Technical documentation

Talking Machines

Voice assistants for Machine interaction in Industry 4.0

Saliu Bah

Jung Eun Park

Marco Lenz

Cristina Vizán Olmedo

Content

Figures

Motivation

Our project aimed to explore the chances of smart assistants, respectively chatbots, for Industry 4.0 in Hekuma machines, as this project is an industry cooperation between the Hekuma GmbH and the HSA. The Hekuma GmbH is a leading special machine construction company from the Munich area. A smart assistant integrated in their special machines would provide the great benefit of an actor being able to intuitively obtain information from the machine. There is no need for a special training concerning the machine's interface, as the actor-machine-interaction would play out similar to an ordinary everyday human to human interaction.

State of the art

The history of chatbots

Chatbots are computer programmes that allow the user to have a conversation with a smart assistant through the use of natural language. This technology emerged in the 1960s with the aim of creating applications capable of behaving like a human, so that users could not determine whether they were talking to a real person or not.

Today there are many types of chatbots, from small FAQs to more complex systems such as Siri or Alexa, all with different characteristics, purposes, structures and functionalities, and thanks to this many companies have become interested in this technology.
The use of chatbots is booming in recent times and this is partly thanks to technological improvements such as Machine Learning, which is a type of artificial intelligence that achieves more accurate results thanks to automation (Burns, 2021).

In 1966, Joseph Weizenbaum launched ELIZA, the first bot capable of conversing in English on any subject, using tags to understand texts and catalogue them (Weizenbaum, 1966).

A few years later, in 1995, Richard Wallace created ALICE, "Artificial Linguistic Internet Computer Entity". It was based on ELIZA, being able to collect examples of natural language across the web. It used patterns to manage conversations with the user (B, 2020).

In 2011, Siri appeared. Its knowledge base grows based on the number of people using virtual agents and uses the data provided (such as pronunciation, meaning and language region) to improve the user experience (Siri Team, 2017).

Current use of chatbots

Knowing the history of the evolution of the main chatbots, it is also necessary to know what makes them possible. Their recent advancements are mainly thanks to Artificial Intelligence and more specifically to the branch of NLP (Natural Language Processing) (Richardson, 2018).

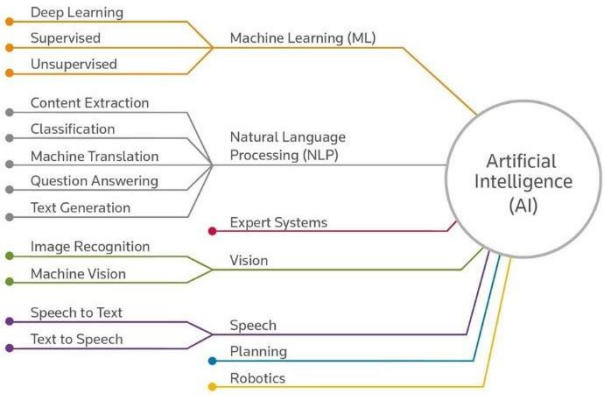


Figure 1 Branches of artificial intelligence

There are many smart assistants on the market for home automation and personal tasks (Petrock, 2020). But the global smart assistant market is still expected to grow significantly in the next 5 years. One of the key drivers in Europe for this development is the adaptation of smart-assistant-based-solutions in the industrial sector (Market Research Future, 2021). Which means that this project takes place right at the frontiers of development.

Related approaches

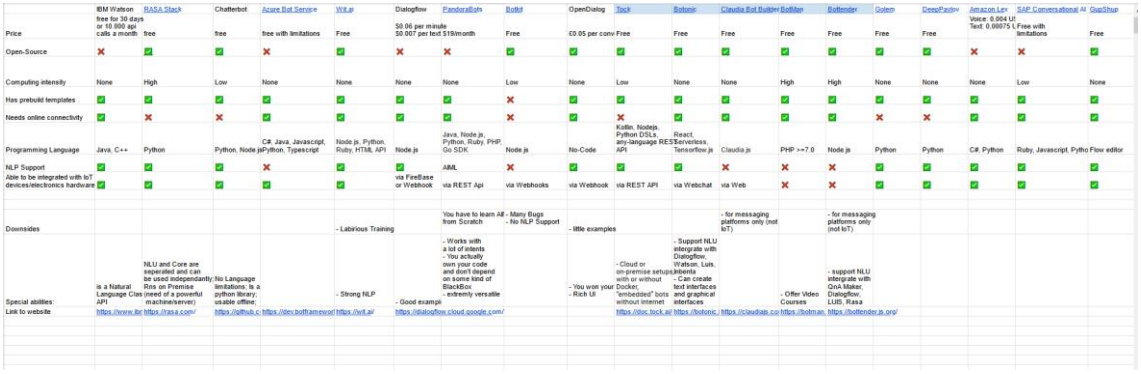
Virtual assistants have been a part of our lives for a few years now, which is why many companies with machines have also considered implementing them in their plants. Companies such as Nokia, Poyry and Infosys have joined forces to create KRTI 4.0, a new development framework that will allow industry to have greater control over their facilities and machinery (Infosys Limited, 2022).

Scientific Question

Hekuma wanted to explore if and how actors in a factory can intuitively request information from their machines by interacting with the machines through a smart assistant.

Picked concept

As we found out there are many frameworks on the market to solve this question, that is why we made a study about the most popular ones.



