# Conversational Task Recognition Agent

Gilberto Ribeiro Paz da Rosa*

2016, v1.0.0

## Abstract

Conversational task recognition system is a practical application of conversational agents using a valued probabilistic transition graph. This work makes an overview of tooling used and why, implementation details of the graph, the probabilistic model used and relevant parts of algorithms.

**keywords**: conversational agents, word recognition, probabilistic model.

## Introduction

Conversational agents are for a long time being used for systems to build a humanized layer between human interaction and task execution of automated environments. More recently, big companies are trying to reach the next level of voice interactive services using large sets of audio and text data to process and classify user audio inputs creating deep knowledge graphs to respond more human likely way.

This work creates a graph with it's nodes, named knots here, via probabilistic transition threshold. Navigation on the graph can only occurs if user input matches, in percentage, at least the minimum value of the edge. Each transition executes a predefined command and template when entering on knot.

## 1 Objective

Create a conversational agent that is capable of executing pre defined tasks throughout voice user inputs and respond with templates texts synthesized as voice output.

*grprosa@inf.ufrgs.br

# 2 Tool set

## 2.1 IBM® Watson®

Watson API is used sessionless for speech synthesizer text responses and speech recognizer making the system process only text transcribed from user voice input.

## 2.2 Web Technologies

User interface and the full system was develop using front-end web technologies, since the graph and the probabilistic linking between knots to audio recording and reproduction. The developed environment was made using HTML and Javascript as final compiled source code.

### 2.2.1 Typescript

The Typescript programming language was used to avoid type checking needed in pure Javascript projects and create good visualization of data flow because it adds Object Oriented Programming principles into Javascript coupling code by meaning and modules and the compile time type check helps to prevent type errors ahead of runtime.

### 2.2.2 Webpack

Webpack is a bundle system to centralize multiples packages into one source file.

### 2.2.3 HTML5

HTML5 API is being used to manipulate audio buffers, display the user interface and communicate with watson servers via http requests.

### 2.2.4 React

React is a Javascript librarie that help to separate the user interface components and integrate different source codes into related meanings mixing css, html and Javascript via JSX language. The JSX files are being integrated with Typescript environment to add type checking to whole compilation pre step required by React.

# 3 Methodology

This conversational agent has different levels of construction. They will be more deeply described as topics from design of the architecture to processing of the voice to be transcribed into text.

## 3.1 Similarity of Jaccard

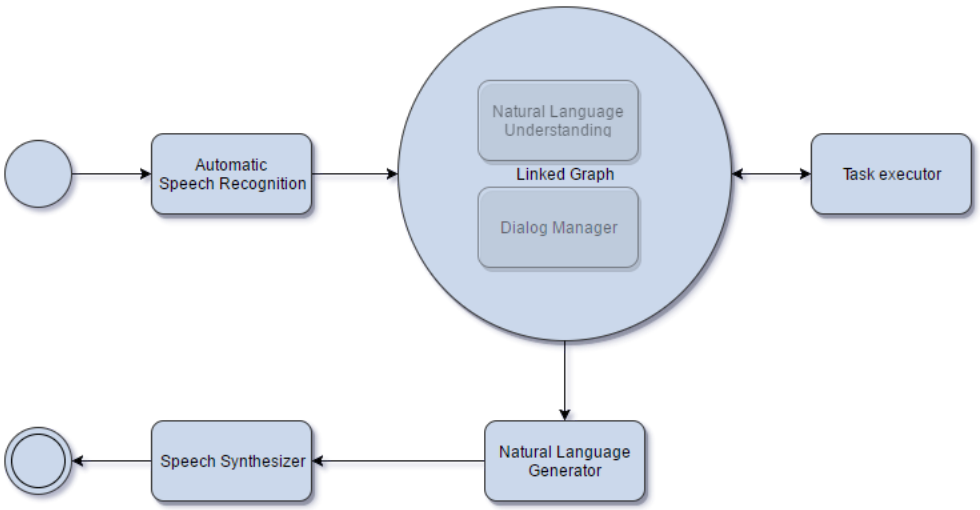
Jaccar method is used to find the similarity percentage between the user input and the edge transition phrase.

## 3.2 Architecture

The initial design of the system started with conceptual components [3]. First, using literature names, the main components were the speech recognition, using watson's API, natural language understander (NLU), to process inputs verifing similarity between edges of current knots, dialog manager, that handle their transitions and commands executions, the natural language generator via template filling method and finally the speech synthesizer to transform final text to voice using watson's API again.

