



Cognitive Assisted agile manufacturing
for a LAbor force supported
by trustworthy Artificial Intelligence

Grant: 957296

Call: H2020-ICT-2020-1

Topic: ICT-38-2020

Type: RIA

Duration: 01.10.2020 – 30.09.2023

Deliverable 2.2

Digital intelligent assistant core for manufacturing demonstrator
– version 1

Lead Beneficiary: BIBA

Type of Deliverable: Demonstrator

Dissemination Level: Public

Submission Date: 14.07.2021

Version: 1.0



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957296.

Versioning and contribution history

Version	Description	Contributions
0.1	Initial version	BIBA
0.3	Initial descriptions	BIBA
0.4	Security mechanisms for Mycroft core and Mobile App	UBI
0.7	Component descriptions added and scenario outlined	BIBA
0.9	First complete draft	BIBA
1.0	Final version	BIBA

Reviewers

Name	Organisation
Konstantinos Grevenitis	Holonix

Disclaimer

This document contains only the author's view and that the Commission is not responsible for any use that may be made of the information it contains.

Table of Contents

1	Introduction	6
1.1	Purpose and Objectives	6
1.2	Approach.....	6
1.3	Relation to other WPs and Tasks	6
1.4	Structure of Deliverable	7
2	Core infrastructure and components	8
2.1	Approach.....	8
2.2	Requirements.....	9
2.3	Infrastructure	9
2.4	Components.....	10
2.4.1	Wakeword detection.....	11
2.4.2	Speech-to-Text.....	11
2.4.3	Adapted Mycroft core	11
2.4.4	Mycroft intent parsing	11
2.4.5	Mycroft skill "Talk to Rasa"	12
2.4.6	Rasa chatbot.....	12
2.4.7	API management	12
2.4.8	Data integration.....	12
2.4.9	Adapted Android App	13
2.4.10	Text-to-Speech.....	15
2.5	Current constraints.....	15
3	Demonstration scenario.....	16
3.1	Product arrival	16
3.2	Checklist management.....	17
3.3	How to record a defect	18
3.4	Information requests.....	19
3.5	Root-cause analysis	21
4	Conclusion and Outlook	26
	Annex A	27

List of Figures

- Figure 1: Core components of the first demonstrator 10
- Figure 2: Initial design variants for the adapted Android App 13
- Figure 3: QR code reading with the adapted Android App 14
- Figure 4: Configuration options in the adapted Android App 15

List of Tables

- Table 1: Requirements fulfillment of the first demonstrator 9
- Table 2: Checklist management intents 17
- Table 3: Entity view on checklist_start_testing 18
- Table 3: Entity view on checklist_mark_as_completed 18
- Table 4: Defect identification intents 18
- Table 5: Entity view on record_new_defect 19
- Table 6: Information request intents 20
- Table 7: Entity view on get_information_about_one_product (case 1) 20
- Table 8: Entity view on get_information_about_one_product (case 2) 20
- Table 9: Entity view on get_matching_products 21
- Table 10: Root-cause analysis intents 23
- Table 11: Entity view on request_table_of_matching_items (case 1) 23
- Table 12: Entity view on request_table_of_matching_items (case 2) 24
- Table 13: Entity view on update_table_of_matching_items 24

List of Abbreviations

Abbreviation	Description
OKR	Objective Key Results
AMA	Augmented Manufacturing Analytics
NLU	Natural Language Understanding
DoA	Description of Action
CAT	Conversational Agent Team
CA	Conversational Agent
DM	Dialog Management
IP	Internet Protocol
API	Application Programming Interface
SST	Speech-to-Text
TTS	Text-to-Speech
SKU	Stock keeping unit
GDPR	General Data Protection Regulation

Executive Summary

This deliverable outlines the core infrastructure, components, and the scenario of COALA's first demonstrator. The demonstrator bases on a so-called "conversational agent team" between the voice-enabled digital assistant Mycroft and a connected chatbot built with the Rasa framework. Bot agents complement each other and make COALA's core infrastructure flexible and powerful in terms of Natural Language Understanding and Dialog Management. The first demonstrator proves the technical feasibility of COALA's core infrastructure and fulfills key requirements from WP1 partially - as planned. There are several constraints, such as single-language support and highly constrained usability that follow-up deliverables will address. The demo uses the white goods use case and Augmented Manufacturing Analytics as the first demonstration scenario. For this purpose, we refined the example dialogs from WP1 into small interaction sequences, identified intents, and entities, and outlined the responses. As planned, this demonstrator does not connect to a database yet.

1 Introduction

1.1 Purpose and Objectives

This deliverable is complementary to the demonstrator "Digital intelligent assistant core for manufacturing demonstrator – version 1" built-in Task 2.2. The demonstrator is the first replicable end-to-end realization of the COALA infrastructure. It is replicable because we prepared all code repositories and a setup script so that all COALA partners can install their core infrastructure instance. End-to-end means the user can interact with the assistant by voice. BIBA implemented the demonstrator in its production lab environment to test the essential functions and identify early constraints (e.g., noise and connectivity).

The key objectives related to this deliverable are:

- OKR 3.1: Availability of the digital assistant core demonstrator.
- OKR 3.3: Recognize 95% of the user's intents correctly (Accuracy).
- OKR 3.4: Recognize 75% of the intent context correctly (Accuracy).

