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 Nº: 825081

WP Nº3:	Multilingual personalised voice interaction
Deliverable Nº3.1:	Initial multilingual interaction library
Lead partner:	Tilde
Version Nº:	1.0
Date:	28/02/2020

Document information	
Deliverable Nº and title:	D3.1 – Initial multilingual interaction library
Version Nº:	1.0
Lead beneficiary:	Tilde
Author(s):	Askars SALIMBAJEVS (TILDE), Thomas KLEINBAUER (USAAR)
Reviewers:	Denis JOUVET (INRIA), Gerrit KLASEN (ASCO)
Submission date:	28/02/2020
Due date:	29/02/2020
Type[1]:	R
Dissemination level[2]:	PU

Document history			
Date	Version	Author(s)	Comments
04/02/2020	0.1	Askars Salimbajevs	Draft deliverable
14/02/2020	0.2	Thomas Kleinbauer & Askars Salimbajevs	Initial version
19/02/2020	0.3	Thomas Kleinbauer & Askars Salimbajevs	Revision based on the reviewers' comments
28/02/2020	1.0	Emmanuel Vincent & Zaineb Chelly Dagdia	Final version reviewed by the coordinator and by the project manager

[1] **R**: Report, **DEC:** Websites, patent filling, videos; **DEM:** Demonstrator, pilot, prototype; **OTHER:** Software Tools

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

Document summary

This deliverable describes effort taken in Work Package 3 to enable any user to interact with dialogue systems in any language.

The document focuses on the research activities and software components that form the foundation of the “Initial multilingual interaction library”, which will combine machine translation, speech-to-text and dialogue management.

The first part of the document is devoted to research on integration of machine translation and speech-to-text. This integration will allow a user speaking his/her own language to interact with a dialogue system originally developed in another language. We introduce a “synthetic data pipeline” approach for training machine translation models that are robust to speech recognition errors. We describe the experimental setup and empirically evaluate our approach in different conditions. Then, the outcomes are analysed and discussed.

The second part of the document is focused on the main software components of the multilingual interaction library which are machine translation and dialogue management. Because of the limitations of modern mobile devices both components are implemented as cloud-based services. Full accurate and up-to-date documentation of both components is provided online for developer convenience. Therefore, this deliverable only documents the base functions of the APIs and gives references to the full documentation.

Table of contents

1. Introduction

The objective of Work Package 3 is to enable any user to interact with dialogue systems in any language. This will be achieved by:

- Combining Machine Translation (MT) with Speech-To-Text (STT) and Text-To-Speech (TTS) in order to provide a transparent interface between a user speaking a given language and a dialogue system (or possibly a human) in another language.
- Further combining MT with the components of a dialogue system, i.e., spoken language understanding, dialogue management, and spoken language generation.
- Adapting these models to every user by running additional computations on the user's device so as to improve performance for that user specifically.

This deliverable, entitled D3.1 – Initial multilingual interaction library, focuses on the first two points and describes research efforts, software components and the documentation for speech-to-text translation and the integration of dialogue systems in the operating branch. The STT and TTS components are not described here (see Work Package 4).

Integrating MT with STT allows a user speaking his/her own language to interact with a dialogue system in another language. This combination is not straightforward. First, we can distinguish between two modes of translation:

- *Simultaneous translation*: The translated text is provided at the same time as the speaker speaks.
- *Asynchronous translation*: The translated text is provided after the speaker finishes an utterance.

Simultaneous translation reduces the delay between speech and translation and can thus improve the user experience. This is especially important for long utterances and monologues. However, it requires both real-time STT and real-time MT and creates a number of additional technological and research challenges.

In COMPRISE, we are interested in voice assistant and dialogue scenarios where utterances are usually short, and translation can be performed at the end of user input. This allows us to focus on asynchronous translation.

Further, there are two main technical approaches for integrating STT and MT:

- *Sequential speech translation*: Perform speech recognition first, then translate the recognised text.
- *End-to-end translation*: Use a single component (usually a neural network) to translate to the target language directly from the audio input in the source language.

End-to-end translation is considered to be a more promising approach. Theoretically it allows the joint optimisation of STT and MT components for a particular task and should be more robust. However, current end-to-end systems do not reach state-of-the-art performance yet. Another downside to this approach is that it requires a rare kind of training data: translated speech corpora. Such data exists only for a few languages and in small quantities, compared to large existing corpora of parallel text and annotated speech. Therefore, in COMPRISE, we will focus on sequential translation.

Another important issue concerns ungrammatical spontaneous spoken language: people speak continuously without pauses and sentence breaks, often make repetitions, correct themselves, or stop in the middle of the sentence. As stock MT expects clean input, the aforementioned issues must be addressed in the STT output. This means either correcting speech disfluencies before the resulting text gets translated or adapting the MT model to handle such artifacts.

