VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DIALOG SYSTEM FOR HUMAN-ROBOT COMMUNICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR’S THESIS

AUTOR PRÁCE MARK BIRGER
AUTHOR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DIALOGOVÝ SYSTÉM PRO KOMUNIKACI S REÁLNÝM ROBOTEM

DIALOG SYSTEM FOR HUMAN-ROBOT COMMUNICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARK BIRGER
AUTHOR

VEDOUCÍ PRÁCE — Doc. RNDr. PAVEL SMRŽ, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Tato práce se zabývá řešením problémů vývoje dialogového uživatelského rozhraní pro roboty. V rámci této práce byla vytvořena knihovna, určena pro rychlou implementace mluvčích dialogových rozhraní pro existující programy. Pomoci této knihovny dá se napsat popis dialogu ve speciálním značkovacím jazyků, abych kontrolovat tok řízení programu podle vstupní fráze. Značkovací jazyk umožňuje asynchronní spuštění funkci a jejich další ovládání, což je důležité pro roboticke prostředí. Tato knihovna využívá Link Grammar Parser pro zpracování přirozeného jazyků. Pomoci této knihovny byl vyvíjen a ohodnocen dialogový systém, určeny pro řízení robotů PR2.

Abstract

In this thesis a problematic of spoken dialog systems was discovered. The dialog system framework was developed for a fast implementation of spoken dialog interfaces for existing robotics software. This framework allows describing a dialog flow in special markup format, which allows scope variables manipulating and controlling a flow of general-purpose programming language software by user input phrase. Markup language is designed for asynchronous function execution and subsequent manipulations with them. It allows robot to solve tasks simultaneously. Developed framework uses Link Grammar Parser for natural language processing. With this framework was implemented a dialog system instance for PR2 robot control.

Klíčová slova

dialogový systém, komunikace člověka a robota, přirozeně uživatelské rozhraní, dialogové rozhraní, zpracování přirozeného jazyků, link grammar, PR2, robot

Keywords

dialog system, human-robot communication, natural interface, dialog interface, natural language processing, link grammar, PR2, robot

Citace

Mark Birger: Dialog System for Human-Robot Communication, bakalářská práce, Brno, FIT VUT v Brně, 2015

Dialog System for Human-Robot Communication

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana …

. . . .
Mark Birger
May 20, 2015

Contents

Chapter 1

Introduction

1.1 Motivation

Estimated personal robots market growth is 6.5 billion dollars (260%) from 2.5 billion dollars in 2015 to 9.0 billion dollars in 2025 [18]. Personal segment includes robots used for cleaning, household applications, entertainment and security. As a final good, such robot will interact with a user and domestic environments.

Spoken language is a very efficient way of communication, if both parties do understand it [8]. Programming in the broad sense is no longer a task relegated solely to professional programmers [16]. As long as robots is widely used in education, we would like to create a tool with lower barrier to entry programming of dialog interfaces.

Consumer expectations is another reason of this project. Within the quiz about robotics thematics 50% of respondents answered (37 respondents in total), that they expect personal robots at the consumer market after 10-25 years from now on and only 9.4% suppose to see the robots on the market in more than 25 years.

1.2 Goals

Dialog system is a concrete case of program with dialog user interface. The goal is to create dialog system framework, which allows to create a dialog user interface for a software.

- Dialog system should be a general domain and applicable for a domestic environment.
- This solution should fit into existing robots platforms.
- There is a need to minimize an entry barrier for dialog interfaces programming.

Also one of the goals is to make natural interface for human-robot communication. For humans, natural communication in domestic environment is a speech with high command formulation flexibility. Later dialog system equals spoken dialog system.

1.3 Outline

The rest of this thesis is organized as follows:

Chapter 2: State of the art. This chapter contains a survey of domestic environment and ethnographic approach behind this work, robotic platforms review, dialog interface basics, natural language processing basics and relevant work review.

Chapter 3: Design. This chapter defines requirements for producing a dialog system framework and contains a project draft, which includes a description language and a dialog management in the developed system.

Chapter 4: Implementation. This chapter explains how this dialog system framework was developed and what tools were used.

Chapter 5: Evaluation. In this chapter the robot framework implementation is tested, it is compared with alternative solutions and the project is reviewed by both, the user and the developer points of view.

Chapter 6: Conclusion. The chapter contains a review of this work and research results of this work.

Chapter 2

State of the art

2.1 Dialog system background

Dialog systems should be definitely be researched in the ethnographic context, because:

- By the Oxford dictionary meaning, artificial intelligence performs tasks, which normally require a human intelligence. Dialog normally require a human, so this task is a part of artificial intelligence. To create a dialog system we need to research how human acts in dialog.
- Dialog system is a Human-Robot interface. To create this interface, we need to take a notice to the human side.

Ethnographic approach

The literature in social anthropology informs us about the complex notions of a family and household and patterns of domestic interaction [19]. Due to this fact, most reliable way of designing dialog systems is based on the use-cases approach.

Natural communication

In domestic environment, human-human communications is based on speech. Speech is a vocalized form of human communication. Of course, spoken communications isn't depends only on the phrase semantic. Human use gestures and intonation to extend this meaning. Current dialog system framework do not considers non-verbal communication. The main focus is semantic processing of natural language (English language). The basic assumption for this is that a service robot which offers an interaction model that matches human language performance in terms of conveying and understanding complex meaning will be perceived as intuitive, efficient and satisfactory by its users, at least to the same extent that interaction with people can be said to have these characteristics [11].

Domestic environment

From the robot's perspective, problems that seem trivial to humans often vary wildly between instances of tasks and environments [7]. Originality it is encouraged by the society, therefore the variability of domestic environment is extremely high. Furthermore, different cultures and lifestyles affect domestic environment variability. In this work we focused on the dialog management and abstracts about this issue. Developers can solve this issue manually by using available tools and libraries.

Robot use-cases

At the start we've analyzed common use-cases of domestics robots. Main use-cases for domestic robots are: home appliances control, question answering, assistance; butler, governess, secretary functionality.

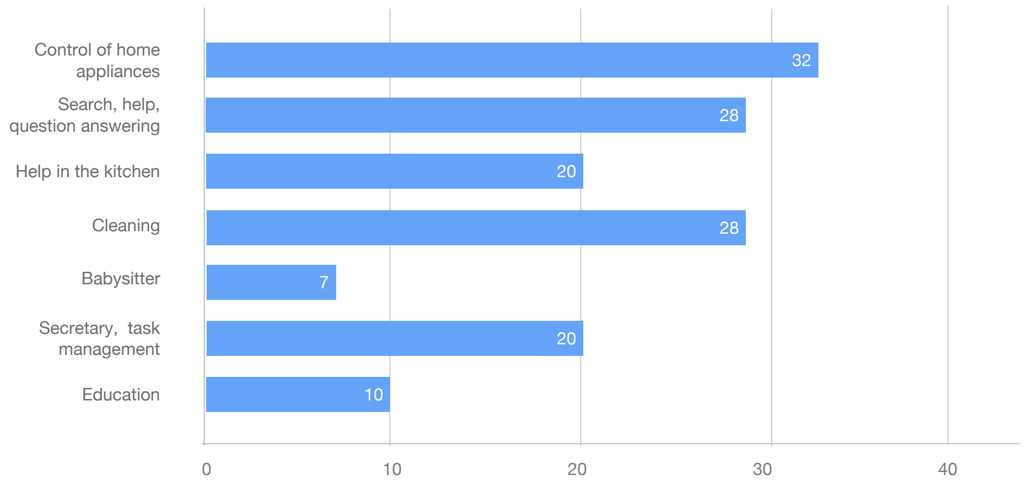


Figure 2.1: Questionnaire results. Which functionality is necessary for a domestic robot?

Figure 2.1 shows a distribution of the tasks importance from the consumer point of view.

Any command for these use-cases contains parameters, which needs to be efficiently parsed. Interface to provide a task should be similar as in human-human communication. For example, dialog system need to parse distance for movement command.

Most of activities and/or talking with a user occur simultaneously. Robot should respond to user questions even, when it moves. It is also should be possible to run two activities, which do not control common hardware at the same time.

Another important situation, which need to be solved by design is situation, when robot need to initialize a dialog. This feature allows to change dialog flow in case of environment changes. For instance, when a robot cannot reach a target, it should ask a user for help.

2.2 Specific robotic environment

This framework is designed for robots, and we need to consider robotics hardware and software.

ROS

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [7].

Willow Garage PR2

The PR2 is a mobile manipulation platform built by Willow Garage. The PR2 software system is written entirely in ROS. As such, all PR2 capabilities are available via ROS interfaces [9].

Domestic robots

A machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer is robot. “Complex series of action” meaning allow to classify some consumer electronics as robots. An example is iRobot Roomba, an autonomous vacuum cleaner [2]. This device has a traditional for consumer electronics buttons interface. Intelligent devices are balanced at the high options variability and automatic decisioning. Voice control is an appropriate tool to tune this class of devices.

2.3 Dialog system structure

Human-Robot communication

Human-Robot communication is a multidisciplinary topic about human-computer interaction and an artificial intelligence. The main aim of this discipline is to understand user expectation and modify robot behaviour for effective collaboration between human and robot. From the other side, we can teach humans how to understand robots [4]. As long as human mind is flexible, society education in robot interaction is an effective way to increase efficiency of human-robot collaboration.

Defining a dialog system

The goal of a dialog system is to interpret conversation with a human. Dialog system is wide meaning term. Simplest systems such a GUI Wizard up to complex expert systems such as Wolfram Alpha can be classified as a dialog system.

