Marie Sklodowska Curie, Research and Innovation Staff Exchange (RISE)

IdeNtity verifiCatiOn with privacy-preservinG credeNtIals for anonymous access To Online services

WP5 – Advanced User Interface / User Experience (UI/UX) Artificial Intelligence (AI)-based assistant
Deliverable D5.1 “Specification and initial design of the Advanced User Experience / User Interface (UI/UX) Artificial Intelligence (AI)-based assistant pipeline”

Editor(s):	UPRC
Author(s):	Kostantinos Papadamou (ESR, CUT - secondee), Markos Charalambous (CUT - outside secondment), Evangelos Kotsifakos (ER, LST - outside secondment), George Kalatzantonakis (ESR, LST secondee), Christos Xenakis (ER, UPRC - outside secondment), Carlos Segura (TID – outside secondment), Anastasia Tsiota (ESR, UPRC - secondee), Angeliki Panou (ESR, UPRC - outside secondment), Nikolaos Episkopos (TS, FOGUS – secondee)
Dissemination Level:	Public
Nature:	Report
Version:	2.0

Project Profile	
Contract Number	824015
Acronym	INCOGNITO
Title	IdeNtity verifiCatiOn with privacy-preservinG credeNtIals for anonymous access To Online services
Start Date	Jan 1st, 2019
Duration	48 Months

Partners

Τεχνολογικό Πανεπιστήμιο Κύπρου	TECHNOLOGIKO PANEPISTIMIO KYPROU (BEN2, CUT)	Cyprus
University of Piraeus	University of Piraeus research center (BEN1, UPRC)	Greece
certSIGN	Certsign SA (BEN3, CSGN)	Romania
Telefónica Investigación y Desarrollo	Telefonica Investigacion Y Desarrollo SA (BEN6, TID)	Spain
LS TECH	LSTech Espana SL (BEN4, LST)	Spain
FOGUS INNOVATIONS & SERVICES	FOGUS INNOVATIONS & SERVICES P.C. (BEN7, FOG)	Greece

Document History

VERSIONS

Version	Date	Author	Remarks
0.1	03/03/2020	Kostantinos Papadamou (CUT), Anastasia Tsiota (UPRC)	Executive Summary and Table of Contents
0.2	04/05/2020	Kostantinos Papadamou (CUT)	Introduction, draft sections
0.3	10/07/2020	Evangelos Kotsifakos (LST), Konstantinos Papadamou	TOC revision
0.4	31/07/2020	Markos Charalambous (CUT), Christos Xenakis, Angeliki Panou (UPRC)	Section 2 initial text
0.5	10/09/2020	George Kalatzantonakis (LST), Carlos Segura (TID), Nikolaos	Section 4 updates

		Episkopos (FOGUS), Anastasia Tsiota (UPRC)	
0.6	1/10/2020	Konstantinos Papadamou (CUT), Markos Charalambous (CUT), Evangelos Kotsifakos (LST), Anastasia Tsiota (UPRC)	Section 3 text.
0.7	15/11/2020	Konstantinos Papadamou (CUT), Markos Charalambous (CUT), Evangelos Kotsifakos (LST), Carlos Segura (TID)	Deliverable review and updates
0.8	20/12/2020	Konstantinos Papadamou (CUT), Markos Charalambous (CUT), Evangelos Kotsifakos (LST), Carlos Segura (TID)	Deliverable review and updates, complete missing sections
1	30/12/2020	Markos Charalambous (CUT), Evangelos Kotsifakos (LST), Anastasia Tsiota (UPRC)	Final version
2	22/01/2021	Vaios Bloulgouras (UPRC), Evangelos Kotsifakos (LST)	Version 2, comments from EU

Contract delivery due date: 31/12/2020 (M24)
Actual delivery date of version n.1: 31/12/2020 (M24)
Actual delivery date of version n.2: 22/01/2021 (M25)

Fellow ID	Name/Surname	Researcher category	Declaration No.	PM
12	Kostantinos Papadamou	ESR	10	3
22	Nikolaos Episkopos	TS	19	2
15	Anastasia Tsiota	ESR	12	5.42
14	George Kalatzantonakis	ESR	14	3

Executive Summary

In this first Deliverable of Work Package 5, we tackled the WP5 objectives from Task 5.1. More specifically, we present the first design of the INCOGNITO AI-based assistant, its architecture and an Natural Language Understanding pipeline, which will be utilized when users interact with the assistant and require help to be provided. We present the basic concepts of Machine learning and AI-assistants, the state-of-the-art in the area, based on the experience of our secondees and their related research. The secondees have studied the various options for implementing such assistants that act as chatbots to provide a conversational interaction with the users. The outcome of this research is described in this document, along with comparison of the various implementation options. We choose to use the RASA framework as it is the most advanced and popular with a lot of community support, tailoring it to address the needs of INCOGNITO. The integration of this framework to our INCOGNITO architecture is also presented in this document. A related research has being carried out for the User Interface, whether it is on a Web or Android app. We will be based on existing implementations to build our assistant.

Through the research of the task T5.1 of the INCOGNITO project, that resulted also in this document, the involved secondees had the chance to learn a lot on the emerging technology of the AI-assistants, that is being adopted by all the big companies that need to have a natural conversational interaction with their customers for supporting them with their inquiries. This knowledge will give the secondees the skills to develop such technologies and make them competitive in the respective market. The outcomes of this research will be presented in future presentations in events and workshop that will be organized by the project and we will put efforts towards related scientific publications as soon as this research will lead to experimental results. Related news posts and articles will also be presented through the dissemination channels of the project.

Table of Contents

Table of Figures

1 Introduction

The purpose of this deliverable D5.1 "Specification and initial design of the Advanced User Experience/User Interface (UI/UX) Artificial Intelligence (AI)-based assistant pipeline" is to provide the first stage design of the architectural components, the machine learning pipeline and their respective modules within the INCOGNITO platform that will support the functionalities that the AI-based assistant running on the Identity Consolidator web and mobile interfaces will offer to the INCOGNITO users.

The developed components and user-interfaces aim to offer to the users a first-of-its kind user-friendly AI-based Assistant that will be responsible to guide and inform the end-users about aspects of their identity and any possible actions that they should take in order to securely manage their identity while preserving their privacy. Among others, the AI-based assistant will be able to inform the users about the following aspects of privacy and identity management: a) users will be informed about the minimum identity attributes required to be revealed in order to access a specific Service Provider and subsequently get access to its resources; b) users will be informed and request for the de-anonymization risks as a result of revealing specific identity attributes to a Service Provider; c) users will be able to see the result of stop sharing specific identity attributes with identity attributes; d) users will be able to manage, delete, and/or create UMA policies associated with specific identity attributes that need to shared or are already shared with specific Service Providers; and e) using the AI-based assistant users should be able to know on-demand what Service Providers can be accessed when specific identity attributes are shared by default.

In this deliverable we provide a detailed description of the design and implementation of the natural language understanding pipeline that has been implemented leveraging state-of-the-art Natural Language Processing (NLP) techniques, as well as the RasaHQ[1] framework (Inc, 2020), which is an open-source machine learning framework that provides tools for developers to build, improve and deploy text- and voice-based chatbots and assistants. RasaHQ will be leveraged and we will built on top of it in order to implement a back-end component running on the Identity Consolidator of the INCOGNITO platform that is able to understand, handle, and respond accordingly to all the identity- and privacy-related management queries that the INCOGNITO users send in textual format using the developed desktop and mobile User Interfaces (UI) that will be developed as part of the user interface of the web application of the Identity Consolidator and the INCOGNITO mobile application.

The rest of the document includes an analysis of the state-of-the-art in machine learning and chatbots, the *dialogue systems* that allows humans interact with the computer in a natural conversation. Basic background about Natural Language Understanding and processing is presented, and we also present the description of the specification and design of the AI-based assistant as well as the state-of-the-art solutions and technologies that we found and analyzed in order to decide which ones will be used for the development of the AI-based assistant of the INCOGNITO platform.

[1] https://github.com/RasaHQ

2 Machine learning and Chatbots

Recent advances in Machine Learning and Natural Language Understanding are making possible the implementation of human-machine interaction systems that are driven by means of natural language conversations. These systems, commonly referred to as chatbots, have been recently gaining popularity in different mobile and online platforms. Their main objective is to serve as "virtual assistants" to support informational and transactional user requirements in a wide variety of domains and sectors.