USAAR's dialogue platform Platon which was originally envisioned as the basis for the developments in COMPRISE is no longer maintained, therefore the consortium opted for TILDE's cloud-based dialogue system Tilde.AI as an appropriate alternative. This system comprises Spoken Language Understanding, Dialogue Management, and Spoken Language Generation in an integrated framework. Tilde.AI is an appropriate choice since it is comparable in terms of features provided and — being provided by one of the consortium partners — allows for individual customisation if required for the execution of the project.

Since the dialogue system has previously been employed only in-house at TILDE, some development effort was necessary to make the new system available to all partners in the form of a web-API. In parallel to the technical tasks, substantial effort has been invested into providing documentation which is also part of this deliverable.

This deliverable is structured as follows. First, we describe the research conducted on robust integration of MT and STT. Next, we provide an introduction to the MT and dialogue management (Tilde.AI) APIs which will be used in the operating branch of COMPRISE. Finally, we conclude and present our plans for the remainder of the project.

2. Integration of machine translation and speech-to-text

In this project, and as explained in Section 1, we will focus on asynchronous sequential spoken language translation. The main problem of the sequential approach is data quality mismatch between STT and MT:

- Due to potential recognition errors, the STT system outputs a noisy textual representation of the *spoken* language. In addition, this representation does not have any punctuation and can contain disfluent and ungrammatical parts.
- An MT system is trained on *written* language; therefore, it expects input to have punctuation and be fluent and grammatically correct.

The implication of this mismatch is poor translation quality if STT and MT components are naively combined. There are several possible approaches to tackle this issue:

- Train MT on spoken language, e.g., on transcripts of STT training data.
- Post-process STT output to adapt it for MT: insert punctuation, correct disfluencies.
- Synthetically imitate spoken language features in the MT training data to train a robust MT model.

Each of these approaches has its own advantages and limitations. Theoretically, training MT on real spoken data should provide the best results. Unfortunately, STT datasets rarely come with translations and are often too small for training modern neural MT models. Still, if translation is available, STT data can be used for adaptation of a pre-trained MT model.

The post-processing approach can be a good option for high-resource languages, such as English, that have various spoken language processing tools and spoken language datasets available. This allows the development of both data-driven and rule-based solutions for correcting ungrammatical and disfluent utterances (Hassan et al., 2014). For low-resource languages, however, which might not have required datasets or processing tools, following this approach is problematic.

The idea of the third approach is to simulate STT errors in the training data of MT. There are multiple ways to do this, for example, randomly substituting correct words by potentially confusing words using acoustic and linguistic embedding similarity (Simonnet et al., 2018). This approach makes MT robust to STT errors, yet additional measures are needed to deal with ungrammatical and disfluent input. These will be investigated in further research.

In our initial experiment, we explored the latter approach, but decided to generate data with synthetic STT errors by using a pipeline of TTS and STT technologies, as will be detailed in Section 2.1. The advantage of this approach is that it generates not only substitution errors, but also insertions and deletions. In addition, it does not require additional data and uses only already available MT training data. However, it is limited by the availability of STT and TTS for a given language. In the case of COMPRISE, such technologies will be available at least for the six languages (English, French, German, Latvian, Lithuanian, Portuguese) and language pairs needed in the project.

2.1 Synthetic data pipeline

The basic idea of this approach is to convert usual MT training data to a form that resembles the raw output from the STT system. This will introduce errors that are typically found in speech recognition output into the training data, and thus make the MT system more robust. We propose the following training data pre-processing pipeline (see Figure 1):

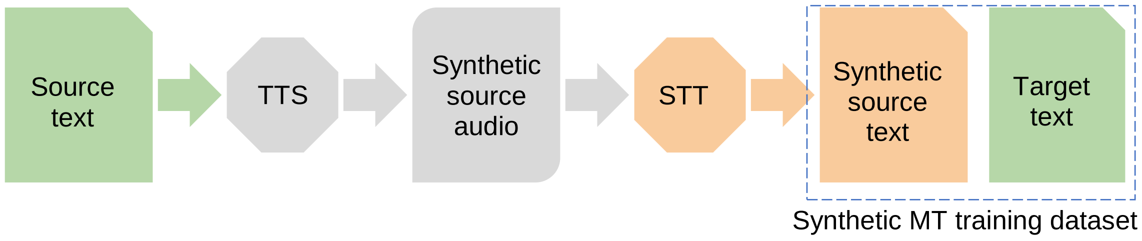

Figure 1: The synthetic data pipeline

- Synthesise source language sentences from the MT training dataset using TTS (the grey boxes in Figure 1).
- Use STT to transcribe the synthesised sentences (the orange boxes in Figure 1).
- Use STT transcriptions together with the original target sentences as the synthetic MT training dataset.

Before performing TTS and STT, the source text should be pre-processed. For example, tokens that are pronounced incorrectly should be replaced to yield a correct pronunciation, and tokens that are not pronounced in spoken language (e.g., special characters) should be filtered out. Theoretically, the pre-processing step could also inject ungrammatical and disfluent segments into sentences. This additional option will be explored in future research.

