



Rapport de Stage de Fin d'Études

Polytech Marseille

Spécialité
Informatique

Réalisé par
Yih-Dar SHIEH

Stage du 5 mars au 31 août 2018

Sujet:

Agent Conversationnel Miroir

Tuteurs de stage:

Sandrine CHABOT, Ionel TOTHEZAN, Viviane COHEN

Organisme d'accueil:

Orange, Arcueil - DTSI/DSI - Groupe CCIC

Référent Polytech de stage: Mohamed QUAFAROU



Mots clés: agent conversationnel, chatbot, bot miroir, test automatique

Keywords: conversational agent, chatbot, mirror bot, automatic test

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Tout d'abord, j'adresse mes remerciements à M. Stephane Ayache de Polytech Marseille, qui m'a aidé dans ma recherche de stage pour avoir un contact d'une entreprise.

Je tiens à remercier vivement les personnes du Groupe CCIC: Mme. Sandrine CHABOT - ma tutrice, M. Ionel TOTHEZAN - directeur de l'équipe, Mme. Viviane COHEN et M. Christian LE BLAINVAUX, pour leur accueil, leur support administratif et matériel et le partage de leur expertise au quotidien. En particulier, le soutien de Mme. Viviane COHEN, non seulement comme tuteur mais aussi comme amie, est très apprécié. Enfin, ils m'ont aussi beaucoup aidé dans mes recherches sur mon premier emploi - recommandé mes candidatures pour les postes au sein du Groupe Orange et les postes externes. L'appréciation de leurs actes de gentillesse et leur confiance en moi ne peut pas être exprimée simplement en mots.

J'adresse aussi mes remerciements à M. Alexis BONNECAZE de Polytech Marseille. Il m'a aidé à entrer à Polytech Marseille quand j'ai décidé de m'orienter vers ingénieur informatique après mon diplôme doctorat et quand je n'ai pas encore eu de piste. Il m'a aussi beaucoup aidé pendant mes 2 années à Polytech Marseille - pour des difficultés dans certaines matières ainsi que pour des procédures administratives à l'école.

M. Mohamed QUAFAROU - le directeur du département informatique du Polytech Marseille - m'a aidé à obtenir la signature du directeur de l'école avant qu'il ne soit trop tard. J'ai également apprécié sa compréhension et son autorisation qui me permettent de rédiger ce rapport en anglais.

Enfin, je voudrais exprimer mes remerciements à M. David Kohel - mon directeur de thèse. Après avoir connu ma décision de poursuivre ma carrière en informatique, il m'a présenté son collègue - M. Alexis BONNECAZE. C'était le point de départ d'un nouveau chemin qui me mène là où je suis maintenant, qui correspond à ce que je cherche!

Table des matières

Résumé	3
Abstract	4
1 Introduction	5
1.1 Host organization	5
1.1.1 Orange	5
1.1.2 DTISI/DSI	5
1.1.3 Centre de Compétences Informatique Cognitive	6
1.2 Internship topic	7
1.2.1 Motivation	7
1.2.2 Issues	8
1.2.3 A similar work	9
1.2.4 Topic	9
1.2.5 Mission	10
1.3 Planning	10
1.4 Summary of contributions	10
1.5 Organization of report	11
2 Main work	12
2.1 Study IBM Watson Assistant	12
2.2 Design	14
2.2.1 Generic chatbot model	14
2.2.2 Mirror Bot system	19
2.3 Implementation	22
2.4 Deep learning - A simple approach to generate new intent variations	25
2.5 Results	27
2.6 Test and Evaluation	29
2.7 Future work	32
3 Conclusion	34
3.1 Contribution	34
3.2 Return of internship	34
3.3 School, Internship, MOOC and Career	35
Bibliographie	37

Résumé

Mon stage se déroule chez Orange France, au sein du groupe *Centre de Compétences Informatique Cognitive* (CCIC) dans le département *La Direction des Systèmes d'Information*.

L'objectif de mon stage est de concevoir et de développer le système *Bot Miroir* pour le test automatique de chatbots. Ce système permet de simuler un utilisateur qui communique avec un chatbot du métier ou du domaine pour un service de l'entreprise. Cette nouvelle approche - au moins au sein de DTSI / DSI chez Orange France - évite les coûts et réduit le temps requis par des tests effectués par des humains. Par rapport à la méthode utilisant des phrases prédéfinies, l'avantage est que nous n'avons plus besoin de prédéfinir des phrases pour chaque nouveau chatbot ou bot modifié. De plus, les énoncés générés par le bot miroir sont plus pertinents à la structure interne d'un chatbot à tester, et cela nous permet d'effectuer des tests plus ciblés sur différents chatbots sans effort supplémentaire.

Au cours de mon stage, je complète les missions principales du projet bot miroir en effectuant les tâches suivantes: étudier comment Watson Assistant fonctionne, extraire les concepts essentiels des chatbots, définir et implémenter un modèle générique de chatbot, écrire un script python pour transformer Watson chatbots au format de notre modèle, concevoir l'architecture globale du système Bot Miroir, développer plusieurs modules de réponse pour générer des énoncés, créer un module de décision qui implémente différentes politiques de sélection et de filtrage, effectuer une expérience de génération d'énoncés par deep learning, enrichir le système par différents paramétrages, examiner des conversations générées par bot mirror et identifier les problèmes, et enfin réaliser un test quantitatif simple.

À partir des conversations générées par le système Bot Miroir, nous confirmons que le système développé est utile pour trouver des problèmes dans les chatbots. Cependant, l'examen des dialogues par les humains prend du temps, est fastidieux et sujet aux erreurs. Nous analysons les sources de problèmes dans §2.5, et le test quantitatif dans §2.6 montre que le système peut être utilisé pour détecter des problèmes d'un certain type dans un chatbot de manière automatique. Néanmoins, le système n'est pas prêt pour la production et nous discutons certaines directions futures du projet dans §2.7.

Mon stage est une expérience merveilleuse au cours de laquelle j'évolue vers un NLP data scientist, mes compétences relationnelles et techniques étant renforcées. Grâce à ce stage, avec la formation à Polytech Marseille et l'auto-apprentissage sur le site Coursera, je serai ingénieur en machine learning et NLP dans une start-up travaillant sur des projets passionnants après mon stage!

Abstract

My internship takes place at Orange France, within the group *Centre de Compétences Informatique Cognitive* in the department *La Direction des Systèmes d'Information*.

The objective of my internship is to design and develop the *Mirror Bot* system for the automatic test of chatbots. This system allows to simulate a user who communicates with a business or domain specific chatbot for a service of the company. This new approach - at least within DTSI / DSI in Orange France - avoids the costs and reduces the time required by tests conducted by humans. Compared to the method using predefined sentences, the advantage is that we no longer need to predefine sentences for each new or modified chatbot. In addition, utterances generated by the mirror bot are more relevant to the internal structure of a chatbot to be tested, and this allows us to perform more targeted tests on different chatbots without additional effort.

During my internship, I complete the core missions of the mirror bot project by carrying out the following tasks: study how Watson Assistant works, extract core concepts of chatbots, define and implement a generic chatbot model, write a python script to transform Watson chatbots to our model format, design the global architecture of the Mirror Bot system, develop several response modules to generate utterances, create a decision module which implements different selection and filtering policies, perform an experiment of utterance generation by deep learning, enrich the system by different settings, examine some conversations generated by mirror bot and identify issues, and finally carry out a simple quantitative test.

From the conversations generated by the Mirror Bot system, we confirm that the developed system is helpful to find problems in chatbots. However, examining dialogues by humans is time consuming, tedious and error-prone. We analyze the problem sources in §2.5, and the quantitative test in §2.6 demonstrates the system can be used to detect certain type of chatbot issues in an automatic way. Nevertheless, the system is not production-ready, and we discuss some future directions for the project in §2.7.

My internship is a wonderful experience during which I grow toward a NLP data scientist, with my relational and technical skills being strengthened. Thanks to this internship, together with the training at Polytech Marseille and self-study on Coursera, I will be a NLP machine learning engineer at a start-up working on exciting projects after my internship!

Chapitre 1

Introduction

1.1 Host organization

My internship is at Orange France, within the group *Centre de Compétences Informatique Cognitive* in the department *La Direction des Systèmes d'Information*. In this section, an overview for the company, the department and the group is given.

1.1.1 Orange

Orange is one of the leading European and African mobile and ADSL Internet access operators and one of the world leaders in business telecommunications services. The following list gives an overview of Orange reported in it's annual report 2017¹:

- turnover: 41.1 billion euros
- 273 million individual customers
- 46 million 4G customers
- 4.7 million customers of high speed network
- 152,000 employees

As a French telecommunication company, Orange is ranked the best mobile network in France for the seventh consecutive time according to the Autorité de Régulation des Communications Electroniques et des Postes (ARCEP). To maintain its leadership position, Orange continues to focus on developing new areas such as fiber, 5G and artificial intelligence.

1.1.2 DTSI/DSI

Within Orange France, the Information Systems Department (DSI, La Direction des Systèmes d'Information) is attached to La Direction technique et du système d'information (DTSI). This direction of 3200 employees is aimed at developing and integrating information systems of the company to better meet customer expectations. This involves improving the availability and ergonomics of applications and services. The renovation of the entire information system will, in the long run, make it possible to gain in performance and agility.

1. <https://rai2017.orange.com/orange2017-content/uploads/2018/05/rai-orange-fr-25.05.pdf>

DSI must ensure that all internal users and external customers are satisfied. For this, DSI manages a patrimony of more than 3200 applications which communicate between them for building customer knowledge and supporting the company's various processes.

The strategy that DSI has been implementing since the beginning of 2015 is an integral part of the **Essentiel2020**. Thus, the projects to which DSI contributes allow Orange France to progress significantly on four strategic axes:

- To provide our customers with enhanced connectivity
- To reinvent the customer relationship
- To diversify by capitalizing on the assets
- To build a digital and human employer model

Figure 1.1 gives an overview of different entities inside DSI and Figure 1.2 shows some projects of each entity.²

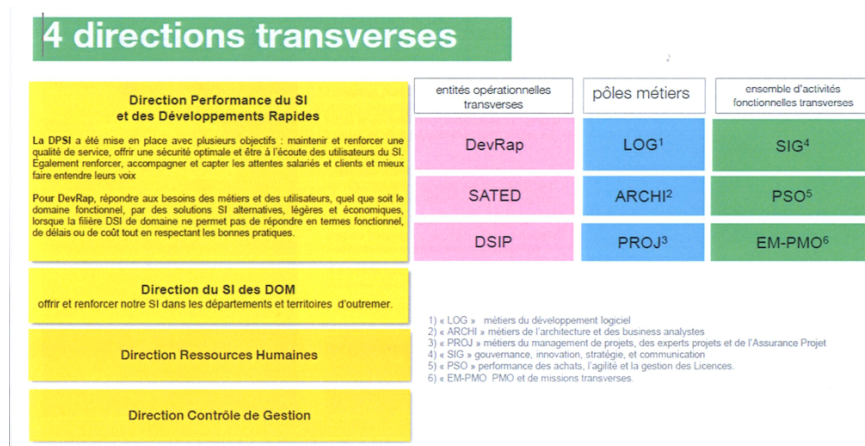


FIGURE 1.1 – Organization

1.1.3 Centre de Compétences Informatique Cognitive

Within DSI, the Cognitive Informatics Competence Centre (CCIC) works on the exploration of the field of artificial intelligence. Under the responsibility of Mr. Ionel TOTHEZAN, CCIC is responsible for ensuring the coherence and synergy of initiatives in the cognitive field at DSI and thus ensures exchanges with the business divisions within Orange France and other entities in the group interested in the field.