Natural Language Processing (NLP) is a subfield of artificial intelligence that allows the machine to understand natural language processing. Moreover, language modelling is a method of predicting the next word in a text given the previous statements.

Several studies in this field present techniques that enhance the NLP. The table below shows the scientific publications in the recent years with their significant findings.

Author	Year	Ref	Significant Findings
D.E. Rumelhart	1986	[1]	Back-propagation
J.Elman	1990	[2]	Vanilla Recurrent Neural Networks (RNNs)
Kneser and Ney	1995	[3]	M-gram language modeling
R. Caruana	1997	[4]	Multitask Learning
Y.Bengio et al.	2003	[5]	Neural Probabilistic Language Model
Mikolov et al.	2010	[6]	Recurrent Neural Networks (RNNs)
R.Collobert et al.	2011	[7]	Neural network architecture and learning algorithm for NLP
Milikov et al.	2011	[8]	Combination of advanced language modeling techniques
Mikolov et al.	2013	[9]	Bag-of-Words model and Continuous Skip-gram model
Graves et al.	2013	[10]	Long Short-Term Memory networks (LSTMs)
I.Sutskever et al.	2014	[11]	Sequence to sequence learning with neural networks
M. Luong et al.	2015	[12]	Global and local approaches in attention mechanism
A.Kannan et al.	2016	[13]	Smart Reply

Hideya Mino et al.	2017	[14]	Neural Machine Translation (NMT) with a key-value attention mechanism on the source-side encoder
A.Kuncoro et al.	2018	[15]	Advantages of Modeling Structure from LSTMs
T. Blevins	2018	[16]	Deep RNNs Encode Soft Hierarchical Syntax
X.Qui et al.	2020	[17]	Survey of pre-trained models for NLP

Table 1: Overview of NLP related studies

Many of the techniques presented in *Table 1* were proposed in all aspects of NLP. The most significant methods used in NLP are multi-tasking, word embeddings, neural networks, sequence-to-sequence, attention, memory-based networks, pretrained language models and reinforcement learning.

The author of the paper [4] presented an algorithm and results for multitask learning in different ways by proving the method. Thus, the method can be applied to many different kinds of domains and can be used with different learning algorithms as well as real-world problems. Moreover, multitasking is the core to many papers like [7] were the authors proposed a neural network architecture and learning algorithm that can be applied to many natural language processing tasks.

With the progress of machine learning techniques in recent years, it has become possible to train more complex models on much larger data sets and outperform the simple models. For example [5], [8]. Word-embeddings has been improved by T.Mikolov et al. [9] by proposing two novel model architectures: a) Bag-of-Words model and b) Continuous Skip-gram model. The result from those models have been compared with previous models and outperformed in accuracy and computational cost.

Furthermore, NLP has improved by using neural networks and are categorized to recurrent neural networks (RNNs), convolutional neural network (CNNs) and recursive neural networks. All methods use a unique architecture and approach and can be used in NLP.

Sequence to sequence learning is another approach which can be used in NLP with neural network. It was proposed from I. Sutskever et al. [11]. This method is about training models to convert sequences from one domain (for example, sentences in Spanish) to sequences in another domain (for example, the same sentences translated to English).

Attention is a method that has lately been used to improve neural machine translation (NMT). It was introduced by M.Luong et al. [12] with two simple and effective classes of attention mechanism, first the local and global attention approach. This model using different attention architecture yields a new state of the art result in English to German translation.

Based on attention mechanisms researchers approach various studies on memory-based networks. Models differ in how they implement and leverage the memory.

Last but not least, pretrained language models has brought NLP in a new era [17]. The biggest advantage of pretrained language models are the usage of textual encoders and pre-trained tasks. Pre-trained tasks are categorized to a) Supervised learning b) Unsupervised learning and c) Self supervised learning. All those methods help learning with less data. Some examples of pretrained language models are 1) GloVe[2] and 2) FastText[3] ,etc.

[2] https://nlp.stanford.edu/projects/glove/
[3] https://fasttext.cc

2.1 Dialogue System

In the recent years the advent of the conversational agents is obvious. In our everyday life we can use conversational agents like Alexa[4], Google Assistant[5] or Siri[6] to help us to answer questions, make recommendations, and perform actions by delegating requests to a set of Internet services.
A dialogue system, or conversational agent (CA), is a computer system intended to converse with a human. Dialogue systems employed one or more of text, speech, graphics, haptics, gestures, and other modes for communication on both the input and output channel.[7]

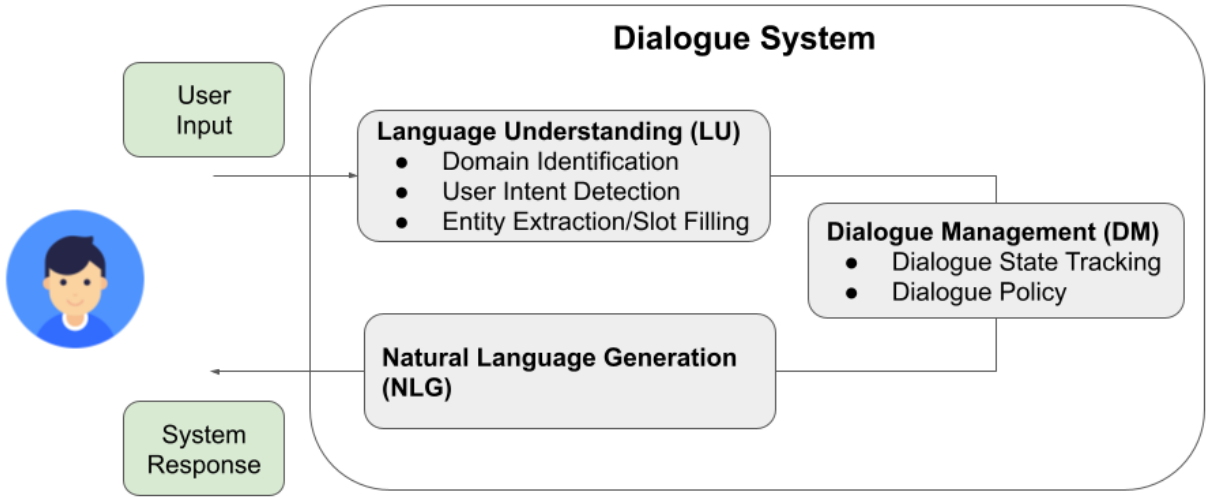

Figure 1: Overall architecture of task-oriented dialogue system

The typical architecture of a dialogue system is demonstrated in *Figure 1*.
It consists of three main components:

- Language Understanding: Is the block in charge of understanding users' input, predicting what is the users' intent and parsing an input sentence into predefined semantic slots.
- Dialogue Manager: It manages the dialogue history, keeping an internal state of the dialogue from which decides what action will it take next.
- Natural Language Generation (NLG): It translates the selected agent's action into a natural response to the user, combining information from semantic slots and external information.

2.1.1 Natural Language Understanding

Given an utterance, the main objective of the Natural Language Understanding component is to extract three main pieces of information. The first task is the domain identification: this block is

[4] https://en.wikipedia.org/wiki/Amazon_Alexa
[5] https://en.wikipedia.org/wiki/Google_Assistant
[6] https://en.wikipedia.org/wiki/Siri
[7] https://en.wikipedia.org/wiki/Dialogue_system

in charge of predicting if a user is talking about booking a restaurant, setting an alarm or controlling the TV. This block is only needed for multi-domain dialogue systems.

The second part is user intent classification, that consists of extracting the purpose or goal of the utterance, basically a classifier that categorizes it into a set of predefined intents. For both the domain and intent classification Deep Learning techniques have been applied [18], [19]. In particular, neural models based on CNN [20], LSTM [21] and more recently Transformer-based [22] neural models are applied.

Finally, the slot filling component assigns semantic labels to particular words and fillers that the user intends the system. For instance, when a user queries for a flight, key slots for origin/destination and preferred time are required for a dialog system to retrieve the appropriate information. This part is a very challenging problem for NLU and is usually defined as a sequence labelling problem, where the input are the words of the utterance and the output is the sequence of name entities and slots for the corresponding words. Although CRFs [23] and LSTM [24] work relatively well for small databases, now the trend is using BERT models to jointly estimate the intent and the slot filling [25].

2.1.2 Dialogue Manager

The Dialogue Manager is usually composed of two main components, the Dialogue State Tracker and Dialogue Policy.

