UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Prototype of a Conversational Assistant for Satellite Mission Operations

Ana Pedro Ferreira Gomes Portela Gonçalves

Dissertação para obtenção do Grau de Mestre em
Engenharia Aeronáutica
(Ciclo de Estudos Integrado)

Orientador: Prof. Doutora Anna Guerman
Co-orientador: Tiago Nogueira

Covilhã, Outubro de 2018

Dedicated to my mom.

Acknowledgments

I have to express my sincere gratitude to all the people who have accompanied me during this journey. Without you this dissertation would not be possible.

First, I would like to thank VisionSpace for the unique opportunity to participate in such a challenging project. I specially thank my supervisor Tiago Nogueira for being always available, helping solve even the non existing problems and keeping me in the right path.

Furthermore I must acknowledge the support given by the rest of the team involved in making Karel a reality, Vlora for developing the always growing and changing Analytics module and Luis for the integration with the web interface and his help on the Analytics methods. Also, I can not forget the people working alongside me during these six months in Germany, for all the shared knowledge along with the laughs.

Moreover, my gratitude goes to my mentor at UBI, professor Anna Guerman, for immediately offering unconditional support and always believing in my work.

I also wish to show my appreciation for my friends, for walking this path with me. I could not have asked for better people to share these years with. Thank you for all the advice, encouragement and sleepless nights.

Finally I must express my gratitude for my family's unconditional presence, never doubting my choices. It is easy to face the challenges with all your support.

Resumo

O primeiro satélite artificial, Sputnik, foi lançado em 1957 e marcou o início de uma nova era. Simultaneamente, surgiram as operações de missão de satélites. Estas iniciam com o lançamento e terminam com desmantelamento do veículo espacial, que marca o fim da missão. A operação de satélites exige o acompanhamento e controlo de dados de telemetria, com o intuito de verificar e manter a saúde do satélite, reconfigurar e comandar o veículo, detetar, identificar e resolver anomalias e realizar o lançamento e as operações iniciais do satélite.

Em 1966, o primeiro Chatbot foi criado, ELIZA, e também marcou uma nova era, de sistemas dotados de Inteligência Artificial. Tais sistemas respondem a perguntas nos mais diversos domínios, para tal interpretando linguagem humana e repondendo de forma similar. Hoje em dia, é muito comum encontrar estes sistemas e a lista de aplicações possíveis parece infindável.

O objetivo da presente dissertação de mestrado consiste em desenvolver o protótipo de um Chatbot para operação de satélites. Para este proposito, criando um modelo de Processamento de Linguagem Natural (NLP) aplicado a missoões de satélites aliado a um modelo de fluxo de diálogo. O desempenho do assistente conversacional será avaliado com a sua implementação numa missão operada pela Agência Espacial Europeia (ESA), o que implica a elaboração do grafico de conhecimentos associado à base de dados da missão.

Ao longo dos anos, várias ferramentas foram desenvolvidas e adicionadas aos sistemas que acompanham e controlam veículos espaciais, que colaboram com as equipas de controlo de missão, mantendo uma visão abrangente sobre a condição do satélite, acelerando a investigação de falhas, ou permitindo correlacionar séries temporais de dados de telemetria. No entanto, apesar de todos os progressos que facilitam as tarefas diárias, as equipas ainda necessitam de navegar por milhares de parametros e eventos que abrangem vários anos de recolha de dados, usando interfaces para esse fim e dependendo da utilização de filtros e gráficos de series temporais.

A solução apresentada nesta dissertação e proposta pela VisionSpace Technologies tem como foco melhorar a eficiência operacional lidando simultaneamente com as suas complexas e extensas bases de dados.

Palavras-chave

Operações de Missão de Satelites, Chatbot, Processamento de Linguagem Natural, Gráfico de Conhecimento

Abstract

The very first artificial satellite, Sputnik, was launched in 1957 marking a new era. Concurrently, satellite mission operations emerged. These start at launch and finish at the end of mission, when the spacecraft is decommissioned. Running a satellite mission requires the monitoring and control of telemetry data, to verify and maintain satellite health, reconfigure and command the spacecraft, detect, identify and resolve anomalies and perform launch and early orbit operations.
The very first chatbot, ELIZA was created in 1966, and also marked a new era of Artificial Intelligence Systems. Said systems answer users' questions in the most diverse domains, interpreting the human language input and responding in the same manner. Nowadays, these systems are everywhere, and the list of possible applications seems endless.
The goal of the present master's dissertation is to develop a prototype of a chatbot for mission operations. For this purpose implementing a Natural Language Processing (NLP) model for satellite missions allied to a dialogue flow model. The performance of the conversational assistant is evaluated with its implementation on a mission operated by the European Space Agency (ESA), implying the generation of the spacecraft's Database Knowledge Graph (KG).
Throughout the years, many tools have been developed and added to the systems used to monitor and control spacecrafts helping Flight Control Teams (FCT) either by maintaining a comprehensive overview of the spacecraft's status and health, speeding up failure investigation, or allowing to easily correlate time series of telemetry data. However, despite all the advances made which facilitate the daily tasks, the teams still need to navigate through thousands of parameters and events spanning years of data, using purposely built user interfaces and relying on filters and time series plots.
The solution presented in this dissertation and proposed by VisionSpace Technologies focuses on improving operational efficiency whilst dealing with the mission's complex and extensive databases.

Keywords

Satellite Mission Operation, Chatbot, Natural Language Processing, Knowledge Graph

Contents

Acknowledgments . . . v
Resumo . . . vii
Abstract . . . ix
List of Figures . . . xiii
List of Tables . . . xiii
List of Acronyms . . . xvii

1 Introduction **1**
1.1 Context . . . 1
1.2 Motivation . . . 2
1.3 Contributions . . . 3
1.4 Objectives . . . 4
1.5 Dissertation Outline . . . 5

2 Theoretical Background **7**
2.1 Satellite Mission Operations . . . 7
2.1.1 Mission Operations Ground Segment . . . 7
2.1.2 Flight Control Team . . . 8
2.2 Chatbots . . . 9
2.2.1 Evolution of Chatbots . . . 9
2.2.2 Architecture of Chatbots . . . 11
2.2.2.1 Components . . . 12
2.2.2.2 Domain . . . 13
2.2.2.3 Conversation . . . 13
2.2.2.4 User's Intent and Entities . . . 13
2.2.3 Common Challenges . . . 14
2.2.4 Chatbot Development Platforms . . . 14
2.2.4.1 Chosen Framework . . . 19
2.2.5 Existing Bots . . . 19
2.3 Natural Language Processing . . . 22
2.3.1 History of NLP . . . 22
2.3.2 Speech Processing . . . 23
2.3.3 NLP for Chatbots . . . 24
2.4 Knowledge Base . . . 25
2.4.1 Different Models . . . 25
2.4.2 Other Knowledge Graphs . . . 26
2.4.3 Knowledge Graph for Mission Operations . . . 28

3 System Architecture **29**
3.1 System Overview . . . 29
3.1.1 Data Archive and Analytics . . . 30
3.1.2 Bot and Knowledge Base . . . 31
3.1.3 Human-Bot Interaction . . . 31
3.1.4 Integration . . . 32
3.2 Knowledge Graph . . . 33

3.2.1 RDF . . . 34
3.2.2 Domain Ontology Model . . . 34
3.2.3 Graph Generation . . . 36
3.3 Rasa Framework . . . 37
3.3.1 Rasa NLU . . . 37
3.3.1.1 Pipeline . . . 38
3.3.1.2 Training Data . . . 41
3.3.1.3 Training the NLU model . . . 43
3.3.2 Rasa Core . . . 44
3.3.2.1 Policy . . . 44
3.3.2.2 Training Files . . . 45
3.3.2.3 Training the Dialogue model . . . 47

4 Implementation and Evaluation 49
4.1 The Gaia Mission . . . 49
4.2 Creating the Knowledge Graph . . . 51
4.3 The Gaia Chatbot . . . 53
4.3.1 NLU model . . . 53
4.3.2 Dialogue model . . . 56
4.3.3 Evaluating and Running the Models . . . 57
4.4 Test Cases . . . 59
4.4.1 Simple Intents . . . 59
4.4.2 Numeric Queries . . . 60
4.4.3 Time Interval Query . . . 62
4.4.4 Transition Queries . . . 64
4.5 Discussion . . . 65

5 Conclusions 69
5.1 Overview . . . 69
5.2 Recommendations for Future Work . . . 70

Bibliography 73

A Chatbot Training 77

B Sentence Classification Example 79

C NLU Model Evaluation 81

D Dialogue Model Evaluation 85

E Plots from the Numeric Queries Test Cases 87

List of Figures

1.1 Karel's initial architecture. 4

2.1 General architecture of chatbots from [1]. 9
2.2 Simple chatbot adapted from [2]. 12
2.3 Chatbot main componentes adapted from [3]. 12
2.4 Exemple of Dependency Parcing from [4]. 24
2.5 Sentence with two possible dependency parses from [5]. 24
2.6 Freebase subgraph of ”Family Guy” from [7] 26

3.1 Karel system stack. 29
3.2 Analytics module architecture. 30
3.3 Karel's architecture. 33
3.4 Karel's knowledge graph development. 36
3.5 Rasa simple architecture from [8]. 37
3.6 Generic pipeline organization using SpaCy from [9]. 39
3.7 Call order of pipeline components during NLU model training from [10]. 43

4.1 Star Map created by Gaia, adapted from [11]. 50
4.2 AOCS systems organization. 51
4.3 Knowledge Graph mapping of FSS1. 52
4.4 Mapping of a FSS1 parameter. 53
4.5 Karel's generated story paths. 58
4.6 Output graph for the average angle rate of the NGY20006 parameter. 62
4.7 Representation of overfitted model from [12]. 66

C.1 NLU Model Intent Confusion Matrix. 82
C.2 NLU Model Intent Prediction Confidence Distribution. 83

D.1 Dialogue Model Action Confusion Matrix. 86

E.1 Maximum current of the NPC10079 parameter, part of the cps. 87
E.2 Maximum current of the NPC10136 parameter, part of the cps. 88
E.3 Maximum current of the NPC10121 parameter, part of the cps. 88
E.4 Maximum current of the NPC10098 parameter, part of the cps. 89
E.5 Maximum current of the NPC10102 parameter, part of the cps. 89
E.6 Maximum current of the NPC10117 parameter, part of the cps. 90
E.7 Minimum voltage of the NFS10020 parameter, part of the fss3. 90
E.8 Minimum voltage of the NFS10018 parameter, part of the fss3. 91
E.9 Minimum voltage of the NFS10021 parameter, part of the fss3. 91
E.10 Minimum voltage of the NFS10019 parameter, part of the fss3. 92
E.11 Minimum voltage of the NFS10017 parameter, part of the fss3. 92
E.12 Minimum voltage of the NFS10016 parameter, part of the fss3. 93
E.13 Average angle rate of the NGY20008 parameter, part of the gyro set. 93
E.14 Average angle rate of the NGY20006 parameter, part of the gyro set. 94
E.15 Average angle rate of the NGY20007 parameter, part of the gyro set. 94

List of Tables

2.1 Comparison table of most prominent AI Services from [13]. 15
2.2 Comparison table of most prominent bot frameworks from [13]. 15

List of Acronyms

API	Application Programming Interface
AI	Artificial Intelligence
AIML	Artificial Intelligence Markup Language
AOCS	Attitude and Orbit Control System
CPS	Chemical Propulsion Subsystem
CDMS	Control and Data Management System
DB	Database
DSP	Deployable Solar array Panels
DSA	Deployable Sunshield Assembly
DSL	Domain-Specific Language
EGSE	Electrical Ground Support Equipment
ECSS	European Cooperation for Space Standardization
ESA	European Space Agency
ESOC	European Space Operations Center
XML	Extensible Markup Language
FN	False Negatives
FP	False Positives
FSS	Fine Sun Sensor
FCT	Flight Control Team
FPA	Focal Plane Assembly
GYRO	Gyroscope
ICD	Interface Control Documents
KB	Knowledge Base
KG	Knowledge Graph
LSTM	Long Short-Term Memory
LGA	Low Gain Antennas
ML	Machine Learning
MT	Machine Translation
MIT	Massachusetts Institute of Technology
MPS	Micro-Propulsion subsystem
MCS	Mission Control System
MIB	Mission Information Base subsystem
NER	Named Entity Recognition
NLP	Natural Language Processing
NLU	Natural Language Understanding
POS	Part-Of-Speech
PM	Payload Module
PAA	Phased Array Antenna

PS	Propulsion System
QA	Question & Answering system
RDF	Resource Description Framework
SCOS-2000	Satellite Control and Operation System 2000
S2K	SCOS-2000
SVM	Service Module
SS	Space Segment
SSM	Space System Model
SOE	Spacecraft Operations Engineer
STR	Star Tracker
TC	Telecommands
TM	Telemetry
TTC	Telemetry, Telecommand & Command
TN	True Negatives
TP	True Positives
W3C	World Wide Web Consortium

Chapter 1

Introduction

The very first artificial satellite was launched in 1957 by the Soviet Union. Sputnik marked a new era, and since then, about 6 600 satellites from more than 40 countries were launched, with the most varied orbits, launchers and purposes. At the same time, satellite mission operations was born. Thenceforward these activities have been permanently evolving with the increase of missions' complexity, both in systems and goals. Nine years later, in 1966, the very first chatbot was born. ELIZA also marked a new era, of rising Artificial Intelligence Systems used to parse natural language with the intention of assisting humans. Everyday new chatbots are created for different purposes, and the list of possible applications seems endless.
This master's dissertation aims to connect both worlds, and develop a conversational assistant for mission operations. Resorting to Natural Language Processing, the project aims to implement a chatbot that helps satellite operators to be more effective and efficient on day-to-day operations. This concept bot was named "Karel" after Czech playwright Karel Čapek who used the word "robot" in the context we use it today, for the first time, in 1920. Karel Čapek wrote a science-fiction play centered around factory-made entities called robots which looked very similar to humans. The word robot derived from the Czech noun "robota" meaning "labor" [14].
This Chapter presents the context, motivation, contributions and objectives of the Karel chatbot project, followed by the overview of this dissertation.

1.1 Context

A chatbot is essentially a computer program aimed at simulating the conversation of a human being [15]. Initially, after ELIZA, which imitated a psychoanalyst, many human-computer dialogue systems have been developed to simulate different fictional or real personalities. With that type of programs, the purpose is generally to simulate conversation and entertain the user. However, more and more, conversational assistants are being created and implemented with useful practical applications. Such question and answering based systems with the intent of getting some information or data requested by the user, are even being used as interfaces to assist search engines. In these cases, the goal of the Natural Language Processing (NLP) system is to represent the true meaning and intent of the user's query, which can be expressed in everyday language as if they were speaking to another person [16]. The recent fast arising of these chatbots is due in part to major advances in Artificial Intelligence (AI), in particular in Machine Learning (ML), which assure greater accuracy in speech recognition and Natural Language Understanding (NLU), assuring the correct queries are answered and the user's intent is satisfied.
Nowadays, the aim of chatbot designers and developers should be to build tools that help people, facilitate their work, and their interaction with computers or systems using natural language. The goal should not be to replace the human role totally, or imitate human behavior perfectly, not even to mimic human to human conversation [17]. A modern conversational interface offers several other advantages over traditional graphical user interfaces as it provides a simpler and easier way to get information or make small actions and it enables extended and always available interaction

[18]. Thus emerged the idea to develop a Conversational Assistant in a field that has not been considered in this direction yet.
Building chatbots from scratch is extremely difficult, time consuming and costly, this was the reason that led to the option of using a free open source framework. With the exponential growth of AI, being one of the focus of researchers all around, today many chatbot development platforms are available, making the work easier and letting the developer focus on the domain. This methodology favors the fast expansion of AI products as agents in everyday life, not only for personal use, but specially in companies and public domains, to enhance both client experience and employee productivity. All these progresses unlock the possibility of a range of innovative projects with many useful applications, much like the one presented to the writer of this dissertation by VisionSpace Technologies, of essentially designing a chatbot in the domain of satellite missions.

1.2 Motivation

Running a satellite mission requires tight cooperation between multidisciplinary teams. The groups of engineers that monitor and control the ground-stations, the communications system and the satellite itself, and design, build and deploy the hardware and software systems that support the mission. The European Space Operations Center (ESOC), located in Darmstadt performs the day-to-day operations of most satellites from the European Space Agency (ESA). Within ESOC's operational environment, the Spacecraft Operations Engineer (SOE) is responsible for monitoring and controlling the spacecraft, along with the rest of the Flight Control Team (FCT). The SOE must generate and uplink new activity plans as required by the mission and investigate and recover from failures when necessary, in addition to the daily tracking of the health and performance of all the systems.
Throughout the years, many tools have been developed and added to the the systems used to monitor and control spacecrafts helping engineers either by maintaining a comprehensive overview of the spacecraft's status and health, speeding up failure investigation, or allowing to easily correlate time series of telemetry data, for instance DrMUST. However, despite all the advances made which facilitate the FCTs' tasks, the teams still needs to navigate through thousands of parameters and events spanning years of data, using purposely built user interfaces and relying on filters and time series plots to answer questions such as:

- When was the last occurence of event X?
- When was the last time component Y was switched ON/OFF?
- What was the maximum fill ratio for packet store Z in the last month?
- How many times did system X exceed the highest temperature threshold in the last year?

As regular information and detailed control is necessary, getting all the data from the databases for mission planning and control should be an efficient process. Benefiting from the technological advancements made in Artificial Intelligence services, this work tries to address these situations by prototyping a Conversational Assistant system for mission operations. This way providing an interface for all queries concerning satellite monitoring and data control. The solution presented in this dissertation and proposed by VisionSpace Technologies focuses on improving operational efficiency whilst dealing with the mission's complex and extensive databases.

1.3 Contributions

The solution introduced by VisionSpace Technologies intends to unveil full capabilities to browse across mission's historic data by automatically translating natural language into queries using an AI mechanism. Essentially, in order to respond correctly to a question, the system must first analyze it, to extract its intent and important entities, then consult the databases, matching the intent and entities with the correct data, and finally present the response to the user in the most appropriated way. The final purposes of this application are:

- To analyze engineer's queries and understand what is requested;
- To provide the correct answer to the engineer's query very efficiently;
- To save the engineer's time since there is no longer need to personally go through the database.

VisionSpace's system is leveraged by the implementation of a Machine Learning mechanism that allows for smart data recognition and correlation. All in one, this solution has been designed to reason as state-of-the-art in cognitive services for mission operations. The chatbot adds value to mission operations by:

- Cutting operation costs;
- Reducing test and validation periods;
- Increasing reaction time to critical situations;
- Relaxing the user's skills (DB and programming);
- Enabling problems' prediction and as a result facilitate fast data driven decisions;
- Promoting consistent data analysis;
- Ensure easy data availability from decommissioned missions.

In addiction, to guarantee its easy implementations at ESOC, the developed chatbot is expected to cope with all the different ESA proprietary systems, such as:

- ARES - provides offline data analyses, correlation and reporting information;
- DARC - makes parameter stream requests;
- Mission DB - browses through artifact definitions.

Overall, this work contributes with:

- Implementing the first conversational assistant for satellite mission operations,
- Developing a NLP model for satellite operations,
- Generating a Knowledge Graph (KG) of satellite mission Database.

1.4 Objectives

The main goal of the activity proposed by VisionSpace Technologies to the author of this master's dissertation is to develop a prototype of a chatbot system, for mission operations using RASA's framework. The prototype will be developed and demonstrated using a dataset from a real satellite mission operated from ESOC, GAIA. Therefor, the project of this dissertation involves the following objectives:

- Get familiar with ESOC's operational environment, and build a basic understanding of mission operations, the activities it involves, the software tools used and the problems usually encountered.
- Conduct interviews with SOEs at ESOC to understand the most useful and common type of queries that the system should be able to answer.
- Get familiar with the mission's database to understand what type of data is provided and how it can be queried.
- From the output of the interviews and the analysis of the database define and build an initial (simple) knowledge graph with the entities, the relation between entities and the intents and the intents supported by each entity.
- For a set of identified entities and intents develop the required database queries and supporting algorithms.
- Generate training and evaluation data sets for the NLU model. Which is responsible for extracting entities and intents from user's input text.
- Train and evaluate the NLU model.
- Validate the model using the mission's database, and created knowledge graph.

Figure 1.1 displays the first conceptual architecture of the proposed project, explaining all the requirements graphically, for better understanding of the pretended final product.

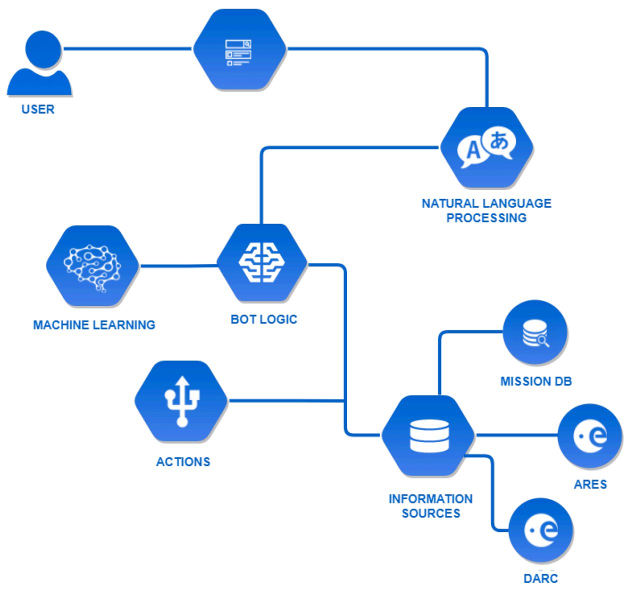


Figure 1.1: Karel's initial architecture.

1.5 Dissertation Outline

The present dissertation is divided into five chapters structured the follows:

- Chapter 1 introduces the scope of this dissertation, the context and motivation of the developed project, as well as the objectives and contributions expected to be achieved.
- Chapter 2 presentes the theoretical background relevant for the project of this dissertation. Framing the chatbot regarding its technological domain, followed by the characterization of such systems. The concepts of Natural Language Processing and Knowledge Graph are also explained, as the tools for the development of these systems.
- Chapter 3 focuses on the architecture of the complete virtual assistant, further detailing the conversational bot and the generation of the knowledge graph.
- Chapter 4 describes the implementation and evaluation of the chatbot. Presenting the chosen mission to do so, besides the complete descriptions of all the the necessary processes. Including as well the test cases considered and the discussion of the achievements.
- Chapter 5 concludes the dissertation providing an overview of the work performed and assessing the created chatbot prototype. Additionally presenting the recommendations for future work.

Chapter 2

Theoretical Background

Before getting immersed in the project, this chapter contains a review of important theoretical aspects to better understand the scope of this Master's dissertation. Firstly, an explanation of the basic concepts associated with satellite mission operations will be presented in Section 2.1, to frame the project regarding the technological domain it is intended for. Section 2.2 introduces chatbots and the evolution of these systems, afterwards, it presentes platforms with the purpose of developing a conversational agent as well as existing chatbots to show its applicability. After, Section 2.3 presentes Natural Language Processing and Machine Learning as a way to develop these systems. Finally, Section 2.4 defines the idea of knowledge base, imperative to connect the program with its application field.

2.1 Satellite Mission Operations

Satellite operations start at launch and finish at the end of mission, when the satellite is decommissioned. For this purpose, flight controllers and other ground-team personnel monitor all aspects of the mission resorting to telemetry, and send all the necessary commands to the spacecraft, from the Operations Center. Satellite operations are conducted to verify and maintain satellite health, reconfigure and command the spacecraft, detect, identify and resolve anomalies and perform launch and early orbit operations [19].
While there are different kinds of space architectures, they can all be broken down into three main physical parts: the space segment, the launch segment, and ground segment. The satellites contain the payload that will accomplish the primary mission, the launch vehicles transport the satellites to orbit and the ground segment consists of processing and distribution facilities that properly format and allocate the raw data from the satellite for the users.

2.1.1 Mission Operations Ground Segment

The Operational Ground Segment of a satellite mission comprises of a center of operations, at least one ground station and the communications system. This segment is responsible for the monitoring and control of the complete mission, ensuring the spacecraft safety and good condition. The tasks to be executed include the transfer of the satellite into its operational orbit, following its launch; the operation of the spacecraft and its payload throughout its in-orbit lifetime according to the mission requirements and the requests generated during the mission itself; and the collection and transfer of all the raw science data generated by the payload instruments, in conjunction with selected spacecraft and mission auxiliary data to the scientific community [20].
The mission operations control center is in charge of maintaining the on-board software and instruments, issuing commands, data uploads, and software updates to the spacecraft, and process, analyze, and distribute telemetry. It is responsible for providing flight dynamics support including determination and control of the satellite's orbit and attitude and providing the telemetry and auxiliar data for the scientific purpose of the mission [21]. By and large the center of operations performs mission control of the spacecraft and payload, compatible with satellite safety, scientific

requirements and overall system resources. Being the sole interface to the spacecraft, it also carries out all activities related to mission planning and control of the overall systems, taking into account all constraints and capabilities, along with payload command request handling. Thus it must support real-time/near real-time data-processing essential to achieve its purpose. Moreover it must support the acquisition and interim storage of raw scientific data, to possibly be accessible together with raw housekeeping and auxiliary data by scientists at remote locations [20].

The ground stations perform telemetry, telecommand and tracking operations as well as payload data transmission and reception, and it may be controlled remotely. Its location and availability along with the characteristics of the spacecraft orbit or trajectory determine the timing of passes, when a line of sight exists to the spacecraft [22] and data can be ”uplinked” or ”downlinked”. The stations should track the satellite so that their antennas are properly pointed and data can be send and received as it moves overhead. Ground stations may temporarily store telemetry to be sent to control centers later.

The ground segment provides capabilities for monitoring and control of the satellites in all mission phases, as well as for reception, archiving and distribution of the payload instrument data. The good performance of this division is of utmost importance, as it contributes to the overall scientific success of the satellite's mission and validates the performance of the satellite.

However, a large scale mission may have large amounts of data to store and process. For instance a space system taking high-resolution imagery, assuming it takes 50 pictures of 50 0Mb each, every day, for 10 years, that leaves a system architect with a whopping 91 terabytes of information to store [23]. Thence, this component may use different data downlink gateway systems, and mission data processing, storage and distribution. To analyze and oversee the database is the duty of the flight controllers. Moreover, the more spacecraft that are placed in more orbits, the more complex and important their effective operation becomes.

2.1.2 Flight Control Team

As stated before, the success of a satellite mission relies on its operation from the ground. To do so flight controllers make use of telemetry and monitor every technical aspect of the space mission in real time or near-real time, assuring its success and safety.

The ground team gathers telemetry from the spacecraft and then explore that data to make decisions on how best to proceed. The personnel generates the necessary procedures, a set of rules and conditions, for ordinary mission operations or for response to mission events and anomalies. There are generic policies that take place for every launch and mission-specific ones that are tailored for each situation. Whichever the case, the analysis of the mission's database is of most value for the determination of the correct course of action.

Once a mission begins, flight controllers have at their disposal multiple monitors receiving data from equipment in space. As nowadays the majority of the systems are operated automatically, their job is to continuously monitor that data and assure all equipments are working as foreseen [24]. In case of changes in the nominal performance of the satellite or malfunctions in any of its systems, FCTs must prepare and perform the necessary actions to restore the satellite's nominal as fast as possible to minimize outage of science data.

To guarantee the reliability of the satellite, accessing the missions data is essential, however as previously exposed, is possible that the size of the database is very considerable. Besides, increased sensor data collection capability and the expansion of constellations replacing individual satellites, possibly require innovative data tracking and handling methods. In this sense, a Conversational Assistant seems very helpful, specially as a time-saving tool for both emergency responsiveness, and routine control operations. Whereas the flight controllers and engineers are responsible for the planning and scheduling of every satellite operation, and the mission depends on them especially when unexpected complications arise.

2.2 Chatbots

A chatbot is a service, powered by rules and sometimes artificial intelligence, that you interact with in natural language, through text or voice, about a certain domain or a particular topic. The service could be almost anything, ranging from functional to fun, and it could live in any chat interface [25]. Different terms have been used for these programs, such as conversational system, virtual agent, dialogue system, and chatterbot [17].
Nowadays, these systems are everywhere, answering users' questions in the most diverse domains, interpreting the human language input and responding in the same manner. Thence, they may be referred to as software agents that pretend to be a human entity, or at least simulate the conversation of one. The programs can be merely conversational or also perform some productive functions. Figure 2.1 demonstrates the architecture of modern conversational agents.

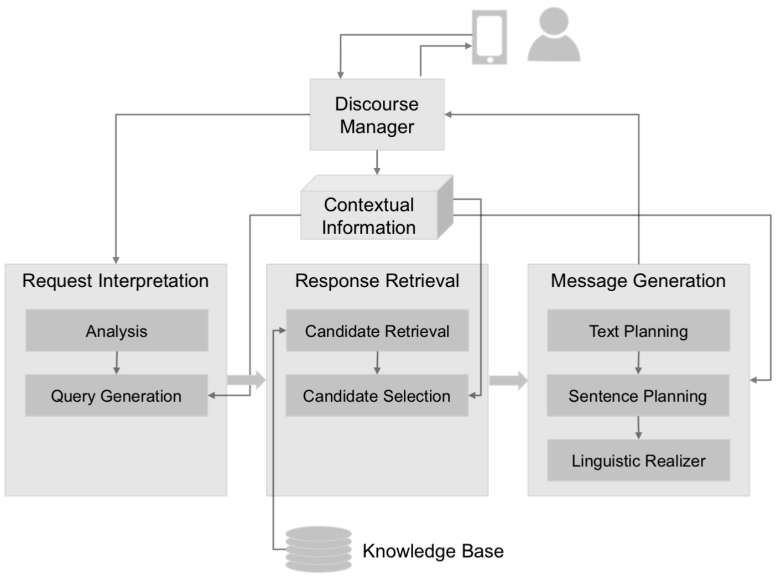


Figure 2.1: General architecture of chatbots from [1].

