Big speech data analytics for contact centers

BISON

http://bison-project.eu/

European Horizon 2020 project No. 645323

Deliverable D5.2

Indexing and database access to big speech data

Due date: 31/10/2016 **Submission date:** 31/10/2016
Project start date: 1/1/2015 **Duration:** 36 months
Lead beneficiary: BUT **Revision:** 1
Lead Author: Jan Černocký (BUT)
Contributors: Tomáš Bia (PHO), Marek Klimeš (PHO), Radim Kudla (PHO), Petr Valoušek (PHO), Bert Beernaert (MYF), Peter Gabriel (MYF)

Dissemination level	
CO: Confidential, only for members of the consortium (including the Commission Services)	
PU: Public	X

Summary

Information Mining for Business aims to perform the analysis of the information extracted from speech and transform it into relevant information for contact centers and their clients. Such a process needs to be supported by a clear structure, functional and easily integrable. This deliverable clarifies the main building stones of smallBison system with respect to API's and database structures. All key parts of the structure are outlined as well as a short insight into the future of the project in form of Voice Biometry Server, serving as a Bison part for Speaker Identification.

Table of contents

1. Introduction

While WP5 deals with mining speech Information for business, this deliverable was conceived and is presented as a description of intermediate layer between speech data mining and business analytics and user visualization, with accent on the actual integration done in smallBison.

This deliverable is interrelated tightly with both D4.1 and D4.2 since the problematic of data mining from speech is crucial for Bison. There is a number of dependencies to other deliverables outlined in the beginning of Chapter 3.

The two main components here are speech data analytics layer and contact center layer, that are communicating via a set of application programming interfaces (APIs) and database structures. Chapter 2 therefore concentrates on the speech API and presents information on the REST interface to speech technologies.

Chapter 3 details the use of databases and their structures in the upper "contact center" layer, presents its API and demonstrates the whole process on an example from a real user scenario. Both Chapter 2 and 3 contain links to detailed documentation elaborated by PHO and MYF, respectively.

Finally, Chapter 4 presents database structure and access to the Voice Biometry Server dealing with speaker recognition. While this server is not yet integrated into smallBison, work is in progress on this integration, mainly having in view scenarios that call both for content mining and person authentication.

2. Speech technologies to REST API interface

2.1 Speech Technologies

The Phonexia speech technologies are used for the project. Phonexia has provided speech technologies approachable via REST API that provides the access to its core technologies as a speech transcription or keyword spotting. The REST API structure was developed for an easier integration and better scalability. The general software architecture is shown in figure 1 below. The proposed system is divided into several modules, which allows to build variety of applications depending on the particular use case needs.
A functionality linked to the specific use-case is described in the Application Layer. An application interface for the integrator is provided. There are two important modules used in the project:

- Speech Analytics
- Voice Biometrics

The core speech technologies are encapsulated in the technology layer.

