
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n: 761999

# EasyTV: Easing the access of Europeans with disabilities to converging media and content.

# D3.6 – EasyTV Remote control with speech recognition final development

**EasyTV Project**

*H2020. ICT-19-2017 Media and content convergence. – IA Innovation action.*

**Grant Agreement n°: 761999**

Start date of project: 1 Oct. 2017

Duration: 30 months

Document. ref.: Deliverable 3.6

## Disclaimer



The EasyTV Consortium consists of the following partners:

| | Partner Name | Short name | Country |
|---|---|---|---|
| 1 | Universidad Politécnica de Madrid | UPM | ES |
| 2 | Engineering Ingegneria Informatica S.P.A. | ENG | IT |
| 3 | Centre for Research and Technology Hellas/Information Technologies Institute | CERTH | GR |
| 4 | Mediavoice SRL | MV | IT |
| 5 | Universitat Autònoma Barcelona | UAB | ES |
| 6 | Corporació Catalana de Mitjans Audiovisuals SA | CCMA | ES |
| 7 | ARX.NET SA | ARX | GR |
| 8 | Fundación Confederación Nacional Sordos España para la supresión de barreras de comunicación | FCNSE | ES |
| 9 | Unione Italiana dei Ciechi e degli Ipovedenti | UICI | IT |

| | |
|---|---|
| **PROGRAMME NAME:** | H2020. ICT-19-2017 Media and content convergence - IA Innovation action |
| **PROJECT NUMBER:** | 761999 |
| **PROJECT TITLE:** | EASYTV |
| **RESPONSIBLE UNIT:** | MV |
| **INVOLVED UNITS:** | ENG, MV, ARX |
| **DOCUMENT NUMBER:** | D3.6 |
| **DOCUMENT TITLE:** | EasyTV Remote control with speech recognition final development |
| **WORK-PACKAGE:** | WP3 |
| **DELIVERABLE TYPE:** | Report |
| **CONTRACTUAL DATE OF DELIVERY:** | 31-10-2019 |
| **LAST UPDATE:** | 25-10-2019 |
| **DISTRIBUTION LEVEL:** | PU |

**Distribution level:**

**PU** = *Public,*

**RE** = *Restricted to a group of the specified Consortium,*

**PP** = *Restricted to other program participants (including Commission Services),*

**CO** = *Confidential, only for members of the LASIE Consortium (including the Commission Services)*

# Document History

| VERSION | DATE | STATUS | AUTHORS, REVIEWER | DESCRIPTION |
|---|---|---|---|---|
| V1.0 | 15/09/2018 | Draft | MV/ENG | First draft<br>Table of Contents<br>definition and<br>document structure |
| V2.0 | 26/10/2018 | Draft | MV/ARX | Added Dialog analysis for tv app domain |
| V3.0 | 12/11/2018 | Draft | MV | Added Speech Technology state of the art and first draft of EasyTV speech platform |
| V4.0 | 20/11/2018 | Release Candidate | MV/ENG | Completed EasyTV speech platform part with RNN and prototype software component descriptions |
| V5.0 | 28/11/2018 | Reviewed | CCMA/UPM | Reviewed By CCMA/UPM |
| V6.0 | 28/11/2018 | Correction | MV | First revision |
| V6.1 | 29/11/2019 | Completed | MV | Final Revision |
| V6.2 | 08/10/2019 | Draft | MV | Added the first description of the final architecture and SDK and speech personal voice assistant considerations |
| V6.3 | 14/10/2019 | Draft | MV | Completed the description of the architecture and SDK |
| V6.4 | 15/10/2019 | Draft | MV | CSApp VUI chapter and reviewed executive summary and conclusions |
| V6.5 | 21/10/2019 | Release Candidate | MV | Minor fixes and general review |
| V6.6 | 25/10/2019 | Final Release | MV | Included corrections from peer reviewers. |

# Definitions, Acronyms and Abbreviations

| ACRONYMS / ABBREVIATIONS | DESCRIPTION |
|---|---|
| API | Application programming interface |
| ASR | Automatic Speech Recognition |
| OS | Operating System |
| DoW | Description of Work |
| DSP | Data Signal Processing |
| DSR | Distributed Speech Recognition |
| EPG | Electronic Program Guide |
| FFNN | Feed Forward Neural Network |
| HBBTV | Hybrid broadcast broadband TV |
| CSApp | Companion Screen Application |
| HTML5 | HyperText Markup Language |
| ID | Identifier |
| IT | Information Technology |
| IVR | Interactive Voice Response |
| LSTM | Long Short Term Memory |
| RNN | Recurrent Neural Network |
| SRGS | Speech Recognition Grammar Specification |
| SRSI | Semantic Interpretation for Speech Recognition |
| SSML | Speech Synthesis Markup Language |
| TTS | Text To Speech |
| VOD | Video On Demand |
| VT | Voice Template |
| VUI | Voice User Interface |
| GUI | Graphical User Interface |
| W3C | World Wide Web Consortium |

# Table of Contents

# 1. LIST OF FIGURES

# 2. LIST OF TABLES

# 3. EXECUTIVE SUMMARY

In this deliverable we describe the final version of the EasyTV Speech Platform as described in T3.3 and specifically considering the following:

- the project objectives, presented in the EasyTV Description of Work (DoW),
- the user requirements, presented in the deliverable D1.1 [2] (User scenario and requirements definition),
- the system specifications, defined in D1.2 [3] (EasyTV system requirements specifications)
- the system architecture defined in D1.4 [5] (Final release of the EasyTV system architecture).

In the first release of the document we described the work done for the first prototype in order to experiment the technology using mobile devices and platforms for second screen applications and for testing Neural Networks in the process of voice command interpretations.

In chapter 4 we describe the general architecture of the EasyTV Speech Platform and all the components implemented in the first prototype. We also describe the implementation of the first prototype on the Android OS and also the semantic Interpretation of voice commands modules developed using the RNN (Recurrent Neural Network) technique.

In chapter 5 we report the dialog analysis of EasyTV App domain and describe the dialogs and conversations between end users and EasyTV applications and services. In the Dialog analysis of the EasyTV App Domain we have considered the following applications do define their Voice User Interfaces (VUI).

- Video On Demand catalog
- LiveTV
- EPG (Electronic Program Guide)
- Manage Recordings (Scheduled and Recorded)
- Video Player Controller

In chapters 6, we describe the activities and the work done to complete and develop the final version of the EasyTV Speech Platform SDK (ESP-SDK).

We describe first the final internal architecture of the software components and report the description of the SDK (Software Developer Kit) used to integrate any VUI (Voice User Interface) to the EasyTV application domain. All the functionalities of the SDK elements and interfaces are reported and a description of the whole life cycle of a vocal interaction is described.

In chapter 7, we report a sample of the CSApp application flow and the design of its Voice User Interface in order to support the implementation of the final voice interaction system and prepare the prototype to test with final users.

In chapter 8, we also report some considerations about the voice assistants and the popular smart speaker devices, usually used to access their functionalities (i.e. Amazon Alexa and Google Home), in order to understand the positioning of our EasyTV Speech Platform SDK related to the architectures and business models that they provide.

Finally, to better understand the EasyTV Speech Platform components we also reported in the “Addendum - Speech Technology” a summary of the state of the art of Speech Technology. In this addendum we talk about Speech Recognition systems, both Dictation and Command and Control as well as local and remote systems (Cloud Based ASR and Distributed ones). Features and architecture of Text To Speech Engine are also reported in the addendum.

In the “” we report a simplified version of internal guideline to define VUIs and their representation to be used for the final prototype.

# 4. EASYTV SPEECH PLATFORM

## 4.1. Introduction

In this chapter we describe the main components of the EasyTV speech platform and its first prototype that we implemented in the first six months of the project.

## 4.2. General Architecture

We started analyzing the architecture defined in the deliverable 1.4 in order to define the components to implement and the main interface for their interaction

As we already reported in the final release of the system architecture the Speech Platform is part of the Universal Remote component of EasyTV project. This component will be available using a tablet or a smartphone device when users consume content on a Second Screen application. Blind and visual impaired users will interact with the platform and services using audio microphone and speakers available on the client device.

The Speech Interface Architecture will be based on existing technology and enhanced by exploiting the latest advances in Natural Language Processing as described later in this document.

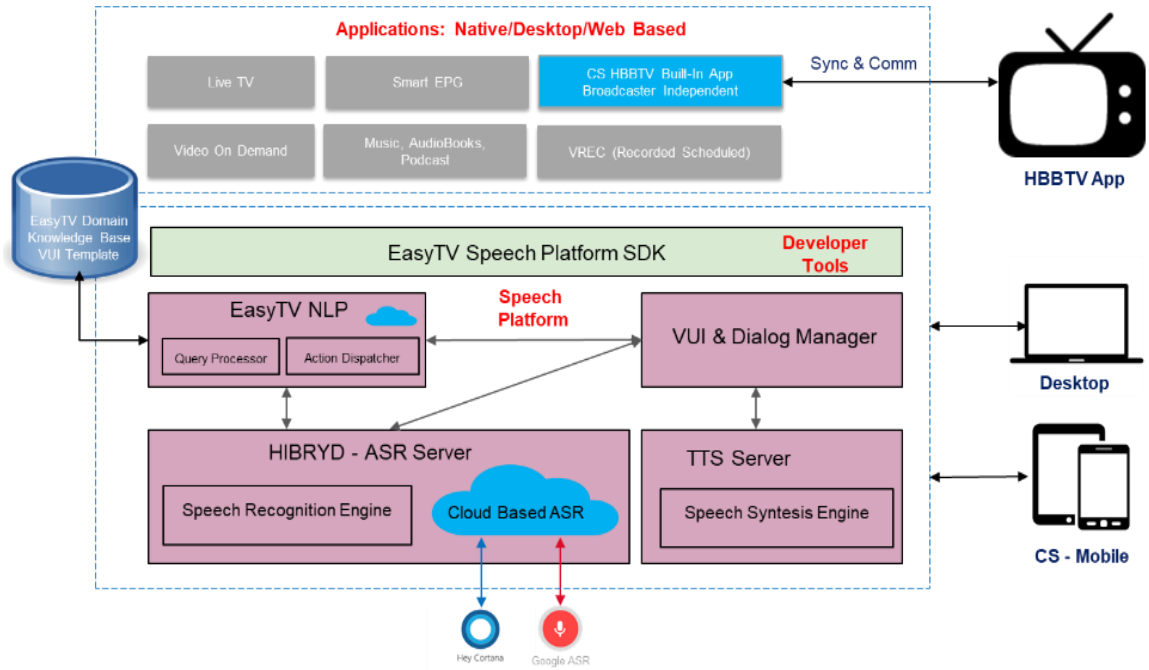



**Figure 1: General Architecture of EasyTV Speech Platform**

The EasyTV Speech Platform components as depicted in the Figure 1 will be the following:

- **Hybrid ASR Server**: this component manages the Local Speech Recognition Engine on the client device and the remote Speech Recognition Engine on the cloud. The speech platform will be able to manage both local and remote ASR depending on the functionality that is going to be used by the application and the network availability.

- **TTS (Text To Speech):** this component manages the speech synthesis engine which will be included in the client device. It will be able to manage the voice to use, the language and its volume and speed.
- **EasyTV NLP**: The Natural Language Processing Component of the speech interface will process the voice query of the user and will understand the user intent. It will also manage the subsequent communication with the Dialog Manager based on a reasoning and planning process. It will also dispatch the actions and receive events from the EasyTV SDK components to interact with the TV based application.
- **VUI and Dialog Manager**: this component is responsible to process the dialog flow between the user and the application through pre-defined voice dialog templates, which include voice prompt templates, semantic annotations for user actions and dialog flow templates. The Dialog Manager of the EasyTV Speech Platform will be implemented using both the Semantic interpretation component and a dialog management system that will consider the dialog states and templates defined in the EasyTV app domain analysis. When the user interacts with the EasyTV applications using voice commands, the Speech Platform will interpret the user intent and classify the user input in order to define which actions to take and how the dialog should progress according to the dialog flow designed for the specific application and its functionalities. The Speech Platform will keep track of the dialog state and will execute the appropriate actions based on the state and the contextual user input.
- The **EasyTV Domain Knowledge Base and VUI Template** is the repository of the EasyTV ontology and data available for the NLP component as described above.

On top of the Speech platform we will implement a specific SDK (Voice Software Developer Kit) which will enable developers to add voice interface to HBBTV/HTML5 based applications other than native applications developed for EasyTV users. Both Speech Platform and SDKs will be available for developing any voice enabled application and will run on the client device.

The application level module is the higher-level part of the architecture and includes all the EasyTV application set. EasyTV applications can be implemented in any native device language and of course in HBBTV/HTML5 technology. Applications will interact with the speech platform using a common communication protocol and technology that is native code for native applications, the WebView API interface for embedded HTML5 applications or WebSocket Technology for any other HTML5 application running on a Web Browser (Chrome, Edge, Mozilla, etc.) on any terminal, including HBBTV.

The Speech Interface component will be able to control and access all the main functionalities of the Universal HTML5 Player (as described above), for both audiovisual and accessible content.

In the next paragraphs we describe the first implementation of the EasyTV Speech Platform using Android devices and technology.

## 4.3. EasyTV Speech Platform with Android

The first implementation of the EasyTV Speech Platform has been implemented and experimented using the Android OS. We defined all the components for the upcoming Easy Speech Platform SDK that will be implemented in the next year and we also implemented the first prototype based on the components we defined.

In this paragraph we describe all these components and their interfaces to be used for the implementation of the EasyTV Speech Platform SDK.

The Automatic Speech Recognition System (ASR), the text interpretation (NLP) and the Text To Speech System (TTS) will consist of three distinct packages plus the Animated Button needed to activate and deactivate the voice user input and speech processing.

- EasyTVASR
- EasyTVTTS

- EasyTVNLP
- AnimatedButton

The three packages will be inside the first level *com.EasyTV.mediavoice.libs* package.

The EasyTVASR package contain the necessary classes and views for configuring Google's speech recognition engines in the cloud and locally on the device, for displaying the Blind Mode layer and for retrieving the results obtained from the speech recognition engine.

The EasyTVTTS package contain the classes necessary for the configuration and use of the synthesis engines and for the management of the notifications that the engine itself will arise.