The Cognitive Informatics Competence Centre carries out the following activities:

- Participate in the analysis of solutions envisaged for business projects in the cognitive domain
- Ensure technology watch by studying the latest advancements in research and development in the field of artificial intelligence
- Make prototypes and models

2. Figures reproduced from *Le Livret d'Accueil - DSI Arcueil et Région Parisienne*.

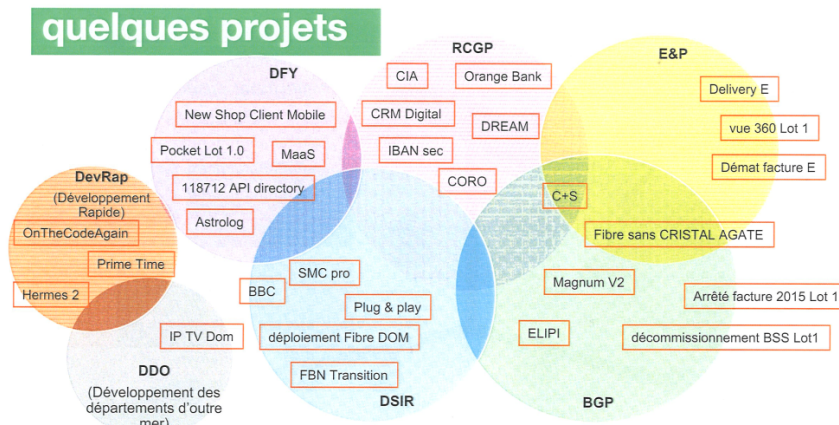


FIGURE 1.2 – Projects

- Participate/Pilot in anticipation studies
- Contribute to the definition of the domain strategy
- Represent DSI in Horizon 2020 projects in the cognitive group
- Lead and federate experts within DSI

To do this, CCIC implements PoC (Proof of Concept) on internal use cases in Orange most of the time, but this is not the only case, and thus demonstrates the feasibility of a project. Direct operational implementation as a result of POC is not a major consideration at the time a project is launched, and that is why the tasks performed can be perceived as between research and development. The projects are thus very varied and can go from natural language processing to robotics, via conversational agents, image processing and tasks on voice.

The group comprises 6 employees, most of them engineers. During my internship, there are about 10 interns working on different projects mentioned above.

1.2 Internship topic

1.2.1 Motivation

Recently, more and more companies use chatbots to connect people to their business and as their support services. However, people's interest in creating chatbots dates back to early 1960s. At that time, the interest in chatbots has no purpose of business, but comes from the concept of Turing test as a criterion of intelligence proposed in Alan Turing's famous article *Computing Machinery and Intelligence* [5]. For a long time, the development of chatbots remains in academic research. Even in the early time of task oriented chatbots used for commercial purpose, it didn't gain a lot of popularity due to poor quality of conversation caused by the lack of advanced technology in natural language understanding and natural language generation.

With the huge increase in computational power and the datasets obtained from internet, rule based methods in NLP are replaced by statistics-based methods which achieve better results. In recent years, using deep learning for solving NLP tasks obtains even better results and can solve more complex tasks like article summary generation. The improved results in

tasks like part-of-speech tagger, named entity recognition and classification of user utterance intents - which are backend components in a dialog system - make chatbots more powerful, hence it becomes very popular in business development.

However, the test and validation of chatbot quality remain in a static manner. The objective of my internship of 6 months at Orange France is to develop a dynamic method to test the quality of chatbots. Orange France has many conversational agent prototypes developed on the IBM platform. These prototypes evolve towards actual deployed conversational agents. So automation of the testing and validation of these chatbots will facilitate the creation of these agents. In the next section, we discuss briefly some approaches and we explain why we need a dynamic approach.

1.2.2 Issues

In this section, we discuss some possible methods to test chatbots. It is not a complete list and the discussion is not in depth for each method. Our goal is to explain the advantages and the necessity of an automatic chatbot test system for which I developed a prototype during my internship. In the rest of this section and this chapter, we use some terminologies about chatbots, which are introduced in Section 2.1.

The most straightforward way is to test chatbots manually. However it is time consuming, tedious and expensive. Furthermore, such tests by humans have a bias toward users' intentions. For example, suppose an utterance \mathbf{u} is defined in a chatbot as an example of intent \mathbf{A} but it is more close to another intent \mathbf{B} in the eyes of human beings. When \mathbf{u} is sent to the chatbot, intent \mathbf{A} is identified, and the corresponding response is sent back to human. He might think the chatbot is not working 'correctly'. However, the chatbot's NLU component 'correctly' identifies \mathbf{u} as intent \mathbf{A} as defined in the chatbot.

A simple automatic test method is to predefine user utterances in a file. However, there are some disadvantages:

- It is suitable only for question-answering type chatbots. For chatbots with follow-up questions (i.e. chatbots with second or higher level dialog nodes in their dialog trees), without specifying the sequences of utterances, the test is meaningless.
- However, predefining sequences of utterances in a file is infeasible for chatbots beyond small size. The number of possible sequences increases exponentially.
- When a chatbot is modified, it is not easy to modify the predefined file.
- Each time a new chatbot is created, we need a different file. For example, we can't use the file defined for a chatbot about bank on the chatbot about customer service.

Other types of method for testing a chatbot include static analysis of its dialog tree. For example, we can check if there is any dialog node that can never be reached, or if the dialogue could be trapped into a cycle. However, such analyses focus on the correctness of program behavior under the hypothesis that the natural language processing components work perfectly, rather than chatbots' capacity to understand user utterances and answer correctly.

1.2.3 A similar work

We searched for similar work and found only one result: Bottester presented in the article [6]. This system also tries to simulate users, but the difference between Bottester and Mirror Bot - to be introduced in Section 1.2.4 and to be developed during my internship - are:

- Bottester uses a corpus of data containing frequent questions and answers which was already collected and built. This means that it is kind of predefined sentences approach. Mirror Bot should not use predefined sentences and it must be capable to generate utterances relevant to a chatbot's goal, as well as the internal structure of individual chatbot³, in an automatic way.
- Bottester generates typos in the words of each utterance to be sure the intent classifier is able to withstand simple variations in the way a user formulates utterances. On the other hand, Mirror Bot is a complete and pipelined system with the knowledge about the chatbot to be tested, a set of response modules that generate user utterances, and a decision module which decides how to use these response modules.

1.2.4 Topic

From the discussion in Section 1.2.2 and 1.2.3, we know that a good automatic chatbot test system has at least the following characteristics:

- It can generate utterances dynamically at runtime.
- It knows what kind of utterances a chatbot is waiting at a given moment during a conversation.
- It is capable of modifying the expected utterances but keeping the original meaning.
- It can tell if a chatbot correctly identifies an utterance's intent.

From now on throughout this report, we name a such automatic chatbot test system **Mirror Bot** or **Bot Miroir**.

The Mirror Bot to be developed is thus not a simple bot that uses predefined sentences in a file as utterances to send to the chatbot to be tested. Instead, it is a system that uses the following elements to generate utterances to send to the chatbot:

- **Static information** These are elements defined in a chatbot such as:
 - Intent
 - Entity
 - Dialog tree
- **Dynamic information** These are information updated during a conversation according to responses of a chatbot, including:
 - Trace of dialogues during a conversation
 - Values of context variables that change over time
 - Other dynamic elements, for example, the detected user geographical location

3. Even for chatbots with the same goal, e.g. bank chatbots created by Orange and LCL, they have different intent-variation and entity-synonym sets, not to say dialog trees.

- **Linguistic elements** These are information external to a chatbot which are added to Mirror Bot. For example, annotations of dialog node (semantic) type can be helpful. If a node asks the user to reformulate his question, and Mirror Bot can identify such situations, it could generate an utterance with a paraphrase of the previous one.

Other than utterance generation, we also wish Mirror Bot could simulate different user behaviours. For this, we also need a decision module that can determine how to use the different modules that generate utterances. It must provide routines for different selection and filtering policies of response modules, child nodes, etc. The objective of my internship is to turn the above idea into a prototype of a functional system.

1.2.5 Mission

The mission of my internship consists of the following tasks:

- Study how IBM Watson Assistant works
- Decide useful information in chatbots for the development of Mirror Bot
- Define a generic chatbot model and implement it
- Write the code to transform a Watson Assistant chatbot into our model format
- Develop the Mirror Bot system:
 - Decide the structure of response modules
 - Implement different response modules
 - Develop a decision module that determines how to use the above response modules.
 - Add configurations to facilitate tests of different purposes
- Define some metrics, use Mirror Bot to test a chatbot and report the results

1.3 Planning

Figure 1.3 represents the initial planning of the development. Overall, the planning is respected. We spent less time on testing / evaluation planned on August. According to the analysis in Section 2.5, we identified at least 3 chatbot conversation issues which are presented at the beginning of Section 2.7. Among them, there is only one issue can be evaluated quantitatively, and the corresponding test / evaluation is carried out in 2.6.

The final version is only determined after several discussions with the internship tutor and the CCIC director. It could be improved by developing the generic chatbot model in an earlier stage.

1.4 Summary of contributions

The main contributions are:

- A generic chatbot model
- A Mirror Bot system that:

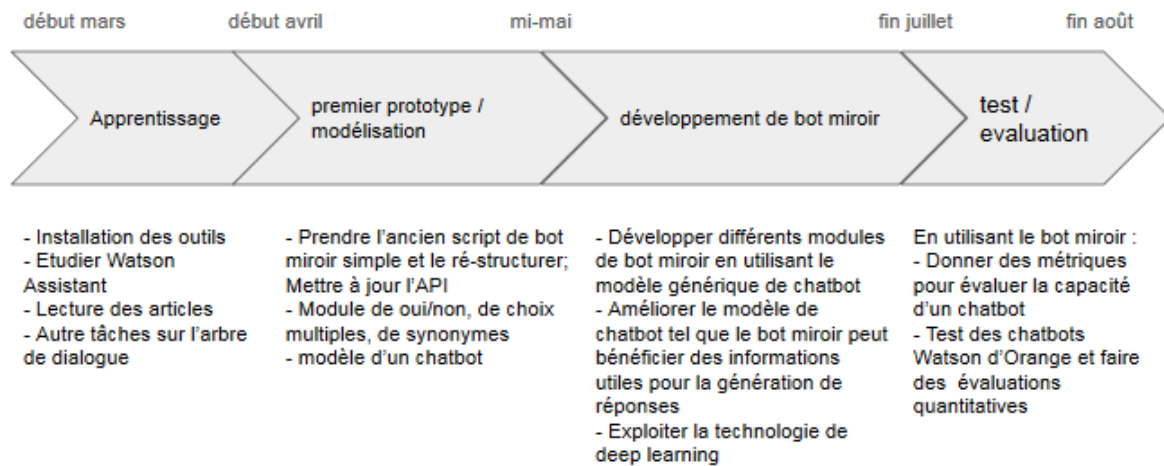


FIGURE 1.3 – Planning

- has different response modules
- has a decision module that provides routines for different selection and filtering policies and is used for determining how the different response modules are used
- has different configurations for performing tests with different purposes.

