Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Martin Vejman

Development of an English public transport information dialogue system

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	Mgr. Ing. Filip Jurčíček Ph.D.
Study programme:	Informatics
Specialization:	Theoretical Computer Science

Prague 2015

Dedication.

Název práce: Development of an English public transport information dialogue system

Autor: Martin Vejman

Katedra: Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: Mgr. Ing. Filip Jurčíček, Ph.D.

Abstrakt:

Klíčová slova:

Title: Development of an English public transport information dialogue system

Author: Martin Vejman

Department: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Ing. Filip Jurčíček, Ph.D.

Abstract: In this paper we present a telephone-based spoken dialogue system developed within Alex dialogue system framework in python. The dialogue system provides public transport information in New York and it gets involved in the MTA App Quest challenge. Crowdsourcing platform Crowdflower is used for collecting speech data and audio transcriptions. Collected data are furnished for training a language model and building a Kaldi decoder. Crowdflower is further used for comparison of Google Speech API and Kaldi ASR. It is shown that Kaldi recognizer achieves better results in restricted public transport domain.

Keywords: spoken dialogue system, crowdsourcing

Contents

Introduction

Providing information to consumers is a common task usually solved by implementing a web page or a mobile application. Spoken dialogue systems on the other hand, are suitable for fulfilling such task while simplifying the access to the information. Spoken dialogue systems allow people to interact with a computer the most natural way, by voice. They are intuitive, direct and hands-free, which renders an opportunity for deployment in many fields. There is no need to find a mobile device and enter queries into some puzzling interface, it is enough to simply say the words. Despite this, it may serve as a valuable asset for obtaining information for the elderly and especially for blind people and people with visual impairment.

However, it is very difficult to implement a dialogue system from ground up. Fortunately The Alex dialogue system platform[1] (ASDF) incorporates all of the key components needed for performing the task. We decided to take the advantage of ASDF and implement a spoken dialogue system in English for providing public transport information (PTI) in New York. New York, the city with the highest ridership in the United States and with one of the most extensive subway systems in the world, presents a formidable challenge. The English language benefits from wider range of use and larger competition to be compared with. This enables us to build better systems with greater potential.

Automatic speech recognition (ASR) is not yet at the point where the meaning of every utterance could be reliably identified by a computer.[2] Although with a restricted domain, it is possible to achieve substantially better results utilizing a trained speech recognizer. A crowdsourcing platform CrowdFlower will be used to evaluate our PTI solution and to collect speech data. The collected speech data will be facilitated to train a Kaldi speech recognizer and it will be compared with the Google ASR. The comparison measures will be based on the Crowdflower contributor's report.

Our PTI solution will be a useful showcase of the ASDF and will further contribute to collect speech data for improving the quality of speech recognizers. ASK: should we mention MTA here?

The outcomes of this thesis include the public transport information telephone-based dialogue system for New York, TODO: and the other stuff.

The paper is organized as follows. In the first chapter we will look over the used technologies. In the second chapter we will discuss the implementation in its principles by each component. The third chapter will cover the features of implemented solution and display its capabilities. The fourth chapter will describe the workflows associated with creating such dialogue system. Training Language model and building Kaldi decoder will also be covered by this chapter. In the fifth chapter we will draw conclusions and summarize the findings encountered in the process.

1. Technologies used

In this chapter we will introduce main technologies that were involved in creating our dialogue system.

1.1 Alex spoken dialogue framework

The Alex Dialogue System Framework[1] (ADSF) serves for utilizing research in the development of spoken dialogue systems. It is maintained by the dialogue systems group at UFAL [2], the Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University in Prague. And it is written in Python.

The ADSF consists of baseline components for assembling spoken dialogue systems. There are tools for processing logs and evaluating spoken dialogue systems. These tools can be used for audio transcriptions or semantic annotation for example. A small set of example dialogue system implementations for different domains is also present.

There is a working Public Transport Information (PTI) [7] for Prague public transport and the Czech Republic transport network in Czech language. Our solution is based on the Czech version. However, switching to English renders a challenge emerging from nation, culture and speech habit differences. It brings the advantage of having more versatile system deployable in different cities or countries just by changing a knowledge base. Collecting English data is also important for creating better models that can be facilitated by other applications within ASDF.

Automatic Speech Recognition

Automatic Speech Recognition (ASR) transforms speech into text. Many applications already use ASR technology as an interface between human and a computer, although it is not yet capable of understanding all speech in any environment. Many factors influence perception of voice.