	IBM Watson	RASA Stack	Chatterbot	Azure Bot Service	Wit.ai	Dialogflow	PandoraBots	Botkit	OpenDialog	Tock	Botonic	Claudia Bot Builder	botMan	Botfender	Golem	DeepPavlov	Amazon Lex	SAP Conversational AI	GupShup
Price	free for 30 days or 10.000 api calls a month	free	free	free with limitations	Free	$0.06 per minute $0.007 per text	$19/month	Free	€0.05 per conv	Free	Free	Free	Free	Free	Free	Free	Voice: 0.004 U Text: 0,00075 U	Free with limitations	Free
Open-Source	✗	✓	✓	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓
Computing intensity	None	High	Low	None	None	None	None	Low	None	Low	None	None	High	High	None	None	None	Low	None
Has prebuild templates	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Needs online connectivity	✓	✗	✗	✓	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓	✗	✗	✓	✓	✓
Programming Language	Java, C++	Python	Python, Node.js	C#, Java, Javascript, Python, Typescript	Node.js, Python, Ruby, HTML API	Node.js	Java, Node.js, Python, Ruby, PHP, Go SDK	Node.js	No-Code	Kotlin, Nodejs, Python DSLs, any-language REST API	React, Serverless, Tensorflow.js	Claudia.js	PHP >=7.0	Node.js	Python	Python	C#, Python	Ruby, Javascript, Python	Flow editor
NLP Support	✓	✓	✓	✗	✓	✓	AIML	✗	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓
Able to be integrated with IoT devices/electronics hardware	✓	✓	✓	✓	✓	via FireBase or Webhook	via REST Api	via Webhooks	via Webhook	via REST API	via Webchat	via Web	✗	✗	✓	✓	✓	✓	✓
Downsides					- Labrious Training		You have to learn All from Scratch	- Many Bugs - No NLP Support	- little examples			- for messaging platforms only (not IoT)		- for messaging platforms only (not IoT)					
Special abilities	is a Natural Language Class API	NLU and Core are seperated and can be used independantly - Ros on Premise (need of a powerful machine/server)	No Language limitations: Is a python library; usable offline;		- Strong NLP	- Good exampl	- Works with a lot of intents - You actualy own your code and don't depend on some kind of BlackBox - extremly versatile		- You won your - Rich UI	- Cloud or on-premise setups with or without Docker, "embedded" bots without internet	- Support NLU intergrate with Dialogflow, Watson, Luis, Libenia - Can create text interfaces and graphical interfaces		- Offer Video Courses	- support NLU intergrate with QnA Maker, Dialogflow, LUIS, Rasa			
Link to website	https://www.ibr	https://rasa.com/	https://github.c	https://dev.botframewor	https://wit.ai/	https://dialogflow.cloud.google.com/				https://doc.tock.ai/	https://botonic.	https://claudiajs.co	https://botman.	https://botfender.js.org/					

Figure 2 Frameworks

Framework candidates

We discarded several frameworks that don't meet the following basic requirements:

- Offline usability
- Multi-Language-Support
- Voice + Text input
- Communication type should be reactive and proactive
- Highly maintained

After reviewing several frameworks, the following frameworks stand out as candidates:

- **ChatterBot**

  ChatterBot is a Python library that makes it easy to generate automatic responses to user input. It has machine learning algorithms, so it is also easy to use and lightweight but hasn't voice integration. A lot of coding has to be done for maintenance and the last stable version was released 1 year ago (ChatterBot, 2021).

Figure 3 Logo ChatterBot

- **Tock**

  Tock is an open conversational language where programming is also possible in Typescript in addition to Python but it has no voice integration and only an NLP, so it would have to be combined with a machine learning framework (e.g. Rasa, TensorFlow). It is still maintained on github because the community is quite small (40 contributors) (Tock, 2021).

Figure 4 Logo Tock

- **Golem**

  Golem is a python framework designed to build chatbots for Whatsapp, Messenger, Telegram and other platforms. A very small team (3 people) is contributing to this project (Golem, 2018).

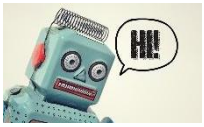


Figure 5 Logo Golem

- **DeepPavlor**

  DeepPavlov is an open-source code for the development of chatbots and smart assistants. It has pre-trained NLP models and has comprehensive and flexible tools but no voice integration. It is still maintained on github as the community is quite small (66 contributors) (DeepPavlov, 2021).

Figure 6 Logo DeepPavlov

- **Rasa Stack**

  Rasa is the platform that allows you to create personalised interactions. It has pre-trained NLP models and is very well maintained (more than 600 contributors) and supports text and speech. The problem is that it can only be developed in Python (Rasa, 2022a). Hekuma's preferred language in this project was TypeScript.

Figure 7 Logo Rasa Stack

After having analysed all these frameworks in depth, we decided that the best one for us was Rasa Stack. The first reason we chose Rasa is that it works offline, a very important fact because Internet access isn't pervasive for every machine. Secondly, Rasa is well maintained, our project can be continued by Hekuma and luckily Rasa constantly updates its version. It also allows one to have a fully customised code, one is able to have custom actions and one can change the NLP (or to be more specific NLU)[1] model and it's training. This framework has a website where the Rasa developers try to answer questions (Forum Rasa, n.d.). In fact, we received a lot of help from them. Finally, we would like to highlight the security that the company would have. The security of the data and its treatment is very important in companies like Hekuma, thanks to Rasa this data is more secure as it is trained and executed on the user's machine and not as in other platforms where this happens on the provider's servers.

Overall architecture of the chatbot

Once the framework is chosen, the need arises to organise all the elements that will participate in the functioning of the chatbot, so that it is possible to see which elements are part of the chatbot and how they interact with each other.

[1] For an explanation of the difference between the term NLP and NLU please read the **Rasa Architecture** chapter.

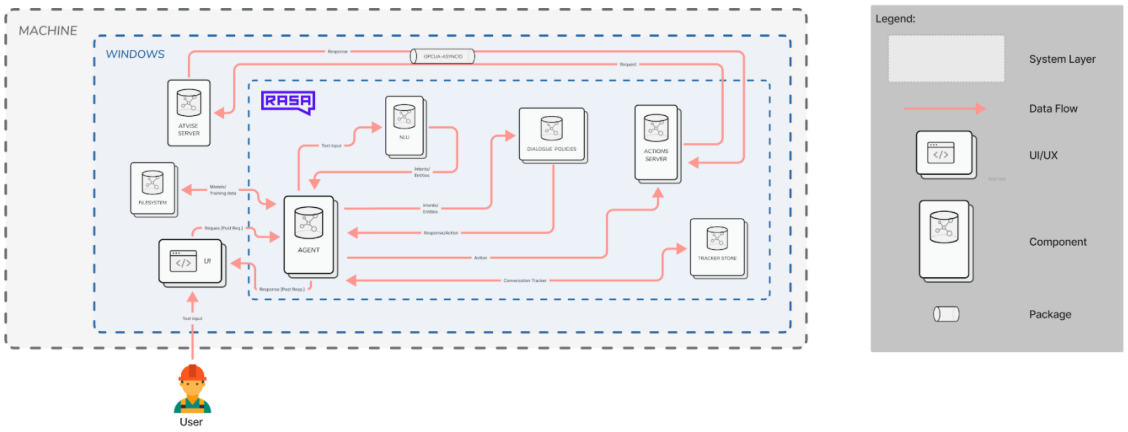


Figure 8 Chatbot architecture

With a very limited amount of training data in the form of classified example sentences, Rasa makes it very easy to analyse intents, extract entities and connect them to answers or possible actions. Actions are defined through Rasa Actions and make it possible for us to attach our chatbot to an industrial machine.

