



Cost effective, Multilingual, Privacy-driven voice-enabled Services

www.compriseh2020.eu

Call: H2020-ICT-2018-2020

Topic: ICT-29-2018

Type of action: RIA

Grant agreement N°: 825081

**WP N°4: Cost-effective multilingual
voice interaction**

Deliverable N°4.1: SDK software architecture

Lead partner: ASCO

Version N°: 1.0

Date: 29/11/2019



Document information	
Deliverable N° and title:	D4.1 – SDK software architecture
Version N°:	1.0
Lead beneficiary:	ASCO
Author(s):	Gerrit Klasen
Reviewers:	Raivis Skadiņš, Askars Salimbajevs (TILDE), Aurélien Bellet (INRIA)
Submission date:	29/11/2019
Due date:	30/11/2019
Type ¹ :	R
Dissemination level ² :	PU

Document history			
Date	Version	Author(s)	Comments
04/10/2019	0.1	Gerrit Klasen	First draft of the deliverable
08/11/2019	0.2	Gerrit Klasen	Completed, not reviewed version
20/11/2019	0.3	Gerrit Klasen	Review comments processed
29/11/2019	1.0	Emmanuel Vincent & Zaineb Chelly	Final version reviewed by the Coordinator and the project manager

¹ **R**: Report, **DEC**: Websites, patent filling, videos; **DEM**: Demonstrator, pilot, prototype; **ORDP**: Open Research Data Pilot; **ETHICS**: Ethics requirement. **OTHER**: Software Tools

² **PU**: Public; **CO**: Confidential, only for members of the consortium (including the Commission Services)

Document summary

This deliverable (D4.1) is the initial deliverable of WP4 which is about cost-effective multilingual voice interaction. It explains the architecture of the COMPRISE SDK which is aligned to T4.1.

The COMPRISE SDK integrates, and interfaces multiple algorithms, APIs and tools developed within the COMPRISE project, which are needed for the development of multilingual, voice-enabled applications. Well-defined interfaces are presented to the developer to easily access a desired functionality. Besides the language abstraction, the SDK allows their users to compile their output cross-platform and to create executable applications both for Android and iOS.

This deliverable explains the needs and purposes of the SDK in the way described above and why the project is not relying on existing SDKs available in the market and which provide a similar functional behaviour.

The actual architecture is presented within a global COMPRISE environment representation. It contains multiple types of components which are introduced separately (Operating and Training Branch, as well as the COMPRISE Cloud Platform). A plan is also provided which describes how everything is implemented and how it is documented in a proper way.

An implementation plan has been developed to ensure that the integration finishes within the given timeframe, respecting due dates of other components.

Table of contents

- 1. Introduction..... 5
- 2. Purpose of the SDK 5
- 3. SDK Architecture 6
 - 3.1 Global Vision 7
 - 3.2 COMPRISE SDK Developer UI 10
 - 3.3 COMPRISE SDK Client Library 15
 - 3.3.1 Operating Branch 15
 - 3.3.2 Training Branch 22
 - 3.4 COMPRISE Cloud Platform API 26
 - 3.5 Future Documentation 27
- 4. Implementation Plan 28
- 5. Conclusion 35

1. Introduction

WP4 “Cost-effective multilingual voice interaction” aims to reduce the cost for developers implementing and deploying multilingual, voice-enabled solutions. The work package is addressing two ways to solve this problem. Firstly, weak (automatic) labelling methods are being developed to allow the transfer of Speech-To-Text and Spoken Language Understanding systems to new domains where the requirement for laborious and time-consuming manual labelling would stand in the way of rapid adoption. This weak labelling is compatible with privacy guarantees. Speech-To-Text and Spoken Language Understanding models are then trained using both strongly and weakly labelled data. Secondly, a Software Development Toolkit (SDK) is being developed, giving access and interfaces to all the functionalities needed in an easy and holistic way.

This deliverable is about the architecture definition of the SDK. It explains both the need and purpose of the SDK for the COMPRISE project, as it is described after this introduction in Section 2. A differentiation is given why the consortium decided not to use similar toolkits present in the market.

Section 3 describes the concrete content of the SDK, beginning with its positioning in the whole COMPRISE environment and its relation to additional entities like the COMPRISE Cloud Platform, the COMPRISE Apps, developers in their local environment, and other services. Afterwards, it states how the toolkit is realised on a technical level and how the functionality is represented for the user. The same is done for the components which are included into the Apps generated by the toolkit. They are containing both components already present in the market (Operating Branch) and some being created within this project (Training Branch). An insight is also given on how the SDK needs to interface to the COMPRISE Cloud Platform and why. Finally, Section 3 states how everything is documented to make everything well-understandable for developers.

As the SDK integrates multiple COMPRISE components, it forms one of the core outputs of the project. Finalising the SDK on time is highly important. Therefore, an implementation plan is provided in Section 4. Additionally, this plan must respect the delivery dates of the training branch components developed within COMPRISE.

2. Purpose of the SDK

The COMPRISE SDK is a toolkit including a developer UI and client libraries to help developers to implement multilingual, privacy-aware, voice-enabled applications.

To achieve this, the SDK offers compactly bundled functionalities to its users, so that they can easily include all needed functionalities in a voice-enabled application. This avoids developers having to search for multiple interfaces and libraries like Speech-To-Text, Machine Translation, Natural Language Understanding, and many more on their own, or even to implement such functionalities by themselves. Hence, the SDK is suitable for rapid prototyping, and first voice-enabled applications can be quickly created. This leads to cost savings in terms of human resources. Firstly, the developers of these applications need much less time to complete their task, which reduces the cost for employers. Secondly, less expertise is needed to build the product, which translates into lower salaries hence additional cost savings for employers.

There exist multiple SDKs for developing voice-enabled applications on the market, which are also designed to achieve cost savings. Examples are Siri for Developers³ by Apple, AVS Device SDK⁴ by Amazon, Google Assistant SDK by Google⁵, just to name the biggest ones.

The COMPRISE SDK is required since it does not just reduce the development cost: it also addresses two other specific objectives of COMPRISE: Privacy-by-design and Inclusiveness. These two additional key objectives cause cost-effectiveness to become a focus again. Indeed, new solutions must be found for the three objectives to be jointly achieved.

Additionally, the competitors out in the market are usually limited to one language and / or have problems to understand users talking with interruptions, accents and other conditions. The development of multilingual voice-enabled applications is hence complicated, error-prone and time-consuming. The COMPRISE SDK supports the creation and usage of multilingual voice-enabled applications. The users of COMPRISE Apps are able to speak in their own language to interact with a dialog system in another language. They also benefit from better system performances, if the way they speak is unusual.

In addition, Apps created with this toolkit remove any kind of privacy-related information from the user's input before it is sent outside of the user's device. The SDK Client Library contains data transformation components to delete as much private information as possible from the users' speech and text, while preserving enough information to label the resulting "neutral" data manually if needed and to train large-scale user-independent Speech-To-Text, spoken language understanding, and dialog management systems on these "neutral" data in the cloud. Compared to competitors who merely claim not to share or abuse users' data, the COMPRISE SDK includes technical solutions to actively protect sensitive contents.

3. SDK Architecture

The project needs to provide ways for developers to easily use the COMPRISE functionalities for their own application development. The answer to this requirement is the COMPRISE SDK, whose role is introduced within the global system environment first (Section 3.1). Afterwards, the usage of the toolkit itself, the SDK Developer UI, is explained in Section 3.2. This includes the usage of all components, summarized within the SDK Client Library. It is running on the client device as presented in Section 3.3. The Developer UI gives the possibility for developers to train and maintain their models within the COMPRISE Cloud Platform (Section 3.4). The provided toolkit also needs to contain extensive documentation to allow developers to easily understand the provided functionality and how to use it (Section 3.5).

The COMPRISE SDK builds a foundation, which can be shared throughout different platforms including Android and iOS. The SDK Client Library integrates the following tools which already individually exist on the market (in the following: Operation Branch). These are mainly responsible for speech interaction:

³ <https://developer.apple.com/siri/>

⁴ <https://developer.amazon.com/de-DE/alexa/alexa-voice-service/sdk>

⁵ <https://developers.google.com/assistant/sdk>

- Speech-To-Text
- Machine Translation
- Spoken Language Understanding
- Dialog Management
- Spoken Language Generation
- Text-To-Speech.

The SDK Client Library also integrates the results produced within the COMPRISE project (in the following: Training Branch), especially regarding privacy awareness and user inclusiveness:

- Privacy-Driven Speech Transformation
- Privacy-Driven Text Transformation
- Personalised Learning.

The Operation and Training Branches exist within the whole system environment and also refer to the COMPRISE Cloud Platform. Within the context of the deliverable, only the SDK-relevant subcomponents are described.

The SDK provides interfaces to the COMPRISE Cloud Platform which allow COMPRISE developers to do the following tasks:

- Training of speech recognition models to be used by the App
- Management of these models
- Management of the neutral training data sent by the app
- Authentication and Authorisation to perform actions on the Platform
- Analytics of the present data.

The SDK is complemented by a documentation, consisting of:

- A description of how to install and use the SDK interface, as well as a technical overview of how the toolkit works in an external document.
- An explanation about the COMPRISE client components included in the Client Library and about their functionality in the corresponding GitLab⁶ repositories.
- Swagger documentation about all the APIs used in the SDK during the App creation. This especially affects the SDK Developer UI itself (as the Developer UI is a webpage connecting to internal backend micro-services) and the interrelation to the COMPRISE Cloud Platform.