Components

Architecture of dialog systems is variable. Typical dialog system architecture is represented in the figure 2.2. Input recognizer produce a plain text (with meta information in case of gesture rezognizer). Natural language understanding unit parse data from this plain text. This data and task managers results are processed by dialog manager component. At the output semantic data, transformed to natural language by output generator, are played by output renderer [13].

Dialog manager

Dialog manager is a dialog system component, which controls a dialog flow. Usually this component receives parsed semantic data. As the output, dialog manager produces command to other parts of the dialog system, including an output generator. Dialog manager is always depends on the state (uses stateful model).

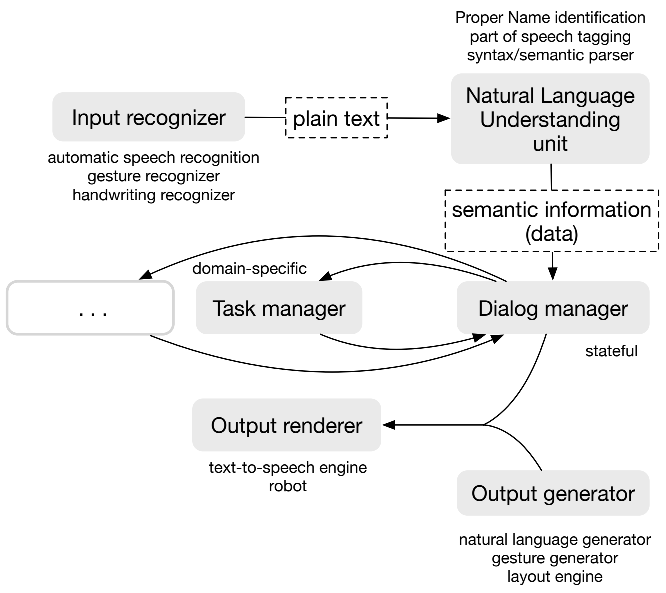


Figure 2.2: Dialog system architecture.

2.4 Natural language processing

Natural language processing (NLP) is computational linguistics topic. Technologies based on NLP are becoming increasingly widespread. For example, phones and handheld computers support predictive text and handwriting recognition; web search engines give access to information locked up in unstructured text; machine translation allows us to retrieve texts written in Chinese and read them in Spanish. By providing more natural human-machine interfaces, and more sophisticated access to stored information, language processing has come to play a central role in the multilingual information society [1].

Natural language understanding

Natural language understanding is NLP subtopic, focused at semantic data parsing.

Natural language generation

Natural language generation is the process of deliberately constructing a natural language text in order to meet specified communicative goals [10]. Layout engine is an example of simple natural language generation unit.

Overview of tools

There are a few tools for a natural language processing:

Gensim is a Python library for topic modeling, document indexing and similarity retrieval with large corpora. Target audience is the natural language processing (NLP) and information retrieval (IR) community [17].

NLTK was originally created in 2001 as part of a computational linguistics course in the Department of Computer and Information Science at the University of Pennsylvania.

Since then it has been developed and expanded with the help of dozens of contributors. NLTK defines an infrastructure that can be used to build NLP programs in Python. It provides basic classes for representing data relevant to natural language processing; standard interfaces for performing tasks such as part-of-speech tagging, syntactic parsing, and text classification; and standard implementations for each task that can be combined to solve complex problems [1].

The **GATE family of tools** has grown over the years to include a desktop application for developers, a collaborative workflow-based web application, an index server, a Java library, an architecture and a process. The GATE family is intended to minimise time and effort in developing and maintaining rich information extraction, retrieval and management systems, while staying at or near to the state of the technological art, partly by favouring interoperation and reuse over reinvention [5].

The **Link Grammar Parser** is a syntactic parser of English, based on link grammar, an original theory of English syntax. Given a sentence, the system assigns to it a syntactic structure, which consists of a set of labeled links connecting pairs of words. The parser also produces a "constituent„ representation of a sentence (showing noun phrases, verb phrases, etc.). The system is written in generic C code, and runs on any platform with a C compiler. There is an application program interface (API) to make it easy to incorporate the parser into other applications. Davy Temperley, Daniel Sleator and John Lafferty develop a formal grammatical system called a link grammar, show how English grammar can be encoded in such a system, and give algorithms for efficiently parsing with a link grammar. Although the expressive power of link grammars is equivalent to that of context free grammars, encoding natural language grammars appears to be much easier with the new system. The performance of this preliminary system both in the breadth of English phenomena that it captures and in the computational resources used indicates that the approach may have practical uses as well as linguistic significance [21].

2.5 Relevant applications and technologies

Intelligent personal assistant

Intelligent personal assistant is a software, mostly at the mobile devices (but it is planed to be realized at the desktops). It provides s speech user interface to existing software. Examples: Siri, Google Now, Cortana.

Siri is a proprietary intelligent personal assistant. Provides speech interface and assistant features. In contrast to the robot dialog system, intelligent personal assistant in smartphones are depend at the context (geodata, time, usage data). Some points are common, for example in secretary use-cases.

Voice command device

Also existing a class of hardware devices, designed for software described above. This devices are controlled by speech, can control some smart home devices, but cannot interact with objects or move (Jibo has moving parts). Examples: Jibo, Amazon Echo, Cubic.

Chatter-bots

Chatter-bot is software, which imitates dialogs with human without any finite task. Mostly is non-commercial, created by enthusiasts. Widely criticised Turing test defines quality of such software.

Chatter bots is an undervalued part of the dialog systems. Entertainment sphere affects robotics a lot. Some robots are designed as an android robots because of entertainment, not because of construction benefits. Another example is gamification. Gamification has been widely applied in software, which has a regular (touch/buttons/mouse) interface. In spoken interface this interaction use-case is natural. For example Siri has at least 95 humorous answers. That positively affects popularization of robotic technologies and user experience. As we can see strict and unnatural speech commands aren't a good option for domestics robots control.

AIML (Artificial Intelligence Markup Language) is a markup language for chatterbots. It uses templates and states, but this system not includes natural language processing. Plus, this solution does not allow to bind dialogs with a code.

Listing 2.1: AIML example.

```
<category>
  <pattern>WHAT IS YOUR NAME</pattern>
  <template>My name is <bot name="name"/>.</template>
</category>
```

VoiceXML

VoiceXML is designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed initiative conversations. Its major goal is to bring the advantages of Web-based development and content delivery to interactive voice response applications [23].