As we were only supposed to develop a prototype, the industrial machine in our case is represented by an Ativise server. It contains information about the state of an imaginary machine with alarm values and other important operating information. Changing values on this atvise server does, at this stage, not lead to any changes on a real machine. But it was incredibly useful for prototyping.

As said, the atvise server exchanges data with our Rasa Actions server through OPC UA. It is a machine-to-machine communication protocol. Therefore, our Rasa Actions server needs to understand this form of communication. This was made possible by using the library opc ua asyncio. opcua-asyncio is a Python 3 library and works perfectly well with our Rasa Actions server, which also needs to be written in Python.

Also note that we use two versions of Opc Ua, the python-opcua version (previous version of asyncio) to get the alarm status information and act accordingly and the asyncio version for everything else.

Implementation

To be able to develop the smart assistant and to be able to run it, you must have Rasa and Opc Ua correctly installed, below you will find the steps to have both elements available.

Architecture of Rasa

In order to understand our implementation, you first need to understand Rasa, that's why we will take a quick look into the Rasa architecture.

It is difficult for a chatbot to understand exactly what you want to say, as there are many different ways to ask the same question. So, the challenge is for them to adapt to any circumstance and not have a single strategy. Rasa is a company that works in conversational artificial intelligence. Its product, Rasa Stack, is the combination of two Open Source Libraries: Rasa NLU and Rasa Core. Rasa NLU aims to understand natural language (Natural Language Understanding), which will help the bot understand what the user is saying. And Rasa Core is a dialogue manager. For example, when the bot knows what the user has said thanks to the NLU, Rasa Core is in charge of knowing what to respond or how to act, thanks to Machine Learning techniques. Again, both applications are based on Machine Learning, so the system learns from the answers provided by the user, not being a classic bot that follows a specific path or always the same (Rasa, 2022b).

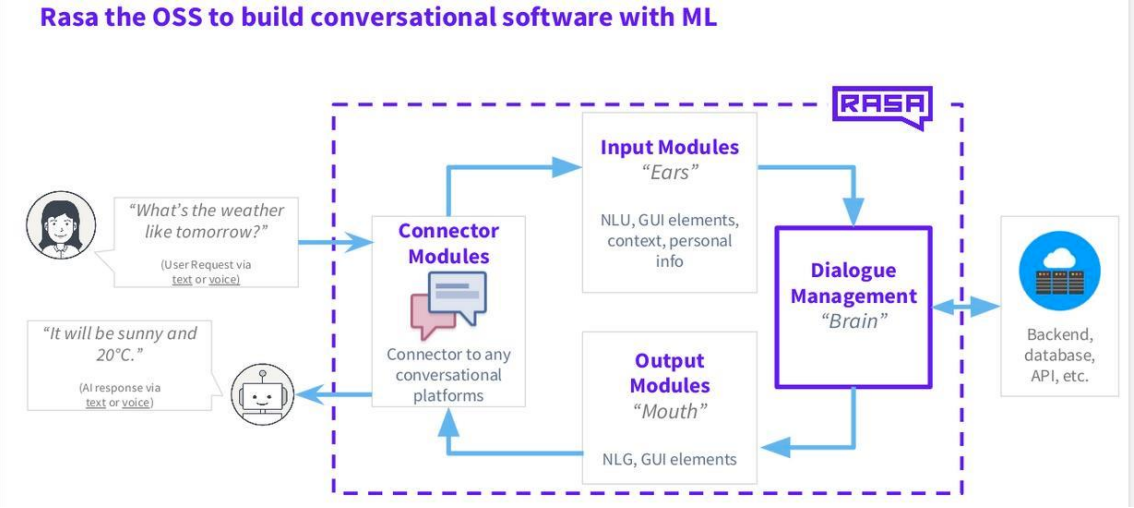


Figure 9 Rasa OSS conversation

RSA NLU: Natural language understanding

Rasa NLU is a library for natural language understanding (NLU) which is a subset of NLP. It takes care of intent classification and extracts the entity from the user input. The difference between NLU and NLP is that natural language understanding understands what the user is saying and the processing is more concerned with structuring the language into standardised form (*Better Intent Classification And Entity Extraction with DIETClassifier Pipeline Optimization*, 2021).

Rasa Core: Dialogue management

Rasa Core is a chatbot framework with machine learning-based dialogue management which takes the structured input from the NLU and predicts the next best action.

Rasa X

Rasa X is a complementary tool to Rasa and although its use is optional, it allows listening to users and using those insights to improve AI assistant.

Tokenizer

This module allows getting a list of tokens from a sentence. Here is an example of how it treats a phrase (*Better Intent Classification And Entity Extraction with DIETClassifier Pipeline Optimization*, 2021).

Please open the security door 4 -> [Please, open, the, security, door, 4]

Featurizer

Machine learning is about statistics, statistics work with numbers.

At this point we are transforming the words into meaningful numbers (or vectors). An example of how this works is attached below. (Bunk et al., 2020)

[Please, open, the, security, door, 4] -> [[1,0,...,1,0], [0,0,...,1,0], [1,1,...,0,0], [1,1,...,1,1], [1,0,...,0,1], [0,0,...,0,0]]

DIETClassifier

Dual Intention Entity Transformer (DIET) is used for intent classification and entity recognition (Saini, 2021).

Inputs: Needs, dense_features and/or sparse_features and optionally the intent

Outputs: entities, intent and intent_classification

CRF (Conditional Random Field)

CRF (conditional random field) is an add-on to TensorFlow that allows classifying which tokens belong to certain entities (Ong, 2020).

Installation of Rasa

Now we need to get our project running, in order to do that, you need to follow the following steps:

Step 1: Pre-requirements

Before starting the installation of Rasa, the following pre-requirements must be considered:

- Install Python: Requires Python 3.7 or 3.8. We use 3.8.10
- Install Visual Studio Build Tools: Download the VS Build Tools and select Desktop development with C++ and check the optional option "C++/CLI support"

Step 2: Create a virtual environment

python -m venv [Virtual Environment Name]
example: python -m venv rasa_venv

Step 3: Upgrade pip

pip install --upgrade --user pip

Step 4: Activate virtual environment

Windows

.\[Virtual Environment Name]\Scripts\activate
example: .\rasa_venv\Scripts\activate

Ubuntu or macOS

source [Virtual Environment Name]/bin/activate
example: source rasa_venv/bin/activate

Deactivate the virtual environment

deactivate

Step 5: Install Rasa Open Source

If you deactivate it, activate the virtual environment

Install Rasa Open Source

pip install rasa

Create a new project

rasa init

Step 6: Install Rasa X

If you deactivate it, activate the virtual environment

Install Rasa X

pip install rasa-x --extra-index-url https://pypi.rasa.com/simple

Start Rasa X

rasa x

Rasa version

We used the following versions for our project. If you have trouble during installation, please take a look into the challenges chapter.