1.2 Approach

The demonstrator's first version serves as basis for **continuous evaluation and improvement**. It is a lab demonstrator, which means that it operates in a model environment. It uses realistic scenarios to allow the COALA partners early trial-and-error experiments and systematic tests (e.g., noise handling and language recognition accuracy). This demonstrator will focus on core functions, i.e., functions relevant for most manufacturing companies, such as information about capabilities, creators, license, and conversation navigation.

This document also describes features not implemented in the first demonstration. We provide them as an outlook for the second demonstrator in D2.5

1.3 Relation to other WPs and Tasks

The demonstrator uses results from the following tasks:

- Completed tasks T1.1 and T1.2 regarding requirements and the demo scenario.
- Parallel tasks T2.1 about the Augmented Manufacturing Analytics (AMA) serves as part of the demo scenario, task T4.1 builds the architecture, and T4.5 refines the Natural Language Understanding (NLU) capabilities of the demo and conducts experiments about language support and recognition.
- This deliverable will be an input for task T2.3 concerning the implementation of AMA functions.

1.4 Structure of Deliverable

This deliverable has two main sections. Chapter 2 focuses on the core infrastructure and the components needed for the demonstrator. It explains our "conversational agent team" approach and recaps key requirements of WP1. Then, it outlines the infrastructure (hardware) and the involved components (software). Chapter 3 summarizes the scenario we created to demonstrate the assistant's end-to-end functionality. **End-to-end** means the user interacts via voice and receives a voice response - all core components in between work.

2 Core infrastructure and components

This section introduces the approach we selected for the infrastructure, covers the related key requirements, and describes the main elements of the technical infrastructure.

2.1 Approach

The COALA DoA already outlined that the solution grounds on Open Source components whenever reasonable. We decided to use **Mycroft** as the centerpiece because it is the most promising assistant framework. It has an active, growing community¹, an active team (Mycroft AI Inc.), and covers software and hardware improvements (e.g., Mycroft App for Android, Mycroft core for Android, Mark II smart speaker). Mycroft has a *modular architecture*, which allows quick adjustments via configuration and higher reliability because users can easily replace unsuitable components (e.g., the project died, outdated code, or license changes).

A significant **downside** is that Mycroft's dialog management capabilities are less advanced. Skills that support dialogs with many turns and persistent context are hard to develop because Mycroft does not have libraries to simplify the development or a tracker store to maintain context long. Its skills typically focus on dialogs with few turns rather than more extended conversations, e.g., 10+ turns. COALA's solution needs to support long conversations, and we expect the context to be complex - besides, it needs to maintain the dialog context over several conversations.

We decided to use a **conversational agent team** (CAT) to compensate for this shortcoming. *Conversational agent* (CA) is the superordinate term for software agents which interact via natural language with their users. CAs include agents that support voice and agents that support text. Media refers to the latter as chatbots. Voice-enabled agents and chatbots use almost identical working principles. The main technical difference is that a chatbot lacks components to transcribe voice and generate voice from a text.

We define a **CAT** as the **technical integration of two or more CAs to minimize the disadvantages of the individual agents**. Integration means that all involved agents can share information, e.g., passing user messages from one agent to another. In COALA's case, we use Mycroft as the leading agent because users begin all dialogs through it. The second agent is a chatbot - both exchange text messages, but only Mycroft responds to users directly.

We identified that **Rasa** is the best possible fit to address Mycroft's downsides. Rasa is an Open Source chatbot framework using the Apache 2 license. It has a large community, various demo chatbots, active developers (Rasa Inc.), and sophisticated support to develop NLU and Dialog Management (DM). Rasa uses a pipeline for NLU that can integrate various other open NLU components, such as Spacy² or Duckling³. It applies a hybrid approach for DM combining rules (reliable) with probabilistic models (flexible). Developers can train the latter based on real conversations and thus continuously improve the assistant.

1 <https://community.mycroft.ai>

2 <https://spacy.io/>

3 <https://github.com/facebook/duckling>

2.2 Requirements

This deliverable will not repeat how components fulfill or relate to specific requirements, as this is part of D4.2 (System architecture and interface specification). The most relevant requirements for this demonstrator concern usability (D1.1 section 5.2). Our system design focuses on five elements:

1. **Subjective satisfaction:** users find the system pleasant and appropriate
2. **Ease of appropriation:** users can remember easily how to use the system
3. **Reliability:** users cannot make too many or too severe errors and can recover easily
4. **Ease of learning:** new users can achieve their goals easily
5. **Efficiency:** users can perform tasks quickly through an easy process

The elements above are not in the first demonstrator's focus, but our future improvements specifically target them (see OKR's above) and evaluate them. Table 1 summarizes the usability requirements and how this demonstrator fulfills them.