Listing 2.2: VoiceXML example.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml" version="2.0">
  <form>
    <block>
      <prompt>
        Hello, world!
      </prompt>
    </block>
  </form>
</vxml>
```

None of the systems above aren't designed for robots. Robots in contrast to virtual helpers can physically interact with environment. Dialog state transitions should depend on the environment interaction state too.

Frameworks for creating dialog systems

There are relevant dialog system frameworks list:

Olympus is a framework for creating spoken dialog systems, created at Carnegie Mellon University. Olympus uses the Raven-Claw dialog management framework (Bohus and Rudnicky, 2003). In a RavenClaw-based dialog manager, the domain-specific dialog task is represented as a tree whose internal nodes capture the hierarchical structure of the dialog, and whose leaves encapsulate atomic dialog actions (e.g., asking a question, providing an answer, accessing a database). A domain-independent dialog engine executes this dialog task, interprets the input in the current dialog context and decides which action to engage next. In the process, the dialog manager may exchange information with other domain-specific agents (e.g., application back-end, database access, temporal reference resolution agent) [3].

OpenDial is a Java-based, domain-independent toolkit for developing spoken dialogue systems. The primary focus of OpenDial is on dialogue management, but OpenDial can also be used to build full-fledged, end-to-end dialogue systems, integrating e.g. speech recognition, language understanding, generation, speech synthesis, multimodal processing and situation awareness. The purpose of OpenDial is to combine the benefits of logical and statistical approaches to dialogue modelling. The toolkit relies on probabilistic rules to represent the domain models in a compact and human-readable format. Supervised or reinforcement learning techniques can be applied to automatically estimate unknown rule parameters from modest amounts of data. The hybrid approach adopted by OpenDial makes it possible to easy incorporate expert knowledge and domain-specific constraints within a robust, probabilistic framework [14].

The **Alex Dialogue Systems Framework (ADSF)** is being developed by the dialogue systems group at UFAL, the Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic. Currently this framework includes mature components for public telephone network connectivity, voice activity detection, automatic speech recognition, statistical spoken language understanding, and probabilistic belief tracking. The ADSF is used in a real-world deployment within the Public Transport Information (PTI) domain. In PTI, users can interact with a dialogue system on the phone to find intra- and inter-city public transport connections and ask for weather forecast in a desired city. Based on user responses, vast majority of the system users are satisfied with the system performance [6].

The **Semaine project** is an EU-FP7 1st call STREP project and aims to build a SAL, a Sensitive Artificial Listener [20]. This multimodal dialogue system can:

- interact with humans with a virtual character
- sustain an interaction with a user for some time
- react appropriately to the user's non-verbal behavior

TRINDI (Task Oriented Instructional Dialogue) is an EU-funded research project concerned with

- human-machine interaction using natural dialogue
- task-oriented instructional dialogues are dialogues that enable the user to make choices in the performance of a certain task
- route planning and instructional texts for xerox machine maintenance (as particulare examples)
- finding general and portable technology that will enable the construction of dialogue systems
- increasing the flexibility and knowledge richness of dialogue systems by representing the information which is exchanged in a dialogue

- developing a dialogue toolbox which allows developers to define dialogue rules for information update and to experiment with different kinds of information representing the current state of the dialogue

TRINDIKIT is a toolbox for the development of dialogue systems. It focuses on the development of dialogue move engines. A dialogue move engine updates the information state of the dialogues system on the basis of observed dialogue moves and selects appropriate moves to be performed. Apart from proposing a general system architecture, the TRINDIKIT also specifies formats for defining information states, update rules, dialogue moves and associated algorithms. In TRINDIKIT toolbox developer describes an information state (dialog system states). Dialogue Move Engine changes this state by dialogue moves [12].

The taking of initiative has significance in spoken language dialogue systems and in human-computer interaction. A system that takes no initiative may fail to seize opportunities that are important, but a system that always takes the initiative may not allow the user to take the actions he favours. **PED** is a dialogue management system that uses a probabilistic nested belief model to choose dialogue strategies from a Bayesian game tree. A nested belief model is a representation of the beliefs of the speakers in the dialogue, including their beliefs about each other. PED constructs game trees to model the possible outcomes of the dialogue [15].

Chapter 3

Design

This system is designed as an appropriate tool for solving natural spoken interface use-cases. In most cases any robots activities are described in general-purpose programming languages. For example, a developer would like to move a robot forward. In most popular environments, such as ROS this task could be done with code in `C++/Python`. At this moment general solution for describing activities does not exist. From here, a code in general-purpose programming language, which describes any robot activity or manipulation is called "activity code".

In most cases domestic dialogs are linked with activities. This dialog systems framework/library allows to easily create spoken dialog interface for existing activity code, except some cases, where a code should be designed for outside interaction. Of course it allows to implement dialog systems without any activities/evaluations, only with dialog description features, but this type of dialog system is not useful in robots, because they can't interact with an environment.

3.1 Requirements

Final dialog system solution needs to be easy to use. It should be easy to integrate into existing robots hardware. Dialog system should not depend on the specific domain. It must be designed for robots, and their domestic use-cases.

The main idea of the dialog description is a state machine. But it has some differences with a formal finite state machine. Let's take a look at the non-formal description of the dialog.

Figure 3.1 demonstrates a scheme of dialog example. User would like to order a pizza, with the specific parameters in the specific pizzeria. Circles represent answer states, except idle and activity states. Transitions are initiated with user phrases. Some transitions change internal state (in our dialog system they change activity code global scope variables). Some transitions can lead to several states. In this figure the question initiates an answer and an activity at the moment. Some user inputs can initiate several answers at the moment. Therefore, this system has a several passed states at the same time.

This dialog can be simplified to a tree structure. In this example user can repeatedly change pizza's type with transition. But inside this task, we can change a state in any moment. With several actual states this process can be simplified to structure at figure 3.2.

Structure like this in dialog description file can be created using indentations. We've defined two levels inside dialog description file: question level and answer level. Not only

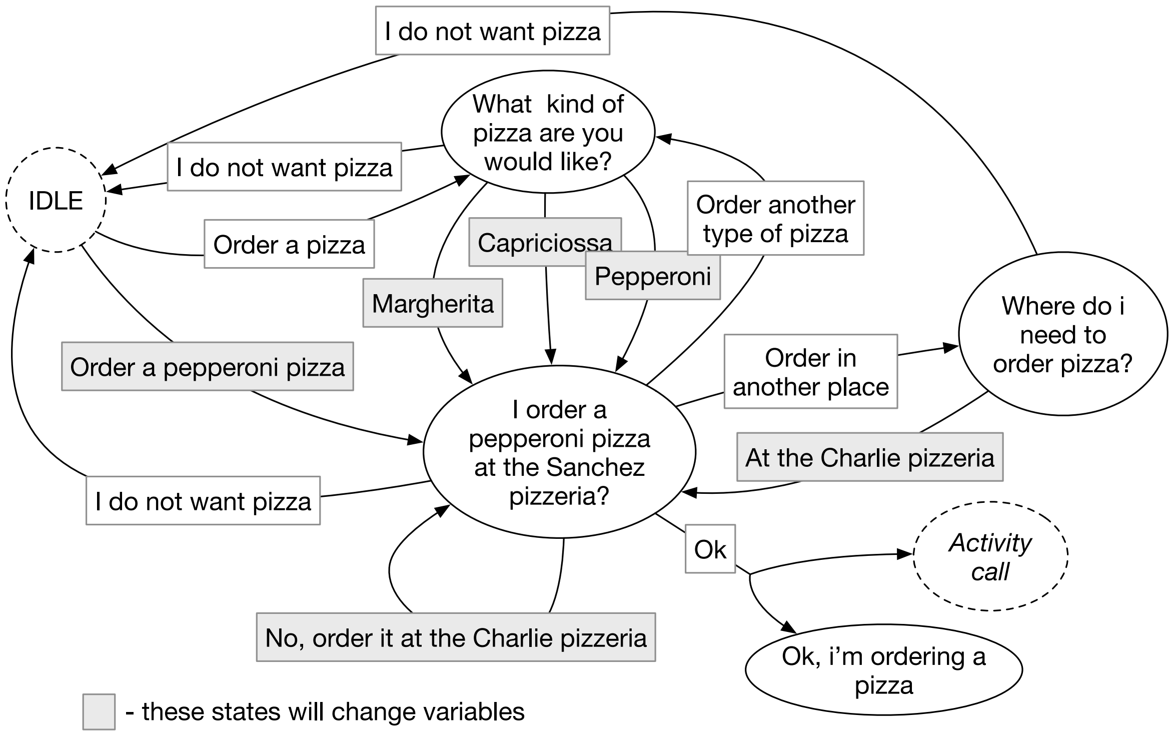

I do not want pizza
What kind of pizza are you would like?
IDLE
I do not want pizza
Order a pizza
Capriciossa
Order another type of pizza
Pepperoni
Margherita
Where do i need to order pizza?
Order a pepperoni pizza
Order in another place
I order a pepperoni pizza at the Sanchez pizzeria?
At the Charlie pizzeria
I do not want pizza
Ok
Activity call
No, order it at the Charlie pizzeria
Ok, i'm ordering a pizza
- these states will change variables

Figure 3.1: Dialog states diagram for pizza ordering dialog.

questions can be at the question level and not only answers can occur at the answer level. There are some more states, which provide linking of the dialog with an activity code. At the listing 3.1 we can see an example of the dialog description for this use-case:

Listing 3.1: Dialog description for diagram 3.2.