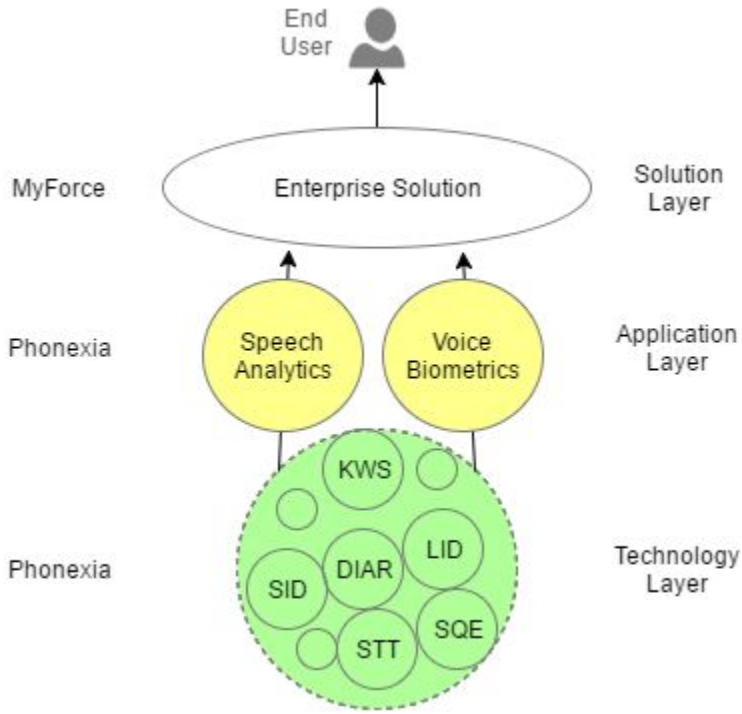


Figure 1: General architecture

2.2 Technology and Application Layer Interface

The speech technology is covered by Phonexia Speech Engine module. Phonexia Speech Engine interface provides access to operations via the URI path. The client application must create and submit

a request and then receive and analyze the response from the server. REST interface for entire communication protocol XML (or JSON) and standard HTTP methods GET, POST, DELETE and PUT is provided.

Commands are divided into two basic types - synchronous and asynchronous. When request for a synchronous operation is sent, response is always in the reply. In terms of an asynchronous request, an asynchronous request ID is sent back to the client. With this asynchronous request ID, client can monitor the state of asynchronous requests (Asynchronous request).

2.2.1 Response format

Response format can be specified in request header or in request query.

- Setting response format in request HTTP header in parameter Accept (allowed values are `application/json, application/xml, application/*, */*, *`). When asterisk notation is applied, JSON will be used. Bad value causes HTTP error 406 (not acceptable).
- Setting response format in request query in parameter format (allowed values are `json` or `xml`). Bad value causes HTTP error 400 (bad request)
  Settings in request query has higher priority. If neither query or accepted property is not set, the response format will be JSON.

2.2.2 Asynchronous Requests

Asynchronous request is used for time-consuming task processing. Each asynchronous request is marked with the asynchronous keyword in this documentation. Processing of an asynchronous request by client is as follows:

- Client sends an asynchronous request:
  `GET /technologies/speakerid?path=/recording.wav&speaker_model=davidid&model=`
- Server returns a 202 HTTP status to the client. In header of the response, using the "Location" parameter, server imparts URI, which should be queried by client, to find out the status of the asynchronous operation.
  If result of asynchronous request is already in cache, final reply (HTTP status 200) is returned instead of redirecting to `GET /pending/{id}`.
- Client repeatedly queries the status of the asynchronous operation until it obtains 303 HTTP status with URI of final result in "Location" parameter in header of the response. To obtain the latest asynchronous operation status, use pending request:
  `GET /pending/{id}`
- Client obtains the request result. To obtain the final asynchronous operation result, use done request:
  `GET /done/{id}`

The example of the result is as follows:

```
{
  "result": {
```

```
    "version": 2,
    "name": "SpeakerIdentificationMultiResult",
    "model": "S",
    "speaker_group": "",
    "calibration_set": "",
    "max_fa_rate": 0,
    "results": [
      {
        "file": "/recording.wav",
        "speaker_model": "david",
        "channel_scores": [
          {
            "channel": 0,
            "scores": [
              {
                "score": -13.1162815
              }
            ]
          }
        ]
      }
    ]
  }
}
```


- The result will be available for a limited time (by default 60 seconds, can be configured on server). Then the result will expire and requests `GET /done/{id}`, `GET /pending/{id}` with specific ID will return error (HTTP status 404). Result also will not be available after `GET /done/{id}` is successfully called.

2.2 Application Layer Interface

The Phonexia Speech Analytics application layer is provided for basic speech analytics use cases. The REST interface was also chosen for this layer of architecture. XML or JSON via standard HTTP methods GET, POST, DELETE and PUT are provided.