In the initial experiment, we focus on Latvian-English speech-to-text MT. For speech synthesis, we use three Tilde Latvian TTS voices. Each sentence is synthesised with one of the three voices chosen at random. The source text is filtered from characters that should not be pronounced by TTS (parentheses, quote signs, some punctuation signs).

For STT, we then use Tilde's Latvian STT system which is based on a hybrid Hidden Markov Model and a Time-delay Neural Network acoustic model. The language model is implemented on a sub-word level using Byte-Pair Encoding (BPE) and consists of 4-grams, 6-grams, and a Recurrent Neural Network language model (RNNLM). However, in order to inject more errors into the resulting synthetic training data, only the 4-gram language model is used here.

As raw speech recognition output contains numbers written with words instead of digits, the MT system will have to learn how to translate words into digits. Therefore, to simplify the training of the MT system, we apply Tilde's in-house number normalisation tool for Latvian which rewrites numbers as digits. In the case when number normalisation tools are not available for a given language, raw transcripts from the STT system could also be used as the source text for MT training.

2.2 Data

The Latvian-English WMT 2017 training dataset (Bojar et al., 2017) is used for training the MT system. The dataset consists of three smaller datasets:

- Europarl: proceedings of the European Parliament (637,599 sentence pairs).
- RAPID: press-releases taken from the Press Release Database of the European Commission (306,588 sentence pairs).
- DCEP: press-releases, session and legislative documents related to the European Parliament's activities and bodies (3,542,280 sentence pairs).

First, the data is processed by the Synthetic Data pipeline described in the previous section. The Word Error Rate (WER) of the synthesised data was approximately 20.7%. Then, we follow our baseline MT training recipe and pre-process the data using standard Moses (Koehn et al., 2007) scripts for normalisation of punctuation, tokenisation, parallel corpora cleaning, and truecasing, all of which are taken from the Moses GitHub repository[3]. The data is further split into sub-word units using byte-pair encoding. For this task, we use SubwordNMT[4] (Sennrich et al., 2016).

[3] https://github.com/moses-smt/mosesdecoder
[4] https://github.com/rsennrich/subword-nmt

For tuning, we use the NewsDev2017 dataset from WMT 2017 (2003 sentence pairs) which is also processed using the same methods and tools.

The evaluation is performed on three datasets:

- NewsTest2017 from WMT 2017 (2001 sentence pairs).
- Synthetic NewsTest2017 which is NewsTest2017 processed by the same TTS-STT pipeline (2001 sentence pairs).
- Real-world test set “Tilde Balss” (1159 sentence pairs).

The real-world test set is based on a subset of data collected by the Tilde real-time Latvian STT. This subset contains 8820 utterances and 37782 words in 39 hours of audio (including silence). Utterances come from various domains: queries, short messages, addresses, interaction with a voice-enabled educational app, etc. In addition, it contains a lot of “noise”: laughing, English and Russian speech, untranslatable or ambiguous utterances, etc. Therefore, several rounds of semi-automatic filtering were performed to select meaningful utterances from this dataset. This resulted in a dataset of 1159 Latvian utterances which were manually translated to English.

2.3 Evaluation results

Since the paradigm-shifting success of Neural Machine Translation (NMT) systems at the 2016 Conference on Machine Translation (WMT) (Bojar et al., 2016), and the invention of the self-attentional Transformer architecture (Vaswani et al., 2017), Transformer-based NMT models have become the baseline choice for MT experiments.

Therefore, for the evaluation of the synthetic data pipeline, three NMT systems with the Transformer architecture are trained:

- Baseline:
  - MT system trained on original parallel data (skipping the synthetic data pipeline).
  - Validated on just clean data (skipping the synthetic data pipeline).
- STT only:
  - MT system trained on synthetic parallel data only.
  - Validated on both clean and synthetic NewsDev2017 data.
- Mixed:
  - MT system trained on both original and synthetic parallel data.
  - Validated on both clean and synthetic NewsDev2017 data.

We use the Marian NMT toolkit (Junczys-Dowmunt et al., 2018) for the training of all three systems and the Marian base model configuration for the model hyper-parameters.

First, trained MT systems are evaluated using the BLEU metric (see Deliverable D6.2 – “Initial scientific evaluation” (submitted to the European Commission on February 28, 2020 − Public)) on clean and synthetic NewsTest2017 test sets (see Table 1). We use two versions of the synthetic NewsTest2017 test set: (1) transcribed using a 4-gram language model (like the training data) and (2) rescored by 6-gram and RNNLM. This imitates both low- and high-quality STT scenarios. The STT WERs are 20% and 8%, respectively.

Table 1: BLEU scores on clean and synthetic NewsTest2017 (higher is better).