```
Order a pizza
        What kind of pizza are you would like?
                I don't want a pizza
                'pizza_type~Pepperoni'
                        I order a 'pizza_type' pizza at the 'pizzeria' pizzeria?
Order a 'pizza_type~Pepperoni' pizza
        I order a 'pizza_type' pizza at the 'pizzeria' pizzeria?
                I do not want a pizza
                Ok
                        Ok, i'm ordering a pizza
                        'order_a_pizza?'
                                "failure"
                                        Sorry, i wasn't able to order a pizza
...
```

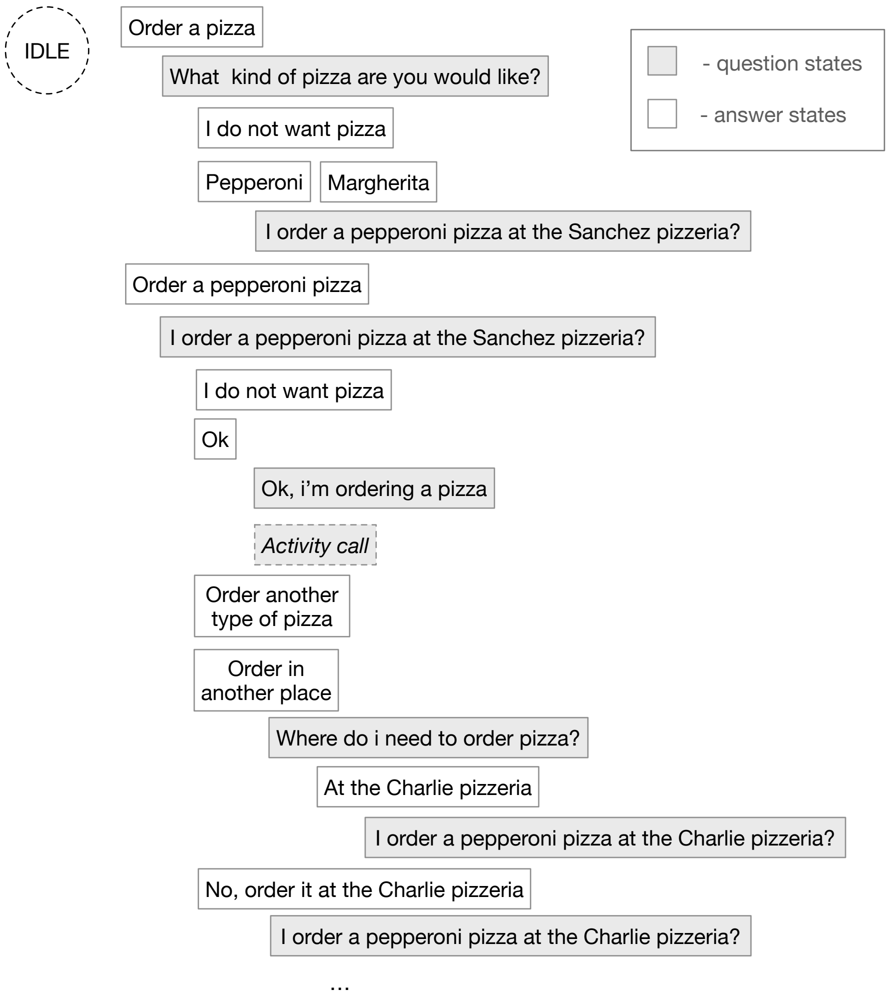

IDLE
Order a pizza
What kind of pizza are you would like?
I do not want pizza
Pepperoni
Margherita
I order a pepperoni pizza at the Sanchez pizzeria?
Order a pepperoni pizza
I order a pepperoni pizza at the Sanchez pizzeria?
I do not want pizza
Ok
Ok, i'm ordering a pizza
Activity call
Order another type of pizza
Order in another place
Where do i need to order pizza?
At the Charlie pizzeria
I order a pepperoni pizza at the Charlie pizzeria?
No, order it at the Charlie pizzeria
I order a pepperoni pizza at the Charlie pizzeria?
...
- question states
- answer states

Figure 3.2: Tree-structured diagram for pizza ordering dialog.

3.2 Key features

For example we have a code, which allows robots to bring an object. Even such a simple task can't be easily done with existing robots software, one needs to deal with computer vision, indoor navigation, object manipulation. In a real environment people delegate this task using speech, and this framework is designed for similar tasks. It is possible to describe communication with a dialog description file.

Dialog description is a special format, which defines, which part of an activity code should be initiated at different states of the dialog. It is tree-structed natural language with special escape sequences. Complete dialog description will be described in section 3.3.

Control flow is always delegated to dialog system. Dialog system interacts with user and with activity code. We've designed two possibilities of programming dialog systems with our library/framework:

- activity code is master. In this case activity code imports dialog system framework at the end of the code, creates dialog system object, selects dialog description file(s) and then dialog system locks the main process until exit.

- activity code is slave. In this case we need to run dialog system interpreter and provide all of the activity programs and dialog descriptions. Then dialog system server imports global scope from each of the activity codes, parses dialog descriptions and starts the dialog.

Slave mode can be useful for implementing multi-agent systems. In section 5.1 will be described realization of web interface for the dialog system. This realization creates new dialog system instance at each connection.

3.3 Dialog description format

To link phrases with activities we need to describe them with a dialog description file. Dialog description file contains dialog description language strings. Dialog system can be run with several dialog description files. It is useful for splitting some topics of the dialog.

Dialog description language is a syntax structure for describing dialog states, activity calls and also it provides some instruments for parsing data from the dialog, and phrase generation. Dialog description language combines declarative and imperative paradigms.

In this project we avoided using XML, which is quite classic for the similar solutions. Our main goal was to provide an easier instrument. Dialog description language interacts with activity code scope, so it is partially a programming language. Coding in the most general-purpose programming languages is not visual (for example Scratch is VPL). Handwriting an XML file isn't easy. So we've designed dialog description format as an instrument, which doesn't require special software to create a description.

Dialog description language has a set of features, which will be discussed in following subsections.

Expected set of phrases

At any moment dialog system expects a set of phrases. This set can be changed by any input/output phrase (called activities can change expected phrases set too).

When user asks a question, dialog system accepts all of its child states. Child states are states below the current state, at the next indentation level. Each acception of answer state extends expected phrases set to its child states. As a result, user can ask different questions.

At the moment dialog system does not lose any states. This depends on the context a lot. In some situations it is needed to loose the previous state immediately. In another situation it is needed to loose all parent states at reaching some of the leaf nodes. In some cases it is not needed to loose any state. Dialog system allows to run one scenario until expected states set is filled. Loosing state could be done manually from the activity code.

Control flow

When human says phrase, dialog system selects the most similar phrase from the expected phrases set. Then it parses data from the input phrase, and extends expected phrases set to this question state's children.

Usually dialog system selects input phrase from the expected phrases set using pattern. AIML uses * for any word number, some system use regular expressions. This dialog system

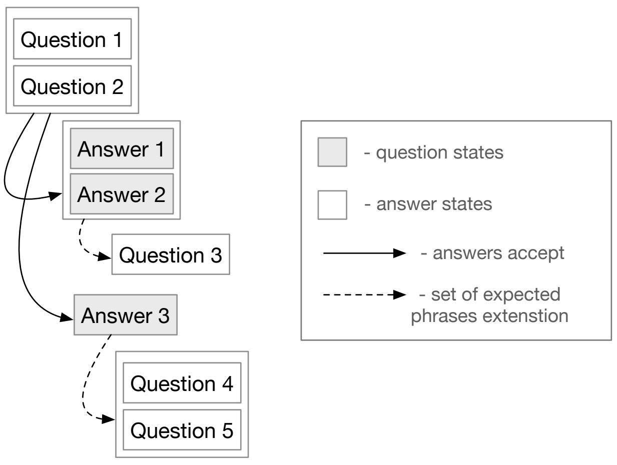


Figure 3.3: Expected set of phrases. Extension example.

uses another mechanism for phrase selection, it is comparison of words links, produced by natural language parser. Link Grammar Parser is used as a natural language processing unit. This parser returns syntax tree of each input sentence. It selects phrase with the most similar structure from expected phrases set. This process will be explained in more details in section 4.4.
In later subsections reviewed dialog description language features:

In-line data

Use case: robot have to answer a with phrase, which contains human name.

In any phrase inside a dialog description file we can use in-line data substitution. The syntax is: ‘`variable_or_function_name`‘. It doesn't matter, which of the two objects (function or variable) will be provided. In case of callable object, it will be evaluated immediately. *Note: function call can lock dialog system, simplex routine (section 3.3) is an appropriate way to call resource-intensive function.*

Use case: human asks a specifying question.

It is also possible to use in-line data substitution in question phrases, each phrase will be evaluated right before syntax structure comparison.

Another option is to use dialog system variables. This variables allows to get recognition quality, origin phrase etc.

Setters

Sometimes we need to get some data from the user phrase. The easiest way to do this is to create a setter inside of the dialog description file. There are two types of setter: fixed setters and flexible setters.

Fixed setters

Use case: user uses abusive language and it should affect dialogs/activity.

Setter allows to set any variable in global scope with fixed setter. In fact it is an assign operation. Another naming is used for flexible setter compatibility. The syntax is: `‘variable_name:Python literal‘`. This setter can be used on both question or answer level. When dialog moves through state, dialog system accepts all fixed setters. Setter acceptation assigns value to the variable at the activity code global scope.

Flexible setters

Use case: dialog system need to parse user name from input phrase.

Flexible setters are another option to set data in global scope. With this setter we set a variable to a word, which is at the similar position inside input inside. It uses link grammar parser to detect the needed word. The syntax is: `‘variable_name~word‘`. They are used only at the question level. Assigned value is always string, developer needs to manually convert this value to the needed data type inside of an activity code. More detailed information in section 4.4.

Routines

It is possible to call a function from dialog description in another asynchronous process. When function blocks dialog process for more than acceptable response time, it is recommended to call this function as a routine, even if it is not a mechanical activity. Routines can communicate with the dialog process via queues. It allows robot to initiate a dialog in case of changing an activity state (real life environment state). There are two types of routine: simplex and duplex. Usage of routines needs special structure of an activity code. At this moment this software becomes a framework.

Simplex routines

Use case: robot needs to download a file, and report after downloading.

Simplex routines only allows to send data from a routine to the dialog. It is possible to run two activities with the same name simultaneously.

The syntax inside a dialog description:

Listing 3.2: Dialog description contains a simplex routine call.

```
question
        ‘function_name?‘
                value1
                        answer phrase
                value2
                        other answers
                                questions etc.
```

Python function receives a Queue object, and can't send any data to the dialog system process.

Listing 3.3: Example of simplex routine activity code.

```
def function_name(responses):
        time.sleep(4)
        responses.put("value2")
```

In this example user asks a question. Dialog system selects `question` state because of most similar sentence structure. Then it accepts its child – simplex routine state. When this state is accepted, `function_name` function is called in another process. After 4 seconds pause, it puts a `value2` string to responses queue. Dialog system, if user not talking with it, receives this value and compares it with each of the simplex routine state children. `value2` equals to `value2`, then dialog system initiates a dialog with a user with `other answers` and extends expected phrases set with `questions etc.` states.

Duplex routines

Use case: robot is going to make a coffee, user would like to undo this command, or specify a sugar portion.

Duplex routines allows us to send data to a routine and to receive a data from a routine. For now it is not possible to detect one of the activities with the same name, when two or more activities with the same name are running simultaneously.

Duplex routines have another syntax, than simplex routines. Code of this function should be designed another way. Not any routine is non-blocking, and our realization of duplex routines allows to create communication with the dialog even if an activity is blocking.

Any activity is a loop of atomic blocking operations. At the beginning of each cycle, routine needs to check the input queue for messages from the dialog system. To avoid manually coding all of these cases, dialog system framework provides decorator, which runs this code in a loop, and can run callback(s) before each loop. It also allows non-obligatory function calls, which run at the start or at the end of an activity. Figure 3.4 demonstrates duplex routine code structure.

Listing 3.4: Example of duplex routine activity code.

```
def callback1(scope, responses):
        pass #it is possible to manipulate with scope object here

def callback2(scope, responses):
        pass

callbacks = {
        value1: callback1,
        value2: callback2
}

def before(scope, responses):
        pass #function called at activity start

def after(scope, responses):
        pass #function called at activity finish

@handle(callbacks, before=before, after=after)
def function_name(requests, responses, scope): # atomic step
        #any blocking code, manipulations with scope
        scope.exit = True #assign to terminate activity
```

What about a scope? Of course there is a problem with changing activity code global scope from another process, but another problem – we need to share data between several functions. Because of closures, it's not possible to share internal scope inside these functions. The solution is to create an empty object scope with metaclass, fill it with global variables from the global scope at the decorator level. All of the duplex activity function can communicate with the dialog system process and get global scope data with scope object. As an opposite to simplex routines, duplex routines can have a question as a child. Also there is a special syntax to send data to routines, showed at lines 12 and 14 in listing 3.5.

Listing 3.5: Example of dialog description for duplex routine.