The EasyTVNLP package contain the classes and interfaces necessary to interpret the text recognized by the Google speech recognition engine.

The AnimatedButton package contain the VoiceView control, that is the activation button of the recognition. The button is animated by a touch gesture and to gives a visual feedback on the intensity of the speaker's voice.

## 4.4. Speech Recognition Package

The EasyTVASR package contains the EasyTVASRView and EasyTVSpeechRecognizer classes.

The EasyTVASRView class implements the display of the BlindMode layout. The BlindMode is a particular layout that allows the phone screen to be touched anywhere to activate the recognition engine. When the BlindMode is not active, the application will display the VoiceView button in a precise part of the screen, this will have to be pressed to activate the voice recognition. For a blind person it is very difficult to be able to touch this button if not through the help of the TalkBack screen reader. Activating the BlindMode, makes it possible for the blind to touch any point on the screen, without the need for TalkBack, and activate the speech recognition engine.


**Figure 2: Speech Recognition System – EasyTVASRView Class**

EasyTVSpeechRecognizer is the class responsible for managing the Google speech recognition engine both in the cloud and locally. In the EasyTVSpeechRecognizer class there is all the necessary code to configure the recognition engine, to activate and deactivate it, to receive engine notifications and to handle errors. All these functionalities are encapsulated and simplified in few methods and

events and are presented to the final product developer, who will not have to worry about all the technical aspects connected to the management of the recognition engine.

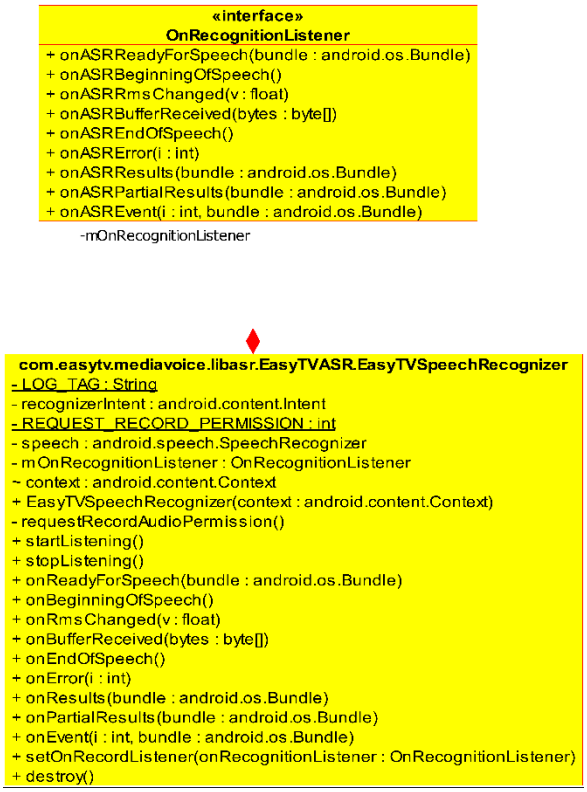



**Figure 3: Speech Recognition System - EasyTVSpeechRecognizer Class**

The public methods of this class are the following:

- Class constructor: initializes the engine and checks and / or requests the necessary permissions for operating of the vocal engine
- void startListening (): activates the speech recognition
- void stopListening (): stops the speech recognition
- Class Destroyer: deals with correctly terminating the active instance of the recognition engine and freeing up the occupied resources. The events returned by the class are defined in the OnRecognitionListner interface:
- void onASRReadyForSpeech (Bundle bundle): Notifies that the engine is instantiated correctly and is ready for recognition.
- void onASRBeginningOfSpeech (): notifes that the voice input has started
- void onASRRmsChanged (float v): notifies in db the level of the voice that is speaking
- void onASRBufferReceived (byte [] bytes): gives back a buffer containing the audio
- void onASREndOfSpeech (): notifies that the voice input is terminated
- void onASRError (int i): notifies that an error has occurred during the recognition
- void onASRResults (Bundle bundle): gives back the totality of various recognized text hypotheses
- void onASRPartialResults (Bundle bundle): Notifies the partial result during the input phase
- void onASREvent (int i, Bundle bundle): reserved for future developments

## 4.5. Speech Synthesis Package

The EasyTVTTS package contains the EasyTVTTS class.

The EasyTVTTS class deals with the management of the speech synthesis engine in the language chosen by the end user. In the EasyTVTTS class there is all the necessary code for the configuration of the synthesis engine, its activation and deactivation, engine notifications and errors and the transformation of the text in vocal synthesis.


**Figure 4: Speech Synthesis System - EasyTVTTS Class**

The public methods of this class are the following:

- void initTTS (Local Local): initializes the synthesis engine in the language indicated "local"
- void installTTS (Local locale): install the voice engine in the language indicated "local" on the device
- void speech (String text): dispenses the text tts by interrupting the currently delivered tts
- void queue_speech (String text): Queues the text up to the currently delivered tts
- int setLanguage (Locale loc): sets the language in which the text will be delivered
- void stop (): stops the synthesis engine
- void shutdown (): closes the active instance of the synthesis engine
- boolean isSpeaking (): returns true if the synthesis engine is speaking

The events returned by the class are defined in the OnUtteranceProgressListener interface as follow:

- void onTTSStop (String utteranceId, boolean interrupted): the synthesis engine has been stopped
- void onTTSStart (final String text): the synthesis engine has started talking
- void onTTSDone (String text): the synthesis engine has finished delivering the text
- void onTTSError (String s): Notifies that an error occurred while delivering the text
- void onTTSRangeStart (final String utteranceId, int start, int end, int frame): notifies the useful information to highlight the text pronounced by the synthesis engine

## 4.6. Component for the Semantic Interpretation of Voice Commands

The EasyTVNLP package contains everything needed to manage the interpretation of the text obtained by the speech recognition engine. Other than minor sub packages it contains two main sub-packages:

- Classifier
- Recognizer

Package **Classifier**

The Classifier package contains the Classifier interface and the classes that implement it. It's the package with the classes that semantically interpret the text acquired from the speech recognition engine.

The Classifier interface is the interface to be implemented to interact with different text classifiers.

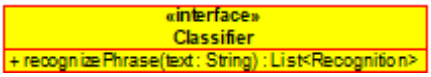



**Figure 5: EasyTV NLP Module - Classifier Interface**

The Classifier interface contains a single method to be implemented:

- List <Recognition> recognizePhrase (String text): returns a list of Recognition objects containing the classifier results

Text classifiers are classes that implement the Classifier interface and are based on two different interpretations:

- Interpretation through regular expressions
- Interpretation by inference of an appropriately trained neural network

The classes interpreting the text using regular expressions utilize a series of specific patterns encoded by regular expressions. When the text obtained from the recognition engine is given as input to one of these classes, the match is made with respect to the various patterns encoded in the class. If a positive match is obtained, the values of interest are extracted from the sentence and returned as a final interpretation.

The classes interpreting the text using the inference of a neural network, utilize within them a neural class model appropriately trained to recognize and classify the text obtained as input from the speech recognition engine. If the confidence returned by the neural network obtained in the inference process is higher than a certain threshold, then the input text is considered to be associated with a given intent and is returned as a valid result.

Package **Recognizer**

The package **Recognizer** contains the Recognizer interface and the classes that implement it. It is also the package in which the classes using the various classifiers in a certain context of recognition are located.

The classes that implement the Recognizer interface are responsible for orchestrating the various objects that implement the Classifier interface in order to obtain valid results in the classification of the input text.

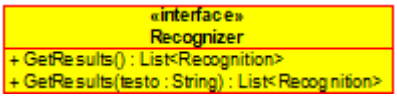



**Figure 6: EasyTV NLP Module - Recognition Interface**

In the Recognizer package there is the **Recognition** class which represents the single result of one of the classifiers.

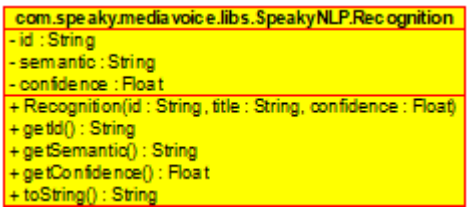



**Figure 7: EasyTV NLP Module - Recognition Class**

The Recognition class contains three properties:

- Id: it is a unique id of the object instance
- Semantic: contains the semantics returned from the classifier result
- Confidence: contains a confidence value for the obtained semantics

## 4.7. Animated Button Package

The **AnimedButton** package contains the VoiceView that allows the visualisation of the single animated input button, also by standing in the BlindMode layer described above. This custom control draws a button that once touched gets animated based on the intensity of the user's voice as a kind of vumeter.

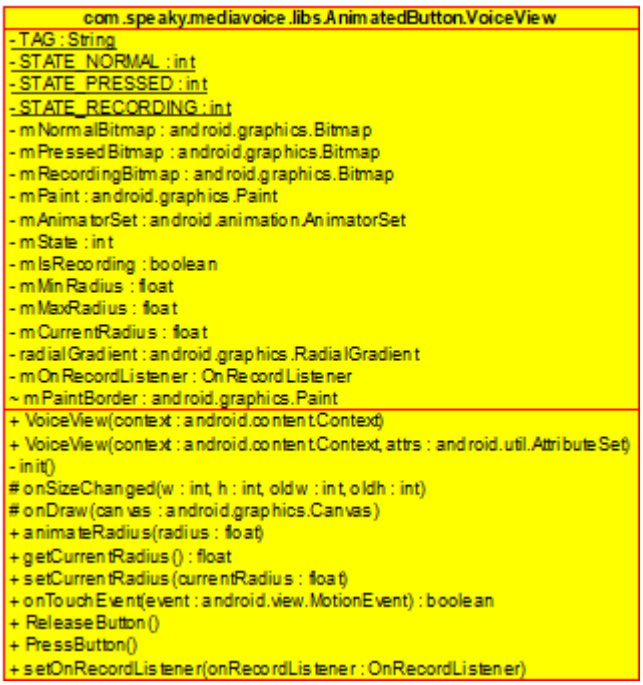



**Figure 8: EasyTV Animated Button for Blind Mode**

The setCurrentRadius function is fundamental for the correct drawing of the button animation in which the value returned by the OnRecognitionListner listener is passed on to the onASRRmsChanged event described above.

## 4.8. Recurrent Neural Networks

The NLP component of the EasyTV Speech Platform defined for the interpretation of the voice commands uses a Recurrent Neural Network (RNN) with a layer of type LSTM (long short term memory).
Recurrent Neural Networks (RNN) are a powerful and robust type of neural networks and belong to the most promising algorithms out there now because they are the only ones with an internal memory.
RNNs are relatively old, like many other deep learning algorithms. They were initially created in the 1980's, but can only show their real potential since a few years, because of the increase in available computational power, the massive amounts of data that we have nowadays and the invention of LSTM in the 1990's.
Because of their internal memory, RNNs can remember important things about the input they received, which enables them to be very precise in predicting what's coming next.
This is the reason why they are the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more because they can form a much deeper understanding of a sequence and its context, compared to other algorithms.

### 4.8.1. How RNNs work

To understand Recurrent Neural Networks, we can start from Feed Forward Neural Networks, properly.

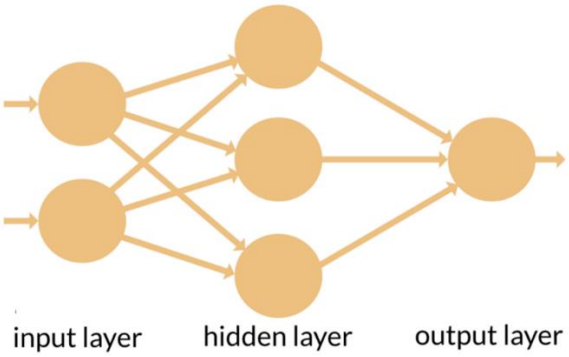



**Figure 9: Feed-Forward Neural Networks**

RNNs and Feed-Forward Neural Networks are both named after the way they channel information.
In a Feed-Forward neural network, the information only moves in one direction, from the input layer, through the hidden layers, to the output layer. The information moves straight through the network. Because of that, the information never touches a node twice.
Feed-Forward Neural Networks have no memory of the input they received previously and are therefore bad in predicting what's coming next. Because a feedforward network only considers the current input, it has no notion of order in time. They simply can't remember anything about what happened in the past, except their training.

### 4.8.2. Recurrent Neural Networks

In an RNN, the information cycles through a loop. When it makes a decision, it takes into consideration the current input and also what it has learned from the inputs it received previously.
The two images below illustrate the difference in the information flow between an RNN and a Feed-Forward Neural Network.

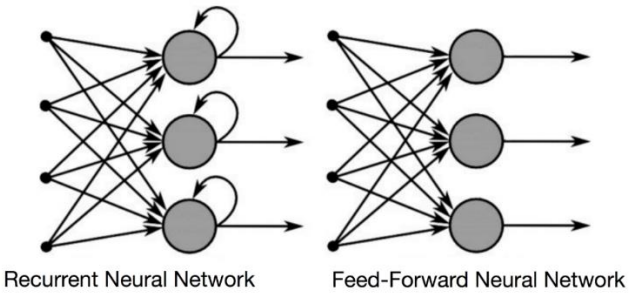



**Figure 10: Difference in the information flow between an RNN and a FFNN**

A usual RNN has a short-term memory. In combination with a LSTM they also have a long-term memory.
Another good way to illustrate the concept of a RNNs memory is to explain it with an example:
Imagine you have a normal feed-forward neural network and give it the word “neuron” as an input and it processes the word character by character. At the time it reaches the character “r”, it has already forgotten about “n”, “e” and “u”, which makes it almost impossible for this type of neural network to predict what character would come next.
A Recurrent Neural Network is able to remember exactly that, because of its internal memory. It produces output, copies that output and loops it back into the network.

### 4.8.3. Recurrent Neural Networks add the immediate past to the present.

Therefore, a Recurrent Neural Network has two inputs, the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why an RNN can do things other algorithms can’t.
A Feed-Forward Neural Network assigns, like all other Deep Learning algorithms, a weight matrix to its inputs and then produces the output. Note that RNNs apply weights to the current and to the previous input. Furthermore, they also tweak their weights for both through gradient descent and Backpropagation Through Time, which we will discuss in the next section below.
Also note that while Feed-Forward Neural Networks map one input to one output, RNNs can map one to many, many to many (translation) and many to one (classifying a voice).

