . . 19

3 **Návrh a Implementace** . . . 21
  - 3.0.2 JVoiceXML . . . 22
  - 3.0.3 XMPP server . . . 23
- 3.1 *Komunikační modul* . . . 23
  - 3.1.1 Agenti . . . 24
  - 3.1.2 Admin . . . 26
- 3.2 *Testovací aplikace* . . . 29
  - 3.2.1 Quiz . . . 29
  - 3.2.2 TODOList . . . 30

4 **Závěr** . . . 33

A **Návod** . . . 39
- A.1 *Instalace* . . . 39
  - A.1.1 Instalace JVoiceXML . . . 39

A.1.2 Instalace serveru OpenFire 40
A.1.3 Instalace CommunicationModule 40
A.2 *Přidávání aplikací* 41
A.3 *Komunikace* 41
B **Gramatika pro rozhodování odpovědi** 43
C **Seznam možných příkazů pro aplikaci TodoList** 47

Kapitola 1

Úvod

Stejně jako se rozvíjela schopnost počítačů zpracovávat zvuk a zvukovou stopu, byl zájem na vytvoření možnosti komunikovat s počítačem přirozenou řečí. Vznikaly různé dialogové aplikace a systémy, které měly za úkol komunikovat s člověkem, a zároveň vždy byla snaha, aby byly schopny komunikovat přirozenou řečí. Na počátku bylo problémem, že neexistovala žádná jednotná specifikace, která by stanovila, jak vlastně promluvu syntetizovat, rozpoznávat, či jak vést dialog samotný. Velký posun do tohoto odvětví informatiky přineslo konsorcium W3 [1], které se ujalo VoiceXML fóra [2] a vytvořilo The Voice Browser Working Group [3]. Pracovní skupina vypracovala specifikace pro kompletní dialog, který se skládá z rozpoznání, vedení dialogu a syntézy promluvy. Konkrétně pro rozpoznání řeči se jedná například o specifikaci Speech Recognition Grammar Specification (SRGS, česky Specifikace gramatik pro rozpoznání promluvy) [4], či specifikaci Semantic Interpretation for Speech Recognition (SISR, česky Sémantická interpretace pro rozpoznání řeči) [5]. Pro vedení dialogu se jedná o značkovací jazyk VoiceXML [6] a pro následnou syntézu značkovací jazyk Speech Synthesis Markup Language (SSML, česky značkovací jazyk pro syntézu promluvy) [7]. V dnešní době již máme velké množství nástrojů. Spousta z nich je však silně komerční a jsou tedy mnohdy finančně nedostupné.

Jedním z nekomerčních řešení je VoiceXML interpret JVoiceXML [8]. Tento interpret dovoluje uživateli komunikovat s VoiceXML programem, avšak uživatel musí komunikovat přímo přes interpret, což se může zdát mnoha uživatelům jako příliš složité. Jednou z možností, jak odstranit tuto složitost, je poskytnout uživatelům možnost snazší komunikace přes zavedené komunikační protokoly, které jsou uživatelům již známé. Jako jeden z vhodných kandidátů se jeví i protokol XMPP

[9], který dnes a denně mnoho uživatelů používá skrze aplikace jako Facebook Chat či Google Talk. [10]

1.1 Cíl práce

Cílem práce bylo pro interpret jazyka VoiceXML vytvořit rozhraní pro komunikaci po Internetu pomocí protokolu XMPP, zastřešeného společností XMPP Standards Foundation. eXtensible Messaging and Presence Protocol(XMPP, česky rozšiřitelný protokol pro posílání zpráv a zjištění stavu) je komunikační protokol, který popisuje komunikaci po počítačové síti. Dále bylo potřeba navržené řešení otestovat pomocí vytvořené testovací aplikace a sepsat podrobný popis, jak interpret zprovoznit a přidávat do něj další aplikace, které ve formě VoiceXML dokumentu poté vedou dialog s klientem. Pro tuto práci byla vybrána textová implementace interpretu JVoiceXML. V první fázi práce byla vytvořena fungující komunikace pomocí protokolu XMPP, na což bylo vyžadováno zprovoznění XMPP serveru. V druhé fázi bylo potřeba vytvořit komunikační modul mezi XMPP serverem a aplikací JVoiceXML. V třetí fázi práce byly implementovány zvolené ukázkové aplikace a byly nasazeny na vytvořenou platformu. Na závěr bylo potřeba tento přístup zdokumentovat a vytvořit z něj obecný postup.

1.2 Struktura práce

Práce je rozpracována do tří částí. Na začátku se zabývá nástroji využitými při realizaci. Druhá část práce se věnuje již samotné dokumentaci implementace komunikačního modulu a ukázkové dialogové aplikaci. V příloze A. se pak nachází návod na zprovoznění XMPP serveru, instalaci a rozběhnutí JVoiceXML serveru a jejich následné propojení.

Kapitola 2

Technologie

2.1 eXtensible Markup Language

eXtensible Markup Language (zkráceně XML, česky rozšiřitelný značkovací jazyk) [11] vznikl roku 1996 jako velice jednoduchý textový formát, jenž měl být čitelný jak člověkem, tak strojem. Vznikl jako derivát jazyku SGML (Standard Generalized Markup Language, česky standardní generalizovaný značkovací jazyk). Hlavní požadavky kladené na XML byly obecnost, aby byl velice snadno rozšiřitelný a velmi jednoduše použitelný.

V dnešní době se XML velice často používá jako formát pro výměnu dat mezi aplikacemi, či pro psaní metadat určených například pro publikování dokumentů.

Základem XML je vždy text psaný v Unicode. Dokument je složen z elementů. Na dokument je kladeno několik podmínek:

- Musí obsahovat pouze jeden kořenový element.
- Neprázdný element musí být ohraničen počáteční a ukončovací značkou.
- Hodnoty atributů musí být uzavřeny v uvozovkách.
- Elementy mohou být zanořeny, ale nesmí se překrývat.

Nejčastěji se dnes na zpracování používá buďto DOM parser (Document object model, česky objektový model dokumentu) [12], který z XML dokumentu v paměti počítače vytvoří stromovou strukturu, nebo SAX parser (Simple API for XML, česky Jednoduché aplikační rozhraní pro XML) [13], který prochází dokument postupně a vyvolává události, na které uživatel či programátor reaguje.

Jelikož XML je opravdu snadno rozšiřitelný a jednoduše zpracovatelný, bylo z něj odvozeno mnoho značkovacích jazyků, jako například XHTML (eXtensible HyperText Markup Language, česky rozšiřitelný hypertextový značkovací jazyk), ODF (OASIS Open Document Format for Office Applications, česky OASIS otevřený formát dokumentu pro kancelářské aplikace), SSML (Speech Synthesis Markup Language, česky značkovací jazyk pro syntézu řeči), či VoiceXML, nebo XMPP. [14]

2.2 VoiceXML

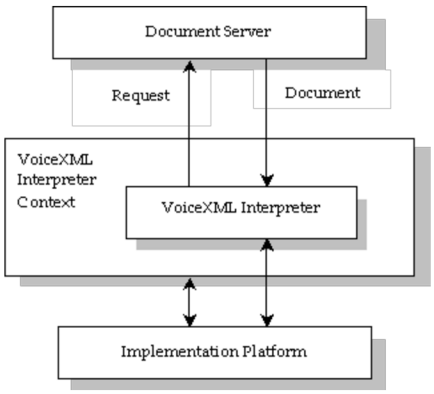
Jazyk VoiceXML 2.0. (zkráceně VXML) [2] je jazyk vytvořený za účelem popsat a řídit dialog člověka s počítačem. První verze číslo 0.9 byla ustanovena roku 1999 VoiceXML fórem, které vzniklo zapříčiněním firem AT&T Corporation, IBM, Lucent a Motorola.

Jelikož VoiceXML je odvozen z jazyka XML, obsahuje množství značek pro dialogová rozhraní. Jsou zde značky, které instruují syntetizér zvuku, jak má vypadat výsledná promluva, či jak správně rozpoznávat, co bylo řečeno. Dále je možné určovat pomocí značek, které části dialogu budou následovat po rozpoznání promluvy, či jen zopakovat uživatelův vstup.

Cílem VoiceXML bylo usnadnit vývoj aplikací pomocí odstranění nutnosti programovat detaily závislé na platformě, poskytnutí nástrojů pro tvorbu jak jednoduchých, tak složitých dialogových aplikací, či oddělení klientského kódu od servisní logiky. [2]

2.2.1 Architektura VoiceXML

VoiceXML se skládá z několika komponent. Jsou jimi dokumentový server, kontext VoiceXML interpretu, samotný VoiceXML interpret a implementační platforma. Dokumentový server, nebo také web server, poskytuje na žádost klientských aplikací jednotlivé VoiceXML dokumenty VoiceXML interpretu přes kontext interpretu. VoiceXML interpret tyto dokumenty zpracuje. Kontext VoiceXML interpretu spolu s interpretem poté monitorují vstupy. Implementační platforma je poté spravována jak kontextem, tak samotným interpretem. Jejím úkolem je vyvolávat události jako reakci na uživatelův vstup. [6]



Obrázek 2.1: Architektura VoiceXML

2.2.2 Syntaxe VoiceXML

Každý dokument jazyka VoiceXML musí obsahovat jeden kořenový element <vxml>, ve kterém pomocí atributu version a jmenného prostoru xmlns určíme, kterou verzi VoiceXML v dokumentu využíváme. Dokument poté může obsahovat libovolné množství elementů <form> a <menu>.