Some modifications were made to the initial state to simplify the development. The NLU is integrated to the dialog manager that is now a valued graph. They booth works together because the NLU is statically defined and the transitions can be made if the similarity of the input with the transition phrase is higher than the value of the edge. Inside the linked graph, if the similarity is higher than 90%, the input phrase will be a new transition phrase from knot A to B.

Figure 1 – Architecture of the system




## 3.3 Finite-State-Machine

A finite state machine, better described on chapter 24 of [2], is how user can navigate between possible tasks and is modeled as a linked graph. When user is in some state, knot, each one has it's own children linked via edge; each edge has two main properties; transition sentence and threshold. For each child transition sentence the Jaccard similarity will be calculated with user input. The edge with the highest percentage that is greater than it's

threshold is going to make the graph navigate to linked child of the current knot. If none of the comparisons had higher value than the threshold, the graph sends back an error message tha the navigation failed.

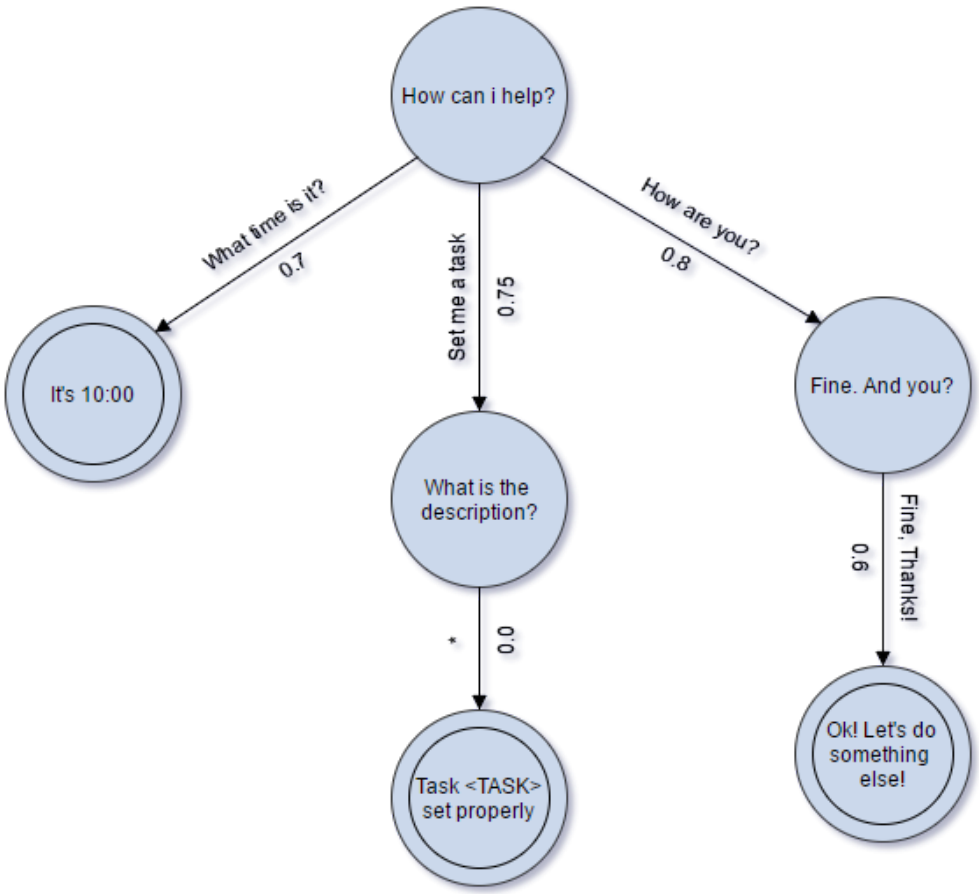
### 3.3.1 Linked Graph

The internal implementation of the finit-state-machine is made using a graph with adjacency lists. Nodes inside are called Knots, because they tie each edge as a link to their children. The transition is calculated inside each knot and returns a list of possible transitions sorted and filtered by similarity threshold. The graph takes the most similar child and apply the navigation to that one and in case that the returned list is empty, it outputs a error on processing requested command.