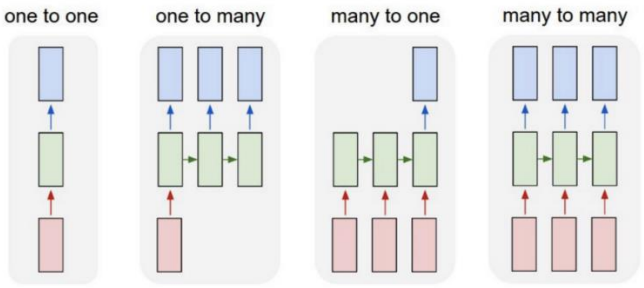



**Figure 11: RNN and FFNN input output mapping**

### 4.8.4. Backpropagation Through Time

In neural networks, we basically do Forward-Propagation to get the output of our model and check if this output is correct or incorrect, to get the error.
Now we do Backward-Propagation, which is nothing but going backwards through our neural network to find the partial derivatives of the error with respect to the weights, which enables you to subtract this value from the weights.
Those derivatives are then used by Gradient Descent, an algorithm that is used to iteratively minimize a given function. Then it adjusts the weights up or down, depending on which decreases the error. That is exactly how a Neural Network learns during the training process.
So, with Backpropagation we basically try to tweak the weights of our model, while training.
The image below illustrates the concept of Forward Propagation and Backward Propagation perfectly at the example of a Feed Forward Neural Network:

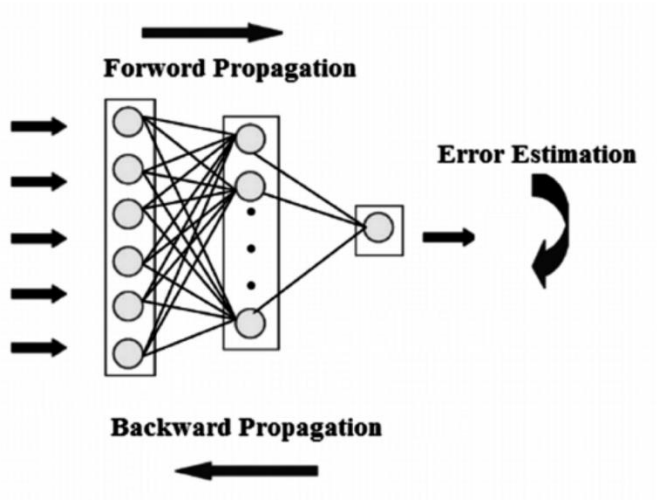



**Figure 12: Forward Propagation and Backward Propagation of FFNN**

We can view an RNN as a sequence of Neural Networks that we train one after another with backpropagation.
The Figure 13 illustrates an unrolled RNN. On the left, we can see the RNN, which is unrolled after the equal sign. Note that there is no cycle after the equal sign since the different timesteps are visualized and information gets passed from one timestep to the next. This illustration also shows why an RNN can be seen as a sequence of Neural Networks.

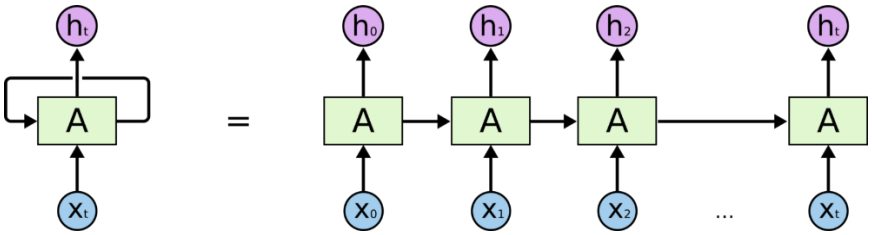



**Figure 13: RNN as a sequence of Neural Networks**

If we do Backpropagation Through Time, it is required to do the conceptualization of unrolling, since the error of a given timestep depends on the previous timestep.
Within Backpropagation Through Time (BPTT) the error is back-propagated from the last to the first timestep, while unrolling all the timesteps. This allows calculating the error for each timestep, which allows updating the weights. Note that BPTT can be computationally expensive when you have a high number of timesteps.

- **Issues of standard RNNs:** A gradient measures how much the output of a function changes, if we change the inputs a little bit.
- **Exploding Gradients:** We speak of "Exploding Gradients" when the algorithm assigns a high importance to the weights, without much reason. But fortunately, this problem can be easily solved if you truncate or squash the gradients.
- **Vanishing Gradients:** We speak of "Vanishing Gradients" when the values of a gradient are too small, and the model stops learning or takes way too long because of that. It was solved through the concept of LSTM by Sepp Hochreiter and Juergen Schmidhuber.
- **Long-Short Term Memory:** Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks, which basically extends their memory. Therefore, it is well suited to learn from important experiences that have very long time lags in between.

The units of an LSTM are used as building units for the layers of an RNN, which is then often called an LSTM network.
LSTMs enable RNNs to remember their inputs over a long period of time. This is because LSTM's contain their information in a memory, that is much like the memory of a computer because the LSTM can read, write and delete information from its memory.
This memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information (e.g. if it opens the gates or not), based on the importance it assigns to the information. The assigning of importance happens through weights, which are also learned by the algorithm. This simply means that it learns over time which information is important and which not.
In an LSTM you have three gates: input, forget and output gate. These gates determine whether to let new input in (input gate), delete the information because it isn't important (forget gate) or to let it impact the output at the current time step (output gate). You can see an illustration of an RNN with its three gates below:

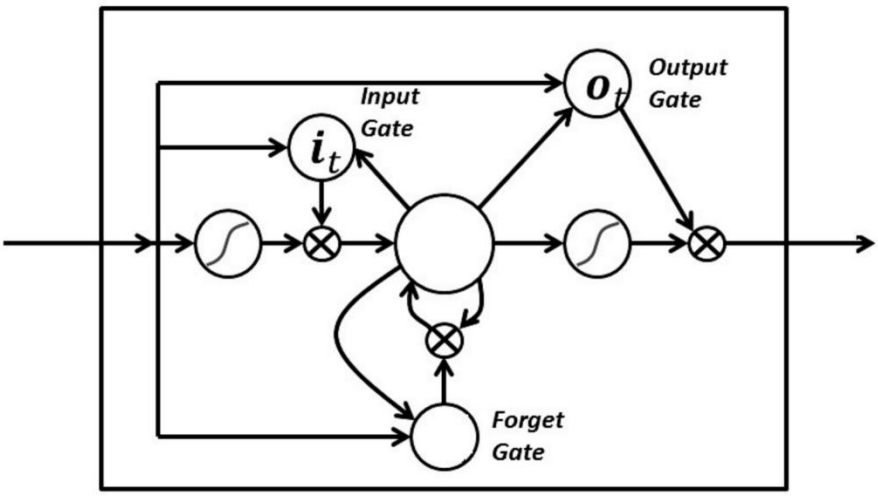



**Figure 14: RNN with its three gates**

The gates in a LSTM are analog, in the form of sigmoids, meaning that they range from 0 to 1. The fact that they are analog, enables them to do backpropagation with it.
The problematic issues of vanishing gradients are solved through LSTM because it keeps the gradients steep enough and therefore the training relatively short and the accuracy high.

## 4.9. EasyTV Speech Platform neural network model

The EasyTV Speech Platform model consists of 4 layers with a sigmoid activation function:

- Embedding layer
- Spatial dropout layer
- LSTM layer
- Dense layer

We have built the neural network using Google Tensorflow and Keras libraries [12] [13] In the Following snippet we show the network definition:

```
model.add(Embedding(max_fatures, embed_dim,weights=[embedding_matrix], trainable = True,
input_length = len(train_x[0])))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(lstm_out, recurrent_dropout=0.2, dropout=0.2))
model.add(Dense(len(train_y[0])))
model.add(Activation('sigmoid'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Input data are defined as a JSON structure in which we define our conversional intents:

```
{"intents": [
       {"tag": "greeting",
         "patterns": [
           "ok easy tv",
           "ciao easy tv",
           "come va easy tv",
           "salve easy tv",
           "buongiorno easy tv",
           "easy tv mi aiuti",
           "easy tv ho bisogno di una info",
           "ciao easy tv",
           "come va easy tv",
        ],
       },
       {"tag": "thanks",
        "patterns": [
           "grazie",
           "mille grazie",
           "grazie mille",
           "sei stato di grande aiuto",
           "grazie delle informazioni",
           "grazie per avermi aiutato",
           "grazie del tuo aiuto",
           "grazie della risposta",
           "ho capito grazie mille"],

       },
……
```

"tag" represents the intent category and is the network output and "patterns" represents the input phrase that neural network learns to categorize.
Input phrases and output intent category have to be transformed before to be passed as input and output to the neural network.
First of all, each sentence is transformed in a list of stemmed words and each phrase is associated with an intent (the "tag"). Using bag of words technique we transform each phrase in a numeric array that can be passed as input to the network and every intent associated to the phrase is transformed in array that can be passed as output value to the network.

### 4.9.1. **Training Phase**

Once input and output data are prepared the training process can start:

```
cbks = [callbacks.ModelCheckpoint(filepath=fname_train, monitor='acc', save_best_only = True, verbose=1, mode = 'max'),
callbacks.EarlyStopping(monitor='acc', patience = 1000)]#patience = number of ages without improvements after which the training will stop