System	**Clean NewsTest2017**	**Synthetic NewsTest2017**	**Rescored Synthetic NewsTest2017**
Baseline	**16.96** (16.35-17.54)	12.75 (12.24-13.24)	13.32 (12.79-13.9)
STT only	12.81 (12.23-13.37)	14.17 (13.57-14.75)	14.33 (13.75-14.89)
Mixed	16.88 (16.3-17.5)	**14.51** (13.98-15.11)	**15.01** (14.48-15.54)

The results show that the MT system trained on original training data performs best on the clean test set NewsTest2017, but it is not really robust, as its quality degrades significantly on the Synthetic NewsTest2017 which imitates STT recognition errors. The MT system trained on synthetic data only outperforms the baseline on synthetic test sets, but shows poor quality on clean written text. The advantages of both are combined if both original and synthetic data is used for training. The “mixed” MT system achieves the best BLEU scores on synthetic test sets and is very close to the baseline on the clean original tests set.

Table 2 shows the outcome of an evaluation on the real-world Tilde Balss test set. We use three variants of this test set: (1) reference transcripts, (2) transcribed with 4-gram LM and (3) transcribed using the full Tilde STT system, rescored with 6-gram and RNNLM. STT WERs are 22% and 17%, respectively.

Table 2: BLEU scores on Tilde Balss test set (higher is better).

System	**Tilde Balss (using reference text)**	**Tilde Balss (4-gram only)**	**Rescored Tilde Balss**
Baseline	14.09	11.14	12.24
STT only	13.65	11.88	12.38
Mixed	**15.69**	**13.50**	**14.12**

The result is similar to the result on the synthetic test set but with one significant exception: the MT system trained on a mix of original and synthetic data outperforms the baseline even when translating reference human transcripts of the test set (i.e., without STT errors). This is mainly because reference transcripts of Tilde Balss test set do not contain punctuation.

2.4 Discussion and future work

The idea of the synthetic data pipeline is to lower the mismatch between actual STT output and expected input of a standard MT system. This is achieved in two ways:

- Removal of punctuation and other tokens or characters that are not found in the spoken language.

- Introduction of noise that is similar to the noise produced by STT in the real-world.

The analysis showed that the synthesised training data have the following features of real-world STT transcripts:

- Morphology mistakes: word endings are not always aligned.
- No punctuation and no capitalisation.
- Split words: a typical STT error when one word is split into several.
- Merged words: several consecutive words can be merged together, because it is very difficult to detect the boundaries between words in spoken language.
- Missing words: sometimes STT can skip short words.
- Wrong (but phonetically similar) lexical choice: sometimes STT fails to disambiguate similarly sounding words.

We also noticed that the synthetic data pipeline introduces new errors that negatively impact the MT training due to the fact that the WMT 2017 dataset contains many foreign person names. This results in two problems:

- The TTS engines used in the synthetic data pipeline are unable to correctly pronounce most of the foreign person names.
- While the STT system used in the initial experiment is open-vocabulary, it is still biased to the most frequently used Latvian words and names.

As a result, the synthetic training data contains many examples where regular, frequently used Latvian words are translated into foreign person names. When analysing the MT of the Tilde Balss test set, we found many cases where the MT system suddenly translates simple language, such as frequent Latvian words and names, to foreign person names. Similar problems might arise for other named entities or terms.

In addition, the number normalisation tool sometimes makes mistakes, resulting in one number being translated into another. Moreover, it seems that pronunciation and normalisation of alphanumeric tokens is very ambiguous in general. This introduces a lot of noise into the training data which is not related to speech recognition.

These problems highlight the fact that there are many open questions on how to prepare synthetic training data. It is also yet unclear how much we should ask the MT system to invent things that are not present in spoken language. For example, should we ask the MT system to restore punctuation in the text, should this be done by a separate model as pre-processing or should we just ignore this issue and output translation without punctuation? The same question also applies to capitalisation, normalisation of numbers, web addresses, etc. This is an area of further research within and beyond the COMPRISE project. Nevertheless, the evaluation results allow us to conclude that:

- Mixing of clean and synthetic parallel data allows the MT system to translate both clean and noisy data.
- Having typical STT noise in parallel data allows MT systems to learn to recover from STT mistakes.
- An adapted MT system outperforms the baseline even in cases where the STT worked perfectly because it learns not to depend on punctuation.

3. Software components

3.1 Machine Translation

Ideally, MT would be a part of the COMPRISE Multilingual Interaction Library and run locally on a mobile device. However, MT is too resource-consuming and requires a large amount of computational, run-time memory, and storage resources, which are typically unavailable on mobile devices. Since the development of MT to run on-device would require a separate project on its own and the development of new MT services is out of the scope of COMPRISE, we employ the cloud-based components of Tilde MT. This cloud service will be called by the COMPRISE SDK Client Library.

This section provides the documentation of the basic Tilde MT API functions which will be used by the COMPRISE SDK (see Work Package 4). The full documentation can be found at https://www.tilde.com/developers/machine-translation-api. The Tilde MT API implements a RESTful calling style over HTTPS, i.e., all calls are inherently encrypted.