For a complete list of contributions with more details, see Section 3.1.

1.5 Organization of report

This report is organized as follows: In Chapter 1, the project motivation, the issues in this field, the topic of internship, the mission and a brief summary of contributions are given. In Chapter 2, we present the Mirror Bot system and we go deeper into each component of this system. We also give some examples of conversation produced by the system and present an analysis. Chapter 3 concludes this report with highlights of my contribution and the returns of my internship.

Chapitre 2

Main work

2.1 Study IBM Watson Assistant

The first task during my internship is to study how IBM Watson Assistant chatbots works. Although the first step in my internship project is to create a generic chatbot model, I still need to understand Watson Assistant due to the following reasons:

- In order to develop a generic chatbot model, I need to learn how chatbots work in general. Watson Assistant is a well-known and well-developed framework, so it is a good to learn from it.
- Some entities, including the CCIC group, at Orange France use Watson Assistant as the framework to develop chatbots. I need the knowledge on Watson Assistant in order to transform a Watson Assistant chatbot into an object of the *Chatbot* class defined in our generic model.

Our goal is not to present the details of Watson Assistant, but only the notions that are important in order to understand the generic chatbot model and how the Mirror Bot system can interact with a chatbot. Therefore, we give a concise introduction to Watson Assistant.

IBM Watson Assistant

There are four basic elements in Watson Assistant which can be found on other chatbot frameworks too¹:

- **Intent.** Intents are purposes or goals expressed in a customer's input, such as finding the best nearest Chinese restaurant, processing a bill payment or reserving a flight ticket. By recognizing the intent expressed in a customer's input, the Watson Assistant service can choose the correct dialog flow for responding to it.
- **Entity.** Entities represent a class of object or a data type that is relevant to a user's purpose. By recognizing the entities that are mentioned in the user's input, the Watson Assistant service can choose the specific actions to take to fulfill an intent. For example, a *reservation* intent indicates a user wants to reserve something, and sentences like *Siri, book a flight from New York to Paris.* or *Alexa, find a French restaurant for two persons*

1. Some content from this section is extracted from the official tutorial <https://console.bluemix.net/docs/services/conversation/getting-started.html#gettingstarted> with/without modification.

this Friday evening contain entities like *flight* and *restaurant* respectively that specify ‘what’ the users want to reserve.

- **Context.** The dialog is stateless, meaning that it does not retain information from one interaction with the user to the next. It is the responsibility of the application developer to maintain any continuing information that the application needs. The application must look for, and store the context object in the message API response, and pass it in the context object with the next /message API request that is made as part of the conversation flow. The simplest way to retain the information is to store the entire context object in memory in the client application - a web browser, for example. As an application becomes more complex, or if it needs to pass and store personal information, then you can store and retrieve the information from a database. The application can pass information to the dialog, and the dialog can update this information and pass it back to the application, or to a subsequent node. The dialog does so by using context variables.
- **Dialog Tree.**² The dialog uses the intents that are identified in the user’s input, plus the context from the application, to interact with the user and ultimately provide a useful response. The dialog is represented in Watson Assistant as a dialog tree. It has dialog nodes, which contain, at a minimum,
 - **Condition.** Conditions specify the information that must be present in the user input for this node in the dialog to be triggered. The information is typically a specific intent. It might also be an entity type, an entity value, or a context variable value.
 - **Response.** The utterance that the service uses to respond to the user. The response can also be configured to show an image or a list of options, or to trigger programmatic actions.

We can think of the node as having an if/then construction: if the condition is true, then return the response. Dialog nodes are connected via parent-child relation as shown in Figure 2.1. But there is also jump-to relation.

Dialog Flow. During runtime, the created dialog tree is processed by the service from the first node in the tree to the last. As it travels down the tree, if the service finds a condition that is met, it triggers that node. It then moves along the triggered node to check the user input against any child node conditions. As it checks the child nodes it moves again from the first child node to the last. The services continues to work its way through the dialog tree from first to last node, along each triggered node, then from first to last child node, and along each triggered child node until it reaches the last node in the branch it is following. When the service reaches the end of a branch, or cannot find a condition that evaluates to true from the current set of child nodes it is evaluating, it jumps back out to the base of the tree. And once again, the service processes the root nodes from first to the last. If none of the conditions evaluates to true, then the response from the last node in the tree, which typically has a special *anything_else* condition that always evaluates to true, is returned. See Figure 2.2 for a simple example of dialog flow.

2. It’s actually a graph rather than a tree. However, we adopt the terminology used by IBM Watson documentation.

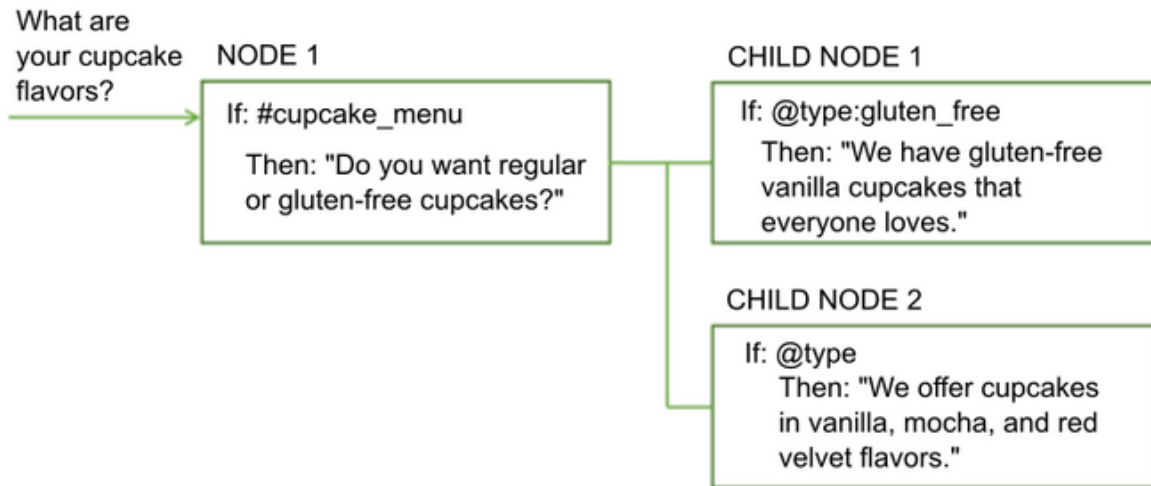


FIGURE 2.1 – Dialog Nodes

2.2 Design

2.2.1 Generic chatbot model

After studying how Watson Assistant works, the next step is to think about what elements are essential for a chatbot model from the perspective of a Mirror Bot system and to create a generic chatbot model. The ultimate goal of having such a generic model is to make the development of the Mirror Bot system independent of particular chatbot frameworks. In fact, we discuss only a few elements of chatbot in the previous section, and they are the elements that we use in our generic model. However, before the presentation of our model, it is necessary to point out some essential differences between our model and a complete chatbot model.

What components are not included in our generic model?

- **Natural Language Understanding.** For a real chatbot framework, it is necessary to have a natural language understanding component in order to do *intent classification* and *named entity recognition* which are essential for a chatbot to understand user intention and to detect specific entities. However, our model is only used to serve as an interface between Mirror Bot system and different chatbot frameworks like Watson Assistant or Amazon Alexa. It doesn't have to understand utterances sent from Mirror Bot. What our model does when it receives Mirror Bot's utterance is to send this utterance to the actual chatbot which lives in Watson Assistant or other chatbot services. The actual chatbot uses its natural language understanding component to understand that utterance.
- **Natural Language Generation.** After the actual chatbot processes an utterance and generates a response, the response is sent to our model. Our model post processes

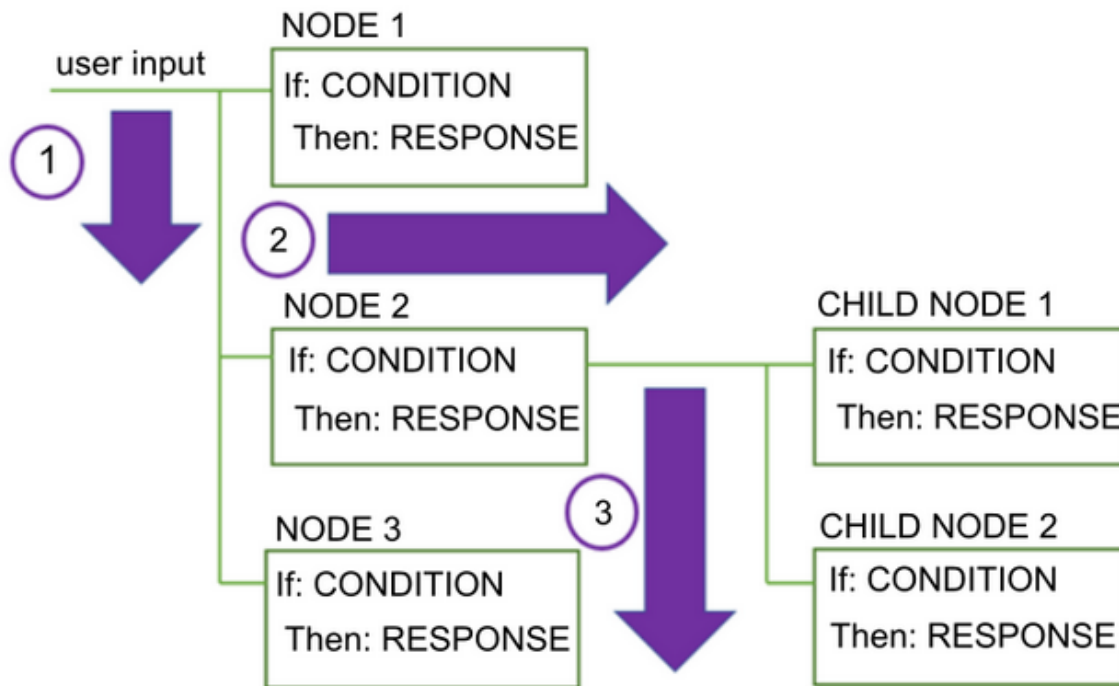


FIGURE 2.2 – Dialog Flow

the response then send it to Mirror Bot. Therefore, the model doesn't need a natural language generation component either.

- **Dialog manager.** Although dialog tree is in our model, the model doesn't have a dialog manager that controls the dialog flow by respecting the dialog tree. Again, it is the dialog manager of the actual chatbot that controls the dialog flow and then sends the next step to our model.

From the above discussion, we see that our model do not contain important components of a traditional chatbot system. It only gathers static information of a chatbot (like intents, entities and dialog tree) and it possesses some methods to pass/convert dynamic information between Mirror Bot system and the actual chatbot to be tested. However, from the perspective of Mirror Bot, this is exactly what we expect. The goal of Mirror Bot is to test an actual chatbot, and this means it measures the quality of the NLU component and the dialog tree design of the actual chatbot. Our chatbot model as an interface is just to make Mirror Bot system capable of handling chatbots built by different frameworks.