The functionality is provided via REST API and is divided into several logical groups:

- Sources
- Keyword groups and keywords
- Results
- Settings
- Scheduling
- Additional data for recording evaluation
- Application version
- Application codes and states
- Additional information

3. Database structures for accessing big speech data

As described in previous deliverables, the BISON architecture contains a lot of building blocks (GUI, windows services, and API's) that are interconnected to ensure the complete process of storing recordings, anonymizing information, sending speech processing commands from and to the speech processing engine, presenting results to the end user, etc. A schematic of all building blocks can be found in Figure 1 above, while a number of submitted Bison deliverables cover the individual building blocks:

- *D4.1 Initial speech mining* (M10) describes the initial speech mining technologies
- *D4.2 Optimizing speech data mining for CC operation* (delivered concurrently with this D5.2) outlines up-to-date versions of speech mining technologies.
- *D6.1 Architecture and API's* (M6) sets APIs and basic architecture for the current phase of the project.
- The text part of *D6.2 smallBison* (M12) describes the architecture of the first demonstrator that we are currently building on.
- *D6.3 User Data collection infrastructure* (M17) defines the ways user data was collected in Bison, which has implications to data handling.
- Finally, *D8.2 Legal, ethical and societal issues of BISON – General framework* (M12) and *D8.3 Legal, ethical and societal issues of BISON II* (M24, in preparation), define the legal and ethical framework for our development.

With regards to the topic of big data and automated processing, 3 building blocks are important here: Contact Center API, Speech Processing API and SQL Server (Database):

- **Speech Processing API** is a REST API, making sure that processing commands (such as "search for keyword variant X in this recording", "start transcription for this job/task", "tell me the talking speed of the agent for this recording", etc) are sent to/received from the speech processing engine
- **Contact Center API** is a REST API, enabling you to steer the complete BISON speech processing process from an external source or tool. Essentially, this means that all base functionality that has been made accessible in the BISON front-end GUI (Recording Manager) can also be steered externally.
- **SQL Server:** All speech processing information is stored separately into the Microsoft SQL Server database, and can be requested through the Contact Center API described above.

Both Contact Center API and underlying speech annotations database have been set up with intelligent third party data mining in mind:

- By default, an end user can set up key word groups and key word variants to spot using the GUI. Keyword variants can be linked to key word groups, which can in turn be linked to certain jobs and tasks (described in previous WP's).
- However, using the Contact Center API, external data mining tools can be used to extract all available speech processing information (eg spotted keywords on older recordings, or transcription information, stored as annotations in database), perform data mining tasks using advanced mining models, and automatically present new keyword variations to spot in future conversations through the same API.

- One can even go a step further and use all available information + REST API's to create data mining models that can actively search for recurring topics or themes in ongoing conversations (topic mining).

In conclusion, the openness of the BISON architecture easily allows building such connectors with data mining in mind.

In the following sections, we present the way the database is structured right now, and describe the most relevant parts of the Contact Center API. Each API call's detailed documentation can be found online, links are accessible through this deliverable.

3.1 SQL Server: statistics database

We chose to work with Microsoft SQL server as database technology, as this is widely adopted in the business world. Microsoft SQL technology has also proven to be a robust solution when handling huge amounts of data.

When it comes to storing speech processing data, the table Statistic_Annotation, stored within a general Statistics database, is the most important one. The structure is as follows:

Name	**Type**	**Description**
`StatAnnotationId`	Int	Primary key (autoincrement)
`CallId`	Int	General call identifier
`Source`	Int	Manual = 0, Auto = 1, Script = 2, Edit = 3
`Type`	Int	Anonym = 0, Trigger = 1, Crosstalk = 2, Speedproblem = 3, Transcription = 4, Keyword = 5, Custom = 6, Language = 7, Info = 8
`ChannelId`	Int	Audio channel index
`AgenIdCreator`	Int	User identifier of annotation creator
`AgentIdModifier`	Int	User identifier of annotation last modification
`StartTimeUTC`	Datetime	Start time of the annotation
`EndTimeUTC`	Datetime	End time of the annotation
`Deleted`	Bit	Annotation deleted flag
`Name`	Nvarchar(255)	Holds keyword in case of a keyword annotation, holds main value in case of other types
`Description`	Nvarchar(255)	Extra information for the annotation
`CreationTimeUTC`	Datetime	Creation time of the annotation
`LastModifiedTime UTC`	Datetime	Last modification time of annotation
`JobId`	Int	Job identifier (Job holds the definition)