3.1 Global Vision

Figure 1 shows the whole system environment and the global architecture of the project, including the role of the SDK. The system can be subdivided into four major parts:

COMPRISE SDK Developer UI (Developer's machine / Local desktop environment)

This is for the actual developer using his / her client machine, on which the SDK can be installed locally. The SDK Developer UI consists of multiple backend (micro-)services coordinated by the Express.js⁷ Framework, which the user is able to access via a browser of his / her choice. During implementation, the user broadly designs the App by adding simple template bundles consisting of HTML, CSS and JavaScript to the App.

⁶ <https://gitlab.inria.fr/comprise>

⁷ <https://expressjs.com/>

These templates contain all COMPRISE related interfaces and functionality which need to run on a client device. The user also trains linguistic models within the COMPRISE Cloud Platform, through interfaces provided by the SDK. They are trained and stored into the cloud, where future client Apps download them later to make usage of them. As the toolkit needs to support cross-platform development, it has been decided to let Angular⁸ be the technical foundation of the App, as it is suitable to be compiled to both Android and iOS phones, and it can be easily modified by adding or removing HTML, CSS and JavaScript. The framework to execute the compilation to native devices is Ionic⁹.

COMPRISE App (Client's (mobile) device / Smartphone)

This is the actual output of the SDK generation process. The App is a native Android (Java code) or iOS (Swift code) representation based on an Angular App compiled by the Ionic framework. It contains all Client Library components needed for multilingual, privacy-aware voice interaction, which are summarised in Section 3.3. During runtime, the App neutralises both speech and text input with the help of privacy-aware speech and text transformations and sends it to the COMPRISE Cloud Platform, where it is used for additional training purposes. The App receives domain-specific models from the Cloud, to be later reused in the Automatic Speech Recognition component, as well as for Natural Language Understanding and Dialog Management. These models are not only trained on the COMPRISE Cloud Platform but also personalised within the App regarding accents or “hard-to-understand” users to achieve a better user experience.

COMPRISE Cloud Platform

It gives the developer access to several functionalities: Model Training, Model Management, Neutral Training Data Management, Analytics, Authentication and Authorisation. The exact interfaces and requirements are described within D5.2 “Platform hardware and software architecture”. The main purpose is to allow developers to securely store, curate and label data and to provide access to user-independent models trained on these data. These models are used later on by the COMPRISE SDK Client Library within the COMPRISE App, where personalised learning takes place. To protect the privacy of the user, the neutralisation components are executed on the client. Then, once free of sensitive information, the resulting neutral content is sent to the COMPRISE Cloud Platform for training purposes.

Other Cloud Services

Other cloud services host components which are not related to the COMPRISE Cloud Platform. These are typically Operation Branch components which require more computing power than available on the client device. For example, Machine Translation is affected by this, as it needs to run on powerful servers to guarantee an acceptable performance. Like the COMPRISE Cloud Platform, these other services are not accessible by unauthorised parties and privacy-aware neutralisation is employed before the submission of any data.

⁸ <https://angular.io/>

⁹ <https://ionicframework.com/>

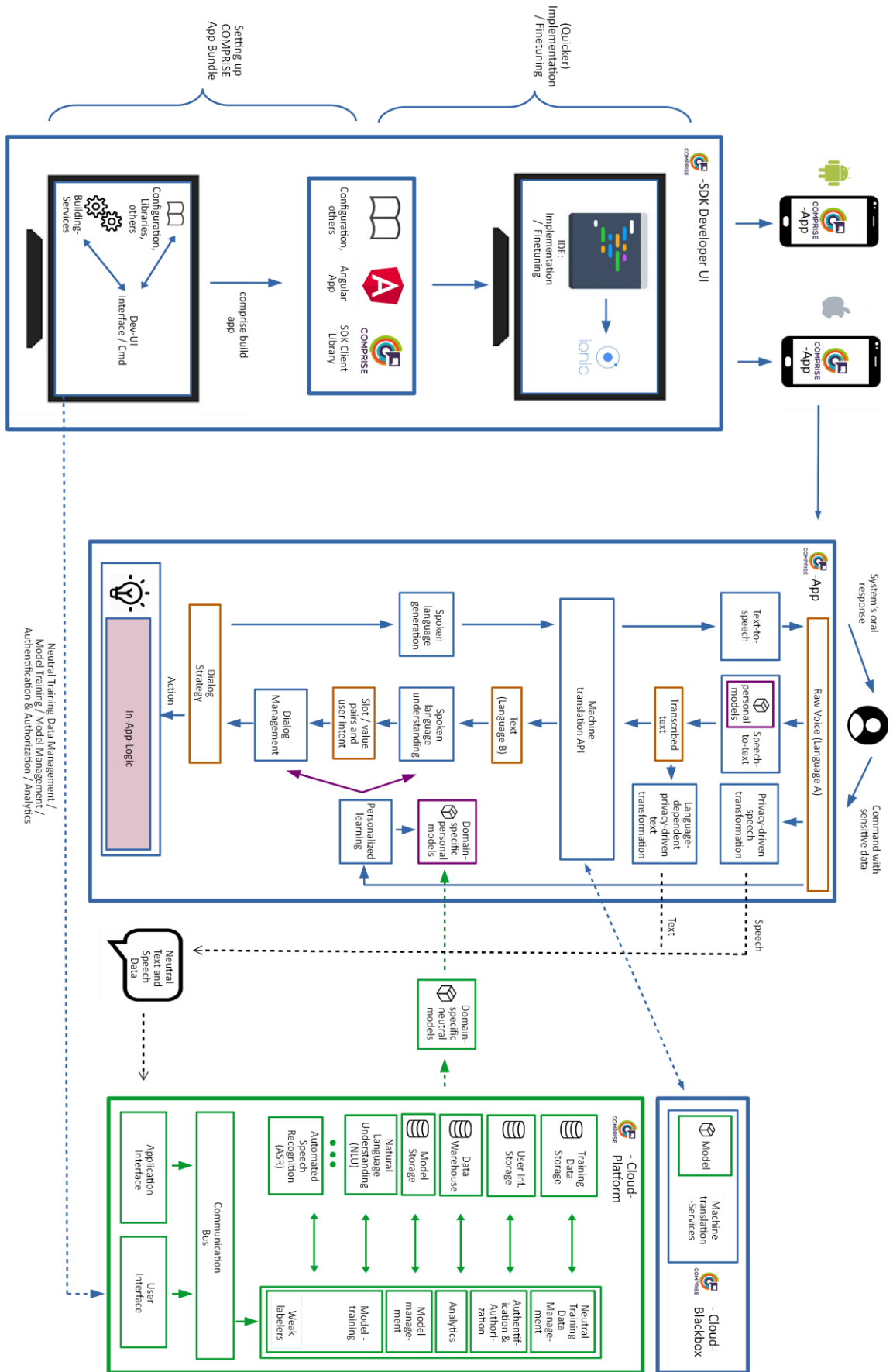


Figure 1: COMPRISE global architecture.

3.2 COMPRISE SDK Developer UI

The COMPRISE SDK offers a Graphical User Interface (GUI) editor to allow application developers to use the implementation of end-user interfaces for their solutions. The SDK is accessible as a plain browser. This browser is connected to multiple, external (micro-) services which run at the backend and execute the commands given within the browser. This also delivers callbacks to the browser UI if needed. These services provide large ranges of visual templates implemented in HTML and CSS, as well as the COMPRISE Clients Component's related functionality in JavaScript / TypeScript. Developers can combine them within the Developer UI to preview the future COMPRISE App. The templates can stand for themselves (e.g., customised GUI elements or simple panels) or contain/bind to additional behavior and JavaScript logic (e.g., Microphone button enabling Speech-To-Text). Application developers, thus, have a high degree of flexibility and power as there are barely any restrictions in merging templates. The editor still develops an abstraction layer concept, subordinating the entities into abstraction levels.

Example:

```

Layer 0: Application-Body
    Layer 1: Page
        Layer 2: Microphone Button
    Layer 1: Page
        Layer 2: About us Dialog
  
```

This strategy produces additional guidance for the developer and speeds up the process of rapid prototyping. The editor also allows the inheritance of external configurations like translations for internationalisation, as well as internal and individual configurations or attributes for each template.

The Toolkit is split into two parts:

Server-Side component(s)

This component is doing all the logic and consists of multiple micro-services working together. They listen to user inputs, transform them into a clickable, virtual prototype (= future COMPRISE App) and return a preview back to the user. From a technical point of view, the micro-services are likely to be coordinated within the Express.js¹⁰ environment which is a “minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications”. Within here, RESTful APIs are provided and are connected to the client interface.

Client-side component

This component is nothing more than a single HTML file processing the user input. It accepts various user commands to create an App and forwards them to the server. Based on the prototype delivered back, users have the possibility to insert other commands.

Figure 2 gives a short overview of the workflow. A client wants to create a template called “XYZ” within his/her COMPRISE App. The client sends the task to the server, where a

¹⁰ <https://expressjs.com/>

central component forwards the task to subsequent micro-services. They investigate how the task is translated into a viewable representation and include all GUI (HTML/CSS), Logic (JS) or global settings and translations for the App's panels (JSON) bound to the chosen template. After the required building blocks have been collected, they are combined and nested within each other. The resulting output is a clickable prototype of an App and is presented back to the client.