Rasa Version : 2.8.12
Minimum Compatible Version: 2.8.9
Rasa SDK Version : 2.8.2
Rasa X Version : 0.40.0
Python Version : 3.8.0

Installation of OPC UA

Next, you need to install and connect to the atvise Server mentioned in the architecture section via OPC UA. OPC is an acronym for Open Platform Communications and UA is an acronym for Unified Architecture. It is one of the most important communication protocols for Industry 4.0. (*Unified Architecture*, 2019.)

In our case we will use the library called Python FreeOpcUa, as its name suggests it uses python and has Lesser General Public License (LGPL).

Opc Ua is necessary as it enables communication between the machine and an external system. In order to use it we need to have two libraries installed and also have a client, in this case UaExpert, below you will find how to have both. (*OPC Unified Architecture*, 2022)

Step 1: Install OPC UAlibrary

Install opcua-asyncio (*Opcua-asyncio*, 2015)

pip install asyncua

Install python-opcua (*Python-opcua*, 2015)

pip install opcua

Step 2: Install UaExpert (Rocket Systems, 2018)

2.1. Go to webpage and register for download

https://www.unified-automation.com/downloads.html

2.2. Click “OPC UA Clients”

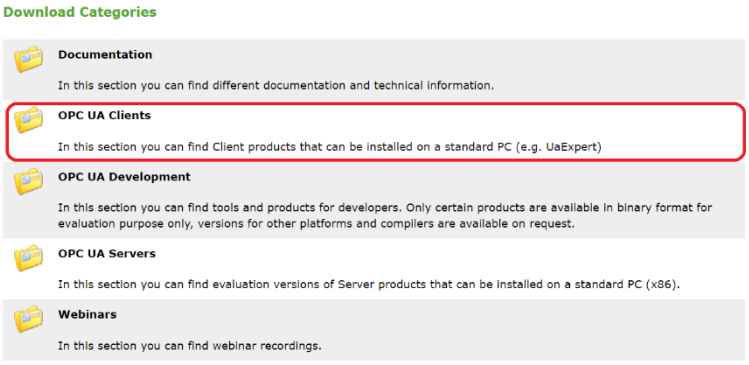


Figure 10 OPC UA Clients

2.3. Click “UaExpert”

Figure 11 UaExpert

2.4. Download file for Windows, Unzip downloaded file and install

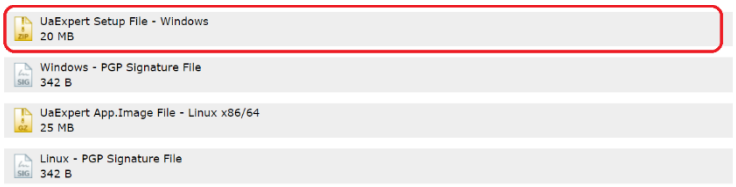


Figure 12 Download UaExpert for Windows

Step 3: Connect to UaExpert

3.1. Open UaExpert

3.2. Click “+” button

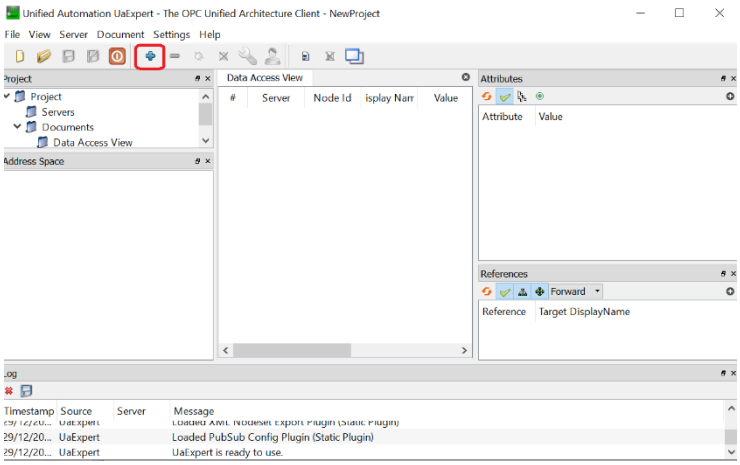


Figure 13 Add server

3.3. Configure the server

Input Configuration Name and Server url

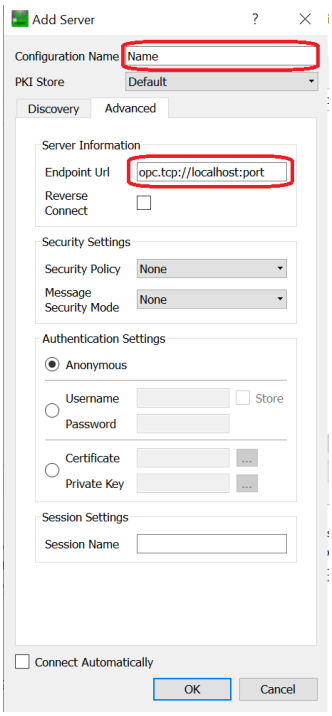


Figure 14 Configure the server

3.4. Click “OK” button

3.5. Click “Connect” button of generated server

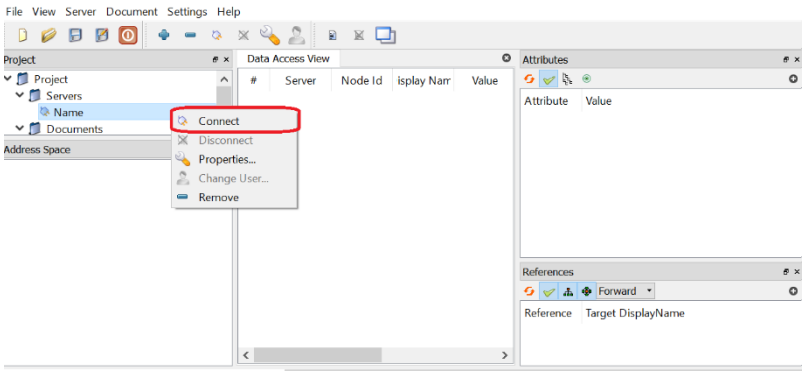


Figure 15 Connect to UaExpert

Once you have finished these steps you should be connected to the server, you should have something similar to the image.