Table 1: Requirements fulfillment of the first demonstrator

Requirements	Descriptions	Fulfilment
REQ-0850	Present understandable, jargon-free messages and actions users can take	Low
REQ-0860	Limit options and show essential information	Low
REQ-0870	Follow established norms regarding the interactions with the user	Medium
REQ-0880	Avoid disruptions (e.g. forced logins)	Low
REQ-0890	Make very simple forms easy to complete (and only if necessary)	Medium
REQ-0900	Include warnings and autocorrect features to minimize errors	Low
REQ-0910	Offer efficient and easy-to-use help functionality	Low
REQ-0920	Ensure a good level of reactivity	Medium
REQ-0930	Ensure a good visibility of the system status	Low

For the infrastructure, we have to consider the following newly identified **factory-specific requirements**:

- Headset in one factory: The user must be *able to hear* the surrounding
- Headset in a second factory: The user must wear *ear protection*
- Mobile device constraints in a third factory: For compliance reasons, *only specific devices* are allowed company-wide.

2.3 Infrastructure

By infrastructure, we refer to the hardware needed to operate the assistant. This demo's infrastructure has three main parts: the server, the Android mobile devices, and peripherals.

Server

The entire software infrastructure for the first prototype bases on a **Linux server**, operated by BIBA. A reverse proxy exposes the Mycroft core IP address. The demo server runs **Docker** and various Docker containers that build the *COALA lab-demo container stack*.

UBI replicated the demo sever and already uses the adapted Mycroft core (see below), which secures the communication with valid certificates. Their implementation uses Keycloak for access control.

Android mobile device(s)

In COALA, users interact with Mycroft via mobile devices. We focus on the existing Apps provided by the Mycroft community. The **Mycroft App for Android** (Android App) provides voice and graphical interfaces. It uses a websocket connection to the Mycroft backend to exchange messages. **Mycroft core for Android** is the complete core software deployable on Android devices. It is experimental but has the advantage that users can interact with Mycroft without connectivity - they cannot use Internet-based skills. The implementation focus is on the Android App. We use *Samsung Galaxy Tab S5e* tablets to test the assistant.

Peripherals

Noise can reduce the transcription quality significantly, and it makes hearing the assistant's responses difficult. Therefore, we use an industry-grade wireless headset that filters noise when users speak to the assistant via the Android App. We use a **headset inlay** compatible with noise protection (e.g., ear shells) and a **separate headset** allowing the users to listen to the assistant and the environment.

2.4 Components

Figure 1 illustrates the components of the first COALA demonstrator. It covers the devices, the two coupled backends, and the API management to manage queries to various data sources.

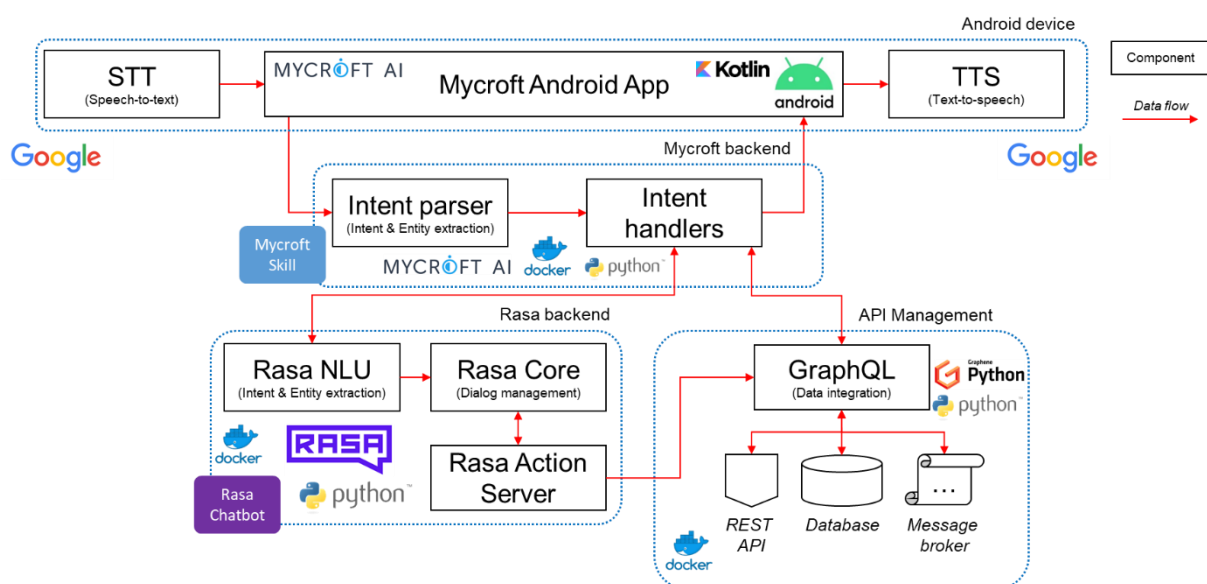


Figure 1: Core components of the first demonstrator

The organization of the following sub-chapters grounds on a typical execution sequence for user-initiated conversations, i.e., when the user activates the assistant and asks questions.

2.4.1 Wakeword detection

The wakeword detection is a speech recognition system that activates the assistant. Mycroft uses a combination of *PocketSphinx* and *Precise*. Wakeword detection is essential to use the assistant hands-free. However, the Mycroft App requires wakeword detection on the mobile device to work. The App does not support wakeword detection, but COALA will extend it to support hands-free interactions too. There are three approaches: first, we integrate a wakeword detection service in the App; second, we install a wakeword detection service on the mobile device and link it to the App; third, we operate a native Mycroft core on a mobile device.