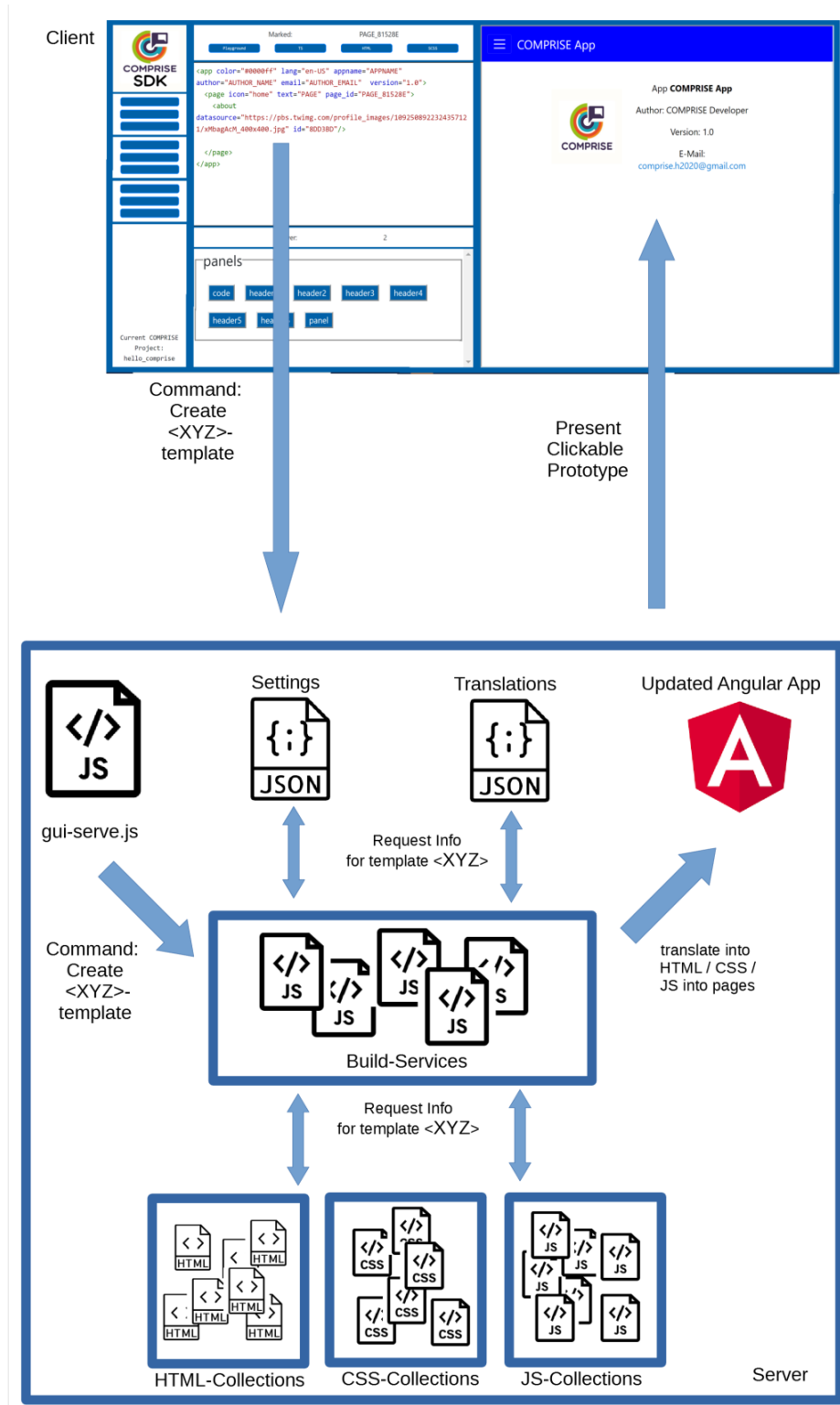
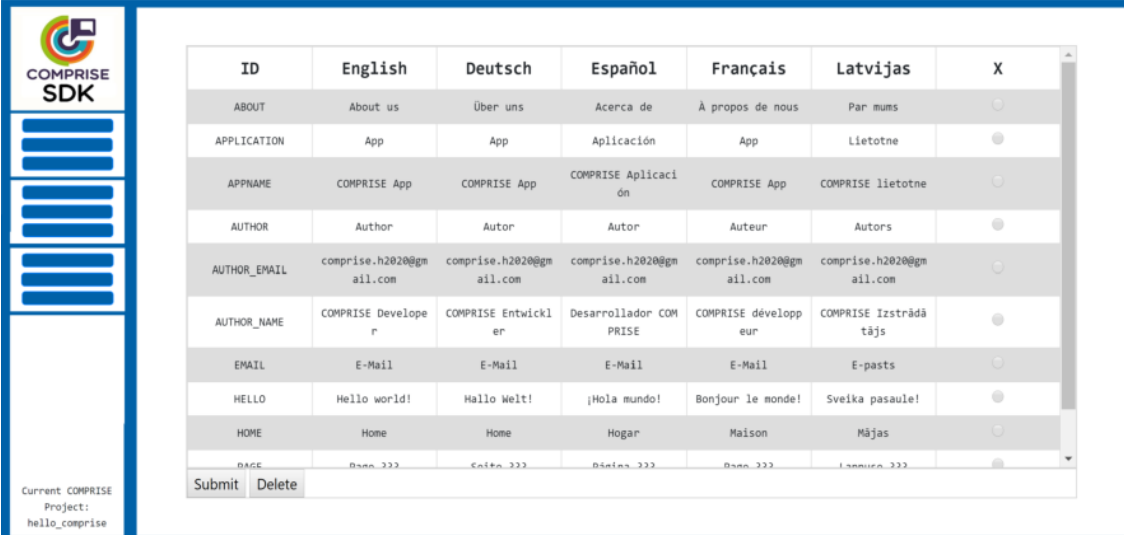


Figure 2: Application component inclusion.

As COMPRISE is about multilingual applications, besides the support of user inputs in various languages, the SDK also needs to provide internationalisation aspects for the text panels used. The UI provides the possibility for developers to assign translations for multiple languages to a certain language key. These keys can be matched to the panels which show the desired language. Figure 3 shows the procedure with some example languages. The user can assign as many translation key/value pairs as he or she wants, and assign them to related panels, popups, tooltips, alerts and similar.



ID	English	Deutsch	Español	Français	Latvijas	X
ABOUT	About us	Über uns	Acerca de	À propos de nous	Par mums	<input type="radio"/>
APPLICATION	App	App	Aplicación	App	Lietotne	<input type="radio"/>
APPNAME	COMPRISE App	COMPRISE App	COMPRISE Aplicación	COMPRISE App	COMPRISE lietotne	<input type="radio"/>
AUTHOR	Author	Autor	Autor	Auteur	Autors	<input type="radio"/>
AUTHOR_EMAIL	comprise.h2020@gmail.com	comprise.h2020@gmail.com	comprise.h2020@gmail.com	comprise.h2020@gmail.com	comprise.h2020@gmail.com	<input type="radio"/>
AUTHOR_NAME	COMPRISE Developer	COMPRISE Entwickler	Desarrollador COMPRISE	COMPRISE développeur	COMPRISE Izstrādātājs	<input type="radio"/>
EMAIL	E-Mail	E-Mail	E-Mail	E-Mail	E-pasts	<input type="radio"/>
HELLO	Hello world!	Hallo Welt!	¡Hola mundo!	Bonjour le monde!	Sveika pasaule!	<input type="radio"/>
HOME	Home	Home	Hogar	Maison	Mājas	<input type="radio"/>
PAGE	Page 333	Seite 333	Página 333	Page 333	Lappuse 333	<input type="radio"/>

Submit Delete

Current COMPRISE Project: hello_comprise

Figure 3: Internationalisation for App panels.

In the course of the project, the SDK supports this functionality with the languages of the project partners, which are:

- English
- French
- German
- Latvian
- Lithuanian
- Portuguese.

When the developer has reached a prototype of the COMPRISE App and assigned translations to the textual representation, as a next step, he/she needs to train domain-specific user-independent models for Speech-To-Text and/or Spoken Language Understanding for usage in the application.

Concerning Natural Language Understanding, for example, Figure 4 shows how SDK users can assign multiple questions to intents and train an intent detector.

This could be done with the usage of existing interfaces from TILDE.AI (as it is used for Spoken Language Understanding, Dialogue Management and Natural Language Generation, see Section 3.3.2). The models generated are available in the COMPRISE Cloud Platform later. The interfaces are used to train models which are reused within the Client Library, as explained in the following (Sections 3.3 and 3.4).

The UI representation of this can be done as shown for the text panel internationalisation or the Spoken Language Understanding training example, for instance. Also, it is possible to embed already existing user interfaces from TILDE.AI as an iframe instead.

The concrete way on how to support these user interfaces within the SDK is under discussion now.

Developers can also make usage of the COMPRISE Cloud Platform UIs for model analytics purposes for example, which can be embedded within the SDK UI. Before users can access this functionality, they need to authenticate and be authorised to access the COMPRISE Cloud Platform. The exact specifications are found in D5.2 “Platform hardware and software architecture”.