The Dialogue State Tracker is needed in order to add robustness into the dialogue system. A dialogue state represents a dialogue session at any instant of the conversation and it usually contains a history of all previous turns and keeps a belief of current slots and intents. Deep-learning based approaches have achieved state of the art performance on dialogue state tracking tasks. The most common approach consists of estimating the dialogue state as a distribution over all possible slot-values [26], [27] whereas [28] proposed a Neural Belief Tracker to detect the slot-value pairs.

The Dialogue Policy is the main AI component of the system. The Dialogue policy learns the next most likely action to take, conditioned on the state representation of the DST. The policy can be rule based or learnt either supervised or by using reinforcement learning [29]. In the latter, reinforcement learning can be used to discover policies that that optimize an objective performance measure, like reducing the number of turns to complete a task and maintaining a high user satisfaction index.

2.1.3 Natural Language Generation

The Natural Language Generation Component converts an action provided by the Dialogue Policy into a natural language utterance to the user. A very good summary on the task of task of NLG can be found in [30]. The most common approaches to NLG nowadays consist of LSTM [31] and Transformer-based [32], [33] models.

2.2 Chatbots / AI Assistants

Chatbots are programs, utilized by users to interact with information systems in an intuitive manner, using natural ways of communication, like texting or talking. AI assistants are based on the same principles, offering help to the users whenever it is needed. Such solutions vary regarding the intelligence level of the assistants, which may settle for simply answering FAQs, or extend their capabilities well beyond that point, by resolving issues users face when using the platform. More specifically, because of the augmented capabilities and functionalities offered to the users from the combination of modern hardware and software application, the level of complexity regarding the choices one is able to make while using such a system, are quite complicated. AI assistants offer a solution to this problem, by simplifying the process and making it possible for the users to just make an inquiry regarding the actions they need to take.
The AI assistants are based on certain technologies that allow them to offer their services to the users whenever it is required. The main pillars are related to the interaction methods with the users; if a user utilizes his voice to interact with the assistant, then an Automatic Speech Recognition (ASR) technology is being employed. After that initial step, the methodology followed is the same with the text input, i.e., NLP is utilized to understand what the user needs. Usually assistants respond through written text, or vocally. In a later case, Text to Speech (TTS) tools are required.

2.2.1 Chatito/Chatette

Chatito[8] is a Domain Specific Language (DSL) tool, which aims to be used in order to generate datasets that are required for the training of NLP models, like Rasa. In essence, with Chatito developers are given the ability to automatically simulate user input. NLPs take simple and unstructured human language and extract structured data in the form of **intents** and **entities**.

Intents can be pictured as labels attached to the user's input based on the overall goal of the corresponding message. For example:

- **User input**: Hello!
  Intent: greet, 95%

Entities are pieces of information that an assistant may need in a certain context. For example:

User input: Hello! My name is Jane Doe.
Intent: greet, 98%
Entities:
Name: Jane Doe

Chatito receives as input categorized keywords, which afterwards uses to construct grammatically and syntactically correct sentences.

[8] https://rodrigopivi.github.io/Chatito/
https://github.com/rodrigopivi/Chatito

```
import ./common.chatito

%[findBookstoresByCity]('training': '100', 'testing': '100')
    *[60%] ~[hi?] ~[please?] ~[find?] ~[bookstores] ~[located at] @[city] ~[city?] ~[thanks?]
    *[40%] ~[bookstores] ~[located at] @[city]

@[city]
    ~[Athens]
    ~[Bucharest]
    ~[Madrid]

~[find]
    find
    i'm looking for
    help me find

~[located at]
    located at
    in the area of
    near

~[bookstore]
    bookstores
    places to buy books
    where to buy books
```

```
~[hi]
    hi
    hello
    hey

~[please]
    please
    plz
    pls

~[thanks]
    thanks
    thank you
```

Figure 2: Chatito word feed

In Figure *2* it can be seen how the sentences are created. Two general constructs are being provided at the beginning, with a presence of 60% and 40% of in the produced files, respectively. For each variable in the sentence, values are provided below by the developers. Once a sentence is created, it is then analyzed based on its intent and entities. This process is followed twice, once to produce a training file for the Rasa’s NLU component, and another one to test the efficiency of the training that was conducted beforehand. The content of the files produced by Chatito resembles to Figure *3*. Firstly, the intent of the user message is made clear, which is to find a bookstore. Afterwards, the key-value entity is the place, which in this example is the city of Madrid. Rasa’s NLU component will be trained with a file containing such examples, learning how to match the user input by locating keywords with the corresponding intend and values of entities, and afterwards its efficacy will be tested with a similar dataset. This process is important to take place, since it introduces easily diversity in the training data, which yields a better recognition in unseen queries in the testing dataset, thus ensuring the best user experience.

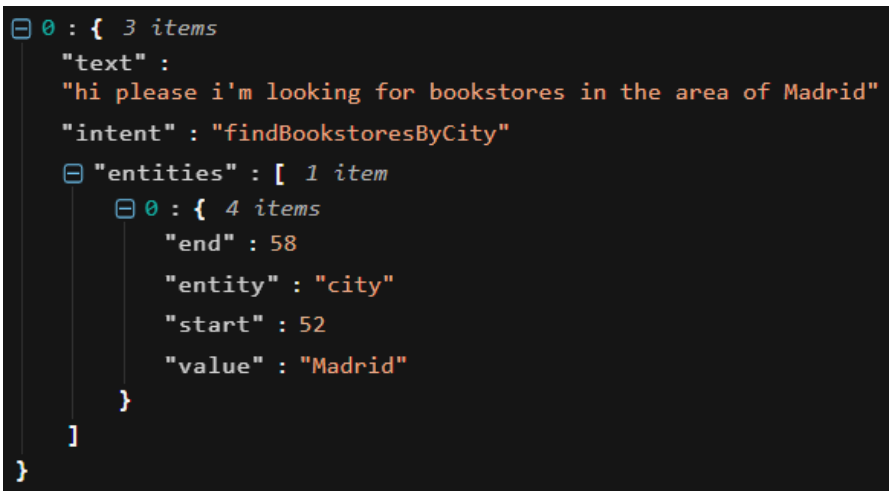

Figure 3: Chatito output

Chatette[9] is also a tool much like Chatito, generating datasets for training Rasa's NLU. Their input and output files have the same requirements and structure, providing though some advantages. Chatette is able to manage larger volumes of data easier and can break down templates into multiple files, making them easier to manage.

2.2.2 Rasa

Rasa[10] is an opensource framework for building conversational software, which can be easily tailored to the requirements that must be met on occasion. The assistants that are based on Rasa, are not static and do not make binary decisions, but rather learn from conversations that take place and adapt. This is achieved through the adoption of a machine learning model, which is initially trained by the developers. Rasa is composed by two main components: the NLU and Core.

The Rasa NLU component could be kept in mind as an interpreter between the users and the machines. The users' input is broken down and analyzed in order to understand what the corresponding *intent* is, along with the *entities* that accompany it. After this step, the Rasa Core component takes over. Rasa's dialog management component, Core, makes predictions and decides how the AI-assistant should respond based on the specific state of the conversation and the context. Core learns how to respond by observing the patterns of past and example conversation data. Its input depends on the NLU analysis, which is then processed through the policies that have been defined in order to decide which response is most appropriate to provide the user with.

3 Advanced User Experience Artificial Intelligence (AI)-based Assistant Architectural Design and Specifications

3.1 AI-based Assistant Technical Specifications - User Actions

[9] https://pypi.org/project/chatette/
[10] https://rasa.com/

This section describes all the actions that users will be able to perform using the AI-based Assistant of the INCOGNITO platform and these actions are also supposed to drive the design and implementation of the AI-based Assistant. Using the developed user interfaces of the AI-based Assistant either from their desktop or their mobile devices, users will be able to perform the following actions by selecting predefined queries to initiate each action and providing in the meantime any required by the AI-assistant additional information in textual format that the AI-based Assistant will processing using several Natural Language Processing (NLP) techniques.