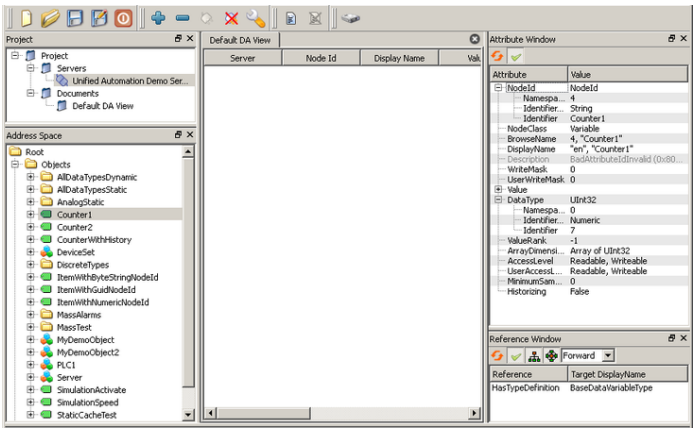


Figure 16 Server connected

You can change the values manually, just choose the variable you want to change and look for its status. To see better the data of a variable you can drag it to Data Access View, here is an example.

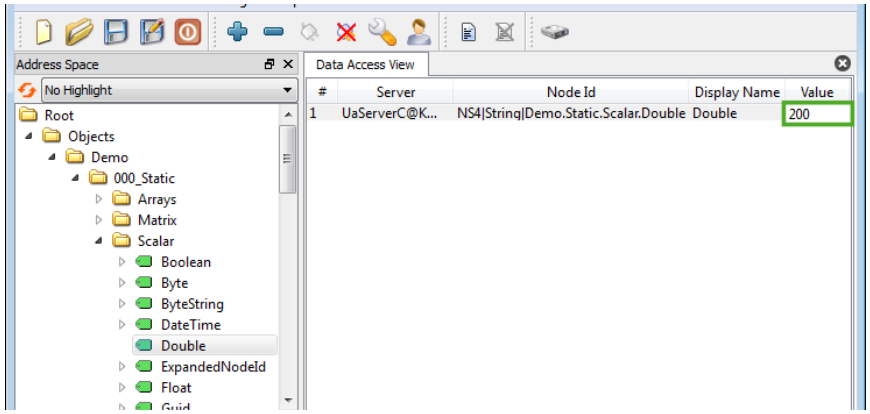


Figure 17 Change value on the server

The atvise server is running on an IPC, located in the Distributed Systems Lab of the HSA, you can connect to it using a cable or through the ip that you can find on the office computer itself…

Please note that if the server is taken offline and you will lose the connection, subsequently the prototype won't work.

Code structure

In order to better understand our program, in image X you can see the structure of our code. As you can see it is composed of several files, we will explain the functionality and importance of each of them. (Rasa, 2022c)

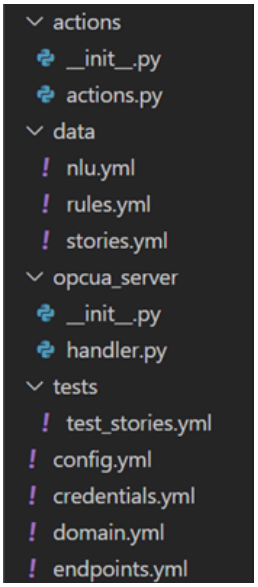


Figure 18 Code structure

- _init_.py: It is an empty file that helps Python to find its actions
- actions.py: This is made up of the custom actions and is used to communicate with the atvise server
- nlu.yml: Contains NLU training data to understand user messages. Use this file to tell the user how to interpret user messages.
- stories.yml: Dialogue training data, use this file to tweak the chatbot's dialogues.
- handler.py: This file is responsible for managing external alarms coming from the atvise server.
- test_stories.yml: This file contains tests to evaluate that your bot behaves as expected. In this project tests were done manually and the file is empty.
- config.yml: For pipeline and policy configuration.
- credentials.yml: This is used to make the assistant available on messaging platforms.

- domain.yml: In the environment from where the wizard works. It includes the answers to the user, the name of the personalised actions, the entities, intentions and slots.
- endpoints.yml: This file contains the different endpoints your bot can use.

Training and testing of your first model

Now it's time to create your actual bot. In this case, this means training a model using your NLU data and stories, saving it and then testing it.

To train the model

This step is only necessary if your code has been modified since the last time you ran it.

rasa train

To test the model

Execute rasa actions are only necessary if you have custom actions if so please note that the following commands must be executed on different terminals.

rasa run actions

When the action endpoint is up and running. This command will open a browser window, where you can chat with your newly created chatbot.

rasa x

Challenges

Throughout this work we have encountered different challenges, some of which consisted of overcoming problems in the working environment. This section lists the main challenges we have had to face in order to help future users. If you have any other challenges you can consult the Rasa forum.

Error	**Solutions**
Tensorflow error	Run *pip install tensoflow==2.1.1*
No module named pip	Run *python -m ensurepip*
Error install specific versions of rasa x	Run *pip install rasa-x==0.32.2 --extra-index-url https://pypi.rasa.com/simple*

Rasa X Installation failed by "Can't locate revision identified by" error	Create new environment and then do the experiment apart from running environment conda create -n “Environment_Name” python=3.8 conda activate Environment_Name pip install pip==20.2 --user pip install rasa==2.6.2 pip install rasa-sdk==0.40.0 --extra-index-url https://pypi.rasa.com/simple rasa init, rasa train and rasa shell (this is for creating new rasa project and training of rasa shell) *rasa x* (running rasa x)
ImportError: cannot import name ‘RowProxy’ from ‘sqlalchemy.engine’ Install	Run *pip install SQLAlchemy==1.3.22*
Sorry, something went wrong (see error above). Make sure to start Rasa X with valid data and valid domain and config files. Please, also check any warnings that popped up.	Try to resolve all warnings
Version conflict	Review the warnings you had during installation. If it doesn't work, try to have the same versions as mentioned above.
Rasa X isn’t opening the ui	Delete the events*-files + rasa.db file
OSError: [Error 10048] error while attempting to bind on address ('0.0.0.0.0', 5005): only one use of each socket address	Run *rasa run actions* with another port, for example *rasa run actions -p 50055*