Acoustic conditions, voice differences, distance from the recording device, heavy accent, even voice emphasis, these are few of the issues versatile ASR has to cope with. Very good overall performance delivers cloud-based speech recognition API by Google which can be utilized withing the ASDF. Achieving better quality requires great many hours of transcribed text.

However, when we restrict the recognition scope to a specific domain, the amount of words the recognizer needs to handle becomes quite limited. There is only so many expressions that can be used in a common conversation about particular subject. With a recognizer trained on narrower domain, better results can be achieved.

Kaldi is an open-source [3] toolkit for speech recognition based on finite state

[1]`https://github.com/UFAL-DSG/alex`

[2]`https://ufal.mff.cuni.cz/grants/vystadial`

[3]Apache License 2.0

transducers. We use python wrapper Pykaldi[4] within the ADSF for building a Kaldi decoder and effectively deploying trained ASR. Kaldi decoder requires statistical models – an acoustic model (AM) and a language model (LM). The AM is trained within the department [3]. It defines probabilities of acoustic features for a given word. The LM is more domain specific as it refines probabilities of a word being recognized. We use ASDF scripts that utilize the SRI Language Modeling Toolkit[5] (SRILM) for training LMs. The process of training LM and testing Kaldi will be expanded upon in chapter 4 on page 14.

Voice Activity Detection

We need to be able to determine the end of the utterance for the computer to take turn and respond. This role is performed by the Voice Activity Detection (VAD). VAD cuts speech into sentences and sends it to ASR for processing. It separates noise and silence from the speech.

Text To Speech

Text To Speech (TTS) makes an instantaneous impression as this is the first and in most cases the only output end user is able to perceive. The ASDF supports multiple TTS alternatives, Google, Flite, SpeechTech and VoiceRSS. We have utilized VoiceRSS[6], the free online service. The VoiceRSS API requires API key which is limited to per day requests. With the ASDF caching most of the web requests it suffices our intents.

VoIP interface

Our spoken dialogue system communicates with users over a phone. The ASDF exploits a modified version of communicating library PJSIP[7] for implementing VoIP applications. There is no need for registering a telephone number, for running a dialogue system it is suffice to enter SIP account details. SIP account can be freely registered at numerous providers.

For accepting incoming calls from USA, a toll-free number was provided by the department.

1.2 Crowdsourcing

Crowdsourcing is a method for acquiring data by delegating work to a community of people. In particular online communities tend to be employed for their convenience. By dividing tasks into smaller independent parts, one can eliminate the need for expert workers and therefore reduce costs associated with acquisition of the coveted data. In some cases the cost savings can be a tenfold of what in-house solution may provide as mentioned in [4]. However, this method can barely achieve the quality or accuracy of expert workers.

[4]`https://github.com/UFAL-DSG/pykaldi`
[5]`http://www.speech.sri.com/projects/srilm/`
[6]`http://www.voicerss.org/`
[7]`https://github.com/UFAL-DSG/pjsip`

Collecting speech data for training ASR models in English is easier with the help of crowdsourcing. There are several crowdsourcing platforms connecting workers with work requesters such as Amazon Mechanical Turk[8], Samasource[9], CrowdFlower[10] and many more.

Samasource is a non-profit organization with a noble cause of lifting people out of poverty through digital work. It does not, however, meet our need of employing native English speakers. While Amazon Mechanical Turk would match our requirements, it is no longer available for non-US requesters. With Crowdflower, we are able to implement a custom solution directly within the platform.

Crowdflower has mechanisms such as monitoring answer distributions and computing confidence score for maintaining quality of the output data. They claim great amount of contributor force which promises prompt job resolution. The platform contains comprehensible templates for common tasks. It features a web interface for building a custom job from scratch with a sensible support and demonstrative examples, too.

1.3 Deployment

Running a real-time dialogue system can claim considerable amount of system resources. All of the components of the dialogue system in the ASDF are separate processes. Also the static knowledge-base may outgrow ordinary computer's memory. Hence, it is only right to employ multi-processor machine with sufficient amount of memory.

1.3.1 Docker

Docker[11] is a platform for rapid deployment of applications. It contains a packaging tool and a lightweight runtime. An application wrapped in docker is easily portable to any laptop or virtual machine (VM). The chain of events leading from discovering a flaw and changing source code, to deploying compiled dialogue system can be excessively accelerated with docker.

Dockerfile is a specification file used for automating docker image builds. It allows to specify a sequence of instructions executed on a base image in order to create a new one. Built docker image with the dialogue system can be executed in an isolated container.

The Alex dialogue system framework (ADSF) is now using docker which is very useful for resolving dependencies. Our Dockerfile is based on the base ASDF Dockerfile with instructions for downloading newest models and knowledge-base. There is a `-i` flag for mounting any directory into a docker container.