Zpracování elementů form a menu

Element <form> slouží pro prezentování výstupu a získávání vstupu a je tedy klíčovou komponentou jakéhokoliv VoiceXML dokumentu. Tento element může obsahovat dva atributy: id a scope (česky rozsah). Atribut id nám slouží k identifikaci pro budoucí skoky na tento dialog. Scope zase určuje, zdali gramatiky uvedené ve formuláři platí pouze pro aktuální formulář, když je zvolena hodnota dialog, či jestli budou platné v celém dokumentu, v případě zadané hodnoty document. Atribut scope je defaultně nastaven na hodnotu dialog. Každý element <form> může obsahovat:

- Množinu elementů <form>
- Deklaraci proměnných jiného typu, nežli <form>
- Obsluhu událostí
- Blok logických operací vykonaných v případě určité kombinace přiřazených vstupních proměnných

Prvky elementu <form>

Prvky elementu <form> můžeme rozdělit do dvou kategorií. První kategorie jsou prvky řídící. Jsou jimi elementy <block> a <initial>. Element <initial> slouží k vyzvání uživatele k poskytnutí vstupu. Jakmile je alespoň jedna proměnná naplněna, je tento blok odstraněn. Element <block> slouží jako sekvence určená pro poskytnutí výstupu a počítání s proměnnými. Neslouží však k získávání vstupu. Druhou kategorií jsou prvky pro získání vstupu. Jsou jimi elementy <field>, <record>, <transfer>, <object> a <subdialog>. Element <field> slouží k získání vstupu skrze gramatiky typu DTMF či ASR. Element <record> slouží k zaznamenání vstupu ve formátu nahrávky, kterou je možné uživateli přehrát. Element <transfer> přesměrovává uživatele na jiné telefonní číslo. Element <object> vytváří platformě specifický objekt s nejrůznějšími parametry. Výsledek platformě specifického objektu je uložen ve formě ECMAScript objektu. Element <subdialog> se chová jako volání funkce. Vyvolá další dialog, jehož návratem je ECMAScript objekt jako výsledek.

Element <menu>

Element <menu>, jak už z názvu vypovídá, nám poskytuje možnost vybrat si z předem připravených možností. Jednotlivé možnosti vytváříme pomocí elementu <choice>. Zjednodušeně se dá říci, že element <menu> je pouze omezeným elementem <form> s jedním políčkem field, jehož akce jsou vybudovány pomocí elementů <choice>.

Algoritmus interpretace elementu <form>

Form Interpretation Algorithm (FIA, česky algoritmus interpretace elementu <form>) pohání interakci uživatele a VoiceXML elementy <form> a <menu>. Algoritmus se musí vypořádat s následujícím:

- Inicializace formuláře
- Napovídání, zahrnující i management příslušných proměnných
- Aktivace a deaktivace gramatik na úrovni formuláře a prvků formuláře

- Poskytnutí promluvy, která odpovídá gramatice daného formuláře, promluvené v jiném menu či formuláři
- Opuštění formuláře, pokud uživatel vytvořil promluvu odpovídající jinému formuláři či menu
- Zpracování mnohočetného naplnění gramatik z několika různých formulářů.
- Vybrání, který element bude navštíven dále
- Vybrání záchytného elementu na zpracování jakékoli události vyplynuvší ze zpracovávání prvku elementu <form>

Algoritmus interpretace se dá rozdělit do několika fází. Tyto fáze jsou iniciální fáze a hlavní smyčka, skládající se z fáze výběru, fáze sběru a fáze zpracování. Během iniciální fáze je potřeba všechny čítače nastavit na jedničku. Každá proměnná dokumentu je nastavena do iniciální hodnoty v závislosti na pořadí v dokumentu.

Během hlavní smyčky pravidelně dochází k fázi výběru dalšího elementu <form>, který bude vykonán. Pokud algoritmus narazí na element <goto>, je jako další <form> vybrán takový, který má id specifikované v atributu next. Pokud algoritmus nenarazí na element <goto>, je vybrán takový element, kterému je jako první nastavena vstupní podmínka na false. Pokud jsou již zpracovány všechny elementy, pak algoritmus zavolá implicitní volání elementu <exit>, který ukončí běh programu.

Jakmile je vybrán další element, následuje fáze sběru. V této fázi se zpracují vstupní elementy a je získán vstup od uživatele. Jestliže byl sběr úspěšný, algoritmus přechází do další fáze, kterou je fáze zpracování. Stejně tak do další fáze přechází, jestliže byla vyvolána nějaká událost.

Během fáze zpracování zpracovává algoritmus buďto událost, nebo vstup uživatele. Jestliže se jedná o událost, pak je vyvolán příslušný záchytný element. Pokud se jedná o vstup, pak pokud splňuje gramatiku jiného formuláře než je momentálně vybrán, tak se algoritmus postará o přechod do správného formuláře. Pokud vstup splňuje gramatiku vybraného formuláře,

pak algoritmus přiřadí do příslušné proměnné sémantický význam, nastaví atribut filled na true a přechází znovu do fáze výběru.

Pohyb mezi elementy <form>

Pro pohyb v dokumentech VoiceXML slouží element <goto> neboli skok. Tento element je určen nejen pro řízení programu v rámci jednoho dokumentu, ale také nám umožňuje skočit na jiný dokument, či na specifický <form> v jiném dokumentu. Element <goto> obsahuje atributy next, nextitem a expr. Atribut next může být určen buďto k odkazu na jiný <form> v rámci našeho dokumentu, nebo může být užit ke skoku do jiného dokumentu. Pokud má sloužit ke skoku v rámci našeho dokumentu, pak atribut next obsahuje id vybrané značky <form>. Pokud je určen ke skoku do jiného dokumentu, je potřeba do něj vložit URI adresu druhého dokumentu. Atribut nextitem nám slouží ke skoku v rámci elementu <form>. Stejně jako v případě atributu next i zde se používá id vybraného formuláře. Poslední atribut expr dovoluje definovat adresu dokumentu pomocí výrazu v ECMASciptu.

2.3 Gramatiky a Speech Recognition Grammar Specification

Speech Recognition Grammar Specification 1.0. (SRGS, česky specifikace gramatiky pro rozpoznání řeči)[4] je vytvořen v rámci projektu W3 Voice Browser Activity [15]. Tato specifikace slouží k tvorbě gramatik, jež jsou používány pro rozpoznávání promluvy. [4]

2.3.1 Syntaxe SRGS

Syntaxe SRGS je prezentována ve dvou formách:

- Augmented BNF syntax
- XML syntax

Obě dvě formy mají výrazovou sílu kontextových gramatik, jež je dostačující pro většinu aplikací rozpoznávajících řeč. Dále se budu věnovat pouze XML syntaxi, která byla využita v rámci této práce.

SRGS gramatiku je možné využít buďto v externím souboru, či ve VoiceXML dokumentu pomocí inline gramatiky definované uvnitř elementu <form>.

Ukázka 2.1: Příklad inline SRGS gramatiky

```
<grammar mode="voice" root="anone" scope="document">
        <rule id="anone">
                <one-of>
                        <item>ano</item>
                        <item>ne</item>
                </one-of>
        </rule>
</grammar>
```

Pokud je využita gramatika v externím souboru, pak se za kořenový element považuje element <grammar>, naopak pokud využíváme inline gramatiky, je nutné tento kořenový element specifikovat pomocí atributu root, kterému přiřadíme id příslušného elementu.

Každá gramatika v externím souboru by se pak měla skládat ze tří částí:

1. Úvodní xml prolog
2. Kořenový element <grammar> s atributy určujícími tyto vlastnosti:
   - jmené prostory XML (xmlns)
   - XML schéma (xmlns:xsi spolu s xsi:schemaLocation)
   - Verze (version)
   - Jazyk (xml:lang)
   - Mód (mode)
   - Kořenové pravidlo (root)
   - Formát značek (tag-format)
   - Bázové URI (xml:base)
3. Obsah elementu <grammar> definujícího gramatiku, který se může skláda z:

- definice výslovnostního lexikonu
- metadat
- pravidel

XML prolog

Jelikož je gramatika SRGS zapsána v XML formátu, je nutné definovat legálně správný prolog. Jedná se o informaci ohledně verze a typu dokumentu. Prolog může obsahovat další informace jako komentáře, příkazy procesoru a další obsah povolený XML pro prolog.

element <grammar>

Kořenovým dokumentem každé externě definované gramatiky je element <grammar>. Tento element obsahuje pět zajímavých atributů:

- Atribut xml:lang pro definici jazyka použitého uvnitř elementu <grammar>.
- Atribut mode pro stanovení, zdali se jedná o gramatiku pro hlasovou komunikaci, či pro DTMF.
- Atribut root pro stanovení hlavního pravidla.
- Atribut tag-format, pro stanovení.
- Atribut xml:base pro určení bázové URI.