(protocol/network address/port) is allowed	
SpaCy Error	Run *python -m spacy download en_core_web_sm*
Rasa DIET architecture is very obscure in the aspect of describing how DIET processes new data outside the learning cycle	Research how similar models accomplish this and consulting of the rasa community
Model reacts with seemingly random action which has nothing to do with the stories you defined.	**First possible cause:** • If you work with entities and Rasa can't detect the entity correctly, it shows unexpected behaviour. This can be debugged by checking if entities are shown correctly visualised in Rasa X. E.g. the NLU was not detecting the following entity: - intent: ask_component_location_info examples: | - Where is component [-Z2.3z2](component)? We fixed this by changing the *WhiteSpaceTokenizer* to a different one in the settings. **Second possible cause:** • Use of entities on the *Rasa Actions server* in this fashion: next(tracker.get_latest_entity_values("door_number"), None) Let's assume we have the following story: story: Warn user of an external cylinder alarm steps: - intent: EXTERNAL_warn_cylinder_alarm entities: - cylinder_with_alarm - action: utter_warn_cylinder_alarm - intent: ask_cylinder_with_alarm_location_info - action: action_utter_supply_alarm_cylinder_location_info You can see that the entity cylinder_with_alarm is recognized in the first intent. If we use *next(* in both actions the second use of *next(*

	will find a null value, the iterator will have been pushed past the entity value. To fix this you could use slots: component_name = tracker.get_slot('cylinder_with_alarm')
Rasa Action is not triggered even though it was defined in the story as the next action and the story is identified correctly.	Rasa Actions need to be defined with the word **action** in front. Example which won't work: - story : ask for session id steps: - intent: ask_conversationID - action: rasa_actions-utter_id Fix: - story : ask for session id steps: - intent: ask_conversationID - action: action_utter_id

Use Cases

The use cases show the possible conversations that the user and the smart assistant can have. In our case we have divided these use cases into three groups, depending on their function.

Below you will find all the use cases that were implemented, these are named in the same way as in the stories.yml file.

Use case one: Simple State Requests

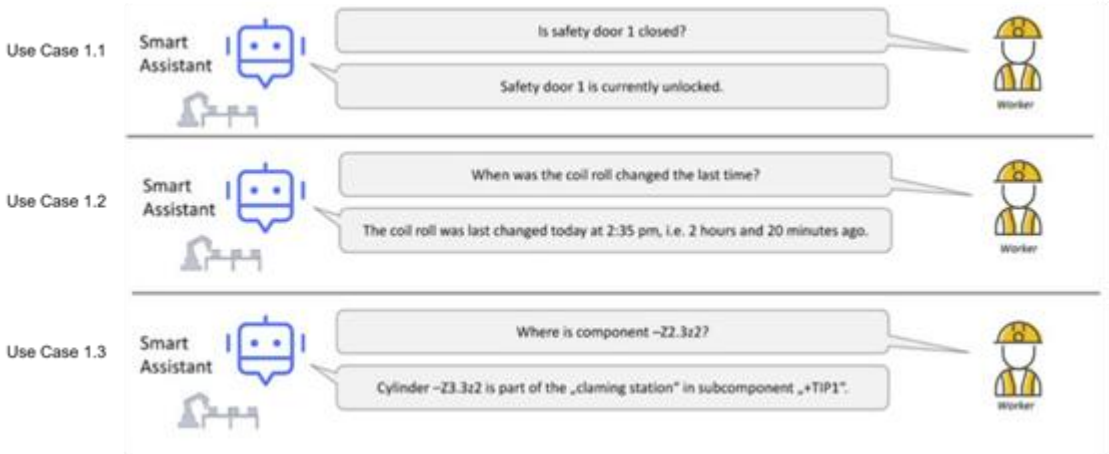


Figure 19 Use Cases 1.1, 1.2 and 1.3

- Use Case 1.1: check if the safety door is open or closed

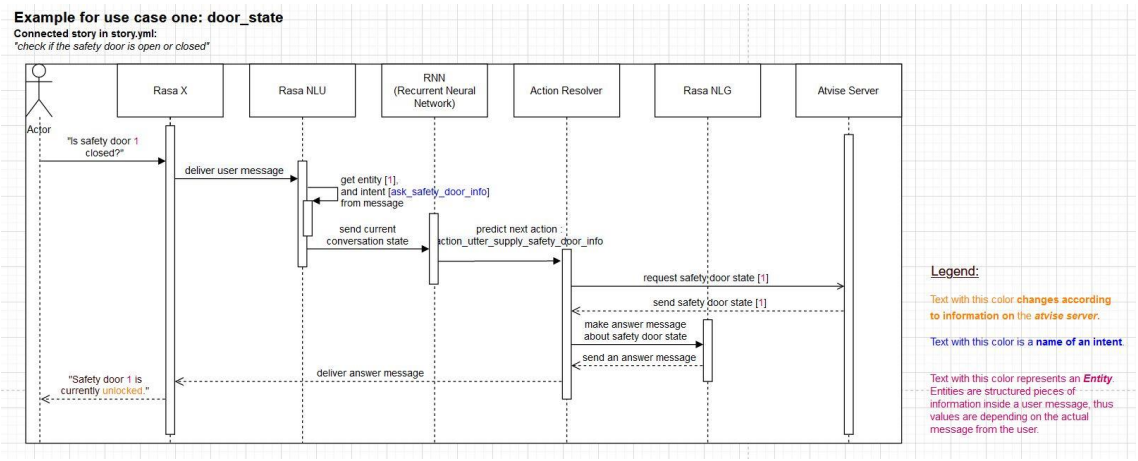


Figure 20 Sequential Diagram Use Case 1.1

In this example a worker wants to know the state of a safety door on the machine. They input their command through Rasa X which is a chat UI already integrated into Rasa. The different components integrated into Rasa then take over and decipher the user's intent and extract the exact door number. Next, Rasa Actions contacts the atvise server to find out about the specified safety door state. Finally, the answer is passed back to the worker.

- Use Case 1.2: Supply user with the datetime when component x was changed the last time

This example follows a similar sequence as in use case example 1.1, but in this case the worker wants to know when a component was last modified..

- Use Case 1.3: Supply user with the location of the component

As in the previous example, this use case follows a similar sequence to use case 1.1, however in this case the worker wants to know the location of a component.