```
1  question
2          answer
3          ‘function_name?!‘
4                  >value
5                          answer phrase
6                  >value
7                          answer phrase
8                  >value
9                          other answers
10                                 questions etc.
11                 question
12                         answer ‘function_name < value‘
13                 another question
14                         okay ‘function_name < value‘
```

In this example, when user asks a question, dialog system accepts it child states. Dialog system answers with the answer states, and calls a `function_name` in another process. This call will be decorated, and `atomic_step` will be run in a loop, until `scope._exit` attribute is set to `True`. This routine can put any value to responses multiprocessing queue. In this case, dialog system will initiate dialog with an answer phrase. But after routine calls a set of expected phrases was extended to question and another question states. When user asks a question, dialog system will respond with an answer, and this sequence `‘function_name<value‘` will put a value to the requests multiprocessing queue. Decorator at the next iteration of loop will check this queue and find a key in key-value callbacks storage. Value is `callback1` function, it will run before the next loop iteration. This function can change attributes of a scope object, which affects routine behavior.

3.4 Dialog management

In this project we focused on dialog management and semantic parsing. Produced dialog system should contain Speech-To-Text (STT) and Text-To-Speech (TTS) units. Flexible setters and the Link Grammar parser allows us to parse unknown words from an input phrase. Most local STT engines (such as CMU Sphinx) limited by acoustic model and dictionary of sound sequences. We decided to use an external application programming interfaces (APIs) for STT and TTS processing. Dialog manager in designed dialog system can be compared to a regular finite state machine with multiple states. But the model do not equals because of an assign operation at the variables scope, which affects dialog flow.

Chapter 4

Implementation

As a programming language for this project we selected Python. This project requires a lot of text processing, parallel programming and algorithms implementations. High-level language as Python and object oriented paradigm give us a chance of faster development of an easy-to-contribute software. Also Python has a low entry barrier to start programming robots, which is good for education purposes.

Manufactured robots like a PR2 have a regular operating system. Spoken dialog system requires a STT and TTS modules. Most of microcontrollers aren't powerful enough for STT and TTS. This dialog system is not intended for an embedded robotic solution with a microcontrollers control boards.

4.1 Architecture

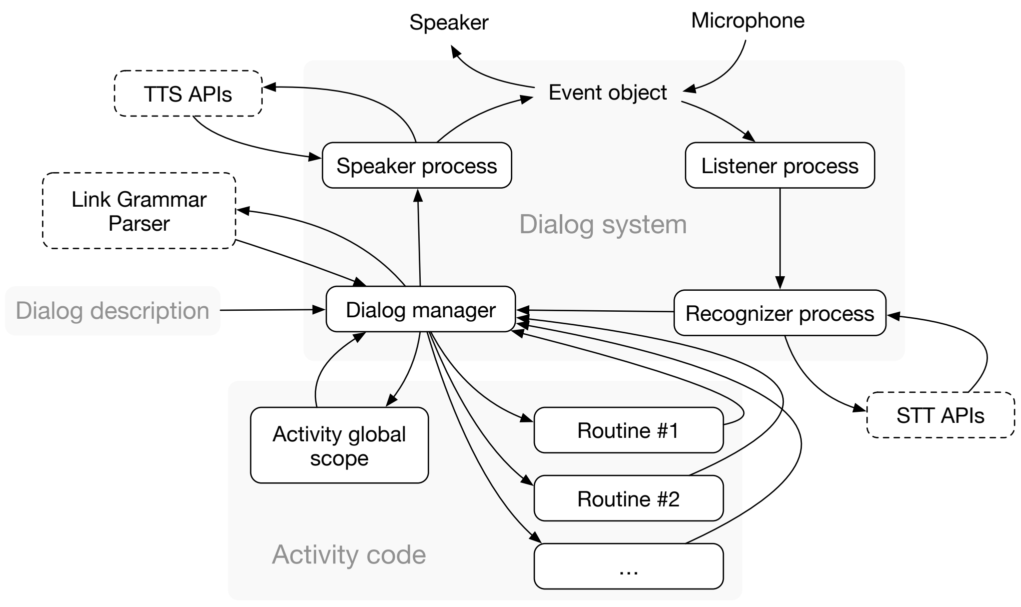


Figure 4.1: Architecture of the dialog system framework.

Dialog system in the architecture level is a multiprocess application. Listener process listens a microphone and starts recording after a threshold value. Then it sends raw recorded data to a recognizer process. This process communicates with a Google or AT&T APIs and sends recognized string to a dialog manager process. After evaluation of this input phrase, dialog manager sends an output text string to a speaker process, which also uses Google or AT&T API and plays synthesized answer. Also a dialog manager creates a new process for each started routine. Speaker or listener processes use shared event object. One of them locks itself, when another uses an audio input/output.

4.2 Modules

Dialog system package contains these modules:

- *interpreter.py* is a dialog manager, which manages input/output phrases and states transitions
- *link_parser.py* Link Grammar Parser bindings
- *parser.py* module for parsing dialog description files
- *phrase.py* implements a phrase object, which can be accepted, compared and evaluated
- *returns.py* manages routine calls to a dialog manager
- *scope.py* manages shared dialog description and activity code variables scope
- *server.py* implements a websocket server
- *speech.py* is a STT and TTS converter, which uses the third-party APIs
- *states.py* is a state structure of different types, to which transitions can be applied.

4.3 Distribution

In ROS a regular way to create a package is to make a workspace. Workspace is a folder structure, which contains sources, build, development and installation spaces. Due to the fact, that Python is a scripting programming language, its software does not require a compilation stage and is distributed as sources. This framework hasn't any ROS dependencies, so it's distributed as a Python package, not a ROS package. More information about how dialog system was run at ROS at section 5.1. Source codes of the dialog system framework was published at the GitHub social coding platform under the MIT license.

4.4 Third-party components

Link Grammar Parser is a natural language processing library. It will be discussed in more details in section 4.4.

This packages are used for audio processing:

- Portaudio and Pyaudio (portaudio Python bindings) used for recording raw data from the microphone.

- `flac` linux utility for wav to flac conversion.

These external APIs used for speech to text and text to speech processing:

- AT&T STT, TTS API. Has a high quality speech synthesis. Has higher requests limits, than Google API. Supports a lot of input formats, including a raw data.
- Google STT API. At the moment limited to 50 requests/per day for non-commercial purposes. Works better than AT&T STT. Receives only FLAC data format.
- Google Translator unofficial speech synthesis. Lower quality, than AT&T TTS. Has a lower response time, unlimited requests.

Link Grammar Parser

For natural language processing this dialog system uses an existing parser named Link Grammar Parser. This parser is based on the link grammar theory of syntax by Davy Temperley and Daniel Sleator. This parser is language independent and depends only on rules database. Rather than NTLK parser, link grammars are suitable for languages with flexible word order. Besides English language, it has Russian language support.

At the moment of dialog system creation, there wasn't any link parser bindings for Python 3, but link parser can generate postscript serialized data format. Dialog system parses postscript format.

Principles. Link grammar uses link rather than part-of-speech tags. Each word has connectors. Each connector has a type, and direction (left or right). A link connects connectors. Each word has rules. Sentence is valid, when each of its words has a valid linkage and global rules are satisfied [21].

Here is an example of a rule:

```
word1 word2: (A- & B+) or (B- & C- & @D+) & \{E-\};
```

This rule works for word1 and word2. `A-` is type A link at the left side. `&`, `or` and brackets create possible variants for linkage. `@` represents multi-connector, curly braces defines non-obligatory connection.

Global rules are planarity rule and connectivity rule. Link can't be crossed and each word should be indirectly connected with any other.

Here is an example a of structure, received by the `link-parser`:

Listing 4.1: Parsed sentence structure example.

```
    +-------->WV------->+
    +------Wd-----+     |
    |        +Ds**c+--Ss-+-Ost-+
    |        |     |     |     |
LEFT-WALL my.p name.n is.v John.m
```

Postscript serialized format:

Listing 4.2: Postscript serialized format example.

```
[(LEFT-WALL)(my.p)(name.n)(is.v)(John.m)]
[[0 3 2 (WV)][0 2 1 (Wd)][2 3 0 (Ss)][1 2 0 (Ds**c)][3 4 0 (Ost)]]
[0]
```

Dialog system compares parsed structures not strictly, it compares only high-level links (Wi - W). As a future imprrovement it is possible to define links weights. That could increase phrase selection accuracy.

Usage in phrase selection. Natural language processing used as a mechanism for input phrase selection. Dialog system selects a phrase with the most similar sentence structure from the expected phrases set. It allows to avoid expressions inside of the dialog description. And this comparison has a higher quality, than regular expression match.

Main weakness of this method, is that doesn't use recognition quality quotient at the phrase selection level. Because of it, phrase selection depends a lot on the speech to text engine quality.

Usage in flexible setters. Natural language processing is also used in flexible setters mechanism. Following listings showing main principles.

At the dialog description:

Listing 4.3: A phrase at the dialog description.

```
My name is ‘name~John‘
```

Preprocessed string is "My name is John„ is directed to the Link Grammar parser. Parsed dialog description:

Listing 4.4: Syntax tree of sentence "My name is John".