Atribut xml:lang je důležitý pro všechny gramatiky, jejichž mód je hlasový. Pokud není atribut nastaven, přebírá se nastavení z rodičů.

Atribut root stanovuje, které z pravidel bude hlavní. Tento atribut nemusí být vyplněn, pokud se jedná o gramatiku zapsanou v externím souboru, avšak je dobrým zvykem tento atribut vyplňovat u jakékoliv gramatiky.

Atribut tag-formát je volitelný a je potřeba jej vyplnit, pokud využíváme rozšíření pravidel o sémantické značkování. Obsahem atributu tag-formát je tag-formát identifikátor identifikující typ obsahu a verzi. Tento identifikátor se skládá ze dvou částí: první je označení semantics, které určuje, že obsahem je sémantické

značkování, a druhá je verze specifikace W3C Semantic Interpretation for Speech Recognition specification (SISR, česky specifikace konsorcia W3 pro sémantickou interpretaci pro rozpoznání řeči). Výsledný atribut poté může vypadat takto:

```
<gramma tag-format="semantics/1.0" ...>
```

Výslovnostní lexikon
Pokud využíváme SRGS gramatiku pro rozpoznávání řeči a ne pro DTMF, je možné v gramatice definovat vlastní výslovnostní lexikon. Tento lexikon může být využit v případě, že vestavěné lexikony řečových rozpoznávačů nejsou dostatečné. Do SRGS je možné tyto lexikony přidat pomocí elementu <lexicon> s atributem uri, který odkazuje na příslušný lexikon.

Metadata
SRGS poskytuje také možnost přidat do gramatiky metadata pomocí elementu <metadata>. Tento element slouží jako kontejner pro informace zapsané v Resource Description Format (RDF, česky formát popisu zdrojů) [16] spolu s využitím dalších popisných formátů jako Dublin Core (DC) [17] či MAchine-Readable Cataloging (Marc, česky Strojově čitelná katalogizace) [18].

Element <rule>
V SRGS jsou jednotlivá pravidla definovaná pomocí elementu <rule>. Gramatika těchto pravidel může obsahovat více, avšak pouze jedno z nich je hlavní, a to takové, které je uvedené v atributu root elementu <grammar>. Element <rule> má dva atributy, povinný id, který obsahuje identifikátor pravidla a volitelný scope, který značí, zdali je pravidlo privátní (private), či veřejné (public). Pokud je pravidlo veřejné, může být volané pomocí elementu <ruleref> z jiného dokumentu. Implicitně je atribut scope nastaven na hodnotu private.

elemty <one-of> a <item> V jednotlivých pravidlech je potřeba zaznamenat přijatelné výrazy. Toto děláme pomocí elementu <item>. Tento element nám určuje, které vstupy jsou validní.

Samozřejmě programátor chce rozeznat, zdali uživatel řekl například výraz "ano"nebo jestli vyřknutým výrazem bylo "ne". Pokud je tedy výrazů, které jsou validní, více než jen jeden, pak je možné v gramatice použít element <one-of>. Ten umožňuje nést více elementů <item> a vrací hodnotu, pokud byl vyřknut alespoň jeden z výrazů uvedených v elementech <item>.

Pokud použijeme kombinaci elementu <one-of> a <item>, je možné v elementu <item> specifikovat v atributu weight váhu daného výrazu. Tato váha se pak může využít u rozpoznávání, avšak specifikace SRGS nezaručuje, že daný rozpoznávač bere atribut weight v úvahu.

Sémantické značkování

Velice důležitou součástí gramatik je také možnost přiřadit řeči sémantický význam. Způsob sémantického značkování je popsán v protokolu, který rozšiřuje SRGS, Semantic Interpretation for Speech Recognition 1.0., zkráceně SISR [5]. Tato technologie dovoluje v gramatice určit, na co se přepíše daný vstup uživatele, tedy například že výraz "jasně"se interpretuje jako výraz "ano"nebo hodnota true. Toto je velice důležité, neboť ve větších gramatikách by následné testy na to, kterou hodnotu uživatel myslí, narůstaly do závratných velikostí.

SISR poskytuje pro sémantické značkování značku <tag>. Tato značka je určena pro použití jako potomek elementu <item>. Pokud pak gramatika přijme nějaký vstup, vrátí místo daného vstupu to, co je vyplněno v textovém elementu značky <tag>. [5]

2.4 JVoiceXML

Java VoiceXML (zkráceně JVoiceXML nebo JVXML) [8] je otevřený interpret jazyka VoiceXML, vytvořený v jazyce Java. Jazyk Java byl využit kvůli snadné přenositelnosti mezi platformami.

Cílem JVoiceXML je vytvořit open-source, volně přístupný a snadno použitelný interpret VoiceXML, který by požíval levné technologie a byl dobře použitelný pro Interactive voice response systémy (IVR, česky interaktivní hlasová odezva).

JVoiceXML je vydáno pod licencí GNU Lesser General Public License, která je méně restriktivní oproti GNU General Public License, a která dovoluje aplikacím používajícím knihovnu použít jinou než copyleftovou licenci.

JVoiceXML je klient-server aplikací. Programátor zajišťuje komunikaci se serverovým JVoiceXML pomocí Java Naming and Directory Interface (JNDI, česky rozhraní Java objektů a adresářových služeb) [19].

JVoiceXML pro programování klientské části poskytuje balík knihoven org.jvoicexml, které obsahují metody pro spojení se serverovou aplikací, obsluhující VoiceXML dokumenty a dokumentový server, metody pro obsluhu událostí a další.

JVoiceXML je stále ve vývoji a může se tedy stát, že ne všechny funkcionality VoiceXML jsou stoprocentně funkční. [8]

2.5 XMPP

Extensible Messaging and Presence Protocol [9] je komunikační protokol určený pro takzvaný Instant Messaging (česky instantní posílání zpráv) a zjišťování přítomnosti protějšku komunikace. Samotná komunikace je zabalena ve formátu XML a XMPP servery ji poté přenáší k určenému klientovi. Díky formátu XML se tedy ukázalo, že je stejně vhodná i pro přenos dat mezi aplikacemi.

První verze protokolu XMPP vznikla v lednu roku 1999 pod názvem Jabber a její tvůrcem byl Jeremie Miller. Protokol byl oznámen jako technologie pro instant messaging a zjišťování přítomnosti a Jeremie Miller zveřejnil i první server nazvaný jabberd. Během roku 1999 proběhl rapidní vývoj v této oblasti a vzniklo několik open-source klientů stejně jako pár knihoven a rozšíření pro tento protokol. V půli roku 1999 bylo pak zažádáno o podporu protokolu při zpracování standardu Internet Engineering Task Force (IETF, česky „Komise techniky Internetu“). První verze Jabberu však neuspěla kvůli nedostatečné organizovanosti a aktivitě komunity. V roce 2002 pak bylo znovu zažádáno o zpracování protokolu a IETF jej přijala s tím, že název bude změněn na neutrální Extensible Messaging and Presence Protocol.

Hlavními výhodami XMPP je decentralizovanost, otevřenost a flexibilita. Architektura sítě XMPP je velice podobná e-mailu. Každý má

možnost zprovoznit si svůj vlastní XMPP server a není nijak omezen. Jelikož standard je otevřený, je možné si jej dobře upravit.

Každý uživatel sítě XMPP má svůj jednoznačný identifikátor, takzvaný JabberID nebo JID, který je velice podobný e-mailové adrese. Je tvaru uživatelskéJméno@doména, kde doménou může být i IP adresa. [9]

2.5.1 XEP

XMPP Extension Protocols (XEP, česky protokoly rozšiřující XMPP) jsou standardy schválené organizací XMPP Standard Foundation k rozšíření protokolu XMPP.

Existuje několik typů XEP protokolů. Mohou být standardní, které definují buďto protokol, který má záměr být součástí XMPP, nebo například soubor protokolů, jen stanovuje požadavky na přizpůsobení se požadavkům, informační, které typicky definují nejlepší chování již existujících protokolů, historické, které popisují protokoly, jež vznikly dříve něž byl uveden popis schvalování, humorné či vtipné, které vznikají za účelem pobavit a nebudou pravděpodobně nikdy využity, nebo procedurální, které definují procesy či aktivity, jež by měly být následovány organizací XMPP Standard Foundation.

Jednotlivé protokoly XEP se poté mohou nacházet v nejrůznějších stádiích vývoje. Tyto stádia jsou:

- Experimentální (Experimental)
- Navržený (Proposed)
- Náčrt (Draft)
- Finální (Final)
- Aktivní (Active)
- Odložený (Deferred)
- Stažený (Retracted)
- Odmítnutý (Rejected)
- Zavržený (Deprecated)

- Zastaralý (Obsolete)

Zajímavé je, že samotný popis toho, jakého typu jsou protokoly a jak se nazývají stádia, je popsán v protokolu XEP-0001: XMPP Extension Protocols. [20]

XEP-0095: Stream Initiation

XEP-0095: Stream Initiation [21] je protokol, který definuje zahájení datového streamu mezi jakýmikoli dvěma XMPP entitami. Protokol navíc povoluje poslání jednoduchých metadat o streamu a nejrůznějších připojitelných frameworcích, aby bylo možné definovat profily pro jednotlivé případy užití, jako například přenos souborů.