1.3.2 MetaCentrum

In order to provide service for more than one caller at a time, we need to have multiple instances of our dialogue system. Computing and storage resources of

[8]https://www.mturk.com/mturk/welcome
[9]http://samasource.org/
[10]http://www.crowdflower.com/
[11]https://www.docker.com/

MetaCentrum[12] can be used freely by all students of academic institutions in the Czech Republic. Various VMs were deployed on MetaCentrum for our dialogue system groundwork.

For each instance of MetaCentrum VM, configuration with 4 processors and 16GB was used. Half of the memory would be sufficient, however, the same VMs were employed for training language models which involves excessive memory usage.

TODO: říct kolik VMs jsme použili na MTA, na CF a na ostatní věci

[12] `http://www.metacentrum.cz/cs/`

2. Implementation - Public Transport Information for New York

In this chapter we will go through each component of our dialogue system we needed to arrange. The principle will be described along with relevant instruments. All of these components are domain specific descendants of domain independent components of the ASDF.

First we will cover keyword database and matching words against the input, than we will describe changing states in our dialogue system. Last we will explain how the state is translated to the output which will conclude in an outline of utterance processing.

2.1 Spoken Language Understanding

To be able to process, evaluate and respond to user requests, semantic meaning needs to be extracted from utterances. This is realized via Spoken Language Understanding (SLU), which uses a vast static keyword database for analyzing words and phrases of each utterance. Being able to handle such semantic representation makes it possible to change state of the dialogue system. There are different approaches for SLU development. There are SLU techniques based on statistical models learned from data. We have, however, implemented handcrafted SLU, based on simple keyword rules. Both approaches are supported by the ADSF as demonstrated in [5].

Keyword database

Public transport information domain demands the ability to respond to two major constraint queries - location and time. The time and supplementary keywords can be defined explicitly or generated by a simple script. However, the location data are specific for the region we decided to cover and therefore must be gathered.

We ultimately need just the name of the waypoint for keyword matching. However, the stop or city names might be ambiguous which is why we need to keep further knowledge about geographic information and more general area.

Location data terminals

The types of waypoints are streets, stops, boroughs, cities and states. All of these location categories are listed in a separate file for the convenience of adding new or updating existing entries. It is obvious that the borough list will be very narrow, may be even unnecessary because there is only five boroughs in New York. But the idea is to be able to distinguish between streets and stops with the same name that are very likely to appear within the same city. If we decided to expand the system to cover Los Angeles region for example, we might need to add not only LA boroughs but to specify a finer administrative division altogether.

In terms of the stops, we collected the latest data from MTA[1], PATH[2], NJ Transit[3], NY Waterway[4] and Amtrak[5] for long-distance trips. Most of the companies are providing their schedules for developers in a unified format. The General Transit Feed Specification (GTFS) defines a common format for public transportation schedules and associated geographic information.[6]

We could adopt the GTFS format which would be very convenient for updating. However, some of the datasets were not strictly following the GTFS making it unfeasible to work with. Missing values, overflowing columns or disunited expressions occurred infrequently, nevertheless throughout notable portion of the data. The benefit of an easy update and access to additional information did not outweigh the shortcomings encountered.

We opted for a simple format that takes only the most important features into account. Selecting fewer columns makes it easier to add places that are not available in the GTFS. This includes popular sites as well as few smaller transport companies commuting between tens of terminals that we also included into our database.

In addition to official stops, we added over a hundred of the most popular sites in New York from various top n lists. Those can be used as good reference points in everyday commutes so that for example the following sentence can be handled.

"From Chrysler building to Columbus Park.".

We used Google Geolocation API[7], to obtain longitude, latitude and borough for each popular site.

The obtained data in raw form can not be used for keyword matching. As opposed to the Czech language, there is no necessity to take inflection into account, however, there is a number of ways to express a stop or a street. Stops in particular, mostly called after an intersections, can be unfolded and expressed in different order and coupled with a different conjunction.

For example the stop `1 Av/E 111 St` can be expressed as

"east hundred eleventh street first avenue".
"east hundred eleventh street at first avenue".
"east hundred eleventh street and first avenue".

⋮

"east one hundred eleventh street at first avenue".