Model

Figure 2.3 gives an overview of our model. The following explains the role and the functionality of each component.

- **Chatbot.** This is the main class of our chatbot model. It uses objects of other classes as its components. It encompasses static information like intents, entities and dialog tree

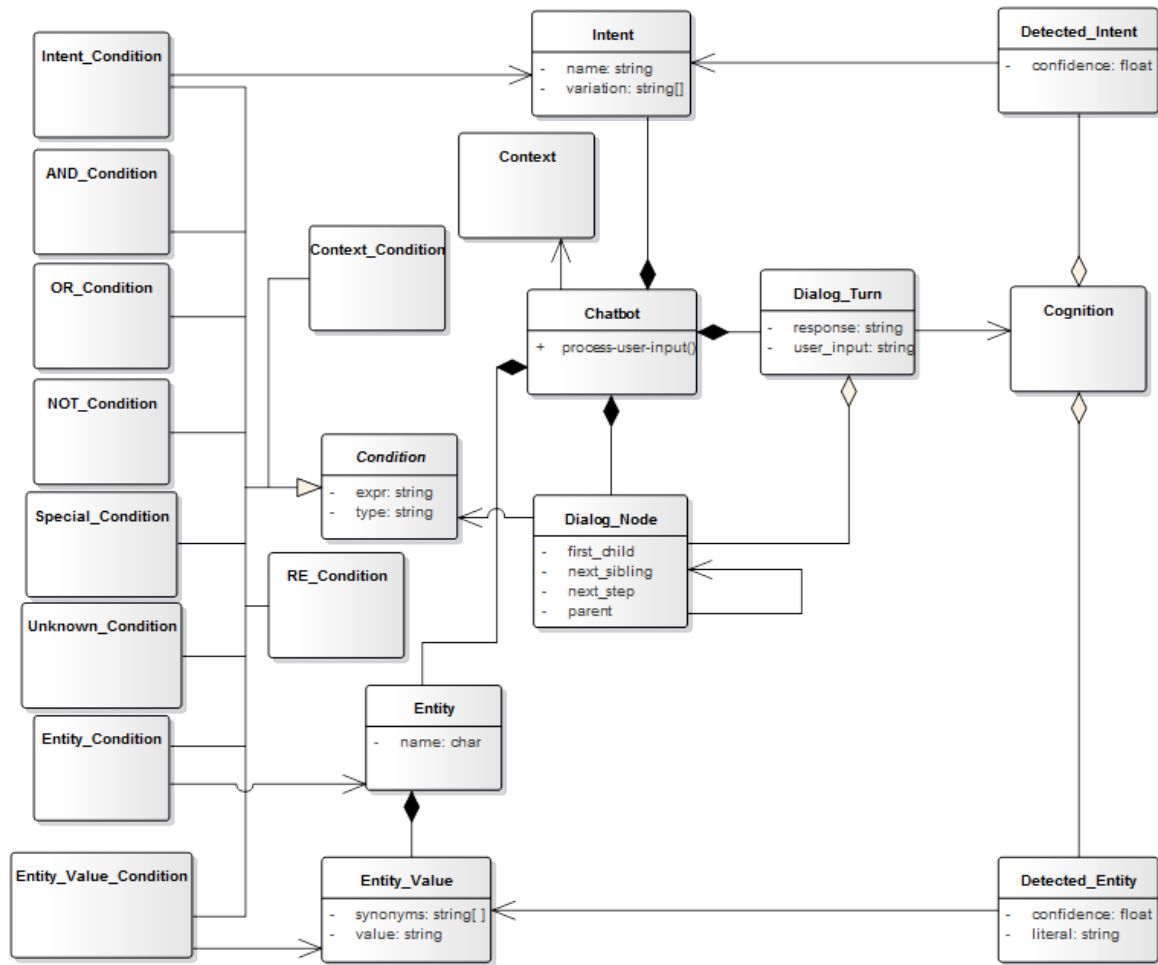


FIGURE 2.3 – Generic chatbot model

of chatbot, as well as dynamic information evolving during a conversation: context and dialog turns. It contains an important method **process_user_input** which abstracts an hidden method defined in a transformer (see Section Transformer below) that is responsible to: send Mirror Bot’s utterances to the actual chatbot, receive text responses and other information, process/convert them into the format defined in our model, and send them to Mirror Bot.

- **Intent.** Each intent has a name as well as a list of examples (also called variations) that share the same user intention. For a real chatbot, these variations are used to train the intent classifier which is an important part of the NLU component of the chatbot. However in our model, these variations are only used to help Mirror Bot to generate utterances.
- **Entity.** Each entity has a name and contains a list of entity values. For example, the *appliances* entity may have the following entity values: television, refrigerator, air conditioner and washing machine, etc.
- **Entity_Value.** Each entity value has a value (which is a string) as well as a list of

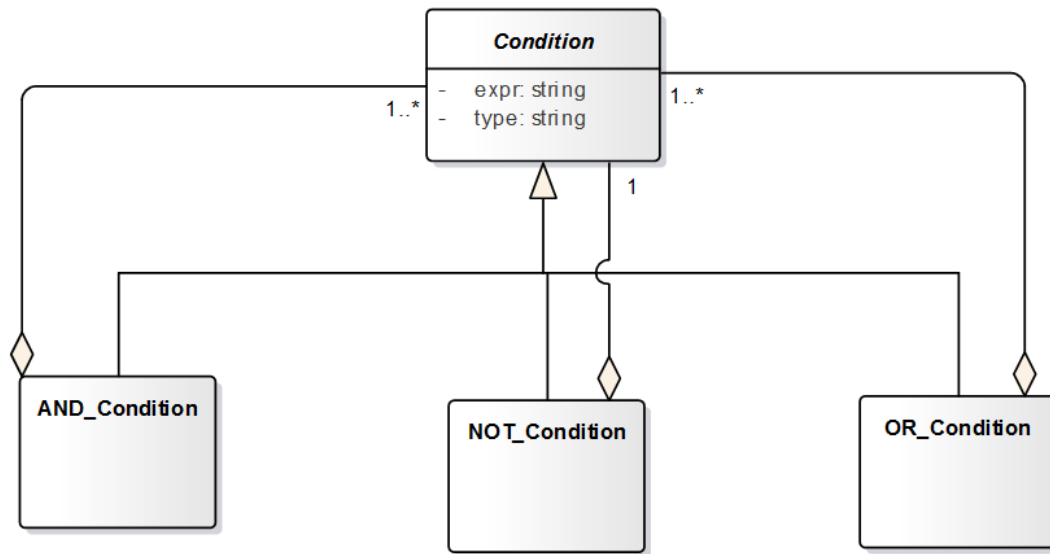


FIGURE 2.4 – Condition

synonyms. For example, the entity value *refrigerator* may contain synonyms like freezer and fridge. For convenience, in each entity value, we add a reference to the entity to which it belongs .

- **Condition.** Conditions are used in dialog nodes. A dialog node can be activated only if the condition defined in this node is met. The class *Condition* is abstract, and it is generalized by several types of condition. Important condition types include *Intent_Condition*, *Entity_Condition* and *Entity_Value_Condition*. Obviously, each of these contains an intent, entity or entity value object respectively. The conditions can be combined by logic operators like AND, OR or NOT, and the corresponding classes allow an elegant representation of model as well as a clean method to visit condition objects.
- **Dialog_Node.** A chatbot contains a list of dialog nodes of its dialog tree. The tree structure is represented by relationships between nodes which are characterized by the following attributes in a node:
 - parent node
 - first child node
 - next sibling node
 - next step node

The next step node attribute allows a node to jump to another node.

- **Context.** The *Context* class is an inherited class of the *dictionary* class, and each context variable is characterized by its variable name as a key of the dictionary and its value is the corresponding value for that key in the dictionary.

- **Dialog_Turn.** A chatbot contains, for each conversation, a trace (list) of dialog turns. A dialog turn contains the user utterance, the response from the actual chatbot, a list of visited nodes during this turn and an object of the class *Cognition*.
- **Cognition.** An object of the class *Cognition* represents the ensemble of understanding of an utterance. At this moment, it contains a list of objects of type *Detected_Intent* and a list of objects of type *Detected_Entity*.
- **Detected_Intent.** This class encapsulates intents that are detected by an actual chatbot in a dialog turn in a conversation. It contains the following two attributes:
 - intent: Intent, the detected intent
 - confidence: Float, the confidence that an user utterance corresponds to the detected intent. It must be between 0 and 1.
- **Detected_Entity.** This class encapsulates entities that are detected by an actual chatbot in a dialog turn during a conversation. It contains the following three attributes:
 - entity_value: Entity_Value, the detected entity value.
 - confidence: Float, the confidence that an user utterance contains the detected entity value. It must be between 0 and 1.
 - literal: String, the substring in an user utterance that corresponds to the detected entity value.

Transformer

The above generic chatbot model interacts with the Mirror Bot system, and the ultimate goal of having such a generic model is to make the development of the Mirror Bot system independent of particular chatbot frameworks. In this way, we develop the system once and it can be used to test chatbots built by different frameworks. However, when we want to test a chatbot built by a particular framework, we must convert it to the format defined in our model. Since in the Centre de Compétences Informatique Cognitive group, we use Watson Assistant as our chatbot development tool, we need a transformer that converts a Watson Assistant chatbot into an object of type *Chatbot* in our model. The following list of main methods gives an overview of what this transformer does:

- `watson_assistant_to_chatbot`: Using the methods below to create an Chatbot object from a Watson Assistant chatbot.
- `static_data_from_workspace`: Using the following methods to transform static information of a Watson Assistant chatbot into the format defined in our model:
 - `watson_intents_to_intents`: Deal with intents.
 - `watson_entities_to_entities`: Deal with entity and entity values.
 - `watson_dialog_nodes_to_nodes`: Deal with dialog nodes and the dialog tree.
- `watson_context_to_context`: Remove some unused information from the context dictionary obtained from the actual chatbot during conversation.
- `process_user_input`: After each dialog turn, we obtain some information from the actual chatbot: context, visited nodes, detected intents/entities and chatbot responses. This method processes and converts these **dynamic** information into our model format.