XEP-0096: SI File Transfer

XEP-0096: SI File Transfer [22] je protokol definující přenos souborů po síti pomocí protokolu XMPP. Momentálně definuje profil pro protokol XEP-0095: Stream Initiation. Proces přenosu dat se pak skládá ze dvou částí. První je nabídka přenosu dat, kterou může druhý účastník buď přijmout nebo odmítnout. Druhá fáze je pak odesílání samotného souboru. SI File Transfer obsahuje kořenový element <file>. Kořenový element poté může obsahovat čtyři atributy:

- name - jméno souboru
- size - velikost dat v bytech
- date - datum poslední modifikace souboru
- hash - MD5 suma obsahu

Tyto atributy jsou posílány pouze při fázi nabídky přenosu dat.

Dále může obsahovat potomky <desc> a <range>, z nichž oba jsou nepovinní. Element <desc> smí být použit pouze při nabízení přenosu a poskytuje stručný popis dat, která uživatel zasílá. Element <range> může být zaslán při nabízení přenosu, čímž klient signalizuje možnost odeslání dat po částech. Pokud se <range> nachází v odpovědi na nabídku, může obsahovat dva atributy: offset, který určuje od kolikátého bytu budou data zaslána, a length, který určuje kolik dat v bytech bude přijato.

Oba atributy jsou nepovinné a zaslání elementu <range> bez nich je považováno za zaslání odpovědi bez elementu <range>.

XEP-0085: Chat State Notifications

XEP-0085: Chat State Notifications [23] definuje indikaci stavu komunikujícího partnera. Tento protokol přináší do komunikace nové prvky, jež nám oznamují stav partnera. Jsou jím elementy:

- <active/>
- <inactive/>
- <gone/>
- <composing/>
- <paused/>

Ty pak mohou být zasílány buďto jako součást zprávy, či případně jako samostatné notifikace.

2.5.2 Smack

Smack [24] je open-source knihovna pro tvorbu klientských aplikací pro XMPP, vytvořena komunitou Ignite Realtime. Tento software je pod licencí Apache License, která dovoluje použití jak pro komerční, tak i nekomerční účely.

Knihovna Smack je určena pro snadnou použitelnost, proto dovoluje programovat klienta pro XMPP i bez znalosti protokolu XMPP.

Knihovna poskytuje vysokoúrovňové metody pro práci se zprávami a nenutí uživatele programovat na úrovni packetů. Umožňuje například využití třídy org.jivesoftware.smack.ChatManager, která zpracovává jednotlivé chaty, či rozhraní org.jivesoftware.smack.ChatManagerListener, které dovoluje reagovat na události spojené s chaty. Dále Smack poskytuje rozhraní org.jivesoftware.smack.MessageListener, které slouží na zpracování zpráv.

Knihovna obsahuje nejrůznější funkce pro zpracování jednotlivých rozšíření protokolu XMPP jako FileTransfer, Prezence Indicators a další. [24]

2.5.3 OpenFire

Server OpenFire [25], dříve známý pod jménem WildFire, je XMPP server, který je napsán v jazyce JAVA. Tento server je licencovaný jak pod proprietární licencí, tak pod Apache Licence 2.0.

Openfire poskytuje rozsáhlé webové rozhraní, ke kterému je možno se standardně připojit na portech 9090(HTPP) a 9091(HTPPS). Toto webové rozhraní nám dovoluje například nakonfigurovat pluginy serveru, spravovat uživatele, skupiny a nastavení databáze. [25]

2.6 Open Virtualization Format

Open Virtualization Format (OVF, česky otevřený formát pro virtualizaci) [26], vytvořený organizací Distributed Management Task Force, Inc. (DMTF, komise distribuovaného managementu), je standardní balíkovací formát pro softwarová řešení založená na virtuálních systémech. International Organization for Standardization (ISO, česky Mezinárodní organizace pro standardizaci) tento protokol publikovala pod označením ISO 17203.

V dnešní době již existuje spousta virtualizačních nástrojů, mezi nejznámější se řadí programy Microsoft Virtual PC od společnosti MicroSoft, VirtualBox od společnosti Oracle, nebo VMware workstation od společnosti VMware, Inc. [26]

2.6.1 VirtualBox

VirtualBox, celým jménem Oracle VM VirtualBox [27], je multiplatformní virtualizační nástroj distribuovaný pro Linux/Unix, Windows i Mac OS. Zásadní výhodou VirtualBoxu je hardwarová virtualizace, jež dovolí virtualizovat i systémy hardwarově závislé. [27]

Kapitola 3

Návrh a Implementace

V následující kapitole rozeberu, jaké byly možnosti při návrhu řešení a která a z jakého důvodu byla vybrána. V další části kapitoly pak bude podrobně popsána testovací aplikace. Na závěr bude vysvětleno, jak se řešení nasazovalo na virtuální platformu.

Možnosti, jak zprovoznit komunikaci s JVoiceXML pomocí XMPP, byly v podstatě tři. První možností bylo vytvořit uvnitř JVoiceXML XMPP server, který by přijímal komunikaci ze sítě a tu následně zpracovával. V provozu by to pak vypadalo tak, že uživatel napíše na speciální adresu a komunikuje již s interpretem. Problémů by však mělo toto řešení několik. S aplikacemi nasazenými na tuto platformu by musel uživatel aplikace zadat specifickou cestu, alespoň pro spuštění VoiceXML programu, neboť by nebylo zřejmé, s kterou aplikací chce uživatel komunikovat. Další problém by byl s komplikovaným nasazováním nových aplikací. Tato možnost by kvůli neustálému rychlému rozvoji XMPP navíc nebyla dlouhodobě udržitelná.

Další možností je opačná situace, kdy by XMPP server měl uvnitř sebe implementován JVoiceXML a každou příchozí zprávu by zpracovával. Tato možnost se zdála vhodná, neboť komunikace by mohla probíhat tak, že vlastník nasazené aplikace by vytvořil uživatele, a každá příchozí zpráva by se zpracovávala ihned po příchodu na server. Problémem zde by však bylo, že pro každou aplikaci je takto potřeba simulovat samostatný běh JVoiceXML, což by mohlo zbytečně zatěžovat stroje, na kterých by byl takovýto server nasazen. Další nevýhodou oproti následujícímu řešení je, že by bylo při každé změně protokolu XMPP potřeba upravovat XMPP server.

Třetí možností, kterou jsem nakonec zvolil, bylo vytvořit modul, který by přijímal zprávy od samostatně fungujícího XMPP serveru a následně je předával JVoiceXML, který je serverovou aplikací. Výho-

dou tohoto řešení je, že díky přesné specifikaci XMPP je nezávislý na implementaci XMPP serveru. A tím pádem by i platforma byla dlouhodoběji udržitelná. Díky JVoiceXML, běžícímu jako server, se také velice snadno vyřeší nasazení více aplikací naráz, neboť s JVoiceXML komunikují vzdáleně. Serverové JVoiceXML pak již pomocí sessions (česky sezení) samo zvládne zpracovávat více požadavků naráz.

3.0.2 JVoiceXML

Pro vybrané řešení bylo potřeba nainstalovat JVoiceXML. Zvolil jsem verzi JVoiceXML 0.7.7.EA, jejíž zdrojový kód bylo možné nalézt na serveru sourceforge.org. Jelikož JVoiceXML je naprogramováno v Javě 1.7, je možnost jej sestavit s Javou 1.7, nebo je potřeba upravit konfigurace na pár místech. Prvně bylo nutné nahradit staré knihovny swingframework, které jsou v projektu využívány, za nové. Vybral jsem nejnovější dostupnou verzi knihovny 4.1.6, která bezchybně pracuje s Javou verze 1.8. JVoiceXML si bere tyto knihovny ze složky:

> jvoicexml-code/org.jvoicexml.config/3rdparty/
> /springframework/lib/spring_3.2.1

Bylo tedy možné je podstrčit čistě přejmenováním a nahrazením starých knihoven novými.

Následovně bylo potřeba vytvořit konfigurační soubory. JVoiceXML k tomuto poskytuje automatizované nástroje, takže je to velice rychlé. Ve složce jvoicexml-code/org.jvoicexml se zavolá příkaz *ant -f configuration.xml*, který vytvoří konfigurační soubory. Ještě před zavoláním tohoto příkazu je nutné vybrat si platformu pomocí xml souboru platforms.xml ve složce config-props/. Pro tuto práci byla vybrána textová implementace. Aby bylo možné JVoiceXML v textové variantě spustit, bylo potřeba ještě upravit Konfiguraci JNDI. Tu lze najít ve složce config/ v souboru jvxml-jndi.xml. V tomto xml souboru je potřeba přepsat existující obsah elementu <repozitory> na "text".

Jakmile byly správně nastaveny všechny konfigurační soubory, bylo nutné JVoiceXML zkompilovat pomocí příkazu *ant compile*. Jakmile je JVoiceXML zkompilované, je možné jej už jednoduše spouštět pomocí příkazu *ant -f run.xml run*, popřípadě pomocí zkrácené verze *ant run*. Při vypínání serveru JVoiceXML je pak nutné jej vždy ukončit pomocí příkazu *ant shutdown*.