Theses systems' architecture, based on AI and NLU, which will be presented in the next Section, integrates a language model and computational algorithms to emulate communication flow, as well as some learning capability. This allows any person to chat with computers and request actions from them using natural language, to some extent. The predefined knowledge base helps develop a response to the query, and can define the domain of the bot, but its ability to learn can be determining for its performance. To better understand the concept, it is important to explore its history, in the following subsection.

2.2.1 Evolution of Chatbots

The first conceptualization of the chatbot is attributed to Alan Turing, who asked "Can machines think?" and described the problem as an "imitation game" in 1950. This game, later called the "Turing Game" challenges an interrogator, or a panel of interrogators, to distinguish between a

person and a computer only based on the answers given anonymously (via keyboard, for example) to his questions. If the unknown entity is classified as human while it is actually a computer, then the system is said to have passed the test. For this to be possible, the computer scientist contemplated machines having some sort of thinking but which would be very different from what a human does and even believed machine errors could be a positive contribution [26].
Allan Turing proposed instead of trying to produce a program to simulate the adult mind, rather try to produce one which simulates the child's. Stating that if this program was then subjected to an appropriate course of education, the adult brain would be obtain. Mimiquing the process which brings an adult brain to the state that it is in, that includes the initial state of the mind, say at birth, the education to which it has been subjected, and other experiences besides education. He thus divided the problem into two parts, the child-program and the education process. However, these two remain very closely connected. The mathematician defended that to find a good child-machine is necessary to experiment with teaching it and see how well it learns, trying different approaches and evaluating if it is better or worse. Concluding that there is an obvious connection between this process and evolution [26].
Although, a chatbot was only a reality 16 years later, the Turing test is still one of the most popular measures of intelligence of such systems. In addiction, most of them still utilize Allan Turing's idea of having a data set as the base of its knowledge that is then expanded by "learning" with interaction.
ELIZA, the first chatbot in history, was implemented in 1966 by Joseph Weizenbaum at the Massachusetts Institute of Technology (MIT). It was described as a computer program for the study of natural language communication between man and machine. The system analyzed input sentences on the basis of decomposition rules which are triggered by key words appearing in the input text. The responses were generated by reassembly rules associated with selected decomposition rules. ELIZA was design to emulate a psychotherapist. This mode of conversation was chosen because in a psychiatrist interview, the doctor is free to assume the pose of not knowing basic things about the real world, and the client may assume that he had some purpose in directing the conversation. In any case, it has a crucial psychological utility in that it serves the speaker to maintain his sense of being heard and understood [27].
To understand natural language, ELIZA's fundamental actions were based in:

1. The identification of the "most important" keyword occurring in the input message;
2. The identification of some minimal context within which the chosen keyword appears, for instance if the keyword is "you", if it is followed by the word "are" an assertion is probably being made;
3. The choice of an appropriate transformation rule and, the transformation itself;
4. The generation of "intelligent" responses when the input text contained no keywords;
5. The provision of an editing capability that facilitates altering and extending scripts, on the script writing level.

Hence, the simple bot could apply a kind of simple template to the user's sentence, and reassemble an answer specifically associated with it. This way specifing that any sentence of some form can be answered likewise, independently of the meaning [27].
This reveals that the program was not really "intelligent", but rather pretended to. However, seeing that at the time the aim was for chatbot systems to fool users that they were real humans, ELIZA was thought to have a very good performance. The next aspired step was to provide the

system with memory in order for it to build a model of the conversation. Thus being able to make assumptions and detect rationalizations or contradictions of the user, which could then initiate arguments.
To the second model was added an "evaluator" capable of accepting expressions and evaluating them, furthermore storing the results for subsequent retrieval and use. As well as the possibility of the program containing three different scripts simultaneously and fetching new scripts from among a supply stored on a disk storage unit, with intercommunication among coexisting scripts. By this time, several computer papers were available dealing with different components of the machine understanding problem, which could be added to ELIZA or used to improved the system. It was clear that the problem was not yet solved.
One of the most known chatbots derivative from ELIZA is ALICE, which name is an abbreviation for Artificial Linguistic Internet Computer Entity. It is a system that you can chat with, implemented by Dr. Richard S. Wallace in 1995. ALICE's knowledge is stored in Artificial Intelligence Markup Language (AIML) files, an abbreviation of The Artificial Intelligent Markup Language that is a derivative of Extensible Markup Language(XML). Each AIML file contains data objects, made up of topics and categories, which contain either parsed or unparsed data. The first unit is an optional top level element that has a name attribute and a set of categories related to it. Each category contains a pattern representing the user input and a template implying the robot response. The AIML interpreter tries to match word by word to obtain the largest pattern matching, best one [15].
Another important adaptation of ELIZA in the evolution of theses programs is Elizabeth, in which the various selection, substitution, and phrase storage mechanisms have been enhanced and generalized to increase both flexibility and potential adaptability. In this bot, knowledge is stored as a script in a text file, where each line in this text is started with a script command notation to distinguish between: welcome message, quitting message, void input, input transformation, keyword pattern, keyword response pattern, output transformation, memorized phrase, action to be performed within a message, and comment [15].
Both programs are good examples of Artificial Intelligence, Natural Language Understanding and Pattern Matching historical research. The main differences between the two are that ALICE stores a complex corpus text and uses simpler patterns and templates to represent input and output, while Elizabeth provides more grammatical analysis for sentences and uses more complex rules. The main advantage of the first over the latter is the ability to partition the user's input and then combine the answers.
Since the first bot, a variety of new architectures and technologies have arisen, each attempting to simulate natural human language more accurately and thoroughly. Creating lifelike simulations of human beings is a critical task in itself. Such entities may be interesting in a game or maybe as a friendlier user interface, but they are not part of the core science of creating intelligent machines, that is, machines that solve problems using intelligence [18]. The following Subsections describe the current state of the conversational agents.

2.2.2 Architecture of Chatbots

AI designing generally requires the system to be differentiated into simple modules of work. In chatbots, there is a separate module that manages the transfer and understanding of messages between human user and the AI system, another module that stores the useful key-points during the chat that may be required later-on, another module that handles the errors and so on. A typical architecture of a simple conversational system is shown in Figure 2.2.

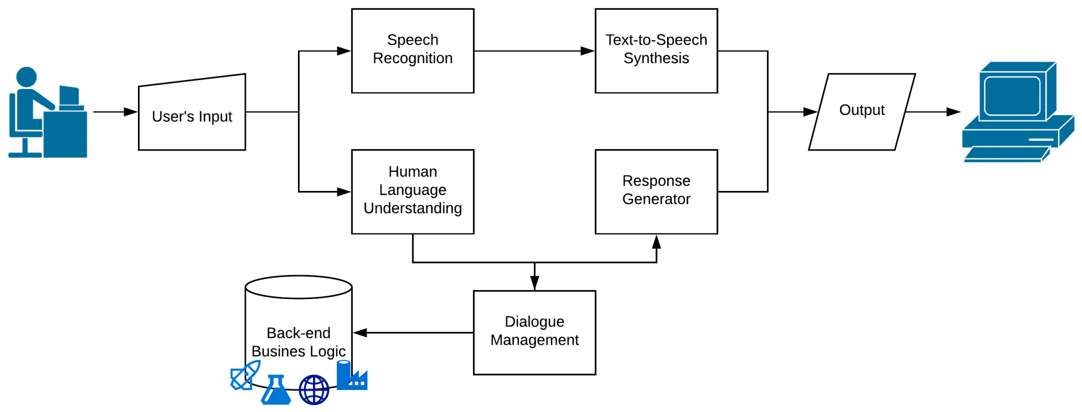


Figure 2.2: Simple chatbot adapted from [2].

The input to the virtual agent is processed by the Dialog Management component that updates the system's belief state and decides on the next action. This may involve interacting with the back-end part of the system that includes business logic or access to web services. The system's output is then produced [2].

2.2.2.1 Components

The chatbot software is composed of three main components, as shown in Figure 2.3, and detailed hereafter:

- The Responder: Is the interface between the user and the core routines. It handles the input and output, transfers user data to the Classifier, and delivers the bot's response to the user.
- The Classifier: Normalizes and filters the input. It applies substitutions and splits the user input into logical components. The normalized strings are then transferred to the Graphmaster. Also processes the output from the Graphmaster, handles various instructions, and delivers the bot's response to the Responder.
- The Graphmaster: Organizes the storage of the contents. It handles the pattern matching process, which involves an advanced search-tree algorithm. The database may be stored as a graph [3].

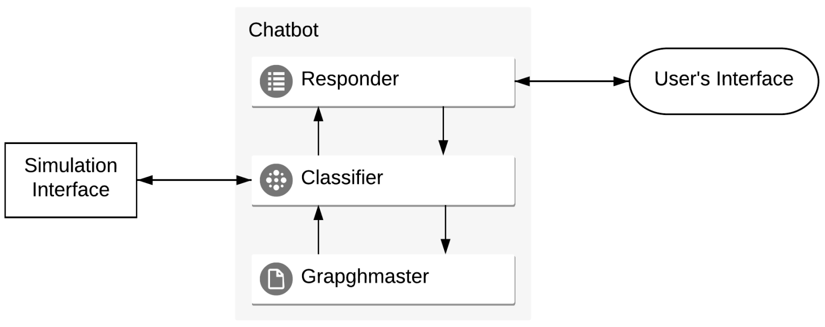


Figure 2.3: Chatbot main componentes adapted from [3].

2.2.2.2 Domain

Chatterbots can be grouped according to the domain in which they operate, according to its knowledge or area of action, as:

- Closed Domain: The space of possible inputs and outputs is somewhat limited because the system is trying to achieve a specific goal. Karel' satellite mission operations domain is an example of closed domain. These systems aim at fulfilling their specific task as efficiently as possible. It is not required of them to handle all possible conversational cases.
- Open Domain: The user can take the conversation anywhere. The bot does not necessarily require a well-defined goal or intention. The infinite number of topics and the fact that a certain amount of world knowledge is required to create reasonable responses makes this a very hard problem. [28].

2.2.2.3 Conversation

It is possible to divide dialogue systems in two main types, a transactional chatbot is used to accomplish some task, such as booking a flight, making a purchase, or asking about football scores, while a conversational chatbot engages in "chit-chat" for primarily social purposes. Where the conversation is text-based, the interaction is similar in form to a chat between two human users on a messaging app, except that in this case one of the participants in the interaction is a chatbot. With voice-based conversations the interaction is similar to a conversation between two human users on the telephone.
Additionally, there are two main types of conversational interaction that can be used by current conversational interfaces:

- One-shot queries: Are user-initiative and take the form of simple input-output pairs in which the user asks a question or issues a command. Typically there are no follow-up questions from the user and no questions are initiated by the system.
- Slot-filling dialogues: Are system-initiative with the interaction being controlled by the system. Usually, the user is not able to take the initiative and ask questions.
- Open-ended dialogue: Are a more natural conversation involving mixed-initiative interaction and potentially multi-turn, with context maintained across the conversation [2].

For the developer, the longer the conversations are the more difficult to automate. With short-text conversations the goal is to create a single response to a single input. The system may receive a specific question from a user and reply with an appropriate answer. For long conversations, the system may have to go through multiple turns and keep track of what has been said. Customer support conversations are typically long conversational threads with multiple questions [28].

2.2.2.4 User's Intent and Entities

After receiving the input sentence by the user, the chatbot must interpret it in order to match it with an intent, which is essentially the user's goal. This way being able to perform the planned action and answer the user. The dialogue system may follow the following process, not much different from the described in Subsection 2.2.1 from ELIZA:

1. Filter the input by removing prepositions and articles from it;
2. Extract the keywords from the input, for example nouns or verbs;

3. Apply a pattern matching algorithm that matches the keywords with an intent;

4. Perform the requested action or return the appropriated response.

To improve the system's accuracy, other useful information may be extracted from the sentence. Entities are the words considered important form the message. Either to add context to the request or better specify what is intended. The classification of these words is not mandatory, some simpler sentences and intents do not need any additional entities, to be answered. However most queries intended to retrieve some sort of data from the bot require extra details as to what is really meant to be obtain.

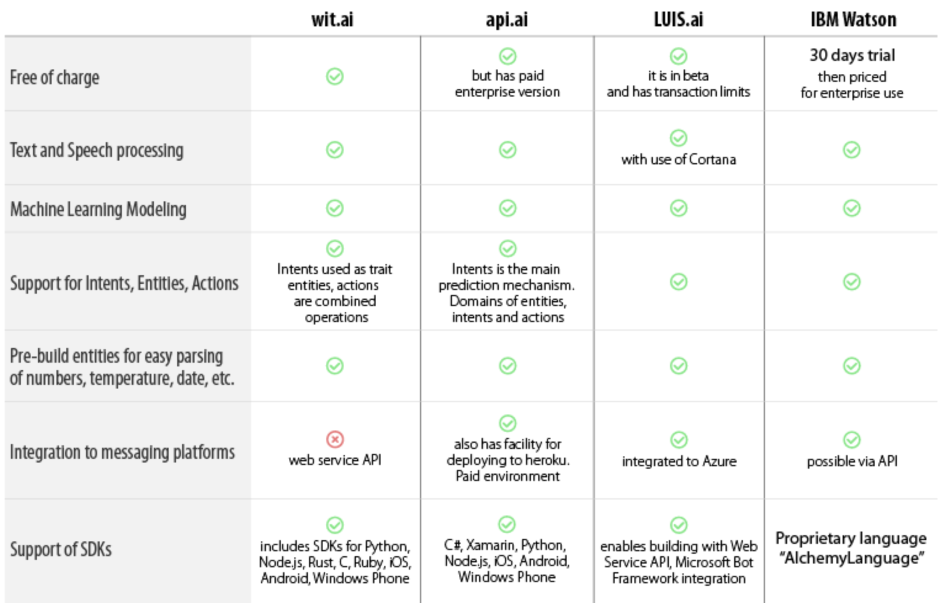
2.2.3 Common Challenges

Despite all the advances in speech recognition and natural language understanding, chatbots are still less than proficient in conversational abilities, for example, in dealing with follow-up questions, unexpected inputs, changes of topic, and recovery from error. This way often interactions with bots are unnatural and error-prone. Developing a conversational agent presents diverse challenges, mostly in research areas. Some will be now explained.
To produce sensible responses systems may need to incorporate both linguistic context and physical context. In long dialogues is important to keep track of what has been stated and what information has been exchanged. The most common approach in solving this for bots is to embed the conversation into a vector, but doing that with long conversations is challenging. It may also be required to incorporate other kinds of contextual data such as date/time, location, or information about a user.
When generating responses the agent should ideally produce consistent answers to semantically identical inputs. A system trained on a lot of data from multiple different users may learn to generate linguistic plausible responses, but is not trained to generate semantically consistent ones.
The ideal way to evaluate a conversational agent is to measure whether or not it is fulfilling its task, in a given conversation. But such labels are expensive to obtain because require human judgment and evaluation. Sometimes there is no well-defined goal, as is the case with open-domain models. All these aspects difficult the evaluation of chatbot models.
A common problem with generative systems is that they tend to produce generic responses that work for a lot of input cases. That is partly a result of how these systems are trained, both in terms of data and in terms of actual training objective/algorithm. Humans typically produce responses that are specific to the input and carry an intention. For this reason, generative systems, particularly with open-domain, are not trained to have specific intentions, lacking this kind of diversity [29].

2.2.4 Chatbot Development Platforms

Creating a Natural Language Understanding model from scratch implies a significant work volume and hours, however there are some good tools that assist this process. For development of Virtual Assistants, there exist bot frameworks and AI services:

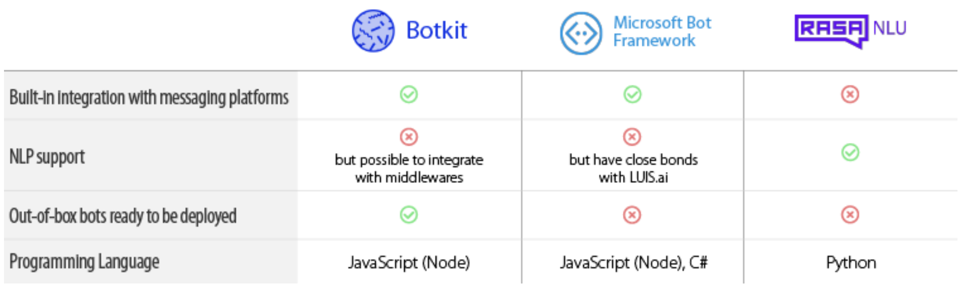
- AI services are independent, cloud-hosted platforms, often exposing a GUI for interactive creation of chatbot logic, featuring Machine Learning powered NLP capabilities, and enabling communication via RESTful Application Programming Interface (API) [13]. Table 2.1 shows a comparison between some of these programs. These platforms will be presented first in the next Subsections.



	wit.ai	api.ai	LUIS.ai	IBM Watson
Free of charge	✓	✓ but has paid enterprise version	✓ it is in beta and has transaction limits	30 days trial then priced for enterprise use
Text and Speech processing	✓	✓	✓ with use of Cortana	✓
Machine Learning Modeling	✓	✓	✓	✓
Support for Intents, Entities, Actions	✓ Intents used as trait entities, actions are combined operations	✓ Intents is the main prediction mechanism. Domains of entities, intents and actions	✓	✓
Pre-build entities for easy parsing of numbers, temperature, date, etc.	✓	✓	✓	✓
Integration to messaging platforms	✗ web service API	✓ also has facility for deploying to heroku. Paid environment	✓ integrated to Azure	✓ possible via API
Support of SDKs	✓ includes SDKs for Python, Node.js, Rust, C, Ruby, iOS, Android, Windows Phone	✓ C#, Xamarin, Python, Node.js, iOS, Android, Windows Phone	✓ enables building with Web Service API, Microsoft Bot Framework integration	Proprietary language "AlchemyLanguage"

Table 2.1: Comparison table of most prominent AI Services from [13].

- Frameworks constitute an abstraction for the generic functionality of chatbot workflows in a packaged and convenient way. The chatbot frameworks are much like any other software frameworks, provide tools and utilities. Are usually implemented for a certain programming language. In addition, some of the bot frameworks also have hosted and interactive development environments to facilitate creating bots to even bigger extent [13]. These platforms are simply compared in Table 2.2 and discussed further on.



	Botkit	Microsoft Bot Framework	RASA NLU
Built-in integration with messaging platforms	✓	✓	✗
NLP support	✗ but possible to integrate with middlewares	✗ but have close bonds with LUIS.ai	✓
Out-of-box bots ready to be deployed	✓	✗	✗
Programming Language	JavaScript (Node)	JavaScript (Node), C#	Python

Table 2.2: Comparison table of most prominent bot frameworks from [13].

Most of the tools share the same basic concept. With example data, the user can train a classifier to categorize so called intents, which represent the intention of the whole message and entities, keywords necessary to sustain the dialogue. When it comes to the core of the services, the machine learning algorithms and the data on which they are initially trained, all services are very secretive. None of them gives specific information about the used technologies and datasets [1]. These platforms function mostly with closed domain, and can answer with predefined templates or utilize external code for more complex requests. The next Subsections will discuss the main features of each option.
Among the most popular NLU services are Microsoft Bot Framework, IBM Watson, DialogFlow (former Api.ai), wit.ai, Pandorabots and Amazon Lex, all cloud platforms presented on an easy to use interface. Must of them have been developed for over ten or more years. Therefore, their experience provides the most advanced tools and offers the most flexible solutions for businesses.

These platforms make it possible to use different programming languages. Each has developed its own SDKs, uses cutting edge data processing and analysis technologies, and supports dozens of natural languages. Further, they are already embedded in customer services, sales, marketing, order processing, social media, payment, recruitment and other industries. However, many startups were created over last few years, some of them grow fast and are already well-known, these options will be presented in Subsection 2.2.4).

DialogFlow

Dialogflow, previously called API.ai, is based on the concepts of Intents and Contexts, used to model the behavior of the chatbot. Intents are the mapping between what a user inputs and what response or action should be carried out by the bot. Contexts are used for distinguishing user inputs which might have different intents depending on the preceding dialogue. They are maintained across inputs by setting an output context for a parent intent and the same context value for follow-up intents. Contexts for the built-in follow-up intents are generated automatically [2].
When a user inputs data in the Dialogflow.com platform, it is first checked if it matches a pre-defined intent, which address the most common use cases: fallback, for queries that the chatbot cannot answer, yes or no, later, cancel, and custom. The feature named “Default Fallback intent” handles the user inputs that do not match any pre-defined intent. The match cases of an intent can be limited by stating a list of contexts that should be working. The match cases of an intent can create and deleted the contexts. This system of intents and contexts make possible to develop chat-bots with large and complex flows [30].
There are predefined knowledge packages collected over several years. Available SDKs are Android, iOS, Cordova, HTML, JavaScript, Node.js, .NET, Unity, Xamarin, C++, Python, Ruby, PHP, Epson Moverio, Botkit, and Java. Now it is even possible to integrate the agent with Actions on Google, which enables building applications for the Google Assistant, the only assistant in Google Home, an application that lets users interact with services through voice commands [31]

IBM Watson

Watson is a platform built on a neural network, accessing one billion Wikipedia words, it understands intents, interprets entities and dialogs, supports English and Japanese languages, and provides developer tools like Node SDK, Java SDK, Python SDK, iOS SDK and Unity SDK. This program is the first choice as a bot-building platform for businesses [31].
IBM Watson Conversation makes use of slots to handle follow-up questions as well as under-specified user queries. If a user asks a follow-up question with new variable values, the original slot values are over-written. This method also supports a user-correction of a value. For under-specified queries that require a clarification request by the system, the system enters a slot-filling dialogue and asks for the required variables. IBM Watson Conversation makes use of a parameter table similar to that of DialogFlow [2].

Wit.ai

In Wit.ai platform, the important concept to model a chat-bot are stories. Each story is an example of a conversation. The intent here is a user entity, which is not mandatory. In order to model a complex chat-bot, the platform can group a large number of intents in stories. The NLP engine of Wit.ai is trained with examples. When a user writes a request of similar nature, the entities are extracted and the logic implemented by the developer is applied [30].

It is available for developers to use with iOS, Android, Windows Phone, Raspberry Pi, Python, C, and Rust; it also has a JavaScript plugin. Wit.ai allows using entities, intents, contexts, and actions, and it incorporates natural language processing. It is said to be used by over 100 000 developers [31].

Pandora Bots

The Pandorabots API allows to integrate a bot hosting service and natural language processing engine into your own application. Is a chatbot development platform based on AIML and includes ALICE (one of the bots referred to in Subsection 2.2.1). The purpose of it is to enable human computer conversation without considering a task or action-oriented scenario (chit-chat). It can take much effort to scale up the conversational patterns, built manually [30].
The platforms developed SDKs are Java, Node.js, Python, Ruby, PHP, and Go and it is multilingual. Common use cases include advertising, virtual assistance, e-learning, entertainment, and education. Academics and universities also use the platform for teaching and research [31].

Amazon Lex Developer

Powered by the same deep learning technologies as Alexa, Lex has a large number of built-in intents for common dialogue situations, as well as a built-in intent library, these can be customized. A new skill-specific utterance can be added to a built-in intent. Similarly the system can be set to prompt for missing or ambiguous information. Instead of entities, Lex is using so-called slots, which are not trained by concrete examples, but example patterns [2].
In Amazon's Lex a Dialog Model is created in which required slots are specified along with optional slots and intent confirmation prompts. The Dialogue Interface manages the slot-filling using the Dialogue State property to determine if additional actions are required in the dialogue or all have been completed [2]. Lex's bots can be not only text based, but voice-based as well, with this application, one backend can handle both text and voice requests.

Botkit

Botkit is designed to help people build bots for their needs fast and easy, without having to dig deeply into the base principals. It provides a unified interface for sending and receiving messages. Botkit's workflows are intuitive and organized in a clear and concise manner, it is well documented and provides an abundance of live chatbot examples. This way, it is very accessible to get started with and use furthermore. The core functionality is implemented in the form of event listeners.
Initially intended for Slack, it now has extended functionality to support connection to various messaging platforms. The framework has no NLP capabilities, but it can be resolved by connecting existing or custom-built services of NLP via middlewares. The library constitutes of core functionality and connectors that support different platforms and whose implementation varies depending on the platform it serves for [13].

Microsoft Bot Framework

The Microsoft Bot Builder is one of the most advanced public available machine learning solutions, it has its own Bot Builder SDK (Software Development Kit) that includes .NET SDK and Node.js SDK. The entire system consists of three parts: Bot Connector, Developer Portal, and Bot Directory. The ramework provides the Direct Line REST API, which can be used to host a bot in an application or website. It is open source and and supports automatic translation to more than

30 languages. It is possible to incorporate LUIS for natural language understanding, Cortana for voice, and the Bing APIs for search [31]. Microsoft provides different approaches to slot-filling, like FormFlow or Waterfall dialogue model, and also provides a range of methods for maintaining state, like user data or conversation data [2].
LUIS, which stands for Language Understanding Intelligent Service, is the application that allows the user's input to be understood, in order to build a model of a chat-bot, using intents and entities. All its applications are closed domain or content related. Active learning technology is possible, besides using pre-existing, world-class, pre-built models from Bing and Cortana [31]. Example phrases or utterances are supplied for the intents. The chat-bot developer can label the utterance examples with specific details. An utterance is user's input which is supposed to be understood by the chat-bot, conveying an intent as the user's goal. An intent may be mapped to many utterance variations, depicting actions the user is expected to perform. An entity represents relevant detailed information in the utterance [30].

RASA

Moreover, there is a popular open source alternative which is RASA. The platform is essentially a combination of Rasa NLU and Rasa Core, two python libraries. The former is in charge of interpreting messages and the latter decides what what action should take place. Both packages are well documented. This platform offers the same functionalities, without the cloud-based solutions and with the typical advantages of self-hosted open source software (adaptability, data control). It can either use MITIE or spaCy as ML backend[1]. It is an open source tool that runs locally and has HTTP API and Python support, intent classification, and entity extraction.
Rasa NLU is one of the solutions that facilitate building chatbots back-end. The framework is similar to NLP services, providing processing power on premises. Additionally, Rasa has Python interface to integrate the entity extractors directly into applications written in this language. It can also run as a service within other frameworks, exposing REST API endpoints. The framework is attractive for being of fast and easy development, in addition to its NLP service being controlled through deployment on premises [13].

Smaller Platforms

As stated before, with the increasing popularity of chatbots, many platforms have been created over the last years, some of them show promising features. Twyla learns from agent/customer live chats, blends machine learning and rule-based methods, answers questions, and deflects tickets. Msg.ai integrates with popular customer support offerings while leveraging intent models and tone classifications. It supports deep learning, interactive smart cards, and A/B testing. Reply.ai is a visual bot builder that easily leverages NLP engines wit.ai and api.ai for advanced use cases. The basic functionality of ManyChat allows for new users, sending them content, scheduling posts, setting up keyword auto-responses (text, pictures, menus), automatically broadcast RSS feed, without requiring coding and being free.
KITT.AI built its proprietary ChatFlow platform that enables users to create conversational agents, or smart bots, using a simple drag-and-drop interface that visually describes a dialogue and at the same time implements the flow that can be executed on the server as the dialogue is designed. The features of the platform are hotword detection, semantic parsing, NLU, conversational engine (multi-turn support), and neural network powered machine learning model. ChatFlow supports Alexa, Facebook Messenger, Kik, Skype, Slack, Telegram, and Twilio and is free.
It's Alive is a free Facebook page chatbot building platform. The core feature of the platform is

recipes. This enables automatic responses when users write specific keywords or phrases. If the developed chatbot has missed a keyword, it will be added to new or existing recipes. The team believes in decision trees and buttons that easily drive users towards the answer they are looking for [31].

2.2.4.1 Chosen Framework

RASA was the chosen platform to built the Conversational Assistant for Satellite Operations, mainly for its adaptability, possible integration with other applications, potential for code changing, constant evolving and improving which comes from being an open source framework additionally to the possibility of being deployed internally, as it is not cloud based. The main purpose of the framework is to make machine-learning based dialogue management and language understanding accessible to non-specialist. The design aims for ease of use, and bootstrapping from minimal initial training data. Rasa's architecture is modular by design, allowing easy integration with other systems. Rasa Core can even be used as a dialogue manager in conjunction with any NLU services. The code is implemented in Python, but both services can expose HTTP APIs to be used easily integrated by projects using other programming languages [8]. However, one of the downfalls of RASA is that it updates its contents quite frequently, so it is important to always check the used version and what are the features of it, or mistakes can happen.

2.2.5 Existing Bots

Initially, the approach to chatbots was an extension of the technique used in ELIZA. Having users interact with these applications primarily to engage in small talk. Today these systems occupy a different role, offering help and answering questions in the most diverse fields, depending on the operating domain. Any interaction with an app or web page can utilize a chatbot to increase the user's experience, with these programs clarifying any queries the clients might have. Bots are helpful explaining information which might not be clear or present on the website. This way becoming virtual assistants in everyday life, with applications such as customer service, mobile personal assistants, advertisement, entertainment apps, talking toys and even call centers [17]. Some applications of chatbots will be introduced hereafter.

Business matters require professional organizational assistance, and now chatbots can fulfill that necessity. The assistants might be programmed to keep track of work schedule and remind the user about any upcoming events, or even events overlapping. This type of bots is useful since it is very simple and can benefit many different users.

The chatbot may be representative of a company in conducting interaction with clients, this way taking on a more formidable task. The customer support workflows are mostly predictable and scripted, therefore easy to implement. The typical bot behavior algorithm is to receive the user's query, parse it for information, find the similar cases in the database, and respond with a prebuilt answers.