JobInstanceId	Int	Job instance identifier (An instance can be run multiple times for the same job definition)
KeywordGroupId	Int	Keyword group identifier
IsProcessed	Bit	Annotations go through processing, this flag indicates the state of this processing
OccurenceType	Int	This field is a result of the processing, eg. Positive/negative flag for keyword annotations

3.2 Contact Center API

As said above, all information stored in the database described above can be requested using a REST API (Contact Center API). In this section, we will go into detail on the most relevant API calls that can be made with regards to data analysis. Each link below is clickable. For a full list of all API calls, please visit the on-line documentation[1].

3.2.1 Calls

Allows to fetch call information (such as call length) and/or speech data (such as key word, transcription, or other speech processing information)

GET	/Calls/{callId}/Annotations/{id}	
GET	/Calls/{callId}/Annotations	
PUT	/Calls/{callId}/Annotations/{id}	
POST	/Calls/{callId}/Annotations	
GET	/Calls/RecordingProcessingStatus	
GET	/Calls?FirstCallId={FirstCallId}&MaxResults={MaxResults}&Criteria={Criteria}	Get list of Calls (Find Call)
GET	/Calls/{id}	Get call details

3.2.2 Recordings

Allows to fetch the corresponding recording audio file, taking anonymization rules into account

GET	/Recordings/{callid}	Get recording

3.2.3 Speech Keyword Groups:

Very important for big data mining, as these API calls allow data mining tools to automatically update keyword groups based on analytics models etc…

[1] http://195.144.75.91/CcaWebAPI

GET	/SpeechKeywordGroups/{id}/UsedBy	Lists the ids of all SpeechJobs and SpeechKeywordGroups in which this SpeechKeywordGroup is used
GET	/SpeechKeywordGroups/Export	Export all keywordgroups to MS Excel. Specific keyword group ids can be exported by specifying a comma separated query parameter 'ids'
GET	/SpeechKeywordGroups/{id}/Export	Export specific keywordgroup to MS Excel. SpeechKeywordGroup id
POST	/SpeechKeywordGroups/Import	Import keywords
GET	/SpeechKeywordGroups	Get list of SpeechKeywordGroup
GET	/SpeechKeywordGroups/{id}	Get a specific SpeechKeywordGroup
POST	/SpeechKeywordGroups	Create an SpeechKeywordGroup
PUT	/SpeechKeywordGroups/{id}	Update SpeechKeywordGroup
DELETE	/SpeechKeywordGroups/{id}	Delete SpeechKeywordGroup

3.2.4 SpeechJobs:

Again very important for data mining, as these API calls allow the data mining software to automatically assign newly created key word groups to certain tasks/projects or jobs. Small example: “from now on, all calls related to project X need to search for a new key word group A, this at the start of each call. As the use of this key word in a conversation has a positive effect on the outcome of the call, this new key word group needs to be tagged as “positive”.

PUT	/SpeechJobs/{id}/Start	Start job
PUT	/SpeechJobs/{id}/Stop	Stop job
GET	/SpeechJobs/{id}/Instances	Get instances
GET	/SpeechJobs/Instances/{id}	Get instance
GET	/SpeechJobs	Get list of speechjobs
GET	/SpeechJobs/{id}	Get a specific SpeechJob
POST	/SpeechJobs	Create an SpeechJob
PUT	/SpeechJobs/{id}	Update SpeechJob
DELETE	/SpeechJobs/{id}	Delete SpeechJob

3.3 Use cases example

The way all components and database described above work together can be illustrated by a use case example: In this use case, a contact center Supervisor will create a new keyword group, fill it with

keyword variants, and create a "Job", allocating the newly created key word group to a selection of recordings (based on a certain task/project, or data range, or other variables). Finally, the Supervisor will start the job in order to initiate the process of processing all recordings (and getting results back in the GUI).