3.0.3 XMPP server

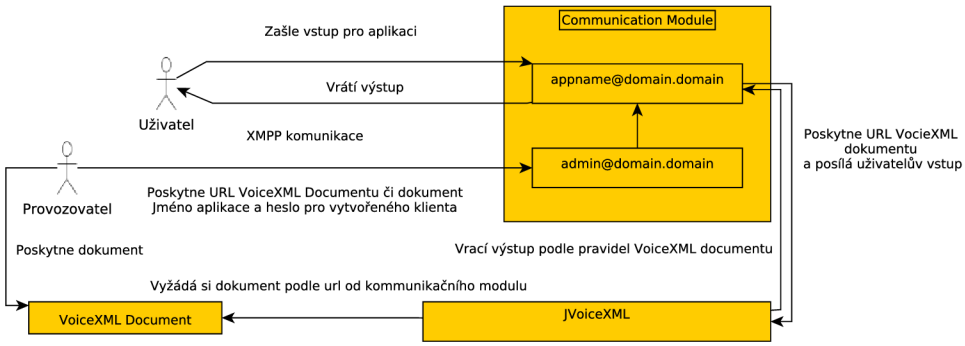
Pro zvolené řešení bylo potřeba vybrat a zprovoznit XMPP server. Platforma ale není omezena použitým serverem a na zprovoznění může být využita libovolná implementace XMPP serveru.

Z množství implementací XMPP serverů jsem zvolil Java implementaci OpenFire. OpenFire byl vybrán, protože plně podporuje knihovnu Smack, která byla vybrána pro tvorbu XMPP klienta.

Pro samotnou instalaci většina implementací poskytuje intuitivní instalátor, který pomoci řady kroků pomůže s nainstalováním severu.

OpenFire navíc poskytuje i přehledné webové rozhraní pro prvotní nastavení serveru.

3.1 Komunikační modul



Obrázek 3.1: Diagram komunikace mezi všemi aktéry

Pro tvorbu komunikačního modulu jsem zvolil prostředí Netbeans, které velice napomáhá při tvorbě Javových aplikací.

Komunikační modul měl zajišťovat komunikaci mezi uživatelem a běžícím JVoiceXML serverem. Možností, jak zprostředkovat tuto komunikaci, bylo několik, ale jako nejvíce vhodná se jevila možnost vytvoření Agentů, kteří se pak o vše starají. Pro tvorbu Agentů se mi jevilo jako vhodné použít třídu Javy Thread a pro každého Agenta vytvořit vlákno. Agent by pak obstarával komunikaci pro jednotlivé nasazené programy. Díky tomu, že by každé vlákno obstarávalo část

nezávisející na ostatních, odpadají i problémy s režií paměti, jako jsou race conditions či deadlocky.

V komunikačním modulu je pak možné rozeznat dva typy Agentů. První je Administrátor, zkráceně Admin, a druhý je jednoduše běžným Agentem obstarávajícím běžnou činnost.

3.1.1 Agenti

Agent v komunikačním modulu je vlákno obsluhující jeden VoiceXML dokument. Funguje jako automatický XMPP klient. V XMPP síti se tedy tváří jako klient, který je připojen. Pokud mu uživatel napíše, začne vykonávat činnost VoiceXML dokumentu.

Během inicializace tohoto klienta je potřeba vykonat několik činností:

1. vytvořit spojení na lokální server OpenFire
2. přihlásit se k účtu na lokálním serveru OpenFire
3. vytvořit chat manažer
4. přiřadit chat manažerovi příslušný chatListener
5. přiřadit chat manažerovi příslušný chatStateListener

Jakmile je Agent inicializován, zůstává nečinný, dokud mu někdo pomocí XMPP nepošle zprávu. Ve chvíli, kdy takovýto Agent dostane zprávu, se stane několik věcí. Při nově příchozí zprávě se na základě Jabber id vytvoří nový chat, jemuž se přiřadí unikátní thread id. Na tuto akci zareaguje příslušný chatListener, který pomocí metody chatCreated(Chat, bool) vykoná námi zvolený kód. Jakmile je vytvořen nový chat, znamená to, že uživatel má zájem komunikovat s JVoiceXML serverem. Je tedy potřeba se na server připojit.

JVoiceXML nám přináší pro textovou komunikaci speciální třídu TextServer. Tato třída nám pomáhá komunikovat s JVoiceXML serverem. Pro to, aby tuto komunikaci mohl vykonávat, je potřeba této třídě předat informace o spojení. JVoiceXML využívá pro komunikaci Java naming and directory interface spolu s Java remote method invocation (Java RMI, česky Java vzdálené vyvolání metod), které dovolují volat metody z jiného Java virtuálního stroje. Pro správnou inicializaci

je nutné mít nadefinované proměnné prostředí. Možnosti, jak je nadefinovat, jsou dvě. Buďto je možné poskytnout konfigurační soubor jndi.properties, nebo je můžeme nastavit přímo v kódu. Java naming and directory interface poté prvně načte proměnné prostředí, které nalezne v projektu, a poté je přepíše pomocí kódu v programu.

Jakmile je TextServer nainicializován, je potřeba načíst VoiceXML soubor a připravit ho pro JVoiceXML server. JVoiceXML server potřebuje URI souboru. Proto načteme soubor pomocí třídy File. Tato třída nám poskytuje metodu pro získání URI, které si modul uchová.

Dalším krokem je vytvoření sezení se serverem. JVoiceXML přichází s vlastní třídou Session, která pomáhá již obsloužit celé sezení pomocí tří metod. Metodou call(URI) spustí interpret JVoiceXML na zvolený dokument, metodou waitSessionEnd() pozastaví vykonávání kódu komunikačního modulu, dokud příslušné sezení neskončí a metodou hangup() ukončí sezení. V této fázi byla využita jen metoda call(URI) pro zahájení sezení.

Pro komunikaci je také potřeba předávat informace zpět uživateli. Pro to poskytuje JVoiceXML interface TextListener. Tento interface dovoluje upozornit či ošetřit případy, kdy je server odpojen, kdy je připojen, či kdy je připraven přijmout vstup či naopak kdy už skončil s přijímáním vstupu. Tento interface nám navíc poskytuje metodu outputSsml(SsmlDocument), která vrací výstup ve formátu Speech synthesis markup language(SSML, značkovací jazyk pro syntézu promluvy). JVoicexml nám poté poskytuje nástroje SsmlDocument a Speech pro získání pouze textové části dokumentu bez značek pro syntézu, kterou modul zasílá zpět uživateli.

Zároveň s připojením se k serveru je potřeba příslušně zpracovat přicházející zprávy. Metod, jak zpracovávat příchozí zprávy, je několik, ale jako nejvhodnější se jevilo využít interface knihovny Smack MessageListener. Tento interface má pouze jednu metodu, a to processMessage(Chat, Message). Tato metoda je automaticky vyvolána s každou zprávou patřící Chatu, ke kterému je interface přiřazen. V metodě processMessage(Chat, Message) je využit TextServer, který dovoluje zasílat vstup pomocí metody sendInput(String). Než je příchozí zpráva zaslána na server, je potřeba zkontrolovat, jestli vůbec nese zprávu, neboť XMPP dovoluje zasílat i packety, které informují o stavu či žádají o poskytnutí nějaké služby. Ačkoliv podle dokumentace knihovny Smack by tyto packety neměly spustit metodu processMessage(Chat,

Message), během implementace se ukázalo, že se tak často děje a že metodu spouští packety nesoucí informaci o stavu komunikace.

Pro ukončení komunikace s uživatelem modul využívá interface knihovny Smack ChatStateListener. Tento interface pomocí metody stateChanged() vyčkává, dokud se stav chatu nezmění. Jednotlivé stavy chatu jsou popsány v kapitole 2.5.1 v odstavci XEP-0085: Chat State Notifications. Pro ukončení komunikace je velmi důležitý stav <gone/>, který signalizuje, že uživatel ukončil chat. V tuto chvíli Agent přeruší běh JVoiceXML pomocí zavolání session.hangup() a ukončí Chat. Nové otevření Chatu ze strany uživatele tedy znovu spustí metodu ChatListener.chatCreated() a nastartuje dialog od začátku, jako by uživatel znovu vytočil telefonní číslo. [28]

3.1.2 Admin

Admin je rozšířením třídy Agent. Jeho hlavní náplní je obsluhovat požadavky na vytvoření nových aplikací, přijímat VoiceXML dokumenty a vše validovat. Komunikační modul obsahuje tolik Agentů, kolik je nasazených VoiceXML aplikací, avšak obsahuje pouze jednoho Agenta typu Admin. Tento Agent musí mít na XMPP serveru kompletní administrátorská práva. Tento Agent je zabudován přímo do kódu modulu a předpokládá se, že příslušný XMPP server tohohle uživatele vždy má.