Use case two: Proactive communication, optional continuations and processing contextual information

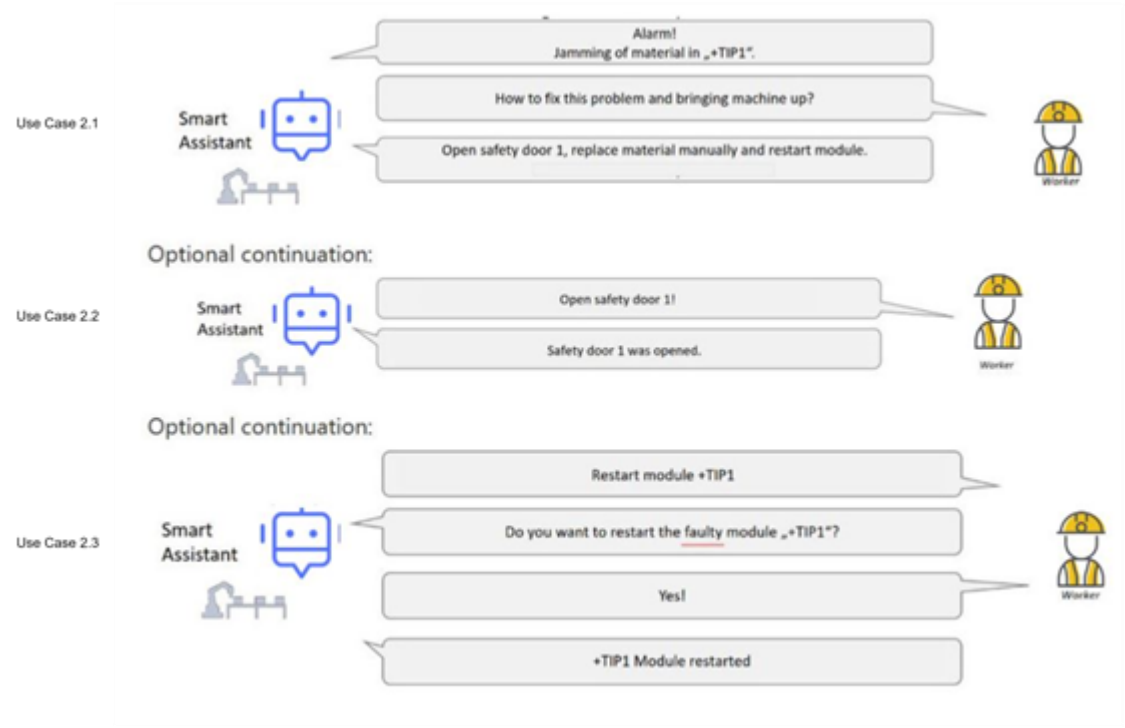


Figure 21 Use Cases 2.1, 2.2 and 2.3

- Use Case 2.1: Warn user of an external jamming material alarm

This use case is very similar to use case 3 (see Figure 23). The user is informed that a jamming material alarm has been activated. When the worker asks how to fix it, the smart assistant accesses the atvise server to find out the concrete solution for that alarm.

(The following Use Cases are optional and not part of one single story, as the worker could or could not follow the advice given by the smart assistant.)

- Use Case 2.2: Open the safety door

In this example, the user will request the smart assistant to open a door. First Rasa will check the status of the door, as explained in use case 1.1. If the door is already open, the worker will be informed that the action cannot be carried out, but if the door is already closed, the status of the door will be changed on the atvise server. So the smart assistant's answer depends on the environment, the context of the situation.

- Use Case 2.3: check faulty module and restart

This use case acts very similar to use case 2.2. The worker will request the smart assistant to restart a particular module that appears to be faulty. The smart assistant then checks if the module is faulty at all. If it is, it provides the user with the option to restart it.

The restart is achieved by changing a value on the atvise server.

So the smart assistant's answer again depends on the environment, the context of the situation.

Use case three: Proactive communication and processing contextual information

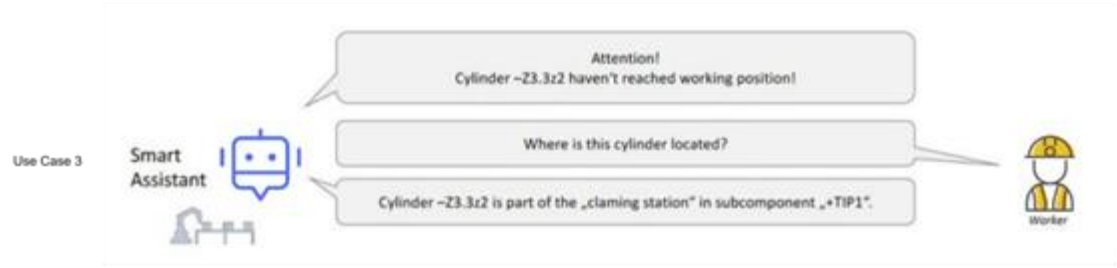


Figure 22 Use Case 3

- Use Case 3: Warn user of an external cylinder alarm

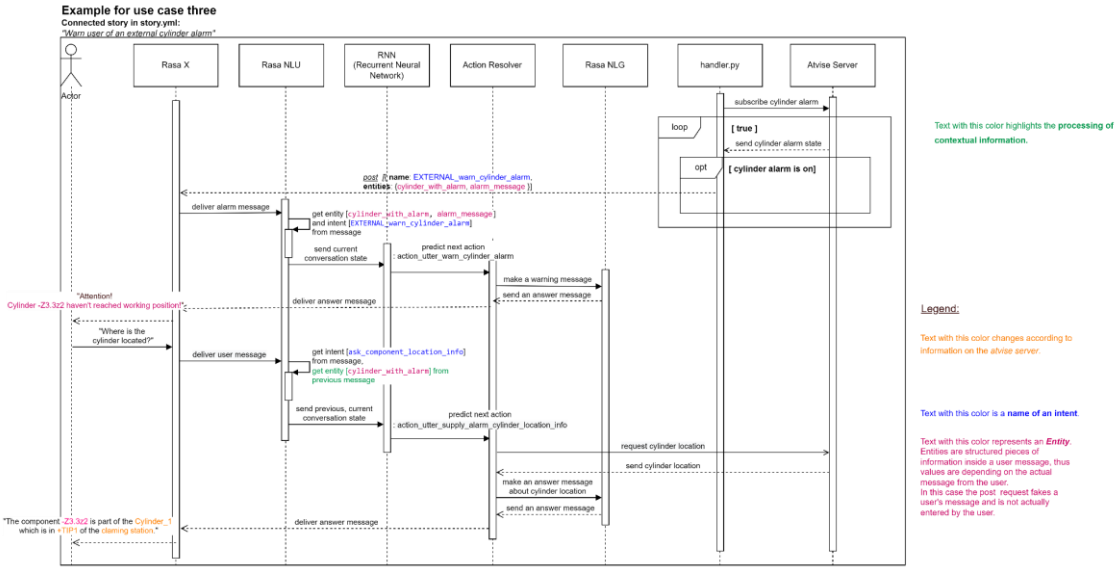


Figure 23 Sequential Diagram Use Case 3

In this example, the worker will be informed of the activation of the cylinder alarm. Code in Handler.py, which was originally started during initialization of the chatbot through Rasa Actions, continuously pulls the alarm status from the atvise server.