Users will be able to perform the following actions using the AI-based Assistant:

1. **List of the minimum required attributes that are needed to access a specific Service Provider:** The user will be able to get informed about all the necessary identity attributes that he needs to have and share with in order to access a specific Service Provider. When selecting this action, the AI-based Assistant will subsequently ask the user for the URL of the Service Provider that the user wants to learn this information. In the end, the assistant will retrieve the required information and present to the user a list with all the identity attributes that needs to be shared with that specific Service Provider.
2. **View de-anonymization risks as a result of sharing specific identity attributes with a Service Provider:** The user will be able to get information about the de-anonymization risks that the sharing of specific identity attributes with a specific Service Provider will have. For example, if the user is about to share his/her age with a specific Service Provider that already knows country of the user, then the Service Provider may be able to also predict the financial status of that user based on the information that this Service Provider for other users that come from the same country and has a similar age as our user. When selecting this action, the AI-based assistant will subsequently ask the user for the URL of the Service Provider that the user wants to access and the attributes that the user is about to share, and the assistant will respond back with the corresponding de-anonymization risk information.
3. **View the result of stop sharing specific identity attributes with Service Providers:** The user will be able to see the consequences of stop sharing a specific identity attribute with all the Service Providers that he/she use to access. When this action is triggered by the user using the UI of the AI-based assistant, the Assistant will subsequently request from the user to choose the identity attribute that he/she wants to stop sharing from among all his/her identity attributes. Then, the AI-based assistant will process the request of the user and respond with a list of the Service Providers that will not be accessible if the user devices to stop sharing that specific identity attribute. If the user is then satisfied with the consequences of this action, then he/she will be able to automatically delete all the UMA policies or add the required UMA policies associated with these specific attributes by answering yes to the last specific question of the AI-based Assistant. If the users choose to do so, the AI-based Assistant will communicate delete all the UMA policies or create a new one that will all result to stop sharing the specific identity attribute of the user with all the Service Providers. This last step will be performed by calling the appropriate REST endpoints of the Keycloak's UMA REST API (Authorization Server).
4. **Know what INCOGNITO Service Providers can be accessed if sharing specific identity attributes:** The user will be able to get informed at any time about all the Service Providers that can be accessed if the user chooses to create a specific UMA policy that enables the sharing of one or more of his/her identity attributes with all the Service Providers that required

this attribute. When this action is triggered by the user, the AI-based Assistant will prompt the user to choose the identity attribute that he/she wants to stop sharing from among all his/her identity attributes. Then, the AI-based Assistant will process the request with one custom action implemented for that specific user action and will respond to the user with a list that includes all the Service Providers that can be access if the user share this specific identity attribute. This list may also include Service Providers that require more attributes to be shared so that the user is also informed about other attributes that he/she can share to be able to easily access other Service Providers too. If the user is satisfied with the result, then he/she will be able to automatically request the creation of the required UMA policies for these Service Providers so that he/she can automatically access them and subsequently the AI-based Assistant will create these policies by calling the appropriate REST endpoints of the the Keycloak's UMA REST API (Authorization Server).

3.2 Architectural design and the Identity Consolidator

As defined in the INCOGNITO reference architecture, the backbone of the AI-based Assistant will be implemented as a server-side module using Python programming language and it will be deployed and run on the Identity Consolidator server. This implementation will include all the required sub-modules that will be responsible for the preprocessing of the received textual actions that the user will send and will communicate with all the necessary modules of the Identity Consolidator in order to perform the required actions and respond to the user with the proper information. In this subsection, we provide more details about all the components of the AI-based Assistant that will be part of the Identity Consolidator (IDC).

3.2.1 Interaction with other INCOGNITO modules within the IDC

Figure *4* depicts all the components within the Identity Consolidator that the AI-based Assistant will be able to communicate with in order to retrieve the required identity information or perform all the required actions depending on the action that the user has performed using the UI of the AI-based assistant. We note that, both the desktop client, as well as all the mobile app clients will be able to access the AI-based Assistant back-end only through its REST API.

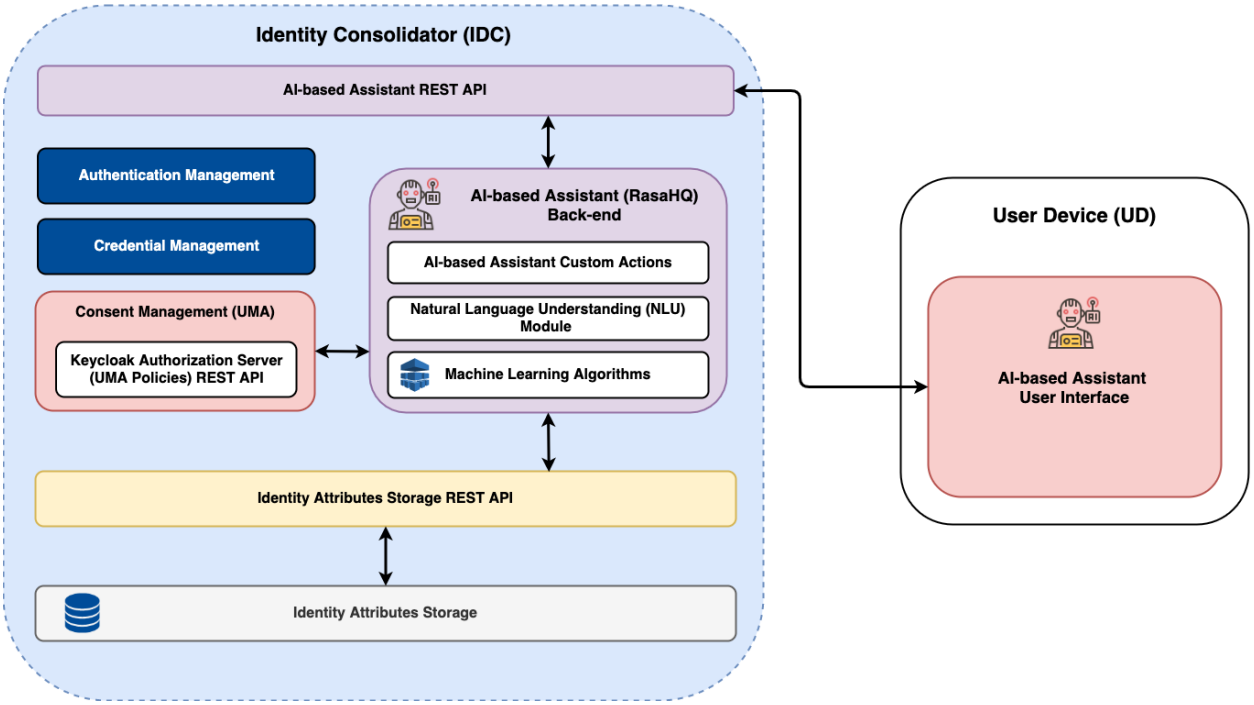

Figure 4. AI-based Assistant interaction with other modules of the Identity Consolidator

3.2.1.1 Consent Management and ABAC Policies

The AI-based Assistant custom actions module will be able to view, manage, and delete a user's UMA policies depending on the action that the user has performed, using the UI of the AI-based Assistant, using the Keycloak's Authorization Server (UMA Policies) REST APIs.

3.2.1.2 Identity Attributes Storage

The back-end module of the AI-based assistant will be able to communicate with the Identity Attributes Storage of the Identity Consolidator through the Identity Attributes Storage REST API. Through this REST API the assistant will be able to retrieve or modify all the required identity information of a specific user, depending on the action that the user has performed.

3.3 User Device

On the client-side (web and mobile), there will be existing open-source chatbot UI implementations for Android and the Web (e.g.). The AI-based Assistant's client-side will act as the interface that captures the user input and inquires and passes them to the client communication component and will also send them to the back end for analysis. All the communication between the components residing in the client and the server will be done through REST-APIs.

3.3.1 User Interface/User Experience

The User Interface will include some basic actions that the user will be able to perform. There will also be predefined queries which will be shown like an open Facebook chat. The user can choose his suitable answer by pressing a button for each question. By clicking means that he sends the query to the backend and he receives the appropriate answers.

3.3.1.1 Web User Interface

1. BotUI

BotUI[11] makes it super easy to create conversational/bot interfaces. It has an intuitive JavaScript API to add messages and show actions that a user can perform. It also gives you total control over how everything looks.

Features:

- Fully customizable layouts and messages
- Text messages can also contain links, images and icons
- Very good graphical user interface
- Quick implementation
- Open source

2. React Simple Chatbot

React Simple Chatbot[12] is a web UI for chatbots. It has a very friendly Graphical User Interface and can be fully customizable.

Features:

- Simple Form
- Custom components
- Speech recognition
- Very detailed documentation
- Open source

3. Conversational Form

Conversational Form[13] is an open-source concept by SPACE10 to easily turn your content into conversations. It features conversational replacement of all input elements, reusable variables from previous questions and complete customization and control over the styling.