Pravomoce Admina se skládají ze tří důležitých věcí:

- Vytváření nových uživatelů
- Přijímání souborů
- Testování přístupnosti souborů nově vytvořených aplikací

Vytváření nových uživatelů

Jelikož se každá aplikace chová jako jeden uživatel XMPP sítě, je potřeba je vytvářet. Aby mohli být vytvořeni na serveru, bylo potřeba zajistit přístupová práva. XMPP server OpenFire vytváří vždy jednoho hlavního uživatele s administrátorskými právy. Z toho důvodu jsem zvolil, že bude obsluhovat veškeré vytváření uživatelů. Každý uživatel se skládá ze tří důležitých částí. Tyto části jsou:

- uživatel na XMPP serveru

- XMPP klient
- VoiceXML dokument

Tvorba nového uživatele probíhá zhruba ve čtyřech fázích. Nejprve vlastník VoiceXML dokumentu kontaktuje Admina. Ten mu vrátí informaci, jak má vypadat požadavek na nasazení nové aplikace. Vlastník si poté může vybrat jméno a heslo, pod kterým bude aplikace registrována v XMPP serveru, a buďto nějakou formu URI, například URL, nebo jméno VoiceXML dokumentu uloženého na serveru. Admin poté zkontroluje, zdali je dokument přístupný, a pokud kontrola uspěje, začne vytvářet nového uživatele.

Pro tvorbu nového uživatele bylo využito třídy AccountManager z knihovny Smack. Tato třída dovoluje snadné vytvoření nového účtu pomocí metody crateAccount(Name,Password). Pro vytvoření nového učtu se předpokládá, že XMPP server má nastavenou možnost vytváření nových účtů.

Přijímání souborů

VoiceXML dokument je možno použít dvěma způsoby. Buďto jej vlastník může umístit na server, nebo kdekoli, kde budou vzdáleně přístupné. Pokud si zvolí první variantu, musí využít možnosti XMPP na posílání souborů. Tento transfer se řídí rozšířením XEP-0096: SI File Transfer. Přenos souboru se tedy skládá z více částí. Nejprve je potřeba umožnit vůbec započetí přenosu – na to je nutné mít Admina v seznamu, neboli rosteru.

Pro spravování tohoto seznamu bylo využito třídy Roster a rozhraní RosterListener. RosterListener reaguje na události jako změna stavu přítomnosti či přidávání nebo odebrání uživatele ze seznamu. Roster samotný pak poskytuje možnost nastavení subscription módu na možnost přijímat vše, což dovoluje přijmout každého uživatele, který o přidání do seznamu požádá.

Jakmile je Admin na seznamu, může začít přenos souboru. Uživatel pomocí svého klienta požádá o zaslání souboru, Admin požadavek příjme a zpracuje. Na přijímání požadavků na přenos využívá modul třídu FileTransferManager. Tento manager

umožňuje zpracovávat příchozí požadavky pomocí rozhraní FileTransferListener. Toto rozhraní obsahuje speciální metodu fileTransferRequest(request), která je vyvolána s každým příchozím požadavkem. Příchozí požadavek je poté možné na základě kritérií přijmout či odmítnout. Jelikož se předpokládá, že uživatel pošle VoiceXML dokument, popřípadě externí gramatiky, modul přijímá všechny soubory, jež jsou mu zasílány. Jako výsledek potvrzení přenosu je vytvořena instance třídy IncomingFileTransfer, která umožňuje uložit soubor do dokumentu pomocí metody recieveFile(File).

Testování přístupnosti aplikací

Za běhu komunikačního modulu se může stát spousta neočekávaných událostí, například že soubor přestane být přístupný, JVoiceXML server spadne, či XMPP server přestane běžet. Tyto události nelze vždy předvídat a mohou být zapříčiněny různými důvody, jako třeba smazání VoiceXML dokumentu či jeho přemístění, spuštění JVoiceXML session se špatným VoiceXML dokumentem, ať už třeba nevalidním, či že JVoiceXML jej není schopen interpretovat. Ošetření těchto situací je v komunikačním modulu vyřešeno na třech místech:

1. Pokud XMPP server přestane běžet, uživatel nebude moci dosáhnout komunikačních Agentů a nebude tedy mít možnost aplikaci ani spustit.

2. Pokud z jakéhokoliv důvodu spadne JVoiceXML, pak komunikační modul při vytváření session nebude úspěšný a oznámí uživateli, že program selhal.

3. Komunikační modul při vytváření Agenta kontroluje, zdali je modul dostupný. Pokud se však v průběhu stav změní a VoiceXML dokument bude třeba přesunut, Agent sice vytvoří session, ale nebude s ním schopen se připojit na JVoiceXML server a oznámí uživateli, že aplikace selhala. [28]

3.2 Testovací aplikace

Pro otestování jsme s panem doktorem Bártkem zvolili dvě aplikace. Jedna je vytvořená jen v rámci jednoho VoiceXML dokumentu a obsahuje krátký vědomostní kvíz a druhá je aplikace sestávající se z více VoiceXML dokumentů, externích gramatik a jednoduché serverové aplikaci napsané v jazyce Java. Druhá aplikace je jednoduchý TODO list.

3.2.1 Quiz

Testovací aplikace Quiz měla za cíl ukázat nasazení jednoduché aplikace na vytvořenou platformu a zároveň pobavit a informovat uživatele této aplikace. Quiz je napsán v jazyce VoiceXML a sestává se pouze z jednoho VoiceXML dokumentu, v rámci něhož jsou použity inline gramatiky a všechna data využitá v aplikaci.

Implementace

Aplikace se skládá z uvítacího textu, série pěti otázek, nápovědy, zhodnocení a ukončení. Kdykoliv během průchodu aplikací je možné aplikaci ukončit, či si zavolat nápovědu.

Ukončení, stejně jako nápověda, je spouštěno jako reakce na event. Na odchycení eventu je využito elementu <catch>. Event samotný je vyvolán elementem <throw>. V případě nápovědy je využito elementu <help>, který odchytí event *help*. Na odchycení využije univerzální gramatiku, kterou má JVoiceXML implementovanou. Aby bylo možné danou gramatiku využít, je potřeba na začátku dokumentu umístit příkaz:

```
<property name="universals" value="help"/>
```

Každá otázka se skládá z jednoho elementu <form>. Uvnitř tohoto formu se nachází element <field>, pomocí kterého získáme odpověď a element <filled>, který vypíše reakci na odpověď. Uvnitř <field> se nachází gramatika, jejíž příklad je možné vidět v ukázce B.1. Dále je zde vypsána testová otázka pomocí elementu <prompt>. Uvnitř <filled> se nachází dvakrát <if>. První slouží k rozpoznání, zdali uživatel nechce ukončit aplikaci a druhý k vyhodnocení odpovědi. Pokud odpověď byla správná, aplikace přičte do proměnné points bod. Uživateli je pomocí

promptu oznámena správná odpověď, odpověď, kterou zvolil, a zajímavá informace týkající se otázky. Následovně aplikace přechází na další otázku pomocí elementu <goto>.

Po projití všech pěti otázek aplikace oznámí, jak si při odpovídání uživatel vedl a přejde k ukončení. Vyhodnocení probíhá vypsáním procent úspěšných odpovědí. Na získání procentuální hodnoty využívá aplikace element <value>, kde v atributu expr pomocí ECMAscriptu zpracuje proměnou points.

Úvod a ukončení obsahuje výpis pomocí elementu <prompt>. Aplikace je ukončena vypsáním informace, že skončí, a následným zavoláním elementu <exit>.

SRGS Gramatiky

Jelikož cílem této testovací aplikace bylo ukázat funkčnost řešení při nasazení čistě v rámci jednoho dokumentu, bylo nutné použít inline gramatiky. Dokument se skládá z řady testových otázek, na které jsou čtyři různé odpovědi. Správná je jen jedna odpověď. Gramatika se tedy skládá ze čtyř itemů "a", "b", "c"a "d"a dvou kontrolních itemů odkazujících na konec a nápovědu. Každý z těchto itemů pak má nadefinované možnosti, jak mohou vypadat odpovědi. Na závěr těchto variant je elementem <tag> určena správnost odpovědi pro danou otázku.

Obdoba gramatiky, kterou můžete nalézt v příloze B., se poté nachází v každém elementu <form>, který nese otázku, uvnitř elementu <field>.

3.2.2 TODOList

Druhou testovací aplikací je jednoduchý TODOList s rozhraním skrze VoiceXML. Aplikace se skládá ze tří částí:

- Databáze
- Webová aplikace
- VoiceXML rozhraní

Databáze

V aplikaci je potřeba pro uživatele ukládat textový záznam.

Předpokládal jsem, že aplikace nebude čítat spousty záznamů, a proto jsem se rozhodl vytvořit textovou databázi sestávající se z XML dokumentů. Každý uživatel při přihlašování do aplikace uvádí jméno a heslo. Pod tímto jménem je mu vytvořen XML dokument. Dokument je formátu:

Ukázka 3.1: Formát databázových souborů

```
<?xml version="1.0" encoding="UTF-8"?>
<todo>
        <passwd>Heslo uživatele</passwd>
        <todoItem id="pořadí v~seznamu"
          done="boolean vyjadřující stav úkolu">
                Text úkolu samotného
        </todoItem>
</todo>
```

Jednotlivé elementy <todoItem> pak vyjadřují jednotlivé úkoly zadané uživatelem a uchovávají jejich stav a jejich popis.