⋮

"first avenue and east hundred and eleventh street"

⋮

Also the data contain numbers and unpronounceable characters like parenthesis, slashes, dashes and also abbreviations that are not unified. For example

[1]Metropolitan Transportation Authority - `http://www.mta.info/`
[2]Port Authority Trans Hudson - `http://www.panynj.gov/`
[3]New Jersey Transit - `http://www.njtransit.com/`
[4]New York Waterway - `http://www.nywaterway.com/`
[5]The National Railroad Passenger Corporation - `http://www.amtrak.com/home`
[6]GTFS - `https://developers.google.com/transit/gtfs/`
[7]`https://developers.google.com/maps/documentation/business/geolocation/`

the `St` can mean both *street* and *saint* and to continue, the word *expressway* is abbreviated by `ep`, `ex`, `exp`, `expy` and `expwy`. Thus for each category we have a separate file with possible forms generated by an expansion script.

Dialogue Act Scheme

Intents of the user as well as actions of the spoken dialogue system are represented by Dialog Acts (DA). They consist of one or more Dialogue Act Items (DAI) that are elementary semantic information units. DAIs are defined by a type, slot name and slot value. The slot name and value are domain specific and further define the meaning. In our case slot names may refer to a place for instance. Exemplary Dialogue Act is shown at 2.1. We can see that the *When does* and *leave* correspond with the request DAI and that the inform is gathered from the word *bus* in the sentence.

Utterance	*When does the bus leave?*
Dialogue Act	`request(departure_time)&inform(vehicle="bus")`

Table 2.1: Example of semantic notation of an utterance

Sometimes it is not clear how an utterance should be transformed into DA due to unknown context or ASR lapse. The ADSF contains a Dialogue Act Confusion Network that deals with this issue. The confusion network stores a probability for each DAI and it presents the most likely DA based on the probability distribution of DAIs.

A confusion network is best utilized when processing ASR n-best hypothesis and using statistical SLU.

Handcrafted SLU

Handcrafted SLU works only with 1-best hypothesis from the ASR. After an utterance is passed into our SLU, it is matched against class labeled database keywords and an abstract utterance marked with labels is produced. Each label has a special parsing procedure that yields Dialogue Acts into the dialogue act confusion network.

A search for semantic meaning takes place in designated routines, the following class labels are handled separately.

- `NUMBER` - Parsing hour and minute values and time fractions.
- `PLACE` - Parsing waypoints from stop, street, borough, city and state values.
- `TIME` - Absolute and relative time periods matching.
- `TASK` - Conversation topic which is either weather or finding connection.
- `VEHICLE` - Preferred means of transport matching.

Due to the iconic Manhattan street grid, people in New York are likely to know their position based on the street and avenue names, which are commonly numbers. They may not know the closest bus or subway station. Therefore we decided to support streets as valid input for finding connections. The idea is to let users specify an intersections rather than stops. Stops however, make more accurate search queries possible, because we have access to the latitude and longitude values mentioned earlier at 2.1. The ambiguity of streets and stops is not negligible, hence boroughs are also parsed as waypoint entries.

Further series of matching steps take place after those routines. We search for keywords and phrases in the whole utterance regardless of the context and add more DAIs to the dialogue act confusion network based on simple if-else rules. It handles particular utterances for courtesy, greeting, acknowledgement as well as requests and notifications about transport query restrictions. Also DAIs from non-speech events like silence or noise DAIs are extracted.

2.2 Dialogue Manager

A Dialogue Manager (DM) is a component responsible for processing and changing dialogue states in order to take appropriate actions in response to the user's query. The history of the dialogue and inner states are recorded for better comprehension of current query.

Ontology

Ontology contains a static domain knowledge information that can be used for better understanding relations between entities. It defines dialogue act item slot types and values from database mentioned at **??** and relationships between them. This allows DM to gain more relevant information for example by context resolution.

In addition, it provides relations between locations for discovering compatibility conflicts and for implicit value inference. The compatibility lists are bidirectional and concern street-borough, stop-borough and city-state relations.

Handcrafted DM

Our handcrafted DM extracts facts from the combination of its inner states, history of the dialogue and the DAI probability distribution taken over from the dialogue act confusion network from SLU. In an if-else rule block it selects a subroutine for deciding what will be the next action taken.

Simple responses to elementary facts are among the first served by the rule block. Those include actions for greetings, repetition of the last system utterance, a context specific help, reseting the system or a back-off action. In case the user's turn yields no change since the last time, or the input from ASR was invalid, the DM executes a back-off action, which is randomly selected from providing help, repeating the last utterance, silence and an act for saying it simply did not understand. The reset action, for example, can be used when user asks about two unrelated topics, the DM mistakenly combines facts from both and produces unexpected responses. TODO: leave this to the feature chapter and add an example for this behavior

2.3 Natural Language Generation

Natural Language Generation (NLG) component transforms inner states of the dialogue system into readable text form. Having a limited domain, we are able to cover NLG by a template dictionary that has entries for each dialogue act item and combinations of some dialogue act items. Seamless communication can be achieved by constructing adequate NLG templates. The slot value of each dialogue act item is treated as a variable that can be inserted in the translated sentence. An example of NLG translation is displayed at 2.2. It is evident how the slot value *Broadway* is injected into the template and also that the time value is translated to word representation.