2.2.2 Mirror Bot system

After developing the generic chatbot model, the Mirror Bot system based on this model is constructed. In this section, we describe the global system architecture and some details in some components. The relationships between the major components: the class *Bot_Miroir*, the module *decision* and the package *response*, as well as the dependency on the module *basic* of other response modules are demonstrated in Figure 2.5

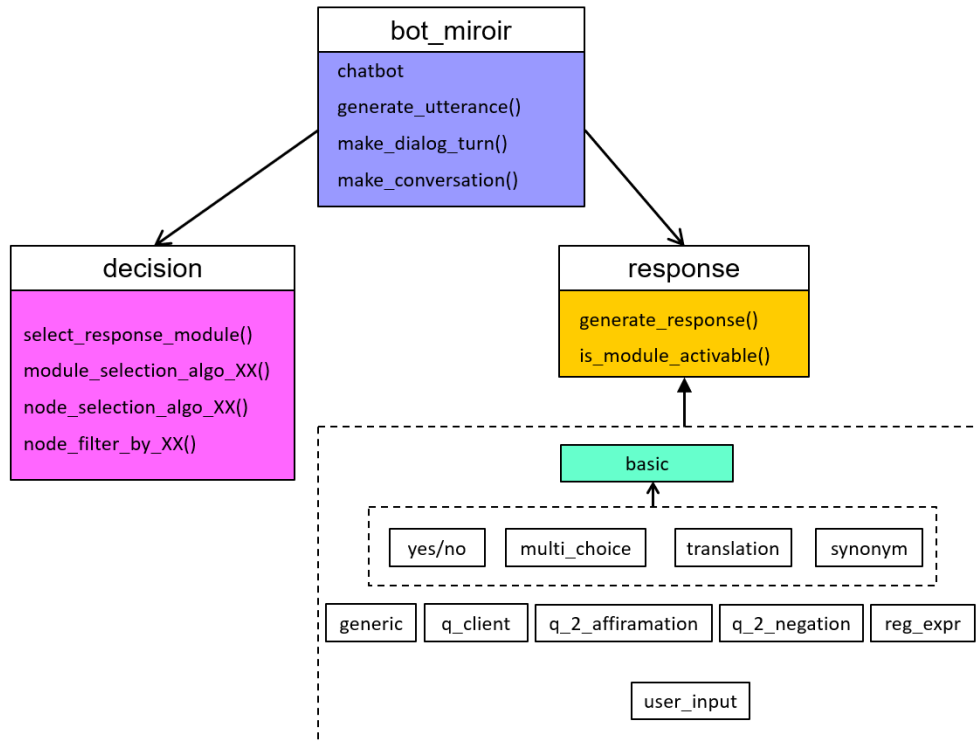


FIGURE 2.5 – Architecture of the Mirror Bot system

Class: Bot_Miroir

This part of Mirror Bot system plays the role of a user talking with a chatbot. It depends on the *decision* module to make some decisions and on the *response* modules to generate utterances to be sent to the chatbot. The following list gives an overview of important elements in the class *Bot_Miroir*:

- An attribute *chatbot* which holds an object of the class *Chatbot* which is obtained from the transformation of an actual chatbot to be tested into our model format.
- Several attributes which hold history (as lists) of several kind of information. For example, utterances sent to the chatbot, dialog nodes/intents/entities that are selected for generating utterances, etc.
- A method **generate_utterance()** which uses *select_response_module()* in the *decision* module to select a response module and uses *generate_response()* in that response module to generate a utterance.

- Methods **make_dialog_turn()** and **make_conversation()** that organize dialog turns in a conversation and other work like printing results, or logging information for making decisions in later dialog turns and for analyzing after a conversation is done.

Package: modules

This package groups the modules of our pipelined system. At this moment, we have a module that make decisions and different modules for generating utterances.

- **Package: response** This package groups the modules that generate utterances. These are called *response modules*. Each response module must contain the following two methods:
 - **generate_response()**: This method generates an utterance for Mirror Bot using the information it contains (so the information about the chatbot is included).
 - **is_module_activable()**: This method determines if a response module is suitable for being used at the current situation in a conversation. For example, the *yes_or_no* module can be activated only if the chatbot ask a question and it expects intents GN_OUI or GN_NON in the next user utterance. This method is called by the method **select_response_module()** defined in the *decision* module (introduced after the response modules). Only those activable modules can be selected.

At this moment, we have the following response modules³:

- **basic**: Although the *basic* module is an response module just as the following modules, it has a special role. It implements the fundamental idea demonstrated in Figure 2.6. It's the original idea of the project, and it's the basis of many of the other response modules which use the *basic* module to obtain an utterance then modify it.
- **yes_or_no**: This module identifies if the dialog is paused at a Yes/No question. In this case, it gives a variation in an intent that represents Yes or No intention.
- **multiple_choice**: This module identifies if the dialog is paused at a multiple choice situation. In this case, it gives an entity/entity value (or its synonyms) in these choices.
- **translation**: This module uses double (or multiple) translations (for example, FR \rightarrow EN \rightarrow FR) to modify an utterance defined in a chatbot.
- **synonym**: This module replaces one word in an original utterance by one of its synonym.
- **generic**: This module gives generic utterances like *Aucune idée*, *Je ne sais pas*, *Tu n'as pas compris*, etc.
- **question_client**: This module gives utterances that are comment questions to a chatbot which are not related to the chatbot's goal. For example, *A quoi tu sers ?*, *Quel est ton nom ?*, *Tu parles anglais ?*, etc.

3. The modules translation, generic, question_client, question_to_affirmation and question_to_negation are developed by Nicolas MICHEL, a temporary partner on the project.

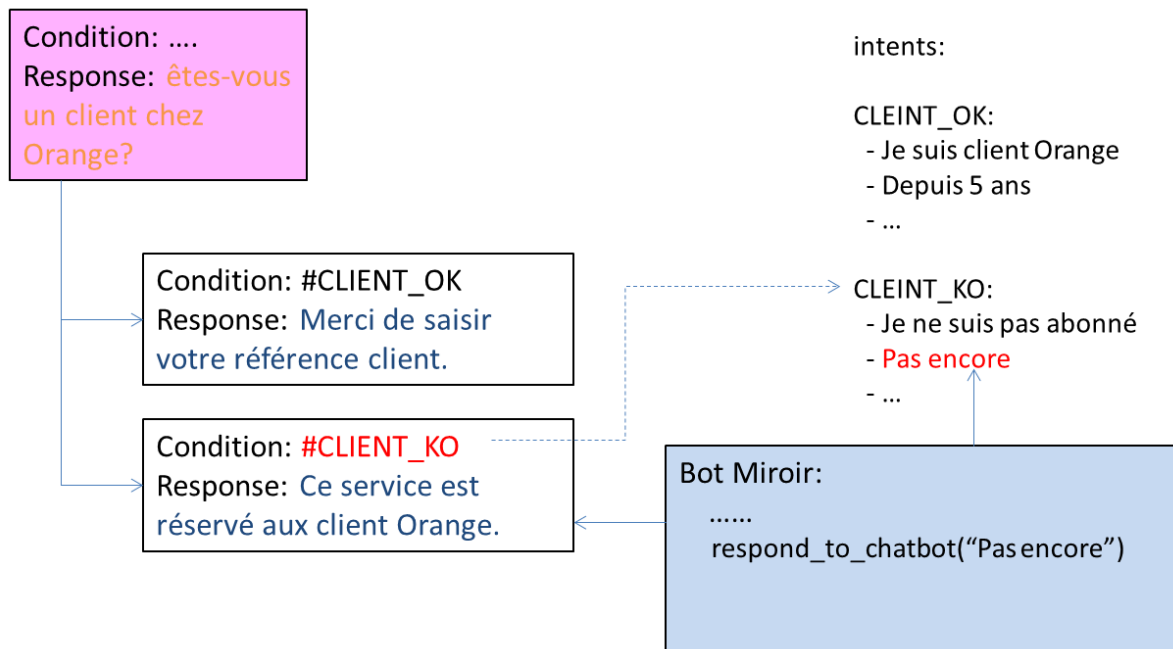


FIGURE 2.6 – The basic module

- **question_to_affirmation**: This module generates utterances by transforming a question to an affirmative answer. For example, *Est-ce que le réseau s'affiche à nouveau sur votre mobile?* to *on peut dire ça, ce est que le réseau s'affiche à nouveau sur mon mobile*.
- **question_to_negation**: This module generates utterances by transforming a question to a negative answer. *Est-ce que le réseau s'affiche à nouveau sur votre mobile?* to *certainement pas, ce n'est pas que le réseau s'affiche à nouveau sur mon mobile*.
- **regular_expression**: This module generates utterances that satisfy regular expressions defined in dialog node conditions in the chatbot. For example, the chatbot asks *Pouvez-vous me donner votre numéro de mobile Orange ou votre adresse mail Orange.fr afin que je puisse vérifier si vous bénéficiez de l'option ?*, and a child node of the current activated node has a regular expression condition

```
input.text.matches(".*[a-zA-Z0-9._-]+@[a-zA-Z0-9._-]{2,}\.[a-z]{2,4}.*")
```

which can be satisfied by a telephone number in different forms.

- **user_input**: This module is only used to manually guide the flow of dialogue to a particular node.
- **Module: decision** This module, in an informal word, plays the role of brain of Mirror Bot system. Inside this module, the following stuffs are defined:
 - A routine that selects a response module among a list of specified modules, given a list of modules to exclude as well as the module selection method corresponding to a specified policy.

- Methods used for the response modules to filter out some nodes and select a node (and a corresponding utterance) among a list of nodes (e.g. child nodes of the current node or the top level nodes if the dialog reaches a leaf in the dialog tree).
- Methods that map selection/filtering modes (specified in commandline arguments) to selection/filter methods.

The system at runtime

Figure 2.7 illustrates how Mirror Bot tests a Watson Assistant chatbot through the generic chatbot model with the support from Transformer.

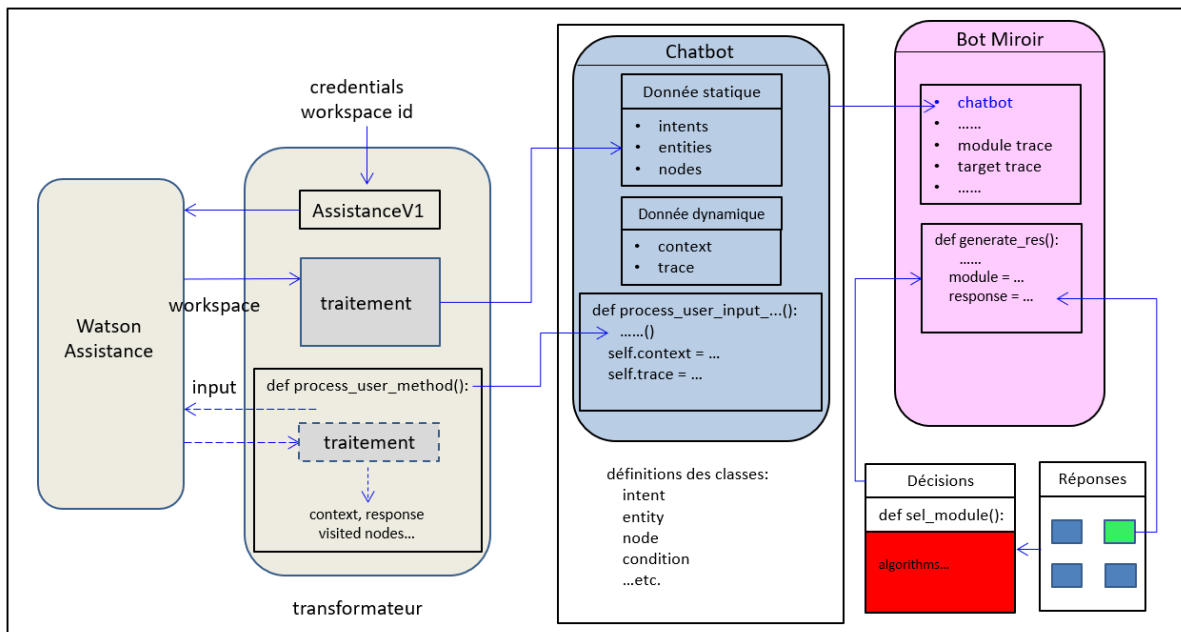


FIGURE 2.7 – Mirror Bot at runtime

2.3 Implementation

The implementation of the whole system, including the transformation of an actual chatbot into an object of class `Chatbot` is quite straightforward. In this section, we highlight some extra efforts and justify some choices of implementation. We also talk about the tools used in the development.