model.fit(np.array(train_x), np.array(train_y), epochs = 1000, batch_size=batch_size, verbose = 1, callbacks = cbks, validation_split = 0.05, shuffle = True)
```

At the end of training process, we obtain a model that can be used in our Android application to infer input text from speech recognition engine.

The Figure 15 shows the training phase of the RNN.

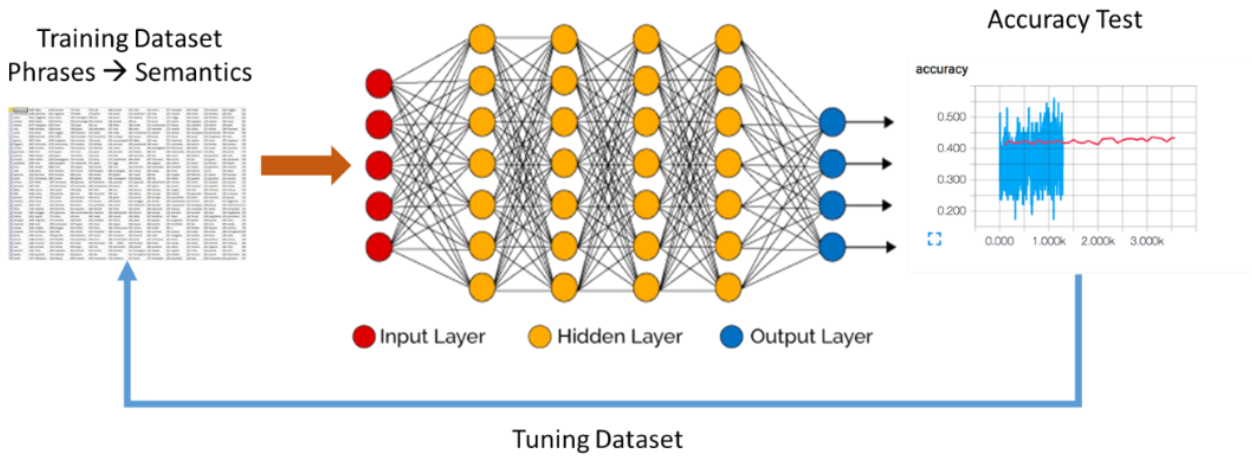



**Figure 15: Training Phase of EasyTV RNN**

Based on tests results, tuning operations are performed on the training phrases dataset.Training and tuning iterations are performed until acceptable network behavior is achieved. Once the training process is completed, we obtain a model that can be used in our Android application to infer results coming from the speech recognition engine.

### 4.9.2. **Interpretation Phase**

Neural network interpretations generate two types of results:

- Directly executable commands: ex. Start the movie

- Commands with extra information to extract for example: rewind video 5 minutes and 30 seconds

Regular expressions are used to extract the contents from second commands type.

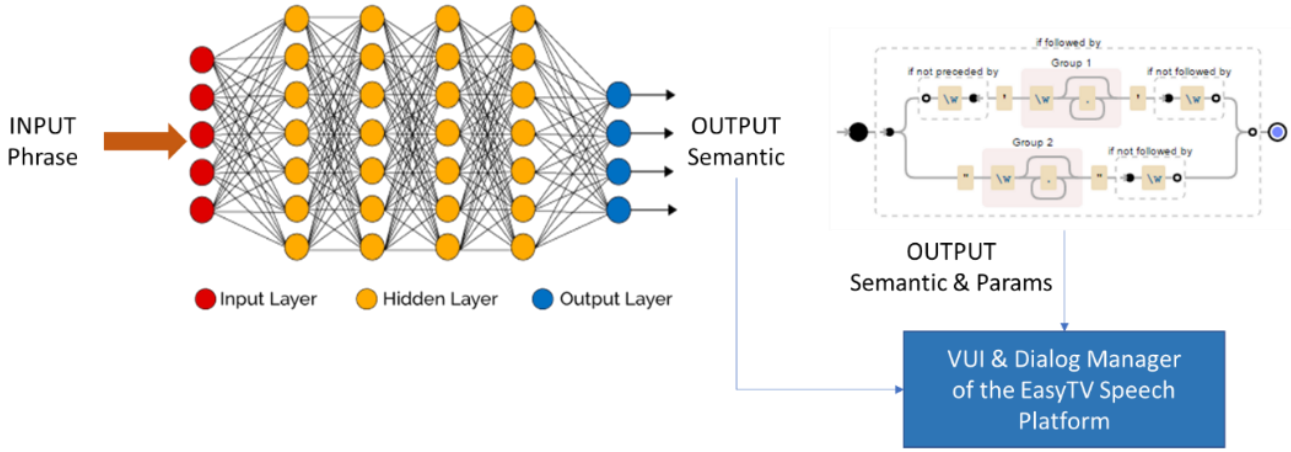



**Figure 16: Interpretation Phase of EasyTV RNN**

For every intent that expects content extraction, one or more regular expressions are created.In order to create a regular expression, we must take into account the syntactic structure of the sentence and detailed technical knowledge on regular expression syntax.
Based on the classification obtained from the neural network, the text is analyzed by one or more regular expressions. In case of MATCH the values (parameters) are extracted and used by the Dialog Manager, in case of NO MATCH we use for now a generic answer "I did not understand".

# 5. DIALOG ANALYSIS FOR TV APP DOMAIN

## 5.1. General considerations

The dialog analysis of EasyTV App domain describes the dialogs and conversations between end users and EasyTV applications and services. To accomplish this, we took into account all the previous work done in the tasks of analysis of user requirements, presented in the deliverable D1.1 [2] (User scenario and requirements definition) and the system specifications, defined in D1.2 [3] (EasyTV system requirements specifications).
The EasyTV Dialog analysis consider not only features and functionalities of the TV applications but also the final users and their capabilities.
The Voice User Interface hence is part of a well-designed product and makes it inclusive and universally accessible. Designing for different populations means leveraging inclusive design or universal design strategies. With this kind of VUI, EasyTV makes this accommodation benefiting everyone of its end user even normal people.
The Analysis reported in this chapter is a first version of the EasyTV App domain dialog and represents the milestone M3.1 related to the EasyTV Speech Platform. Moreover it will be completed with the detailed information as reported in the Voice User Interface Guidelines (Addendum – Voice User Interface Guidelines) ones the EasyTV Application and service to implement have been finally defined and the VUI refined during its implementation.
Finally, the EasyTV VUI will be integrated to the EasyTV Applications and services using the EasyTV Speech Platform, where this integration will be done through the EasyTV Speech Platform SDK and its components.

## 5.2. TV App domains, dialogs and sub dialogs

The Dialog analysis of the EasyTV App Domain consider the following application domains do define its Voice User Interface.

- Video On Demand catalog
- LiveTV
- EPG (Electronic Program Guide)
- Manage Recordings (Scheduled and Recorded)
- Video Player Controller

Now we describe the functionalities and the related dialog flows defined using the VUI guidelines described in the Addendum – Voice User Interface Guidelines.

### 5.2.1. **EasyTV Video On Demand Catalogue**

#### *5.2.1.1* Description

In this application domain the end user can speak with the EasyTV video catalogue to search, browse and control video playing. The user can also activate or deactivate services as needed.
We report here a list of functionalities for the VOD catalogue and the VUI that have been analyzed and defined.

- Searching video.
- Browsing Video by categories.
    - Choosing a category or a subcategory recursively
- Controlling the video
    - Play/Pause/Stop/FastForward/FastRewind/Skip
    - Activate or deactivate subtitles
    - Activate or deactivate secondary audio channels

#### 5.2.1.2 Voice User Interface

In the Figure 17 we depicted the dialog flow schemas of the VOD Domain catalogue VUI:
The main dialog allows the user to change dialog context and navigate to a sub dialog that direct the user to accomplish a specific task. For example, the user wants to go directly to a predefined VideoList (suggested video or favorite videos and so on) and want the system to read aloud the corresponding titles and information. In this case the user can access this dialog context by a specific voice command.
The main dialog can lead to a list of sub dialogs which defines the conversation flow between the user and the specific set of functionalities.

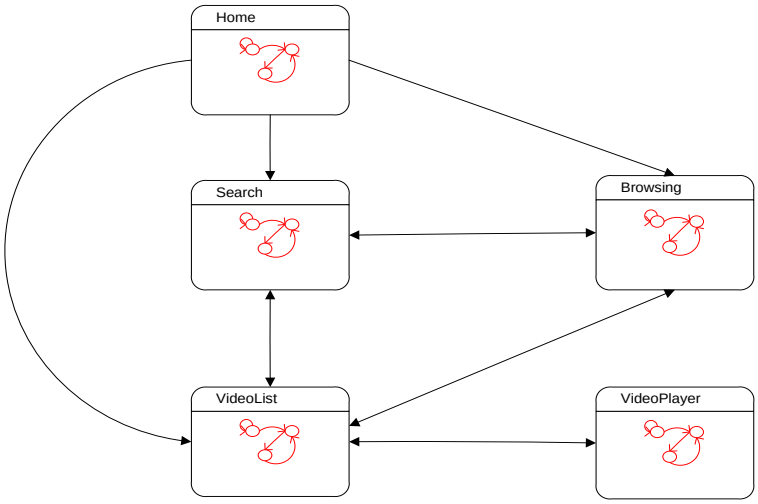



**Figure 17: Video On Demand Main Dialog**

In the Figure 18 we report the single context of the sub dialog home which can lead to one of the following sub dialogs: Search, VideoList or Browsing.
To move to a specific sub dialog the user will use a set of defined voice commands that are represented in the schema with the action names on the Transition arrows. Each action name on the list represent a specific set of voice command.
For example, when the user wants to move to the the VideoList sub dialog he/she can say one of the voice commands related to the action names reported in the schema. The VideoList sub dialog is activated then on one of the following video lists depending on the user voice command; that is: the list of favorites video, last accessed videos (video history) or the list of video suggested by the EasyTV platform.

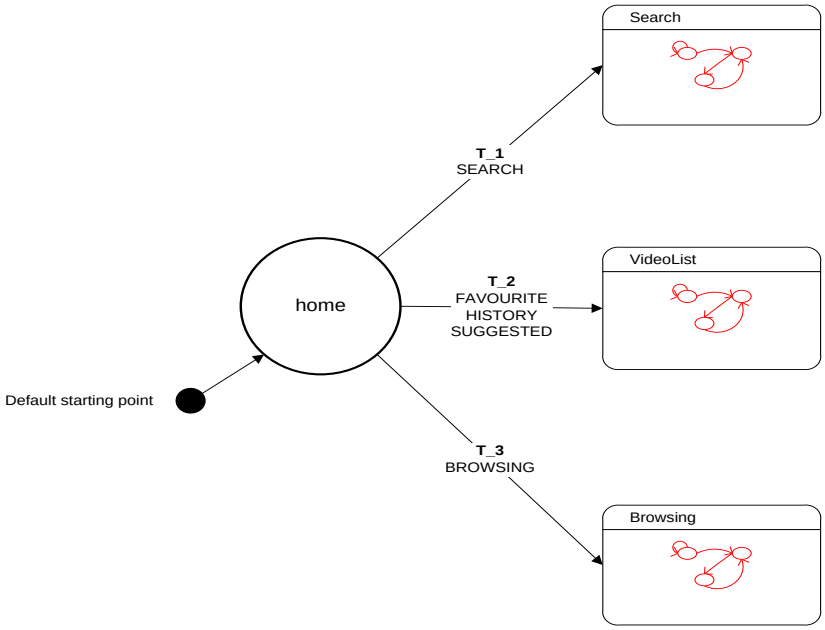



**Figure 18: Video On Demand Sub Dialog Home**

The Figure 19Figure 19: Video On Demand Sub Dialog Search is related to the Search sub dialog. In this sub dialog the user can search videos by means of spelling the keywords and phrases or dictating the phrase using the Speech Recognition Engine in dictation mode. For the scope of this analysis we consider having a search functionality on the application that just need keyword or phrases to look for. This feature of course can be any kind of search depending on the application specific.
The spelling dialog context allows the user to spell any character, number or symbol using a country dependent spelling alphabet and to start search at any moment during the voice input.
The dictation dialog context instead can lead to any video list, depending on the search result related to the phrase or keywords dictated by the user or to the browsing sub dialog if the user changes his mind and wants to access video lists browsing by categories.

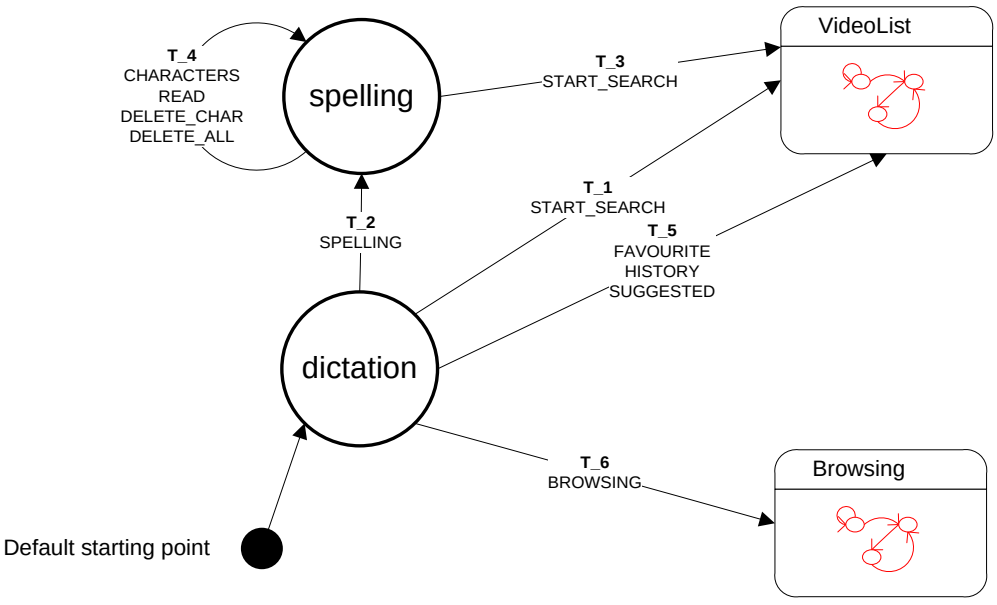


**Figure 19: Video On Demand Sub Dialog Search**

The Figure 20 is related to the Browsing sub dialog. In this sub dialog the user can browse videos selecting any listed category or subcategory provided by the VOD catalogue.
Choosing a category, the user can recursively select categories if the one selected contains a list of sub categories or can access the full video list provided in any moment during the recursive search. Moreover, the user can access directly a specific video list browsing directly the category or sub category of his choice. For example, the favorite video list or the history video list and so on.
A special list can be also a list of TV live channels which are particular videos streamed live through the VOD catalogue. We will report this special category of live video channels later in this chapter.
In this sub dialog context, again, the user, can also come back to the Search sub dialog to access video lists.

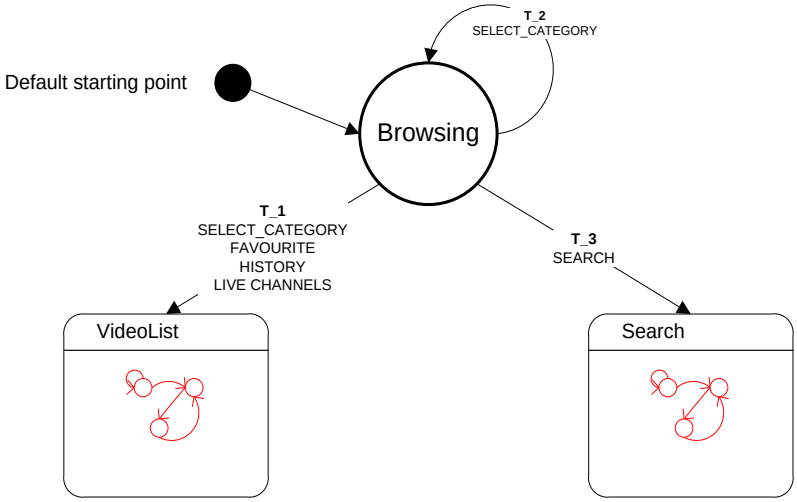



**Figure 20: Video On Demand Sub Dialog Browsing**

The Figure 21 is related to the VideoList sub dialog. In this sub dialog the user can execute all his voice commands in a single dialog context. Here is a list of voice commands that the user can execute:

- Navigate the video list by page going to the previous or next page and get the page list of titles read aloud.
- Navigate the next or previous video and ask the detailed information.
- Add the video to the favourite video list.
- Change the video list going directly to other predefined video list such as favourites or history video list.
- Go back to the Browsing sub dialog.
- Execute the selected video and manage its execution through the VideoPlayer sub dialog.

Choosing a category, the user can recursively select categories if the one selected contains a list of sub categories or can access the full video list provided in any moment during the recursive search. Moreover, the user can access directly a specific video list browsing directly the category or sub category of his choice. For example, the favorite video list or the history video list and so on.
A special list can be also a list of TV live channels which are particular videos streamed live through the VOD catalogue. We will report this special category of live video channels later in this chapter.