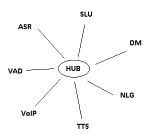
Dialogue Act	`inform(to_stop="Broadway")&inform(arrival_time="04:26:PM")`
NLG template	`inform(to_stop={to_stop})&inform(arrival_time={arrival_time}):`
	`"It arrives at {to_stop} at {arrival_time}."`
NLG output	*It arrives at Broadway at four twenty six P M.*

Table 2.2: Translation example of dialogue act to sentence by Natural Language Generation component

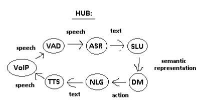
There may be multiple expressions defined for each dialogue act, which is useful for making overused dialogue acts, such as greetings, seem more natural and less robotic. The NLG templates can be overlapping and proper translation rule has to be searched for. The search proceeds from exact to general and from long to short sequences of dialogue act items.

2.4 Main System Hub

The central hub gives the dialogue system modularity. All of the components are connected together via main hub in a star-like shape shown in figure 2.1a. Each component runs as a separate process and the hub essentially chains them via standard stream pipelines as shown in 2.1b and coordinates their continuity.



(a) Hub configuration



(b) Component chain

Figure 2.1: On the left there is a typical star-like shape configuration of a dialogue system components and the right figure shows the inner component chain of the central hub.

3. Features - Public Transport Information for New York

In this chapter we will describe what the PTIEN is capable of. The features are derived from the DM capabilities, which is a brain of the dialogue system, but each component has to oblige. When we describe how the DS is capable of responding to a concrete request, the SLU has to extract semantics from the text input, DM has to what to do and NLG has to generate text from the dialogue acts.

3.1 Features of PTIEN

In this section we will describe the functionality of PTIEN as a whole.

3.1.1 Accompanying sentences

tabulka: pozdravy, -
ticho -
rozloučení -
nerozuměli jsme -
help requested - context help provision
poděkování - reakce a zeptání se na další úkoly
restart - restart dialogu

to že jsme nerozuměli může být buť tím a nebo tím, context help vychází z toho, o čem naposledy člověk hovořil, vybere se náhodně nějaká z vět o tématu, třeba že se můžou zepat na počasí v jejich rodném městě, nebo pokdud se bavili o PTI, tak : můžeš se třeba zeptat kolik to má přestupů...

3.1.2 tasks

3.1.3 Providing current time

The user can make better route selection decisions based on what time it is. This is why it is important for the system to be able to provide current time. As opposed to the Czech Republic, there are places at different time zones in the United States. Therefore we decided to support queries specifying a city or a state for providing more accurate, localized time. The state is enough information to receive a location specific time, however specifying a city is more accurate as some states occupy two timezones. If only a city is specified, the DM will respond with a dialogue act requesting the state name, unless the city is not ambiguous. In that case it will infer the state from the ontology.

The time zone data are received from the Google Time Zone API[1]. With inaccessible API it returns apology act and default computer time for New York, which is Eastern Time Zone. Time zone names could be included in the ontology

[1]`https://developers.google.com/maps/documentation/timezone/`

as additional data, however this way, there is no need to keep track of the daylight saving offsets used in some sates.

Weather forecast

Following the example of utilizing weather forecast in the Czech PTI, we have implemented an English version. It is comforting for the users to be able to obtain weather information at once. OpenWeatherMap API[2] is used for collecting weather data. The DM takes the action based on gathered location data much like at providing current time 3.1.3. In addition to city and state, the weather forecast can be specified by time. The output dialogue act contains a range of temperatures as well as weather condition for given restrictions.

Finding connection

The prime asset of our dialogue system is the ability to effectively respond to transport connection requests. The key restrictions are the location from and where to travel, which is either a city, stop or an intersection of two streets. The ambiguities of waypoints are resolved in similar manner as in time zone or weather forecast case. The DM tries to infer city or borough and returns a request for waypoint specification only if an ambiguity is found. If according to ontology, the specified street or stop is not located in a given borough or city, an apology dialogue act is produced. User can further specify the departure or arrival time both in absolute and relative forms, a preferred vehicle and the maximum number of transfers. The nature of the DM allows to specify these restrictions at once or one by one which makes for better utilization of the dialogue system. If any of the key restrictions are missing, the DM issues a dialogue act demanding appropriate additional information. After the DM responded with a route proposition, the user can either further specify his query or ask about the connection attributes. This includes the origin, destination, arrival and departure times, number of transfers and the duration of the trip. The trips in New York can be very long, hence the sequence of instructions is exhaustive. This is why we have added the option to ask about the length of the trip, which tells not only the mileage, but also the number of stops to pass through before each transfer. The user can also browse through offered connections back and forth by requesting next, previous or explicit number of the connection in case the current route does not satisfy user's needs. For a given time, there are four alternative connections on offer by default.