From a schematic point of view, this kind of operation works as described in the scheme in Figure 2. We will run through a couple of key steps (indicated with numbers in red circles):

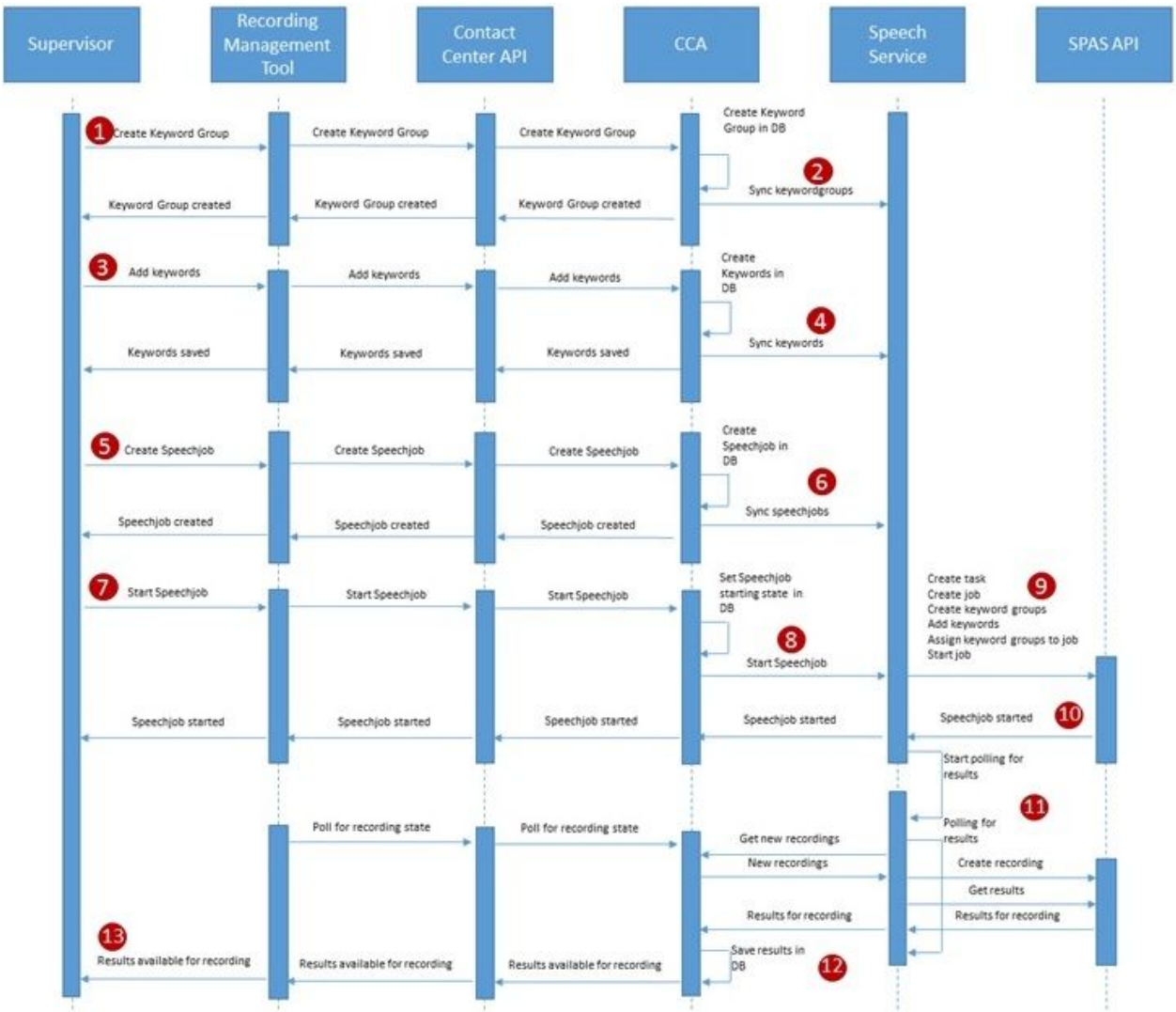


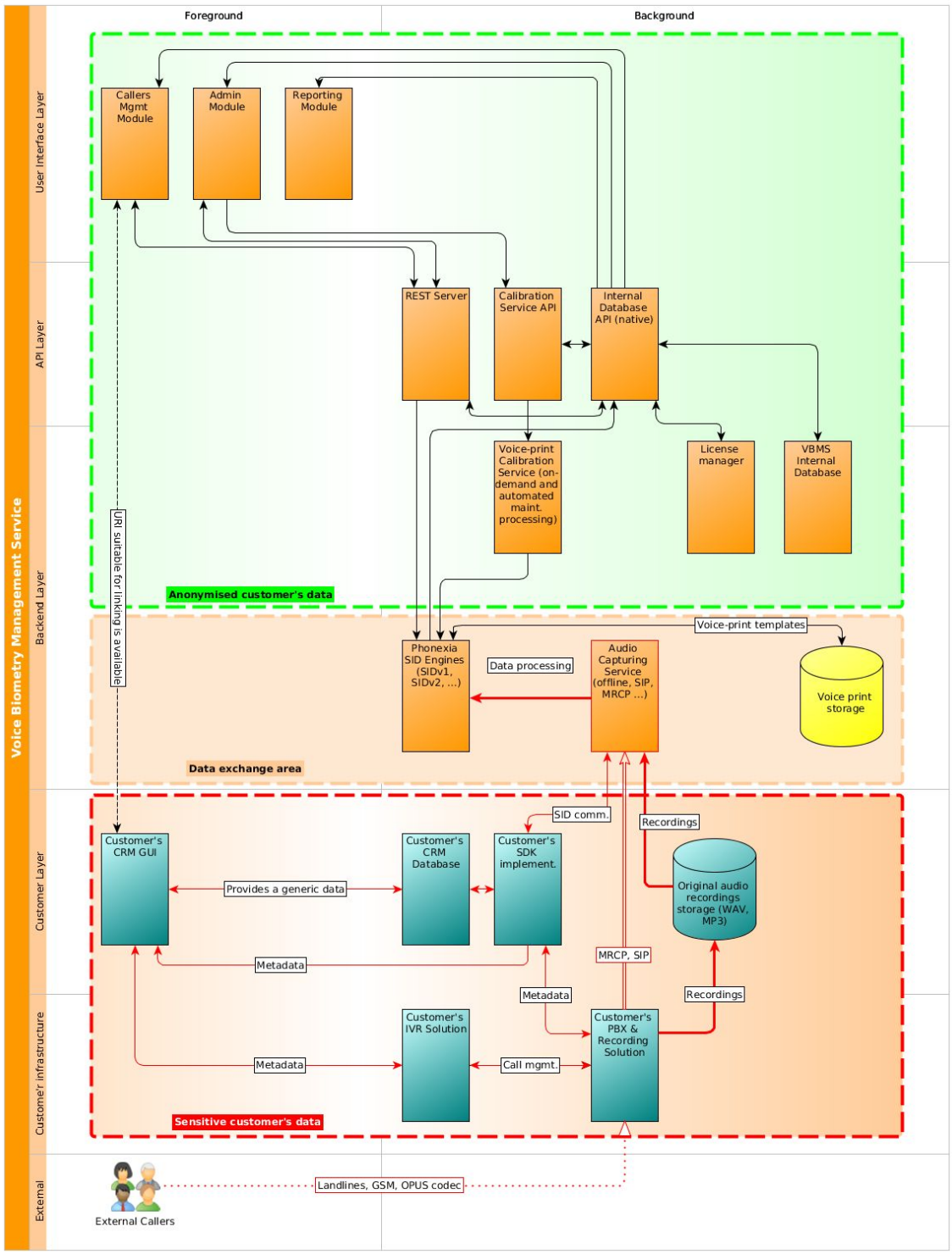
Figure 2: Scheme representing a contact center use-case.