Using bots to support a team of developers is now very popular, as it dwells in the development environment and thus is constantly under improvement by software engineers whose requirements for quality are higher. However, the bot resolves a very strict set of tasks and accordingly does not require the complexity of commercial bots. These usually represent simple scorekeepers, guard development servers and report the commit information, or simple schedulers.

Many large news sources (WSJ, NYT), as well as technological outlets (TechCrunch, MIT Technology Review) share content in a convenient form of brief text messages via major platforms like Facebook Messenger. The principle behind the publisher type of bot is simple, it gathers subscription information from the user, schedules the delivery of relevant news, and handles simple user requests, like and searches.

Entertainment bots are still rare and serve a specific purpose, like managing reservations of events tickets in a dialogue-style workflow. Some can also provide an immersive experience of entertainment website via messenger.

Yet another popular and fast-growing use case for the bots is the assistance with travel. In this case, the customer-oriented chatbot strives to help people with the selection of the optimal transportation mode in a casual chatting kind of way. The travel chatbot is not only able to retrieve and confirm the booking information, but also notify about the times like check-in beginning and boarding, update on the status of the travel, and gather valuable feedback from the customers. [13]

In many situations Virtual Assistants provide a better user experience, whether for the constant availability, or for processing the requests instantly. A few examples of successful chatbots at use will be discussed hereafter, first options outside the space domain, and the last two in similar contexts as the project of this master's dissertation.

Alexa

Amazon's Alexa's concept started as a speaker that had some smart voice commands built in, Echo. However, The company quickly added more features and skills, even allowing it to connect with some of the most popular apps, and Echo became a whole system, and definitly pioneered the category. Today, Alexa is the most successful and most used virtual assistant [32].

Alexa can also be reached through mobile apps in addition to Amazon's hardware. With these interfaces, users can talk to Alexa to perform tasks such as searching the web or any connected systems and perform actions on them. Being integrated with the Amazon marketplace, Alexa is even able to perform hands-free purchases with queries like "Order [item] from [store] Now" to any kind of product, even groceries.

Google Assistant

The Google Assistant is available on phone, speaker, watch, laptop, television, car and smart display and connects with most daily used apps with over 1 million possible actions [33]. Google, as the most used search engine, benefits from its huge inventory, and capability to deliver answers to queries, which covers the base for a virtual assistant [32].

Google Home is the smart speaker powered by Google Assistant, similar to Amazon's Echo, with the goal of delivering a more personal experience. It aims to provide information to the user even before he asks for it, identical to the guessing made by the search engine, or even giving the user personalized recommendations based on the personal aspects of its everyday life.

Cortana

Microsoft's Cortana has the advantage of Microsoft being the dominant player in business productivity software for the last several decades. And with the notorious reputation of the brand, Nissan and Volkswagen wish to embed the technology in their cars. The assistant allows the interaction between the user, the system and its devices. It enables to search the web, find apps and documents, keep control of the calendar, among other tasks. In addition, Cortana also autonomously informs the user of details related with its preferences [34].

Siri

Apple's Siri was the first truly mainstream voice agent. It can make calls or send texts and connects to phone, watch, car, and television. Siri also performs everyday tasks, similarly to its competitors, and even anticipate what the user might need help with through your day. Homekit is Apple's own smart home products, which, integrated with the assistant, lets the user interact with its environment hands-free. Allowing to control of the smart appliances, check their status, or even do various tasks at once. The system performs these actions while still answering random queries and doing research, as characteristic of chatbots [35].

Mitsuku

Mitsuku, from pandora bots, is a four-time winner of the prestigious Loebner Prize Turing Test. It is widely considered the world's best, most humanlike, conversational chatbot, talking with millions of people monthly on Messenger, Kik, and the web. The bot is also available as a fully conversational character, with an avatar. The Mitsuku codebase can be licensed, along with othermodules, from Pandorabots and incorporated into other chatbot applications [36].

WolframAlpha

WolframAlpha is a unique engine for computing answers and providing knowledge. It using a vast store of expert-level knowledge and algorithms to automatically answer questions, do analysis and generate reports. The chatbot performs linguistic analysis, curated data, dynamic computation and computed presentation. Its domain is very wide, working similarly to a search engine, it includes areas such as mathematics, statistics and data analysis, words and linguistics, units and measures, history, physics and chemistry, finances, art and design, astronomy, music, medicine, engineering, geography, education, shopping, meteorology, sports, transportation and many more [37].

Geocento's Chatbot

Geocento is an online provider of Satellite and Drone imagery and image anaytics from several suppliers. The company has recently initiated the project of a natural language processing Chatbot applied to the satellite imaging business, using its in-depth knowledge of user cases. Still in designing the workflow, intents and entities associated with user's possible interactions, the aim is to facilitate the process of discovering and ordering imagery.
Geocento wants to improve the experience for users who can be overwhelmed by the complexity of the first contact with selecting and ordering imagery. Using their own API as source of information within the chatbot domain, the business hopes to ensure that any client's doubt can be enlightened with a chatbot inquiry as easily as possible [38].

CIMON

Airbus, a global leader in aeronautics, space and related services, is developing an astronaut assistance system, CIMON (Crew Interactive MObile CompanioN). In cooperation with IBM, using Watson AI technology from the IBM cloud, the company is creating an AI-based mission and flight assistant for astronauts. CIMON is designed to support astronauts in performing routine work or offering solutions to problems, being able to learn with its AI network. Face, voice and artificial intelligence, are given to the assistant with the intent of it becoming a genuine 'colleague' on board for the rest of the crew.

CIMON's aim is to make work easier for the astronauts when carrying out every day routine tasks, help to increase efficiency, facilitate mission success and improve security, with the capacity to act as an early warning system for technical problems. The technology demonstrator, which is the size of a medicine ball and weighs around 5 kg, will be tested aboard the ISS during ESA's Horizons mission.
Airbus believes social interaction between people and machines, astronauts and assistance systems equipped with emotional intelligence, could play an important role in the success of long-term missions in the future. Airbus' developers are also convinced that this project could eventually also find use in hospitals and social care. First the system must learn to orientate itself and move around, as well as be trained to recognize its human partners, in addiction to the NLP and AI development needed for any bot [39].

2.3 Natural Language Processing

In order to build a Conversational Assistant, the obstacle for computers is not just understanding the meanings of words, but understanding the endless variability of expression in how those words are collocated in language use to communicate meaning [40]. Natural language processing (NLP) is a theory-motivated range of computational techniques for the automatic analysis and representation of human language that aims to solve this problem [41].
NLP seeks the understanding of how human beings use language in order to develop computer systems capable of comprehending and manipulating natural language to perform required tasks. Some applications of this research field include Machine Translation (MT), text processing and summarizing, speech recognition and expert systems [42]. Since NLP strives for human-like performance, it is appropriate to consider it an Artificial Intelligence discipline. Which is the creation and analysis of intelligent agents (software and machines) and can be implemented in nearly each and every sphere of work [18]. Being its goal full Natural Language Understanding, the System should be able to paraphrase an input text, translate it into another language, answer questions about the contents and draw inferences from it [16].

2.3.1 History of NLP

NLP research has evolved from the era of punch cards and batch processing, in which the analysis of a sentence could take up to seven minutes, to the era of Google and the likes of it, in which millions of webpages can be processed in less than a second [41]. Natural language processing studies began in the 1950s, as the intersection of artificial intelligence and linguistics, and MT was its first computer-based application. Early simplistic approaches, such as word-for-word machine translation, exposed the difficulty of language analysis when defeated by homograph words and metaphors. Natural language's vastly large size, nonrestrictive nature, and ambiguity led to problems when using early standard parsing approaches that relied purely on symbolic, hand-crafted rules.
The conclusion that NLP must ultimately extract meaning ('semantics') from text, resulted in the birth of statistical NLP in the late 1990s. With statistical-frequency information intended to disambiguate, fewer, broader rules replace numerous detailed rules. A statistical parser determines the most likely parse of a sentence depending on context. By learning with copious real data, utilizing the most common cases, statistical approaches show quality results in practice. This way, they improve with abundant and representative data [43].

Today, the most popular NLP technologies view text analysis as a word or pattern matching task. Trying to ascertain the meaning of a piece of text by processing it at word-level. However, to achieve its main goal NLP systems must gradually stop relying too much on word-based techniques while starting to exploit semantics more consistently and, this way, make a leap from the Syntactics to the Semantics approach. The latter focuses on the intrinsic meaning associated with natural language text. Rather than simply processing documents at syntax-level, semantics-based systems rely on implicit denotative features associated with natural language text, hence stepping away from the blind usage of keywords and word co-occurrence count [41].

2.3.2 Speech Processing

There are multiple methods or techniques to accomplish language analysis. It is thought that humans normally utilize multiple types of language processing when producing or understanding it, and each level of linguistic analysis convey different types of meaning. Besides, texts used by humans to communicate to one another can be of any language, mode, genre and even be oral or written [16].

A current Natural Language Processing System extracts simpler representations of the textual information that describe syntactic or semantic aspects [44] and generates a Subject vector representation of the text, which may be an entire document or a part there of [45]. To extract meaning from the users input NLP must accomplish four crucial tasks Tokenization, Part-Of-Speech (POS) tagging, Named Entity Recognition (NER) and Dependency Parsing. This way, NLP enables computers to perform a wide range of natural language related tasks at all levels, ranging from machine translation to dialogue systems [46].

The first step in NLP is to identify tokens, that represent a unit of text such as a word, sentence, or document, for subsequent processing. The design of a tokenizer for a given set of texts should involve a detailed analysis of all the linguistic phenomena. Such as the use of punctuation, quotes and hyphens, upper-case letters at the beginning of sentences, numbers, acronyms, emphasis marks or even accents [47].

POS aims at labeling each token with a unique tag that indicates its syntactic role and generate a tree of the text constituents. It is a fundamental problem in the sense that most NLP applications need some kind of POS tagging previous to construct more complex analysis [48]. This type of classifiers is trained with a large tagged corpus and automatically derives information from it, which is subsequently used to categorize unknown tokens. Together with the interpretation of context, preceding and following tagged tokens and associations of tokens [49].

NER labels elements of the sentence into categories (entities), assigning to each token a unique tag prefixed by an indicator of the beginning or the inside of an entity. They are also trained on databases and can use a combination of classifiers and gather their features, included tokens, POS tags, or prefixes and suffixes, to correctly classify the tokens [44].

Finally, the dependency parser establishes dependencies between individual words from the resulting POS tree, improving it, as demonstrated in Figure 2.4. These word-to-word interactions should provide full syntactic and semantic analysis. A dependency parse represents the syntactic structure of a text in terms of binary relations between tokens. Dependencies categorize types of relations including nominal subject, direct object, indirect object, auxiliary verb, prepositional phrase, object of preposition, determiner, noun compound modifier, adjectival modifier, adverbial modifier and punctuation [50].

Combining these concepts, enables the retrieval of information from natural language and its organization into labeled segments to be perceived by computers. The efficiency of the process is

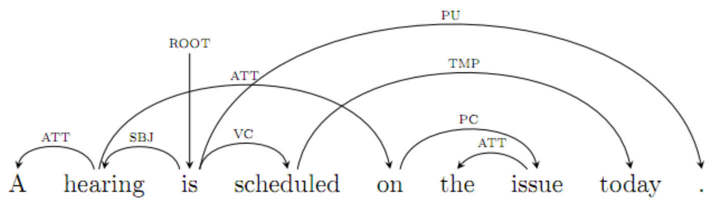


Figure 2.4: Exemple of Dependency Parcing from [4].

associated with the speed of it and the accuracy of its results, specially when handling a large data set [44].

One of the main problems that makes parsing so challenging is that human languages show remarkable levels of ambiguity. A natural language parser must somehow search through all of the alternatives, and find the most plausible structure given the context. Figure 2.5 shows a sentence with more than one possible dependency parses. The first option (on the left) corresponds to the (correct) interpretation that Alice is driving in her car. Contrarily, the second (on the right) corresponds to the interpretation where the street is located in her car, impossible in a real scenario, but possible in a purely linguistic way. The ambiguity arises because the preposition "in" can either modify drove or street. This example is an instance of prepositional phrase attachment ambiguity [5].

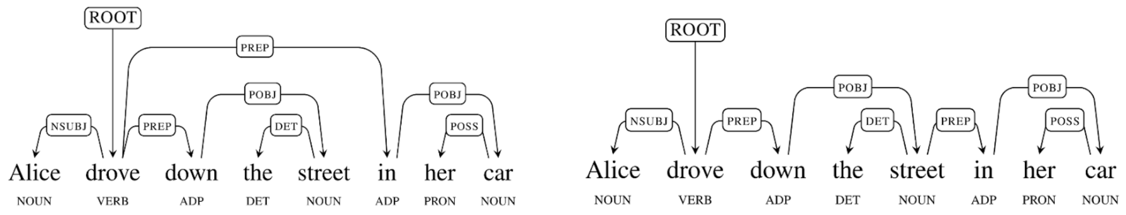


Figure 2.5: Sentence with two possible dependency parses from [5].

2.3.3 NLP for Chatbots

Natural Language Processing is not usually considered a goal in and of itself, except perhaps for AI researchers. For others, NLP is the means for accomplishing a particular task [16]. Conversational applications, or Chatbots are very prolific examples of these systems. Simpler bots scan for keywords within the input, and pull a reply matching the most keywords, or the most similar wording pattern, from a database, while others are sophisticated natural language processing systems. Whichever the case, these advanced NLP applications require deep semantic interpretation.

Artificial Intelligence in machines is a very challenging discussion, as it involves the creation of machines which can simulate intelligence. As the aim of a Chatbot is to interact and stimulate a conversation with a person without compromises, it should receive an input sentence from the user and identify what is meant to say to then answer accordingly and, if required, take the necessary actions to assist the user. In this sense, NLP models are utterly necessary to interpret the text [16].

2.4 Knowledge Base

A Knowledge Base (KB) stores complex structured or unstructured information, being the combination of an ontology and instances of the classes in the ontology [51]. For machines to be able to understand user technical questions, a background Knowledge Base plays a critical role, as the initial ”intelligence” of the system. Many existing KBs are about lexical knowledge or open domain facts. A Knowledge Graph is a Knowledge Base with its data connected in a grap-like way, creating a database similar to a tree. It combines characteristics of more than one data management paradigms, which is ideal for AI systems, as a representation of data that can be directly mapped to a logical form [7].

Knowledge Graph is a means of storing and using data defined as a set of concepts connected by relationships where the concepts form the nodes of the graph and the relationships are the labeled edges. An instance of edge is a triplet of fact, denoted as (h, r, t) for: head entity, relation, tail entity. Forming a large network of entities, their semantic types, properties and relationships. This architecture is further explained in the next Subsection. KGs acquire and integrate information into an ontology and apply a reasoner to derive new knowledge. When using knowledge graphs, the availability of relevant knowledge, both in terms of applicability to the task and to the domain of interest is the prominent concern. Which relates closely to the way it is created. [52]

This type of graph has been used extensively in Information Retrieval, Recommendation Systems and Natural Language Processing. The primary benefits of a graph are how easy it is to connect new data items as they are injected into the database and to link remote parts of the domain. Besides, a graph is one of the most flexible formal data structures, so it enables mapping other data formats to graphs using generic tools and pipelines. Furthermore, the meaning of the data is encoded alongside the data itself in the graph, in the form of ontology, which facilitates the submission of queries in a style much closer to natural language, using a familiar domain vocabulary. A knowledge graph is self-descriptive, this way providing a single place to find the data and understand what it is. Data is typically expressed in terms of entities and relations familiar to those interested in the given domain, which narrows the communication gap between stored data and users. The result is smarter search and more efficient discovery. Being actual graphs, KGs allow for the application of various graph-computing techniques and algorithms which add additional intelligence over the stored data [53].

2.4.1 Different Models

In the past decade, there have been great achievements in building large scale KGs, however, the general paradigm to support computing is still not clear. The fact that a knowledge graph is a symbolic and logical system while applications often involve numerical computing in continuous spaces and how difficult it is to aggregate global knowledge over a graph, are some of the difficulties causing this problem. Some different models are briefly introduced next:

- TransE - the relation between entities is presented as a translation vector that connects the pair of embedded entities in a triplet with low error.
- Unstructured - considers the graph as mono-relational and sets all translations r = 0, this way, this simplified version of TransE, can not distinguishing different relations.
- Distant Model - entities in a relation have two independent projections, revealing weaknesses capturing relations.

- Bilinear Model - models second-order correlations between entities, making each component of each entity interact.
- Single Layer Model - uses nonlinear transformations by neural networks.
- NTN - is an extension of the Single Layer Model, that considering the second-order correlations into nonlinear transformation. However, the model's level of complexity makes it difficult to apply to large scale graphs [54].
- DGEM - is an extension of the DecompAtt, uses OpenIE to construct syntactic structures (as graphs) for hypothesis, and computes node attention based on the premise text [52].

2.4.2 Other Knowledge Graphs

Knowledge graphs have become very important resources to support many AI related applications, and many openly available external knowledge sources options are now available. Such as Freebase, Google Knowledge Graph, DBpedia, Wikidata, Cyc, YAGO, ConceptNet and WordNet which will be introduced in this Subsection.

Freebase

Freebase was a KG created by Metaweb Technologies, Inc. in 2007. Exceptionally, it provided an interface that allowed end-users to contribute to the graph by editing structured data. Freebase used a proprietary model that also stored complex statements. Now, it is integrated with the most prominent KG options. An example subgraph from this KG is presented in Figure 2.6.

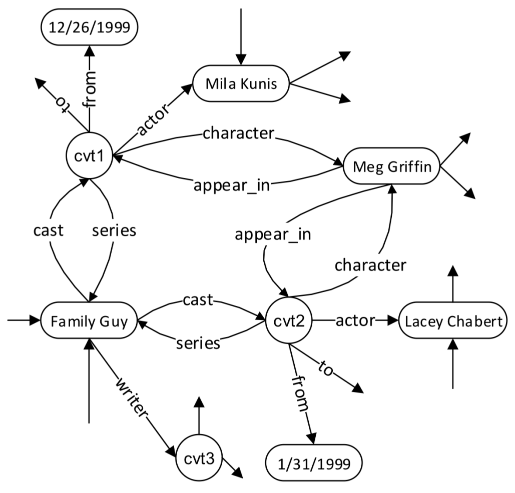


Figure 2.6: Freebase subgraph of "Family Guy" from [7]

Google Knowledge Graph

One of the best know and most used KGs of today is the Google Knowledge Graph. It exploits authoritative sites, such as Wikipedia, CIA World Factbook and Freebase to gather data, creating a very large database. Google's graph interconnects facts, people and places to create more accurate and relevant search results. It basically connects users' keywords with all the available data, giving the users all the information interconnected in one way or the other. Semantic search is considered to produce better relevant search results, and entity indexing catalogs the related entities to produce factual information about each [55]. Essentially the KG enhances the search engine's results based on semantic information from several sources. It also has an associated Search API that can be used to make requests to the KG.

DBpedia

DBpedia is the most popular and prominent Knowledge Graph. The project was initiated by researchers from the Free University of Berlin and the University of Leipzig, in collaboration with OpenLink Software, having its first release in 2007. DBpedia is created from automatically-extracted structured information contained anywhere in the Wikipedia, even from infobox tables, geo-coordinates, and external links. This KG is used extensively in the Semantic Web research community, but is also relevant in commercial settings being used by companies to organize their content [51].

Wikidata

Wikidata is a project of Wikimedia Deutschland which started in 2012. The aim of the project is to provide data which can be used by any Wikipedia project, including Wikipedia itself. This KG not only store facts, but also the corresponding sources, with the intent of checking the validity of the information. Labels, aliases, and descriptions of entities are provided in more than 350 languages. The graph is a community effort since users collaboratively add and edit data [51].

Cyc

The Cyc project started in 1984 as part of Microelectronics and Computer Technology Corporation. The aim was to store millions of common sense facts. While the focus started on inferencing and reasoning, today its centered on human-interaction. A smaller version of the KG called OpenCyc was released under the open source Apache license for this purpose. A larger version containing more facts than OpenCyc is ResearchCyc, published for the research community [51].

YAGO

YAGO, standing for Yet Another Great Ontology, has been developed at the Max Planck Institute for Computer Science in Saarbrücken since 2007 and was released in 2015. It comprises information extracted from Wikipedia, WordNet, and GeoNames [51].

ConceptNet

ConceptNet originated from the crowdsourcing project Open Mind Common Sense, launched in 1999 at the MIT Media Lab. It has since grown to include knowledge from other crowdsourced resources, expert-created resources, and games with a purpose. This graph is used to create word embeddings, representations of word meanings as vectors, which are free, multilingual, aligned across languages, and designed to avoid representing harmful stereotypes. ConceptNet may be more appropriate then other KGs if common sense reasoning is required [56].

WordNet

WordNet is a lexical database of english words grouped in synsets (word-sense synonym sets) connected by semantic relationships such as hypernym, hyponym, and antonyms [52].

2.4.3 Knowledge Graph for Mission Operations

To generate a knowledge graph for satellite mission operations, it is necessary to define the classes and relations of entities for all the table files associated with it. Thus allowing potentially interrelating arbitrary entities with each other and covering the whole mission's domain. While graph structures represent the existing hypothesis, external knowledge sources are not used in this model, as the focus is purely on the satellites' domain. Input is specific knowledge derived from the knowledge base using the given premise and hypothesis.
As already affirmed, knowledge graphs have a flexible structure, the ontology can be extended and revised as new data arrives. This makes it convenient to store and manage data in use cases where regular updates and data growth are important. A knowledge graph can support a continuously running data pipeline that keeps adding new knowledge to the graph, refining it as new information arrives. Additionally, the KG can answer what it knows, and also how and why it knows it. All these features are very important for a mission operations' knowledge graph has it needs to be constantly be updated. Besides, as every mission has a different domain, different KGs need to be generated frequently, specially in the beginning of the implementation, however, some of the entities and their relations are common among missions and satellites, this way the process of creating a different knowledge graph for a new mission can be simplified.

Chapter 3

System Architecture

This Chapter focuses on the architecture of the chatbot, introducing all its components and explaining their integration. The generation of the knowledge graph from the database (DB) and the development of the conversational assistant will be further detailed, being the elements of the project developed in the scope of this dissertation.
First, the overall structure of the system is explained in Section 3.1, and how all the elements together make an answering bot. Section 3.2 details how the knowledge graph basis is constructed and how it is generated for a generic mission. Finally, Section 3.3 describes how the bot was created through the Rasa platform, providing examples of how it works and must be implemented.

3.1 System Overview

As stated in Chapter 1, the prototype stage of Karel's development, which is the object of this dissertation, should be able to answer simple questions related to elements of a satellite. With the aim of assisting SOE to control and monitor spacecraft missions in a more efficient way and even facilitate decision making regarding unexpected events. This type of chatbot can be referred to as a Question & Answering system (QA). Which are systems that allow a user to ask a question in everyday language and receive an answer quickly and succinctly, with sufficient context to validate the answer [57]. To do so, Karel must interconnect the satellite's database with the user after decoding a question. Being a complex project, the system is divided in five distinct modules, independently built on each other, that together form the stack of the chatbot, as Figure 3.1 illustrates.

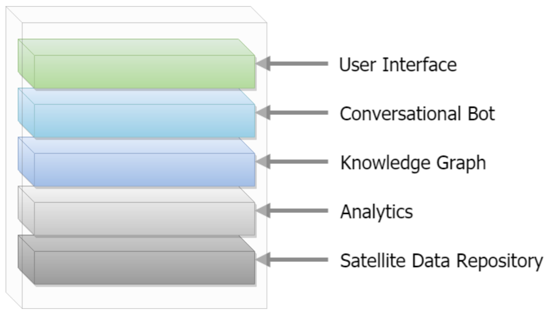


Figure 3.1: Karel system stack.

A software stack is a group of programs that work in sequence towards a common result. This way, the program is structured in layers, from simple data in the bottom to more complex layers on the top. With well defined interfaces between layers, it is easy for the project to be distributed by more than one developer who can focus on one or two aspects of the chatbot using lower layers created by someone else to run their part on, then integrating them to constitute the final program. This approach is advantageous resulting in an easily developed system, able to be implemented on different domains. The last aspect is of great very importance, seeing that the program must be adaptable for different missions. However, the possibility of easier scalability is also very appealing for the bot, since it may cover more areas of mission operation

The first layer of Karel is the user interface, which must be appealing and easy to use. This element is built on the conversational assistant itself, the part that does the natural language processing and answer creation. This model is developed based on the information previously gathered by the knowledge graph. In its turn, the KG is organized from the data analysis covered by the data analytics. Which, finally, is modeled according to the mission database. The satellite repository is the basis for the whole system, as it is where the information sought is collected. Every element runs on top of the other, so establishing the software's stack is important as it allows to visualize all the dependencies and plan the development of each, focusing on its integration. The stack can be considered the foundation of the system, however for a chatbot to be successful, the user should not identify the different modules, but receive it as a single program. Karel's components will be further explained hereafter.

3.1.1 Data Archive and Analytics

The domain of the developed chatbot is satellite missions, which mean the whole system must be build on the satellite data repository, which contains the Satellite Database and the Data Archive. This database is a group of files with different purposes, including telemetry (TM) and telecommands (TC) data, monitoring parameters and all the system's architecture definitions and considerations. For Karel to accomplish its goal, some packets of the Database are specially important to map the KG. The Monitoring Data files contain information related to the definition and processing of monitoring parameters, telemetry packets and monitoring displays, which is essential for understanding all the mission's components. The Commanding Data packets describe the contents of a command, defining its characteristics and behavior once released to the spacecraft. The Data Archive contains all historical data produced by the mission. Connecting to this database allows the access of reporting data, events and activities, as well as the timestamps associated with each one, necessary for the bot's answers. This component is access through the Analytics module, which is comprised of three components, Karel-DB, Karel-Analytics and Karel-API, as Figure 3.2 shows.

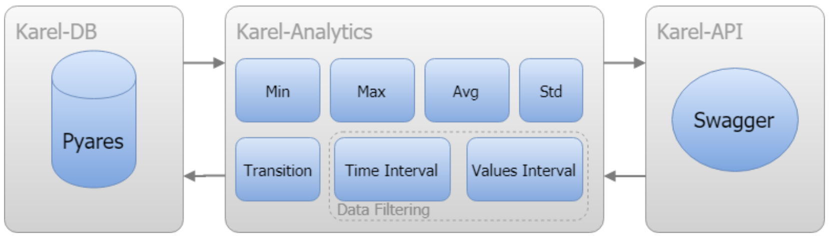


Figure 3.2: Analytics module architecture.

The Karel-DB is the component that ensures the connection with databases. Currently it accesses information from pyares, the used by ESA. However, it may be adapted to access various forms of data from different types of databases.

The second component, the Karel-Analytics, is in charge of the calculations and analyses provided. Currently, the Analytics covers the main statistical analysis, namely the calculation of minimum, maximum, average and standard deviation of a parameter's values. The module also calculates the number of distinct transitions of states, for instance on, off and stand by for a parameter. Another of the program's methods is Data Filtering that, as the name implies, allows to return filtered data for specific periods, using timestamps, or value ranges.

The third component is Karel-API. Which is a RESTful API serving as the interface that enables the connection between the bot's systems. A RESTful API was chosen as it is the most commonly used and brings, among others, the following benefits:

- High flexibility,
- Usage of http for obtaining data,
- Handling multiple types of calls,
- Support for different types of data formats.

For the prototype, the data is obtained though http and the responses are generated in json format. Furthermore, Swagger is chosen as the platform for the API. This platform helps with rules and specifications which assist with the documentation of the API by making it easily understandable, supports tools such as User Interface that helps users save time and test their methods and Swagger Editor that enables to find errors in documentation. Finally, it is easily integrated other tools and ensures better management of the API.

3.1.2 Bot and Knowledge Base

The module in charge of the conversational side of Karel comprises natural language processing as well as dialogue flow and management. This is the core of the chatbot, assuring its definition as a conversational agent, which should interact similarly to a human. The knowledge base on the other hand guarantees the connection between natural language words and the parameters they try to convey. This means both modules must be in total agreement concerning the entities classified by the bot and the strings used to associate the KG with the database parameters. These systems will be further detailed in the next Sections. However, for the bot to receive the user's question and for the user to receive the bot's answer, an interface is necessary.

3.1.3 Human-Bot Interaction

The first level of Karel is human-to-bot interaction, which happens through a web interface and ideally sends rich and interactive content to the user and has the option of guiding users to other relevant information, increasing the overall support provided by the cognitive service. Mattermost was the selected software, an open source team messaging designed to efficiently keep data and operations under IT control. For the bot, this interface assures:

- Security: it keeps vital communications within private environments, guarantying data stays on encrypted servers controlled by VisionSpace, which is vital when dealing with confidential satellite information.
- Configurability: its deployment adapts to the systems' needs, and other existing modules, with the option of bringing all the team communication into one place, while adding the bot to the chat.
- Scalability: it offers the possibility of integrating all the satellite team on the same server, deploying with horizontal scaling and advanced performance monitoring on a single infrastructure.

In addition to all these features, Mattermost is able to send messages in different formats to the user, such as tables and graphs. Which is a characteristic that adds a lot of value to a bot aiming at data monitoring and control. Tables can be sent revealing parameter names and definitions directly from the database. Generating them requires simple user messages identifying the required file and patterns for the desired content. Sending plots adds context to the answer and allows visual perception of the element's behavior.

Rasa already offers a solution to be associated to Mattermost. To connect the conversational AI with the users through Mattermost, input channels are defined in the `rasa_core.channels` module, passing the input channels and the port as arguments for the `handle_channels`. The code available for connecting to the software also needs some credentials provided in a specific yaml file and the creation of a Mattermost-compatible webserver. After adding a webhook with the bot channel and name on the team site, the callback url, where the webhook runs, a user and a password for the bot, must be identified in the yaml file.