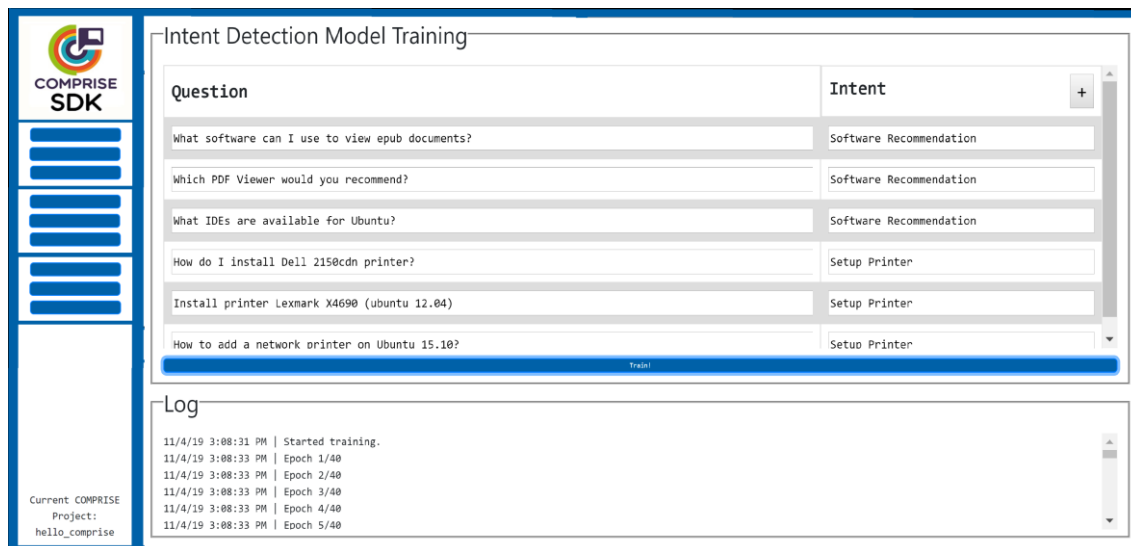


Figure 4: Intent detection training within the SDK.

Once this is done, one could build the app. A preview of the result will be shown directly within the Developer UI to give the developer a chance to do additional quick changes before the App is deployed. As most of the functionalities are auto-generated to ease the programming process for the user, he/she still might need to implement his/her own modifications. The toolkit therefore grants access to major parts of the generated source-code directly within the Developer UI without the need of opening it within an external IDE. The final step is the deployment of the solution towards the client device. This workflow is illustrated in Figure 5: Overview of the SDK workflow..

To summarise, COMPRISE developers style their App broadly in terms of optical representation and logic. They ensure internationalisation for the App’s visual representation in the form of panels. They mainly train and manage models, which will be used by the App. The App is built, which generates previews to check on again within the editor. Once all the work is done, the user deploys the App to smartphones.

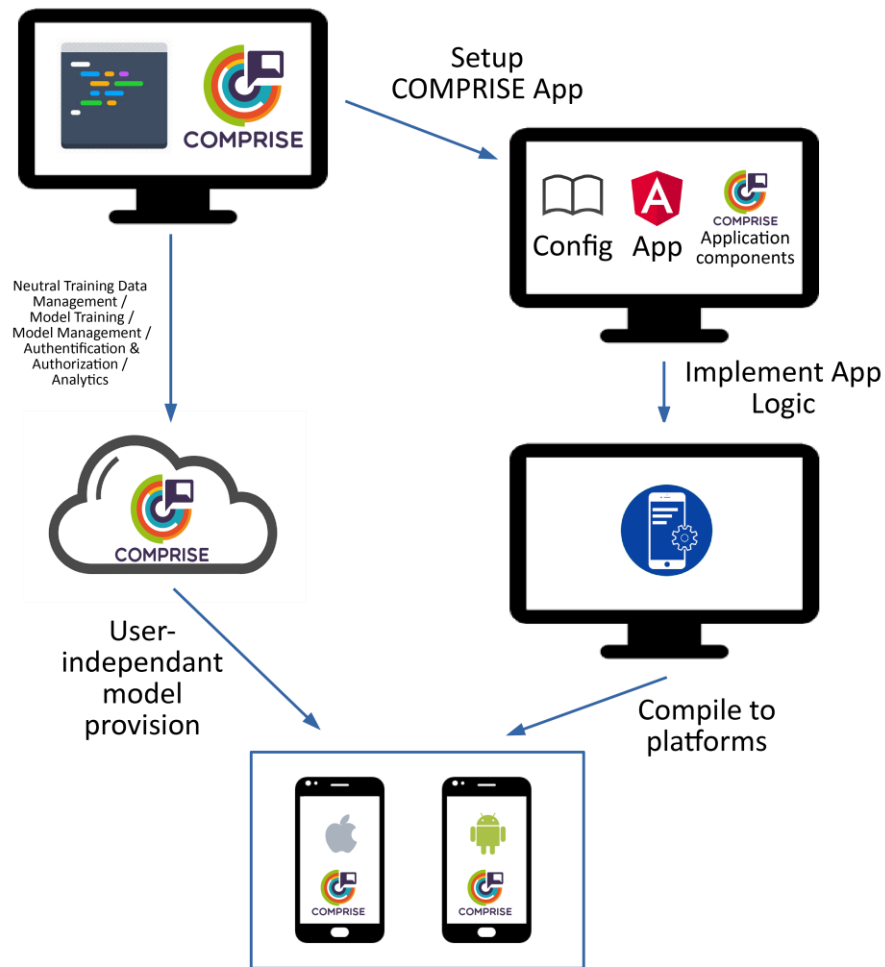


Figure 5: Overview of the SDK workflow.

The SDK Developer UI can be found in the COMPRISE GitLab. An NPM package is not provided, as the SDK Developer UI is not a Client Library itself, but a framework to use. The source acts as a reference for the current status of the solution over the deliverable milestones (additionally to a downloaded archive) and the project lifetime in general.

Component Name	COMPRISE SDK Developer UI
Responsible Partner	ASCO
Most relevant deliverables	<ul style="list-style-type: none"> • D4.1 “SDK software architecture”, R, PU, M12 • D4.3 “Initial COMPRISE SDK prototype”, R+DEM, PU, M21 • D4.5 “Final COMPRISE SDK prototype and documentation”, R+DEM, PU, M30
GitLab-Repository	https://gitlab.inria.fr/comprise/comprise_sdk_developer_ui
NPM-Repository	- none -

3.3 COMPRISE SDK Client Library

The COMPRISE SDK Developer UI is designed to deploy multilingual, privacy-aware Apps which reuse the models trained on the COMPRISE Cloud Platform. To achieve this, the SDK must include suitable interfaces into the deployed App, which can receive these models. Additionally, standard voice technology must be included to ensure voice-enabled behaviour itself. Parts of the App also need to neutralise both spoken and textual information to support privacy awareness.

These requirements make it necessary to provide a Client Library, which carries the various COMPRISE-related components and functionalities listed above. Although each component may be used individually, the components are categorized into Operation Branch vs. Training Branch components. They are detailed in the following two subsections.

As introduced in the beginning of Section 3, the Operation Branch is mainly about the speech interaction itself which is not in the scope of COMPRISE. Existing solutions are reused from the market and these are:

- Speech-To-Text
- Machine Translation
- Spoken Language Understanding
- Dialog Management
- Spoken Language Generation
- Text-To-Speech.

The Training Branch entities are being developed over the project lifetime and ensure the realisation of some of the overall and specific project goals like privacy-driven transformation, and language model personalisation or inclusiveness:

- Privacy-Driven Speech Transformation
- Privacy-Driven Text Transformation
- Personalised Learning.

The library is provided in GitLab for maintenance reasons, as well as an installable library on NPM which is used for the client device.

Component Name	COMPRISE SDK Client Library
Responsible Partner	ASCO / AII
Most relevant deliverables	None for the Operation Branch as components exist. Training Branch related deliverables are mentioned in Section 3.3.2.
GitLab-Repository	https://gitlab.inria.fr/comprise/comprise_sdk_client_library
NPM-Repository	https://www.npmjs.com/package/comprise_sdk_client_library

3.3.1 Operating Branch

The Operating Branch consists of a set of components which are used to ensure user-interaction, namely chains of speech and language processing tools. They allow voice-enabled behaviour with the user in terms of understanding the content given, properly fulfilling the tasks given by that command, and providing a suitable answer. For instance,

a speech command telling to order a bottle of wine would cause the system to add an order into the shopping list, generating the order itself and verbally informing the user that the order has been completed. As described in Section 2, this kind of functionality already exists on the market with good performance. The architecture of the SDK adapts this already approved behaviour as the project does not aim to reinvent the wheel, and it focuses on other aspects instead. Nevertheless, the Operating Branch features a few differences in comparison to existing solutions:

1. Some of the Operating Branch components run locally on the user's device. This is done to reach a higher degree of privacy for the user, as the App then does not need to send voice data with potentially sensitive information to the cloud. Rather than asking the user to trust the cloud provider not to misuse his/her data, COMPRISE SDK-Developer-UI-generated Apps process the data locally as much as possible. Certain components, for example Machine Translation or Spoken Language Understanding, may still run within the cloud due to performance reasons - or other reasons. In that case, the Client Library neutralises the sensitive content prior the use of non-local components.
2. The Operating Branch includes on-the-fly Machine Translation. Compared to other intelligent virtual assistants, this allows the users of COMPRISE Apps to interact with the App verbally, even when their mother tongue is not a popular language. Reusing the example of ordering a bottle of wine, competing virtual assistants would not be able to fulfil this task when the wine distributor only allows English language as an input. This would exclude non-English speaking users. Including Machine Translation, COMPRISE SDK-Developer-UI-generated solutions allow the translation from niche languages into the pivot needed to execute in-App commands, and translate the result back into the language of the speaker.
3. Like other solutions, voice-related components are aligned with the language models they use. As presented in Section 3.3.2, COMPRISE Apps contain Personalised Learning modules. They are responsible for receiving domain-specific, neutral models that have previously been trained within the Cloud Platform. These models become personalised models as they are customised by the Personalisation module from the Training Branch. They are assigned to the corresponding components again after modification. So, the Operation Branch components like Speech-To-Text, Natural Language Understanding or Dialog Management, are not affected, but work with continuously updated models, which lead to an improved user experience, for instance for accented or hard-to-understand users.