[11] https://docs.botui.org/concepts.html
[12] https://lucasbassetti.com.br/react-simple-chatbot/#/docs/themes
[13] https://github.com/space10-community/conversational-form

Features:

- Build fully dynamic conversational experiences using our extensive API
- Build conditional flows with multiple paths and outcomes
- Smooth functionality on both desktop and mobile
- Reusable answers and question variations
- Easily customize the conversation
- Open source

	BotUI	React Simple Chatbot	Conversational Form
Installation/ Configuration	Easy installation	Easy installation	Very Easy installation
User Interface	Good Graphic User Interface easy to use and understand	Good Graphic User Interface easy to use and understand	Very good Graphic User Interface, easy to use and understand
Supported Browser	Chrome, Safari, Firefox	Chrome, Safari, Firefox	Chrome, Safari, Firefox
Admin UI	Yes	Yes	Yes
Open source	Yes	Yes	Yes
Development activity	Active Development, last commit was in 2020	Active Development, last commit was in 2020	Active Development, last commit was in 2019
Documentation	Detailed	Detailed	Very Detailed
Quick starts	Yes. Can be found on Github.	Yes. It offers many quick starts whose source code can be found on Github.	Yes. It offers many quick starts whose source code can be found on Github.

Moreover, there are many web users' interfaces that can be used for a chatbot. In INCOGNITO we are going to use the most appropriate based on our needs and actions. Most of the user interfaces are not open-source thus we chose the best open-source ones which are also fully customizable so we can modify it as we want.

3.3.1.2 Android User Interface

1. **Chatkit:** ChatKit[14] is a library designed to simplify the development of UI for such a trivial task as chat. It has flexible possibilities for styling, customizing and data management.

[14] https://github.com/stfalcon-studio/ChatKit?fbclid=IwAR0rJqY2bLFAaZzaAylvRS7VnG7HZQmyb3iuiObXioFCzJlbQ99opjzbkoU

Features:

- Ready-to-use already styled solution for quick implementation
- Default and custom media messages
- Fully customizable layouts - setting styles out of the box (use your own colors, text appearances, drawables, selectors and sizes) or even create your own custom markup or/and holders for unique behaviour
- List of dialogs, including tete-a-tete and group chats, markers for unread messages and last user message view
- List of messages (incoming and outcoming) with history pagination and already calculated dates headers
- Different avatars with no specific realization of image loading - you can use any library you want
- Selection mode for interacting with messages
- Links highlighting
- Easy dates formatting
- Own models for dialogs and messages - there is no converting needed
- Ready to use message input view
- Custom animations
- Open source

2. **Android Chat UI:** Android Chat UI[15] is a library that is still in its very early stages and can be fully customizable.

Features:

- Ability to use custom item layout
- Ability to send and receive multimedia messages like images, embedded locations and even videos
- Ability to track and update individual messages (Useful to be able to show delivered/read/unread status or the like)
- Open source

3. **ChatBot:** Chatbot[16] is a library using Artificial intelligence Markup Language (AIML). It is a very basic UI that can be customized based on your needs.

Features:

- User friendly Graphical Interface
- Quick implementation
- Fully Customizable

[15] https://github.com/timigod/android-chat-ui?fbclid=IwAR3cEtLHkBqfFp47_7VB2ye3BsFTJqA1I90aF8nO2vhz-MxR_7M6CxWS1JY
[16] https://github.com/Hariofspades/ChatBot?fbclid=IwAR2rIcWK3vnlPe7pws4EeH9E9vSatCtjLSPdYUqEfn4NFuT0RC2W6wVkXQQ

- Open source

4. **Simple Chat:** Simple Chat[17] is a room chat system for Android that supports real-time, IRC Style using firebase as database.

   Features:

   - Real-time
   - Very easy to integrate in your project
   - Anonymous users or with their own username
   - Anti-flood protection
   - Profanity filter
   - Change nickname notifications
   - Helper methods for customizations
   - Open source

5. **Chateau:** Chateau[18] is a framework for adding (or improving) chat functionality in any Android app. Built in a modular way using Model-View-Presenter (MVP) and Clean Architecture, it can easily be integrated with your chat backend with only minor changes to the included UI.

   Features:

   - Easy to understand code, by consistently applying design patterns across the framework and example app
   - Easy to integrate with any chat backend
   - Well documented with good test coverage
   - As few as possible external dependencies, because no one likes a bloated library
   - Robust Architecture
   - Open source

	Chatkit	Chateau	Chat UI	ChatBot	Simple Chat
Installation/ Configuration	Very Easy	Very Easy	Easy	Easy	Easy
User Interface	Customizable and very good Graphic User Interface	Customizable and basic Graphic User Interface	Customizable and very basic Graphic User Interface	Customizable and very basic Graphic User Interface	Customizable and very basic Graphic User Interface

[17] https://github.com/AndreiD/SimpleChat?utm_source=android-arsenal.com&utm_medium=referral&utm_campaign=4206
[18] https://github.com/badoo/Chateau?utm_source=android-arsenal.com&utm_medium=referral&utm_campaign=4127

Admin UI	No	No	No	No	No
Open source	Yes	Yes	Yes	Yes	Yes
Development activity	Last activity was in 2019	Last activity was three years ago	Not active the last two years	Not active the last few years	Not active the last few years
Documentation	Detailed and very simple	Very Detailed and helpful	Includes only the basics	Proof of concept, includes very basic documentation	Not helpful and very limited
Quick starts	Yes	Yes	Yes	No	No

In a nutshell, there exist many plain UI libraries on the web and most of the aren't open source. Most of the UI libraries that are open source are full stack solutions and work with firebase (Google, 2020). However, Rasa will be our backend in INCOGNITO and is compatible with almost all UIs (More details for Rasa in subsection 4.1). Therefore, we decided to implement a UI from scratch by combining all the available open-source UI libraries that we found with the aim to create our own Chatbot UI customized to the needs, the look and feel of an AI-based Assistant.

4 Natural Language Understanding Pipeline

For the Natural Language Understanding Pipeline we will use the powerful and popular RASA framework[19]. Below we present the RASA architecture, components and introductory examples on its function.

4.1 Rasa Architecture

An overview of the Rasa Open-Source architecture is provided by Figure *5*. The two primary components are Natural Language Understanding (NLU) and dialogue management (Core).

NLU is the part that handles intent classification, entity extraction. It is shown below as the *NLU Pipeline* because it processes user utterances using an NLU model that is generated by the trained pipeline.

The dialogue management component decides the next action in a conversation based on the context. This is displayed as the *Dialogue Policies* in Figure *5*.

[19] www.rasa.com

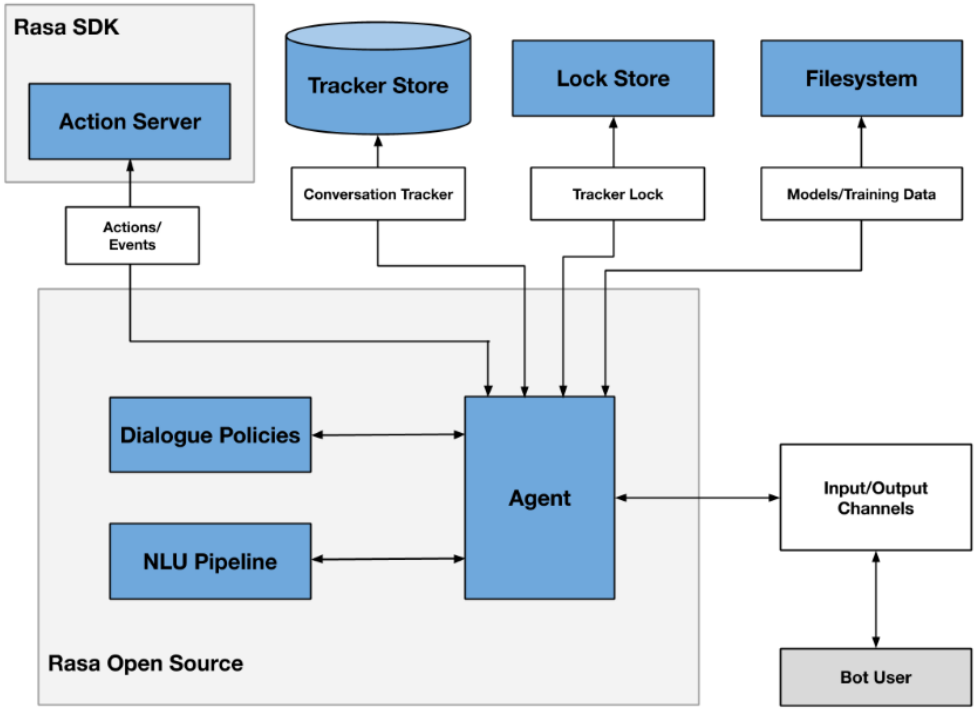

Figure 5: Rasa Architecture

NLU and Core are independent and one can use NLU without Core, and vice versa. Though Rasa recommends using both. In Figure *6* a simplified visualization of Rasa's functionality is presented, showcasing the internal processes that take place from the moment Rasa will receive input from the user. The *Database* component contains information that is used in order to determine the appropriate responses that need to be given to the user.