Asking Karel a question through the team digital workspace is done the same way as talking to any colleague. All it takes is a "mention", which is made by preceding the Mattermost username with the symbol. This way, "`@Karel Hello!`" is a message handled by the chatbot, that will trigger an answer from it.

3.1.4 Integration

For Karel to be successful and efficiently assist the Spacecraft Operations Engineers all the modules must be connected and able to request each other what they may need to function. This interaction is displayed in Figure 3.3 and now detailed. The user accesses the bot through Mattermost and once a question is sent, it is parsed by the NLU model. Which classifies the message and sends the results to the Dialogue model.

According to the intent identified by the natural language processing, the dialogue model can send a template message to the web interface, if a simple interaction is made or the intent is not confidently classified. If this is not the case, the dialogue model triggers the first action, which queries the Knowledge Graph using the entities parsed by the NLU model with the objective of finding the parameters corresponding to it. If some information is missing, the bot must query the user to gather the necessary elements.

Otherwise, the KG will search the satellite database to find parameter keys matching the request, attaching its units and definitions. If no match is found the bot will send a message stating that to the user and a new question can be made, if one or more are found, a list must be returned to the chatbot, which will trigger a second action. This code will take the obtained parameter and the time period from the user input and make an API call to the analytics method corresponding to the user's query type.

The API then fetches the telemetry samples from the archive for the asked information. If the request makes sense and an answer is found, it is returned to the bot. Which must organize an answer including the entities in the user's input, to guarantee the right request is being fulfilled, the parameter key and its description for the user to know exactly what element it is receiving and finally the value or group of values. In the case of a numeric parameter, a plot should be added for context and validation purposes. This should be sent through Mattermost. If instead the the analytics does not find a valid answer to the query, this information is passed to the bot which should then answer negatively to the user. The user is then ready to analyze the output answer on the web interface and take action or make decisions accordingly, or input a new query to the bot if necessary.

For Karel to deliver realistic answers, all the systems must work together and all the connections must be well made. This is something that must be assured by the conversational bot. The NLU model must match the user messages' entities with strings that can be later parsed by both the KG and the Analytics module. The Dialogue model has to process the intent and chose the right

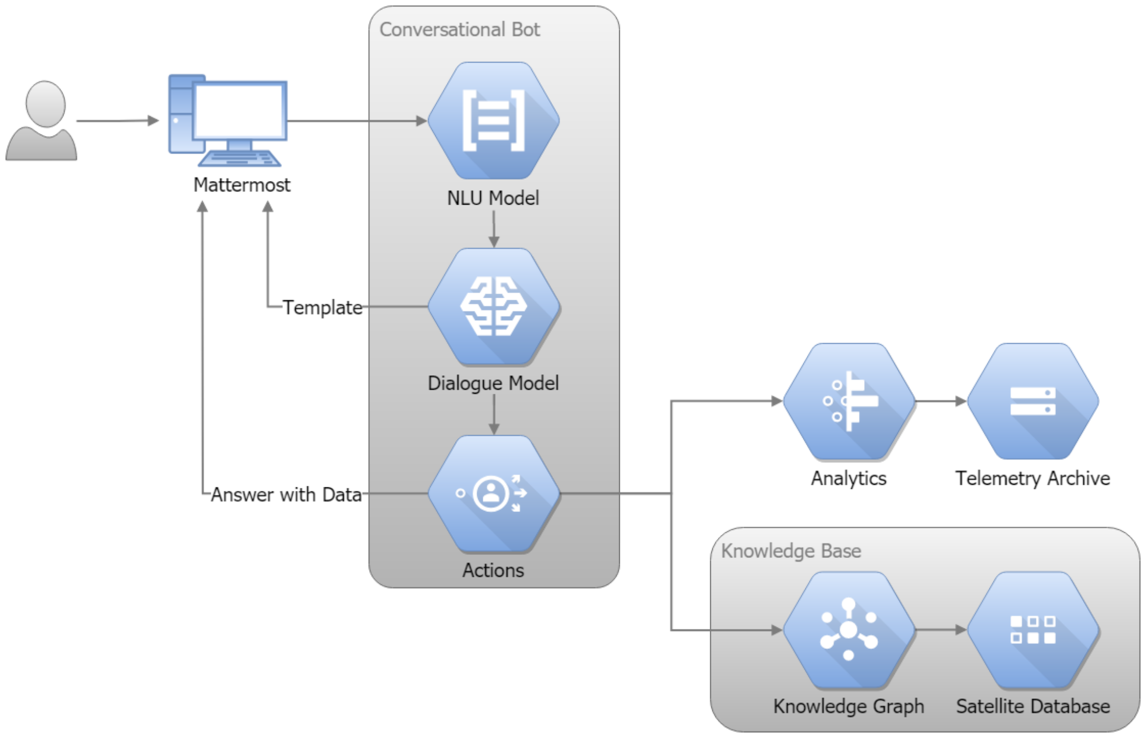


Figure 3.3: Karel's architecture.

action to take, as well as know what to deliver the user in many different possible cases, in addition to making the accurate API call. This construction of this important module will be explained next, followed by the description of the process for the generation of the knowledge Graph. Both these modules must correspond to each other when created for a determined mission, specially naming the elements and establishing the relations between them.

3.2 Knowledge Graph

A Chatbot requires structured data, not just tables with numbers and strings, as are present in the mission and satellite description database, but knowledge of the relations between the different objects and even object's additional information. For the project of this dissertation, the Knowledge Graph is essentially a description of entities and their interrelation in the satellites' universe, organized in a graph, defining classes and relations of entities as a scheme for all the table files associated with the mission. The KG reflects the information that can be queried by the user, mapping the complex mission's information and data. Although some parameters are common to all missions, each has different specific properties and systems. This means the Graph must be a continuous work-in-progress and should be refined and extended with every new mission. Building the Knowledge Base for satellite operations requires:

1. Build the mission ontology from mission documentation;
2. Generate the graph from the ontology and the mission database.

To build the mission ontology, the information model should be based on the European Cooperation for Space Standardization (ECSS) Space System Model (SSM), dividing the collected data accordingly in System Element, Reporting Data, Event and Data, and extending its knowledge with:

- Relations (predicates): that model the correlation between System Elements, Reporting Data, Events and Activities;
- Data Units: that associate the physical units to a given Reporting Data;
- Data Types: that model the type of data (are for instance numeric or enumerated). Different data types support different types of queries;
- Queries: that link the type of queries provided by the data analytics engine (can be for example minimum, maximum or average);
- Database Definition: that connect database entries to vertices in the graph.

3.2.1 RDF

Most successful large semantic databases are represented in Resource Description Framework (RDF), the World Wide Web Consortium (W3C) standard for Semantic-Web data, essentially a language that you can use to build knowledge graphs The core structure of the abstract syntax is a set of triples, each consisting of a subject, a predicate and an object. All things being described by RDF expressions are called resources. A specific aspect, characteristic, attribute, or relation used to describe it is a property, which has a specific meaning, defines its permitted values, the types of resources it can describe, and its relationship with other properties. A specific resource, a named property and the value of that property for that resource is an RDF statement.
This framework includes a standard syntax for describing and querying data, which is crucial when building a knowledge graph for a QA system [58]. Besides, it can be efficiently implemented and stored compared to other models requiring variable-length fields that would require a more cumbersome implementation due to it being made of triples. Further, this architecture facilitates data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed [59]. This is a very useful aspect remembering that each mission will have to generate a specific KG extended from a common base knowledge. This simple model allows structured and semi-structured data to be mixed, exposed, and shared across different applications. The scheme or description of the collection of classes can itself be written in RDF language. The RDF data model provides an abstract, conceptual framework for defining and using metadata and metadata vocabularies.
Thus RDF was the chosen software to generate the bot KG using RDF Schema, a semantic extension. This class and property system is similar to the type systems of object-oriented programming language, but describes properties in terms of the classes of resource to which they apply. One benefit of the RDF property-centric approach is that it allows anyone to extend the description of existing resources, with the domain and range mechanisms.

3.2.2 Domain Ontology Model

The core model, the ontology, is encoded in two RDF N-Triples files, ssm.nt and satops.nt. The first file gathers generic mission operations information based on ECSS SSM. It organizes the information in classes, predicates, data unit, data type, queries, model and database definitions. Classes are divided by the SSM standard definitions relevant for monitoring and control, which are:

- System Element: the elements of the space system resulting from the functional decomposition. From the highest level downwards, these are progressively: system, subsystem, set, equipment or software product, assembly, part (hardware) or module (software).

- Event: an occurrence of a condition or group of conditions of operational significance. Widely used within the space system to trigger the execution of functions.
- Activity: a space system monitoring and control function implemented within the Electrical Ground Support Equipment (EGSE) or mission control system. It can be implemented as a telecommand either to the space segment or to the ground segment, a procedure, an operating system command or any other command type that is specific to a given implementation of the space system.
- Reporting Data: information that a system element provides, irrespective of how this information is used. It can comprise measurements which reflect the state of the associated system element or an output product whose purpose is to be used by another system element. Reporting data can have different representations depending on its life cycle within the space system. Data units, as the name suggests, define every existing physical units in the satellite's reporting data.

Different data types implement appropriated types of queries, which must be performed by Karel's analytics module:

- Numeric: value, maximum, minimum, average, standard deviation and time interval.
- Enumeration: value, count, transition and time interval.
- Text: value.
- Event: value, count and time interval.

Some queries are intelligible, other may need further clarification, count refers to the number of elements in a system, transition identifies the number of status changes of an element from one value to a another, time interval defines when an element's data was between two values. The model label attaches models to predict the behavior or effects of some parameter. While data base definition is a string with the description of a parameter. Predicates describe the relations between two elements of the graph, connecting all the nodes, and are:

- Is part of: relating any class with a specific system element;
- Is an instance of: connecting some element with a class;
- Implements: between data and query types;
- Has type: associating reporting data with its type;
- Has unit: correlating reporting data with its unit;
- Has model: linking data with its model;
- Has database definition: relating data with its definition.

The other file of the ontology model, satops, extends the previous explained one with further details specific of satellite operations, common to all satellites, such as telemetry parameters, labeling the possible types of reporting data with data type and units associated with it, as well as generic reporting data, and events, similarly labeling types of events and generic ones. Each mission extends the base model to fit its design, generating the final KG. Generating the complete Knowledge Graph is a semi-automated process that maps the mission database to the pre-created ontology.

3.2.3 Graph Generation

For mission control at ESOC, ESA developed and now operates the Satellite Control and Operation System 2000 (SCOS-2000), a generic satellite Mission Control System (MCS). SCOS-2000 (S2K) is configured by the Mission Information Base subsystem (MIB), a set of ASCII files containing mission-specific data. This means that to create the KG adapted to a specific mission it is essential to access some tables from MIB static data. The parameter characteristics file, named pcf, contains the definition of monitoring data. It specifies the characteristics of monitoring parameters, telemetry packets and monitoring displays. The packet identification file, called pid, contains the definition of TM packets and the correspondence with the source packets generated by the spacecraft. It is used in order to identify the incoming packets and enable their further processing. The first of these two Monitoring Data files is used to map the Reporting Data for the Knowledge Base, the second one for the Events. From the Commanding Data files, the command characteristics file, named ccf, which defines the commands and the command sequence file, called csf, that defines the command sequences assist in mapping the Activities.

To extend the ontology and generate the complete Knowledge Graph for a mission operation, regular expressions are used to do the mapping of the satellite database using SPARQL. Which is a semantic query language for databases able to retrieve and manipulate data stored in RDF format, allowing for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns. The results of SPARQL can be results sets or RDF graphs. The patterns can match the information of a given file column, or a combination of patterns appearing in several columns. Patterns work similarly to python regex, so coding the mapping is accessible.

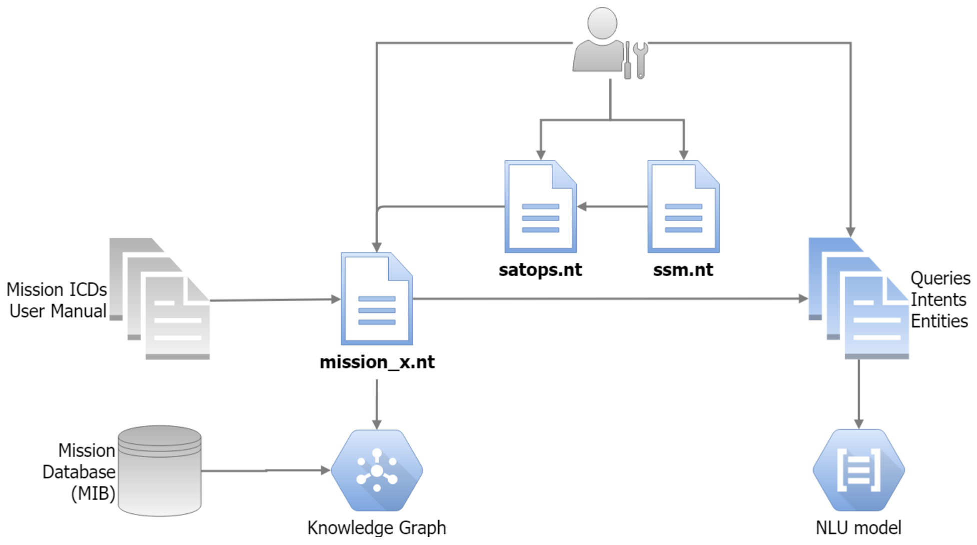


Figure 3.4: Karel's knowledge graph development.

The Knowledge Graph is encoded in RDF N-Triples file as well as the core model and at runtime is queried using SPARQL, finding the requested parameters by the user. Figure 3.4 summarizes all the process of generating the mission KG. The developer must build the satops and ssm rdf files, which will constitute the core model of the a generic mission knowledge graph and should not be built more than once. For a chosen mission, this ontology must be gathered with the mission documentation, including Interface Control Documents (ICDs) and the User Manual, thus creating the mission particular RDF file. Together with the MIB tables, this file generates the knowledge graph, that is then queried by the bot and using SPARQL finds the most appropriated parameters that will be sent to the analytics module. The satellite mission's RDF file is also used by the developer to create the possible queries, intents and entities to train the NLU model on. This way, Karel will process the queries and send them to the knowledge graph in a way that it can understand and query the mission database.

3.3 Rasa Framework

As previously explained in Chapter 2, Rasa was used for the development of this dissertation's project, an open source python library for building dialogue systems. The selection is primarily based on the benefits of it being an open source software allowing any necessary customization of models, in addition to not being cloud based as most other options, granting the option of its deployment internally, not sharing any data or passing it though a third party. The framework is divide in two tools, Rasa NLU and Rasa Core. The first, as the name suggests, assists with natural language understanding, interpreting the user's message, the latter manages the dialogue, determining what should be the next action of the bot. This way Rasa assures the system holds a contextual conversation with the user. Figure 3.5 shows the basic architecture of Rasa, and the process is as follows:

1. A message is received and passed to an Interpreter to extract the intent and entities, this is the only task preformed by Rasa NLU;
2. The Tracker maintains conversation state, after receiving the notification of the new message;
3. The policy receives the current state of the tracker;
4. The policy chooses the next action;
5. The tracker logs the chosen action;
6. The action is executed [8].

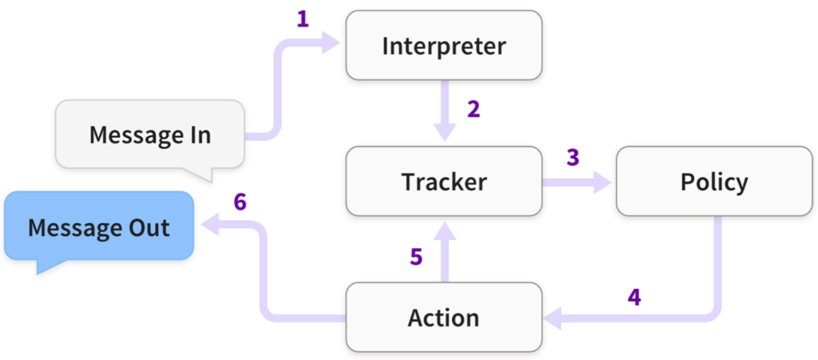


Figure 3.5: Rasa simple architecture from [8].

To build a bot, both Rasa NLU and Rasa Core models need to be trained, to learn .This Section will explain how the platform was applied to the satellite mission operations domain integrated in the bot's architecture. Foremost, it will be described how natural language processing is achieve, as it is the base for the bot's logic, then the agent in charge of conversational flow will be considered.

3.3.1 Rasa NLU

RASA NLU is the part of the open source platform in charge of intent classification and entity extraction wherewith is possible to implement a personalized human language parser, using its

NLP and ML libraries. The interpreter is responsible for parsing messages and transform it into structured output. For text classification, sentences are represented by pooling word vectors for each constituent token, using pre-trained word embeddings. Aiming for a balance between customisability and ease of use, the natural language understanding module comprises loosely coupled modules combining a number of NLP and ML libraries in a consistent API [8]. To train the natural language processing model, two thing are crucial, a pipeline and training data. Often, more possible input examples are associated with better performance, however, that is not always the case. Next the essential components of this tool will be presented along with the best practices selected.

3.3.1.1 Pipeline

The first step of natural language understanding is choosing the sequence of components which will do text parsing of incomming messages, by detecting the entities in the input sentence, as well as the intention behind it. These components are executed one after another in a so-called processing pipeline. Each element processes the input and creates an output that can be used by any succeeding component in the pipeline. Some components only produce information to be used by other elements in the pipeline and other that produce output attributes which will be returned after the processing has finished [10].
RASA offers pre-defined pipelines, templates, with sensible defaults, or the option to resort to several libraries and assemble a personalized pipeline which best suits its purpose [8]. At the time of the chatbot's NLU model development there were only two important pipelines, `mitie_sklearn` and `spacy_sklearn`. Since they reportedly work very well, one was chosen for this project. The main distinction between the two is that MITIE only returns the confidence level of the chosen intent, while sklearn also provides the classification ranking of the rest of the existing intents, which were not preferred. For this reason, the `spacy_sklearn` template was selected, which uses pre-trained word vectors. The advantage of this pipeline is the fact that it can relate sentences with similar structures, and classify the same intent even with less training examples, or even understand that a new word is also an entity of the same type because of its placing in the sentence.
To define the pipeline, a YAML file must be created. YAML is a human-readable data serialization language commonly used for configuration files, which in this case configure the text parsing parameters and the initial settings of the Machine Learning Model. English is the predefined language used for the initial settings of Rasa NLU, and since Karel will be an English speaking assistant, it is not necessary to specify the language in the configuration file. For the model to be complete and ready for a chatbot, it needs intent and entity extractors as well as pre-processing components, which first "prepare" the text. The components of the adopted pipeline will be explained next to better understand the NLU model's behavior. The configuration file used for Karel can be seen in section A.

SpaCy and Scikit-learn template

The spaCy library is used to prepare text for deep learning, first it tokenizes the input message, then does Part of Speech tagging, and finally Named Entity Recognition, this way establishing the syntax tree of the input sentence. A pipeline based on this library's elements is showcased in Figure 3.6. Its models are statistical and based in prediction, derived from the examples used during training. First SpaCy will analyze unlabelled text and make a prediction. Since the training data has the correct answer associated, the parses receives feedback on its prediction in the form of an error gradient of the loss function that calculates the difference between the training example and the expected output. The greater the difference, the more significant the gradient and the updates to the model [9]. SpaCy is said to excel at large-scale information extraction tasks, which is ideal for the prototype's case, given the amount of data and data analysis existing for mission operations.

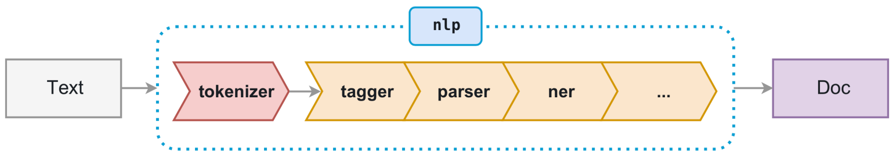


Figure 3.6: Generic pipeline organization using SpaCy from [9].

SpaCy interoperates seamlessly with Scikit-learn, a simple and efficient tools for data mining and data analysis, that does the intent classification in the chosen pipeline. The components that constitute the `spacy_sklearn` template are:

- `nlp_spacy`: SpaCy language initializer, it has no output, essentially initializes spacy structures. Every spacy component relies on it, hence this should be put at the beginning of every pipeline that uses any spacy components.

- `intent_featurizer_spacy`: SpaCy intent featurizer, creates feature for intent classification. It also has no output, being used as an input to intent classifiers that need intent features, as the case of sklearn intent classifier.

- `intent_classifier_sklearn`: Scikit-learn intent classifier, has intent and intent ranking for outputs as formerly explained. The spacy classifier must to be preceded by a featurizer, to use its created features for the classification.

- `intent_entity_featurizer_regex`: Regex feature creation, to support intent and entity classification, both text features and tokens pattern are its outputs. During training, the regex intent featurizer creates a list of regular expressions defined in the training data format. For each, a feature will be set marking whether this expression was found in the input, which will later be fed into intent classifier and entity extractor to simplify classification. There needs to be a tokenizer before in the pipeline.

- `tokenizer_spacy`: Creates tokens using the spaCy tokenizer, having no output.

- `ner_crf`: Conditional random field entity extraction, it appends entities. Is essentially in charge of assigning the correct entities. Since it is integrated in the chosen template and considered good with training custom entities, which is what is need, as all the satellite related words are not pre-trained in any model. It requires sklearn-crfsuite and uses a conditional random field model to work. This entity extractor determines the confidence with which it classified a word or expression, returning this value in the end of all the text parsing.

- `ner_synonyms`: Maps synonymous entity values to the same value. The outputs are the modified entities that already existed, previously classified by entity extraction components. Makes sure that if the training data contains defined synonyms, detected entity values will be mapped to the same value.

Duckling

Temporal expression recognition and classification is not easy, due to its natural ambiguity. As in human language there are many ways to express the same moment or interval of time, one user input can produce many potential results. Duckling is a Clojure library that parses text into structured data, which can be considered a hybrid based on both rules and examples [60]. Thus the Duckling library was added to the default pipeline, with the objective of turning human expressions that mean some time period into actual datetime objects usable by the system [60]. The library is run on a HTTP server making a simple request. To be integrated with Rasa, it is necessary to add it to the configuration file and specify the url for the duckling server, to train the model, and then run the duckling server in a docker container with the same url, to use it. As duckling tries to extract as many entity types as possible without providing a ranking, it is important to you specify the dimensions for the duckling component, in this case Time. Dimensions, locale and timezone can be defined as well on the pipeline. With the final object returned after text parsing, Duckling will always return an entity extraction confidence of 1, by defect for being a rule based system. The main features of this library are:

- Agnostic: it makes no assumption on the kind of data pretended to be extracted, or the language. It is trainable with a combination of examples and rules for any task that takes a string as input and produces a map as output.
- Probabilistic: it assigns a probability on each potential results a given string my produce. Then it decides which results are more probable using the corpus of examples given in the configuration. This way, rules can be more robust to user input [60].

At the time of Karel's NLU model develpment, for example, "first Thursday of last month at 8pm" would be correctly extracted as "value":"2018-09-06T20:00:00.000+01:00". However, unless the Input message specified, a beginning and end date, Duckling always returned temporal expressions as a single datetime object. For example "last month" was parsed as "value":"2018-09-01T00:00:00.000+01:00", when in reality what is meant by the user is a time period between 2018-09-01 and 2018-10-01. To deal with this issue some codding was necessary on the bot side, as most questions related to mission operations will have to define a time period. For this adaptation, some additional information provided by duckling was used. The object returned by Duckling after parsing a time expression is a python dictionary similar to:

```
{
  "extractor": "ner_duckling_http",
  "confidence": 1.0,
  "end": 65,
  "text": "last month",
  "value": "2018-09-01T00:00:00.000+02:00",
  "entity": "time",
  "start": 55,
  "additional_info": {
    "grain": "month",
    "values": [
      {
        "type": "value",
        "grain": "month",
        "value": "2018-09-01T00:00:00.000+02:00"
      }
    ],
    "value": "2018-09-01T00:00:00.000+02:00",
    "type": "value"
  }
```

Here `["additional_info"]["values"]["grain"]` is the smallest time unit found on the input message. Manipulating the datetime entity value according to its "grain", an interval can be created. The basis for the created script is to add a delta time to the value parsed by Duckling, according to the grain. Returning the initial datetime as the start time for the time interval of the request and the time after the sum as the end time. An example is shown bellow:

```
def time_period(self, tracker):
    time_slot = tracker.get_slot('time')

    if time_slot['additional_info']['grain'] is 'month':
        # number_days = (month, days in month)
        number_days = calendar.monthrange(time_slot.year, time_slot.month)
        delta = timedelta(days=number_days[1])

    time0 = time_slot
    time1 = time_slot + delta

    return [time0, time1]
```

The time interval returned using the example above would be: [2018-09-01, 2018-10-01]. Using Duckling, no input examples are required for correct time parsing, nonetheless, without some temporal expressions in the example sentences, some confusion is expected from the bot, when these words appear for the first time. Having this in mind, temporal expressions were added to the training data.

3.3.1.2 Training Data

The training data contains a number of examples of possible user inputs along with their mapping to a suitable intent and the entities present, it is used, as the name implies, to train the NLU model. With Rasa, more examples do not necessarily mean a better model. However, intent classification is independent of entity extraction. So NLU can get the intent right but entities wrong, or the other way around. Which suggests providing enough data for both intents and entities is essential. If intents are easily confusable, more training data is needed. Implying that as much intents are added, more training examples should be given for each. Identically, the number of training examples for entities depends on how closely related different entity types are and how clearly entities are distinguishable from non-entities [61].
The Data Format can be markdown or json, a single file or a directory containing multiple files. Although markdown is the easiest format for humans to read and write, considering the amount of different parameters provided by the knowledge base of the satellite mission, it is advantageous to generate plenty of entities examples. To achieve this, there is a tool for generating training data sets in Rasa's json format using a simple Domain-Specific Language (DSL), Chatito. The json file consists of an object called `rasa_nlu_data`, with the keys `common_examples`, `entity_synonyms`

and `regex_features`. Next parts of the Chatito's file format used to generate the training data for the NLU model is shown:

```
%[ask_query]('training': '35000')
    ~[ask?] @[query_type] @[reporting_data] of the @[system_element] @[time] ~[please?]

~[ask]
    Show me
    Can you tell me the
    What was the

@[query_type]
    ~[average]
    ~[maximum]
    ~[minimum]

~[average]
    average value for
    avg
    mean
    average

@[reporting_data]
    temperature
    current
    voltage
```

Intents are defined with %, entities with @ and "alias" with ~, furthermore a ? in front of any of these identifiers means the sentence can also be formed without it. After defining all the intents, entities, alias and ways to position each in sentences, Chatito then randomly creates, for each intent, sentences with the possible combinations until it reaches the maximum number establish next to the intent identifier. All these examples will generate the training data file. For the Rasa adaptor of Chatito, alias are not only used to extend the possible ways of defining a sentence, but are mapped as synonyms to entities or entity values [62]. Which are stored in the `entity_synonyms` array and used to improve entity classification, as well as replacing the "value" of it. This way entities may have several types or values each with more then one possible expression. Whenever the same text is found, the value will use the synonym instead of the actual text in the message. In the next subsection, the benefits for the bot of using this feature will be better explained.
The training examples are placed in the `common_examples` array, that is the most important part of the file, used to train the model. Common examples have three components:

- Text: an example of what would be submitted for parsing, it is required for training and must be a string.
- Intent: is an optional string identifying the intent that should be associated with the text, as it can be used only to train entity classification, it may be omitted.
- Entities: an optional array mapping the entities present in the text which need to be identified. Seeing that entities can span multiple words and the value field does not have to

correspond exactly to the substring in the text. The array should specify the entity start and end value, which make a python style range to apply to the string, in addiction to its value, allowing to map synonyms or misspellings, to the same value [61].

`regular_expressions` can be provided to the training data to help the CRF model learn to recognize some entities. Each regex provides the NER with an extra binary feature, 1 if it was found or 0 otherwise. By adding regular expressions, the model will look for words matching it, to quickly associate it with a given entity. For the initial version of Karel, this feature will not be used. However, when expanded to receive parameter names directly in the input it may by beneficial to create a regex for it. This array is not filled by the DSL tool, so it has to be filled by the developer. The training data is randomly generated by Chatito, and after the NLU model can be trained.

3.3.1.3 Training the NLU model

To train a NLU model, the `rasa_nlu.train` command is utilized, and it is important to specify where to find the configuration and training data files. Project and model names can also be defined, the first allows to work simultaneous on more than one project, the second helps keep track of different versions of the project. The files regarding the NLU model will be generated and all saved on a specified directory. This code can be seen in section A.
Figure 3.7 shows how a pipeline with ”Component A”, ”Component B”, ”Last Component” calls each component by order during training. Before text parsing begins, a python dictionary, ”context”, is created and used to then pass information between components. Initially the context is filled with all configuration values. In the picture the arrows reveal the call order of the pipeline elements, establishing the path of the passed context. After all components are trained and persisted, the final context dictionary is used to persist the model’s metadata [10]. However, this model cannot retrieve any answer to the user, to create a complete Chatbot, the conversational flow must be addressed. The next subsection explains how Rasa achieves this, creating a Dialogue model.