The tables below list the details of every Operation Branch component. As mentioned, every entity is already available on the market and can be included in the COMPRISE environment with a different degree of effort. The components are not modified regarding their functionalities, as they are not the core focus of the project.

Each component is introduced via its actual name, a short description of the provided functionality, and via the specification of the partners who are involved in the integration process or in the component development itself. As the components need to be compatible with Android and iOS, investigations have been made for suitable technical solutions for both operating systems. The results are reported with their names and their corresponding source URLs. After performing the modifications needed for integration, for further usage and if needed, the modified components are both published on GitLab for future work, and on NPM as a package which then can be installed on the client device. If running locally, the content represents the component itself. If the components

functionality runs in the Cloud, the repository represents the client API. Optionally, additional comments are made if special conditions are present for a certain component.

For some components the needed functionality can be executed in JavaScript/Typescript only and does not need a middleware or wrapper to native code to access the device functionality. In this case, only the installation of the NPM package is needed. For other components a wrapper is needed. The Cordova¹¹ framework is used to fulfil this task. This requires the additional installation of the Cordova component in addition to the NPM package.

Component Name	Speech-To-Text (STT)
Responsible Partner	ASCO / All
Description	Receives speech input from the App user and transforms it into a textual representation.
Planned / Current Technical Solution for Android	Kaldi Speech-To-Text ¹² (Android adaptation)
Repository / URL	https://github.com/alphacep/kaldi-android
Planned / Current Technical Solution for iOS	Kaldi Speech-To-Text (iOS adaptation)
Repository / URL	https://github.com/edwardvalentini/kaldi-ios-poc
GitLab-Repository for COMPRISE	https://gitlab.inria.fr/comprise/comprise_speech_to_text
NPM-Repository for COMPRISE	https://www.npmjs.com/package/cordova-plugin-comprise-speech-to-text
Cordova-Plugin?	Yes (Kaldi is native Code)
Additional Comments	Realised as a Cordova ¹³ Plugin

Component Name	Machine Translation (MT)
Responsible Partner	ASCO / TILDE

¹¹ <https://cordova.apache.org/>

¹² <https://kaldi-asr.org/>

¹³ <https://cordova.apache.org/>

Description	<p>Transmits the textual, privacy-neutralised representation of the user's voice input to other COMPRISE Cloud Services. There, TILDE's Machine Translation framework translates from the input language to a pivot language, and returns the result back to the client app, before processing by the Spoken Language Understanding component.</p> <p>Machine Translation is also needed after the Natural Language Generation component has created an answer in the pivot language. This answer is translated back into the user's input language in the same way.</p>
Planned / Current Technical Solution for Android	TILDE's Machine Translation API
Repository / URL	https://tilde.com/developers/machine-translation-api
Planned / Current Technical Solution for iOS	- same as for Android -
Repository / URL	- same as for Android -
GitLab-Repository for COMPRISE	https://gitlab.inria.fr/comprise/comprise_machine_translation
NPM-Repository for COMPRISE	https://www.npmjs.com/package/comprise_machine_translation
Cordova-Plugin?	No
Additional Comments	<p>The Machine Translation component requires too much computational power to run with an acceptable performance on the client device. Therefore, Machine Translation is running within other COMPRISE Cloud Services, being accessed through an API within this component.</p> <p>This component can surely be replaced by locally deployed solutions after the project lifetime, as soon as the footprint of future solutions (e.g., the one currently being developed by the Bergamot project for desktop browsers) has reduced enough, or smartphone hardware has become powerful enough to fulfil the requirements.</p>
Component Name	Spoken Language Understanding (SLU)
Responsible Partner	ASCO / TILDE

Description	Within COMPRISE, takes textual user input in a pivot language and analyses it to figure out the intent of the user, on which the dialog management can react.
Planned / Current Technical Solution for Android	TILDE.AI ¹⁴
Repository / URL	https://dev-nlu1-am.portal.azure-api.net/
Planned / Current Technical Solution for iOS	- same as for Android -
Repository / URL	- same as for Android -
GitLab-Repository for COMPRISE	https://gitlab.inria.fr/comprise/comprise_natural_language_understanding
NPM-Repository for COMPRISE	https://www.npmjs.com/package/comprise_natural_language_understanding
Cordova-Plugin?	No
Additional Comments	<p>The Natural Language Processing (i.e., Spoken Language Understanding + Dialogue Management + Natural Language Generation) component requires too much computing power to run with an acceptable performance on the client device. Therefore, Spoken Language Understanding runs within other COMPRISE Cloud Services, being accessed through an API within this component.</p> <p>This component can surely be replaced by locally deployed solutions after the project lifetime, as soon as smartphone hardware has become powerful enough to fulfil the requirements.</p>

Component Name	Dialog Management (DM)
Responsible Partner	ASCO / TILDE
Description	Takes the user intent, combines it with other parameters such as the current conversation status, configuration restrictions, user settings and others to figure out the action to perform.

¹⁴ <https://www.tilde.ai/>

Planned / Current Technical Solution for Android	TILDE.AI
Repository / URL	- not known yet -
Planned / Current Technical Solution for iOS	- same as for Android -
Repository / URL	- same as for Android -
GitLab-Repository for COMPRISE	https://gitlab.inria.fr/comprise/comprise_dialog_management
NPM-Repository for COMPRISE	https://www.npmjs.com/package/comprise_dialog_management
Cordova-Plugin?	No
Additional Comments	<p>Although Dialogue Management is mentioned separately in this overview, it is likely in the further development that Dialogue Management and Natural Language Generation are merged into one component.</p> <p>The Natural Language Processing (Spoken Language Understanding + Dialogue Management + Natural Language Generation) component requires too much computing power to run with an acceptable performance on the client device. Therefore, it runs within other COMPRISE Cloud Services, being accessed through an API within this component.</p> <p>This component surely can be replaced by locally deployed solutions after the project lifetime, as soon as smartphone hardware has become powerful enough to fulfil the requirements.</p>

Component Name	Natural Language Generation (NLG)
Responsible Partner	ASCO / TILDE
Description	Verbalises the identified action into a textual answer to be presented to the application user. The answer is in the pivot language and must be translated back to the user's language before that.
Planned / Current Technical Solution for Android	TILDE.AI
Repository / URL	- not known yet -

Planned / Current Technical Solution for iOS	- same as for Android -
Repository / URL	- same as for Android -
GitLab-Repository for COMPRISE	https://gitlab.inria.fr/comprise/comprise_natural_language_generation
NPM-Repository for COMPRISE	https://www.npmjs.com/package/comprise_natural_language_generation
Cordova-Plugin?	No
Additional Comments	<p>Natural Language Generation is mentioned separately in this overview, but it is likely in the further development that Dialogue Management and Natural Language Generation are merged into one component.</p> <p>The Natural Language Processing (Spoken Language Understanding + Dialogue Management + Natural Language Generation) component requires too much computing power to run with an acceptable performance on the client device. Therefore, it runs within other COMPRISE Cloud Services, being accessed through an API within this component.</p> <p>This component can surely be replaced by locally deployed solutions after the project lifetime, as soon as smartphone hardware has become powerful enough to fulfil the requirements.</p>

Component Name	Text-To-Speech (TTS)
Responsible Partner	ASCO / All
Description	Transforms the textual answer in the input language into an audio signal, which is spoken out to the user of the app.
Planned / Current Technical Solution for Android	<ol style="list-style-type: none"> 1. Kaldi Speech-To-Text / "Idlak" (Reversed adaptation to achieve Text-To-Speech) (English), 2. TILDE Integrated solutions API (Latvian, Lithuanian), 3. native Android Text-To-Speech Component (Other Languages)
Repository / URL	<ol style="list-style-type: none"> 1. https://github.com/Idlak/Idlak 2. https://www.tilde.com/developers/integrated-solutions-api 3. https://developer.android.com/reference/android/speech/tts/TextToSpeech
Planned / Current Technical Solution for iOS	<ol style="list-style-type: none"> 1. - same as for Android - (English), 2. - same as for Android - (Latvian, Lithuanian),

	3. Native iOS Text-To-Speech Components (Other Languages)
Repository / URL	<ol style="list-style-type: none"> 1. - same as for Android – 2. - same as for Android - 3. https://developer.apple.com/documentation/avfoundation/avspeechsynthesizer
GitLab-Repository for COMPRISE	https://gitlab.inria.fr/comprise/comprise_text_to_speech
NPM-Repository for COMPRISE	https://www.npmjs.com/package/cordova-plugin-comprise-text-to-speech
Cordova-Plugin?	Yes (Option 2. does not need it, but Option 1 and 3 do, and all three are jointly distributed)
Additional Comments	For Text-To-Speech, usable training data sets are barely publicly available. The preferred solution, Ildak, runs locally and can be trained on available data in English language. To support Latvian and Lithuanian, a secured connection to TILDE's Integrated Solutions API is planned. Data sets for other languages are not available yet, which requires us to use the native Text-To-Speech functionality of the corresponding OS. In the latter case, the user shall download native languages packages first to support offline functionalities in favour of privacy.