Due to time constraints, we decided to postpone implementing the wakeword detection in the first demonstrator. The user activates the assistant with a button.

2.4.2 Speech-to-Text

Speech-to-Text (STT) uses a trained language model to transcribe audio utterances to text. The first demonstrator version (D2.2) uses the *Google STT* service available on most Android mobile devices. It is not open but reliable and the easiest to implement. Privacy concerns exist because Google processes the audio in its cloud. We plan to replace Google STT with an Open Source solution, such as DeepSpeech. The specific decision will consider the reliability of the STT transcriptions.

2.4.3 Adapted Mycroft core

The public IP address of the Mycroft core will be accessible through a reverse proxy. This proxy has been configured to support secure connections based on valid certifications managed by the Security Mechanisms of the COALA Solution. Besides, Mycroft core has been integrated with Keycloak⁴ to provide access for authorized users only. In this direction, we also extended the Mycroft App app to support a valid token exchange with the Keycloak authorization mechanism. The App receives an authorization token from Keycloak and adds it as a header in every request it makes to Mycroft core. Then, Mycroft core accepts requests performed by validated users checking their credentials.

The security features will be implemented in the second demonstrator version in D2.5.

2.4.4 Mycroft intent parsing

The intent parser is part of Mycroft's NLU and identifies the user's intent and entities (i.e., named values) from the transcribed user utterance. Mycroft tries to match an intent with a suitable skill and passes over the identified entities to that skill.

Mycroft offers the Adapt parser and the Padatious parser. The former uses keywords, and the latter uses a neural network and is more experimental. We use both but focus on the Adapt parser. Complex dialogs will be handled by Rasa as described in Section 2.1.

⁴ <https://www.keycloak.org/>

2.4.5 Mycroft skill "Talk to Rasa"

Mycroft uses skills to fulfill user intents. Typically, users activate skills and then talk to the assistant. Skills execute code and access external data sources if required. We use a customized skill called "Talk to Rasa" to exchange messages between Mycroft and a Rasa chatbot. For this purpose, we modified an existing skill template.⁵ The changes allow the COALA solution to differentiate content as voice (speak), text (silent), and images (silent) - this allows us to support multi-modal interactions.

2.4.6 Rasa chatbot

Rasa is a chatbot framework using a configurable NLU pipeline, a dialog manager based on rules and probabilistic models, and a fulfillment server performing custom actions via Python code. The action server queries information from GraphQL and uses the results to decide how the chatbot responds (e.g., using a template with parameters filled by the query result).

We use Rasa for the major part of the NLU design and dialog modeling. The Rasa chatbot contains two parts. First, the **base features** contain universally applicable intents and conversation patterns. Second, the **demo features** contain specific training data, domain descriptions, and custom actions. The demo actions will later interact with the other COALA components, such as PREVENTION, Cognitive Advisor, and the WHY-Engine, via the API management component. The demo features are the focus of the demonstration scenario in Section 3.

2.4.7 API management

An assistant's utility largely depends on its access to data sources/channels. Therefore, we decided to use an API management called **GraphQL**⁶ to simplify this access. GraphQL runs on a server and provides a single query language to access all connected data sources. We use a Python GraphQL client called **Graphene**⁷ to build schemas/types and connect Rasa with the GraphQL server.

The first demonstrator (D2.2) will not use the GraphQL server since there are no databases to connect to at this point. D2.5 will contain this feature.

2.4.8 Data integration

The API management solution has access to BIBA's SEMed⁸. SEMed semantically integrates heterogeneous data sources (i.e., virtual data integration). It is more flexible than GraphQL but not as performant and, therefore, experimental.

The first demonstrator (D2.2) will use SEMed, since it does not connect to any database yet.

5 <https://github.com/jamesmf/skill-rasa-chat>

6 <https://graphql.org/>

7 <https://github.com/graphql-python/graphene>

8 <https://www.fiware4industry.com/portfolio/semantic-mediator-semmed/>

2.4.9 Adapted Android App

We extend the existing Android App in several areas. First, there is a COALA design with the color palette and an icon set. BIBA created several designs and selected two for further testing and user experience improvements. Figure 2 illustrates the current designs.