### 3.3.2 Knots

Inserted and linked into the graph, one by one, the knots control their edges links to each child with a sentence that will be measured the similarity with user inputs. Each knot has a minimum threshold to navigate to it.

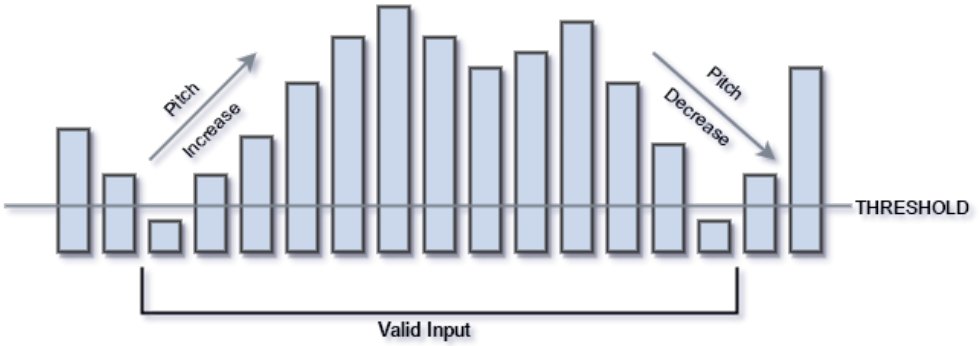
Figure 2 – Decision tree of the current graph



## 3.4 User Input Detection

The most challenge obstacle is observed on watching the user audio input. Determining the valid interval of audio buffer that represents an intended one is hard. Therefore, the system simplifies this task creating an thirty seconds frame buffer and makes a constant scan looking for the frame slice that starts with a pitch increase that is higher than a euristically pre defined threshold volume and ends with pitch decrease that is lower than the same threshold.

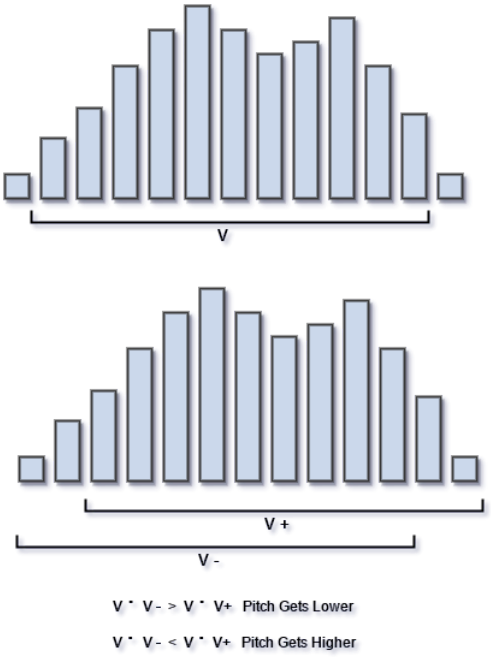
Figure 3 – Detection of valid user input




## 3.5 Pitch Detection

Using vector dot product between last and current audio buffer slices we can detect if the pitch has an increase or a decrease. These vectors are two dimensional; the x and y components are formed by the frequency and the amplitude. The current vector uses the array at indexes one to its length minus two, the previous vector uses the indexes zero and the length minus three and the next vector uses indexes two and length minus one and all of them uses the sabe buffer. If the dot product between the previous and the current vector is higher than the dot product between the current and the next vector the pitch gets lower otherwise the pitch gets higher [1].

Figure 4 – Pitch detection




## 4 Conclusion

The system got working pretty well; the agent is correctly executing the pre defined tasks. Some fixes may be required to voice detection to try to ignore noise and use a better valid user input detection. The tasks may have an ordered way of execution to avoid confusion on programming functions.

## Bibliography

[1] Takeo Igarashi, John F. Hughes, *Voice as Sound: Using Non-verbal Voice Input for Interactive Control*. Computer Science Department, Brown University 115 Waterman Street, Providence, R1 029912, USA. Page: 5.

[2] Daniel Jurafsky, James H. Martin, *Speech and Language Processing, 2nd Edition*. ISBN-13: 978-0131873216. ISBN-10: 0131873210. Page: 3.

[3] Dan Jurafsky, `http://web.stanford.edu/class/cs124/lec/chatbot.pdf`, Stanford University. Page: 3.

[4] Thibault Debatty, `https://github.com/tdebatty/java-string-similarity`, Github.