We use the Google Directions API[3] to acquire connection data. For simple from-to queries we use free API accessible through HTTP and we only use API key for more restrictive queries with preferred vehicle and transfer count limitations. This maximizes the utilization of the key before it reaches a monthly fee threshold. The transfer count is not directly mapped to an API request, it is rather computed from the API response. When no connection suites the restrictions, an apology dialogue act is produced.

TODO: context resolution, compatibility conflict example

[2]`http://openweathermap.org/api`

[3]`https://developers.google.com/maps/documentation/directions/`

4. Workflows - Development processes

This chapter is concerned with the process of several procedures repeatedly used while developing spoken dialogue system providing public transport information in New York. Very similar approaches might be taken for the development of dialogue systems in different domains.

4.1 Creating CrowdFlower Job

Assembling a Crowdflower job can be realized through one of many templates for ordinary tasks such as various data analysis, entity annotation, categorization, comparison, revision and many more.Custom and more sophisticated tasks can be carried out from scratch. It is desirable that the tasks are as simple as possible to eliminate errors resulting from the lack of knowledge or misinterpretation.

Crowdflower provides a web interface for work requesters to edit the task by CrowdFlower Markup Language (CML), CSS and custom JavaScript that runs once on page load. There is a possibility to inject custom HTML code as well. CML and JavaScript are essential for leveraging Crowdflower's quality control. Both mandatory and optional input controls have to be specified with the CML.

4.1.1 Call job

We created a call job for testing operational dialogue system. Its purpose is to encourage solvers to call on a toll-free number and ask questions about the public transport in New York and to evaluate and rate the system.

To ensure the call is carried out thoroughly by the contributor, we employed a simple generator of four digit codes. This code is handed out by the dialogue system after finishing a call. It is spelled number after number three times over. In the same time, the code is registered at a validation server running on a dedicated MetaCentrum VM. Without this code it is not possible to submit a feedback form and finish the job.

This behavior of the CrowdFlower job is enforced by a CML control with a custom JavaScript validator. When contributor inserts a code to the CML control, the validator sends a request with the code to the validation server. Server compares the code with a set of registered codes from the dialogue system. Only after positive server response is acquired the validator passes. It is unnecessary to match callers identity, this is sufficient measure for enforcing the call. ASK: should we introduce a figure of the validation process to destroy the block of text? or later figure of the feedback form?

To further maximize the efficiency, the dialogue system only hands out the validation code after minimum number of turns is passed. This prevents the callers from saying *"Hello, Good bye!"* and collecting the validation code and therefore the reward without fulfilling the task.

The job web page was built as a survey job from scratch. In the premise of the job, we declare four paragraphs concerning the job.

- **Intro** - Introduction to the whole process, mentioning restrictions and remarks.
- **Instructions** - Exact procedure description, how to behave, how to end the call, how to fill the feedback form.
- **Example call** - Demonstrative dialogue between caller and our dialogue system.
- **Consent** - Legal statement concerning the data management and recording the call.

ASK: should we expand on that?

Stops between which caller wants to find a connection are quoted after the premise. Additional question about the link are urged for exploiting the dialogue system features.

A feedback form of subjective user satisfaction concludes the job page. In addition to the following question an optional field for general comments and mandatory field for the validation code are within the form.

Have you found what you were looking for?	Yes/No question
The system understood me:	range of 1 to 4 from Very poorly to Very well
The phrasing of the system's response was:	range of 1 to 4 from Very poor to Very good
The quality of the system's voice was:	range of 1 to 4 from Very poor to Very good

ASK: should this be mentioned in the results rather?

A toll-free number was used for this job. Crowdflower allows to geographically limit work force only to United States. ASK: which was important for collecting local Acoustic data? TODO: mention one call per job to ensure diversity of callers? Four VMs on MetaCentrum were dedicated to this job to serve multiple callers.

4.1.2 Transcription job

After collecting enough calls a transcription job was built from a template for audio transcriptions. For each audio track there is a radio button for marking comprehensible tracks and a field for writing transcribed text. Only instructions and data are needed for launching a transcription job.