Transformer

Conceptually, there is no difficult task here: We have a `.json` file, we load it and read data, and we organize the information in our desired format. However, in order to build such information (like intents, entities and dialog nodes), all we have are lists of item name/information. It's inefficient to find an item in a list by iterating over it, especially when we need to refer

to particular items a lot of times during the initialization of a Chatbot object. So in an intermediate step, we build dictionaries with item names as dictionary keys and item information as dictionary values. And in a final step, we use these dictionaries to build a Chatbot object. This is done by converting the previous dictionaries into dictionaries with the same item names as dictionary keys but the dictionary values are item information in terms of objects defined in our model.

In fact, at the final step, we don't really need dictionaries to store intents, entities and dialog nodes of a Chatbot object (because the references are no longer name references but object references, we have direct access to information), but we keep this format by assuming that a name reference will be necessary at some point in a future development.

It's worth pointing out that, even with this efficiency consideration, it still takes some time to convert a chatbot (sometimes no less than 10 seconds) each time we launch a new conversation. It would take much more time without our careful implementation!⁴

package: condition

The package *condition* contains an abstract class *Condition* which is inherited by several subclasses like *Intent_Condition*, *Entity_Value_Condition*, *Intent_Condition*, logical conditions and other conditions. Although our goal is not to develop a complete condition parser, it's useful to have a minimal parser that helps to analyzer dialog node conditions for utterance generation. It reminds me an exercise in a practical work (en français, Travaux pratiques, TP) in the course *Analyse et Conception Objet* taught by Mr. [Alain SAMUEL](#). The **Composite** design patten for parsing arithmetic expression in this exercise helps me to build a simple condition parser, which is demonstrated in Figure 2.4 and on the left side in Figure 2.3.

Module system in Mirror Bot

From **Package: response**, we see that the response modules can be organized in a more elegant way by using the object-oriented approach, e.g. by defining an abstract class *Response* and create subclasses inheriting this class. However, we decide to not use this approach, but just require all the response modules must contain the methods *generate_response()* and *is_module_activable()*. Our reasons are:

- The different response modules are not to be used as classes and to create objects. It doesn't make good sense to create objects of such modules for different mirror bot objects. The methods in these modules are class methods rather than instance methods. Because of this property, plus the flexibility of defining things as *python modules*, our approach is a more proper way in this case.
- It also simplifies tasks in the development of future response modules.

Tools used in response modules

We are not diving into implementation details of each response module. Instead, we mention the tools that are used in these modules.

4. It is not that important if we generate first a lot of conversations then analyze them. However, during the development phase, we want to launch the system several times.

- **translation.** We use the module **Translator** from **googletrans** Python library which implements Google Translate API. It uses the Google Translate Ajax API to make calls to such methods as detect and translate.
- **synonym.** We use the module **wordnet** from the Python package **Natural Language Toolkit (NLTK)** which is a natural language processing library. WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. The *wordnet* module in NLTK can access WordNet and **Open Multilingual WordNet** which includes French.
- **question_to_affirmation.** We use **StanfordPOSTagger** to find part of speech tag of each token in a sentence. This helps the syntactic analysis of sentences and the conversion of a question into an affirmative statement.
- **question_to_negation.** StanfordPOSTagger is used as above.
- **regular_expression.** We use the python module **exrex**. It is a command line tool and python module that generates all - or random - matching strings to a given regular expression and more.

Other Tools used in the development

1. **Python** is the programming language for the development. We use Python 3.6.
2. **Anaconda** is used for package, dependency and environment management.
3. **PyCharm** is our integrated development environment.
4. **Jupyter Notebook** and later **Jupyter Lab** are our tools for quick experiments.
5. **git** is the version control system we use for tracking changes within one of Orange's tooling environment - **OrangeForge**, which is the internal name of **Tuleap** at Orange Labs. Based on Tuleap, OrangeForge is the collaborative portal of projects. Tuleap is a software platform for project management that enables managers and developers to utilize diverse development methodologies including Agile, traditional, or hybrid or custom processes.
6. **SmartGit** Although some tasks like *git status*, *git push*, *git pull*, etc. are done in the command line, I use SmartGit for other tasks like *stage* and *commit*, as well as navigating the version tree in git. SmartGit is a Git GUI client with support for SVN, GitHub, BitBucket pull requests and comments. SmartGit targets professional users.
7. **IBM Watson Assistant** is the framework we use for building chatbots. It is a robust platform that allows developers and non-technical users to collaborate on building conversational AI solutions. Its graphical UI, powerful NLP and familiar developer features allow the rapid creation of anything from simple chatbots to complex enterprise grade solutions for customer service and more.
8. **AssistantV1** is a python module for communicating with Watson Assistant services in python, as part of **watson_developer_cloud** - the client library to use the IBM Watson services in Python.
9. **OpenNMT** For a side project (See Section 2.4) which relates to utterance generation using deep learning, OpenNMT is used. OpenNMT is an open source (MIT) initiative for

neural machine translation and neural sequence modeling. Despite its name OpenNMT and its initial application to machine translation, it is a general-purpose **attention**-based **seq2seq** system. There is very little code that is translation specific, and it can be effective for many sequence to sequence tasks.

10. **Moses Tokenizer** is used for text processing for the task in Section 2.4. **Moses** is an implementation of the statistical (or data-driven) approach to machine translation (MT) which was the dominant approach in the field of machine translation before the neural machine translation approach emerged in recent years (2014 -).
11. **Related Tools.** OpenNMT is a project of high level sequence to sequence model training. As an developer working in industry, it's a good practice to do experiments first with easy-to-use/efficient/high performance system and check quickly if an idea works as expected. If yes, we can decide if we need an customized implementation using lower level tools. For this reason, I choose OpenNMT for experiment. However, as a learner and as a future NLP machine learning engineer, it's important to understand how things actually work. I decided to work with **Keras** and **Tensorflow** for simpler sequence to sequence tasks to gain hands-on experiences. For example, *named entity recognition*, *calculator for evaluating arithmetic expressions* using sequence to sequence architecture based on recurrent neural network, and *image caption* using CNN encoder and RNN decoder.

2.4 Deep learning - A simple approach to generate new intent variations

As mentioned before, my internship task is to develop the global architecture of the Mirror Bot system, rather than delicate engineering of each response component to generate fully human-like utterances. However, as a machine learning and deep learning enthusiast who wants to continue the career in these areas, I carried out a small project related to the Mirror Bot projet, although the result is not good enough to be integrated into the main project. In this section, I give a brief introduction of this work and show some results.

1. **Goal.** For this side project, the goal is to use deep learning to generate new variations for a given intent other than those already defined for that intent in a chatbot.
2. **Data.** Given a chatbot, we have a list of intents, and for each intent, we have a list of variations expressing that intent. We have the following potential approaches:
 - A. For each variation, use it as a source sentence, then take a random variation with the same intent as the target sentence.
 - B. For each pair of variations $(v1, v2)$ which share the same intent, use $v1$ as a source sentence and $v2$ as the target sentence.

For the chatbot I work with, it has about 80 intents with roughly 20 variations for each intent. With approach **A**, we only get 1600 samples for training data, which is a dataset too small to be useful. For approach **B**, we get about $80 \cdot 20 \cdot 20 = 32000$ samples. It is still small, however, the training/generating results are quite good for our simple task.

3. **Tools.** We use **OpenNMT** for this sequence to sequence task.

4. **Prediction.** After training, we select random variations defined in the chatbot. For each variation, we use the trained model to generate sentences. We are interested in whether it can generate new variations. We use beam search to make the model generating the 30 most probable sentences, because the most 20 probable sentences are almost the variations already defined in the chatbot for a given intent.
5. **Results.** The training is fine. For a given variation, the predicted sentences are complete sentences (i.e. no `<UNK>` token in them) and they make sense. However, the first 20 output sentences are variations defined in the chatbot. For example, the intent *AC_RES_FID*, which represents an intent of canceling something (résilier qqch) has 24 variations including *je souhaite résilier ma livebox*. The output sentences for this input sentences are given in Figure 2.8. The first 24 outputs appear in the variations of the intent *AC_RES_FID*. The last 6 outputs are new. However, not all of them

```

je veux résilier sans frais
je souhaite résilier ma livebox
je veux fermer un compte
fermeture du contrat
je veux mettre fin à ma à mon contrat
arrêt de l'abonnement internet
la résiliation du contrat
désabonnement </s>
je veux résilier ma livebox tv
je voudrais savoir le délai qu'il faut pour résilier une ligne fixe
je veux arrêter cette ligne
c'est pour couper la ligne
je souhaite interrompre mon abonnement
je veux résilier mon numéro
une résiliation de l'abonnement
je souhaite arrêter mon contrat
suspendre la ligne et résilier internet
je voudrais résilier mon contrat d'abonnement internet
résilier ma livebox
je désire supprimer l'abonnement du mobile 07 86 22 90 99
pour arrêter un abonnement
problème résiliation </s>
je demande la résiliation de la ligne téléphonique
je souhaite résilier un abonnement fibre que j'ai jamais commandé
oui résiliation de l'abonnement
je veux savoir fin à ma à mon contrat
comment arrêter mon abonnement
ma résiliation du contrat
je veux interrompre mon abonnement
je voudrais mettre le délai qu'il faut pour résilier une ligne fixe

```

FIGURE 2.8 – Generated variations

are interesting outputs. From Figure 2.9, we see that they are almost identical to some variations defined in the chatbot for intent *AC_RES_FID*. The differences are syntactic, like *la résiliation du contrat* → *ma résiliation du contrat* or *je souhaite interrompre mon abonnement* → *je veux interrompre mon abonnement*. Other outputs are a bit strange like *je veux mettre fin à ma à mon contrat* → *je veux savoir fin à ma à mon contrat*. However, we still have an interesting example! The output *comment arrêter*

mon abonnement doesn't correspond to any variation of the intent *AC_RES_FID*. Among the variations of all intents, there is a unique variation that starts with *comment arrêter*, which is the variation *Comment arrêter les pubs téléphoniques ?* of the intent *AC_ANNUAIRE* (which is about lists of telephone numbers). This shows that, although such interesting results are rare, **our generative model learned something new!** Since this is a side project, I didn't go further for this task.

```

Output:      oui résiliation de l'abonnement
Correspondence: une résiliation de l'abonnement

Output:      je veux savoir fin à ma à mon contrat
Correspondence: je veux mettre fin à ma à mon contrat

Output:      comment arrêter mon abonnement
Correspondence: Comment arrêter les pubs téléphoniques ?

Output:      ma résiliation du contrat
Correspondence: la résiliation du contrat

Output:      je veux interrompre mon abonnement
Correspondence: je souhaite interrompre mon abonnement

Output:      je voudrais mettre le délai qu'il faut pour résilier une ligne fixe
Correspondence: je voudrais savoir le délai qu'il faut pour résilier une ligne fixe

```

FIGURE 2.9 – Comparison new outputs with variations defined in chatbot

In recent years, deep learning approaches have obtained very high performance across many different NLP tasks. One task related to Mirror Bot is question generation, and the work in [1] is interesting and potentially useful for Mirror Bot. However, we decide not to pursue this approach because the modified SQuAD dataset⁵ provided in [1] is for question generation for sentences from text passages in reading comprehension, which is not suitable for generating next utterance in a conversation.