3.1.1 Authentication

All Tilde MT API requests must contain an authentication token in the HTTP header that identifies a user of Tilde MT.

The developers of applications using COMPRISE libraries will need to contact Tilde and get their client-id authorisation_token. It is also possible that requesting tokens could be integrated into the COMPRISE Cloud Platform or into the SDK. Below is the header example of a HTTP request with an authorisation token:

```
GET https://www.letsmt.eu/ws/service.svc/json/GetSystemList?appID=appid HTTP/1.1
client-id: authorisation_token
```

3.1.2 List of available systems

Tilde MT API provides a method to list available translation systems for an authenticated user:

```
GET https://www.letsmt.eu/ws/service.svc/json/GetSystemList?appID=appid HTTP/1.1
client-id: authorisation_token
```

where *appid* is a string application identifier chosen by the developer.

The response will contain a JSON document containing the description of the systems available to the user or the application. The table below presents a short description of the main metadata entries of the MT system:

Name	Type	Description
ID	string	MT system ID
SrcLanguage	complex	Source language (IETF RFC 5646)
TrgLanguage	complex	Target language (IETF RFC 5646)
Domain	string	Translation domain
Title	string	Title of the MT system

Description	complex	Description of the MT system
Metadata	complex	More metadata about the system

Response example:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 916
{
"System":[
{
"Description":{ … },
"Domain":"law",
"ID":"smt-ea364952-816a-4491-9fbc-35b41056730a",
"Metadata":[ … ]
}, … ]}
```

3.1.3 Translation

Knowing the MT system ID, developers can translate plain text or text with inline tags using the Tilde MT API:

```
GET
https://www.letsmt.eu/ws/service.svc/json/Translate?appID=appid&systemID=sys&text=text
client-id: authorisation_token
```

where:

- *appid* is a string application identifier.
- *sys* is a string translation system ID, obtained using the method in Section 3.1.2.
- *text* is the text to be translated (with or without inline tags).

For translating longer text, it is recommended to use the HTTP POST method:

```
POST https://www.letsmt.eu/ws/service.svc/json/Translate HTTP/1.1
client-id: authorisation_token
Content-Type: application/json; charset=utf-8
Content-Length: XXX
{
"appID": "appid",
"systemID": "sys",
"text": "saule spīd spoži",
"options": ""
}
```

If a request is successful, the response will contain the translated text:

```
HTTP/1.1 200 OK
Content-Length: 23
```

```
Content-Type: application/json; charset=utf-8
"the sun shines bright"
```

In the case of an errors the response will contain an error description:

```
code: 22
exception: systemNotFound
description: Translation system not found
```

3.2 Dialogue Management

The Dialogue Management functionality of the multilingual interaction library will be provided by Tilde.AI's cloud-based dialogue system (see Figure 2). This system comprises Spoken Language Understanding, Dialogue Management, and Spoken Language Generation in an integrated framework.

Normally, the developer of a mobile app will create a dialogue application with the Tilde.AI dashboard by defining a dialogue scenario, intents, actions, etc. Then, the created dialogue can be integrated into a mobile app using the Bot Service API.

In some cases, a developer might want to implement dialogue management and text generation externally and use Tilde.AI only for spoken language understanding functions. For these cases, a separate NLU API is provided by the Tilde.AI dialogue system.

Interaction with the Bot Service API and Tilde.AI NLU API will be integrated into the COMPRISE SDK Client Library, therefore developers using COMPRISE SDK will not need to access these APIs directly.

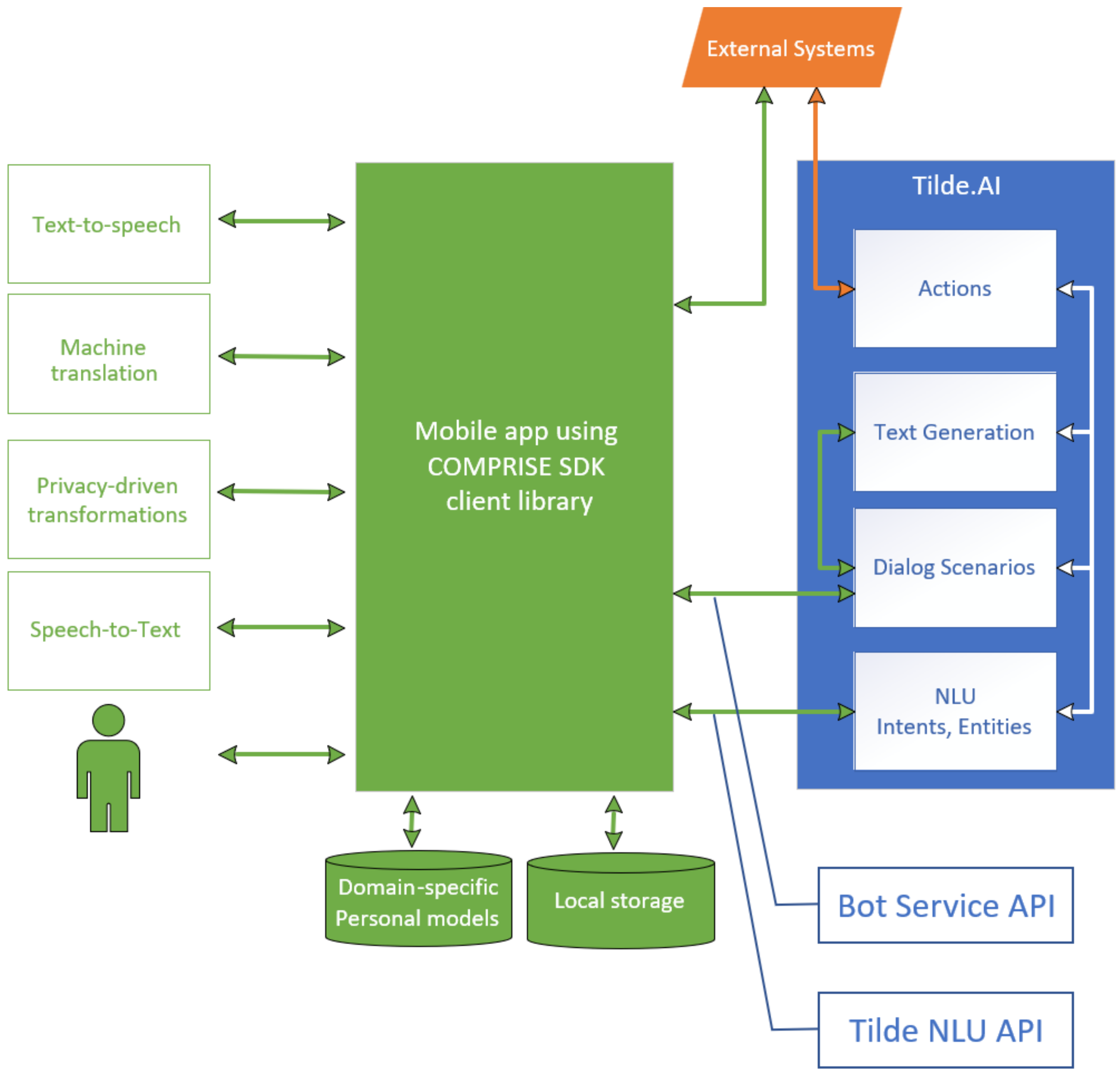

Figure 2: Interaction between Tilde.AI dialogue system and the COMPRISE SDK Client Library.

3.2.1 Comparison of Tilde.AI and Platon

The end-of-life of the Platon system necessitated a change of dialogue platform in COMPRISE. We opted for Tilde.AI as a replacement because it is backed by one of the consortium partners, but also because it is comparable in terms of relevant features. Table 3 lists the key features that both Platon and Tilde.AI cover:

Table 3: Feature comparison between Tilde.AI and Platon.

Feature	Platon	Tilde.AI
Approach	Domain-specific language	Web service
Network access	Yes (via RPC)	Yes (cloud-based web service)
Interaction with multiple users	Yes	Yes

Multilingual input/output	Yes	Yes
Built-in SLU	Yes (rule-based)	Yes (both rule-based and data-driven)
External SLU integration possible?	Yes	Yes
Built-in Dialogue Management	Yes (hierarchical task decomposition)	Yes (dialogue states, transitions, contexts)
Built-in NLG	Yes (template-based)	Yes (template-based)
External NLG integration possible?	Yes	Yes
Developer interface	Scripting	Web interface, API
Integration of business logic / backend	Java Interface, Apache Thrift	REST services, JavaScript

This comparison illustrates the feature overlap between two systems. The individual differences are mostly a consequence of the different architectural approaches (domain-specific language vs. Web Service) but in general, one does not imply practical superiority over the other. The important point to notice here is the substantial overlap of which features are provided by both systems.

3.2.2 Dialogue management API

The Tilde.AI dialogue system is hosted on Azure Bot Service and uses Microsoft Bot Framework for user communication. To still ensure the privacy aspect of the project, the COMPRISE SDK Client Library can neutralise the text data before sending it to the Microsoft servers. The preferred method of interacting with the Bot Framework is the SDK provided by Microsoft for such programming languages and platforms as Python, JavaScript, .NET and Node.js.

If this SDK is not available for the chosen language or if cannot be used for some other reason, then there are several Bot Service APIs that can be called directly:

- Bot Framework REST API.
- Direct Line API.

Full documentation of these APIs can be found on Microsoft's website[5,6]. Therefore, here we will only provide a short description of base Direct Line API 3.0 functions.

[5] Bot Framework REST API reference https://docs.microsoft.com/en-us/azure/bot-service/rest-api/bot-framework-rest-connector-api-reference?view=azure-bot-service-4.0

[6] Bot Framework Direct Line API 3.0 reference https://docs.microsoft.com/en-us/azure/bot-service/rest-api/bot-framework-rest-direct-line-3-0-api-reference?view=azure-bot-service-4.0

Authentication

A mobile app can authenticate requests to Direct Line API 3.0 either by using a secret obtained from the Tilde.AI dashboard or by using a token which is obtained at runtime. The secret or token should be specified in the authorisation header of each request using the following format:

```
Authorization: Bearer SECRET_OR_TOKEN
```

A Direct Line secret is a master key that can be used to access any conversation that belongs to the associated bot. A secret can also be used to obtain a token. Secrets do not expire.

A Direct Line token is a key that can be used to access a single conversation. A token expires but can be refreshed.

Start a conversation

Direct Line conversations are explicitly opened by the mobile app and may run as long as the user and dialogue system participate and have valid credentials. While the conversation is open, both the dialogue system and the user may send and receive messages.

```
POST https://directline.botframework.com/v3/directline/conversations
Authorization: Bearer SECRET_OR_TOKEN
```

If the request is successful, the response will be a JSON object containing an ID for the conversation, a token, a value that indicates the number of seconds until the token expires, and a stream URL that the client may use to receive activities via a WebSocket stream:

```
{
"conversationId": "abc123",
"token":
"RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qbOF5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qbOF5xPGfiCpg4F
v0y8qqbOF5x8qbOF5xn",
"expires_in": 1800,
"streamUrl":
"https://directline.botframework.com/v3/directline/conversations/abc123/stream?t=RCurR_X
V9ZA.cwA..."
}
```

Send an activity to the dialogue system

Using the Direct Line 3.0 protocol, the user and the dialogue system may exchange different types of activities, including message activities, typing activities, etc.

The following snippet provides an example of how to send a text message to a dialogue system:

```
POST https://directline.botframework.com/v3/directline/conversations/convid/activities
Authorization: Bearer SECRET_OR_TOKEN
Content-Type: application/json
[other headers]
```

where *convid* is a string identifier of the conversation obtained after starting a conversation.

The request body shall contain a JSON object describing the text message:

```
{
   "type": "message",
   "from": {       "id": "user1"    },
   "text": "hello"
}
```

If the POST request is successful, the response contains a JSON payload that specifies the ID of the activity that was sent to the bot.

```
{
   "id": "0001"
}
```

Receive activities from the dialogue system

By using the Direct Line 3.0 protocol, clients can receive activities via a WebSocket stream or retrieve activities by issuing HTTP GET requests.

To retrieve messages for a specific conversation using HTTP GET, the client should issue the following request:

```
GET
https://directline.botframework.com/v3/directline/conversations/convid/activities?watermark
=w
Authorization: Bearer SECRET_OR_TOKEN
```

where:

- *convid* is a string identifier of the conversation obtained after starting a conversation.
- *w* is an optional watermark parameter to indicate the most recent message seen by the client.

The response will be a JSON object containing new activities (messages) from the user and the dialogue system:

```
{
   "activities": [
      {
         "type": "message",
         "channelId": "directline",
         "conversation": {
            "id": "abc123"
         },
         "id": "abc123|0001",
         "from": {
            "id": "bot1"
         },
         "text": "Nice to see you, user1!"
      }
   ],
   "watermark": "0001a-95"
}
```

Clients should page through the available activities by advancing the watermark value until no activities are returned.

3.2.3 Spoken language understanding API

A developer might want to implement dialogue management and text generation by himself, but does not want or has capability to develop spoken language understanding. In this case, a developer can use the Natural Language Understanding (NLU) API provided by Tilde.AI.

The mobile app might be interested in two particular functions of NLU API: intent detection and entity recognition. Full documentation of these and other NLU API functions can be found on the Tilde.AI developer portal[7]. However, in this document we will only provide a short description of intent detection and entity recognition functions.

Authentication

All Tilde.AI NLU API requests must contain a subscription key in the HTTP header that identifies a user. This key can be obtained after signing up on the Tilde.AI developer portal. Below is the header example of an HTTP request with an authorisation token:

```
GET https://dev-nlu1-am.azure-api.net/api/Train/languages
Ocp-Apim-Subscription-Key: subscription_key
```

Intent detection

The intent of the given text can be detected using the following POST request:

```
POST https://dev-nlu1-am.azure-api.net/api/Guess/AppId/LangId
Ocp-Apim-Subscription-Key: subscription_key
```

where:

- *AppId* is the app to be used to detect the intent. It is obtained via other API methods (see full documentation) or by using the COMPRISE SDK.
- *LangId* is the language of the text, e.g., “en-en”, “lv-lv”, etc.

The request body shall contain a text wrapped in double quotes.

[7] Tilde.AI developer portal https://dev-nlu1-am.portal.azure-api.net/docs/services/natural-language-understanding-nlu/operations/PostAddEntities

```
"Hello. Can you detect intent on this text."
```

The response will be a JSON object containing the detected intents sorted by descending confidence.