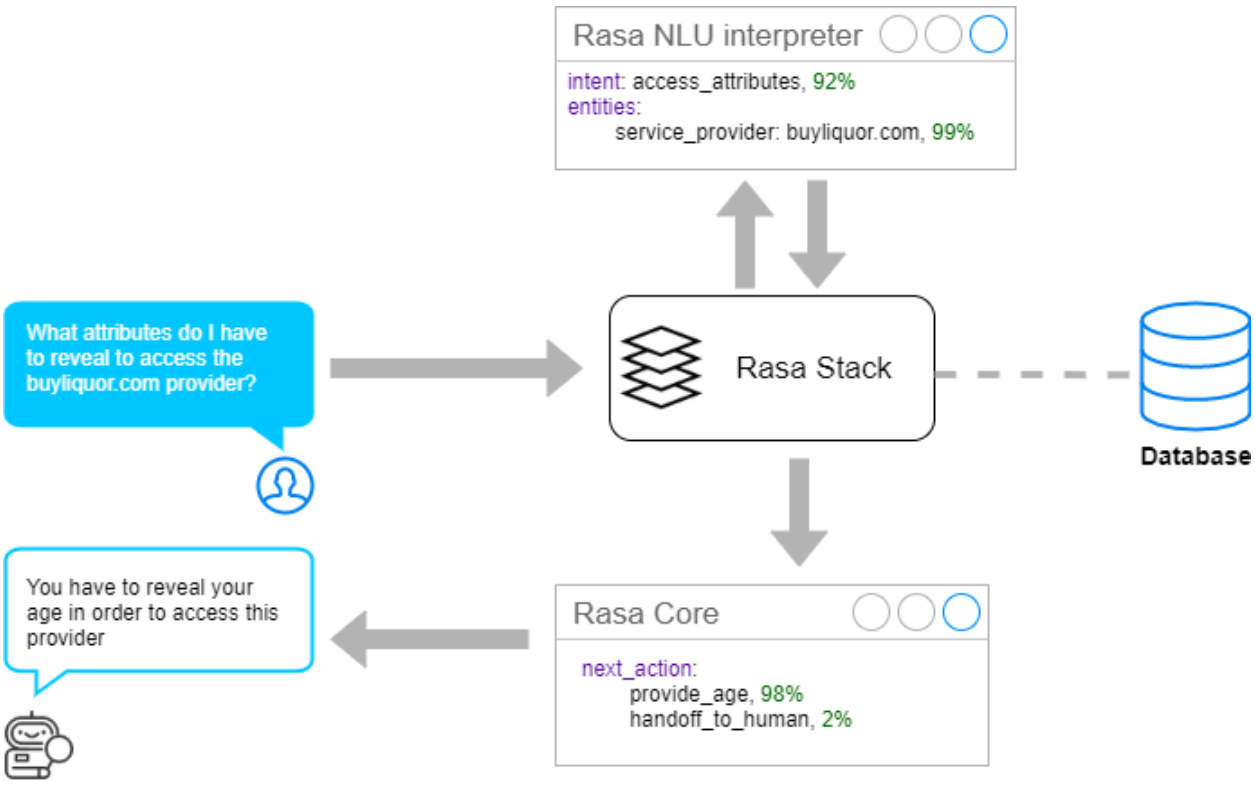

Figure 6: RASA internal and external interaction and data flow

4.2 Important concepts

- *Intent* — Intent is nothing but what the user is aiming for. For example — if the user says *"Reserve a table at Cliff House tonight"* the intent can be classified as to book the table.
- *Entity* — Entity is to extract the useful information from the user input. From the example above "*Reserve a table at Cliff House tonight*" the entities extracted would be place and time. Place — Cliff House and Time — tonight.
- *Stories* — Stories define the sample interaction between the user and chatbot in terms of intent and action taken by the bot. Like in the example above bot got the intent of booking the table and entities like place and time but still, there is an entity missing — no of people and that would make the next action from the bot.
- *Actions* — Actions are basically the operations performed by the bot either asking for some more details to get all the entities or integrating with some APIs or querying the database to get/save some information.

The following Figure 7 shows how the various concepts are connected under the RASA framework.

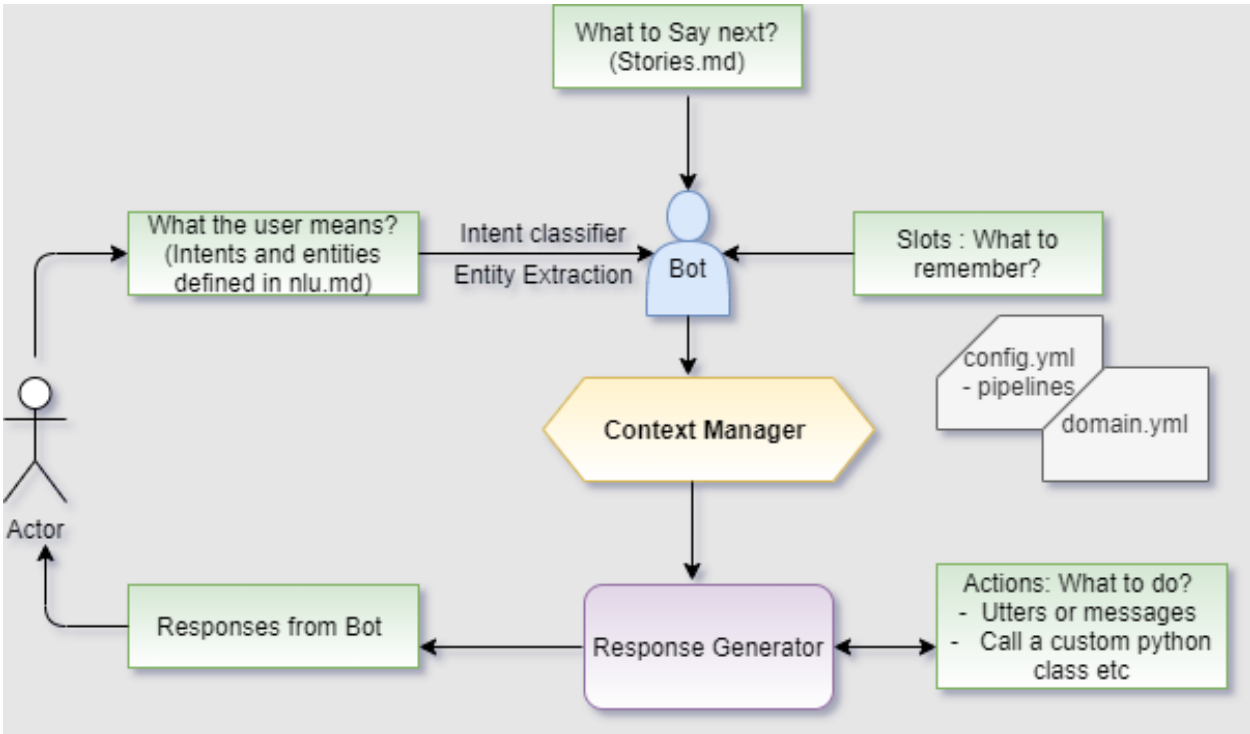

Figure 7: RASA internal processing flow

4.3 NLU files

NLU training file: It contains some training data in terms of user inputs along with the mapping of intents and entities present in each of them. The more varying examples you provide, better your bot's NLU capabilities become. For the creation of this file is where it comes in handy tools for sentence-template expansion, like Chatito.

Stories file: Though we will be discussing it in detail in an upcoming section, it is worth to briefly mention that this file contains sample interactions the user and bot will have. Rasa (Core) creates a probable model of interaction from each story.

Domain file: This file defines all the intents, entities (slots), possible actions taken by the agent, response templates and some more information. The intents and the slots should match those of the NLU training file, whereas the intents, slots and actions should match the ones present in the Stories file. We will discuss this file in detail in the RASA Core section.