In this sub dialog context, again, the user, can also come back to the Search sub dialog to access video lists.

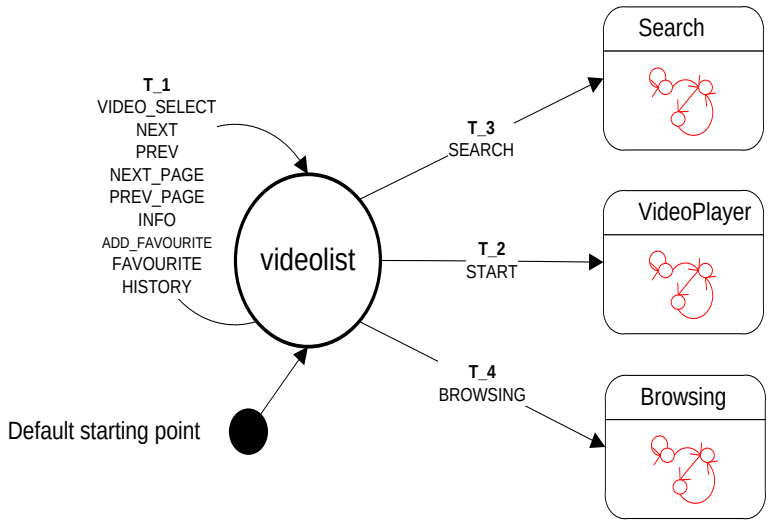



**Figure 21: Video On Demand Sub Dialog VideoList**

The Figure 22 is related to the Player sub dialog. This is one of the important sub dialogs of the EasyTV VUI. In this sub dialog the user can control the video playing and all its features.
Here below we report a list of voice commands that the user can execute on the video:

- Play/Pause/Stop the video.
- Go back and forth on the video even specifying the time frame.
- Activate or deactivate fullscreen.
- Show or hide subtitles.
- Activate or deactivate secondary audio channels
- Add the video to his favourite video list.
- Ask detailed information of the current video
- Go back to the Browsing sub dialog
- Go back to the Search sub dialog

In this sub dialog context, the user can also come back to the Search sub dialog or to the browsing sub dialog to access any other video list.

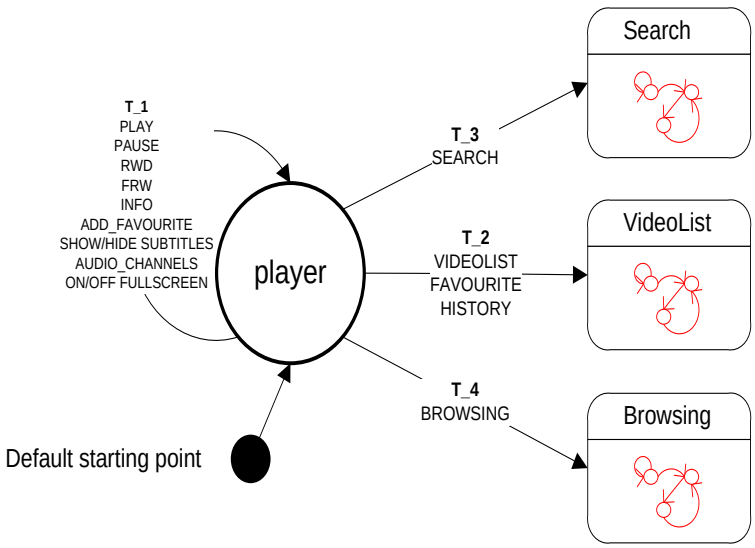



**Figure 22: Video On Demand Sub Dialog Player Control**

### 5.2.2. EasyTV Live Channels, VREC and EPG

#### 5.2.2.1 Description

Live Channels, VREC and EPG are related to the “LiveTV” App domain. To analyze the VUI of LiveTV channels we defined a special category for a video list that represent the list of LiveTV channels which are particular videos streamed live through the VOD catalogue. The VUI analysis of EasyTV Live Channels include also all the functionalities of the Video Recording and the EPG which normally are related to the live channels.
This App domain includes the following functionalities:

- Playing and controlling a LiveTV channel
- Accessing Audio Description Channel
- Recording of a LiveTV Program
- Schedule recordings of a LiveTV channel or LiveTV Program
- Browsing Recorded Programs
- Playing and control a Recorded Program
- Browse the EPG

#### 5.2.2.2 Voice User Interface

Here we report the dialog flow schemas of the LiveTV VUI and its associated functionalities of VREC and EPG:
The main dialog allows the user to change dialog context and navigate to all sub dialog that direct the user to accomplish a specific task related to LiveTV application domain. With this dialog the user can specifically move to the following sub dialog systems:

- RecTimer: VUI that manage recording timer for a specific LiveTV channel
- Scheduled: VUI that manage all the functionalities to control scheduled video recordings.
- Recorded: VUI that manage all recorded videos
- EPG: VUI that manage all the functionalities to Browse and execute actions on Scheduled programs for all LiveTV channels.

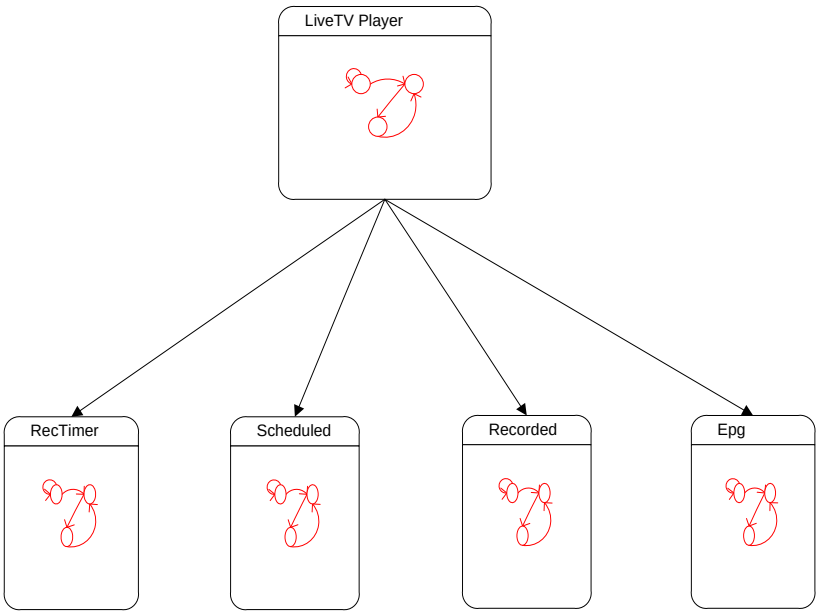



**Figure 23: LiveTV Main Dialog**

The Figure 24 is related to the Scheduled sub dialog. In this sub dialog the user can ask for a specific Program item on the list and can get detailed information or can delete the scheduled recording.
The user can also go back to the Live TV Player which manages all the LiveTV channels.

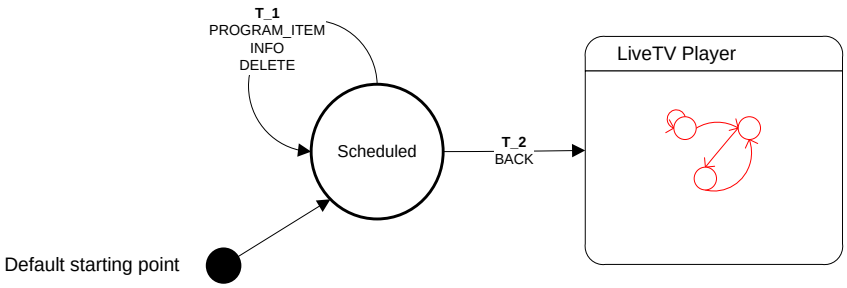



**Figure 24: LiveTV Sub Dialog Scheduled Recordings**

The Figure 25 is related to the RecTimer sub dialog. In this sub dialog the user can schedule a specific timer to record a live channel filling by voice all the information needed that is the LiveTV channel to record, the date and the corresponding start time and end time.
The user can also go back to the Live TV Player which manages all the LiveTV channels.

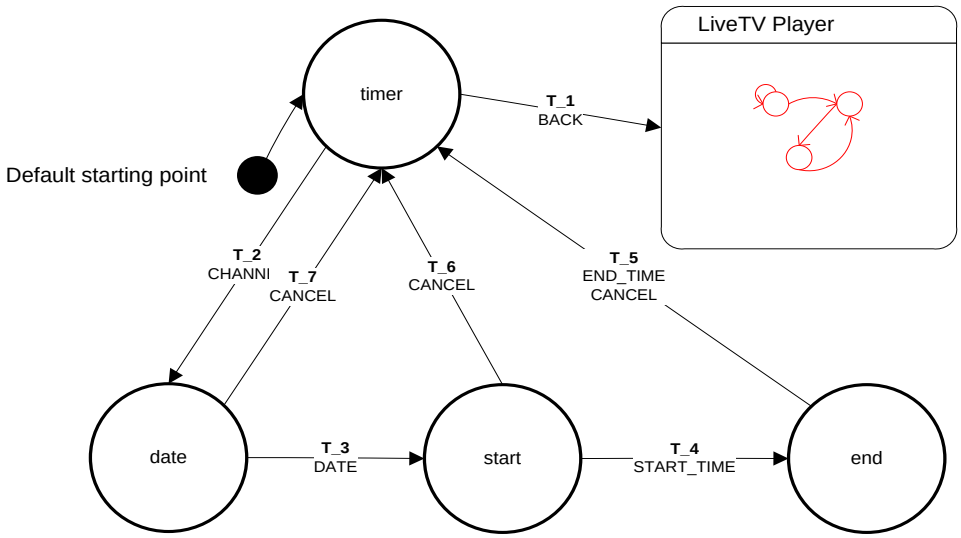



**Figure 25: LiveTV Sub Dialog Recording Timers**

Figure 26 is related to the Recorded sub dialog. In this sub dialog the user can ask for a specific Program item on the list and can get detailed information or can delete the Recorded item.
The user can also go back to the Live TV Player which manages all the LiveTV channels.

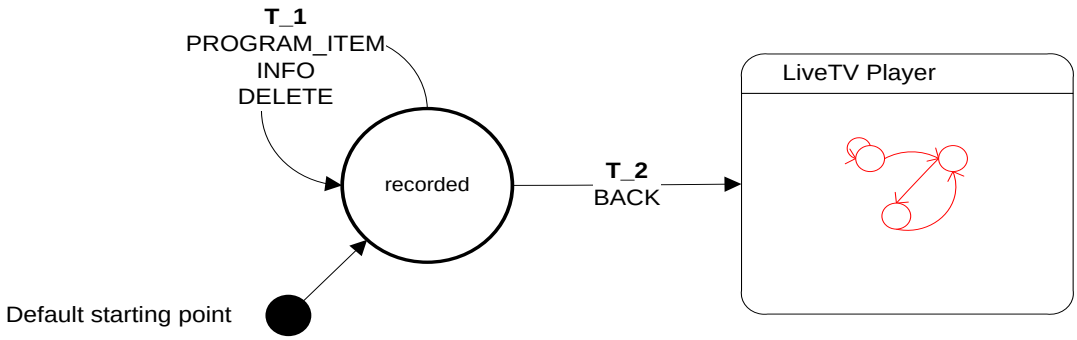



**Figure 26: LiveTV Sub Dialog Recorded Programs**

Figure 27 is related to the EPG sub dialog. In this sub dialog the user can ask for a specific channel and/or a specific time slot to filter the list of program titles. The user can also select directly a specific program item on the list to get detailed information or schedule recording of the program. The user can also go back to the Live TV Player which manages all the LiveTV channels.

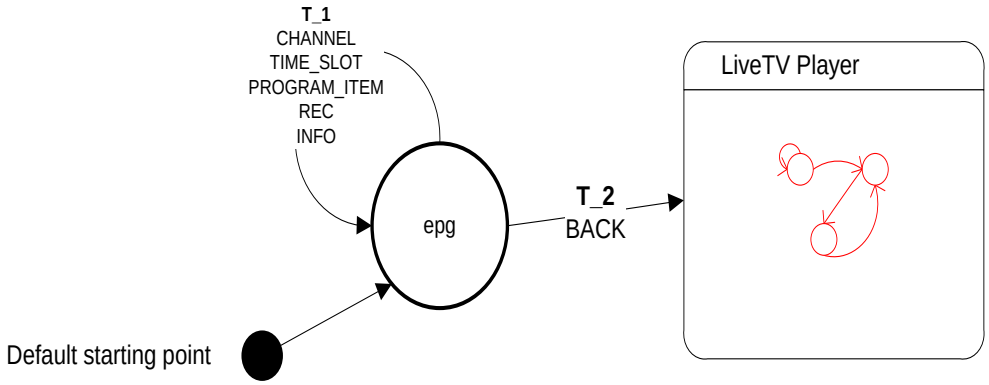



**Figure 27: EPG Sub Dialog**

Figure 28 is related to the LiveTV Player sub dialog. This is one of the important sub dialogs of the EasyTV VUI. In this sub dialog the user can control the video playing and all its features like the VOD Player.
Here below we report a list of voice commands that the user can execute on the video:

- Play/Pause/Stop the video.
- Go back and forth on the video if timeshifting feature is available.
- Activate or deactivate fullscreen
- Show or hide subtitles
- Activate or deactivate secondary audio channels
- Record the current LiveTV channel and program

- Stop Recording the current LiveTV channel and program
- Add the channel to his favourite videolist.
- Ask detailed information of the current program
- Access the LiveTV channel list.

In this sub dialog context, the user can also come back at any time to the Scheduled or Recorded sub dialog, RecTimer and EPG sub dialogs.

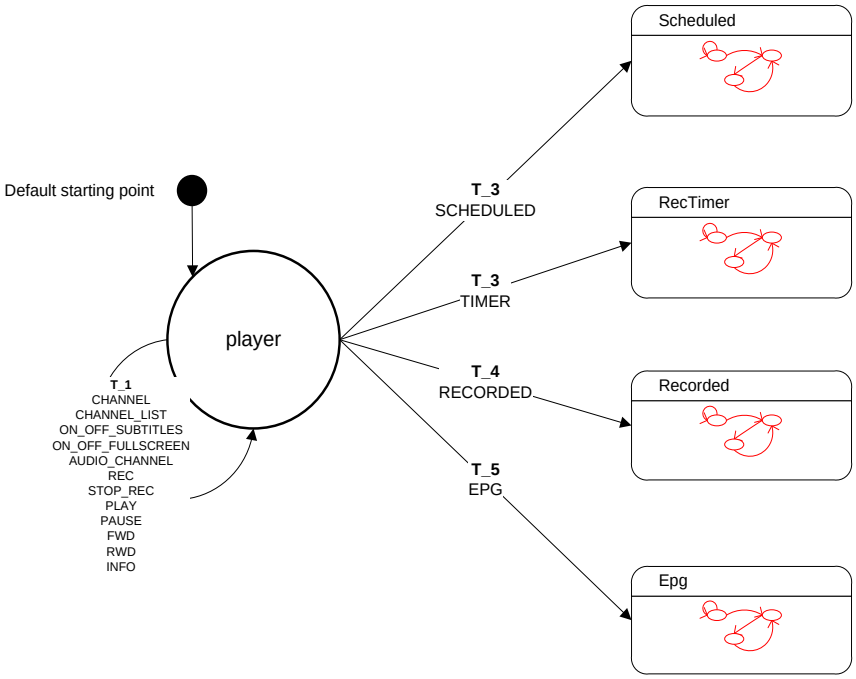



**Figure 28: Player Sub Dialog**

# 6. FINAL ARCHITECTURE AND SDK

## 6.1. Overview

After the first prototype of the EasyTV Speech Platform, we refined the architecture and its components mainly in the interpretation part of the voice command and the dialog flow management. The following **Figure** 29, reports the final version of this architecture.

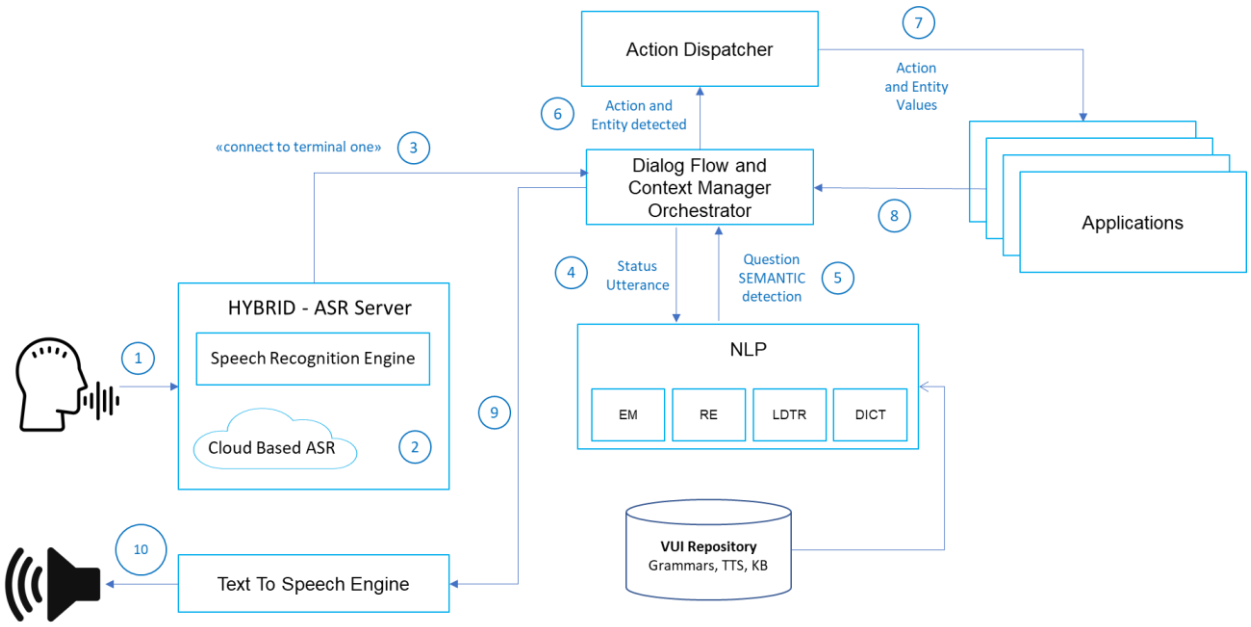



**Figure 29: EasyTV Speech Platform – Final internal architecture**

The main components of this architecture are the "**Dialog Flow and Context Manager Orchestrator**" (**DFCMO**) and the "**NLP**" component with its sub-components.

The **DFCMO** deals with managing the various phases of the dialog process and acts like an orchestrator for all the steps of the process. The **NLP** component includes different techniques for the interpretation of the voice command that are reported later in this section and is designed to be modular in such a way that many subcomponents can be added beside the ones already implemented, including RNN subcomponents.

The DFCMO carries out the following activities:

- Semantic analysis of the input text through the **NLP** component.
- Coordination of the activities related to the current dialog state and voice command semantic:
  - Dispatching of actions to perform by the application through the "Action Dispatcher" (AD) component.
  - Mange dialog flow and moving through dialog states.
  - Manage related voice prompts with Text To Speech Engine

The semantic analysis is one of the core activities of ESP-SDK (EasyTV Speech Platform SDK), since with this activity the ESP associates the voice command to one or more actions to be performed by the application or by the ESP itself. The semantic analysis component (NLP) can assign a semantic value to the sentence pronounced by the user using different specialized sub-components. In each dialog state the developer can indicate which sub-components to use for semantic analysis.

The sub-components already implemented in the final ESP are the following:

- EM: Exact Match. The NLP component matches this type of grammar only if the input phrase from the ASR matches exactly one of the phrases listed in the EM grammar.
- RE: Regular Expression. The NLP component matches this type of grammar where the input phrase from the ASR matches one of the Regular Expressions defined in the RE grammar.
- LDTR: Levenshtein Distance Text Recognition. The NLP component matches this type of grammar where the input phrase from the ASR has the lower distance from one of the phrases defined in the LDTR grammar. This technique is a string metric for measuring the difference between two sequences of characters.
- DICT: Dictation of a generic text. The NLP component returns the text to be used by the application as returned by the ASR without any interpretation.

More details of these sub-components are also reported in the description of the final SDK.

Other components of the architecture are already described in the general architecture reported in chapter 134.2, so to understand the functionalities and the implementation of the final system we report the description of the voice command life cycle.

## 6.2. EasyTV voice command life cycle

In this section we report the life cycle of an EasyTV voice command in order to understand how the whole system works during the voice interaction with an end-user. We also report the number of the step depicted in the final architecture while describing the voice command life cycle.

The interaction with a voice enabled application with the ESP, usually starts with a voice command issued by the end-user (step 1). The EasyTV Speech Platform SDK activates the Speech Recognition engine for each voice command (step 2). The transcription of the input voice can be performed both in cloud and locally, based on the availability of the network connection. The cloud-based recognition is usually more accurate than the local one since more resources are available to perform this task (step 3).

The result of the ASR will be a list of phrases each with its own confidence level (a probability related to each phrase with values from 0 to 1) The more the confidence is close to 1, the more the phrase returned by the ASR is close to the user utterance.