```
[
        { "intentid": "greeting", "confidence": 0.607079566 },
        { "intentid": "some_intent", "confidence": 0.261552066 },
        { "intentid": "some_intent_2", "confidence": 0.131368339 }
]
```

Entity recognition

The Tilde NLU API can be used to extract entities from a given text. Entities should be defined beforehand using specialised API functions (see full documentation) or by using the COMPRISE SDK. Entity recognition is available by performing an HTTP POST request:

```
POST https://dev-nlu1-am.azure-api.net/api/Entity/AppId/LangId
Ocp-Apim-Subscription-Key: subscription_key
```

where

- *AppId* is the app to be used for entity recognition. It is obtained via other API methods (see full documentation) or by using the COMPRISE SDK.
- *LangId* is the language of the text, e.g. “en-en”, “lv-lv”, etc.

The request body shall contain a text for which entity recognition should be performed:

```
"I would like to book a flight to London”
```

If the POST request is successful, the response will be a JSON object containing the detected entities and their position in the input.

```
[
        {
        "dim": "@is_alive_NoNoKnowldgeAPI",
        "body": "is_alive_NoNoKnowldgeApi",
        "value": { "value": "is_alive_NoNoKnowldgeAPI", "type": "value" },
        "start": 0,
        "end": 39

        }, {
        "dim": "@destination",
        "body": "destination",
        "value": { "value": "London", "type": "value" },
        "start": 33,
        "end": 39
        }
]
```

"is_alive_NoNoKnowldgeAPI" is an example entity which is hardcoded for every app for debugging.

4. Conclusion

This document presented the work performed in Work Package 3, specifically in tasks T3.1 and T3.2, to develop a research and technological foundation for the Initial Multilingual Interaction Library which will enable any user to interact with dialogue systems in any language. The achievement of this goal involves dealing with both research and development challenges.

A "synthetic data pipeline" approach was proposed to overcome the mismatch between the output of the STT and the expected input of the MT. The idea is to imitate typical errors in the STT output by processing the MT training with a TTS and STT pipeline.

Several MT systems were trained and evaluated. It was noticed that using synthetic data in MT training resulted in a clear benefit when translating STT output (we see gains about 1.5 BLEU). The best strategy is to train MT on both original clean and synthetic parallel data, and then the system will be able to perform well on both written text (traditional MT input) and STT output.

Although the project and the innovations developed therein have not reached completion yet, the evaluation has given us important insights. There are many open questions on how to prepare synthetic training data and it is also yet unclear how much we should ask the MT system to invent things that are not present in spoken language. In addition, these initial experiments do not try to solve the spoken language disfluency problem.

As COMPRISE develops, more methods of STT and MT integration will be explored and evaluated. These newer results will be included in deliverable D3.3 which will be submitted to the European Commission on January 31, 2021.

The second half of the deliverable describes the main software components that lay the foundation of the Multilingual Interaction Library: MT and dialogue management.

USAAR's dialogue platform Platon which was originally envisioned as the basis for the developments in COMPRISE is no longer maintained, therefore the consortium opted for TILDE's Cloud-based dialogue system (Tilde.AI) as an appropriate alternative. This document provides documentation for the main functions of the Tilde MT API and Tilde.AI Dialogue system. We present a short documentation on how to authenticate requests to these services, how to translate a given text, send user input to the dialogue system and receive replies, detect intents in a given text, etc. Full, up-to-date documentation of these APIs is provided online (links can be found in the corresponding sections of the document).

5. Bibliography

Bojar, O., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huck, M., . . . & Negri, M. (2016). Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers* (pp. 131-198).

Bojar, O., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huang, S., Huck, M., Koehn, P., Liu, Q., Logacheva, V., Monz, C., Negri, M., Post, M., Rubino, R., Specia, L., & Turchi, M. (2017). Findings of the 2017 Conference on Machine Translation (WMT17). In *Proceedings of the Second Conference on Machine Translation*, *Volume 2: Shared Task Papers* (pp. 169–214).

Simonnet, E., Ghannay, S., Camelin, N., & Estève, Y. (2018). Simulating ASR errors for training SLU systems. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation* (pp. 3157-3162).

Hassan, H., Schwartz, L., Hakkani-Tür, D., & Tür, G. (2014). Segmentation and disfluency removal for conversational speech translation. In *Proceedings of INTERSPEECH* (pp. 318–322).

Junczys-Dowmunt, M., Grundkiewicz, R., Dwojak, T., Hoang, H., Heafield, K., Neckermann, T., . . . & Martins, A. F. (2018). Marian: Fast Neural Machine Translation in C++. In *Proceedings of the 56th Annual Meeting of the ACL, System Demonstrations* (pp. 116-121).

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., . . . Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL, Interactive Poster and Demonstration Sessions* (pp. 177–180).

Sennrich, R., Haddow, B., & Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the ACL* (pp. 1715–1725).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998-6008).