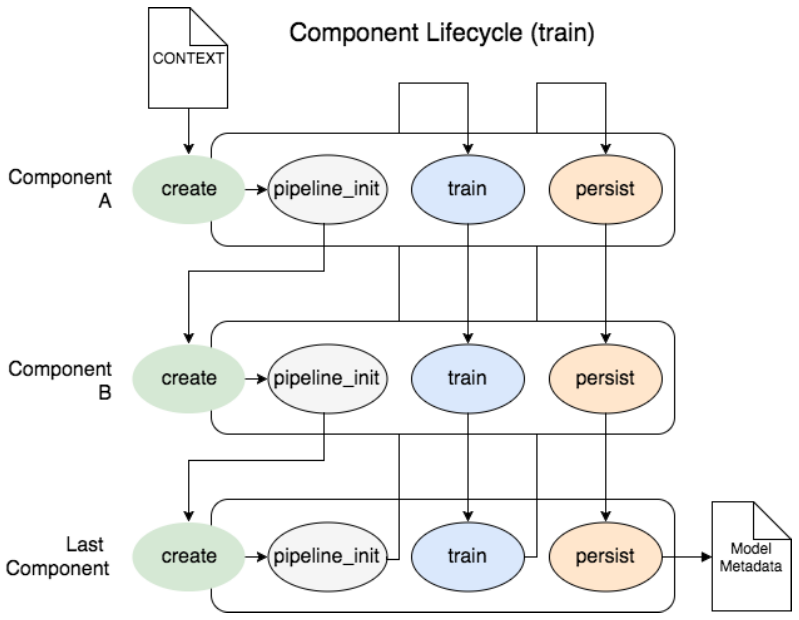


Figure 3.7: Call order of pipeline components during NLU model training from [10].

3.3.2 Rasa Core

Rasa Core is the open source framework's dialogue manager, that keeps track of the state of the conversation and chooses the best next action accordingly. Analogously to Rasa NLU, it learns from example conversations. Before explaining the necessary components for the creation of the dialogue model, some arguments used by Rasa need to be introduced:

- Agent: allows to train, load, and use a model. It is a simple API accesses most of Rasa Core's functionality.
- Events: allow the bot to modify the internal state of the dialogue. This information is be used to predict the next action. Events can for instance set slots, to store additional information, or even restart the conversation [63].
- Tracker: stores the dialogue state, Keeping slots, as all the events leading to the present state. The state of a conversation can be reconstructed by replaying all of the events that have occurred within a conversation. There is one tracker object per conversation session. [8].

3.3.2.1 Policy

The policy selects the action to be execute given the tracker object. The featurizer, instantiated along with it, creates a vector representation of the current dialogue state, describing the last action, the intent and entities in the most recent message and the urrently defined slots [8]. There are different policies to choose from in Rasa, and multiple can include in a single Agent. The policy which predicts the next action with the highest confidence will be used. For Karel, two policies were selected:

- MemoizationPolicy: just memorizes the conversations in the training data. It predicts the next action with confidence 1.0 if that exact conversation exists in the training data, otherwise it predicts None with confidence 0.0.
- KerasPolicy: uses a neural network implemented in Keras to select the next action. The default architecture is based on an Long Short-Term Memory (LSTM) [64].

When the bot cannot parse the input correctly, is important to have a fallback action that answers the user, stating the bot did not understand the request. For this, Rasa provides the FallbackPolicy, which can be added to the policy ensemble. This feature was implemented in Karel as well. It will be executed if the intent recognition has a confidence below a specified value or if none of the dialogue policies predict an action with confidence higher than stated. The `rasa_core.train` scripts provides parameters to adjust these thresholds:

- `nlu_threshold`: is the minimum confidence value needed to accept an NLU prediction.
- `core_threshold`: is the min confidence value requested to accept an action prediction from the dialogue model.
- `fallback_action`: is the action to be called if the confidence of intent or action prediction is below the threshold. This will send the `utter_default` template to the user, which must be specified in the domain file. It will also revert back to the state of the conversation before the input that caused the fallback, so that it will not influence the prediction of future actions. If a specific intent triggers this, like `out_of_scope`, the path should be added as a story [63].

3.3.2.2 Training Files

Before the Rasa Core is ready to be trained, some files must be created, of different formats and with different information, containing all the necessary specifications which allow the bot to know what to answer. First a domain must be created, and then each component of it must be further defined, on the same file or in additional ones if necessary. Then the training file has to be developed to train the actual decision making process of the bot. These files and its components will be exposed next.

Domain

The domain file defines the universe of the bot, all the intents, entities, actions and additional similar information. It must be written in a yaml file and have a list of actions that the chatbot could execute. It is also were simple answers from the chatbot are defined, called templates. These strings are used as actions for straightforward interactions, and can be assigned more than one possibility, which will be randomly sent to the user. The slots, information to keep track of during conversation, are also determined in this file.
Slots are the bot's memory, acting as a key-value store which can be used to store information the user provided as well as information gathered by the bot, for example the result of a database query. Usually, slots are meant to influence how the dialogue progresses. Rasa offers different slot types for different behaviors:

- Text: only only stores whether the slot has a value. The specific value does not influence the dialogue.
- Categorical: used if the value of the slot is important, possible values should be stated in the domain.
- List: for slots that can have more than one value per input. The length of the list stored does not influence the dialogue.
- Unfeaturized: used merely to store some data and not to affect the flow of the conversation.

Karen's conversational pattern consists in collecting pieces of information provided by the user in order to used it to search the knowledge graph, which requires slot filling. For this slots are set either by returning events in actions, which needs to be included in stories, or automatically if the NLU model picks up an entity, and the domain contains a slot with the same name.

Actions

Dialogue management is faced as a classification problem. In this sense, Rasa Core predicts which action should take after each interaction. Essencially, actions are the things the bot runs in response to user input, it can be:

- Default actions,
- Utter actions,
- Custom actions.

A default action is predefined by Rasa and can be used without being specified in the domain. There are three default actions:

- `action_listen`: which makes the bot stop taking further actions until next user input.

- `action_restart`: that resets the whole conversation, by clearing the tracker.
- `action_default_fallback`: ignores the user input and utters designated message. It is used when the bot cannot understand the intent or entity of the message with a high confidence level, designated by the developer [63].

Utter actions state a simple message which should be sent to the user. As declared when explaining the domain, an utterance template must be added to the file and start with ”utter”. Any other action programmed by the developer is a custom action, that can run arbitrary code. A new action is created in a python file much like a Class, and added to the domain file by its name. The run method returns a list of Event instances, receiving three arguments:

- Dispatcher: parameter used to send messages back to the user.
- Dialogue State Tracker: tracker the state of the current conversation with the user. It can access substantial slot values and even the most recent user message.
- Domain: the bot's domain [63].

When an action is executed, it is passed a tracker instance to be able to use any relevant information collected by it. After creating all the possible actions for the bot to perform, Karel needs to know when to use them, so the stories must be described.

Stories

Stories are the equivalent to the NLU training data. Instead of showing how the text should be classified, it demonstrates what the bot should do after, according to what was already understood from the input. From each stories, Rasa Core creates a probability model of interactions.
Stories should be written in a Markdown file, easy to build and understand. A story starts with ## followed by an optional name. Lines starting with * are the intent of the messages sent by the user, and the entities, if they make a difference on the choice process of next action. Lines that start with - are actions taken by the chatbot. It can be simple messages sent back to the user, or something more complex, including calling an API and interacting with other systems. Next an example story from Karel is shown:

```
## story_query_check
* ask_query
  - action_query_check
  - slot{"query_check": "not_valid"}
  - utter_query_invalid
* ask_query
  - action_query_check
  - slot{"query_check": "valid"}
  - action_answer_num
```

In this story, the slot `query_check` is set by the action `action_query_check`, and its assigned value determines the next taken action. After, the domain must be written, connecting all the components in one file. With all the required training files defined, the bot can be trained.

3.3.2.3 Training the Dialogue model

Rasa Core works by creating training data from your stories and training a model on that data, so the next step is to train a neural network on our example stories. Rasa's dialogue training method will, by default, create longer stories by randomly glueing together stories from the file, this is called data augmentation. Karel, was developed for simple Query & Answer stories, so extra arguments were not needed.

For the project, an agent was created and the train method used as shown in section A. The "debug" flag used when training the dialogue, prints information that helps understand what is happening during training, useful specially when errors arise. The `domain_file`, where the bot's domain is specified, must be passed to the Agent, for it to access all the actions it can perform and the slots it must identify, in addiction to the policies chosen, that determine how the bot will consider the stories, which should also be passed the Agent with the `load_data` method. Then the Agent's train method is used with the stories epochs number, that refer to how long the model must be trained for, batch size, which determines how many training examples bot will be trained on at a time, and `validation_split`, the percentage of training samples used for validation. These values depend on how complex the training files are, and were found based on training accuracy. Finally the new dialogue model will be saved in the determined `model_path`. After both models are trained, the basis of the conversational bot is ready, however, without the KG it is not capable of finding the requested parameters.

Chapter 4

Implementation and Evaluation

The next step to complete the scope of this master dissertation is to apply Karel to a chosen mission in order to assess its behavior and performance. In this Chapter a mission which adapts the conditions and requirements is selected. Section 4.1 presents the chosen mission and its specifications to frame the case study. From this mission a particular Knowledge Graph needs to the generated, process which is explained in Section 4.2. Then Section 4.3 details the creation, training and evaluation of the NLU and Dialogue models matching the mission's KG. After, the case studies must be determined, so the bot can be fully tested and its efficiency proved, which is shown in Section 4.4. Finally, Section 4.5 discusses the achieved results and their value. Thus the purpose of this Chapter is to implement all the modules discussed in Chapter 3 in order to create a bot suitable to a mission, that demonstrates its contributions to satellite operations.

4.1 The Gaia Mission

GAIA was the mission chosen as scenario to test Karel, operating successfully since 2013 and expected to last until 2020. It is an example of a stable satellite, whose flight is routinely monitored, while still being a relative new mission, with a very active assignment. Gaia's primary science purpose is devoted to the understanding of the Milky Way's composition, structure and evolution. The objective is to provide a very accurate dynamical 3-D map of our Galaxy by using global astrometry from space, complemented with multi-color multi-epoch photometric measurements. The aim is to produce a catalog complete for star magnitudes up to 20, which corresponds to more than one billion stars. Furthermore, the measurements will not be limited to the Milky Way stars, providing unique contributions to extended extragalactic astronomy. These include the structure, dynamics and stellar population of the Magellanic Clouds, the space motions of Local Group Galaxies and studies of supernovae, galactic nuclei and quasars. The latter will be used for materializing the inertial frame for Gaia measurements. The catalog unprecedented astrometric and photometric measurement accuracies will enable a breakthrough in numerous areas of Astrophysics, including our Galaxy formation and dynamics, stellar classification and astrophysics, planet and asteroid detection, reference frame and general relativity tests. To achieve its purpose, Gaia's payload includes, Astrometric instrument, Photometric instrument and Radial Velocity Spectrometer. By April 2018, the satellite had already plotted the positions of more than 1.7 billion stars, the Star Map created from those 22 months of charting the sky is presented in Figure 4.1 [11].
The mission ground system is made of operational entities (Mission Operation and Science operation Centre) and ground stations for command control and payload data downlink from Gaia. Additional institutes complement the data processing system which is in charge of storing, archiving and processing large scientific data everyday. Mission operations are carried out by ESOC, which is one of the conditions for testing Karel, as it was intended for the prototype to be build for this operation center. These began following separation of the satellite from the launch vehicle and are to be maintained until disposal at the end of the mission. ESOC provides a ground segment that comprises all facilities, hardware, software, documentation and staff, required to conduct the

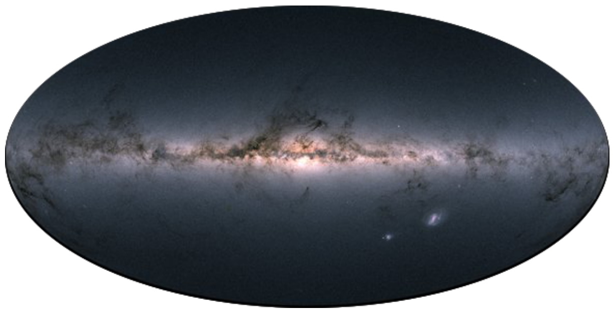


Figure 4.1: Star Map created by Gaia, adapted from [11].

mission operations. All mission and flight control facilities, except the ground stations, are located at ESOC, including the interfaces for the provision of science telemetry to the scientific data reduction teams. Activities undertaken by ESOC in support of Gaia, similarly to any satellite operation, include:

- Overall mission planning and co-ordination.
- Satellite status monitoring utilising subsystem housekeeping telemetry and control via telecommand.
- Orbit determination using ground tracking data and additional optical observations.
- Scheduling of ground segment resources for the payload data downlink.
- Manoeuvre planning for orbit maintenance primarily to counteract the effects of solar pressure and micro propulsion disturbances.
- Maintenance of on-board software.

The spacecraft is composed of several main assemblies:

- The core Service Module, composed of Gaia'a structure, harness and thermal control on which are accommodated the Electrical Service Module equipment, the Chemical and Micro Propulsion and the Payload module electronics.
- The Payload Module (PM) instrument, made of the optical bench and the Focal Plane Assembly (FPA) mounted onto the Service Module (SVM) through two parallel sets of 3 bipods. The launch lock set withstands the launch mechanical environment and is released at the beginning of the cruise phase ,during the execution of the autonomous separation sequence. The in-orbit set allows conductive decoupling of the payload module instrument from the SVM.
- The Service Module appendages, composed of the Deployable Sunshield Assembly (DSA) stabilising the thermal environment of both payload module and service module, the Deployable Solar array Panels (DSP) mounted on the DSA, the thermal tent attached to the service module and the antennas, three Low Gain Antennas (LGA) and one Phased Array Antenna (PAA).

The Service Module is essential for Gaia'a operation. The complete SVM consists of the Control and Data Management System (CDMS) units, Attitude and Orbit Control System(AOCS) units, Electrical power units, Telemetry, Telecommand & Command (TTC) units, Chemical and Micro Propulsion units, Solar Arrays and DSA thermal items. For Karel's testing, the Attitude and

Orbit Control System was selected, being a vital part of any satellite, as its orientation in space is crucial for its performance, it is a good example to show the performance and effectiveness of the bot behavior. Figure 4.2 displays the constituents of the satellite's AOCS used to build the Knowledge Graph.

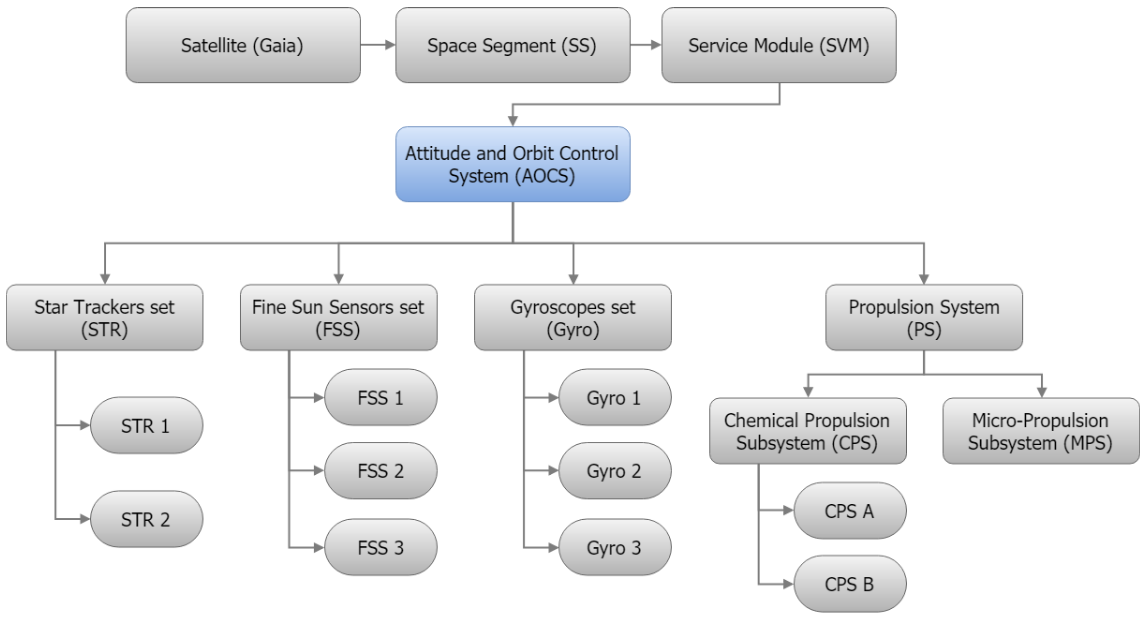


Figure 4.2: AOCS systems organization.

At this point of the mission, the spacecraft is in the Extended Operations Phase which takes place after completion of the Routine Operations phase, and is similar to it. Consisting in scientific data acquisition from the different payloads, storage and transmission of these data to the Earth during the Cebreros and New Norcia visibility periods. Which means during this time the AOCS is stable and mostly needs to be updated and monitored, unless some unexpected event takes place. This should translate in stable parameter values, so anomalies are easy to spot. After choosing the mission, the knowledge graph and the NLU model can be created.

4.2 Creating the Knowledge Graph

Generating the complete Knowledge Graph is a semi-automated process that maps the mission database to the pre-created ontology. Which implies its extension of the base model to fit each mission, using regular expressions to do the mapping, as explained in Section 3.2. Considering the S2K MIB database, the one used for this mission, as it is operated by ESA, the tables PCF for Reporting Data, PID for Events and CCF and CSF for Activities were only mapped for the system elements present in Figure 4.2. The Knowledge Graph is encoded in RDF N-Triples and queried using SPARQL. An example of this queries can be:

```
# SVM -> AOCS -> FSS -> FSS-1
fss:fss1 rel:instance_of ssm:system_element .
fss:fss1 rel:part_of aocs:fss .
fss:fss1 rdfs:label "Fine Sun Sensors 1 (FSS-1)" .
fss:reporting_data rel:has_dbdef
    """result = get_pcf(pcf_file,'.*','.*FSS1.*')""" .
fss:event rel:has_dbdef
```

```
    """result = get_pid('pid_file','^5$','.*','.*','.*FSS1.*')""" .
fss:activity rel:has_dbdef
    """result = get_ccf(ccf_file,'.*','.*FSS1 .*','.*','.*','.*')""" .
```

This case refers to the mapping of the Fine Sun Sensor 1 (FSS1) in the MIB, which is part of the Fine Sun Sensors set (FSS), that is integrated in the Attitude and Orbit Control System (AOCS). From the code all the relations are clear, FSS1 is a system element part of the FSS and AOCS. Its label clearly identifies its name along with its initials, which are how the set is recognized by most SOE. After the clear description of the element and the relations with the other classes, each type of parameter related to it must be mapped through SPARQL from the MIB. Reporting Data is searched for in the pcf, as previously explained, using the function `get_pcf` to define the patterns that must exist on the necessary columns. The first argument of the function determines the path to the file, the next indicates that the first column of the table can have any number or letter in it, and the last states that in this example the second column must contain the string ”FSS1” in it. The mapping of Events and Activities is done in a similar way. However, it is noticeable that for Events, the first column must have the number 5, because this is the way designated by ESA to define this class of data in the pid table. The subgraph of the FSS1 done through the described process is visually represented in Figure 4.3

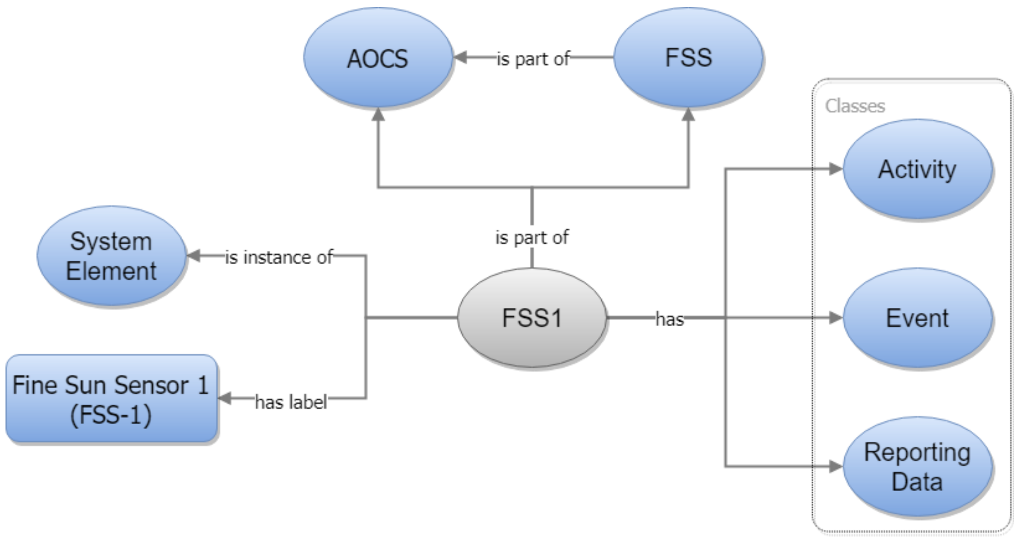


Figure 4.3: Knowledge Graph mapping of FSS1.

At run time, many parameters would get connected to Reporting Data, Event and Activity, depending on the user input and dependent on the regex used. Figure 4.4 adds an example of a reporting data parameter mapped by the function `get_pcf`. Which could be added to the above subgraph. This parameter might answer a user query about maximum, minimum, average or time interval of the Fine Sun Sensor 1 temperature. For the three first analysis, a value should be the response, associated with a graph of the parameter's behavior. After defining the Knowledge Graph or part from the mission database, it should be used to define the specifications of the Conversational side of the bot, building the NLU and core models based on it, as explained hereafter.

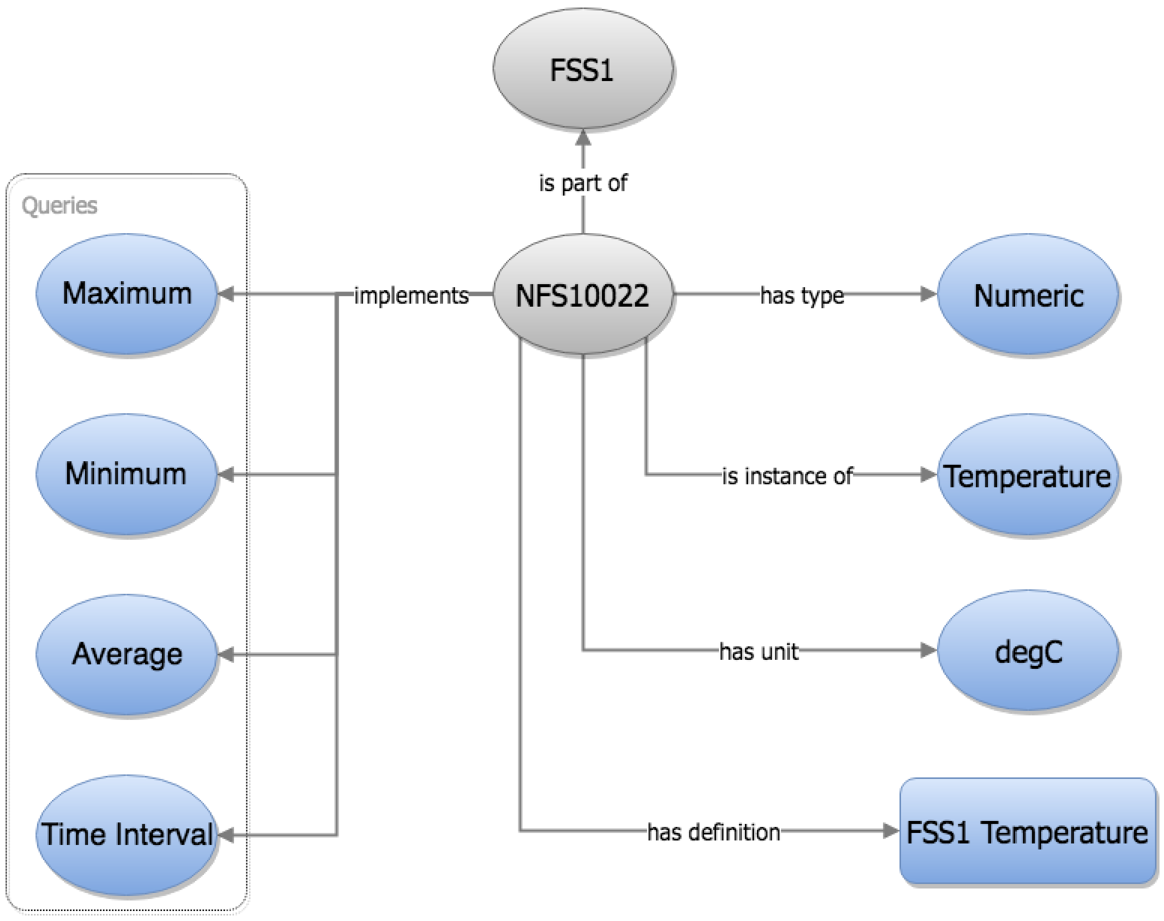


Figure 4.4: Mapping of a FSS1 parameter.

4.3 The Gaia Chatbot

Generated the KG, all the particular aspects of Gaia can be adapted to create Karel's final prototype. How the NLU and Dialogue Rasa models are build and trained was exposed in Section 3.3, now all the specification must be defined to adapt to the mission.

4.3.1 NLU model

The NLU model can be defined based on the mission's KG, its creation is an iterative process, starts by generating the training data, then training the model and finally evaluating it. Based on its classification effectiveness it may be required to repeat the steps, until a satisfactory model is found. Foremost, the important aspects present in a user message must be defined. To achieve this, first it is important to remember the meaning of intent and entity from Chapter 2. The intent is essentially what the user aims to obtain with the message. For the implementation of the chatbot developed in this dissertation, the intents are:

- `Greet`: for any greeting message.
- `Bye`: for goodbye messages.
- `Out_of_scope`: for messages which the bot does not know how to answer, out of its domain.
- `Ask_query`: for questions related with the domain, which is the main purpose of the bot.

It was decided at the beginning of the project development to have only one intent to handle all the questions. This was thought as a way to simplify the first prototype and still allow for some flexibility to add new questions later on. However, this required more training data and specially more processing on the Dialogue model actions side to handle all the currently possible queries. This meant the entities had to be divided in:

- `Query_type_num`: used in question regarding numeric reporting data to state the type of analyses required, its values current supported by the Analytics module are ”maximum”, ”minimum” and ”average”, which are self explanatory.
- `Query_type_enum`: defines the type of analysis requested for an enumerated reporting data parameter. The Analytics model is currently able to perform ”transition”, returning the number of times the parameter value changed, and ”value”, returning the value of the parameter, all during the time period specified in the input.
- `Query_type_cond`: refers to the ”condition” type of query on the KG and time interval on the Analytics module. It is necessary to separate it from the rest of the numeric and enumerated types for two reasons, first because it is common to both and that could lead to wrong parsing by the NLU model, but specially because its value should not only be ”condition”, it is important to classify the type of condition, which can be ”above”, ”below” or ”between”. When this entity is received by the first action, the query type is assigned as ”condition” and the value of the entity is used to make different API calls to the Analytics accordingly.
- `Reporting_data_num`: establishes the type of numeric reporting data asked, it can be a variety of values, among temperature, voltage, power, current, velocity, angle, and many more. It can be related with a numeric or conditional query type.
- `Reporting_data_value`: used to store the values above, below or between associated with the condition query type. This entity is a list which defines the interval that must be respected by the reporting data, being the answer the timestamp, time interval or list of both when this condition is met. Its values can be any number with or without any physic unit.
- `Reporting_data_state`: its a list entity as well, similarly used to define the state values of enumerated reporting data, to use for the enumerated query type transition. Its values can be any string indicating states of a parameter, for instance on and off, enabled and disabled. If its value is simply defined as ”status”, or any of the trained synonyms, all the transitions of the queried system element's parameters, implementing any state change, will be counted.
- `System_element`: determines the system element on the user input, for this prototype version it can be any of the identified systems and elements present in Figure 4.2. This entity must be present in any message with the `ask_query` intent, for the request to be successful.
- `Time`: as detailed in subsubsection 3.3.1.1, this entity collects the temporal expressions from the user message, using the Duckling library, the value is then transformed into a time interval, which will be used to send the API call to the Analytics module as a Unix timestamp.
- `Query_check`: as explained in **??**trainfiles, its an entity assigned by an action, so it is not present in the training data. Its purpose is to be set to `valid` if parameters correspondent to the entities assigned by the user exist in the Knowledge Graph, which triggers the next action or otherwise be set to `not_valid`, ending the conversation.

After establishing the intents and entities, the training data can be generated, Chatito will be used, as revealed in subsubsection 3.3.1.2. The above presented possible combinations of entities are used, with three major types of questions formulated:

```
~[ask?] @[query_type_num] @[reporting_data_num] of the @[system_element]
    @[time] ~[please?]

~[ask?] @[reporting_data_num] @[query_type_cond] @[reporting_data_value]
    ~[unit?] of the @[system_element] @[time] ~[please?]

~[ask?] @[query_type_enum] of the @[system_element] @[reporting_data_state]
    @[time] ~[please?]
```

More training examples are generated by varying the entities placement within the same type of query and adding synonyms to each entity in the Chatito file. This is very important for two reasons, ESA's FCTs are groups of people from diverse backgrounds, engineers, scientists, managers, technicians, quality control inspectors, designers and more, usually multicultural, all working together for the same goal, which suggests questions can be made in many different ways. In addiction, there are a lot of ways to refer to the same entity, so the bot should be trained to understand several of these possibilities, acronyms are specially significant as that is the way system elements are usually named by the SOEs. An example from the training data file generated by Chatito for Karel is:

```
{"rasa_nlu_data": {
  "regex_features": [],
  "entity_synonyms": [
    {
      "synonyms": [
        "mean",
        "average value for",
        "avg",
        "average"
      ],
      "value": "average"}],
  "common_examples": [{
      "text": "Can you tell me the avg temperature of the gyro 3 yesterday",
      "intent": "ask_query",
      "entities": [
        {
          "end": 23,
          "entity": "query_type",
          "start": 20,
          "value": "average"
        }, {
          "end": 35,
          "entity": "reporting_data",
          "start": 24,
          "value": "temperature"
        }, {
          "end": 49,
          "entity": "system_element",
          "start": 43,
          "value": "gyro3"}]
}]}}
```

The text field was randomly generated with the given alias, and the intent of it as well as the entities present are identified. Synonyms are determined based on the alias used for entities. Except for temporal expressions as they are parsed by Duckling. With the training data defined, the NLU model can be trained. How this training is done and the pipeline used, was already detailed in subsubsection 3.3.1.3. An example of an input parsing created after training as a combination of the different components in the chosen pipeline can be seen in Appendix B. The object returned after parsing has two important fields that impacted the entities returned. The ”extractor” of an entity, that shows which entity extractor was responsible of its identification. The ”processors” field, that contains the name of components that altered this specific entity, when applied [10]. The use of synonyms cause the ”value” of the entity to not necessarily match the text, instead returning the trained synonym. Despite not improving the model's classification of the entities, this attribute is very beneficial when necessary to pass the entity to the KG or the Analytics, this way not depending on the text expression used by the user, always utilizing the same value. Next subsection explains the Dialogue model created for Gaia.