Webová aplikace

Jelikož VoiceXML umí odesílat získaná data pomocí elementu <submit> pomocí HTTP metody POST, rozhodl jsem se testovací aplikaci napsat pomocí jazyka Java. Aplikace se řídí modelem Model-View-Controller(MVC, česky model-pohled-ovladač) a obsahuje tři důležité části:

- Třída TodoList
- JavaServer Page (JSP)
- Třída Exchanger

Třída TodoList je klasickou částí controller. Jedná se o potomka třídy HttpServlet, který obsahuje pouze implementaci metody doPost(). Příchozí požadavky jsou předávány Třídě Exchanger, která pomocí metody processMessagge(request), zpracuje data a vrátí obsah dynamicky generované odpovědi. Metoda doPost poté pomocí šablony view.jsp a dynamicky vygenerovaného obsahu sestaví výslednou odpověď která je odeslána zpět interpretu JVoiceXML

JavaServer Page slouží jako dokumentové vzory, do kterých program doplní dynamická data, která se pak vykonávají. V naší aplikaci je view.jsp využito pro výpis akcí, které uživatel provedl. celý dokument je VoiceXML, které připraví VoiceXML dokument který je zasílán jako odpověď na Post request zaslaný z JVoiceXML serveru. V modelu MVC zastává view.jsp část view, neboli pohled.

Třída Exchanger je poslední částí MVC modelu, a to model. Tato třída obstarává veškerou komunikaci s textovou databází. Třída Exchanger poskytuje metody pro vytváření nových dokumentů, pro vytváření nových záznamů, úpravu stávajících záznamů a také jejich mazání.

Každý požadavek od uživatele obsahuje atribut action obsahující hodnoty například *vypis, pridej, odeber, oznac*, pomocí kterých pak třída Exchanger zpracovává requesty.

VoiceXML rozhraní

Během vývoje této aplikace se projevil zatím největší nedostatek nedodělanosti JVoiceXML. Ukázalo se, že JVoiceXML neumí zpracovávat element gramatiky <ruleref special="GARBAGE"/> ani <ruleref special="NULL"/>. Stejně tak textová implementace není schopna zpracovat element VoiceXML <record>. Z těchto důvodů není aplikace schopna přijímat libovolné řetězce, ale pouze takové, které jsou definované v gramatikách. Také je stále omezena možnost tvorby gramatik zapomocí využití opakování pravidel, neboť i to JVoiceXML nezpracovává správně. Z tohoto důvodu je možnost použití aplikace TodoList velice omezena na několik jmen a několik kombinací činností. O omezeních v použití TodoListu se dozvíte v příloze C.

Kapitola 4

Závěr

Dialogové aplikace nabízí světu netušené možnosti nejen jak se vyrovnat s hendikepem, ale i jak zjednodušit rozhraní pro běžné uživatele. Aby využívání dialogových aplikací bylo příjemné pro uživatele, je potřeba jim poskytnout vhodné kanály. Komunikace pomocí XMPP je jedním z kanálů, které jsou využívány dnes a denně a jeví se jako vhodný prostředek pro komunikaci. Tato bakalářská práce představila možnost komunikace uživatele s dialogovou aplikací pomocí protokolu XMPP. Během práce byla vytvořena platforma, na kterou lze snadno nasadit dialogovou aplikaci a se kterou jde lehce komunikovat.

Během vývoje se ukázalo, že zvolený způsob řešení pomocí komunikačního modulu je vhodný a funkční, je také snadno rozšiřitelný na další způsoby komunikace, nejen textové. Modul je také dlouhodobě udržitelný, neboť není příliš závislý jak na XMPP, tak na JVoiceXML. Ukázalo se také, že interpret JVoiceXML je stále ve vývoji a vnáší tudíž velké omezení na nasazované aplikace. Také se během implementace objevil bug v knihovně Smack – knihovna není schopna vyjednat datový tok v rámci komunikace na localhostu a realizace přenosu dat tudíž není možná. V současnosti se podílím na řešení této situace a zdá se, že bude brzy vyřešena.

Jako vhodné pokračování by mohlo být zlepšení zabezpečení, rozšíření o hlasovou komunikaci, či přidání možností pro poskytovatele aplikací. Také by bylo vhodné rozšířit JVoiceXML o podporu textového vstupu pro DTMF gramatiky, speciálních gramatik, či textovou variantu elementu <record>. Dále by bylo vhodné tuto platformu rozšířit o snadný instalátor, jenž by zájemcům o provozování platformy umožnil snadné nasazení.

Literatura

[1] W3 consorcium. *World Wide Web Consortium (W3C)* [online]. 30. 4. 2015 [4. 5. 2015]. <http://www.w3.org/>.

[2] *An Important Announcement to Members Of The VoiceXML Forum | VoiceXML Forum* [online]. 20. 4. 2015 [4. 5. 2015]. <http://www.voicexml.org/>.

[3] W3 consorcium. *The Voice Browser Working Group* [online]. 14. 6. 2014, 14. 6. 2014 [4. 5. 2015]. <http://www.w3.org/Voice/>.

[4] W3 consorcium. *Speech Recognition Grammar Specification Version 1.0* [online]. 16. 3. 2004, 15. 3. 2004 [4. 5. 2015]. <http://www.w3.org/TR/speech-grammar/>

[5] W3 consorcium. *Semantic Interpretation for Speech Recognition (SISR) Version 1.0* [online]. 5. 4. 2007, 2. 4. 2007 [4. 5. 2015]. <http://www.w3.org/TR/semantic-interpretation/>

[6] W3 consorcium. *Voice Extensible Markup Language (VoiceXML) Version 2.0* [online]. 16. 3. 2004, 15. 3. 2004 [4. 5. 2015]. <http://www.w3.org/TR/voicexml20/>

[7] W3 consorcium. *Speech Synthesis Markup Language (SSML) Version 1.0* [online]. 7. 9. 2004, 8. 9. 2004 [4. 5. 2015]. <http://www.w3.org/TR/speech-synthesis/>

[8] *JVoiceXML - The Open Source VoiceXML Interpreter* [online]. 4. 5. 2015 [4. 5. 2015]. <http://jvoicexml.sourceforge.net/>.

[9] *The XMPP Standards Foundation* [online]. 4. 5. 2015 [4. 5. 2015]. <http://xmpp.org/>.

[10] *Hicks Matt.Facebook Chat Now Available Everywhere [online]. 10. 2. 2010, 4. 5. 2015 [4. 5. 2015]. <https://www.facebook.com/notes/facebook/facebook-chat-now-available-everywhere/297991732130>.*

[11] W3 consorcium. Extensible Markup Language (XML) [online]. 20. 4. 2015 [4. 5. 2015]. <http://www.w3.org/XML/>.

[12] W3 consorcium. DOM Parsing and Serialization [online]. 4. 5. 2015 [4. 5. 2015]. <https://dvcs.w3.org/hg/innerhtml/raw-file/tip/index.html>.

[13] SAX [online]. 4. 5. 2015 [4. 5. 2015]. <http://www.saxproject.org/>.

[14] List of XML markup languages - Wikipedia, the free encyclopedia[online]. 29.4.2015 [5. 5. 2015]. <http://en.wikipedia.org/wiki/List_of_XML_markup _languages>.

[15] W3 consorcium. Voice Browser Activity Statement [online]. 15. 4. 2015 [4. 5. 2015]. <http://www.w3.org/Voice/Activity.html>

[16] W3 consorcium. RDF - Semantic Web Standards [online]. 23. 3. 2015 [4. 5. 2015]. <http://www.w3.org/RDF/>.

[17] DCMI Home: Dublin Core® Metadata Initiative (DCMI) [online]. 4. 5. 2015 [4. 5. 2015]. <http://dublincore.org/>.

[18] MARC STANDARDS (Network Development and MARC Standards Office, Library of Congress) [online]. 4. 3. 2015 [4. 5. 2015]. <http://www.loc.gov/marc/>.

[19] JNDI Overview[online]. 6. 5. 2015 [6. 5. 2015]. <http://www.oracle.com/technetwork/java/overview-142035.html>.

[20] SAINT-ANDRE Peter. XEP-0001: XMPP Extension Protocols[online]. 8. 4. 2014 [6. 5. 2015]. <http://xmpp.org/extensions/xep-0001.html>.

[21] *MULDOWNEY Thomas. XEP-0095: Stream Initiation*[online]. *13. 4. 2004 [7. 5. 2015]. <http://www.xmpp.org/extensions/xep-0095.html>.*

[22] *MULDOWNEY Thomas. XEP-0096: SI File Transfer*[online]. *13. 4. 2004 [7. 5. 2015]. <http://www.xmpp.org/extensions/xep-0096.html>.*

[23] *SAINT-ANDRE Peter. XEP-0085: Chat State Notifications*[online]. *23. 9. 2009 [7. 5. 2015]. <http://xmpp.org/extensions/xep-0085.html>.*

[24] *Ignite Realtime: Smack API [online]. 4. 5. 2015 [4. 5. 2015]. <https://www.igniterealtime.org/projects/smack/>.*

[25] *Ignite Realtime: Openfire Server [online]. 4. 5. 2015 [4. 5. 2015]. <http://www.igniterealtime.org/projects/openfire/>.*

[26] *DMTF. Open Virtualization Format Specification*[online]. *1. 12. 2010 [6. 5. 2015]. <http://dmtf.org/sites/default/files/standards/documents/DSP0243_1.1.0.pdf>.*

[27] *Oracle VM VirtualBox [online]. 4. 5. 2015 [4. 5. 2015]. <https://www.virtualbox.org/>.*

[28] *Overview (Smack 4.1.0 API) [online]. 29. 3. 2015 [4. 5. 2015]. <https://www.igniterealtime.org/builds/smack/docs/latest/javadoc/>.*

Příloha A

Návod

Tato příloha se zaobírá nasazením a zprovozněním řešení, jež vzniklo jako výsledek práce.