The paraphrase database (PPDB) introduced in [2] and [3] could be used for the task of generating new intent variations. This work is carried out by other interns. The result is not interesting because the paraphrases in PPDB are often obtained by changing one word in the original sentences.

2.5 Results

In this section, we give some conversation snippets with a Watson Assistant chatbot generated by the Mirror Bot system. We do not attempt to provide such snippets for each response module. Instead we select interesting examples. We discuss what Mirror Bot could tell us about a chatbot.

Figure 2.11 is one conversation when Mirror Bot uses the response module *yes_or_no*. It works well although some utterances are a bit intrigue or strange. For example, the chatbot says *Est-ce que le réseau s'affiche à nouveau ...?* and Mirror Bot targets the intent *GN_NON* and picks one of its variations *t'as pas compris..*. The intent is detected by the chatbot as

5. See [4] for the original SQuAD dataset.

expected, and it answers with *Si ça ne marche pas,* This part of dialog is OK. But later, the chatbot says *Cela fonctionne-t-il?*, Mirror Bot responds *pas besoin.* with target intent *GN_NON* also being detected by the chatbot. This time, the utterance *pas besoin.* makes the dialog strange, but the system works correctly. **Even everything works well from the perspective of Mirror Bot, sometimes it is strange for human beings. This might indicate the design of a chatbot has some issue.** In this example, the issue is that the intent *GN_NON* have many variations which are used for different context. In fact, after diving into the chatbot, the utterance *ça ira.* is also an variation of the intent *GN_NON*. This implies that we may have a conversation like Chatbot: *Cela fonctionne-t-il?* → Mirror Bot: *ça ira.* → (intent *GN_NON* detected) → Chatbot: *Je vais vous mettre en relation avec un conseiller.* This is not strange, this is completely wrong! And this possibility is confirmed in Figure 2.10 by using Watson Assistant service directly.

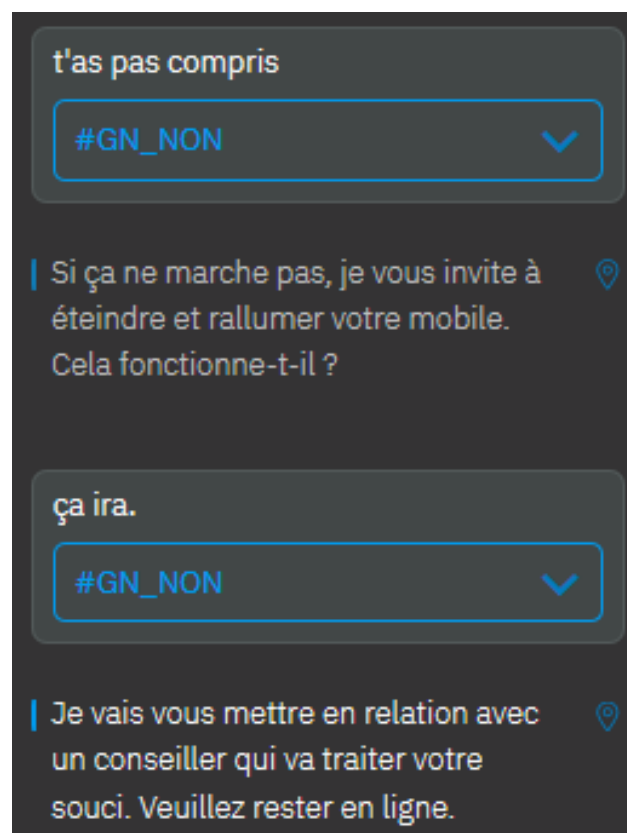


FIGURE 2.10 – Wrong dialog

There is one more problem in Figure 2.11: Chatbot: *Pouvez-vous me donner un numéro ...?* → no (usable) child node or next step → Mirror Bot: *mettre mes...(new topic)* → Chatbot: *Merci! Nous reviendrons vers vous* Again, the system still works as expected and this phenomenon reflects an issue in the chatbot: the user utterance is not treated correctly in child nodes (or the target node of a jumping) of the current node. In fact, in this particular scenario, the current node has a unique child node with the condition being *True* which can not be used by Mirror Bot system to generate utterance. So it opens a new topic, but then the chatbot always activates this child node and says *Merci*.

```

Bot Miroir:
| code PUK, svp
Chatbot:
| Nous allons voir ensemble comment débloquent votre mobile avec le code PUK. | Comme vous êtes connecté-e via l'espace client Orange, je vous ai bien
authentifié-e. | Voulez-vous le code PUK de votre carte SIM ou celui de votre carte multisim ?

Selected Response Module: bot_miroir.modules.response.basic
Utterance = SIM | Reference = SIM
detected_intent = AC_MOB_GESTION | detected_entity = SIM
Bot Miroir:
| SIM
Chatbot:
| D'accord, je vais vous donner le code PUK de votre carte SIM. | Votre code PUK est 45892511, je vais maintenant vous indiquer la manipulation à
réaliser. | Votre mobile est allumé et demande le code PUK. Composez votre code PUK à 8 chiffres, puis entrez un code PIN à 4 chiffres de votre choix.
Enfin confirmez à nouveau ce code PIN. Est-ce que le réseau s'affiche à nouveau sur votre mobile ?

Selected Response Module: bot_miroir.modules.response.yes_or_no
Utterance = t'as pas compris. | Reference = GN_NON
detected_intent = GN_NON, detected_entity = None
Bot Miroir:
| t'as pas compris.
Chatbot:
| Si ça ne marche pas, je vous invite à éteindre et rallumer votre mobile. Cela fonctionne-t-il ?

Selected Response Module: bot_miroir.modules.response.yes_or_no
Utterance = pas besoin. | Reference = GN_NON
detected_intent = GN_NON | detected_entity = None
Bot Miroir:
| pas besoin.
Chatbot:
| Je vais vous mettre en relation avec un conseiller qui va traiter votre souci. Veuillez rester en ligne. | Malheureusement, aucun conseiller n'est disponible
en ce moment. | Pouvez-vous me donner un numéro de téléphone afin que nous puissions vous recontacter ultérieurement ?

There are next steps, but none of them contains intent or entity. Selected Response Module: bot_miroir.modules.response.basic
Utterance = mettre mes trois numéros en illimités | Reference = AC_NUM_ILL
detected_intent = AC_NUM_ILL | detected_entity = None
Bot Miroir:
| mettre mes trois numéros en illimités
Chatbot:
| Merci ! Nous reviendrons vers vous dans les plus brefs délais. | Y a-t-il autre chose que je puisse faire pour vous ?

Selected Response Module: bot_miroir.modules.response.yes_or_no
Utterance = exactement. | Reference = GN_OUI
detected_intent = GN_OUI | detected_entity = oui
Bot Miroir:
| exactement.
Chatbot:
| Je vous écoute !

```

FIGURE 2.11 – response module - yes_or_no

Figure 2.12 demonstrates that the *multiple_choice* module can identify multiple choice situation and can be activated by Mirror Bot to generate utterances. Figure 2.13 is a conversation when the *question_to_negation* module is used. When Mirror Bot says *je ne veux pas le code puk de mon carte sim*, the chatbot responds *D'accord, je vais vous donner le code PUK de votre carte SIM*.. This reflects that the chatbot sometimes misunderstands user utterances. **This time, the issue is not on the design of a chatbot, but a problem of the NLU component of the chatbot framework, or the trained NLU model for a particular chatbot.**

The conversation generated by the *translation* module is shown in Figure 2.14. At the end, the utterance *je veux bien* is translated to Spanish then English and finally back to French and we get *Je m'en fiche*, thus the detected intent *GN_NON* doesn't correspond to the target intent *GN_OUI* (which means the user still has other questions to ask the chatbot). **However, in this case, there is nothing wrong on the chatbot or the NLU component. The problem is on our response modules!**

2.6 Test and Evaluation

After Mirror Bot is developed, we performed a simple quantitative test. The test focus on the trained NLU model of a Watson Assistant chatbot. More precisely, we use Bot Mirror to

```

Bot Miroir:
| code PUK, svp
Chatbot:
| Nous allons voir ensemble comment débloquent votre mobile avec le code PUK. | Comme vous êtes connecté-e via l'espace client Orange, je vous ai bien
authentifié-e. | Voulez-vous le code PUK de votre carte SIM ou celui de votre carte multisim ?

Selected Response Module: bot_miroir.modules.response.multiple_choice
Utterance = Multisim | Reference = Multisim
detected_intent = AC_MOB_GESTION | detected_entity = Multisim
Bot Miroir:
| Multisim
Chatbot:
| D'accord, je vais vous donner le code PUK de votre carte Multisim. | Votre code PUK est 45892511, je vais maintenant vous indiquer la manipulation à
réaliser. | Votre mobile est allumé et demande le code PUK. Composez votre code PUK à 8 chiffres, puis entrez un code PIN à 4 chiffres de votre choix.
Enfin confirmez à nouveau ce code PIN. Est-ce que le réseau s'affiche à nouveau sur votre mobile ?

.....

```

FIGURE 2.12 – response module - multiple_choice

```

Bot Miroir:
| code PUK, svp
Chatbot:
| Nous allons voir ensemble comment débloquent votre mobile avec le code PUK. | Comme vous êtes connecté-e via l'espace client Orange, je vous ai bien
authentifié-e. | Voulez-vous le code PUK de votre carte SIM ou celui de votre carte multisim ?

Selected Response Module: bot_miroir.modules.response.question_to_negation
Utterance = je ne veux pas le code puk de mon carte sim
detected_intent = DP_PUK | detected_entity = souhaite
Bot Miroir:
| je ne veux pas le code puk de mon carte sim
Chatbot:
| D'accord, je vais vous donner le code PUK de votre carte SIM. | Votre code PUK est 45892511, je vais maintenant vous indiquer la manipulation à
réaliser. | Votre mobile est allumé et demande le code PUK. Composez votre code PUK à 8 chiffres, puis entrez un code PIN à 4 chiffres de votre choix.
Enfin confirmez à nouveau ce code PIN. Est-ce que le réseau s'affiche à nouveau sur votre mobile ?

Selected Response Module: bot_miroir.modules.response.question_to_negation
Utterance = certainement pas, ce n'est pas que le réseau s'affiche à nouveau sur mon mobile
detected_intent = AT_MOB_RESEAU | detected_entity = nouvelle
Bot Miroir:
| certainement pas, ce n'est pas que le réseau s'affiche à nouveau sur mon mobile
Chatbot:
| Si ça ne marche pas, je vous invite à éteindre et rallumer votre mobile. Cela fonctionne-t-il ?

.....