4.3.2 Dialogue model

The intents and entities found for the NLU model can the used as the starting point to design the Dialogue model. Planning the possible answers or actions the bot can deliver for each intent, as well as how each entity is classified as a slot or not, defines Karel's domain file. As informed in subsubsection 3.3.2.2, the domain should be written in a YAML file and define all the universe of the bot, this way:

```
slots:
  system_element:
    type: text
  time:
    type: unfeaturized
  query_check:
    type: categorical
    values:
      - valid
      - not_valid

intents:
  - greet
  - ask_query

entities:
  - system_element
  - time
```

```
templates:
  utter_greet:
    - "Hello there! What can I do for you?"

actions:
  - actions_file.Answer
  - utter_greet
```

In this excerpt of Karel's domain file, three types of slots presented in the previous Chapter are defined. `System_element` is a text slot set by having the same name as an entity classified by the NLU model. The slot time is set by duckling as already determined in subsubsection 3.3.1.2, for this reason is of the type unfeaturized. As seen in the stories, the slot `query_check` will determine the next actions, for this reason it must be categorical. An utter action and a custom one are identified. `Utter_greet` has an example template assigned, the action Answer is coded in a python file named `actions_file`. How actions are created has been explained in Subsection 3.3.2.2 as well. Actions cannot directly mutate the tracker, however when executed may return a list of events, which are utilized to update the tracker's state [8]. Two Events returned by Karel Actions are:

- AllSlotsReset(): erases all the slots previously set, so they do not influence the next action.
- SlotSet(`"query_check"`, `query_validity`): sets the slot `query_check` to the string stating its validity.

For the prototype, two main actions were developed, the first is immediately triggered by the `ask_query` intent. This function checks if all the required entities are assigned, for instance if the `query_type_num` slot is part of the user input, at least `system_element` and `query_type_num` must be present as well. `Time` has to always be specified by the user, as all the Analytics methods require a time interval on which the analysis should be done. If the necessary slots are set, the action queries the KG for parameters using them. If some required slots are missing or no parameters are found, `query_check` will be set as not valid and the conversation will end, clearing all the slots and sending a message to the user explaining why it could not find the asked values followed by template utter. If on the contrary all the conditions are checked, `query_check` will be set as valid, and the second action will be triggered. This function makes an API request to the Analytics method, choosing it based on the slots set and sending the slots values. Receiving the answer returned by the RESTful API, this action formulates an answer which displays the values mimicking how a human would. In the end, the conversation is ended and all slots reset.
The last training file required must contain the stories. This data was created as explained in Subsection 3.3.2.2 using the intents, entities, actions and utters presented before in this Subsection. Generated using Rasa's `visualize` mode, Figure 4.5 graphically displays the possible story paths, developed for Karel. The domain and the stories files must be passed as arguments to this method. With all the required training files defined, the bot can be trained. Which was performed as described in the previous Chapter, using the training files now specified. The conversational Assistant seems ready, however, to know if the models are a good fit for the chatbot, it is important to evaluate them, Rasa's solution is presented hereafter.

4.3.3 Evaluating and Running the Models

Training the models does not guarantee the correct classification of the user input or the selection of the right action according to it. Thus, it is fundamental to determine if the best models for the

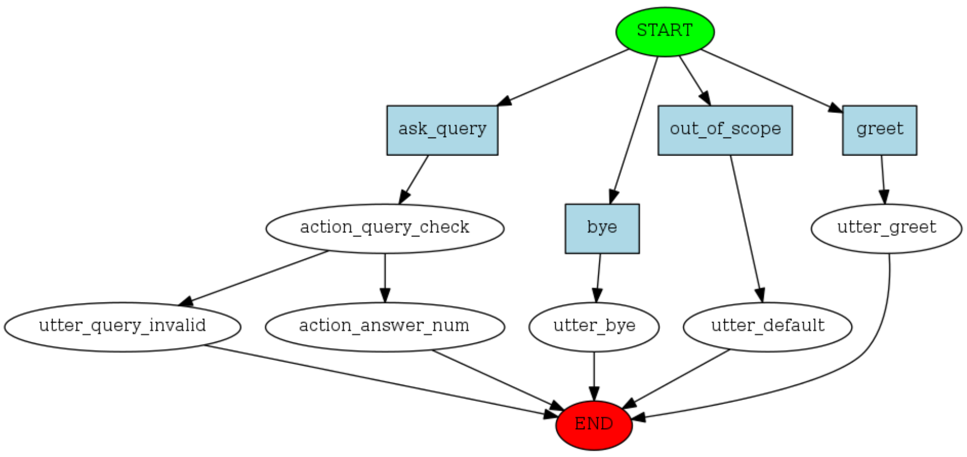


Figure 4.5: Karel's generated story paths.

intended outcome were built, for the bot to efficiently accomplish its goal.
To realize how the NLU Model is performing, if there is enough data, and if the intents and entities are well designed, Rasa as an `evaluate` mode. The testing of the model can be achieved by creating some testing data, similarly to how the training data was generated. Chatito was used again to randomly mix more text examples, with intent and entities classification. Synonyms are not required, as one of the goals of the evaluation is to know if this classification is well done. This test set is then passed as an argument to the `evaluate` method, together with the path for the trained NLU model. The return shows how well the model predicts the test cases. The evaluation script will log precision, recall, and f1 measures for each intent and one summarized for all, as presented in section C. Furthermore, it creates a confusion matrix to visually display which intents are mistaken for which others, which is shown in section C. Finally, the evaluation script creates a histogram of the confidence distribution for all predictions, separating the confidence of wrong and correct predictions in different bars, displayed in section C. The entity extracted by Duckling, `time`, will not be included in Rasa's evaluation. The evaluation reports precision, recall, and f1 scores for each entity type `ner_crf` is trained to recognize [65] followed by the number of supporting examples. Before explaining these performance measurements, some concepts must be introduced:

- True Positives (TP): The correctly predicted positive values, when the true label is positive and the predicted label matches that value.
- True Negatives (TN): The correctly predicted negative values, the cases when a negative true label is matched by the predicted label value.
- False Positives (FP): When the true label is negative and the predicted label is positive.
- False Negatives (FN): When the true class is positive but the predicted label in negative [66].

This means the false positives and false negatives occur in the cases where the predicted label contradicts the truth, and the other two classify the correctly parsed labels. The above mentioned Rasa evaluating measures can now be fully explained:

- Accuracy: the most intuitive performance measure, simply being a ratio of correctly predicted observations to the total observations. Accuracy = (TP+TN)/(TP+FP+FN+TN)
- Precision - the ratio of correctly predicted positive observations to the total predicted positive observations. Precision = TP/(TP+FP)

- Recall: the ratio of correctly predicted positive observations to all the classifications of positive true label values. Recall = TP/(TP+FN)
- F1 score: the weighted average of Precision and Recall. Therefore, takes both false positives and false negatives into account. F1 Score = 2*(Recall * Precision) / (Recall + Precision) [66]

To evaluate the Dialogue model, the same Rasa mode is utilized, passing it the model and the stories file as arguments. For the conversational flow evaluation, the stories are assessed, printing the failed stories, as Appendix D shows, which are the conversations with at least one action being incorrectly predicted. Furthermore, the evaluation generates an action confusion matrix, reveling how often each action present in the domain was correctly and incorrectly predicted, which is presented in Appendix D [67]. The analysis and discussion of these evaluation results will be performed in the last Section. To use the models and have a functional chatbot, Rasa's `run` method is employed. Both models must be specified, agent and interpreter, as in any Rasa system. However Karel final prototype requires additional code to load the knowledge graph, connect to the Duckilng server and be linked to Mattermost. With a finished conversational assistant implemented, some use cases should be tested to assess its functionality. These situations are explored below.

4.4 Test Cases

A test case is a specification of the inputs, execution conditions, testing procedure, and expected results to be executed to achieve a particular software objective, such as to exercise a particular program path or to verify compliance with a specific requirement [68]. For Karel, the test cases evaluate which actions are being triggered based on the user input, and if the right data is being fetched from the API. Additionally, it indirectly assess how the natural language messages are being classified. Formally defined test cases allow the same tests to be run repeatedly against successive models of the system, which also assists with version control. Asking more then one test case as a chain also assess if entities are being cleared after the output and how the bot continues the conversation after an incomplete request, for which it could not deliver an answer. The determined tests, which will be explained in the following Subsections, are:

- Simple Intents,
- Numeric Queries,
- Time Interval Query,
- Transition Queries.

4.4.1 Simple Intents

The Simple Intents test case was designed to evaluate how interactions that only require a utter answer are classified. The output from the bot reveals which intent was extracted as each has a different template associated. The examples chosen to cover all the possibilities were the simplest way were:

- Greet: "Hello karel"
- Bye: "bye bot!"

- Out of Scope: ”How was the weather yesterday?”

Each example sentence is supposed to trigger the intent with its name, for instance the Greet message should be classified as having Greet intent. Being simple interactions, these are answered based on Rasa’s utter template option, consisting of strings created by the developer and defined on the domain, as explained in Subsection 3.3.2.2. Karel’s answers for this cases should be:

- Greet: ”Hello there! What can I do for you?”
- Bye: ”Goodbye!”
- Out of Scope: ”Sorry, I cannot answer your request”

If the output on Mattermost is as defined above, the test can be considered successful and the intents well classified. The next cases are not so simple, as entities should be classified and custom actions must be triggered.

4.4.2 Numeric Queries

The first queries developed for the mission operation chatbot were numerical, being some of the most asked information that enable the control of many parameters. These questions can be maximum, minimum and average, which assist the SOE to check if the element’s values are within boundaries and if the mean is as expected. The Numeric Queries test case was created to assure not only if the correct actions are performed and the input entities are well classified, but also what happens if necessary information is not provided or if no element exists satisfying the request and even if different ways of asking similar questions are still well parsed. The examples used for this test were:

- Maximum: ”What was the maximum cps current in the first week of last month?”
- Minimum: ”fss3 min volt friday”
- Average: ”can you tell me the mean angle rate of the gyroscopes set last week”
- No Parameter: ”min power star trackers this month?”
- Entity Missing: ”Avg volt yesterday”

All these strings’ intents should be classified as `ask_query`, and the bot should find the entities `query_type_num`, `system_element`, `reporting data_num` and `time`. With the Maximum, Minimum ans Average tests, Karel is supposed to behave very similarly. After the classification of the sentences, the Dialogue model must perform the `ActionQueryCheck`, and the `query_check` slot must be set to valid. Which will send a message informing the user that the request is being processed and `ActionAnswer` should start. This is the action responsible for making the Analytics API call and sending the answer to the user in a visually appealing and easy to understand way. Each case connects to the Analytics method with the corresponding name and the Mattermost platform associates a graph to the found value with all the data used to do the analysis, granting context and confirmation.
The No Parameter test is performed to assess Karel’s operation towards a request for which the KG does not have a correspondent parameter. The established sentence does not make sense as power is not a reporting data that can be associated with the Star Tracker. When these situations occur the first action must set the the `query_check` slot to not valid and send a response to the user

specifying the request that was not found on the database, for entity classification confirmation, and clear all the slots. Followed by a template message that ends the conversation. The final string tests if the bot knows that a required entity was not provided by the user and if the output identifies that problem for the user. In the testing string the system element is clearly missing.
The answers from the bot to the Numeric Query case tests must be:

- Maximum:

  Please wait a second, I'm on it.

  Here's what I found for the maximum current of the cps during the first week of last month ['2018-09-03 00:00:00', '2018-09-10 00:00:00']:

  NPC10079 (PCDU RTA CPS PT2 cur): 0.012002 A

  NPC10136 (PCDU RTA CPS PT4 cur): 0.011479 A

  NPC10121 (PCDU RTA EIU CPSB cur): 0.00229299 A

  NPC10098 (PCDU RTA CPS PT1 cur): 0.011384 A

  NPC10102 (PCDU RTA EIU CPSA cur): 0.00305732 A

  NPC10117 (PCDU RTA CPS PT3 cur): 0.010431 A

  Is there anything else you would like to ask?

- Minimum:

  Please wait a second, I'm on it.

  Here's what I found for the minimum voltage of the fss3 during friday ['2018-10-05 00:00:00', '2018-10-06 00:00:00']:

  NFS10020 (FSS3 posVolt): 11.7425 V

  NFS10018 (FSS3 T3): 1.0691 V

  NFS10021 (FSS3 negVolt): -11.7796 V

  NFS10019 (FSS3 T4): 1.0691 V

  NFS10017 (FSS3 T2): 1.0691 V

  NFS10016 (FSS3 T1): 1.07157 V

  Is there anything else you would like to ask?

- Average:

  Please wait a second, I'm on it.

  Here's what I found for the average angle rate of the gyro during last week ['2018-09-24 00:00:00', '2018-10-01 00:00:00']:

  NGY20008 (GYROST rateUnfiltUncompZ): 1.57400584695e-07 rd/s

  NGY20006 (GYROST rateUnfiltUncompX): -0.000290530697302 rd/s

  NGY20007 (GYROST rateUnfiltUncompY): 4.57730145259e-07 rd/s

  Is there anything else you would like to ask?

- No Parameter:

  Sorry, I couldn't find the minimum power of the str

  Would you like to ask something else?

- Entity Missing:

  Sorry, I did't understand the system element requested.

  Would you like to ask something else?

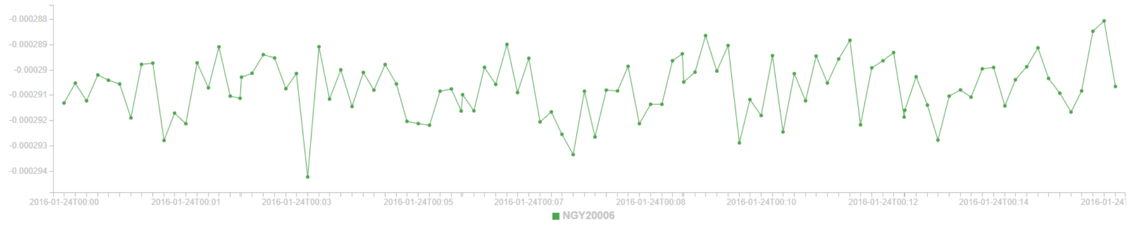


Figure 4.6: Output graph for the average angle rate of the NGY20006 parameter.

As previously mentioned, the answer for the completed requests must contain the entities values from the input to control if the bot understood well the question and the returned value is the asked for, as well as the temporal expression used and the time period it translates to. One of the Mattermost associated graphs is shown in Figure 4.6, and all are displayed in Appendix E.

4.4.3 Time Interval Query

Similarly to the above testing situation, the Time Interval Query case tests the inputs that will connect to the Analytics method with the same name. This type of question is very important when monitoring out of limits values, as it returns the time period when it happened, as well as when analyzing specific value ranges. As the Numeric Query test, during this evaluation the defined cases also test the correct classification of the sentence and the reaction to lack of entities specifications or parameters found. The defined examples were.

- Time Interval: ”when was the temperature of the fss1 between 39 and 41 DegC during the first day of january?”
- No Parameter: ”Vel fss above 3 m/s friday?”
- Missing Entity: ”str1 between 29 and 33 during jan”

Once the message is classified, the first action must be triggered, the same way as for the Numeric Queries, and if parameters are found, the Dialogue model will continue to the `ActionAnswer`. Finding reporting data values in the Time Interval case, the Analytics method used should be the homonym, for which a from and a to value can be determined if an interval is queried about, or solely one if the question regards all the information above or below a value. For the requests to be completed, Karel formulates an answer that makes sense and is easy to visually analyze. The second and third cases behave identically to the cases with the same name on the above Subsection. All the expected answers are now listed:

- Time Interval:

  Please wait a second, I'm on it.

  During the first day of january ['2019-01-01 00:00:00','2019-01-02 00:00:00']:

  The temperature of the NFS10022 (FSS1 Temperature) was between 39 and 41 degC during:

    ['2016-01-01 00:58:00', '2016-01-01 01:01:00']

['2016-01-01 04:13:00', '2016-01-01 05:01:00']

['2016-01-01 05:07:00']

['2016-01-01 06:29:00', '2016-01-01 07:02:00']

['2016-01-01 10:14:00', '2016-01-01 10:19:00']

['2016-01-01 10:21:00', '2016-01-01 11:01:00']

['2016-01-01 12:30:00', '2016-01-01 13:03:00']

['2016-01-01 16:15:00', '2016-01-01 17:02:00']

['2016-01-01 18:25:00']

['2016-01-01 18:30:00', '2016-01-01 19:04:00']

['2016-01-01 22:16:00', '2016-01-01 23:03:00']

The temperature of the NEI00028 (ANP008 TCS TH55-FSS1) was between 39 and 41 degC during:

['2016-01-01 04:39:35']

['2016-01-01 04:41:05', '2016-01-01 05:29:35']

['2016-01-01 05:30:35', '2016-01-01 05:37:05']

['2016-01-01 05:38:05', '2016-01-01 05:45:05']

['2016-01-01 05:46:05', '2016-01-01 06:36:05']

['2016-01-01 06:40:35', '2016-01-01 06:41:05']

['2016-01-01 10:38:05']

['2016-01-01 10:39:35', '2016-01-01 10:40:05']

['2016-01-01 10:42:05', '2016-01-01 11:48:05']

['2016-01-01 11:49:05', '2016-01-01 11:58:35']

['2016-01-01 12:00:05', '2016-01-01 12:07:35']

['2016-01-01 12:08:35', '2016-01-01 12:37:35']

['2016-01-01 16:44:05', '2016-01-01 17:25:35']

['2016-01-01 17:26:35', '2016-01-01 17:30:35']

['2016-01-01 17:31:35', '2016-01-01 18:38:05']

['2016-01-01 22:38:05']

['2016-01-01 22:44:35', '2016-01-01 23:41:05']

Is there anything else you would like to ask?

- No Parameter:

  Sorry, I couldn't find the value velocity of the fss.

  Would you like to ask something else?

- Missing Entity:

  Sorry, I did't understand the data requested.

  Would you like to ask something else?

The returned time intervals are received by the bot from the Analytics module as timestamps, and converted into readable time objects for the answer. All the input parsed entities are present in the answer as well.

4.4.4 Transition Queries

The Transition Queries test case essentially assess Karel's behavior when handling a message regarding the status of an enumerated parameter. This is one of the most important controls done during mission control as it check the nominal operation of the satellite systems. Assuring for instance that some component is not accidentally switched off risking the mission fulfillment. The testing examples operate similarly to the above described. However, the No Parameter case is not established as it is not possible to happen. As explained in Subsection 4.3.1, the reporting data state entity will always be transformed into "state" and seeing that every system element has at least one enumerated data parameter which employs states, from the KG an option will always be returned. If the "state" slot is not correctly assigned, or not specified in the input, the case is processed as a missing reporting data, handled the same way as the above Missing Entity example. The created messages for this test are:

- Status: "number of transitions of the str1 status on 25feb2016"
- On and Off: "Transitions of the str1 from on to off during last week?"
- Missing Entity: "tell me the gyro 1 transitions to on"

All the possible ways of asking for state transitions are covered by the strings, general status changes, from one value to another and simply to a value. When these inputs are classified. The `ActionQueryCheck` acts equally to the other case tests. In this example however, the missing entity is time, which does not constitute a possible requested since all the stored information from the beginning of the mission till today, is an amount of data too large to be processed in a fast and simple way. When the second action is triggered, the answer returned by the Analytics transition method can be processed in two ways. Either values are specified by the user and the function analyzes the API json looking for those values in the parameters data even if zero transitions are found, to always give some valuable information on the state of the parameter. Or all the data strings returned by the Analytics are analyzed and exposed in the output to the user in additions to the number of transitions. Considering the examples defined, the answers should be:

- Status:

  Please wait a second, I'm on it.

  Here's what I found for the status transitions of the str1 during 25feb2016 ['2016-02-25 00:00:00', '2016-02-26 00:00:00']:

  > NST83407 (STR1 TEC current status) was always valid.
  >
  > NST83421 (STR1 PROM loading error) was always FALSE.
  >
  > NDW11315 (STR1 intTimeStatus) was always TRUE.
  >
  > NDW48802 (AOCS basic meas STR1) was always DISABLED.
  >
  > NPC11074 (STR1 mode transition) was always ON.
  >
  > NST83401 (STR1 param out of range) was always valid.
  >
  > NDW48793 (STR1 bkg local status) was always DISABLED.
  >
  > NST83405 (STR1 ASIC RAM overflow) was always valid.
  >
  > NDW48800 (AOCS STR1 unit HK) was always ENABLED.
  >
  > NST83439 (STR1 Voltage2 status) was always OK.

NST82006 (AOCS STR1 meas) had 104 transitions from valid to INVALID.

NST83422 (AOCS STR1 meas part2) was always FALSE.

NST83400 (AOCS STR1 status) was always valid.

Is there anything else you would like to ask?

- On and Off:

  Please wait a second, I'm on it.

  Here's what I found for the transitions from on to off of the str1 during last week ['2018-09-24 00:00:00', '2018-10-01 00:00:00']:

  NPC12128 (RTAhtr112 STR HSTR12-MC) had 0 transitions and was always off.

  NEI00467 (STR1 tmi3 Available) had 0 transitions and was always on.

  NPC13074 (AOCS SM99 STR1 Mode INI) had 0 transitions and was always off.

  NPC11074 (STR1 mode transition) had 0 transitions and was always on.

  NPC12156 (STR1 CPU Wstate regchk) had 0 transitions and was always off.

  Is there anything else you would like to ask?

- Missing Entity:

  Sorry, I did't understand the time period requested.

  Would you like to ask something else?

Only an excerpt of the returned answer is present in the first example to display the idea behind it without extending too much, as the data is not important to asses the bot behavior, only if it was requested to the API, how it is displayed and if all the entities were well classified. By analyzing this first response is possible to determine if all the parameters of the elements are behaving as expected. The second example assists the FCT similarly, although only showing the status important for a specific monitoring. The last is processed the same way as the other homonym cases, this time missing a temporal expression.
All the case tests were preformed in several occasions during the development of Karel's prototype, to assess how the conversational bot was behaving everytime a new model version was created or some changes were made to the code. While the system grew and more types of queries were added, examples had to be build to assess the new functions. The evaluation of the bot, including the test cases, are analyzed in the following Section.

4.5 Discussion

After defining all the evaluating measurements and testing cases, the prototype is complete and its results can be discussed. The values obtain with Section 4.3 and Section 4.4 will be exposed hereafter.
For the NLU Model, the idea that the more varying examples provided, better its capabilities become can be deceiving. The relation between training data and NLP performance is not that linear and too many examples can result in on overfitted model. An overfitted model corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably, which seems to be happening to Karel. In Figure 4.7 The green line represents an overfitted model and the black line represents a regularized model. While

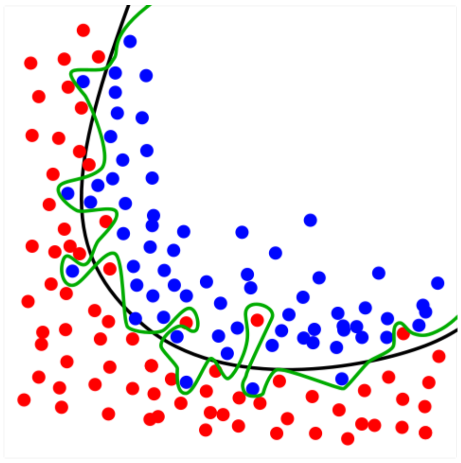


Figure 4.7: Representation of overfitted model from [12].

the green line best follows the training data, it is too dependent on that data and it is likely to have a higher error rate on new unseen data, when compared to the black line [12].
Looking at the evaluation measurements, Karel appears to be almost perfect. However, its important to consider that the confidence score is not a true probability that the prediction is correct, it is just a metric defined by the model that approximately describes how similar the input was to the training data. However, when training a model, the goal is not for it to memorize the training examples, but create a theory that can be generalized across other examples. Meaning that a high confidence measure does not reveal a better model, but usually indicates an overfit [9]. Unless the datasets are symmetric, with the number of false positive and false negatives values being almost the same, which is not the case.
Although it is not intuitively understandable, the F1 score is usually more useful than the accuracy, especially with an uneven class distribution, as it is the case. This measurement is very significant, improving its value when precision and recall are moderately high, reaching a compromise which shows the best classification. On the other hand, high precision relates to the low false positives rate, and recall asses the true positives rate, above 0.5 is already considered good. Values as high as Karel presents, of 100% or close, indicate a probable overfitted NLU model. This may result in typos or different ways of expressing a system element being wrongly classified. Which causes the entities to be set as the exact string from the user message and not its equivalent value, trained by synonyms. Receiving these slots, the Dialogue model will perform as if the parameter does not exist in the Knowledge Graph, since the KG classes are named as the entity values.
The training data should always be representative of the processed data, however testing must challenge the trained model, guaranteeing the correct classification of unknown user input. Karel's intent classification is perceived as even better, but this is also deceiving. The results come from the large difference between the types of sentences possible for each, in addition to the difference in quantity of examples generated for the query asking intent compered to the rest and the similarity between the bye and greet testing and training strings. However this model assures the correct classification of the `ask_query` intent and, this way, the accomplishment of the bot's main goal. The problems appear specially with entity classification. The absence of histogram bars on the left in section C, with the confidence of the wrongly classified intents, confirms this evaluation.
Gathering all the type of queries in the same intent was a request made by VisionSpace technologies to the author of this dissertation, this design choice was based on the possibility to further extend the types of questions handled by the bot, and with the aim of simplifying the Dialogue model. However, this path proved to be harder to handle, not only contributing to the NLP generalization

issue, but specially when developing the dialogue actions. Having the same actions performed in consequence of several types of questions, associated with distinct API calls, requires multiple ways of processing the returned json as well as particular answer formulation for each. This situation emerged especially with the expansion of the Analytics functions.

The results of the Dialogue Model evaluation, displayed in Appendix D, reveal the incorrect prediction of `action_answer`. This can be due to it being chosen from two possibilities based on the value of a slot assigned by an action, `query_check`, since this situation is not possible to consider for this evaluation. The rest of the actions are accurately classified. The correct performance of the Dialogue model is confirmed with the test cases.

All the testing circumstances generated the expected output on Mattermost, confirming the models accuracy when performing the desired mission assistance. The Simple Intents case reveals the correct classification of the intents greet, bye and out of scope, and the right choice of template accordingly, as expected from the evaluations. For the rest of the tests, the intent was always classified as asking query, which is the desired, confirming the strong correct intent classification values. However, this situation does not establishes the performance of the bot.

The Numeric Queries, also acted as expected, excepted when the entities were not identified by one of the trained texts. This condition resulted in Karel acting identically to when no system parameter is mapped in the KG matching the user input's entities together. Thus sending a response stating that no parameter was found and showing the utilized words to search the Graph. When the request is completed, the structure of the answer sent through Mattermost evidences the benefits of using this bot feature. Not only the bot presents the parameters data requested without the SOE having to search the database tables, but it immediately performs numerical analysis on the values as asked and even shows all the values graphically for context and easily visual spotting of anomalies. This may seem simple queries, but are highly helpful controlling sensitive numerical data within its limits or nominal values. Additionally, the dialogue model correctly identified the entity not present in the Missing Entity example, the system element, and the expected answer was given, identifying what was absent.

Time Interval queries can be made regarding a reporting data between two values, or above or below a number. All worked as presumed, returning a list of single timestamps or intervals by the Analytics request, which are then transformed by the answer action into readable time objects. No parameter found and wrongly classified entities behaved similarly to the above discussed cases. The missing entity in this sample was the reporting data, and identically to the numeric example, the response correctly recognized the entity not specified. This type of queries controls mission periods having reporting data within expected values or not acting nominally. For instance time periods when the parameter data was outside its limits, constituting an error or a warning in the expected conduct of the system element. Contrary, it can also be used for a positive check, verifying the data range as expected.

When the defined Transition Queries tests were performed, Karel also conducted as trained to do. The example without temporal expression created an output recognizing this absence, as succeeded in the other corresponding cases. This type of queries return data regarding the state of enumerated parameters, which are identified as strings in the database. The Analytics module returns not only the number of transitions, which in the considered cases were only different than zero once, but all the strings classifying the elements during the requested time period as well. This allows the bot's answer action to analyze the json object received and also retrieve the status of the system if the number of transitions was zero. Considering the mission in a stable phase, most state values will not have many changes, so the current situation adds valuable information. Thus, these questions assure the engineer saves the time of having to go through all the database tables

to find the modes of a system element during a specific period, remembering that each timestamp has a value associated to it, this translates in a lot of data. State transitions give a fast perspective of the elements' behavior and allow the fast monitoring of its nominality. All the original stings used by the database are used to formulate the answer for better connection to it if necessary.
The evaluation of the NLU and Dialogue Models together with the performance of the test cases assure all of Karel's stack modules are well connected and create an operating chatbot for satellite mission operations.