A.1 Instalace

Pro nasazení platformy je potřeba zprovoznit tři aplikace: JVoiceXML, OpenFire a CommunicationModule. Na jejich zprovoznění je potřeba mít prostředí vybaveno několika nástroji:

- systém na správu verzí SubVersion (svn)
- nástroj na automatické sestavování ANT

A.1.1 Instalace JVoiceXML

Pro správnou instalaci JVoiceXML je potřeba udělat několik kroků. Nejdříve je potřeba pomocí nástroje SubVersion stáhnout ze serveru https://sourceforge.net zdrojové kódy z aktuální větve JVoiceXML. To lze snadno použitím příkazu:

```
svn checkout svn://svn.code.sf.net/p/jvoicexml/code/trunk
jvoicexml-code
```

Jakmile je JVoiceXML stažené je potřeba jej nakonfigurovat. Pokud prostředí využívá Javy verze 1.8. je potřeba nahradit spring-framework novějšími verzemi, aby šlo JVoiceXML sestavit. Pokud je využito Javy verze 1.7., systém by neměl mít problém se sestavením.

Dalším krokem je nakonfigurování implementační platformy. Pro nastavení textové platformy je potřeba upravit konfigurační soubor:

```
jvoicexml-code/org.jvoicexml/config-props/platforms.xml
```

V tomto souboru je potřeba odkomentovat textovou implementaci.

Dále je potřeba vygenerovat konfigurační soubory. Toho je dosaženo spuštěním příkazu *ant -f configure.xml* ve složce org.jvoicexml/.

Následovně je potřeba ve složce config/ upravit soubor jvxml-jndi-0-7.xml, kde je potřeba nahradit textový řetězec uvnitř elementu <repozitory> řetězcem *text*.

Poté je JVoiceXML připraveno k použití.

A.1.2 Instalace serveru OpenFire

Pro instalaci serveru OpenFire je potřeba postupovat podle návodu pro zvolenou platformu. Návod je k nalezení na stránkách:

> https://www.igniterealtime.org/builds/openfire/docs
> /latest/documentation/install-guide.html

Po nainstalování serveru je potřeba jej nastavit. Nastavení se provádí ve webovém rozhraní na adrese localhost:9090/. Zde záleží na preferencích, jakou databázi využijete.

A.1.3 Instalace CommunicationModule

Pro zprovoznění komunikačního modulu je potřeba několik kroků. Nejdříve je potřeb připravit pro komunikační modul složku. Do vybrané složky je potřeba umístit CommunicationModule-1.0.jar a skript prepare.sh, jež jsou přílohou této práce. Rozbalení poté probíhá zavoláním skriptu prepare.sh, jenž vytvoří další skript run.sh, složku xml a konfigurační soubory, které komunikační modul využívá.

Pokud se rozhodnete složku jvoicexml-code umístit do stejné složky jako CommunicationModule-1.0.jar, je možné spustit obě aplikace zavoláním skriptu run.sh. Pokud se složka jvoicexml-code nenachází ve stejné složce jako .jar soubor komunikačního modulu, je potřeba spustit ručně JVoiceXML příkazem *ant run* ve složce /path/to/jvoicexml-code/
org.jvoicexml/ a poté spustit komunikační modul příkazem *java -Djava.security.policy=jvoicexml.p*
-jar CommunicationModule-1.0.jar. Pro oba dva způsoby je zapotřebí mít spuštěny a nakonfigurovaný server OpenFire na stejném zařízení.

A.2 Přidávání aplikací

Přidávání aplikací probíhá vzdáleně přes rozhraní XMPP. Je potřeba vybrat VoiceXML dokument, který bude vstupním dokumentem. Je možné předat buďto celou VoiceXML aplikaci, pokud se skládá pouze z VoiceXML a SRGS dokumentů, nebo pouze úvodní VoiceXML dokument aplikace, či pouze URL k veřejně přístupnému VoiceXML dokumentu. Pro první dvě možnosti je potřeba uživateli admin@doména.-XMPP.serveru odeslat pomocí XMPP klienta soubory. Uživatel admin automaticky přijímá žádost o přidání do roasteru i poslané soubory.

Následně je potřeba uvědomit komunikační server, na jaké adrese má aplikace reagovat. To je možné v případě prvních dvou možností zasláním uživateli admin@doména.XMPP.serveru zprávy ve tvaru:

> username "zvolené jméno" passwd "heslo k účtu na XMPP serveru" vxmlname "jméno úvodního VoiceXML dokumentu"

V třetím případě pak pomocí:

> username "zvolené jméno passwd "heslo k účtu na XMPP serveru" url "url úvodního VoiceXML dokumentu"

Pro vyzkoušení platformy můžete využít testových aplikací. Aplikace Quiz lze přidat použitím první metody a jméno VoiceXML dokumentu je Quiz.vxml. Pomocí druhé metody je možné přidat aplikaci TodoList, jejíž URL je:

> todolist-jakubpeschel.rhcloud.com/Greeting.vxml

A.3 Komunikace

Komunikace s aplikacemi probíhá kontaktováním uživatele s JID "jménoAplikace"@doména.XMPP.serveru skrze XMPP klienta. Aplikace reaguje ihned po odeslání první zprávy, kterou ignoruje.

Komunikaci je možné vyzkoušet buď s předpřipraveným uživatelem jarda@domena.XMPP.serveru, obsahující aplikaci asistivní galerie, či s vlastně vytvořenými Agenty pro Quiz, či TodoList.

Příloha B

Gramatika pro rozhodování odpovědi

Ukázka B.1: Gramatika pro rozhodování odpovědi

```
<grammar mode="voice"
root="volby"
scope="document"
xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xml:lang="en-US"
version="1.0">
<rule id="volby">
<one-of>
<item>
<one-of>
<item>a</item>
<item>A</item>
<item>ačko</item>
<item>Ačko</item>
</one-of>
<tag>false</tag>
</item>
<item>
<one-of>
<item>b</item>
<item>B</item>
<item>béčko</item>
<item>Béčko</item>
</one-of>
```

```
<tag>true</tag>
</item>
<item>
<one-of>
<item>c</item>
<item>C</item>
<item>céčko</item>
<item>Céčko</item>
</one-of>
<tag>false</tag>
</item>
<item>
<one-of>
<item>d</item>
<item>D</item>
<item>déčko</item>
<item>Déčko</item>
</one-of>
<tag>false</tag>
</item>
<item>
<one-of>
<item>konec</item>
<item>ukonči</item>
<item>odejít</item>
</one-of>
<tag>exit</tag>
</item>
<item>
<one-of>
<item>nápověda</item>
<item>pomoc</item>
</one-of>
<tag>help</tag>
</item>
</one-of>
</rule>
```

</grammar>

Příloha C

Seznam možných příkazů pro aplikaci TodoList

Pravidla gramatiky Null.grxml určují všechny možné vstupy na volné otázky.

```
<?xml version= "1.0"?>
<grammar version="1.0"
mode="voice"
tag-format="semantics/1.0-literals"
xml:lang="cs"
root="all"
xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd">

<rule id="infinitiv" scope="public">
<one-of>
<item>dělat</item>
<item>muset</item>
<item>připrvit</item>
<item>upravit</item>
<item>psát</item>
<item>přečíst</item>
<item>vysát</item>
<item>umít</item>
<item>obhájit</item>
<one-of/>
</rule>
```

```
<rule id="podmět" scope="public">
<one-of>
<item>Kuba</item>
<item>Jirka</item>
<item>User</item>
<item>Uživatel</item>
<item>pes</item>
<item>kocka</item>
<item>bakalářka</item>
<item>gauč</item>
<item>nádobí</item>
<one-of/>
</rule>

<rule id="all" scope="public">
<one-of>
<item>
<ruleref uri="#infinitiv"/> <ruleref uri="#podmět"/>
<item/>
<item>
<ruleref uri="#infinitiv"/> <ruleref uri="#podmět"/>\\ a <ruleref uri="
<item/>
<item>
<ruleref uri="#infinitiv"/> <ruleref uri="#podmět"/>\\ a <ruleref uri="
<item/>
<item> <ruleref uri="#podmět"/> <item/>
</one-of>
</rule>
</grammar>
```

Jak je možné vyčíst z gramatiky, volné vstupy se mohou skládat z infinitivu a podmětu, infinitivu a dvou podmětů rozdělených spojkou a, či podmětu samotného. Pro jméno a heslo je možné využívat podmětů.