The phrase with a higher confidence is then processed by the EasyTV Speech Platform if it's higher of a threshold configured as a setting of the ESP. The input phrase is now managed by the "Dialog Flow and Context Manager Orchestrator" (DFCMO) component of the ESP SDK (step 4).

The **DFCMO** manages the input text by executing the following activities:

- Performs the semantic analysis of the input text through the NLP component.
- Coordinates the activities related to the current dialog state and voice command semantic:
  - Dispatching of actions to perform by the application through the "Action Dispatcher" (AD) component.
  - Mange dialog flow and moving through dialog states.
  - Manage related voice prompts with Text To Speech Engine

The semantic analysis is performed by the NLP component using different specialized sub-

components. In each dialog state the developer can indicate which sub-components to use for semantic analysis by assigning a priority to each one so that the NLP can improve the activities of matching the right semantic (step 5).

The semantics obtained from the NLP component is then returned to the DFCMO which, by analyzing the data of the active dialogue state, performs all the following activities: manages the actions to be performed by the application through the Action Dispatcher (AD) (steps 6 and 7), moves to the next dialogue state and manages the voice prompts to be sent to the user via the speech synthesis engine (step 9).

The actions received by the application (step 7) are executed and, as soon as the result is returned to the DFCMO (step 8), a final voice prompt can be sent to the user again (step 9) and so the ESP is ready to perform a new cycle of interaction by waiting a new voice command from the user.

## 6.3. Final EasyTV Speech Platform SDK

In this section we present the ESP-SDK (EasyTV Speech Platform SDK), in particular the Cordova plugin[1] version which is a set of add-on code that provides JavaScript interface to native components. This plugin allows developers to use native device capabilities other than features available for pure web apps. Cordova is an open-source mobile development framework that allows using standard web technologies (HTML5, CSS3, and JavaScript) for cross-platform development. With Cordova, applications execute within wrappers targeted to each platform, and rely on standards-compliant API bindings to access each device's capabilities such as sensors, data, network status, etc.

The native ESP-SDK is a set of Android classes that can be easily ported in other platform in order to use native components and modules of the specific platform (i.e. speech engines). With the development of the Cordova plugin version of our ESP-SDK we allow cross-platform development for any CSApps.

All the source code of the Cordova plugin and a Speech Platform Demo sample are available in the project repository in GitLab[2].

With the development of the final ESP-SDK and particularly the NLP and Dialog Manager components we finalized the development and technical testing of this modules along with the whole EasyTV Speech Platform SDK and tested it successfully in a HelloCordova application. This application is a base application with code commented and ready to use as a guide in the activities of the integration of the final VUI in the CSApp

The ESP-SDK plugin is is included in the **dialogmanagerplugin** object. It is the main component and acts as an orchestrator. It's developed to manage the dialog flow and the whole life cycle of the speech interaction, particularly the text To speech and speech recognition engines, the speech interpretation activities and the actions management. It includes different sets of methods and events that can be grouped as follow:

- Initialization
- Text To Speech
- Dialog State Management
- NLP Objects
- Event Management

[1] https://cordova.apache.org/

[2] https://gitlab.com/easytv-cs/speech-sdk

In the next sections we describe the methods and events of the ESP-SDK.

### 6.3.1. Initialization Methods

When the Cordova 'deviceready' event is fired a **dialogmanagerplugin** object has to be declared:

*dlgmanager = new cordova.plugins.dialogmanagerplugin();*

then InititDialogManager method must be called to initialize the TTS and ASR engines:

*dlgmanager.initDialogManager('en-Gb');*

After the initialization phase, the system is ready to start the user voice interaction and receive events from the voice platform

**initDialogManager**(String *language*): This method initializes the Text To Speech and Speech Recognition engines. At the end of init process the **onPlatformInitialized** event is fired.

The *language* parameter is the

**start**(): This method starts speech recognition engine activity and, as soon as the recognition process ends the **onResult** event is fired.

**stop**(): This method stops the speech recognition engine activity.

### 6.3.2. Text To Speech Methods

**speech**(String *text*, boolean *queue*): This method sends the text to the TTS engine.

- text: string value parameter that represent the text to be converted in audio.
- queue: boolean value. If true, the text is queued; false to stop the TTS engine and start immediately the new text conversion.

**speechOut**(boolean *queue*): send one of the text in tts_out list (chosen randomly) to TTS engine

- queue: boolean value. If true, the text is queued; false to stop the TTS engine and start immediately the new text conversion.

speechIn(boolean *queue*): send one of the text in tts_in list (chosen randomly) to TTS engine

- queue: boolean value. If true, the text is queued; false to stop the TTS engine and start immediately the new text conversion.

### 6.3.3. Dialog State Management

In this section we present the methods that manages the dialog flow and the dialog states which are the core concepts of the user and application conversations.

The dialog flow is a state chart representation between user and application conversations. The dialog state is the most important part of the dialog flow and identifies a specific moment of the interaction between the user and the application. In ESP-SDK both static and dynamic states can be defined in order to make more powerful and complex interactions.

The following objects and methods to create and manage dialog states and dialog flows.

**VoiceState**: This object builds a single dialog state of a dialog flow.

**constructor**(String *stateId*, NLP[] *resultManager*, String[] *tts_in*, String[] *tts_out*)

- *stateId*: String value. This parameter id a unique identifier of the voice state
- *resultManager*: This parameter is the most important in the dialog state, it represents the list of NLP objects (grammar objects) used to get the semantic value from the recognized text using NLP component and its sub-components. When the voice command is issued, the NLP components looks only at *resultManager list of objects* to find and match the right user intent and its related semantic. The semantic value is then used to convert the voice command in actions for the application.
- *tts_in*: This parameter represents the list of voice prompts (phrases) used when the VoiceState is activated. When a voice state is activated one of the texts in the list (chosen randomly) is used by TTS engine.
- *tts_out*: This parameter represents the list of voice prompts (phrases) used when the VoiceState is left to move to a new one in the dialog flow. When a new voice state is activated one of the texts in the list (chosen randomly) can be used calling "**speechOut**" method from **dialogmanagerplugin**.

**createVoiceStates**(VoiceState[] *states*, Callback *success*, Callback *error*): This method creates a list of voice states for the dialogmanager object.

- *states*: Is the list ov VoiceState objects.
- *success*: Is the callback function raised if setting voice state process ends successfully.
- *error*: Is the callback function raised if setting voice state process is interrupted by an error.

**addVoiceState**(VoiceState *voiceState*, Callback *success*, Callback *error*): add a voice state to voice state list. See createVoiceStates function.

- *voiceState*: Is the VoiceState object to add to the list of VoiceStates objects.
- *success*: Is the callback function raised if setting voice state process ends successfully.
- *error*: Is the callback function raised if setting voice state process is interrupted by an error.

**setVoiceState**(VoiceState *voiceState*, boolean *speechIn*, boolen *queue*, Callback *success*, Callback *error*): activates a voice state.

- *voiceState*: Is the VoiceState object to activate.
- *speechIn*: boolean value. Is the parameter that tells to the **dialogmanagerplugin** whether to play a *tts_in* prompt (randomly) when the *voiceState* is activated.
- *queue*: boolean value. If true, the text is queued; false to stop the TTS engine and start immediately the new text conversion
- *success*: Is the callback function raised if setting voice state process ends successfully.
- *error*: Is the callback function raised if setting voice state process is interrupted by an error.

**setVoiceStateById** (String *stateId*, boolean *speechIn*, boolean *queue*): activates a voice state.

- *stateId*: Is the VoiceState id to activate.