```
    +-------->WV------->+
    +------Wd-----+     |
    |       +Ds**c+--Ss-+-Ost-+
    |       |     |     |     |
LEFT-WALL my.p name.n is.v John.m
```

Parsed input phrase "My name is Mark„:

Listing 4.5: Syntax tree of sentence "My name is Mark".

```
    +-------->WV------->+
    +------Wd-----+     |
    |       +Ds**c+--Ss-+-Ost-+
    |       |     |     |     |
LEFT-WALL my.p name.n is.v Mark.b
```

As we can see, O-type link connects transitive verbs to an objects. Then this phrase will be selected. Dialog system detect a word in input structure, which has an O-type link with `is.v` word. After that, it assigns variable name to the string "Mark„. We can use this variable inside later dialog.

Effectivity. At the initialization, dialog system need to parse structures for each sentence inside a dialog description file. Link parser works relatively slow. To improve its speed, dialog system caches every parsed structure to a binary file. Link parser speed grows exponentially by sentence length. But anyway, it is a useful tool for small phrases. Obviously domestic dialogs do not contains huge monologs. Human-robot communications mostly contain questions or imperative sentences from a user to a robot.

4.5 Parser unittests

Dialog system is a high risk area. Robots, which interacts with human and with a human environment are potentially dangerous. Safety is one of the critical issues that must be guaranteed for successful acceptance, deployment and utilization of domestic robots. Control systems should keep up with mechanical design and actuation advancements to guarantee stability and provide acceptable manipulation capability [22]. Dialog system, as a control system, should be highly tested.

Dialog description parser is the core of right dialog systems interpretation. All available escape sequences was tested. Each method of parser class was tested at least with Edge Coverage (EC), several method was tested using Edge Pair Coverage (EPC). Regular expressions was tested by corrsponding control flow graph. By testing, four bug have been found and all of them are fixed at the moment.

Chapter 5

Evaluation

First of all, this dialog system is an interface. Basic software was implemented for experiments on each platform (e.g., robot, web). After the dialog interfaces was implemented.

5.1 Experiments

For an experiment we have created several dialog systems using this framework.

PR2 experiments

To demonstrate dialog system at the PR2 robot, was implemented a dialog interface for PR2 movement control, this dialog system also allows PR2 to move its head and take a picture using the camera. The aim is to test dialog system framework with an acceptable for a domestic environment use-cases.

PR2 control package differs from the regular dialog system framework distribution due to PR2 hardware/software difference.

First of all we wasn't able to use Kinect microphone at the PR2 robot. Architecture of the dialog system was changed to client-server architecture. Distributed dialog system can be executed in network mode. In this case dialog system manager listens a socket for an input phrase. At the client side starts only part of the dialog system, which listens an input phrase and recognizes it locally at the client. Anyway, network mode is similar to the spoken mode and robot speaks all of output phrases using its speakers.

Another problem is PR2 uses ROS hydro distribution. ROS hydro is seventh ROS distribution release and it's primarily targeted at the Ubuntu 12.04 LTS. Most of this ROS version Python packages uses Python 2 interpreter, and it's hard to develop a Python3 application under this environment. This problem was solved with `execnet` Python package, which allows to run both Python2 and Python3 interpreters and serially communicate between this interpreters.

Implemented dialog system firstly was ran in the simulation with `pr2_gazebo` package. After, when we was tested PR2 control dialog system, we tested it at the PR2 robot. Everything worked as planned, robot is able to move the platform and the head, able to take a picture using a dialog interface. Due to robot availability reasons, dialog system for robot control wasn't tested at the users. We can test all of the metrics, such as recognition quality, phrase selection quality in another experimental dialog system, which doesn't require a robot. In addition to that, we have tested this dialog system from the developer's point of view in section 5.3.

Figure 5.1: PR2 robot, on which the experiment was performed.

WebSocket implementation

To make some experiments with this dialog system publicly, we've developed WebSocket server. Websocket is full-duplex protocol over TCP connection. When connection is established, server boot a new dialog system instance in text mode. When user type a phrase into a browser, JavaScript client side send this phrase to the WebSocket server, which forward the phrase to the dialog system instance. It works both ways. In the end, we could test our dialog system throughout the Internet. This Websocket implementation used in interactive demonstration, included to the dialog system online documentation. Anyone can test predefined dialog system or use this dialog system server in web-based conversation agent implementation.

5.2 Readability comparison with alternative formats

Here is a comparison of the dialog description in this framework and in alternative solutions:

Listing 5.1: Our dialog description format exmaple.

```
make ‘target~me‘ a ‘object_name~coffee‘
        Ok, i’m going to make ‘target‘ a ‘object_name‘
make a ‘object_name~coffee‘
        Ok, i’ll just make a ‘object_name‘
I want ‘object_name~coffee‘
        I give you ‘object_name‘
```

Listing 5.2: AIML domain description.

```
<category>
   <pattern>make me a coffee</pattern>
   <template>
      Ok, i’m going to make <bot name="target"/> a <bot name="object_name"/>
   </template>
</category>
<category>
   <pattern>make a coffee</pattern>
   <template>Ok, i’ll just make a <bot name="object_name"/></template>
</category>
<category>
   <pattern>I want coffee</pattern>
   <template>I give you <bot name="object_name"/></template>
</category>
```

AIML language hasn’t natural language understanding features, so we can’t equal this two descriptions.

Another comparison is with OpenDial dialog system framework. OpenDial domain description can be found in appendix C.1. This dialog system allows movement (`base_move` function) and stopping of the robot platform. `_ARQ` and `_RQ` are the dialog system service variables, which contains recognition quality.

Listing 5.3: Dialog description for the movement dailog.

```
go ‘way~forward‘
move ‘way~forward‘
turn ‘way~left‘
        ‘move?‘
                "Move"
                        Ok, moving ‘way‘
                "AskRepeat"
                        Sorry, could you repeat?
stop
        ‘stop?‘
                "Stopping"
                        Ok, stopping
                "AskRepeat"
                        Sorry, could you repeat?
```

Listing 5.4: Activity code for the movement dailog.

```
def move(responses):
        if _RQ > (0.95 * _ARQ):
                if way in ["forward", "straight", "forwards"]:
                        base_move(1.0)
                elif way == "backward":
                        base_move(0.5)
                elif way == "left":
                        base_rotate(-90)
                elif way == "right":
                        base_rotate(90)
                else:
                        responses.put("AskRepeat")
                        return
                responses.put("Move")
        else:
                responses.put("AskRepeat")
def stop(responses):
        if _RQ > (0.95 * _ARQ):
                responses.put("Stopping")
        else:
                responses.put("AskRepeat")
```

Comparing with the XML-like descriptions, our format is easy-to-read and briefer. From the other side, it does not allow changing a description structure. In case of a deep tree structure this format is not convenient. There are several possibilities to solve this issue:

- named states. Named states, like in OpenDial, should allow to describing transtitions without a tree structure.
- require escape sequence. This should allow to import a dialog description file into another one at the selected indentation level.
- scope based states. Question will be available in the expected questions set in case of valid global scope variable value.
- goto escape sequence. Transition to a tagged set.

The third option is complex, but the most solid. The second option is easy to implement and will be added in the next version of the dialog system framework.

5.3 Developer review

This dialog system was tested from the developer view. After source codes of this dialog system was published, two developers tested this framework for their tasks. Here is a review:

> Starting of developing various applications which needs “talk“ with user is quite simple with presented dialog system framework. On the GitHub page there are straightforward installation instructions which are easy to follow. Several examples are delivered with the dialog system, which user can use as a starting point for his own application. Beside the github readme page, there is also an interactive demo, where user can try “talking„ with example application. User can find there also documentation of DDL (dialog description language), which is used for definition of dialog system. Various features of presented dialog system framework are explained here on examples. Usage of this framework is very easy. User just have to import Dialog class to his python application, define dialog system in .dlg file and create variables and routines which are used in the dialog description file.

– *Ing. Michal Kapinus*

Dialog system (DS) is a comprehensively documented system; it is useful for a simple human-robot communication. Specific robot implementation is shown in several examples, designed for PR2 robot. Dialog description language is easy to understand and flexible enough for a quick start of programming own dialogs. Python implementation is well designed, classes and their methods are documented using standard docstrings; there is a set of unit tests. Moreover, the option is very useful, since it allows to using system in both ways, audio and text modes.

– *Ing. Zdeněk Materna*

5.4 Input phrase variability

Input variability evaluation was done with a dialog system, described by the code in listing 5.1.

Listing 5.5: Text mode execution of the dialog 5.1.