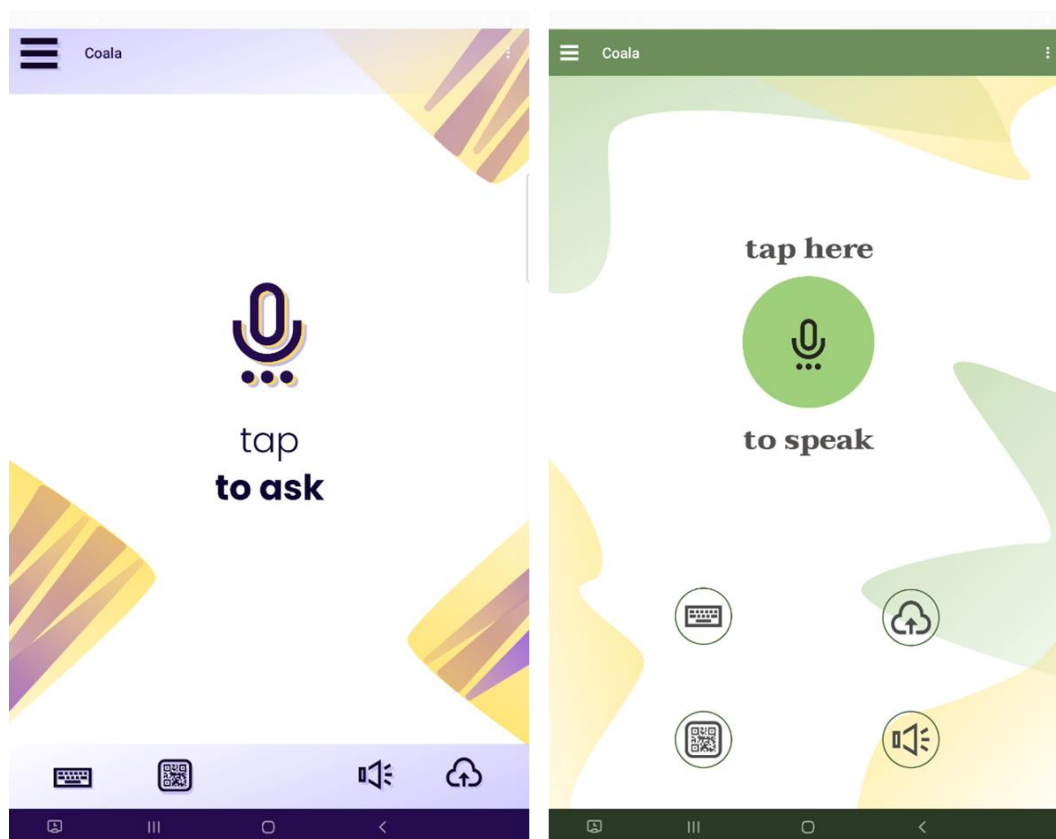


Figure 2: Initial design variants for the adapted Android App

We will rework these designs iteratively based on user feedback and user studies. If needed, additional designs are possible to better align with client branding and the exploitation.

Second, we implement several functions needed to support the various user stories identified in WP1. The following paragraphs outline the most notable changes of the Android App for D2.2.

A **QR-code reader** to capture product identifiers / serial numbers. This feature is critical because experiments with the transcription of spoken alphanumeric input indicated a high error rate, i.e., the transcription mixes up numbers or letters. The problem is mainly relevant when users identify products by serial number or a stock-keeping unit (SKU) identifier.

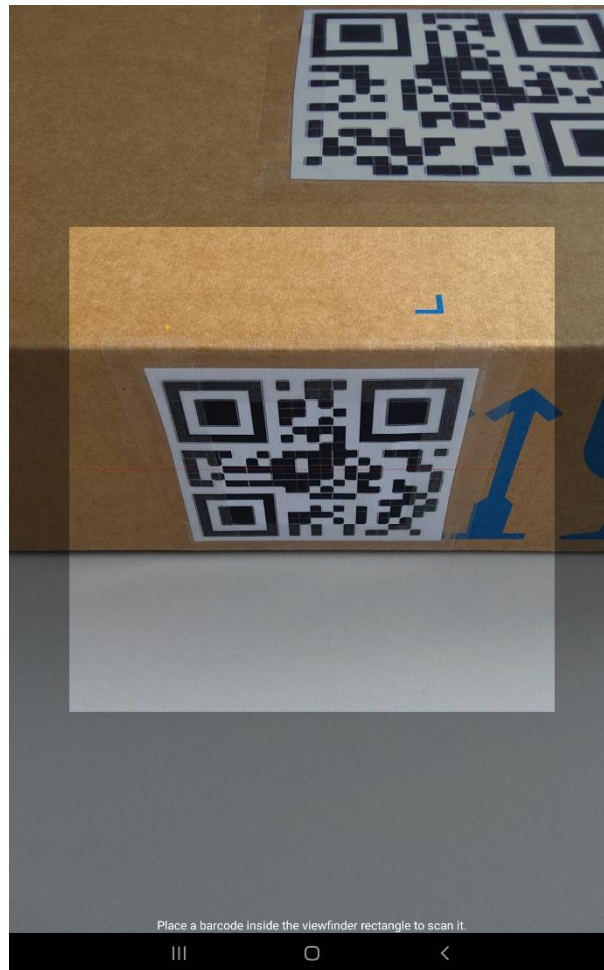


Figure 3: QR code reading with the adapted Android App

An experimental **photo-taking and upload** function. This function will later allow users to send photos during a conversation, which is necessary to document specific tasks or situations. BIBA's NextCloud server stores the photos and the GraphQL server as access to it via an API.

Extended configuration. New functions need their configuration interfaces. This includes the configuration for the backend connection and cloud storage for photos. Figure 4 shows the configuration section for image uploads where users must provide the cloud storage service and credentials to connect to it.

Figure 4: Configuration options in the adapted Android App

Security. We added an authentication mechanism (see Adapted Mycroft core) to improve the App's security. This feature is not available in the first demo, but there is a prototype implementation already, and D2.5 will contain this feature.