- *speechIn*: boolean value. Is the parameter that tells to the **dialogmanagerplugin** whether to play a *tts_in* prompt (randomly) when the *voiceState* is activated.
- *queue*: boolean value. If true, the text is queued; false to stop the TTS engine and start immediately the new text conversion
- *success*: Is the callback function raised if setting voice state process ends successfully.
- *error*: Is the callback function raised if setting voice state process is interrupted by an error.

Beside the VoiceState object and the methods reported above we have a list of methods useful to manage VoiceState objects and its functionalities. In the following list we report some of these methods:

- **setTts_in**(String[] tts_in): set a list of tts_in text. This list replaces the previous list defined in the constructor
  - *tts_in*: This parameter represents the list of voice prompts (phrases) used when the VoiceState is activated. When a voice state is activated one of the texts in the list (choosed randomly) is used by TTS engine
- **setTts_out**(String[] tts_out): set a list of tts_in text. This list replaces the previous list defined in the constructor.
  - *tts_out*: This parameter represents the list of voice prompts (phrases) used when the VoiceState is left to move to a new one in the dialog flow.
- **addResultManager**(NLP recognition): add an new NLP sub-component object to list of NLP objects.
  - *recognition*: This parameter represents the new NLP object to include in the list.
- **addTTSInRange**(String[] *tts_list*): adds a list of voice prompts (phrases) to the existing *tts_in* list.
  - *tts_list*: Is the new list of voice prompts to add.
- **addTTSOutRange**(String[] tts_list): adds a list of voice prompts (phrases) to the existing *tts_out* list.
  - tts_list: Is the new list of voice prompts to add.

### 6.3.4. NLP Objects

The NLP Object performs the semantic analysis through its specialized sub-components. In each dialog state the developer can indicate which sub-components (grammar) to use for semantic analysis by assigning a priority to each one so that the NLP can improve the activities of matching the right semantic. As already reported in the life cycle description of the voice command, in the ESP-SDK we have three NLP sub-components:

**ExactTextRecognition**: This grammar object executes an exact match between the list of phrases defined in the ExactTextRecognition grammar object and the utterance recognized by the ASR engine. When an exact match occurs, the **onResult** event is fired and a result with a confidence of 1 (100%) is returned.

- **add**(String *semantic*, String *phrase*): This method adds a new <semantic, phrase> pair to the list of phrases allowed for the grammar object.
  - Semantic: is the semantic value associated to the *phrase* and returned by the onResult event.
  - *phrase*: Is the phrase that will be added to the list of allowed phrases.
- **setPriority**(int *priority*): Sets the priority of the grammar object to be used when executing

the semantic analysis.
    - *Priority*: Is the value of the priority used by the dialog manager to check the results from NLP objects. The lower priority is processed first.

**Dictation**: This grammar object is used to return the utterance from the ASR without checking the semantic value and is used for dictation activities (i.e. to write some text in a text box and so on).

- **setPriority**(int *priority*): Sets the priority of the grammar object to be used when executing the semantic analysis.
    - *Priority*: Is the value of the priority used by the dialog manager to check the results from NLP objects. The lower priority is processed first.

**LevenshteinDistanceTextRecognition**: This grammar object is used by the NLP component to match the phrase with a lower distance from one of the phrases defined in the LevenshteinDistanceTextRecognition grammar. In the event of a mismatch, a NO_MATCH message is returned.

- **add**(String *semantic*, String *phrase*): This method adds a new <semantic, phrase> pair to the list of phrases allowed for the grammar object.
    - Semantic: is the semantic value associated to the *phrase* and returned by the onResult event.
    - *phrase*: Is the phrase that will be added to the list of allowed phrases.
- **setPriority**(int *priority*): Sets the priority of the grammar object to be used when executing the semantic analysis.
    - *Priority*: Is the value of the priority used by the dialog manager to check the results from NLP objects. The lower priority is processed first.

**RegExprTextRecognition**: This grammar object is used by the NLP component to match the input phrase recognized by the ASR with one of the Regular Expressions defined in the RegExprTextRecognition grammar. In the event of a mismatch, a NO_MATCH message is returned.

- **add**(String *semantic*, String *regExpr*): This method adds a new <semantic, phrase> pair to the list of phrases allowed for the grammar object.
    - Semantic: is the semantic value associated to the *phrase* and returned by the onResult event.
    - *regExpr*: Is the regular expression representing the phrase structure that will be added to the list of those allowed. Example of a typical regExpr could be the following:

      [\w\s]*(?<**command**>volume)\s*[a-zA-Z\s]*\s*(?<**volumevalue**>[\d]*)[\D\w\s]*

      Where <**command**> and <**volumevalue**> are group names in the phrase structure.

      In this *regExpr* allowed phrases are: “set volume to 8”, “volume 7”, “volume to 3 please” and so on.

- **setPriority**(int *priority*): Sets the priority of the grammar object to be used when executing the semantic analysis.
    - *Priority*: Is the value of the priority used by the dialog manager to check the results from NLP objects. The lower priority is processed first.

### 6.3.5. **Event Management**

In this section we report the list of events fired by the ESP-SDK during the life cycle of a voice command, and in general, using the ESP-SDK objects and methods. We can list them as follow:

**onResult** (Event *event*): This event is fired by ASR engine when the speech recognition process end with a valid result and a valid list of utterances are returned by the ASR engine.

- *event*: is an object that contains the list of results. Each result has four elements: *semantic*, *utterance*, *confidence*, *params*
  - *semantic* element contains the semantic as defined in the grammar object.
  - *utterance* element contains the phrase returned by the ASR engine.
  - *confidence element is the probability returned by the NLP Object.*
  - *Params element is an optional element containing the values extracted from the **RegExprTextRecognition** NLP Object in the order they are defined in the Regular Expression.*

**onPartialResult (**Event *event***):** This event is fired by ASR engine when partial speech is recognized and transformed in a string. In this case only the utterance element is filled by the ESP-SDK since it returns only what the ASR is understanding while the user is speaking.

**onTTSStart(**String *tts***):** This event is fired by TTS engine when it starts the conversion of the string in audio format.

- *tts:* is the text that is going to be converted in audio by the Text To Speech Engine.

**onTTSEnd(**String *tts***):** This event is fired by TTS engine when it ends the conversion of the string in audio format.

- *tts:* is the text that is going to be converted in audio by the Text To Speech Engine.

**onPlatformInitialized():** This event is fired after a call to **initDialogManager** function and theTTS, ASR and the dialog manager are initialized and ready to start managing a dialog with the end user.

**OnASREndOfSpeech():** This event is fired when the end of speech event is detected by ASR engine.

**OnASRError():** This event is fired when an error occurred during the speech recognition processing phase.

# 7. COMPANION SCREEN APPLICATION VUI

In this section we report a sample of the CSApp application flow and the design of its Voice User Interface in order to support the implementation of the final voice interaction system and prepare the prototype to test with final users. As a general guideline we designed a dialog flow diagram for each CSApp screen and defined the dialog states and their related elements (grammar objects and semantics, voice prompts and transactions between dialog states).

In the next paragraphs we present an overview of the CSApp screen flow, and the dialog flow diagrams for each CSApp screen VUI considered in this sample. For each dialog flow diagram, we also report the list of tables representing the grammar objects and related information, involved in the vocalization of the screen functionalities (semantics, destination screen and destination voice state). Finally, we also report a table with the overview of all voice states with their related grammar objects and voice prompts.

## 7.1. CSApp screen flow

The Figure 30: Companion Screen Application - Screen flow, shows the screen flows of the companion screen application. While the Figure 31: CSApp VUI Overview, shows an overview of the whole VUI detailed in the next sections.

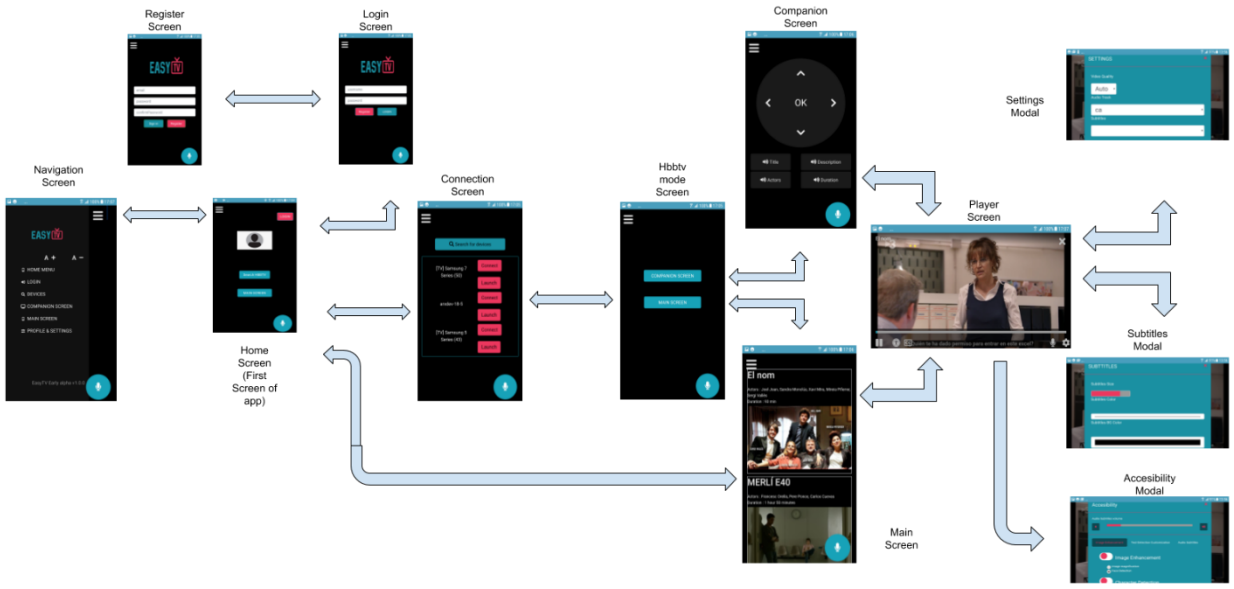



**Figure 30: Companion Screen Application - Screen flow**

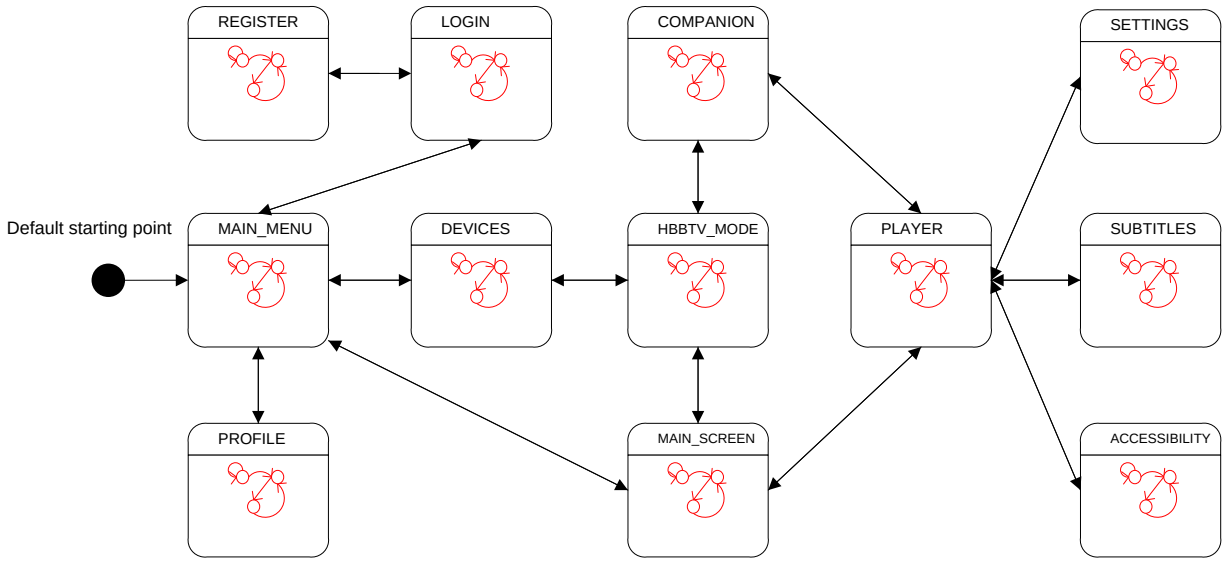



**Figure 31: CSApp VUI Overview**

## 7.2. Dialog States and Grammar Objects

In this section we report the table of all dialog states defined in each dialog flow chart and their related grammar objects and voice prompts that will be used in the ESP-SDK to implement the vocalization of the CSApp (Companion Screen Application).