3.3.2 Training Branch

The Training Branch includes the remaining COMPRISE SDK Client Components which are compiled by the SDK Developer UI into the SDK Client Library. These components are not related to the speech and language processing chain but extend and supplement it with valuable features the project aims to achieve. They are developed during the project lifetime and broadly fall into two categories: privacy-aware transformation of user inputs both in speech and textual representations, and personalisation of domain-specific Speech-To-Text and Spoken Language Understanding models locally on the client device. The latter influences the models provided for Speech-To-Text components, as well as for Spoken Language Understanding and Dialog Management.

Regarding the role of privacy in speech processing chains and workflows, the Privacy-Aware Speech Transformation is positioned as the very first client component after an App user has spoken a command. It removes sensitive information such as voice characteristics. The Privacy-Aware Text Transformation neutralises the resulting text by removing critical information like credit card numbers or other contents users do not want to share. While the removed and/or replaced information is stored locally to be restored if needed, neutralised speech and text data are sent to the COMPRISE Cloud Platform as potential training data.

For example, the statement “I want to order three boxes of insulin against my diabetes” could be transformed into “I want to order three boxes of medicine against my sickness”.

The transformed command contains enough information to be suitable for model training purposes within the COMPRISE Cloud Platform but hides the information about the user's health. This prevents abuse of sensitive information. Still, the intent of ordering three insulin boxes is kept locally to do the actual order with the external distributor.

The second major area to address, personalised learning, is done locally as well. After the application starts, it downloads domain-specific models from the COMPRISE Cloud Platform, which are trained within the COMPRISE Cloud Platform during development (initialisation) and application runtime (receiving training data). There, additional computations are done to personalise the model towards the individual user with the help of adaption data. The purpose of this strategy is to increase the performance of Speech-To-Text towards the typical characteristics of the user, like voice accent, communication speed, or other personal, typical characteristics. Similarly, individual model adaptations are done for Dialog Management and Natural Language Understanding tools.

The components involved in this branch are described in the tables below in a similar way as the Operation Branch components above. The components are named and described shortly, together with the responsible parties who develop the solution. Since the components are being developed, no fixed repositories of existing solutions can be provided consequently. Still, all the technologies planned so far are introduced. As this still is under investigation, this could change during the project runtime. The deliverables mainly reporting about the components are stated as well to give references about additional information during the project. As differentiation between Android and iOS where needed for the Operation Branch, it is planned to develop a common codebase for both platforms. Just as before, the modified components for further usage are both published on GitLab for future work, if needed, and on NPM as package, which then can be installed on the client device. Additional comments are stated, if needed.

Currently, most of the solutions are implemented in Python. As all COMPRISE SDK Client Library components are running in JavaScript / TypeScript, it is expected to include "Bridge"-functionality within the final COMPRISE-NPM-packages, which executes the Python functionality. This is not possible in JavaScript / TypeScript code directly, but within Android and iOS. As result, native Code is needed and all components require Cordova plugins as explained for Text-To-Speech and Speech-To-Text in the previous section. To run Python on native OS, we will likely use BeeWare¹⁵, as it provides templates for Android¹⁶ and iOS¹⁷.

Component Name	Privacy-Driven Speech Transformation (PDST)
Responsible Partner	ASCO / INRIA / USAAR
Description	Runs locally on the client device. Removes characteristics of user's voice (like voice characteristics, speed, loudness or others) and transforms it into a neutral representation to

¹⁵ <https://beeware.org/>

¹⁶ <https://github.com/beeware/Python-Android-template/tree/3.4>

¹⁷ <https://github.com/beeware/Python-iOS-template/tree/3.4>

	ensure privacy. Leaves enough sound quality to the sentence to make it suitable for training purposes.
Pool of Planned Technology to use	Python, Python libraries for deep learning (Pytorch), Python libraries for audio I/O
Most relevant deliverables	<ul style="list-style-type: none"> • D2.1 “Baseline Speech and text transformation and model learning library”, R+OTHER, PU, M6 • D2.2 “Improved transformation library and initial privacy guarantees”, R+OTHER, PU, M17 • D2.3 “Final transformation library and privacy guarantees”, R+OTHER, PU, M27
GitLab-Repository for COMPRISE	https://gitlab.inria.fr/comprise/comprise_privacy_driven_speech_transformation
NPM-Repository for COMPRISE	https://www.npmjs.com/package/comprise_privacy_driven_speech_transformation
Cordova-Plugin?	Yes (Python only runs on native code)
Additional Comments	- none -

Component Name	Privacy-Driven Text Transformation (PDTT)
Responsible Partner	ASCO / USAAR
Description	Runs locally on the client device after Privacy-Driven Speech Transformation and Speech-To-Text. Finds sensitive information in the text and replaces it with neutralised content or even removes the entries to ensure the users' privacy. Leaves enough sound quality to the sentence to make it suitable for model training.
Pool of Planned Technology to use	Python, Python libraries for deep learning (TensorFlow, PyTorch), potentially some kind of encryption library
Most relevant deliverables	<ul style="list-style-type: none"> • D2.1 “Baseline Speech and text transformation and model learning library”, R+OTHER, PU, M6 • D2.2 “Improved transformation library and initial privacy guarantees”, R+OTHER, PU, M17 • D2.3 “Final transformation library and privacy guarantees”, R+OTHER, PU, M27
GitLab-Repository for COMPRISE	https://gitlab.inria.fr/comprise/comprise_privacy_driven_text_transformation
NPM-Repository for COMPRISE	https://www.npmjs.com/package/comprise_privacy_driven_text_transformation

Cordova-Plugin?	Yes (Python only runs on native code)
Additional Comments	- none -
Component Name	Personalised Learning (PL)
Responsible Partner	ASCO / INRIA / USAAR / ROOT
Description	Uses domain-specific, user-independent speech models from the COMPRISE Cloud Platform and leverages adaption data from the users to personalise them to user-dependent models. The results help Speech-To-Text, Spoken Language Understanding and Dialogue Management achieve better performance for “non-average” users in terms of speech characteristics.
Pool of Planned Technology to use	Python, Python libraries for deep learning (TensorFlow, Pytorch), Python libraries for audio I/O
Most relevant deliverables	<ul style="list-style-type: none"> • D3.2 “Initial personalised learning library for speech-to-text”, R+OTHER, PU, M17 • D3.4 “Initial personalised learning library for speech-to-text”, R+OTHER, PU, M27
GitLab-Repository for COMPRISE	https://gitlab.inria.fr/comprise/comprise_personalized_learning
NPM-Repository for COMPRISE	https://www.npmjs.com/package/comprise_personalized_learning
Cordova-Plugin?	Yes (Python only runs on native code)
Additional Comments	- none -

As mentioned, Machine Translation itself is an Operation Branch component, but the way of connecting it to the local workflow, and translating content to a pivot language and backwards, increases the inclusiveness of the user in a new way. This is because Spoken Language Understanding, Dialogue Management and Natural Language Generation can be better trained on that language, as more training data is available. As Machine Translation was previously explained, the research topic still affords the contribution of multiple deliverables, which is stated as a reference in addition.

Component Name	Machine Translation (MT)
Responsible Partner	ASCO / TILDE
Most relevant deliverables	<ul style="list-style-type: none"> • D3.1 “Initial multilingual interaction library”, R+OTHER, PU, M15

- | | |
|--|--|
| | <ul style="list-style-type: none"> • D3.3 “Final multilingual interaction library”, R+OTHER, PU, M26 |
|--|--|

3.4 COMPRISE Cloud Platform API

The COMPRISE Cloud Platform is the core entity regarding model training. It provides various functionalities for COMPRISE developers and applications to define and use domain-specific models. The platform provides interfaces for both the COMPRISE SDK Client Library and for developers within the COMPRISE SDK Developer UI. For all functionalities, users need access authentication and authorisation components, to register and authenticate themselves towards the COMPRISE Cloud Platform and have access to everything.

Once authenticated, applications/developers/annotators can manage the available pool of neutral data and models for different purposes, e.g., annotators label domain-specific neutral speech and text data. The labelled data are then used for training of domain-specific neutral models.

Regarding the Client Library, the COMPRISE Cloud Platform offers an API with access model management for services like TILDE.AI, as COMPRISE SDK developers use their API to initialise the models provided later, like Spoken Language Understanding (see Figure 4). The client App will also make use of the model management components of the platform to get these models. After successful training, the models are downloaded and used in the App at runtime. Every time neutralised data is incoming from the application, the model training component from the COMPRISE Cloud Platform can use the updated data set to continue training Speech-To-Text and Spoken Language Understanding models for instance. The neutralised data are handled by the training data management components of the platform.

Developers themselves also have access to some UI, e.g., analysis of the current data and models stored within the platform or authentication and authorisation.

More information along this general overview is given within D5.2 “Platform hardware and software architecture”.

Figure 6 gives an overview about the workflow. The COMPRISE Cloud Platform offers both Interfaces for the SDK Client Library running in client Apps, and for developers or annotators, represented by the SDK Developer UI. The latter uses TILDE.AI’s APIs to prepare models which are stored in the COMPRISE Cloud Platform. This guarantees the deployment of a multilingual, privacy-aware App instead of a normal application.

Once the App is running, it uses the technical interfaces to immediately download the trained neutral models. They form the foundation for individualisation by the Personalised Learning library on the mobile device for Automated Speech Recognition, Natural Language Understanding and Dialog Management. In return, every user speech and text input is neutralised and transmitted back to the COMPRISE Cloud Platform to allow future additional model training.