2.4.10 Text-to-Speech

Text-to-Speech (TTS) uses a trained language model to generate audio from text. The first demonstrator version (D2.2) uses the *Google TTS* service available on most Android mobile devices. It is not open but reliable and the easiest to implement. Privacy concerns exist because Google may process the text in its cloud.

2.5 Current constraints

The first demonstrator has several constraints that we will address in upcoming versions, such as D2.5, D2.7, and D3.5. Amongst the top-level constraints are:

- The Mycroft App supports user-initiated conversations only (no proactive behavior)
- The Mycroft App does not have a wake word activation (no hands-free interaction)
- The Mycroft App uses Google STT, which is closed but reliable (reliability vs. openness)
- The Mycroft Core and Mycroft App are not communicating securely (no auth-token)
- Multi-modal interactions are mainly missing (e.g., images not visualized)
- The Mycroft App can take photos, but users cannot refer to them in a conversation
- The Mycroft App cannot take videos usable in a conversation, e.g., as attachment.
- Only English language supported (Italian and Dutch missing)

3 Demonstration scenario

This section outlines the demonstration scenario. It consists of user stories aligning with the **white goods use case** (Augmented Manufacturing Analytics) described in deliverable D1.3. We focused the demo on the **most relevant scenarios** and implemented **key functionalities** for them. Each scenario begins with the identified example dialog. The order of the stories roughly follows the workflow in the white goods test laboratory. It has the following steps:

1. Product arrival
2. Checklist management
3. How to record a defect
4. Information requests
5. Root-cause analysis

The following sections contain **dialog notions** for so-called happy paths. A **happy path** is a conversation without breakdowns or other obstacles. We will consistently implement these dialogs first and then incrementally add *unhappy paths* and mechanisms to resolve problems. The more features we implement, the more unhappy paths we need to identify and address to maintain a good user experience. In general, the bulk of the evaluation and improvement will be dedicated to these activities.

The dialog notion uses the following **formats**:

- Users (U)
- **Coala assistant (C)**
- *<an action performed by user or assistant without dialog>*
- A regular dialog text
- [entity to be extracted]

The assistant can recognize the same or semantically similar user utterances - i.e., inputs do not have to be the same as indicated in the following sections. Variations in wording may result in misunderstanding (conversational breakdown). Future assistant versions will include fallbacks to repair these breakdowns.

3.1 Product arrival

The starting point for the demo is the arrival of a new product that needs to undergo a quality test. Each product is uniquely identifiable by its Stock Keeping Unit or Serial Number. Besides, it has a type name and number. Types are relevant if the quality tester has to assess defects because they might affect all products of the same type since they share the same product structure (e.g., components).

Happy path: bot-initiated conversation + scanner

<start>

C: New product arriving. Please scan its code.

U: <scans QR code 4898643HKJGD>



C: Got it. The last 4 digits are KJGD.

U: <silent>

<end>

Happy path: user-initiated conversation + scanner

<start>

U: <scans QR code 4898643HKJGD > here is the new product



C: Got it. The last 4 digits are KJGD.

U: <silent>

<end>

The first demonstrator does not support bot-initiated conversations as this requires "push" events from other components triggering the assistant. Also, we need to identify how Mycroft and Rasa interact in such events - an additional extension of the "Talk to Rasa" skill might be necessary.

3.2 Checklist management

Checklists are an important feature because much of the work in white goods use case follows specific workflows. Besides, checklists are one of the most common and easy-to-use tools in quality management. In order to start testing a new product, the assistant needs the SKU or a serial number (as described in 3.1) and the identification number of the checklist.

Table 2: Checklist management intents

Names	Entities	Descriptions
checklist_start_testing	checklist_id product_identifier	Starts testing process with a specific checklist
checklist_mark_as_completed	checklist_id	Marks the checklist as completed

Example dialog parts with entities:

Input 1: start the testing process for a particular product with a specific checklist

U: let's start with [1](checklist_id) checklist for product [4898643HKJGD](product_identifier)

Table 3: Entity view on checklist_start_testing

Names	Values
checklist_id	"1"
product_identifier	From past turn

Response

C: Got it. The serial number is {product_identifier}, the checklist ID is {checklist_id}.

Input 2: finish execution of a checklist after testing is done

U: please note this checklist [9](checklist_id) as completed

Table 4: Entity view on checklist_mark_as_completed

Names	Values
checklist_id	"1"

Response

C: The checklist {checklist_id} is marked as completed.

3.3 How to record a defect

If the quality tester identifies a defect, they must record that defect. This report has a defined structure, and our assistant can use Rasa's form actions to collect this information reliably.

Table 5: Defect identification intents

Names	Entities	Descriptions
record_new_defect	product_identifier component_name product_parameter_value defect_severity_value	Record defect information of an arriving product.

Example dialog parts with entities:

Input

U: There is a [scratch](product_parameter_value) on the [door](component_name). Please record that defect with severity [high](defect_severity_value).

Table 6: Entity view on record_new_defect

Names	Values
product_identifier	From past turn
product_parameter_value	"scratch"
component_name	"door"
defect_severity_value	"high"

We decided to express defect severity with words rather than letters. The transcription quality for letters turned out to be very low, resulting in many mistakes.