This kind of job is very common and popular and therefore it is solved by contributors very quickly. However, the contributors differ on spelling of some words and it is absolutely crucial for the job instructions to make it perfectly clear how should the contributor write.

Data are uploaded to CrowdFlower via CSV file that contains a list of URLs with audio tracks. The default setup suggests to let each track transcribe three times for accuracy. Even more transcriptions yield from setting up dynamic judgments. However, repeated labeling is costly and may tend to move towards the in-house solution in that regard. We decided to keep multiple transcriptions, while reducing cost per transcription. The ultimate transcription is decided upon later from the job results by a custom semi-automatic script.

CrowdFlower uses test questions for separating the good transcribers from the bad. Test questions in this job are essentially manual transcriptions. We utilized a quiz mode that estimates the quality of a contributor beforehand. It is assembled from test questions and lets only trusted contributors to participate in the job.

In the instructions we defined examples of how common words should be handled and a table with symbols for incomprehensible tracks was specified. It is a good practice to let the users know the context. The contributors were more content when a list of phrases they might hear was included.In our case the list included phrases like *number of transfers*, *duration of the trip*, *weather forecast*, origin and destination stops etc. Even though some of those phrases did not appear in the exact form in the audio tracks, the evidence of improvement was observable in contributor satisfaction stats of the job within CrowdFlower.

4.2 Iterative improvement

At the beginning we had just a vague idea about how the system should behave. We had a general insight of the features from the Czech dialogue system, however we did not know what is the native way of asking for information. Therefore we made a bootstrap list of sentences with their semantic complements, all of which our dialogue system must work on.

When an operational dialogue system was achieved, we employed CrowdFlower workforce for obtaining feedback from real users. Analyzing logs was very important for discovering ways of inquiring information which we initially did not think of. The log analysis and feedback form from CrowdFlower jobs also provided an input on what features are missing or need improvement. This was an iterative process of improvement captured in an essence in the following steps.

1. Launch a CrowdFlower call job
2. Obtain logs from VMs
3. Fix flaws in:
   - **SLU** - enrich bootstrap from user turns and maintain 100% precision
   - **DM** - amend features of the dialogue system
   - **NLG** - add templates from system turns to polish rough expressions
4. Upload source code to VMs
5. Restart the dialogue systems.

ASK: would be a picture better here or both itemize?

The dialogue system on each VM is running in a docker container. Any folder can be mounted to the docker container via `-v` flag. Uploading source code to update dialogue system on VM is therefore effortless and makes the development loop very quick.

4.3 Building Kaldi ASR

For building Kaldi decoder we used Pykaldi[1] docker image containing the essential tools. It is necessary to add dependencies for ASDF if building and evaluation is intended within the platform. SRILM[2] must be installed for training language model (LM). ASK: should we even say this?

Prior to training LM, it is necessary to dump database for creating a labeled list of database entries. It is used for balancing probabilities of every database entry within its class in the LM. Finally, we need to define domain specific corpus for training the LM . Our training data consist of utterances from CrowdFlower call logs, bootstrap utterances and utterances generated by grammar.

Grammar

TODO: subsection or subsubsection?

Creating a good LM entails a good probability distribution of words in the corpus. This can be achieved naturally by collecting a lot of transcriptions. As we do not posses large number of transcriptions, we decided to bootstrap LM by generated utterances by grammar. It should produce utterances that are most likely to be used and therefore it should cover the most frequent cases.

Our grammar is written in Python and consists of the following prescriptions for simple rewriting rules.

- **Alternative** - exactly one of many

$$\begin{aligned} A^i(x_1, x_2, ..., x_n) \text{ adds } & A^i \rightarrow x_1 \\ & A^i \rightarrow x_2 \\ & \vdots \\ & A^i \rightarrow x_n \end{aligned}$$

- **Option** - either present or not

$$\begin{aligned} O^i(x) \text{ adds } & O^i \rightarrow x \\ & O^i \rightarrow \lambda \end{aligned}$$

- **Sequence** - chain of rules

$$S^i(x_1, x_2, \ldots, x_n) \text{ adds } S^i \rightarrow x_1 x_2 \ldots x_n$$

where for the i-th rewriting rule:

$$\begin{aligned} & \left\{A^i, O^i, S^i\right\} \subseteq V_N \ldots \text{nonterminals} \\ & x, x_1, x_2, \ldots x_n, \lambda \in V_T \ldots \text{terminals} \\ & n \in \mathbb{N} \end{aligned}$$

[1]`https://github.com/UFAL-DSG/pykaldi`
[2]`http://www.speech.sri.com/projects/srilm/`

TODO: ASK: we have created more complicated grammar than this - with the probabilities that can be explicitly defined, but it is just a feature, should we go on about it?