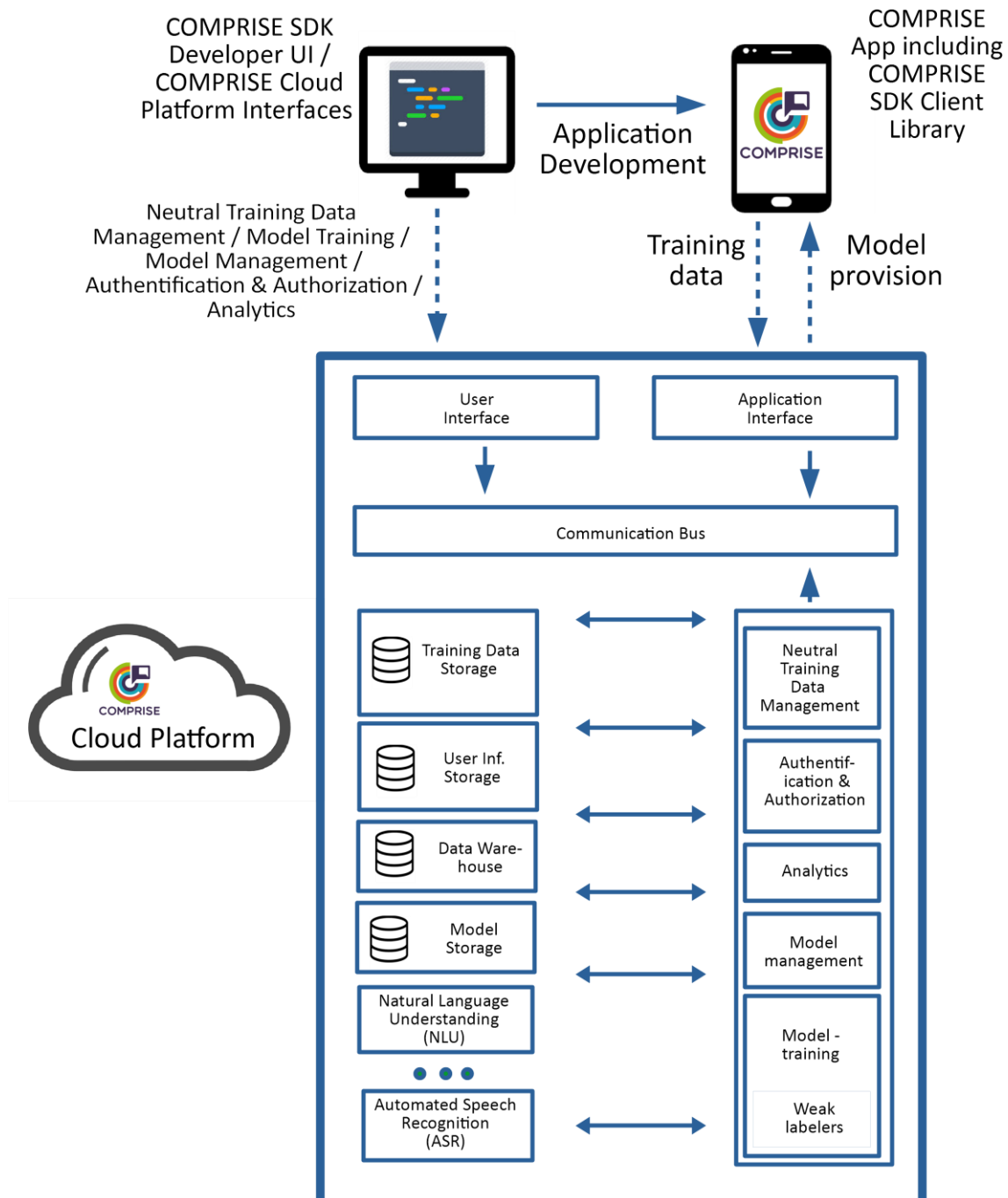


Figure 6: Interdependencies between the COMPRISE SDK Developer UI and COMPRISE Cloud Platform.

3.5 Future Documentation

Cost-effectiveness as a core topic of WP4 requires the SDK to enable COMPRISE App development with low investment of time and human resources. This mainly affects a well-defined architecture and technical concept, but also needs developers to understand all the functionalities very quickly. For that reason, D4.5 provides comprehensive documentation in various regards, the enables developers to start prototyping very quickly, but also to understand all functions in deeply enough.

This is realised in three different ways.

Firstly, an external document is provided in the form of a manual. It contains a technical overview regarding the architecture, technical details, and the way the SDK was realised in a short overview. Mainly, it guides the user through the technical dependencies needed to install and setup the SDK, the installation process itself, and the way how user may use the SDK. This contains explanations about all the functions available in general, but also a small sample project to guide the developer through the creation of a COMPRISE App, from its initialisation, over the speech model training, to the deployment on device.

Secondly, like initially stated in Section 3.3, all of the COMPRISE SDK Client Library components used within the Apps and which are needed to achieve the COMPRISE goals are explained to the user, as well as all the other components which are not directly related to the project, but make it possible to easily style a simple application UI including panels, images, or buttons.

Besides the actual usage of the SDK, developers also get insight about all APIs used, if they want to modify and adapt the toolkit itself for their own purposes. Swagger¹⁸ is used for the documentation, as it is a tool for proper interface specification and descriptions. The reference of this documentation is within D4.5 “Final COMPRISE SDK prototype and documentation” as well as in the manual provided.

4. Implementation Plan

The COMPRISE SDK interfaces and integrates most of the other technical components reused and implemented in the project lifetime. As a result, it is very important to ensure that the SDK is developed on time. An implementation plan is presented in order to have a clear vision about what to integrate and when. This is also mandatory due to the fact that especially the COMPRISE Cloud Platform interfaces might change, the Operation Branch Components have a different degree of effort to be integrated and the final Training Branch components will only be available at a later stage of the project. By taking all these facts into consideration, the implementation started already at M3 (instead of M6), as also reported within D1.4 “Interim Progress report”, so that there is enough time to react if unforeseen issues arise. The plan also includes the SDK deliverables needed in WP4, which are the actual one D4.1, as well as D4.3 and D4.5.

As this deliverable is submitted to internal review on M12, work regarding the SDK has been already delivered. This period, M3 – M10, is also mentioned as already delivered action within the plan. It is stated in bold how the process went and if some deviations were seen and caused. The content since M11 is already the reaction on the progress done so far and future assumptions are given.

Time: Month 3+4	Type: Specification
Action: Define global architecture of the COMPRISE SDK	
All partners owning technical components being integrated by the SDK meet and discuss how it positions itself into the whole COMPRISE system environment.	

¹⁸ <https://swagger.io/>

Technical barriers and changes are identified and reflected to the whole architecture. Decisions are made whether technical parameters require modifications to the architecture suggested in the proposal. The results of these discussions have been included into the global representation stated in Section 3.1.

This was realised as planned.

Time: **Month 5-8**

Type: **Development**

Action: **Enable SDK to create cross-platform applications**

As COMPRISE allows developers to deploy their solution to devices supporting multiple operating systems, the project needs to ensure early on that the SDK is able to generate an executable application. This includes the definition of a suitable cross-platform framework, the deployment on devices, and the inclusion of a library which contains the COMPRISE-related functionalities in the future.

This was realised as planned. It was decided that the focus of all development is to provide a fully working SDK environment for one OS. This is more important than to tackle all operating systems at once. As a result, Android components are realised first. The corresponding iOS versions are adapted once Android is running well.

Time: **Month 7+8**

Type: **Development**

Action: **Inclusion of stub components mocking COMPRISE functionality**

The client App generated by the SDK mostly includes speech- and privacy-related components and interfaces, which depend on each other and follow a pre-defined workflow. Obtaining an impression of these interdependencies early on is needed to figure out additional restrictions and information which could not be foreseen. As a result, a very easily and quickly achievable mockup is developed, that contains mainly Operation Branch and Training Branch components. These components are not COMPRISE-conform yet (lacking privacy, etc.), as they are intended to give an uncomplicated technical overview at first. They are replaced one by one by the solutions planned in COMPRISE until M30.

This was realised as planned.

Time: **Month 9**

Type: **Integration**

Action: **Integration of Speech-To-Text (STT)**

Starting in M9, components from the Operation Branch are integrated, as these components are already present on the market, re-usable with little modifications, and not the core output of the project. Additionally, although an initial version of the privacy-preserving transformations is available, the other new components developed within COMPRISE are not available yet for integration.

The order of integration of the Operation Branch components is the order of their usage within the communication workflow. This requires integrating the Speech-To-Text component first.

This was realised as planned, except the iOS adaption.

Time: **Month 9**

Type: **Integration**

Action: **Integration of Machine Translation (MT)**

Regarding the Operation Branch, Machine Translation is the next component needed to translate the text output by the Speech-To-Text component from the input language into a pivot language for the targeted App, e.g., English. Although the integration of this component is due by M15 within D3.1 (Software components and documentation for Speech-To-Speech translation and integration of dialog systems in the operating branch), the component owner, TILDE, confirmed that this component is already in a good enough status in M9 to be integrated within the operating branch. Note that the improvement of the software components and documentation is due in M15.

This was realised as planned for both the COMPRISE SDK Developer UI and the Client Library.

Time: **Month 10**

Type: **Integration**

Action: **Integration of Spoken Language Understanding (SLU)**

The translated content needs to be understood by the application to detect the user intent. Spoken Language Understanding is therefore integrated next. This includes both a client component and an API interface to the TILDE.AI interface within the Developer UI.

This was realised as planned for both the COMPRISE SDK Developer UI and the Client Library.