#Needs attention:

NLU.yml

nlu:

- intent: inform_attributes

examples: |

- I want information (about the attributes) of this service
- What are the attributes needed for this service

- intent: check_attributes

Examples: |

- What attributes I am sharing (with the service)
- How much personal data I am sharing (with the service)

- intent: grant_access

Examples: |

- (Please) share my [email address]{“entity”:”attribute”, “value”:”email”}
- (Please) give access to my [phone number]{“entity”:”attribute”, “value”:”phone”}

- intent: remove_access

Examples: |

- I want to restrict the access to my [credict card number] {“entity”:”attribute”, “value”:”card”}

Domain.yml:

version: "2.0"

intents:

- inform_attributes
- check_attributes
- grant_access

- remove_access
- greet
- bye

- affirmative

- negative

entities:

- attribute

```
slots:
  attribute:
    type: list
    influence_conversation: true
  current_attributes:
    type: list
    influence_conversation: false
  service_attributes
    type: list
    influence_conversation: false

actions:


- action_get_service_attributes
- action_get_personal_attributes
- action_remove_access
- action_grant_access


- utter_greet
- utter_bue
- utter_inform
- utter_check
- utter_reply
- utter_default
- utter_anything_else

templates:
  utter_greet:
    - text: Hi, how are you?
    - text: Hello, How are you doing?
  utter_bye:
    - text: Bye and have a nice day
    - text: Bbye and have a nice day
  utter_default:
    - text: I am not sure what you're aiming for
    - text: I am sorry but I am not able to get you.
  utter_reply:
    - text: Ok, done.
```

```
  - text: Great, done.
 utter_check:
  - text: The service has access to {current_attributes}
 utter_inform:
  - text: This service needs access to {service_attributes}
 utter_anything_else:
  - text: Is there anything else I can help you with?
  - text: Let me know if there is anything else I can help you with
```

Stories.yml

```
stories:
- story: share attributes
  steps:
  - intent: greet                    # user message with no entities
  - action: utter_greet
  - intent: inform_attributes  # user message with no entities
  - action: action_get_service_attributes
  - action: utter_inform
  - intent: grant_access                    entities:
    - attribute: "email"
    - attribute: "phone"
  - action: utter_reply              # action that the bot should execute
  - action: utter_anything_else
  - intent: negative
  - action: utter_bye
```

Rasa Core — Dialog Management

In Figure 8 we can observe how all the Rasa core components interact with each other.

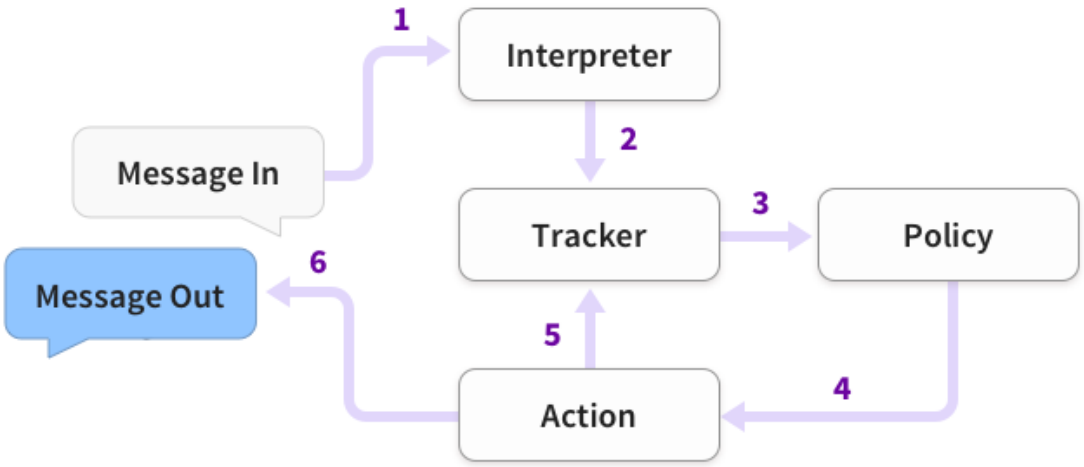

Figure 8: Rasa Core High Level Architecture

The user ’s inputs are expressed as intents with corresponding entities, and chatbot responses are expressed as actions. For dialog training, Rasa has 4 main components:

1. Domain (domain.yml)
Consists of five key parts consisting of intents, entities, slots, actions, and templates.

- **slots**: are the bot’s memory**.** They act as a key-value store which can be used to store information the user provided (e.g., their home city) as well as information gathered about the outside world (e.g., the result of a database query).
- **actions**: the bot's response to user input or data manipulation and internal processing. There are 3 kinds of actions in Rasa Core: **default actions, utter actions & custom actions**
- **templates**: they are messages that the bot will send back to the user.

Example of a domain for our bot:

https://gist.githubusercontent.com/itsromiljain/948d8c1a1569f80f873df6d29c6492af/raw/7b9603ba70ed0d6e8ad8c55923deb9952c33bc51/domain

```
slots:
  category:
    type: text

entities:
- category
```

```
intents:
- greet
- fine_ask
- fine_normal
- news
- thanks
- bye

actions:
- action_restart
- action_get_news
- utter_greet
- utter_reply
- utter_help
- utter_anything_else
- utter_ofc
- utter_bye
- utter_default

templates:
  utter_greet:
    - text: Hey, how are you?
    - text: Hello, How are you doing?
  utter_reply:
    - text: I'm doing great. Please let me know what I can do for you.
    - text: I'm doing great. Tell me How can I help you today?
  utter_help:
    - text: Great. How can I help you?
    - text: Great. Tell me How can I help you?
    - text: Great. Tell me what all news you would like to get.
  utter_anything_else:
    - text: No worries. Is there anything else I can help you with?
    - text: No worries. Let me know if there is anything else I can help you with
  utter_ofc:
    - text: I can definitely help you. The top 5 news of the {category}
    - text: Surely, I can help you. The top 5 news of the {category}
  utter_bye:
    - text: Bye and have a nice day
    - text: Bbye and have a nice day
  utter_default:
    - text: I am not sure what you're aiming for
    - text: I am sorry but I am not able to get you.
```

- text: My appologies but I am not able to get you

2. Stories (stories.md)

Stories are a type of training data used to train your assistant's dialogue management model. Stories can be used to train models that are able to generalize to unseen conversation paths.

User	*intent:greet*	Hi
Bot	utter_greet	Hi, How are you doing?
User	*intent:fine_ask*	I am good, How are you doing?
Bot	utter_reply	I am doing great. Tell me How can I help you today?
User	*intent:news*	Share some news from around the World
Bot	utter_ofc action_get_news	Sure, Here you go. Below are Top 5 news of the World
User	*intent:thanks*	Thanks
Bot	utter_anything_else	No worries, Tell me If I can help you with anything else
User	*intent:bye*	No, I am good as of now. Bye
Bot	utter_bye	Bye and have a nice day

Figure 9: Conversational path example of a story

A story is a representation of a conversation between a user and an AI assistant, converted into a specific format where user inputs are expressed as intents (and entities when necessary), while the assistant's responses and actions are expressed as action names. An example can be seen in Figure 9, and a corresponding file can be found here:

https://gist.githubusercontent.com/itsromiljain/4e5effffac67b9370ce90ae5c59cbc54/raw/fc216e418cbda48d866d5b733bfc5bd2c4a19801/stories

fallback- utter_default
greeting path 1* greet- utter_greet
fine path 1* fine_normal- utter_help
fine path 2* fine_ask- utter_reply
news path* news- utter_ofc- action_get_news
thanks path 1* thanks- utter_anything_else
bye path 1* bye- utter_bye

All **actions** executed by the bot, including responses are listed in **stories** under the action key.

Rasa docs on Stories: https://rasa.com/docs/rasa/stories/

3. Policies (policy.yml)

The rasa core policy decides which action to take at every step in the conversation. There are different policy algorithms to choose from, and one can include multiple policies in a single rasa

core Agent but at every turn, the policy which predicts the next action with the highest confidence will be used.

The process of selecting the policy algorithm(s) for training the agent is pretty straightforward. Only the name of the module along with its parameters have to be defined in the policy file.

Example of a policy file:

https://gist.githubusercontent.com/itsromiljain/c512673350d056ff342da8171da05e7c/raw/7d81d1e06928fc0f928957d91d91ded0b1e5ebea/policy.yml

Rasa docs on policies: https://rasa.com/docs/rasa/policies

4. Rasa Actions (actions.py)

https://rasa.com/docs/rasa/actions

After each user message, the model will predict an action that the assistant should perform next.