In step **(1)**, the Supervisor creates a new keyword group using the Recording Management Tool. This action is passed from the Recording Management Tool onto the CCA (the core contact center management application) using the Contact Center API. CCA then instructs to create this new keyword group in Database, and syncs this new keyword group with the Speech Service **(2)**.
More or less the same flow happens for adding keyword variants to the newly created key word group (step **3** and **4)**, and for creating the Job **(**steps **5** and **6)**.

Finally, when the Supervisor has finished setting up the process, and decides to start the job **(7)**, the same flow is kept, with the difference that CCA not only instructs to write the starting state in DB, but also tells the Speech Service to Start the job **(8)**.
In step **(9)**, the Speech Service then instructs the Speech processing engine to Create a task, job, keyword groups and Add keywords + assign corresponding keyword groups to a job, using the SPAS API. Finally, the Start Job command is also passed onto the processing engine.
In step **(10)**, SPAS API gives feedback on the command, and alerts the speech service that the job is started. From now on, the Speech Service will start polling for results **(11)**. Each time new recordings are added, the Speech Service will pass that information onto the SPAS API and poll until the SPAS API feeds back speech processing results. Once results are available, they are stored into the Database **(12)**.
The moment this happens, these new results are also made available to the end user through the CCA à Contact Center API à Recording Management Tool **(13)**.

4. Database access to speaker recognition

The speaker identification technology (SID) is part of the Voice Biometrics (VBS) system. The VBS is based on several layers, while each layer can be secured according to rules of the customer. The VBS integration is shown in the following scheme:



The VBS architecture works with voice-prints (i-vectors in R&D language), see deliverable D4.1 for more thorough description of VP extraction and references). A VP is a digital representation of speaker's voice saved as biometric data into file or database. It does not carry information on the content of the message (this is suppressed in the steps of sufficient statistics extraction and several transformation layers) and as such, it can not contain any personal information. A VP can be compared against another VP(s) and this comparative results are used for the identification score calculation. We are aware that the processing of biometric data needs to be performed in compliance with applicable data protection law[2].

The key design properties are the following:

- Voiceprints (VP) are saved permanently (while VP is an already anonymized information).
- VBS can be deployed inside customer's ICT infrastructure (so customer can secure each layer according to his internal rules)
- No sensitive data are saved in the VBS system (audio/voice is converted to VP in secured layer of the VBS solution)

The database content is as follows:

- Data: VoicePrints only, never audio or speech content (this can be saved separately from VBS)
- Metadata saved with VP
    - Created as Timestamp
    - Revoked as Timestamp
    - Version of used SID model
    - Version of VP
- Other information saved
    - Watch-list configuration (given through API)
    - Calibration sets configuration

The Watch-list set is an organizational unit for setting virtual collection of Voiceprints from previously created Voiceprints.

The VBS system stores the identification score through the VBS to database for later analysis. The identification score can be also provided to the client's CRM (requires implementation work on client's side).

[2] For example, reference should be made in the information sheet provided to the data subject, consent should be acquired and suitable security measures should be adopted. Speaker identification and biometrics legal issues are examined in WP8.

5. Conclusions and outlook

WP5 aims at mining meaningful information from the business perspective. For the interconnection of speech data mining and these “higher layers”, respectively user use and visualization, clear APIs and database interfaces are required - these also allow to encapsulate the speech data mining, that is often considered “too complicated”,” too much math”, etc. into structures that are easily graspable by developers of other solutions.

The schemes of Phonexia API were outlined, as well as the infrastructure of the integrator, MyForce. Deeper insight in Contact Center API was provided by Chapter 3.2 and its subsections. A reader gains a broader image about all components and database in practical use case, demonstrated on a use-case. A future of speaker identification in Bison project is described in the form of Voice Biometry server interfaces.

In the future development of Bison demonstrators (new versions of smallBison and forthcoming bigBison), APIs and database structures will continue to play an important role. As this is the last deliverable describing such components, the “delta” compared to the current state will be included in the report accompanying D6.4 - the “bigBison” demonstrator in Month 33 of the project.