Chapter 5

Conclusions

5.1 Overview

The main goal of this dissertation, as introduced in Chapter 1, was to develop and implement the prototype version of a chatbot applied to satellite mission operation, Karel, with the purpose of improving its current capacity and effectiveness. To achieve this, the generation of a Knowledge Graph on the domain was also necessary and part of the objectives.
Spacecraft Operations Engineers are responsible for ensuring the safety of the satellite, monitoring its systems and providing constant operational support from the ground. To do so, accessing information present in very large datasets, searching for parameters details through many tables of data. For better and faster decision making the chatbot fetches the necessary data and presents it in a visually easy way to retrieve all the necessary information. Saving the Flight Control Team's time of looking on the database.
First, some research on Karel's domain was made, concerning all the aspects of Ground Satellite Operations. Followed by the investigation of chatbots development, which included the systems itself, Natural Language processing as the means to develop it and Knowledge Graphs as the way to connect it with the data.
It is important to consider that each bot needs to be created for a specific mission, as some of the system elements will be particular to each satellite, and need, in addition to the common ones. This conveys the prerequisite of building Karel, foremost choosing a mission.
With the quantity of systems, different reporting data, events and activities, existing related to one satellite, and considering all the different layers and relations linking them, a knowledge graph is very important for the conversational assistant to be a productive system. However, despite the mapping being very successfully done, the KG implemented as detailed in Chapter 4 is very limited.
The bot created as the same Chapter describes answers three types of questions, all classified as the same intent but divided based on the entities parsed and the data analysis necessary. Which are essentially, numerical questions, such as minimum, maximum and average, time interval and transition. All tested in Chapter 4, section? and performing as expected.
Although Karel is still limited to few possible queries, data analysis and satellite systems, the prototype assures the reliability of a chatbot development in this domain, accomplishes all its objectives, and can be used as the base for an advanced version. Main advantages of the developed service are:

- Being a simple and reliable mechanism for accessing trusted data when operating a satellite;
- Being a trusted channel of communication with the databases used by FCTs;
- Improving the efficiency and speed of monitoring and control of all the necessary information sources by the engineering teams;
- Supporting sustainable decisions and reducing the risk of irresponsible actions by giving quick access to reliable information about the satellite status;

- Facilitating on-board new engineers regardless their coding skills;
- Communicating with mission's databases and all sources of documentation;
- Being fully compatible with Ares (ESA product).

Developing this project had some intrinsic challenges, such as the connection of all the stack modules, that required each part to always be conscious of what it receives from and sends to others and ensuring the strings passed between modules make sense inside all of them. Furthermore, as the modules were being developed at the same time, many changes were made during the whole project, which implied changes on the rest of the modules to accommodate the different or new features.
The ever evolving and changeability of the Rasa Platform. Which assures bugs are fixed faster but also caused base parts of Karel to be completely changed throughout its development. So it can be considered another overcomed difficulty. Covering all the possible query cases in the same actions of the core module was another difficulty felt while creating this bot. Each possible combination of entities had to be considered in every action, extending the size and complexity of the code.
Additionally, people want to use natural language to communicate with computers, the same way it is used for human communication, which constitutes another challenge in the completion of this project. This conjugated with the fact that ESA's FCTs are made of many people from different nationalities and backgrounds results in the need to suppose several ways of asking for the same information. Thus the NLU model must be trained on many random example messages and prepared to easily classify never before seen sentences.

5.2 Recommendations for Future Work

To further extend the applicability of Karel and create a fully operable conversational assistant based on the prototype developed for this dissertation, plenty of development can still be done. Something important to create even without additional system growth is a user manual, informing how do the users communicate with the Chatbot, and in what ways it can help them.
One of the most important actions for the near future is the expansion of the bot's Knowledge Graph. This requires extensive work and as the practical KG is far from completed and every new mission will demand appendages, its further generation automation may also be considered. Growth can subsequently be done not only regarding the spacecraft systems, but also Ground Station information, satellite and mission operation documentation, and system reports. On-demand documentation querying would allow users to fetch a given document, or a set, with minimum effort, even allowing users to visualize it on the browser rather than searching a local or remote library to access it. This feature would reduce the time users need to access data and can also be a future addition.
Regarding Rasa and again considering its open source attributes, a new pipeline for the NLU training now exists, which is the recommended for large number of input examples used to train the bot, as is the casa of Karel. So the use of this pipeline should be discussed, if this platform continues to be used for the NLP. The author of this dissertation also suggests the implementation of more intents to divide the currently supported types of questions. Which will decrease the necessary number of training examples, as well as the complexity of the Dialogue actions. If this path is taken, Rasa's form action should also be considered allied to the interactive training possibility. Since these actions automatically ask the user for required entities that are not present in the input, but were not possible to use with the one intent policy, and the advised way to train

these actions is through interactive learning, as it explores more possible conversations, assuring all the possible story paths are contemplated.
Besides the creation of more analysis modules or the integration with an analytics platform such as Elasticsearch which allow the incorporation of new types of queries, the possibility of adding active bot characteristics must be explored and even be one of the objectives of the next bot version. When implemented, the property could essentially consist of user messages that make the bot execute an operation in a system. This may later allow the bot to further assist decision making and premonitions of events or malfunctions. Furthermore, some features could improve user experience, and better assist the ground team, such as:

- Export of the dynamic plots to other formats;
- Integration with productivity apps;
- Autocomplete with a custom backend service;
- Notifications;
- Voice-based queries;
- Active bot activities.

Overall, the recommendation made in this Section aim at either improving the bot's models, or adding interesting features in the functionality perspective. However, the two are interconnected, as without correcting the NLU flaws and subsequently the Dialogue model, it is not easy to apply new more complex features. On the other hand, if new attributes are not expected to be implemented, the current models preform well and do not require further expansion.

Bibliography

[1] D. Braun, A. Hernandez-Mendez, F. Matthes, and M. Langen, “Evaluating natural language understanding services for conversational question answering systems,” in *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pp. 174–185, 2017.

[2] M. McTear, “Conversational modelling for chatbots: Current approaches and future directions,” tech. rep., Technical report, Ulster University, Ireland, 2018.

[3] D. J. Stoner, L. Ford, and M. Ricci, “Simulating military radio communications using speech recognition and chat-bot technology,” *The Titan Corporation, Orlando*, 2004.

[4] “File:Latex-dependency-parse-example-with-tikz-dependency.png.” `https://commons.wikimedia.org/wiki/File:Latex-dependency-parse-example-with-tikz-dependency.png`. Accessed: 03-09-2018.

[5] “Announcing SyntaxNet: The World’s Most Accurate Parser Goes Open Source.” `https://ai.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html`. Accessed: 07-09-2018.

[6] S. Yang, L. Zou, Z. Wang, J. Yan, and J.-R. Wen, “Efficiently answering technical questions-a knowledge graph approach.,” in *AAAI*, pp. 3111–3118, 2017.

[7] S. W.-t. Yih, M.-W. Chang, X. He, and J. Gao, “Semantic parsing via staged query graph generation: Question answering with knowledge base,” 2015.

[8] T. Bocklisch, J. Faulker, N. Pawlowski, and A. Nichol, “Rasa: Open source language understanding and dialogue management,” *arXiv preprint arXiv:1712.05181*, 2017.

[9] “Training spaCy’s Statistical Models.” `https://spacy.io/usage/training`. Accessed: 25-09-2018.

[10] “Choosing a Rasa NLU Pipeline.” `https://rasa.com/docs/nlu/master/choosing_pipeline/`. Accessed: 25-09-2018.

[11] “Gaia creates richest Star Map of our Galaxy - and beyond.” `http://www.esa.int/Our_Activities/Space_Science/Gaia/Gaia_creates_richest_star_map_of_our_Galaxy_and_beyond`. Accessed: 30-09-2018.

[12] “Overfitting.” `https://en.wikipedia.org/wiki/Overfitting`. Accessed: 02-10-2018.

[13] “A Comparative Analysis of ChatBots APIs.” `https://activewizards.com/blog/a-comparative-analysis-of-chatbots-apis/`. Accessed: 07-09-2018.

[14] “Chatbots: Past, present, & future.” `https://chatbotslife.com/chatbots-past-present-future-13a5cb026b18`. Accessed: 27-09-2018.

[15] B. A. Shawar and E. Atwell, *A comparison between Alice and Elizabeth chatbot systems.* University of Leeds, School of Computing research report 2002.19, 2002.

[16] E. D. Liddy, “Natural language processing,” 2001.

[17] B. A. Shawar and E. Atwell, “Chatbots: are they really useful?,” in *Ldv Forum*, vol. 22, pp. 29–49, 2007.

[18] A. Khanna, B. Pandey, K. Vashishta, K. Kalia, B. Pradeepkumar, and T. Das, "A study of today's ai through chatbots and rediscovery of machine intelligence," *Int. J. ue Serv. Sci. Technol*, vol. 8, no. 7, pp. 277–284, 2015.

[19] "5. Satellite Operations." `https://www.esa.int/Our_Activities/Human_Spaceflight/International_Space_Station/Control_centres`. Accessed: 03-09-2018.

[20] P. Ferri and M. Warhaut, "Cluster mission operations," *Space Science Reviews*, vol. 79, no. 1-2, pp. 475–485, 1997.

[21] R. Much, P. Barr, L. Hansson, E. Kuulkers, P. Maldari, J. Nolan, T. Oosterbroek, A. Orr, A. Parmar, M. Schmidt, *et al.*, "The integral ground segment and its science operations centre," *Astronomy & Astrophysics*, vol. 411, no. 1, pp. L49–L52, 2003.

[22] "Ground segment." `https://en.wikipedia.org/wiki/Ground_segment`. Accessed: 03-09-2018.

[23] D. Hastings and H. McManus, "Space system architecture: Final report of ssparc: the space systems, policy, and architecture research consortium (thrust i and ii)," 2004.

[24] "13 Behind-the-Scenes Secrets of NASA Mission Controllers." `http://mentalfloss.com/article/92769/13-behind-scenes-secrets-nasa-mission-controllers`. Accessed: 03-09-2018.

[25] "The Complete Beginner's Guide To Chatbots." `https://chatbotsmagazine.com/the-complete-beginner-s-guide-to-chatbots-8280b7b906ca`. Accessed: 03-09-2018.

[26] A. M. TURING, "I.—computing machinery and intelligence," *Mind*, vol. LIX, no. 236, pp. 433–460, 1950.

[27] A. OETTINGER, "Eliza a computer program for the study of natural language communication between man and machine,"

[28] "Ultimate Guide to Leveraging NLP & Machine Learning for your Chatbot." `https://chatbotslife.com/ultimate-guide-to-leveraging-nlp-machine-learning-for-you-chatbot-531ff2dd870c`. Accessed: 07-09-2018.

[29] "Deep Learning for Chatbots, Part 1 – Introduction." `http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction/`. Accessed: 07-09-2018.

[30] D. Dutta, "Developing an intelligent chat-bot tool to assist high school students for learning general knowledge subjects," tech. rep., Georgia Institute of Technology, 2017.

[31] "25 Chatbot Platforms: A Comparative Table." `https://chatbotsjournal.com/25-chatbot-platforms-a-comparative-table-aeefc932eaff`. Accessed: 07-09-2018.

[32] "Why Amazon's Echo is totally dominating - and what Google, Microsoft, and Apple have to do to catch up." `https://www.businessinsider.in/Why-Amazons-Echo-is-totally-dominating-and-what-Google-Microsoft-and-Apple-have-to-do-to-catch-articleshow/56539114.cms`. Accessed: 15-09-2018.

[33] "Google Assistant." `https://assistant.google.com/#?modal_active=none`. Accessed: 15-09-2018.

[34] “A Cortana é sua assistente digital verdadeiramente pessoal.” `https://www.microsoft.com/pt-br/windows/cortana`. Accessed: 15-09-2018.

[35] “Siri does more than ever. Even before you ask..” `https://www.apple.com/siri/`. Accessed: 15-09-2018.

[36] “Mitsuku. Your new artificially intelligent BFF..” `https://www.kik.com/bots/pandorabots/`. Accessed: 15-09-2018.

[37] “What Is Wolfram|Alpha?.” `http://www.wolframalpha.com/tour/`. Accessed: 15-09-2018.

[38] “Our newest innovation: chatbots and satellite imaging.” `http://geocento.com/blog/`. Accessed: 15-09-2018.

[39] “”Hello, I am CIMON!”.” `https://www.airbus.com/newsroom/press-releases/en/2018/02/hello--i-am-cimon-.html`. Accessed: 15-09-2018.

[40] J. Hill, W. R. Ford, and I. G. Farreras, “Real conversations with artificial intelligence: A comparison between human–human online conversations and human–chatbot conversations,” *Computers in Human Behavior*, vol. 49, pp. 245–250, 2015.

[41] E. Cambria and B. White, “Jumping nlp curves: A review of natural language processing research,” *IEEE Computational intelligence magazine*, vol. 9, no. 2, pp. 48–57, 2014.

[42] G. G. Chowdhury, “Natural language processing,” *Annual review of information science and technology*, vol. 37, no. 1, pp. 51–89, 2003.

[43] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, “Natural language processing: an introduction,” *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544–551, 2011.

[44] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, “Natural language processing (almost) from scratch,” *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[45] E. D. Liddy, W. Paik, and E. S.-l. Yu, “Natural language processing system for semantic vector representation which accounts for lexical ambiguity,” Feb. 16 1999. US Patent 5,873,056.

[46] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent trends in deep learning based natural language processing,” *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[47] B. Habert, G. Adda, M. Adda-Decker, P. B. de Marëuil, S. Ferrari, O. Ferret, G. Illouz, and P. Paroubek, “Towards tokenization evaluation,” in *Proceedings of LREC*, vol. 98, pp. 427–431, 1998.

[48] N. Nicolov, K. Bontcheva, G. Angelova, and R. Mitkov, *Recent Advances in Natural Language Processing III: Selected Papers from RANLP 2003*, vol. 260. John Benjamins Publishing, 2004.

[49] E. Brill, “A simple rule-based part of speech tagger,” in *Proceedings of the third conference on Applied natural language processing*, pp. 152–155, Association for Computational Linguistics, 1992.

[50] "About Syntactic & Semantic Parsing, howpublished = `http://demo.ark.cs.cmu.edu/parse/about.html`." Accessed: 03-09-2018.

[51] M. Färber, B. Ell, C. Menne, and A. Rettinger, "A comparative survey of dbpedia, freebase, opencyc, wikidata, and yago," *Semantic Web Journal*, vol. 1, pp. 1–5, 2015.

[52] X. Wang, P. Kapanipathi, R. Musa, M. Yu, K. Talamadupula, I. Abdelaziz, M. Chang, A. Fokoue, B. Makni, N. Mattei, *et al.*, "Improving natural language inference using external knowledge in the science questions domain," *arXiv preprint arXiv:1809.05724*, 2018.

[53] "WTF is a knowledge graph?." `https://hackernoon.com/wtf-is-a-knowledge-graph-a16603a1a25f`. Accessed: 15-09-2018.

[54] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes.," in *AAAI*, vol. 14, pp. 1112–1119, 2014.

[55] "The Beginner's Guide to Google's Knowledge Graph." `https://neilpatel.com/blog/the-beginners-guide-to-the-googles-knowledge-graph/`. Accessed: 15-09-2018.

[56] "ConceptNet. An open, multilingual knowledge graph." `http://conceptnet.io`. Accessed: 15-09-2018.

[57] L. Hirschman and R. Gaizauskas, "Natural language question answering: the view from here," *natural language engineering*, vol. 7, no. 4, pp. 275–300, 2001.

[58] "Resource Description Framework (RDF)." `https://searchmicroservices.techtarget.com/definition/Resource-Description-Framework-RDF`. Accessed: 29-09-2018.

[59] "The RDF Advantages Page." `https://www.w3.org/RDF/advantages.html`. Accessed: 29-09-2018.

[60] "Duckling. The linguist duck that parses text into structured data." `https://duckling.wit.ai`. Accessed: 25-09-2018.

[61] "Training Data Format." `https://rasa.com/docs/nlu/master/dataformat/`. Accessed: 25-09-2018.

[62] "Chatito GitHub Repository." `https://github.com/rodrigopivi/Chatito`. Accessed: 25-09-2018.

[63] "User Guide: Actions." `https://rasa.com/docs/core/customactions/`. Accessed: 25-09-2018.

[64] "Training and Policies." `https://rasa.com/docs/core/policies/`. Accessed: 25-09-2018.

[65] "Evaluating and Improving Models." `https://rasa.com/docs/nlu/master/evaluation/`. Accessed: 02-10-2018.

[66] "Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures." `https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/`. Accessed: 02-10-2018.

[67] "Evaluating and Testing." `https://rasa.com/docs/core/evaluation/`. Accessed: 02-10-2018.

[68] "Test case." `https://en.wikipedia.org/wiki/Test_case`. Accessed: 02-10-2018.

Appendix A

Chatbot Training

NLU Model Configuration

```
language: "en"

pipeline:
- name: "nlp_spacy"
- name: "tokenizer_spacy"
- name: "intent_entity_featurizer_regex"
- name: "intent_featurizer_spacy"
- name: "ner_crf"
- name: "ner_synonyms"
- name: "intent_classifier_sklearn"
- name: "ner_duckling_http"
  url: "http://localhost:8000"
  dimensions: ["time"]
  timezone: "Europe/Berlin"
```

NLU model Training

```
def train_nlu():

    logging.basicConfig(level="DEBUG")

    training_data = loading.load_data(training_data_file)

    trainer = Trainer(config.load(nlu_config))

    trainer.train(training_data)

    model_directory = trainer.persist(model_path,
                                      project_name='karel',
                                      fixed_model_name=model_version)
    return model_directory
```

Dialogue model Training

```
def train_dialogue():
    logging.basicConfig(level="DEBUG")

    fallback = FallbackPolicy(fallback_action_name="utter_default",
                              core_threshold=0.3,
                              nlu_threshold=0.3)

    agent = Agent(domain_file,
                  policies = [MemoizationPolicy(), KerasPolicy(), fallback])

    training_data = agent.load_data(stories_file)

    agent.train(
            training_data,
            epochs = 200,
            batch_size = 100,
            validation_split = 0.2)

    agent.persist(model_path)

    return agent
```

Appendix B

Sentence Classification Example

```
{
  "entities": [
    {
      "extractor": "ner_crf",
      "confidence": 0.9998865308134985,
      "end": 20,
      "value": "minimum",
      "entity": "query_type_num",
      "start": 13
    },
    {
      "extractor": "ner_crf",
      "confidence": 0.9999908062803147,
      "end": 32,
      "value": "temperature",
      "entity": "reporting_data_num",
      "start": 21
    },
    {
      "extractor": "ner_crf",
      "confidence": 0.9999555816849093,
      "end": 54,
      "processors": [
        "ner_synonyms"
      ],
      "value": "str1",
      "entity": "system_element",
      "start": 40
    },
    {
      "extractor": "ner_duckling_http",
      "confidence": 1.0,
      "end": 65,
      "text": "last month",
      "value": "2018-09-01T00:00:00.000+02:00",
      "entity": "time",
      "start": 55,
      "additional_info": {
        "grain": "month",
        "values": [
          {
            "type": "value",
            "grain": "month",
            "value": "2018-09-01T00:00:00.000+02:00"
          }
        ],
        "value": "2018-09-01T00:00:00.000+02:00",
        "type": "value"
      }
    }
  ],
  "intent": {
    "confidence": 0.9969311832915636,
    "name": "ask_query"
  },
  "text": "What was the minimum temperature of the star tracker 1 last month?",
  "intent_ranking": [
    {
      "confidence": 0.9969311832915636,
      "name": "ask_query"
    },
    {
      "confidence": 0.0030322849466310163,
      "name": "out_of_scope"
    },
    {
      "confidence": 2.2431887047194716e-05,
      "name": "greet"
    },
    {
      "confidence": 1.4099874758325786e-05,
      "name": "bye"
    }
  ]
}
```

Appendix C

NLU Model Evaluation

Intent and Entity Evaluation

```
Intent evaluation results:
    Intent Evaluation: Only considering those 30038 examples that have
a defined intent out of 30038 examples
    F1-Score:  1.0
    Precision: 1.0
    Accuracy:  1.0
    Classification report:
              precision    recall  f1-score   support

   ask_query       1.00      1.00      1.00     30000
         bye       1.00      1.00      1.00        10
       greet       1.00      1.00      1.00        13
out_of_scope       1.00      1.00      1.00        15

 avg / total       1.00      1.00      1.00     30038

Entity evaluation results:
    Evaluation for entity extractor: ner_crf
    F1-Score:  0.999694355146
    Precision: 0.999695359884
    Accuracy:  0.999693896375
    Classification report:
                      precision    recall  f1-score   support

           no_entity       1.00      1.00      1.00    283823
     query_type_cond       1.00      1.00      1.00     60479
     query_type_enum       0.95      0.97      0.96      1840
      query_type_num       0.96      0.95      0.96      1195
  reporting_data_num       1.00      1.00      1.00     39100
reporting_data_state       1.00      1.00      1.00      1732
reporting_data_value       1.00      1.00      1.00     43924
      system_element       1.00      1.00      1.00     61204

         avg / total       1.00      1.00      1.00    493297

Finished evaluation
```

Intent Confusion Matrix

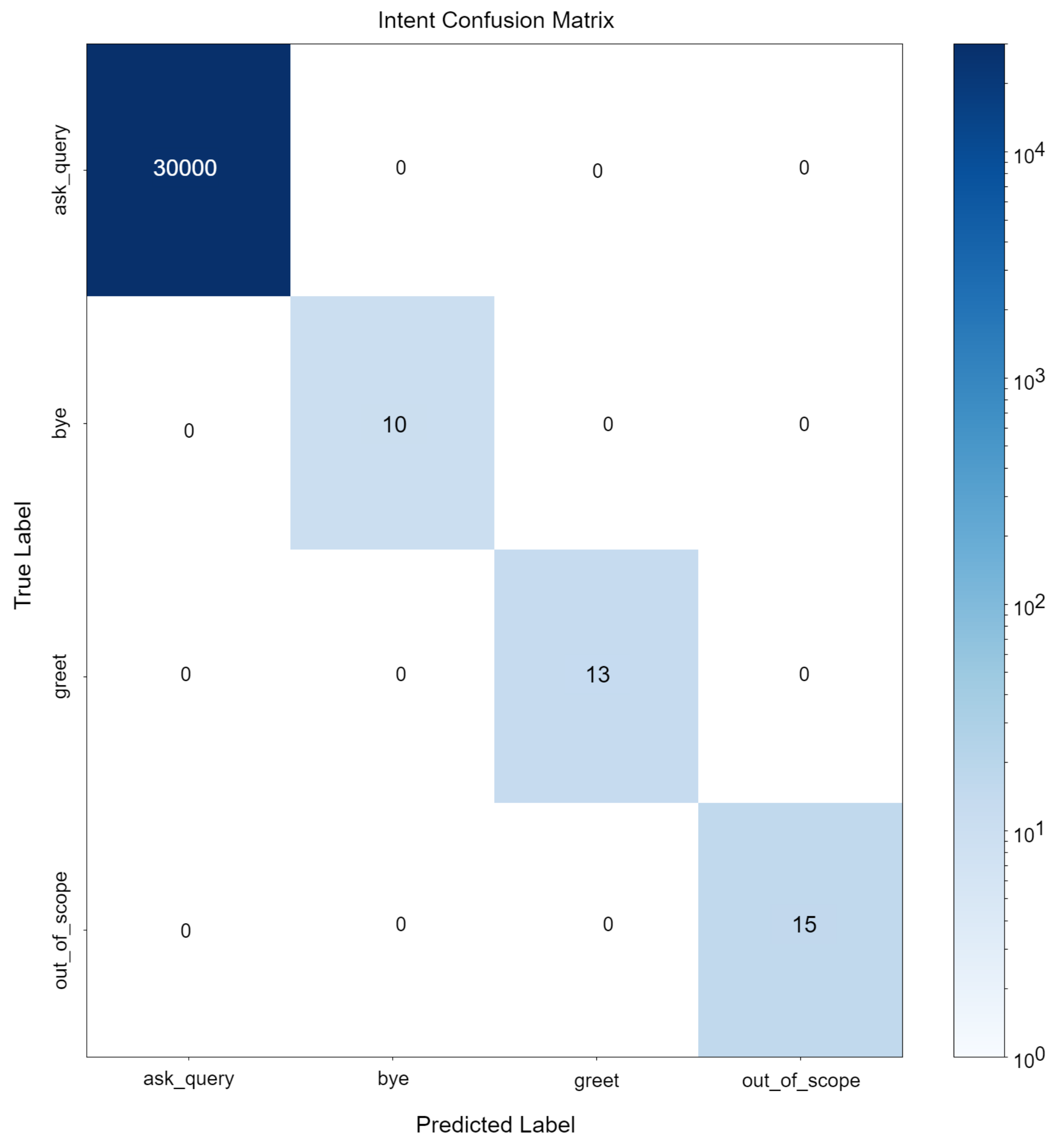


Figure C.1: NLU Model Intent Confusion Matrix.

Intent Prediction Confidence

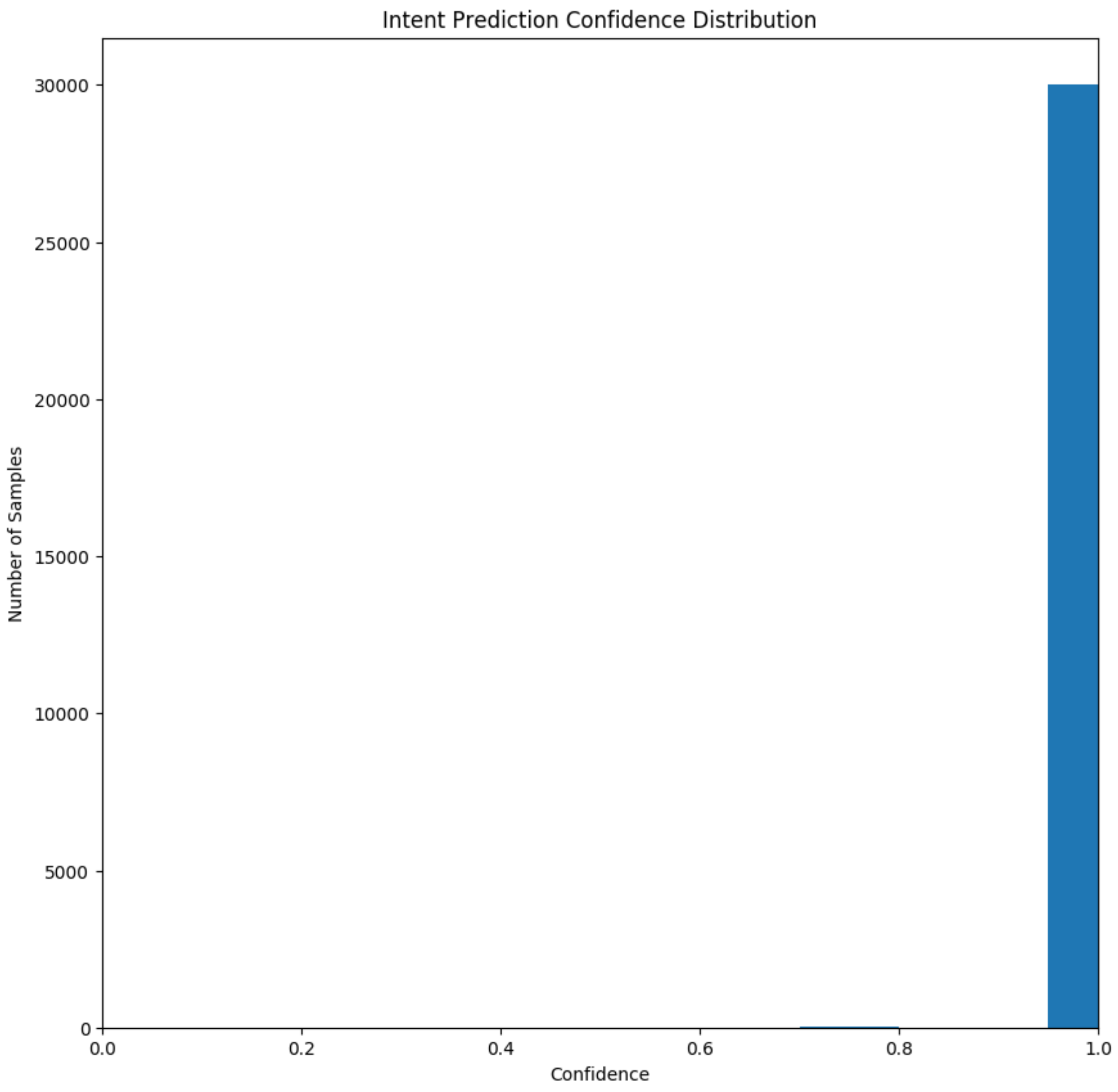


Figure C.2: NLU Model Intent Prediction Confidence Distribution.