Types of actions:

Responses: A response is a message the assistant will send back to the user. This is the action you will use most often, when you want the assistant to send text, images, buttons or similar to the user.

- Custom Actions: A custom action is an action that can run any code you want. This can be used to make an API call, or to query a database for example.
- Forms: Forms are a special type of custom action, designed to handle business logic. If you have any conversation designs where you expect the assistant to ask for a specific set of information, you should use forms.
- Default Actions: Default actions are actions that are built into the dialogue manager by default. Most of these are automatically predicted based on certain conversation situations. You may want to customize these to personalize your assistant.

A custom action can run any code , including API calls, database queries etc. They can ask for identity attributes to reveal, to access a service provider, to check the privacy risk etc.

Rasa Core can call endpoints specified by us or perform SQL database queries when a custom action is predicted. An endpoint could be a web server that reacts to this call, runs the code and optionally returns information to modify the dialogue state.

Rasa docs on Custom Actions: https://rasa.com/docs/rasa/custom-actions

5 Conclusions

Deliverable D5.1 "Specification and initial design of the Advanced User Experience / User Interface (UI/UX) Artificial Intelligence (AI)-based assistant pipeline" is the first deliverable of Work Package 5. Addressing the task T5.1 objectives, the key achievement that is accomplished through the scientific research activities reported in this deliverable, is the definition of the specifications and the initial design of the essential components of the AI-based assistant, which plays a vital role in the INCOGNITO platform. An extensive review of the related available technologies has been performed and a prototype design has been made. Thus, Task 5.1 has been fully achieved, and the results are depicted in this deliverable. We laid the foundation for the INCOGNITO AI-based assistant and designed a new unique NLU-machine learning pipeline, based on the Rasa framework described in section 4, which will be utilized when users interact with the assistant and require help to be provided. The Rasa framework, which has been tailored to meet the INCOGNITO platform requirements provide users with the appropriate response based on the input of the user queries, as it is described in section 4.3. The information exchange between those parties will be entirely in the form of natural language, removing any barriers that could potentially pose difficulties in the communication process between the users and the AI-based assistant. This progress constitutes the first stage of the design and components prototyping of the machine learning pipeline. To this end, Chatito is also utilized to train and test the protype in order to provide a strong machine learning infrastructure. Emphasis has also been given to the User Interface, whether it is on a Web or Android app. The user experience and ease of use for the application heavily depends on the optimal placement of choices on the screen for a seamless interaction.

The research progress presented in this deliverable will be carried out further in D5.2, where the capabilities of the current prototype will be extended. More specifically, as dictated in Task 5.2, the AI-based assistant will adopt a strong dynamic character, which will allow it to evolve and adapt its action suggestions. The assistant will continuously learn from its interaction with the users, along with feedback that will be provided, towards achieving the goal of providing the needed information to the user and minimize the possibilities of imprecise responses. In D5.3, the result stemming from the efforts of Task 5.3 will be depicted, where the overall user experience with the utilized Interfaces will be evaluated, along with effectiveness, efficiency and satisfaction of end-users.

6 References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986, doi: 10.1038/323533a0.

[2] J. L. Elman, "Finding structure in time," *Cogn. Sci.*, 1990, doi: 10.1016/0364-0213(90)90002-E.

[3] R. Kneser and H. Ney, "Improved backing-off for M-gram language modeling," 1995, doi: 10.1109/icassp.1995.479394.

[4] R. Caruana, "Multitask Learning," *Mach. Learn.*, 1997, doi: 10.1023/A:1007379606734.

[5] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A Neural Probabilistic Language Model," 2003, doi: 10.1162/153244303322533223.
[6] T. Mikolov, M. Karafiát, L. Burget, C. Jan, and S. Khudanpur, "Recurrent neural network based language model," 2010.
[7] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, 2011.
[8] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. H. Černocký, "Empirical evaluation and combination of advanced language modeling techniques," 2011.
[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.
[10] A. Graves, A. R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," 2013, doi: 10.1109/ICASSP.2013.6638947.
[11] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014.
[12] M. T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015, doi: 10.18653/v1/d15-1166.
[13] A. Kannan *et al.*, "Smart reply: Automated response suggestion for email," 2016, doi: 10.1145/2939672.2939801.
[14] H. Mino, M. Utiyama, E. Sumita, and T. Tokunaga, "Key-value Attention Mechanism for Neural Machine Translation," *Proc. Eighth Int. Jt. Conf. Nat. Lang. Process. (Volume 2 Short Pap.*, 2017.
[15] A. Kuncoro, C. Dyer, J. Hale, D. Yogatama, S. Clark, and P. Blunsom, "LSTMs can learn syntax-sensitive dependencies well, but modeling structure makes them better," 2018, doi: 10.18653/v1/p18-1132.
[16] T. Blevins, O. Levy, and L. Zettlemoyer, "Deep RNNs encode soft hierarchical syntax," 2018, doi: 10.18653/v1/p18-2003.
[17] X. P. Qiu, T. X. Sun, Y. G. Xu, Y. F. Shao, N. Dai, and X. J. Huang, "Pre-trained models for natural language processing: A survey," *Science China Technological Sciences*. 2020, doi: 10.1007/s11431-020-1647-3.
[18] Y. N. Dauphin, G. Tur, D. Hakkani-Tür, and L. Heck, "Zero-shot learning for semantic utterance classification," 2014.
[19] G. Tur, L. Deng, D. Hakkani-Tür, and X. He, "Towards deeper understanding: Deep convex networks for semantic utterance classification," 2012, doi: 10.1109/ICASSP.2012.6289054.
[20] H. B. Hashemi, A. Asiaee, and R. Kraft, "Query Intent Detection using Convolutional Neural Networks," *WSDM QRUMS 2016 Work.*, 2016, doi: 10.1145/1235.
[21] J. K. Kim, G. Tur, A. Celikyilmaz, B. Cao, and Y. Y. Wang, "Intent detection using semantically enriched word embeddings," 2017, doi: 10.1109/SLT.2016.7846297.
[22] I. Casanueva, T. Temčinas, D. Gerz, M. Henderson, and I. Vulić, "Efficient intent detection with dual sentence encoders," *arXiv*. 2020, doi: 10.18653/v1/2020.nlp4convai-1.5.
[23] P. Xu and R. Sarikaya, "Convolutional neural network based triangular CRF for joint intent detection and slot filling," 2013, doi: 10.1109/ASRU.2013.6707709.
[24] G. Mesnil *et al.*, "Using recurrent neural networks for slot filling in spoken language understanding," *IEEE Trans. Audio, Speech Lang. Process.*, 2015, doi:

10.1109/TASLP.2014.2383614.

[25] G. Castellucci, V. Bellomaria, A. Favalli, and R. Romagnoli, “Multi-lingual intent detection and slot filling in a joint BERT-based model,” *arXiv*. 2019.

[26] M. Henderson, B. Thomson, and S. Young, “Word-based dialog state tracking with recurrent neural networks,” 2014, doi: 10.3115/v1/w14-4340.

[27] T. H. Wen *et al.*, “A network-based end-to-end trainable task-oriented dialogue system,” 2017, doi: 10.18653/v1/e17-1042.

[28] N. Mrkšic, D. Séaghdha, T. H. Wen, B. Thomson, and S. Young, “Neural belief tracker: Data-driven dialogue state tracking,” 2017, doi: 10.18653/v1/P17-1163.

[29] S. Choudhary, P. Srivastava, L. Ungar, and J. Sedoc, “Domain aware neural dialog system,” *arXiv*. 2017.

[30] S. Santhanam and S. Shaikh, “A Survey of Natural Language Generation Techniques with a Focus on Dialogue Systems - Past, Present and Future Directions,” *arXiv*. 2019.

[31] T. H. Wen, M. Gašić, N. Mrkšić, P. H. Su, D. Vandyke, and S. Young, “Semantically conditioned lstm-based Natural language generation for spoken dialogue systems,” 2015, doi: 10.18653/v1/d15-1199.

[32] M. Lewis *et al.*, “BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension,” *arXiv*. 2019, doi: 10.18653/v1/2020.acl-main.703.

[33] B. Peng *et al.*, “Few-shot natural language generation for task-oriented dialog,” *arXiv*. 2020, doi: 10.18653/v1/2020.findings-emnlp.17.