Time: **Month 10**

Type: **Integration**

Action: **Integration of Dialog Management (DM)**

After detecting the intent of the application's user, the Dialog Management component needs to be included to ensure that a suitable answer is brought in response to that intent and that it respects other background dependencies like communication history, additional settings, etc. This includes both a client component and an API interface to the TILDE.AI interface within the Developer UI (if suitable and needed).

This was realised as planned for the Client Library. The connection to TILDE.AI interface within the Developer UI is done in M15, within the timeslot implemented for fixing open issues. As mentioned in Section 3.3.2, it is likely that Dialogue Management and Spoken Language Generation are merged within the next months.

Time: **Month 11**

Type: **Integration**

Action: **Integration of Spoken Language Generation (SLG)**

If the user intent has been interpreted correctly and was executed with success, the Spoken Language Generation component needs to generate a textual representation of the answer, which could be something simple like “Your order has been transmitted successfully”. This includes both a client component and an API interface to the TILDE.AI interface within the Developer UI (if suitable and needed).

Time: **Month 11+12**

Type: **Integration**

Action: **Integration of Text-To-Speech (TTS)**

The answer generated by Spoken Language Generation and translated back to the original language by Machine Translation needs to be converted into speech. This is done by a Text-To-Speech component which is integrated as the last Operation Branch component.

Time: **Month 12**

Type: **Deliverable**

Action: **Submitting D4.1: SDK Software architecture**

This deliverable is submitted at the end of the first year of the project. It includes all the results achieved regarding the SDK specification including the main requirements and the software architecture.

Time: **Month 13+14**

Type: **Development / Integration**

Action: **iOS Adaption (Operation Branch)**

As stated above, we focus on the integration of the Operation Branch for Android first. At this point, this has been achieved, so this slot is used to adapt it to iOS and make it work with the same quality level.

Time: **Month 15**

Type: **Integration / Development**

Action: **Fixing open deployment and integration issues of Operation Branch components**

This slot is used as a buffer to solve open and so far unforeseen issues regarding one of the previous items. This especially addresses integration aspects of the Operation Branch components, but also may aim to address issues regarding deployment of the SDK-generated Apps.

Time: **Month 16**

Type: **Integration**

Action: **Integration of Privacy-Driven Text Transformation**

After having completed the Operation Branch and the COMPRISE Cloud Platform API functionalities, the implementation plan aims to integrate the three components (first version) of the Training Branch until D4.3 in M21.

Firstly, the plan aims to integrate the Privacy-Driven Text Transformation in M17. This is realistic, as the component was delivered in M9 within deliverable D2.1 (design, implementation, and evaluation of baseline transformations focusing on deleting the user's identity and words carrying critical information, and model learning).

Time: Month 17	Type: Integration
Action: Integration of Privacy-Driven Speech Transformation	
After text transformation, the plan aims to integrate the Privacy-Driven Speech Transformation in M17. The component has also been delivered in M9 within deliverable D2.1 (design, implementation, and evaluation of baseline transformations focusing on deleting the user's identity and words carrying critical information, and model learning).	

Time: Month 18+19	Type: Integration
Action: Integration of Personalised Learning for Speech-To-Text	
The Speech-To-Text component implemented in M9 is extended by personalised learning mechanisms. The component is due in M17 within deliverable D3.2 (design, implementation, and evaluation of initial model personalisation strategies for Speech-To-Text).	

Time: Month 19	Type: Development / Integration
Action: iOS Adaption (Training Branch)	
As stated above, we focus on the integration of the Training Branch for Android first. At this point, this has been achieved, so this slot is used to adapt it to iOS and make it work with the same quality level.	

Time: Month 20	Type: Integration / Development
Action: Fixing open deployment and integration issues of Training Branch components	
M21 is the delivery date of D4.3 (see below) which also requires a first prototype showing the SDK work integrated. This also requires resolving bugs and open issues before being able to develop properly. M20 will therefore be used as a buffer to solve open and so far unforeseen issues regarding one of the previous items. This especially addresses integration aspects of Training Branch components this time.	
The plan therefore foresees the completed integration of both the Operation and the Training Branches (in a first version) before the delivery of D4.3 in M21.	

Time: Month 20 + 21	Type: Development
Action: Provision of a sample App using COMPRISE functionality	

D4.3 shows the results generated in WP2, WP3 and T4.1. As this requires a working prototype, the development starts when the most critical bugs (if any) caused by the integration are solved. Besides the Operating Branch, the Training Branch components are integrated as a first version.

Time: **Month 21**

Type: **Deliverable**

Action: **Submitting D4.3: Initial SDK prototype**

D4.3 shows the results generated in WP2, WP3 and T4.1. It includes the prototype just mentioned, including more detailed implementation information regarding the technology used in both the Operation Branch and the Training Branch components.

Time: **Month 22**

Type: **Improvement**

Action: **Improvement of Privacy-Driven Text Transformation**

After having delivered the first prototype, the plan mainly foresees to continuously update the initial deliveries of Training Branch components to improved/final versions.

This affects the Privacy-Driven Text Transformation first. The initial version is updated with the improved one delivered in M17 within deliverable D2.2 (design, implementation, and evaluation of speech and text transformations addressing more types of private information and initial statistical utility/privacy bounds).

Time: **Month 23**

Type: **Improvement**

Action: **Improvement of Privacy-Driven Speech Transformation**

The Privacy-Driven Speech Transformation follows in M23. The initial version is updated with the improved one delivered in M17 within deliverable D2.2.

Time: **Month 24+25**

Type: **Improvement / Development**

Action: **COMPRISE App creation process improvements**

After having initialised the creation of COMPRISE-aware applications 18 months ago, it is likely that dependent frameworks have been improved and updated, offering improvements in development speed and deployments. The SDK is updated in that regard.

Additionally, having gained experience with the usage of the Developer UI at that point, requests made by the consortium and their developers regarding user experience will be fixed.

Time: **Month 26**

Type: **Improvement**

Action: **Improvement of Machine Translation (MT)**

Initialised and integrated in the Operating Branch in M9 (maybe with small corrections by M15, if D3.1 requires), Machine Translation is integrated in the training branch in D3.2 (Software components and documentation for Speech-To-Speech translation and integration of dialog systems in both the operating and the training branch).	
Time: Month 27	Type: Improvement
Action: Final version of Privacy-Driven Text Transformation	
In this month, D2.3 foresees the final design, implementation, and evaluation of text transformations and final statistical utility/privacy bounds. The COMPRISE SDK Client Library has to adapt to the last changes done since M22.	
Time: Month 27	Type: Improvement
Action: Final version of Privacy-Driven Speech Transformation	
D2.3 also delivers the final design, implementation, and evaluation of speech transformations and final statistical utility/privacy bounds. The COMPRISE SDK Client Library has to adapt to the last changes done since M23.	
Time: Month 28	Type: Improvement
Action: Final version of Personalised Learning	
Personalised Learning was firstly integrated in M18+19 for Speech-To-Text only. D3.4 (Final design, implementation, and evaluation of model personalisation strategies for Speech-To-Text, Spoken Language Understanding and Dialog Management), which is to be delivered in M28, finalises this component and extends it to the personalisation of Spoken Language Understanding and Dialog Management.	
Time: Month 29	Type: Integration / Development
Action: Fixing open deployment and integration issues of all components	
M30 is the delivery date of D4.5 (see below), which also requires a final prototype showing the SDK work integrated. This also requires resolving bugs and open issues. M29 will therefore be used as a buffer to solve open and so far unforeseen issues regarding one of the previous items. This especially addresses integration aspects of the Training Branch components in a final version this time.	
Time: Month 29-30	Type: Improvement / Development
Action: Provision of a final demonstration App using COMPRISE functionality	
D4.5 shows the results generated in WP2, WP3 and T4.1. This includes all the components discussed earlier in a final version.	

Time: Month 30	Type: Documentation
Action: Provision of a final demonstration App using COMPRISE functionality	
The COMPRISE SDK is fully developed at this point, enabling the consortium to finalise the documentation as mentioned in Section 3.5.	
Time: Month 30	Type: Deliverable
Action: Submitting D4.5: Final SDK prototype and documentation	
D4.5 is the final output included in WP4 and this implementation plan, showing the final prototype integrating the research results of WP2, WP3, and T4.1 and the Swagger online documentation. The deliverable includes the documentation and the final prototype App mentioned above, as well as the description of changes done within the components since D4.3.	

5. Conclusion

This deliverable explains why the COMPRISE SDK is needed in addition to other toolkits for the development of voice-enabled applications available on the market. Namely, the COMPRISE SDK provides additional benefits in terms of privacy, cost-efficiency and inclusiveness in comparison to its competitors. The document gives an overview of the architecture of the SDK and how it fits into the whole COMPRISE system environment and which dependencies it has to the COMPRISE Cloud Platform, to developers and the applications running on the client device. It is used to design and deploy the App, and it interfaces with the COMPRISE Cloud Platform to train domain specific models. Further insights are given to better describe the usage of the COMPRISE Cloud Platform API, as well as about the COMPRISE Client Library components running within the App. The deliverable differentiates the components already present in the market (Operation Branch) from those which are developed within the project (Training Branch) to clearly highlight the focus. The planning also includes how to document every generated content to provide both quick and deep knowledge for users and developers. An implementation plan is defined to specify when each of the mentioned contents will be integrated into the SDK to achieve a final version at M30.

The next steps within T4.1 are the realisation of the implementation plan as just provided to have an initial prototype at M21. This is to approve that it is possible to translate all of the concepts that were mentioned in the plan into a technical implementation.