Explicit grammar can be assembled using these prescriptions which can than simply generate random utterances in desired number. An example of plain grammar can be built as follows.

```
pref_p = A('can you tell me', 'i would like to know')
pref_q = A('what is', 'what will be')
subj = A('weather', 'forecast', 'weather forecast')
period = A('for tomorrow', 'in the afternoon')
weather = S(O(pref_p), O(pref_q), 'the', subj, O(period))
```

The nonterminal `weather` yields utterances asking about the weather. Terminals can be also loaded from file, which is useful for defining alternatives for waypoints for example.

The final grammar should cover as many utterances as possible. However, it is easy to include utterances that are not used in conversation or does not make sense at all. From the example above a sentence *can you tell me the weather tomorrow* is not exactly what we wanted to include. Even though it is syntactically correct, it is not something to be used in PTI domain. This is undesirable to have in our corpus. TODO: not so bulletproof example, i want to generate some nonsense, but not too much for illustrating my point about over-generalizing

4.3.1 Building a decoder

When LM is ready, Kaldi decoder can be built. An acoustic model is needed, in our case it is downloaded from the department server.

After running build script, it can be also tested within the ASDF. Statistics are computed from a test set that was created earlier from call logs when LM was built. The test set can be also tested with the Google ASR which renders a good comparison between the two recognizers.

We occasionally used CloudASR[3] for manual testing. With CloudASR, it is very easy to deploy Kaldi ASR and access it through web interface by anyone who wants to try the decoder out by his own voice.

[3]`https://github.com/UFAL-DSG/cloud-asr`

5. Results

5.1 MTA Contest

promotional video at:
`http://challengepost.com/software/alex-information-about-public-transportation`

5.2 CrowdFlower - subjective user satisfaction

TODO: do výsledků říct kolik jobů jsme pustili a po jakých dávkách

5.2.1 Google ASR

5.2.2 Kaldi ASR

5.3 Future work

TODO: to implement MTA API which would give us the ability to support more accurate information about current connections, locations of trains etc.

TODO: implement MTA live data support which could be much more complexly used, to be able to ask how many minutes is the next train due our initial station would be cool

PTICS is focused on providing information relevant to prague integration transport. but we have to be more flexible than this. We wanted to support intersections as výchozí body, that is from fifth street and twenty second street or vice versa. We ended up supporting not only intersections but plain streets as valid input. This way one can go from fifth avenue even though it is not really an apt location.

TODO: Develop statistical SLU for robustness

5.4 Acknowledgements

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

Conclusion

This work presented the dialogue system providing public transport information for New York developed in the Alex dialogue system framework. This involved creating a handcrafted spoken language understanding, dialogue manager and natural language generator components for adopting the public transport domain. We collected static easy to update knowledge-base from various public transport providers in New York. Additionally the dialogue system supports weather queries

Crowdflower crowdsourcing platform was utilized for collecting subjective user satisfaction. The dialogue system proved to be stable and beneficial in helping everyday commuters. Furthermore it was capable of competing alongside commercial applications in MTA App Quest.

Crowdflower was also used for transcribing collected calls. Language model was trained with the combination of transcribed calls and generated data with grammar simulating user input. Consequently a Kaldi decoder was trained and compared with Google ASR. It was proven that the Kaldi ASR achieves considerably better results than the Google ASR on the public transport domain.

TODO: expand

Bibliography

[1] nothing yet

[2] nothing yet

[3] nothing yet

[4] Ipeirotis, Panagiotis G., Foster Provost, and Jing Wang., Quality management on amazon mechanical turk, In: *Proceedings of the ACM SIGKDD workshop on human computation*, ACM, 2010, pp. 64–67.

[5] Dušek, O., Plátek, O., Žilka, L., and Jurcícek, F. Alex: Bootstrapping a Spoken Dialogue System for a New Domain by Real Users, In: *15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2014, pp. 79–83.

[6] Biewald, Lukas. *CrowdFlower resource library* [online]. 2015 [cit May 5, 2015]. Available from: `http://www.crowdflower.com/overview`

[7] UFAL-DSG, *The Alex Dialogue Systems Framework - Public Transport Information*, [online]. 2015 [cit May 5, 2015]. Available from: `http://ufal.mff.cuni.cz/alex`

TODO: citace nesmí mít aktivní odkaz!

List of Tables

List of Abbreviations

Attachments