If at any time the alarm is activated, Rasa will be informed, and it will send a message informing the user of this fact. The user can then be informed of the exact location of the alarm, for this

the Rasas actions will contact the atvise server, which will inform the user of the location of the specific cylinder alarm. Finally, the worker will get as a response the location of the alarm.

Note, that the worker doesn't ask for the specific cylinder location, but the smart assistant remembers the context of the conversation and knows which cylinder the worker is talking about.

Evaluation and possible future development

This project was a success, as the main objective of this work was achieved, which was to enable communication between a machine and a worker, in this case, by means of messages written in English. The user can consult the status of certain elements, such as doors or cylinders, but can also be informed when an alarm is triggered and how to solve the problem. At this point it should be noted that this prototype hasn't been tested on Hekuma's industrial machines.

In addition to testing its effectiveness on a real machine, this prototype can be further developed to improve its functionality. Currently, the prototype is limited by the use cases mentioned above, so a first improvement would be to implement new stories and also to extend the possible solutions to the material jamming alarm, as currently only the one shown in use case 2.1, is available.

Another possible evolution would be to implement voice so that both the intelligent and the operator can communicate without the need to write or read. Research into that direction was conducted and documents concerning this matter are in a Nextcloud folder.

Other languages could also be added, as already discussed, our bot only communicates in English, we recommend having an NLU for each desired language, as shown in the image.

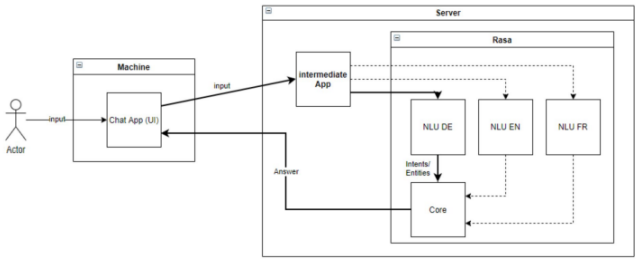


Figure 24 Model multilinguality

References

B, S. (2020, December 8). *A Brief Biography Of Richard Wallace: The Mind Behind ALICE*. YakBots. https://yakbots.com/a-brief-biography-of-richard-wallace-the-mind-behind-alice/

Better Intent Classification And Entity Extraction with DIETClassifier Pipeline Optimization. (2021). Botfront. https://botfront.io/blog/better-intent-classification-and-entity-extraction-with-diet-classifier-pipeline-optimization

Bunk, T., Varshneya, D., Vlasov, V., & Nichol, A. (2020, May 11). *DIET: Lightweight Language Understanding for Dialogue Systems*. Arxiv. https://arxiv.org/pdf/2004.09936.pdf

Burns, E. (2021, March 30). *Machine Learning*. TechTarget. https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML

ChatterBot. (2021) *About ChatterBot —ChatterBot 1.0.8 documentation*. ChatterBot. https://chatterbot.readthedocs.io/en/stable/

DeepPavlov. *DeepPavlov: an open source conversational AI framework*. (2021). DeepPavlov. https://deeppavlov.ai/

Forum rasa. (n.d.). Rasa. https://forum.rasa.com/

Golem. *Prihoda/golem: Open-source chatbot framework for python developers.* (2018). GitHub. https://github.com/prihoda/golem

Infosys Limited. (2022). *KRTI 4.0 - AI Framework for Operational Excellence*. Infosys. https://www.infosys.com/services/engineering-services/service-offerings/ai-framework-industry-operations.html

Ong, R. (2020, April 10). *Day 101 of #NLP365: In-Depth Study Of RASA's DIET Architecture*. Medium. https://towardsdatascience.com/day-101-of-nlp365-in-depth-study-of-rasas-diet-architecture-3cdc10601599

OPC Unified Architecture. (2022, January 13). In *Wikipedia*. https://en.wikipedia.org/wiki/OPC_Unified_Architecture

Opcua-asyncio. (2015). GitHub. https://github.com/FreeOpcUa/opcua-asyncio

Petrock, V. (2020, November 16). *Voice Assistant and Smart Speaker Users 2020*. EMarketer. https://www.emarketer.com/content/voice-assistant-and-smart-speaker-users-2020#page-report

Python-opcua. (2015). GitHub. https://github.com/FreeOpcUa/python-opcua

Rasa. (2022a, January 14). *Introduction to Rasa Open Source*. https://rasa.com/docs/rasa

Rasa. (2022b, January 17). *Rasa Architecture Overview*. https://rasa.com/docs/rasa/arch-overview

Rasa. (2022c, January 17). *Rasa Playground*. https://rasa.com/docs/rasa/playground/

Richardson, F. (2018, September 21). *The AI Paradigm Shift - Becoming Human: Artificial Intelligence Magazine*. Medium. https://becominghuman.ai/the-ai-paradigm-shift-53fa07ae3ab2

Rocket Systems. (2018, February 25). *OPC UA Ep. 1 | Introduction & Installation*. YouTube. https://www.youtube.com/watch?v=mEbPHflLNyc

Saini, M. (2021, December 15). *Using the DIET classifier for intent classification in dialogue*. Medium. https://medium.com/the-research-nest/using-the-diet-classifier-for-intent-classification-in-dialogue-489c76e62804

Siri Team. (2017, August). *Deep Learning for Siri's Voice: On-device Deep*

Mixture Density Networks for Hybrid Unit Selection Synthesis. Machine Learning. https://machinelearning.apple.com/research/siri-voices

Tock. (2021). Tock. https://doc.tock.ai/en/

Unified Architecture. (2019, September 26). OPC Foundation. https://opcfoundation.org/about/opc-technologies/opc-ua/

Weizenbaum, J. (1966, January 1). *Computational Linguistics*. ACM Digital Library. https://dl.acm.org/doi/10.1145/365153.365168