```
(env) dialog git:(master)  python3 examples/evaluation/main.py
Data storage path: /Users/xbirge00/Desktop/dialog/tmp
Dialog system started in text mode by default.
======
            make me a coffee
            make a coffee
            I want coffee
======
You> Please make them a huge sandwich without onion
======
1.00 [4]    make me a coffee
1.00 [3]    make a coffee
------
0.00 [0] R  I want coffee (object_name)
======
        target <= them
        object_name <= sandwich
Bot> Ok, i'm going to make them a sandwich
You>
```

As we can see, dialog system accepted a phrase with semantic difference. 1.00 is a score of sentence structure. This means, that dialog system found a 100% of this phrase structure as an input phrase substructure. Second parameter defines absolute number of shared links. This number is a second sorting key, so dialog system tries to select a phrase with most advanced structure, if more than one phrases have the same score. It allows to parse more data from the dialog (`target` and `object_name` variables in this case). Under the threshold we can see a phrase with an R mark. Dialog system provides a guarantee of semantic parsing. Dialog system can't parse `object_name` variable for "I want cofee" state. This state will be rejected even in case if it has a higher score. At the end we have assigned variables list and an answer phrase, generated by layout engine.

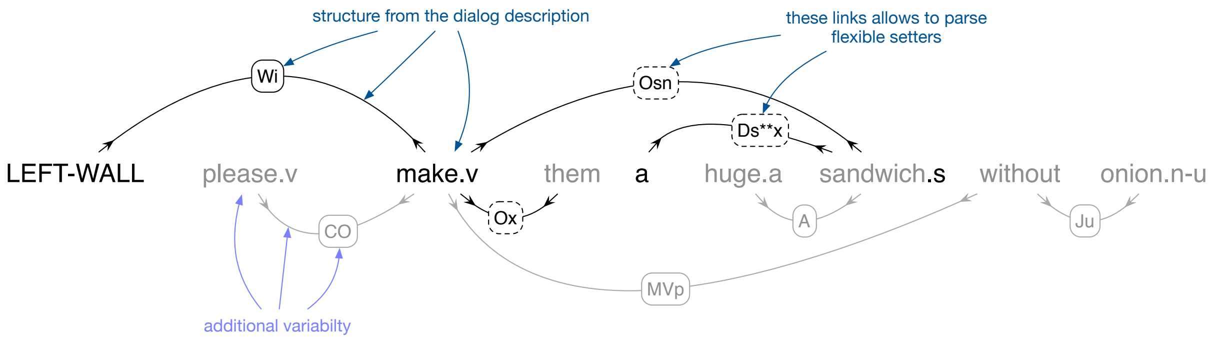


Figure 5.2: Sentence structure with additional words.

At the figure 5.2 showed a structure of the parsed sentences. Potentially we can change input phrase to any other input sentence with this substructure. Phrase variability depends generally at the Link Grammar parser output and particularly at its dictionary size.

Chapter 6

Conclusion

As a result, we have created the dialog system framework, which is an easy-to-integrate solution. It is proven at the real examples, tested at the real robot. This solution is designed as a tool for creating of spoken dialog interface for existing robotic software, not as complete robotic task manager. We can see benefits of dialog system and NLP unit integration. This allows to easily define semantically parsed data in the domain description and more-or-less provide a guarantee of the right context in the phrase. This dialog system is focused at the recognized text-phrases interpretation and uses external speech processing, but dialog system depends at the STT and TTS technologies a lot. Obtained delay on the answer is acceptable for an artificial conversation, but not comparable with a human answers delay. It is hight-priority objective to create fast domain-independent local STT and TTS software. This work bind interaction with the dialog, but there a lot of thing inside a human-robot interaction, which can't work apart, for example computer vision and dialog systems, indoor navigation and object interaction. At my opinion, main goal of the future robotics is to integrate this solution together.

As for a practical usage of dialog system framework, for now it is possible to implement dialog interface for robots, which are working under ROS operation system. Also this framework can be applied for any Python software, without asynchronous calls benefit for robots. Most relevant spheres, where dialog system software can be applied, are: technical support, consulting and help service.

Further development of this software lies in reorganizing of the dialog description language. Dialog description language should be extended to the non-tree structure, which allows to loose states and has some scope condition statements. Also we think, that it is good option to integrate more STT an TTS solutions, including a local TTS engine. This should reduce the answering delay.

Bibliography

[1] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

[2] David Bisset. Service robotics use cases: Cleaning, security, other... Vilamoura, Portugal, 2012. Presented as the IEEE/RSJ International Conference on Intelligent Robots and Systems.

[3] Dan Bohus, Antoine Raux, Thomas K. Harris, et al. Olympus: an open-source framework for conversational spoken language interface research. Technical report, School of Computer Science, Carnegie Mellon University, 2007. http://wiki.speech.cs.cmu.edu/olympus/images/1/1a/Olympus-final.pdf.

[4] Maya Cakmak. Mental model alignment in human-robot communication. Seattle, USA, 2013. Presented as the UW-MSR Summer Institute on Understanding Situated Language in Everyday Life.

[5] H. Cunningham, V. Tablan, A. Roberts, and K. Bontcheva. Getting more out of biomedical documents with GATE's full lifecycle open source text analytics. *PLoS Computational Biology*, 9(2), 2013. http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002854.

[6] Jurčíček Filip, Dušek Ondřej, Plátek Ondřej, and Žilka Lukáš. Alex: a statistical dialogue systems framework. Technical report, Charles University in Prague Faculty of Mathematics and Physics Institute of Formal and Applied Linguistics, 2014. http://www.tsdconference.org/tsd2014/download/preprints/628.pdf.

[7] Open Source Robotics Foundation. About ROS, 2015. http://www.ros.org/about-ros/ [Online; accessed 16-May-2015].

[8] Wilbert O. Galitz. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. Published by Wiley Publishing, Inc., Indianapolis, IN, USA, third edition, 2007.

[9] Willow Garage and Clearpath Robotics. PR2 manual, 2015. https://pr2s.clearpathrobotics.com/wiki/PR2%20Manual [Online; accessed 16-May-2015].

[10] Claire Gardent. Mental model alignment in human-robot communication. Nancy, France, 2012. Presented as the Laboratoire Lorrain de Recherche en Informatique et ses Applications.

[11] Anders Green. *Designing and Evaluating Human-Robot Communication.* PhD thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2009.

[12] S. Larsson and D. Traum. Information state and dialogue management in the trindi dialogue move engine toolkit. *Natural Language Engineering*, 6:323–340, 2000.

[13] James Lester, Karl Branting, and Bradford Mott. *The Practical Handbook of Internet Computing: Conversational Agents.* Chapman and Hall/CRC, Florida, USA, 2004.

[14] P. Lison. *Structured Probabilistic Modelling for Dialogue Management.* PhD thesis, University of Oslo, 2014.

[15] Bryan McEleney and Gregory O'Hare. Decision theoretic dialogue planning for initiative problems. Technical report, Department of Computer Science, University College Dublin, 2005. http://planeffdia.sourceforge.net/um05.pdf.

[16] Timothy Scott McNerney. Tangible programming bricks: An approach to making programming accessible to everyone. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2000. http://xenia.media.mit.edu/~mcnerney/mcnerney-sm-thesis.pdf.

[17] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[18] Alison Sander and Meldon Wolfgang. The rise of robotics, August 2014. [Online; posted 27-August-2014] https://www.bcgperspectives.com/content/articles/business_unit_strategy_innovation_rise_of_robotics/.

[19] Nitin Sawhney and Herve Gomez. Communication patterns in domestic life: Preliminary ethnographic study. Technical report, University of Paris X Nanterre, Nanterre, France, 2000. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.2281&rep=rep1&type=pdf.

[20] Marc Schröder. The semaine api: Towards a standards-based framework for building emotion-oriented system. *Advances in Human-Computer Interaction*, 2010, 2009.

[21] Daniel D. K. Sleator and Davy Temperley. Parsing english with a link grammar. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1991. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/link/pub/www/papers/ps/tr91-196.pdf.

[22] T.S. Tadele, T. de Vries, and S. Stramigioli. The safety of domestic robotics: A survey of various safety-related publications. *Robotics Automation Magazine, IEEE*, 21(3):134–142, Sept 2014.

[23] Steph Tryphonas, Jerry Carter, Daniel Burnett, Brandon Porter, Peter Danielsen, Scott McGlashan, Bruce Lucas, Jim Ferrans, Kenneth Rehor, and Andrew Hunt. Voice extensible markup language (VoiceXML) version 2.0. W3C recommendation, W3C, March 2004. http://www.w3.org/TR/2004/REC-voicexml20-20040316/.

Appendix A

CD Content

```
.
|-- poster.pdf                      poster
|-- sources
|   |-- LICENSE.txt                 MIT license
|   |-- README.md                   readme and manual
|   |-- clean.sh                    environment leaving script
|   |-- dialog                      dialog systems framework sources
|   |   |-- __init__.py
|   |   |-- interpreter.py
|   |   |-- link_parser.py
|   |   |-- parser.py
|   |   |-- phrase.py
|   |   |-- returns.py
|   |   |-- scope.py
|   |   |-- server.py
|   |   |-- speech.py
|   |   +-- states.py
|   |-- docs                        generated documentation
|   |   +-- ...
|   |-- examples                    examples of the dialog systems
|   |   +-- ...
|   |-- init.sh                     environment setup script
|   |-- remote                      remote dialog system components
|   |   |-- spoken_mode.py
|   |   +-- text_mode.py
|   |-- setup.py                    installation scripts
|   +-- tests                       tests for the parser
|       +-- ...
|-- thesis                          tech. report LaTeX sources
|       +-- ...
+-- thesis.pdf                      tech. report
```

Appendix B

Manual

B.1 Virtual environment installation

Please, make sure, that you have a `python3` and `pip3` and/or `easy_install-3.x` installed. Also `curl` utility is needed.

```
sudo apt-get install python3-setuptools python3.4-dev python3-pip
```

It is possible to create a virtual environment with python dependencies. Make sure, that you have `pyvenv-3.4`. Only this version of pyvenv works great with `pip3`. To check, if `pyvenv-3.4` is installed:

```
which pyvenv-3.4
```

You should see a path to the executable. If it is available, you can run initialization script:

```
. ./init.sh
```

`init.sh` file creates a virtual environment, downloads and compiles the Link Grammar parser. Anyway, to run spoken dialog system you also need to install this packages:

```
sudo apt-get install portaudio19-dev mpg123 flac
```

After, you can run examples:

```
python3 examples/pr2_control/main.py
```

To leave the virtual environment:

```
deactivate
```

To activate the environment again:

```
source env/bin/activate
```

To leave and delete the virtual environment:

```
. ./clean.sh
```

B.2 Custom installation

Recommended way to install this package is using the `pip3`. Run this command inside of a project directory.