Appendix D

Dialogue Model Evaluation

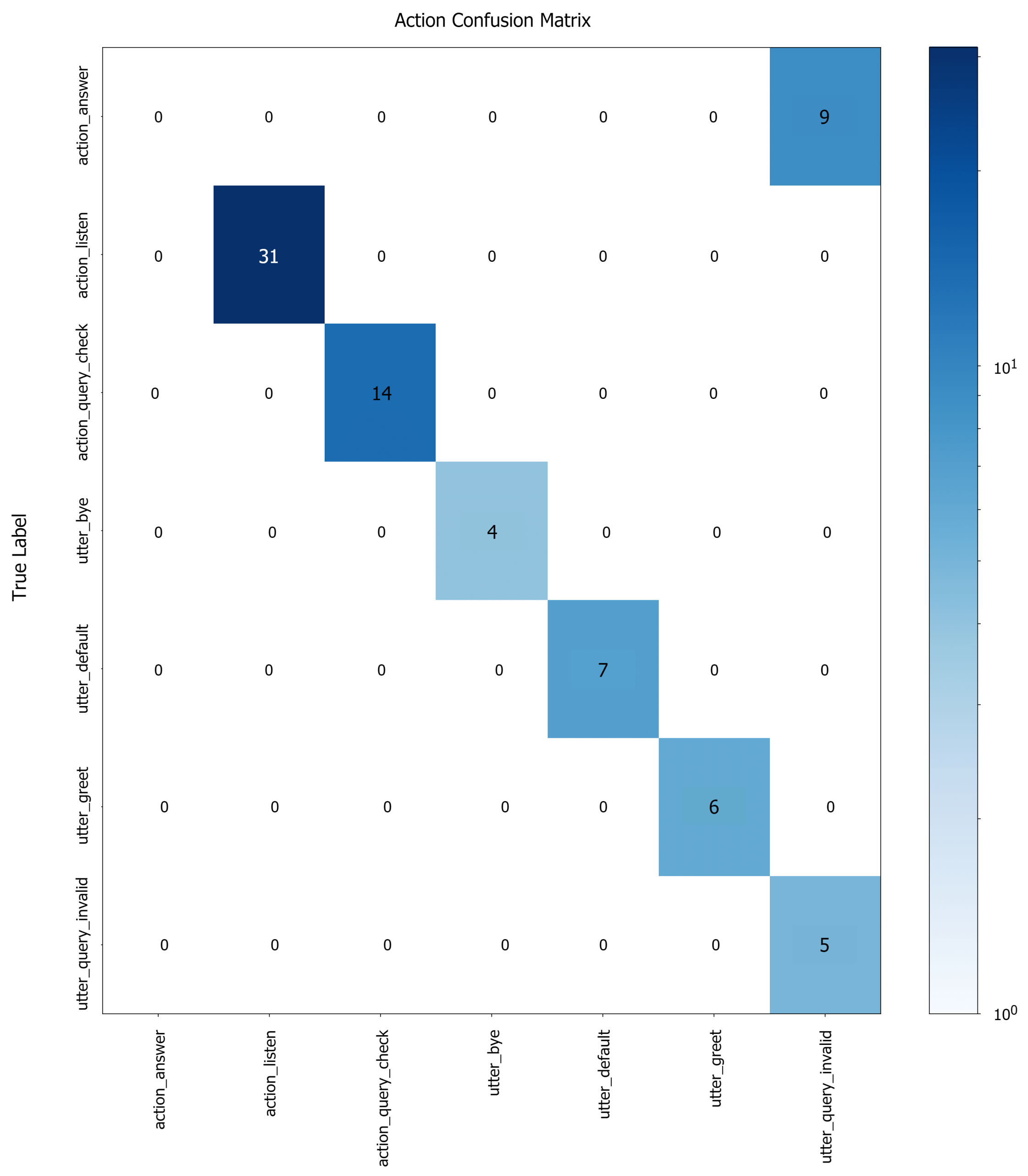
Failed Stories

```
## failed story 0
utter_default
action_listen
action_query_check
utter_query_invalid
action_listen
action_query_check
action_answer                       predicted: utter_query_invalid
action_listen

## failed story 1
utter_greet
action_listen
utter_default
action_listen
utter_default
action_listen
action_query_check
action_answer                       predicted: utter_query_invalid
action_listen

## failed story 2
utter_greet
action_listen
action_query_check
utter_query_invalid
action_listen
action_query_check
action_answer                       predicted: utter_query_invalid
action_listen
```

Actions Confusion Matrix



	action_answer	action_listen	action_query_check	utter_bye	utter_default	utter_greet	utter_query_invalid
action_answer	0	0	0	0	0	0	9
action_listen	0	31	0	0	0	0	0
action_query_check	0	0	14	0	0	0	0
utter_bye	0	0	0	4	0	0	0
utter_default	0	0	0	0	7	0	0
utter_greet	0	0	0	0	0	6	0
utter_query_invalid	0	0	0	0	0	0	5

Figure D.1: Dialogue Model Action Confusion Matrix.

Appendix E

Plots from the Numeric Queries Test Cases

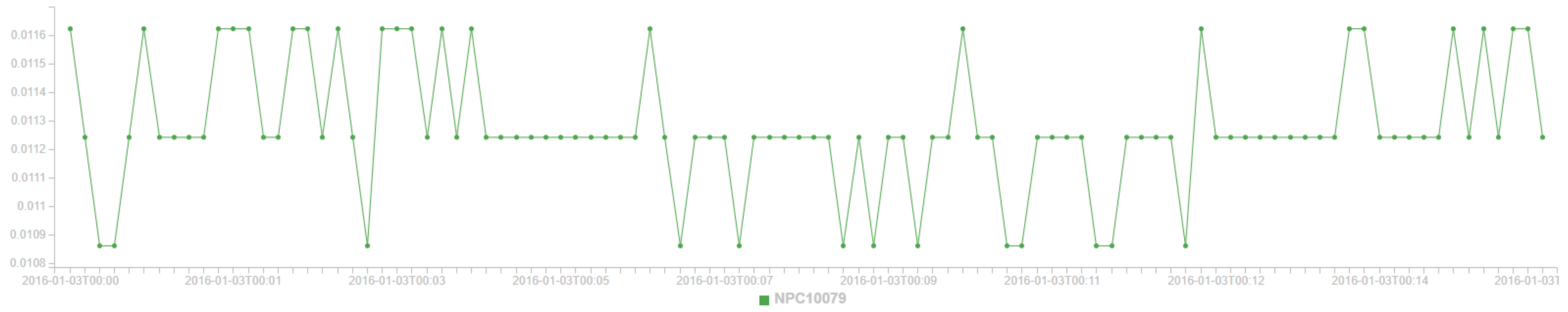


Figure E.1: Maximum current of the NPC10079 parameter, part of the cps.

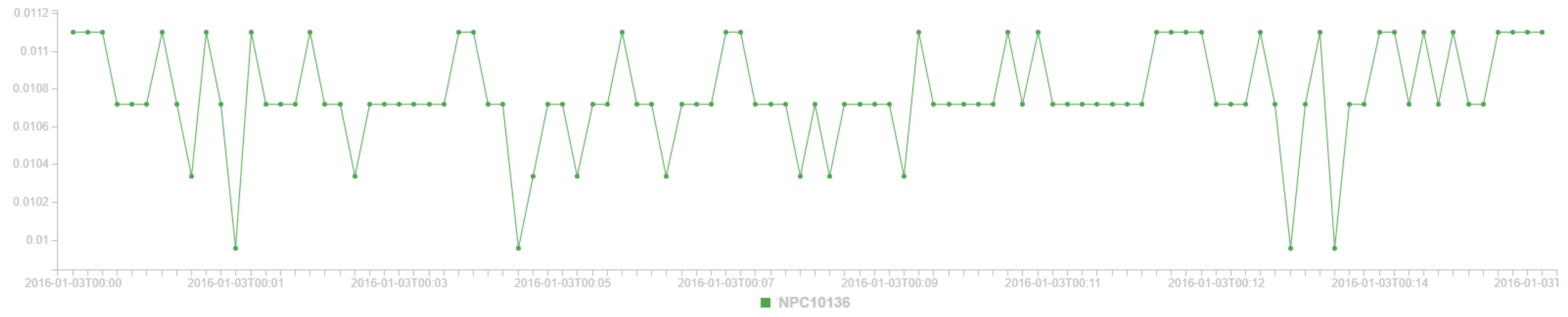


Figure E.2: Maximum current of the NPC10136 parameter, part of the cps.

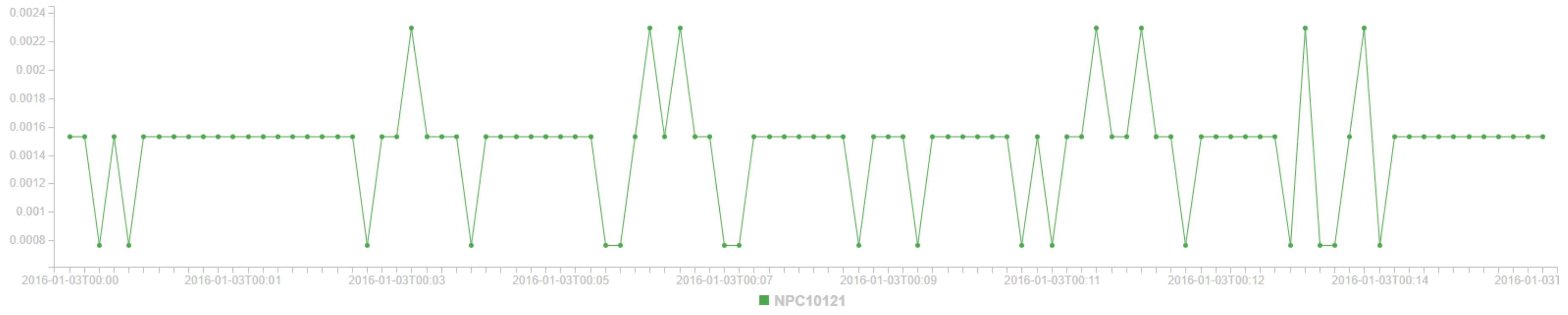


Figure E.3: Maximum current of the NPC10121 parameter, part of the cps.

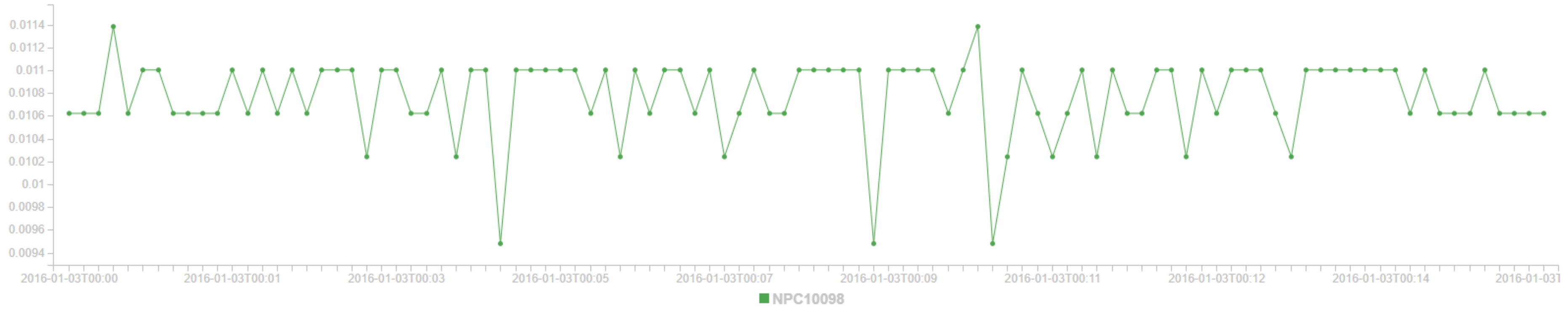


Figure E.4: Maximum current of the NPC10098 parameter, part of the cps.

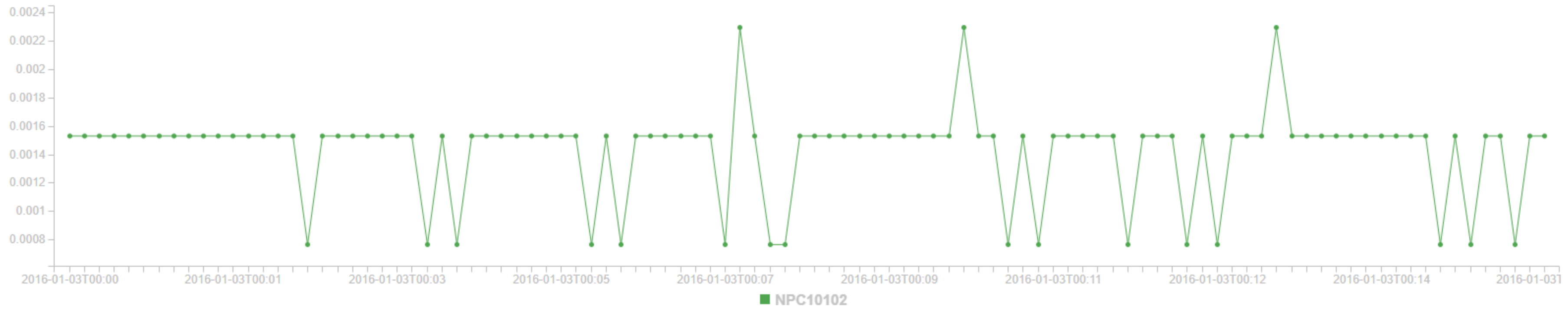


Figure E.5: Maximum current of the NPC10102 parameter, part of the cps.

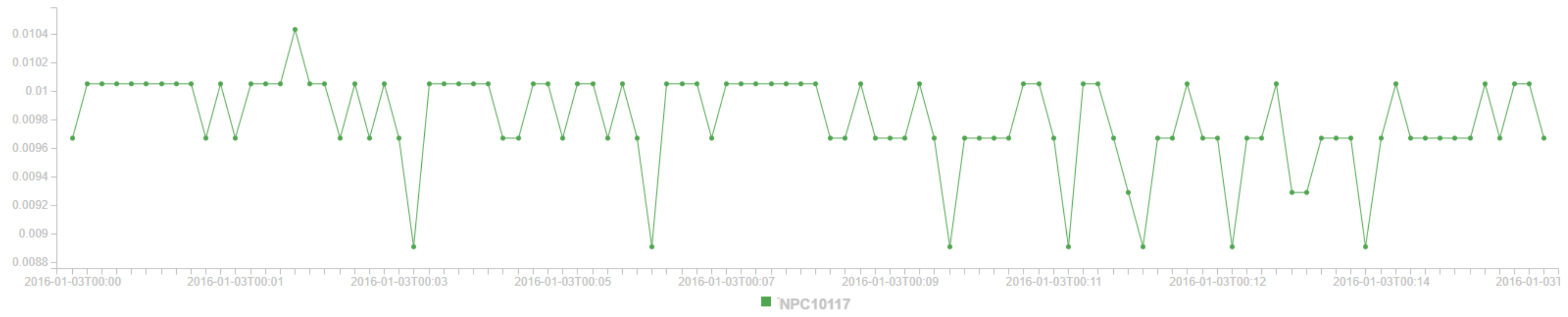


Figure E.6: Maximum current of the NPC10117 parameter, part of the cps.

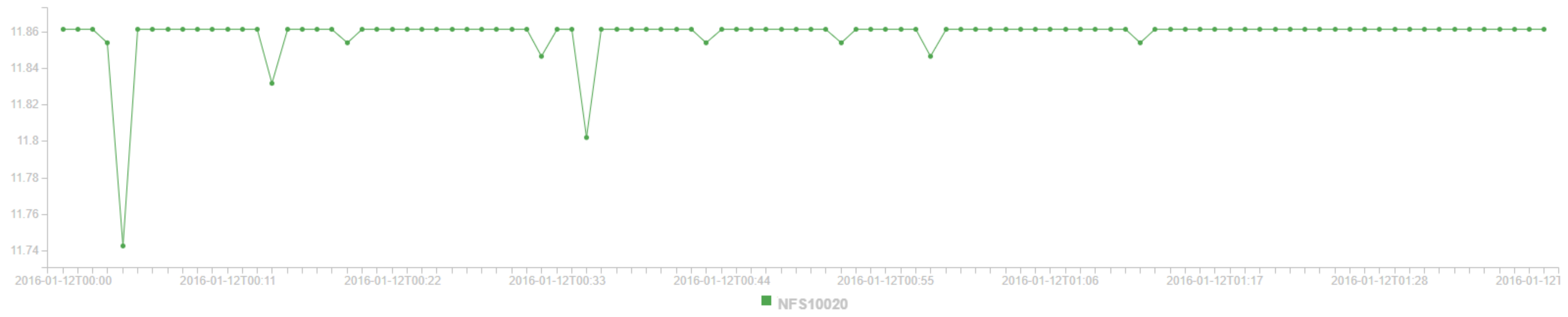


Figure E.7: Minimum voltage of the NFS10020 parameter, part of the fss3.

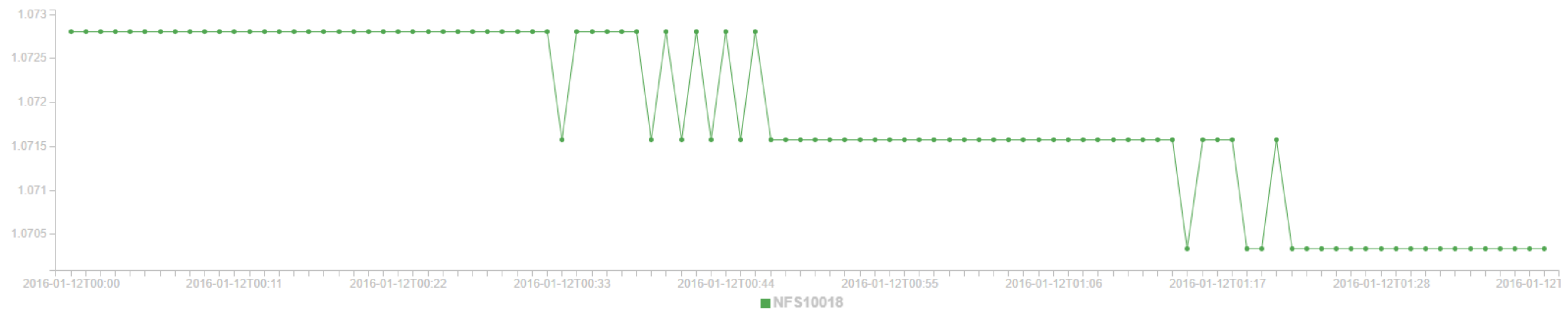


Figure E.8: Minimum voltage of the NFS10018 parameter, part of the fss3.

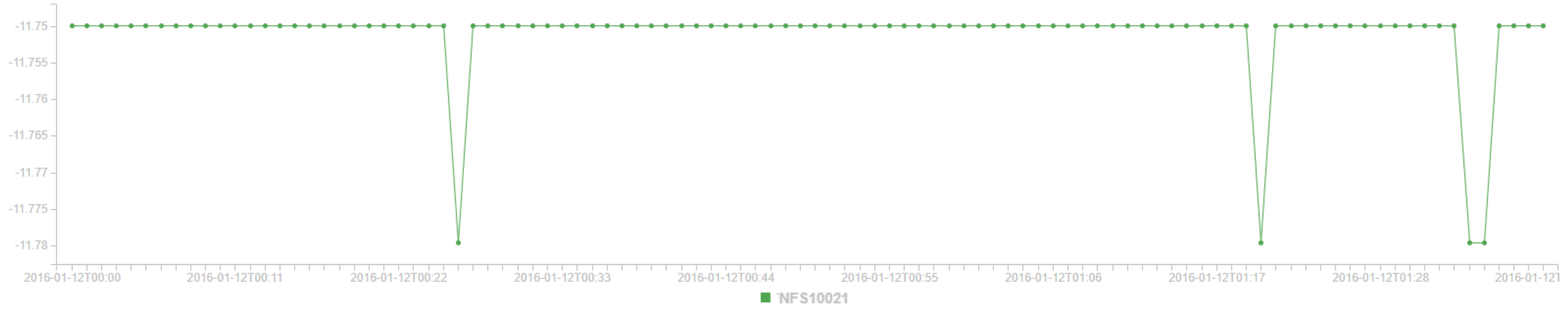


Figure E.9: Minimum voltage of the NFS10021 parameter, part of the fss3.

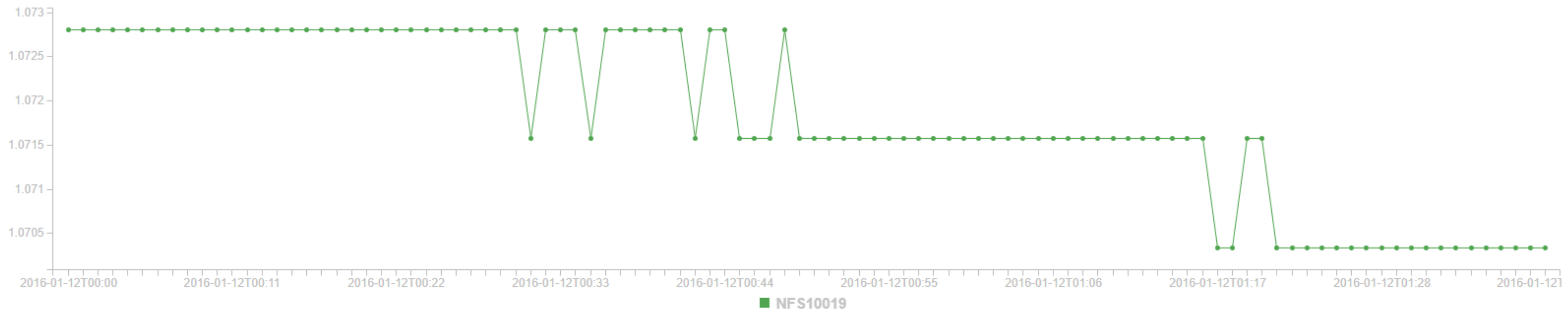


Figure E.10: Minimum voltage of the NFS10019 parameter, part of the fss3.

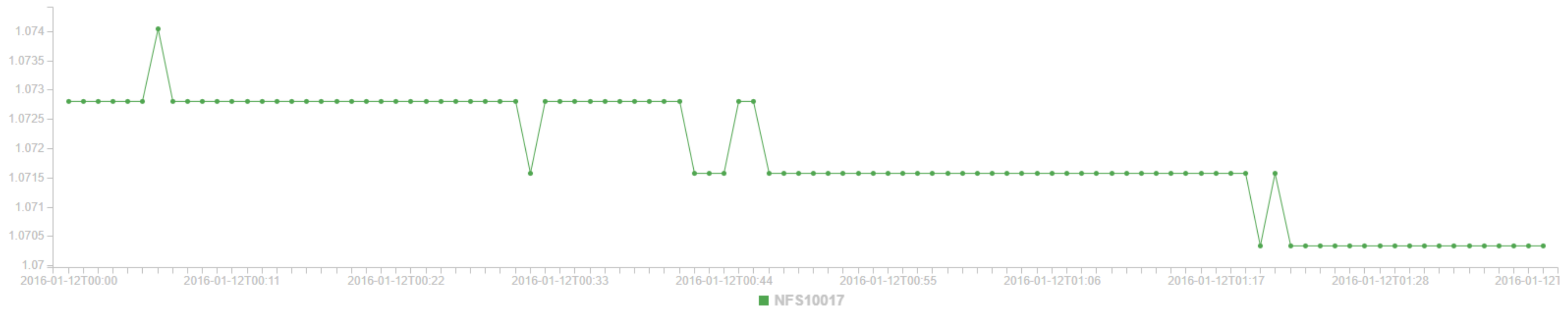


Figure E.11: Minimum voltage of the NFS10017 parameter, part of the fss3.

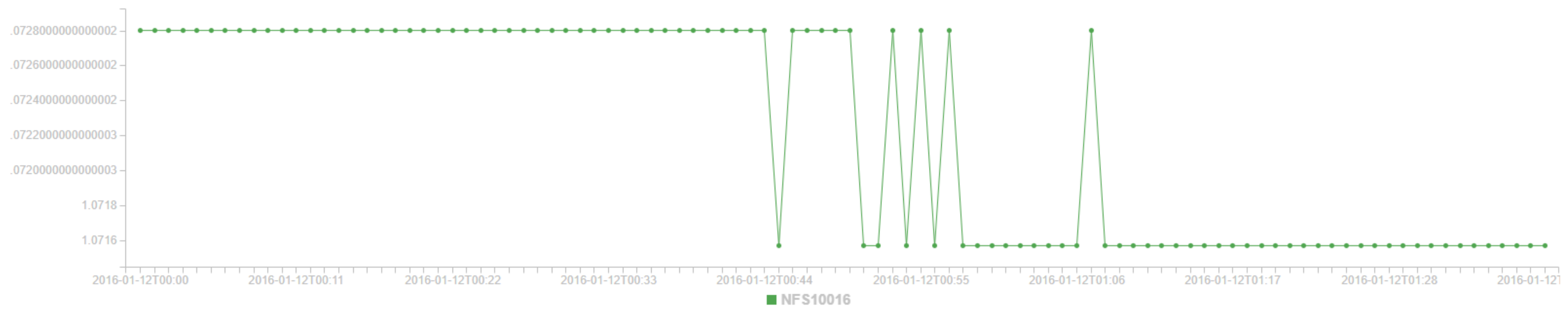


Figure E.12: Minimum voltage of the NFS10016 parameter, part of the fss3.

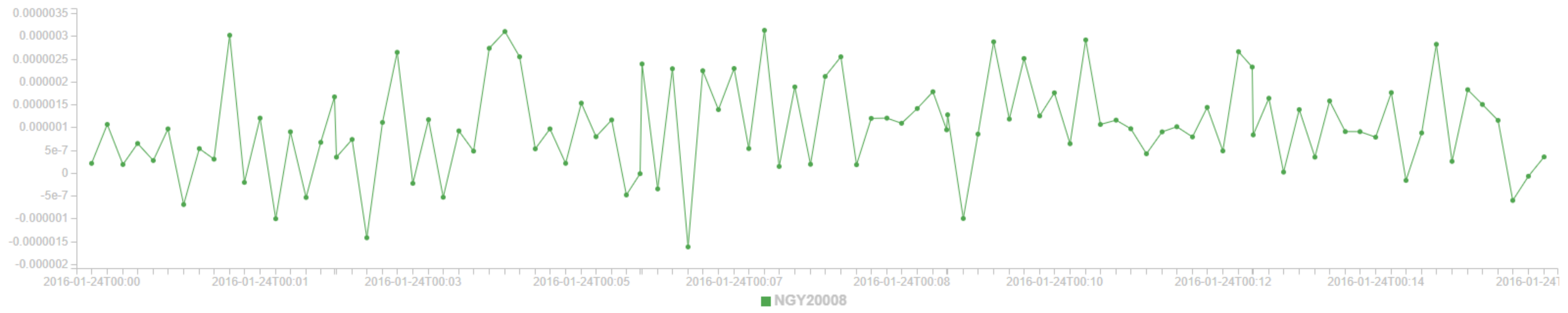


Figure E.13: Average angle rate of the NGY20008 parameter, part of the gyro set.

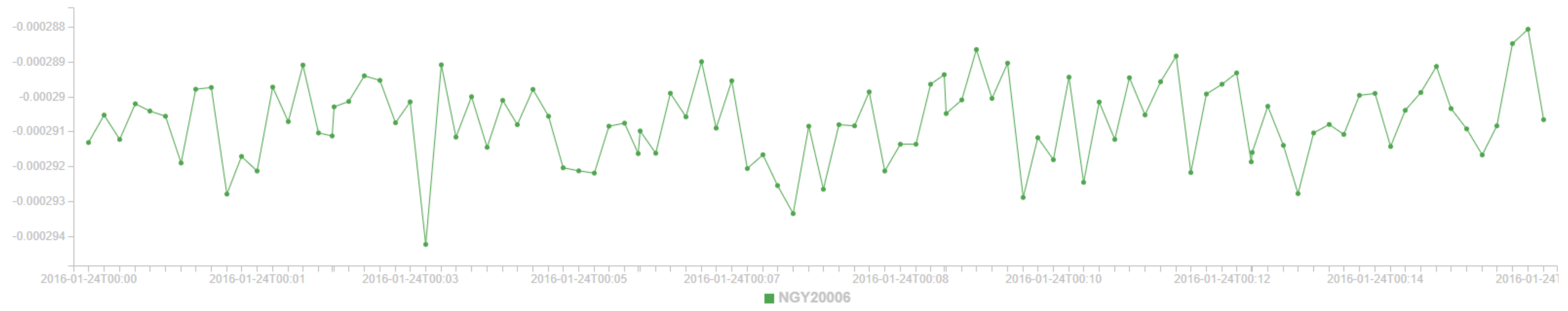


Figure E.14: Average angle rate of the NGY20006 parameter, part of the gyro set.

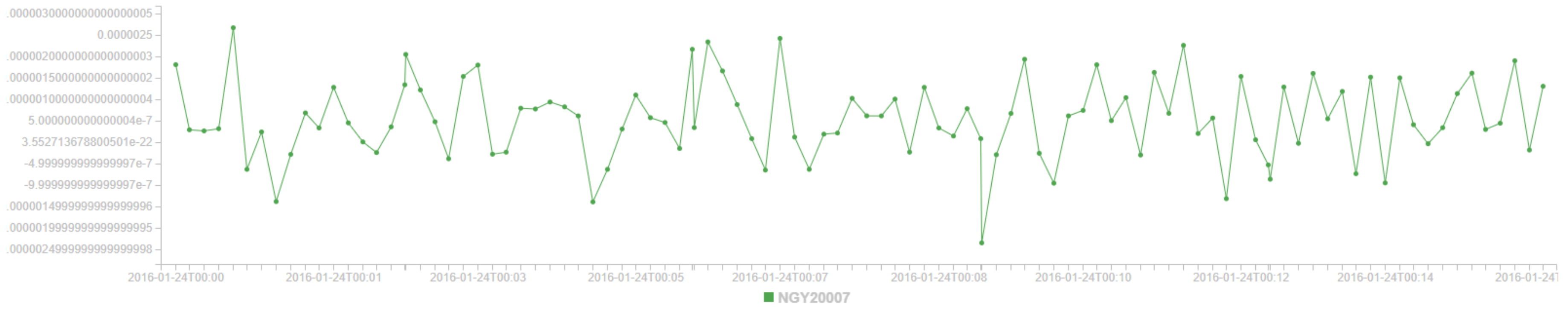


Figure E.15: Average angle rate of the NGY20007 parameter, part of the gyro set.