Response

C: The recorded defect is: product with serial number [4898643HKJGD](product_identifier), component [door](product_parameter_value), defect name is [scratch](component_name), defect severity is [high](defect_severity_value). Is that correct?

The features to upload images or videos partially exist, but users cannot use them in the conversation yet. Blocking serial numbers is not implemented yet.

3.4 Information requests

This scenario introduces so-called **views** to our assistant. A view represents a bot response with one or more data fields. We expect that this approach is a compromise between a fully flexible natural language query and a very specific intent. The former would minimize the number of intents but is challenging for NLU, and the latter increases the number of intents - though the impact on NLU is not clear at this point.

Happy path: user initiated conversation + scanner

<start>

U: I need an overview about this product <scans QR code 4898643HKJGD >



C: Product KJGD has been launched in production today at 12:00, it is a microwave, and will be delivered to the market.

U: Get me the top 3 defects

C: The top three defects are scratch, missing part, and dirt.

C: <display pareto diagram>

U: Is scratch affecting this SKU only

C: Ten units of MN 313 BL A MICROONDE AI found.

U: <silent>

<end>

Table 7: Information request intents

Names	Entities	Descriptions
request_information_about_one_product	product_identifier view_name	Request one view on one item
request_matching_products	parameter_name parameter_value	Search all items matching one parameter with value

Example dialog parts with entities and related GraphQL mock queries:

Input 1

U: ..., I need an **[overview]**(product_view_name) about this product <scans QR code **[4898643HKJGD]**(product_identifier)>

Table 8: Entity view on get_information_about_one_product (case 1)

Names	Values
product_identifier	"4898643HKJGD"
product_view_name	"overview"

GraphQL query 1

```
{
  getInformationAboutOneProduct (
    product_identifier: "4898643HKJGD "
  ) {
    launch_timestamp
    product_family_name
  }
}
```

Response 1

C: Product **[4898643HKJGD]**(product_identifier) has been launched in production **[today at 12:00]**(launch_timestamp), it is a **[microwave]**(product_family_name), and will be delivered to the market.

Input 2

U: Get me the **[top 3 defects]**(product_view_name)

Table 9: Entity view on get_information_about_one_product (case 2)

Names	Values
product_identifier	"4898643HKJGD" (from past turns)
product_view_name	"top 3 defects" (is synonym for "defect_pareto")

GraphQL query 2

```
{
  getInformationAboutOneProduct (
    product_identifier: "4898643HKJGD",
    parameter_name: "defect",
    amount: "3",
    parameter_value: "top"
  ){
    defect
  }
}
```

Response 2

C: The top three defects are **[scratch](defect)**, **[missing part](defect)**, and **[dirt](defect)**.

C: <display pareto diagram>(defect)

The visualization of images is not possible in the first demonstrator. We need to adjust the Android App to support it, otherwise, images are only available via links.

Input 3

U: Is **[scratch](product_parameter_value)** affecting this SKU only

The logical answer is yes or no, but the exact match answers too and is flexible. This way we can avoid adding an additional specific response (i.e. increased maintainability).

Table 10: Entity view on get_matching_products

Names	Values
product_parameter_name	"defect" (from past turns)
product_parameter_value	"scratch"

GraphQL query 3

```
{
  getMatchingProducts (
    parameter_name: "defect",
    parameter_value: "scratch"
  ){
    product
  }
}
```

Response 3

C: **[36](number_of_products)** units of **[MN 313 BL A MICROONDE AI](product)** found.

3.5 Root-cause analysis

This scenario introduces **tables** as a content type. We use a *last_table* slot so Rasa remembers the parameters of the last table, if users want to update it (recreate the table with different filters). The *last_table* slot contains the slot-value pairs used for the last table request.

An alternative solution to get this information is to retrieve it from the Rasa dialog tracker that stores the entire conversation history. This solution would require more parsing though.

This scenario will later **need voice navigation** features for pagination - e.g., "show next page" of a long table. The visualization of tables is not possible in the first demonstrator.

It also **extends the "view" approach** above through view parametrization. Users can utter view filter criteria to scope a view. A filtering by timeframes (e.g. last 6 hours or last 74 hours) is missing in the first demonstration.

Happy path: visualize item list + check actions

<start>

U: Are there any products with scratches?

C: There are 36 SKUs with the defect scratch. Do you want to see them?

U: Yes please.

C: Okay.

C: <visualize table>

U: <silent>

<end>

Happy path: visualize production data + adjust filter

<start>

U: Show me the production data for defective components, both produced and planned

C: 105 items produced.

C: 200 items planned.

C: <visualize table>

U: Focus on the produced items

C: 105 items produced today.

C: <visualize updated table>

U: <silent>

<end>

Table 11: Root-cause analysis intents

Names	Entities	Descriptions
request_table_of_matching_items	no_confirmation_needed item_type_identifier parameter_name parameter_value view_name view_filter	Request a table with items matching parameter(s) with value. The items belong to one item type and are identifiable by identifier (name, serial). Always utters the amount of items first. Asks user for confirmation to visualize the table unless <i>no_confirmation_needed</i> is True. The user could provide a view name and optional view filters to control which data to show.
update_table_of_matching_items	last_table no_confirmation_needed item_type_identifier parameter_name parameter_value view_name view_filter	Update the last item table using the slots from the current turn.