```

FIGURE 2.13 – response module - question_to_negation

evaluate the quality of the trained intent classifier. For this, we count the dialog turns in a conversation where Mirror Bot generates an utterance by selecting an intent and using one of this intent's variations for utterance generation by modifying it. When the chatbot responds to such utterances, we obtain the information of detected intents. For each dialog turn where intents are relevant, we take the detected intent with the highest confidence as the unique detected intent and compare it with Mirror Bot's target intent at that turn. If these two intents are the same, we say that the target intent is correctly detected by the chatbot. We compute the percentage of correctly detected intents over all dialog turns where Mirror Bot has a target intent.

For the *basic* response module, we expect to have 100% correctness. However, one test shows that there are exceptions. Figure 2.15 and 2.16 shows that the utterance *mon problème est résolu* - an utterance of the intent *GN_OUI* used by Mirror Bot for utterance generation - is detected by the chatbot as having intent *GN_PB_OK*. This means that Mirror Bot can actually detect issues in a chatbot's intent classifier.

Another test uses Mirror Bot's *translation* response module, which modifies intent variations by translation. Figure 2.17 shows that the original variation *ok, c'est bon* of the intent

```

Chatbot:
| Je ne vous ai pas bien compris! Je répète alors: | Vous devez insérer votre carte SIM et vous connecter sur orange.fr > espace client > rubrique mobile.
Vous devez vous identifier avec votre numéro de mobile. Vous aurez aussi besoin de votre numéro client à 10 chiffres reçu par mail. Cela fonctionne-t-il ?

-----

Original Utterance : pas a ma connaissance.
Translated Utterance : pas que je sache.
Selected Response Module: bot_miroir.modules.response.translation
Utterance = pas que je sache. | Reference = GN_NON
detected_intent = GN_NON detected_entity = None
Bot Miroir:
| pas que je sache.
Chatbot:
| Je vous conseille alors d'éteindre et de rallumer votre mobile. Cela fonctionne-t-il maintenant?

-----

Bot Miroir:
| Pouvez-vous expliquer la méthode pour des nombres illimités?
Chatbot:
| Connectez-vous sur votre espace client > offres et options > numéro préféré. | Y a-t-il autre chose que je puisse faire pour vous ?

-----

Original Utterance : pas de probleme.
Translated Utterance : aucun problème.
Selected Response Module: bot_miroir.modules.response.translation
Utterance = aucun problème. | Reference = GN_OUI
detected_intent = GN_OUI | detected_entity = None
Bot Miroir:
| aucun problème.
Chatbot:
| Je vous écoute !

-----

Original Utterance : je veux bien.
Translated Utterance : Je m'en fiche
Selected Response Module: bot_miroir.modules.response.translation
Utterance = Je m'en fiche | Reference = GN_OUI
detected_intent = GN_NON detected_entity = None
Bot Miroir:
| Je m'en fiche
Chatbot:
| Très bien. Merci de faire confiance à Orange. Au revoir.

```

FIGURE 2.14 – response module - translation

GN_PB_OK is translated back to *Ok, ça va*, but the chatbot can't detect any intent. However, the utterance *Ok, ça va* as an response to the chatbot's question *Cela fonctionne-t-il ?* does make sense. So we know that the chatbot's intent classifier can be improved. In this test, the correctness percentage is about 50%.

```

-----
GN_OUI
GN_PB_OK
-----

Number of target intents = 14
Number of target intents correctly detected = 13
Percentage of target intents correctly detected = 0.9285714285714286

Process finished with exit code 0

```

FIGURE 2.15 – Mirror Bot tests chatbot intent classifier - Using *basic* module


```

-----

Selected Response Module: bot_miroir.modules.response.basic

Utterance = mon problème est résolu
Reference = GN_OUI

detected_intent = GN_PB_OK
detected_entity = None

Bot Miroir:

| mon problème est résolu
|
Chatbot:

| Très bien. Merci de faire confiance à Orange. Au revoir.
( Dialog node type = ['ending'] )

```

FIGURE 2.16 – Dialog turn in the test

2.7 Future work

From Section 2.5, we see that when a conversation looks strange, there are at least the following possible reasons:

- A.** The design of dialog tree has problems.
- B.** The NLU component of a chatbot development framework, or the trained NLU model, has problems.
- C.** The response modules (or other modules) has problems.

Although the internship topic is the development of Mirror Bot system for *automatic chatbot testing*, it's not specified explicitly what to test. Of course, **A** and **B** suggest that the dialog trees of particular chatbots, the NLU component of a chatbot development framework or a trained NLU model could be the targets for the test. In Section 2.6, we describe a simple test for **B**. For this test, we don't have to examine the dialogs in a conversation to identify the problems. We only need to verify if a selected intent for generating an utterance is detected by the chatbot when the utterance is sent to the chatbot. However, this assumes that we don't have problems of type **C**. Otherwise, a modified variation (for example, by the translation or synonym module) from an intent may no longer correspond to the original intent, and thus being misinterpreted by the chatbot. For **A**, it's more complicated to find an automatic yet reliable issue detection method. And by human examining the conversations, the problems coming from **B** are noisy.

The project can be extended in the following ways:

```

Original Utterance : ok , c'est bon
Translated Utterance : Ok, ça va
Selected Response Module: bot_miroir.modules.response.translation

Utterance = Ok, ça va
Reference = GN_PB_OK

detected_intent = NOT_RELEVANT
detected_entity = oui

Bot Miroir:

| Ok, ça va

Chatbot:

| Je m'excuse je ne peux pas vous répondre. Je suis encore en cours d'apprentissage !

```

FIGURE 2.17 – Mirror Bot tests chatbot intent classifier - Using *translation* module

- Improve the quality of utterances generated by response modules.
- Figure out a method that can detect and identify the sources of problems of type **A**.
- For a more complete test, it's necessary to define clearly the metrics to be used, and decide what to do if the results from some response modules can't provide information for these metrics.
- Find high quality datasets of intents and variations related to the job (e.g. bank, after-sales service, etc) of a chatbot to be tested, and using the approach introduced in Section 2.4, or other deep learning models, to generate new intent variations.
- Enrich the decision module by designing new Mirror Bot's behavior. For example, we can specify the portion of usage of the *generic* or *question_client* modules. If the portion is high, Mirror Bot behaves like a fool. If a chatbot can handle this situation, it is more capable of making chit-chat conversation.
- Exploit the usage of context to generate more logic conversations.
- Enrich dialog node annotation such that the situation of reformulation can be handled.

There are far more many interesting things that can be done!

Chapitre 3

Conclusion

3.1 Contribution

During my internship, the following work was realized:

- Solve a performance issue in calculating paths from one dialog node to another. The computation is reduced from 1-2 days to minutes. Improve loading speed of a webpage that uses javascript to present chatbot dialog trees and paths to a particular node from another node.
- Define and implement a generic chatbot model from the perspective of automatic chatbot testing.
- Write a well-organized python script (with performance consideration) that transforms Watson Assistant chatbots to objects of class *Chatbot* in our model.
- Design the architecture of the pipelined Mirror Bot system, including a plug-and-play response module system.
- Develop the class *Bot_Miroir* that interacts with chatbots to be tested through objects of class *Chatbot*.
- Develop different response modules that generate utterances.
- Develop a *decision* module that provides selection/filtering routines served by the class *Bot_Miroir* and the response modules for determining the way in which those modules are used.
- Using deep learning to generate new utterances from a small dataset of intents/variations in a chatbot by using a data augmentation technique.
- Analyze the generated conversations, identify issues and classify the problem origins.
- Provide a simple quantitative test.

3.2 Return of internship

I gained the following invaluable experience during the internship:

- Technique aspect:

- Orange France provides me a great chance to use **IBM Watson** services. This helps me to build practical knowledge about how conversational agents work. I also gained hands-on experience on how to use Watson Assistant service in python.
- I learned new tools like **WordNet**, **Jupyter Lab**, **SmartGit**, **OpenNMT** and more.
- The practical experience of management of project in a large company through **git**.
- Reinforce my experience on recurrent neural network and some deep learning frameworks like **Tensorflow** and **Keras**.
- A few experience in **javascript**.
- Company/Team aspect:
 - The discussion with other permanent staffs in CCIC group about the Mirror Bot project brings me new understanding of development from the perspective of the team and the team director. For example, my internship projet should focus on the global architecture rather than on individual (response) modules. The insights from the team director - the development of Mirror Bot should depend on a generic chatbot model instead of having different mirror bot systems - brings me to a higher level as a developer.
 - Explained by a technology specialist in the group, I understand better some policies of the CCIC team and of DTISI/DSI. With another internee as a temporary partner on the project, we didn't understand the policy of choosing version control environment within DTISI/DSI, and we just chose one because we couldn't access the other one until our accounts are added to the member list of a project on OrangeForge. Although the policies may vary for different companies, this is my first professional experience on administrative organization within a large company.
 - During the temporary cooperation with another internee, I gained teamwork experience within a company. In particular, this is my first experience on management of project via git system with other people. During an interview for a CDI position, I was asked if I have the skills on git, and I explained my experience during the internship. I believe this is one reason, among others, my application is selected for that NLP machine learning enginner position!

3.3 School, Internship, MOOC and Career

At Polytech Marseille, my major is *Système informatiques critiques et application* where we studied security, embedded systems and real-time critical systems. I then turned my interests toward machine learning and deep learning, and I did a 3 months internship using simple machine learning techniques. Later, I was attracted by modern NLP tasks using deep learning and was lucky to be selected for this internship at the CCIC group at Orange France. Despite not using deep learning directly, the development of the Mirror Bot system is fun and I learned things both from people in the group and from the project itself, in particular knowledge about chatbots. Together with online courses on machine learning, deep learning and NLP, I am recruited as a NLP machine learning engineer by a startup working on

recommendation system on mobile platform. My knowledge and experiences gained from Polytech Marseille, the internships and online courses, as a whole, allow me to achieve my goal to orient toward an artificial intelligence engineer after my PhD in mathematics. Now, it's time for new challenges!

Bibliographie

- [1] Xinya DU, Junru SHAO et Claire CARDIE. “Learning to ask: Neural question generation for reading comprehension”. In : *arXiv preprint arXiv:1705.00106* (2017).
- [2] Juri GANITKEVITCH et Chris CALLISON-BURCH. “The Multilingual Paraphrase Database”. In : *The 9th edition of the Language Resources and Evaluation Conference*. Reykjavik, Iceland : European Language Resources Association, 2014. URL : <http://cis.upenn.edu/~ccb/publications/ppdb-multilingual.pdf>.
- [3] Ellie PAVLICK et al. “PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification”. In : *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Beijing, China : Association for Computational Linguistics, 2015, p. 425-430. URL : <http://www.aclweb.org/anthology/P15-2070>.
- [4] Pranav RAJPURKAR et al. “SQuAD: 100, 000+ Questions for Machine Comprehension of Text”. In : *CoRR* abs/1606.05250 (2016). arXiv : 1606.05250. URL : <http://arxiv.org/abs/1606.05250>.
- [5] Alan M TURING. “Computing machinery and intelligence”. In : *Parsing the Turing Test*. Springer, 2009, p. 23-65.
- [6] Marisa VASCONCELOS et al. “Bottester: Testing Conversational Systems with Simulated Users”. In : *Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems*. ACM. 2017, p. 73.