```
pip3 install ./
```

If you do not have a root:

```
pip3 install --user ./
```

After this command, you can import dialog system framework inside of your `python3` programs. Also this installation adds a command-line `dialog_system` command. Use this command to run slave activity code.

To uninstall this package:

```
pip3 uninstall dialog
```

B.3 External dependencies

All of this dependencies will be automatically installed in case of environment setup. This dependencies required only in case of a custom installation. This DS framework use Link Grammar parser for natural language processing. You can install Link Grammar parser from the aptitude:

```
sudo apt-get install link-grammar
```

Another option is to compile from sources: http://www.abisource.com/projects/link-grammar/#download. DS tested with the 5.2.5 version.

If you would like to run spoken dialog system (not in text mode):

```
sudo apt-get install portaudio19-dev mpg123 flac
easy_install-3.x pyaudio
```

For WebSocket DS you also need a Tornado webserver:

```
pip3 install tornado
```

B.4 Getting started

Your code could have this structure:

Listing B.1: Activity code structure.

```
from dialog import Dialog

# your variables definitions
# your functions/callable objects definition

if __name__ == "__main__":
    DLG = Dialog(globals(), storage="./")
    DLG.load("example.dlg")
    DLG.start_spoken()
```

In storage directory DS will create tmp folder for recorded answers/questions and sentences database. Storage directory by default is the current directory. `DLG.start_text()` or `DLG.start_spoken()` methods defines spoken or text mode.

First of all, take a look at the unfinished documentation here: http://markbirger.info/dialog/. It also includes version 0.2 interactive demo. This version doesn't support duplex routines, but can explain some principles.

In examples directory you can find a few usage examples:

- features_demo uses all DS framework features
- pr2_control allows to control PR2 robot

B.5 PR2 setup

The PR2 robot don't have an access to the Kinect microphone. Spoken DS is not be able to record an audio at the PR2. We've created a `DLG.start_socket(port=42424)` option to run DS. In this example dialog system listens 42424 port, receives text messages and responds with speech. You need to run special remote configuration at the basestation, which can send recognized messages to the socket.

```
python3 remote/text_mode.py -p 42424 127.0.0.1
python3 remote/spoken_mode.py -p 42424 127.0.0.1
```

Note, that spoken mode client stores questions in ./tmp/ directory!

Appendix C

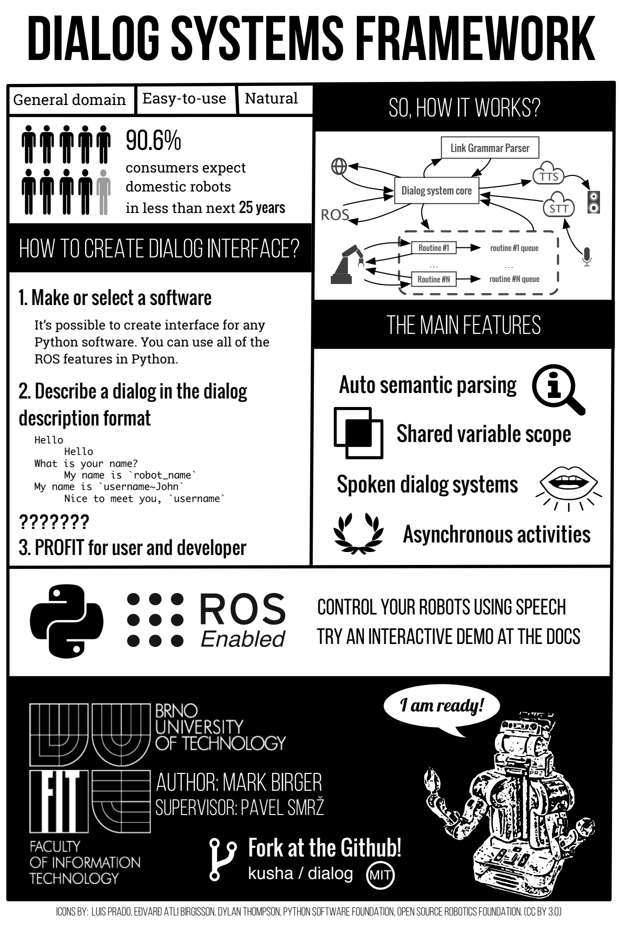
OpenDial domain description

Listing C.1: OpenDial domain definition.

```
<!-- Author: Pierre Lison, Language Technology Group -->
<!-- University of Oslo -->
<!-- Step-by-step example. -->
<!-- The full XML specification for the dialogue domain. -->
<!-- https://github.com/plison/opendial -->
<!-- domains/examples/example-step-by-step_fixed.xml -->

<?xml version="1.0" encoding="UTF-8"?>
<domain>
  <!-- NLU model -->
  <model trigger="u_u">
    <rule>
      <case>
        <condition operator="or">
          <if var="u_u" value="turn * left" relation="contains" />
          <if var="u_u" value="move * left" relation="contains" />
          <if var="u_u" value="go * left" relation="contains" />
        </condition>
        <effect prob="1">
          <set var="a_u" value="Request(Left)" />
        </effect>
      </case>
      <case>
        <condition operator="or">
          <if var="u_u" value="turn * right" relation="contains" />
          <if var="u_u" value="move * right" relation="contains" />
          <if var="u_u" value="go * right" relation="contains" />
        </condition>
        <effect prob="1">
          <set var="a_u" value="Request(Right)" />
        </effect>
      </case>
      <case>
      <condition operator="or">
        <if var="u_u" value="move * forward" relation="contains" />
        <if var="u_u" value="go * forward" relation="contains" />
        <if var="u_u" value="move * forwards" relation="contains" />
        <if var="u_u" value="go * forwards" relation="contains" />
        <if var="u_u" value="go * straight" relation="contains" />
      </condition>
      <effect prob="1">
        <set var="a_u" value="Request(Forward)" />
      </effect>
    </case>
    <case>
      <condition operator="or">
        <if var="u_u" value="move * backward" relation="contains" />
        <if var="u_u" value="go * backward" relation="contains" />
      </condition>
      <effect prob="1">
        <set var="a_u" value="Request(Backward)" />
      </effect>
    </case>
    <case>
      <condition>
        <if var="u_u" value="stop" relation="contains" />
      </condition>
      <effect prob="1">
        <set var="a_u" value="Request(Stop)" />
      </effect>
    </case>
  </rule>
</model>

<!-- Action selection model -->
<model trigger="a_u">

  <rule id="movement">
    <case>
      <condition>
        <if var="a_u" value="Request({X})" />
      </condition>
      <effect util="1">
        <set var="a_m" value="Move({X})" />
      </effect>
    </case>
  </rule>
  <rule id ="negative">
    <case>
      <effect util="-0.5">
        <set var="a_m" value="Move(*)" />
        </effect>
      </case>
    </rule>
    <rule id="repeat">
      <case>
        <effect util="0.2">
          <set var="a_m" value="AskRepeat" />
        </effect>
      </case>
    </rule>
  </model>

  <!-- NLG model -->
  <model trigger="a_m">
    <rule>
      <case>
        <condition>
          <if var="a_m" value="Move({X})" />
        </condition>
        <effect util="1">
          <set var="u_m" value="Ok, moving {X}" />
        </effect>
      </case>
      <case>
        <condition>
          <if var="a_m" value="AskRepeat" />
        </condition>
        <effect util="1">
          <set var="u_m" value="Sorry, could you repeat?" />
        </effect>
      </case>
    </rule>
  </model>

  <!-- Prediction model for the next user dialogue act -->
  <model trigger="a_m">
    <rule>
      <case>
        <condition>
          <if var="a_m" value="AskRepeat" />
        </condition>
        <effect prob="0.95">
          <set var="a_u^p" value="{a_u}" />
        </effect>
      </case>
    </rule>
  </model>
</domain>
```

Appendix D

Poster



DIALOG SYSTEMS FRAMEWORK

General domain | Easy-to-use | Natural

90.6%
consumers expect
domestic robots
in less than next 25 years

HOW TO CREATE DIALOG INTERFACE?

1. Make or select a software

It's possible to create interface for any Python software. You can use all of the ROS features in Python.

2. Describe a dialog in the dialog description format

```
Hello
     Hello
What is your name?
     My name is `robot_name`
My name is `username~John`
     Nice to meet you, `username`
```

???????

3. PROFIT for user and developer

SO, HOW IT WORKS?

THE MAIN FEATURES

Auto semantic parsing

Shared variable scope

Spoken dialog systems

Asynchronous activities

ROS Enabled

CONTROL YOUR ROBOTS USING SPEECH
TRY AN INTERACTIVE DEMO AT THE DOCS

BRNO UNIVERSITY OF TECHNOLOGY

FIT FACULTY OF INFORMATION TECHNOLOGY

AUTHOR: MARK BIRGER
SUPERVISOR: PAVEL SMRŽ

I am ready!

Fork at the Github!
kusha / dialog MIT

ICONS BY: LUIS PRADO, EÐVARÐ ATLI BIRGISSON, DYLAN THOMPSON, PYTHON SOFTWARE FOUNDATION, OPEN SOURCE ROBOTICS FOUNDATION. (CC BY 3.0)

Figure D.1: Research illustration.