Example dialog parts with entities:

Input 1

U: Are there any [**products**](item_type_identifier) with the [**defect**](parameter_name) [**scratches**](parameter_value)?

The logical answer is yes or no, but the exact match answers too and is flexible. This way we can avoid adding an additional specific response (i.e. increased maintainability).

Table 12: Entity view on request_table_of_matching_items (case 1)

Names	Values
no_confirmation_needed	False (default)
item_type_identifier	"product"
parameter_name	"defect"
parameter_value	"scratch"
view_name	None (not provided)
view_filter	None (not provided)

GraphQL query 1

```
{ getMatchingItems (
  item_type_identifier:" product",
  parameter_name: "defect",
  parameter_value: "scratch"
){
  item
}
```

The *item_type_identifier* "product" is an abstract category for *produced* items (i.e. not spare parts or other supplies).

Response 1

C: There are [**36**](number_of_items) SKUs with [**defect**](parameter_name) [**scratch**](parameter_value). Do you want to see a list?

Input 2

U: **[Show]**(no_confirmation_needed) me the **[production data]**(view_name) for **[defective]**(parameter_name) components, both **[produced]**(view_filter) and **[planned]**(view_filter)

Table 13: Entity view on request_table_of_matching_items (case 2)

Names	Values
no_confirmation_needed	True (derived from "show")
item_type_identifier	"product" (derived from view)
parameter_name	"defect"
parameter_value	Any (default)
view_name	"production data"
view_filter	["produced", "planned"]

GraphQL query 2

```
{
  getMatchingItems (
    item_type_identifier: "product",
    parameter_name: "defect",
    parameter_value: "scratch"
  ) {
    item ( status: ["produced", "planned"] )
    {
      start_time
      finish_time
      status
      production_line_name
    }
  }
}
```

Response 2

C: **[105]**(number_of_produced_items) **[produced]**(view_filter) items.

C: **[200]**(number_of_planned_items) **[planned]**(view_filter) items.

C: <visualize table> (items)

Input 3

U: Focus on the **[produced]**(view_filter) items

Table 14: Entity view on update_table_of_matching_items

Names	Values
no_confirmation_needed	True (from past turns)
item_type_identifier	"product" (from past turns)
parameter_name	"defect" (from past turns)
parameter_value	Any (from past turns)
view_name	"production data" (from past turns)
view_filter	["produced"]

GraphQL query 3

```
{
  getMatchingItems (
    item_type_identifier: "product",
    parameter_name: "defect",
  )
```


PUBLIC

```
parameter_value: "scratch"
){
  item ( status: "produced" )
  {
    start_time
    finish_time
    status
    production_line_name
  }
}
```

Response 3

C: <visualize updated table>

4 Conclusion and Outlook

This summary document explained what the first COALA demonstrator includes. The follow-up deliverables D2.5 (M12) will focus on:

1. adapting the Mycroft core functions and the creation of management skills to meet requirements concerning, e.g., data access control and user privacy, and
2. integrating the measures to counter noise as prepared in T4.5.

Regarding 1): UBI prepares access control and user management via Keycloak⁹. Besides, recent versions of Keycloak provide features to address General Data Protection Regulation (GDPR) compliance, such as consent management. UBI adapts Mycroft core and the Android App to support access tokens to protect conversations. Besides, UBI, BIBA, and IDEAL elaborate on how we can address the issue that users inject the personal data of others via dialogs. One approach is to replace names with placeholders via an additional NLU component.

Regarding 2): BIBA conducted initial tests with an industry-grade headset inlay for ear shells. These tests indicated that the noise filtering is strong, and the transcription quality does not decrease much. BIBA and PIA will perform further experiments in the factory.

Besides, there will be various dialog management improvements to concretize the example dialogs provided in earlier deliverables and to improve the user experience.

⁹ <https://www.keycloak.org/>

Annex A

This annex contains a summary of the utterances usable in this demonstrator, variations in wording are possible but may result in misunderstanding (conversational breakdown). The exact answer may differ from the indicated ones due to the continuous evolution of the demonstrator. Also, the order of user utterances is free.

Happy path: coala lab demo 1

<start>

U: Hello COALA.

C: I'm listening

U: <scans QR code 4898643HKJGD > here is the new product



C: Got it. The last 4 digits are KJGD.

U: There is a scratch on the door. Please record that defect with severity high.

C: The recorded defect is: product with serial number 4898643HKJGD, component door, defect name is scratch, defect severity is high. Is that correct?

U: I need an overview about this product.

C: Product KJGD has been launched in production today at 12:00, it is a microwave, and will be delivered to the market.

U: Get me the top 3 defects

C: The top three defects are scratch, missing part, and dirt.

C: <display pareto diagram>

U: Is scratch affecting this SKU only

C: Ten units of MN 313 BL A MICROONDE AI found.

U: Are there any products with scratches?

C: There are 36 SKUs with the defect scratch. Do you want to see them?

U: Yes please.

C: Okay.

C: <visualize table>

U: Show me the production data for defective components, both produced and planned

C: 105 items produced.

C: 200 items planned.

C: <visualize table>

U: Focus on the produced items

C: 105 items produced today.

C: <visualize updated table>

U: <silent>

<end>