| STATE_ID | GRAMMAR_OBJECT | SPEECH_IN | SPEECH_OUT |
|---|---|---|---|
| MENU | • UNIVERSALS<br>• VOLUME<br>• MAIN_MENU | Welcome to Easy TV. Choose from … (*here will go appropriate items listed*) | - |
| USER | • UNIVERSALS<br>• VOLUME<br>• LOGIN_CREDENTIALS<br>• LOGIN | Tell me your username. Otherwise choose from skip or registration. | - |
| PASS | • UNIVERSALS<br>• VOLUME<br>• LOGIN_CREDENTIALS | Tell me your password. | - |
| OK | • UNIVERSALS<br>• VOLUME<br>• LOGIN | Choose from restart or submit. | - |
| EMAIL | • UNIVERSALS<br>• VOLUME<br>• REGISTRATION_CREDENTIALS<br>• REGISTRATION | Tell me your email. Otherwise tell me sign in. | - |

| | | | |
|---|---|---|---|
| CREATEPASS | • UNIVERSALS<br>• VOLUME<br>• REGISTRATION_CREDENTIALS | Create a new password. | - |
| REPEATPASS | • UNIVERSALS<br>• VOLUME<br>• REGISTRATION_CREDENTIALS | Repeat the password. | - |
| REG | • UNIVERSALS<br>• VOLUME<br>• REGISTRATION | Tell me register or cancel. | - |
| CONNECTION | • UNIVERSALS<br>• VOLUME<br>• DEVICES<br>• DEVICES_SELECT_TERMINAL | Select a terminal and tell me connect. | - |
| SELECTION | • UNIVERSALS<br>• VOLUME<br>• HBBTV_MODE | Choose from Companion Screen or Main Screen. | - |
| REMOTE | • UNIVERSALS<br>• VOLUME<br>• COMPANION | Chose a command from up, down, left, right, play, ... | - |
| CATALOG | • UNIVERSALS<br>• VOLUME<br>• MAIN_SCREEN<br>• MAIN_SCREEN_SELECT_VIDEO | Select a movie between.... | - |
| PLAYER | • UNIVERSALS<br>• VOLUME<br>• PLAYER | Starting video. | - |
| CHANGE_SETTINGS | • UNIVERSALS<br>• VOLUME<br>• MAIN_SETTINGS<br>• SETTINGS_VIDEO_QUALITY<br>• SETTINGS_AUDIO_TRACK | Choose from enable or disable subtitles. Or select a video quality or an audio track. | - |
| SUBS_SETTINGS | • UNIVERSALS<br>• VOLUME<br>• SUBTITLES<br>• SUBTITLES_TEXT_SIZE<br>• SUBTITLES_TEXT_COLOR<br>• SUBTITLES_BG_COLOR | Select the subtitles size, colour or background colour.<br>Otherwise tell me close subtitles settings. | - |
| ACC_SETTINGS | • UNIVERSALS<br>• VOLUME<br>• ACCESSIBILITY<br>• ACCESSIBILITY_TEXT_SIZE<br>• ACCESSIBILITY_TEXT_COLOR<br>• ACCESSIBILITY_TEXT_BG_COLOR | Choose from enable image enhancement, enable text detection, enable audio subtitle, .... | - |

**Table 1: CSApp VUI - List of dialog states and features**

## 7.3. UNIVERSALS Grammar Objects

First, we define two main grammars that are always active and available in any dialog flow state of the CSApp VUI. They are the UNIVERSALS grammar object and the VOLUME grammar object defined in the tables below:

The Table 2: CSApp VUI - Grammar Object UNIVERSALS, reports the information about the UNIVERSALS grammar object of type ExactTextRecognition

| Voice Command sample | SEMANTIC | DESTINATION SCREEN | DEST STATE |
|---|---|---|---|
| Help | HELP | - | - |
| Repeat | REPEAT | - | - |
| Back | BACK | - | - |
| Main menu | MAINMENU | MAIN_MENU | MENU |

**Table 2: CSApp VUI - Grammar Object UNIVERSALS**

Here below we report the table regarding the information about the VOLUME grammar object.

| Voice Command sample | SEMANTIC | DESTINATION SCREEN | DEST STATE |
|---|---|---|---|
| set volume to 8 | SET_VOLUME / 8 | - | - |
| volume 7 | SET_VOLUME / 7 | - | - |
| volume to 3 please | SET_VOLUME / 3 | - | - |

**Table 3: CSApp VUI: Grammar Object VOLUME**

The type of this grammar object is RegExprTextRecognition where the following is a sample of a regular expression included in the grammar:

Regular Expression: [\w\s]*(?<**command**>volume)\s*[a-zA-Z\s]*\s*(?<**volumevalue**>[\d]*)[\D\w\s]*

The phrases on the Table 3: CSApp VUI: Grammar Object VOLUME, are instead phrases that matches the aforementioned regular expression.

In the list of voice commands, we report only one sample word/phrase associates to each semantic value, but each semantic value of the voice command can be replicated using any number of words/phrases in order to make the interpretation performed by the NLP component more effective. Furthermore, for each semantic value we report the following information:

- DEST SCREEN, representing the destination screen where the application will fall after the actions related to the semantic of the voice command have been executed. If no destination screen is defined means any screen depending on the result of the actions executed.

- DEST STATE, representing the destination dialog flow state where the voice interaction will fall after the actions related to the semantic of the voice command have been executed. More than one voice states can be defined depending on the result of the actions executed.

## 7.4. Main Menu screen

In the following table we report the dialog flow chart for the main menu screen of the CSApp.

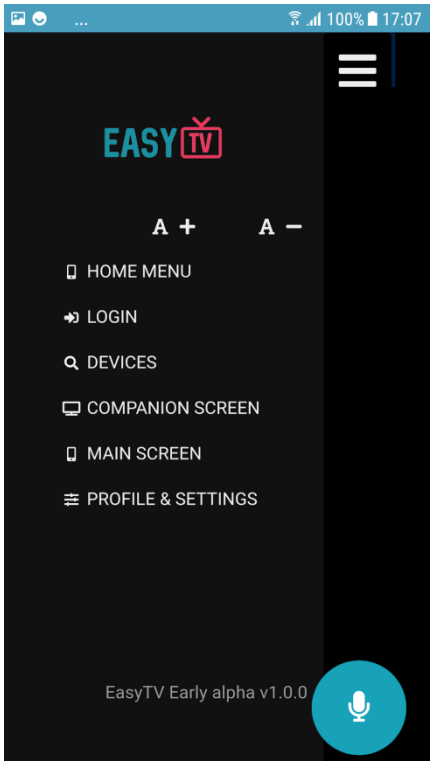
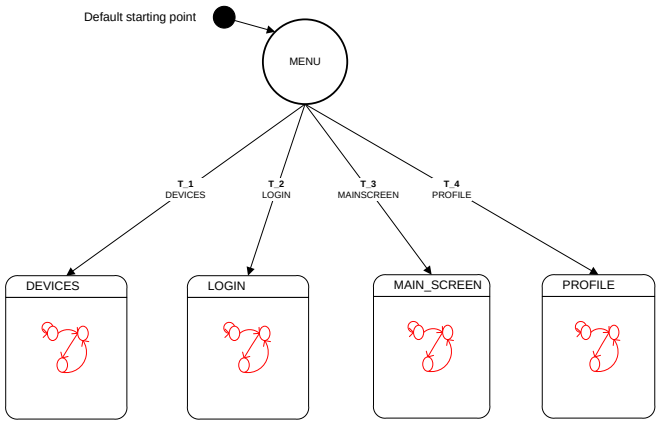
| START SCREEN | DIALOG FLOW CHART |
|---|---|
|  |  |



**Table 4: CSApp VUI: Main Menu dialog flow chart**

The Table 5: CSApp VUI: Grammar Object MAIN_MENU reports the information about the MAIN_MENU grammar object of type LevenshteinDistanceTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| devices | DEVICES | DEVICES | CONNECTION |
| login | LOGIN | LOGIN | USER |
| Main screen | MAINSCREEN | MAIN_SCREEN | CATALOG |
| My profile | PROFILE | PROFILE | PROFILE |

**Table 5: CSApp VUI: Grammar Object MAIN_MENU**

## 7.5. Login screen

In the following table we report the dialog flow chart for the LOGIN screen of the CSApp.

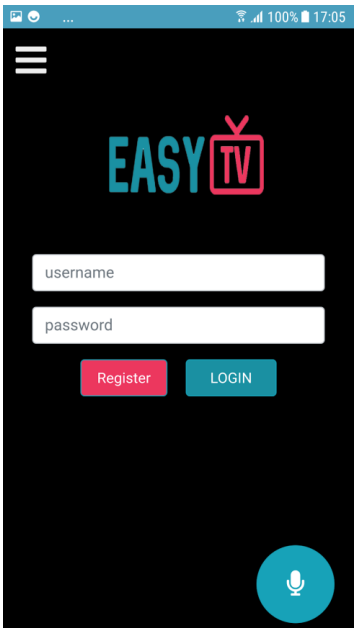
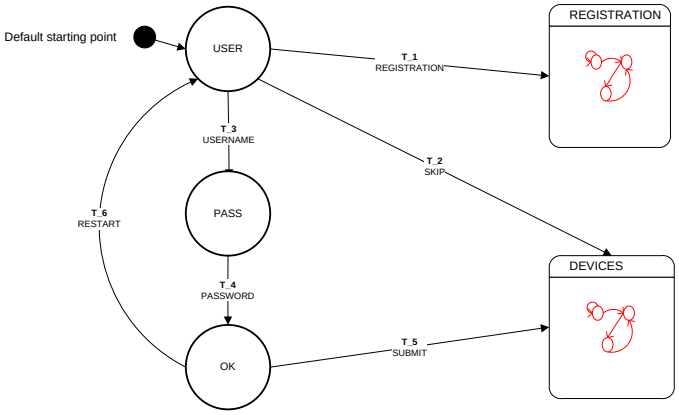
| START SCREEN | DIALOG FLOW CHART |
|---|---|
|  |  |

**Table 6: CSApp VUI: Login dialog flow chart**

The Table 7: CSApp VUI: Grammar Object LOGIN_CREDENTIALS reports the information about the LOGIN_CREDENTIALS grammar object of type Dictation.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| [my username] | USERNAME | LOGIN | PASS |
| [my password] | PASSWORD | LOGIN | OK |

**Table 7: CSApp VUI: Grammar Object LOGIN_CREDENTIALS**

Whenever the voice command sample is reported in square brackets refers to the user utterance (the phrase dictated by the user).

The Table 8: CSApp VUI: Grammar Object LOGIN reports the information about the LOGIN grammar object of type LevenshteinDistanceTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| registration | REGISTRATION | REGISTRATION | EMAIL |
| skip | SKIP | DEVICES | CONNECTION |
| submit | SUBMIT | DEVICES | CONNECTION |
| restart | RESTART | LOGIN | USER |

**Table 8: CSApp VUI: Grammar Object LOGIN**

## 7.6. Registration screen

In the following table we report the dialog flow chart for the REGISTRATION screen of the CSApp.

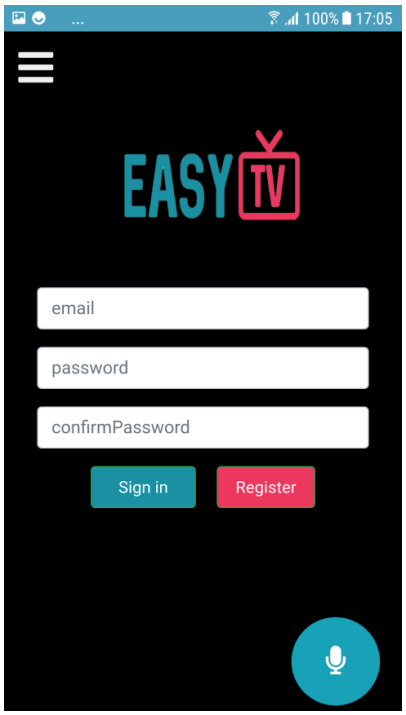
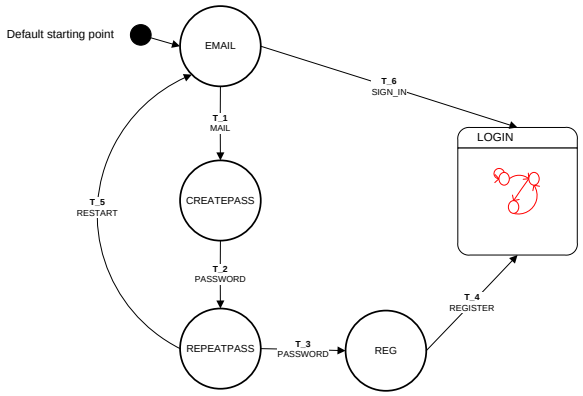
| START SCREEN | DIALOG FLOW CHART |
|---|---|
|  |  |

**Table 9: CSApp VUI: Registration dialog flow chart**

The Table 10: CSApp VUI: Grammar Object REGISTRATION_CREDENTIALS reports the information about the REGISTRATION_CREDENTIALS grammar object of type Dictation.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| [my username] | MAIL | REGISTRATION | CREATEPASS |
| [my password] | PASSWORD | REGISTRATION | REPEATPASS |
| [my password] | PASSWORD | REGISTRATION | REG |

**Table 10: CSApp VUI: Grammar Object REGISTRATION_CREDENTIALS**

The Table 11: CSApp VUI: Grammar Object REGISTRATION reports the information about the REGISTRATION grammar object of type LevenshteinDistanceTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| sign in | SIGN_IN | LOGIN | USER |
| register | REGISTER | LOGIN | USER |
| restart | RESTART | REGISTRATION | EMAIL |

**Table 11: CSApp VUI: Grammar Object REGISTRATION**

## 7.7. Device list

In the following table we report the dialog flow chart for the DEVICES screen of the CSApp.

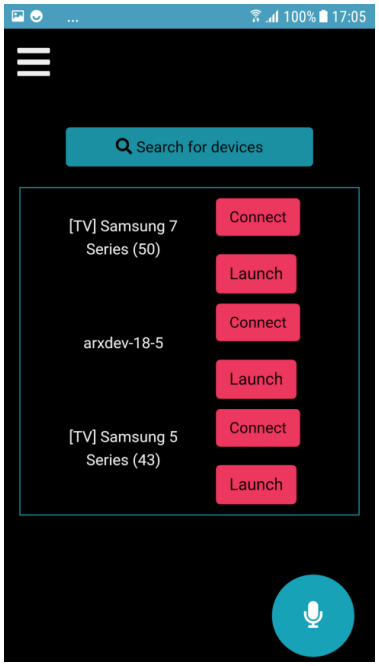
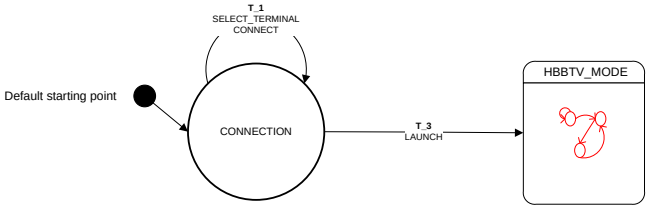
| START SCREEN | DIALOG FLOW CHART |
|---|---|
|  |  |



**Table 12: CSApp VUI: Devices dialog flow chart**

The **Table** 13**: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**reports the information about the DEVICES_SELECT_TERMINAL grammar object of type RegExprTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| select terminal number 3 | SELECT_TERMINAL | DEVICES | CONNECTION |
| terminal 2 | SELECT_TERMINAL | DEVICES | CONNECTION |
| Terminal 4 please | SELECT_TERMINAL | DEVICES | CONNECTION |

**Table 13: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**

The type of this grammar object is RegExprTextRecognition where the following is a sample of a regular expression included in the grammar:

Regular Expression: [\w\s]*(?<**command**>terminal)\s*[a-zA-Z\s]*\s*(?<**terminalvalue**>[\d]*)[\D\w\s]*

The phrases on **Table** 13**: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**, are instead phrases that matches the aforementioned regular expression.

The Table 14: CSApp VUI: Grammar Object DEVICES reports the information about the DEVICES grammar object of type LevenshteinDistanceTextRecognition

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| connect | CONNECT | DEVICES | CONNECTION |
| launch | LAUNCH | HBBTV_MODE | SELECTION |

**Table 14: CSApp VUI: Grammar Object DEVICES**

## 7.8. HBBTV mode screen

In the following table we report the dialog flow chart for the HBBTV_MODE screen of the CSApp.

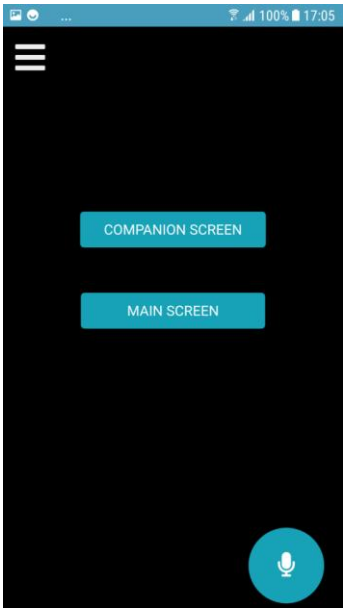
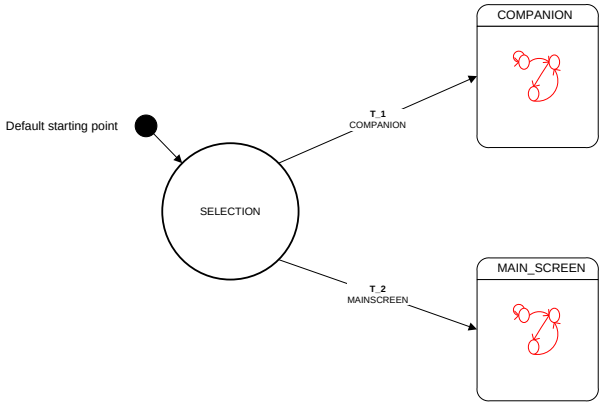
| START SCREEN | DIALOG FLOW CHART |
|---|---|
|  |  |

**Table 15: CSApp VUI: HBBTV mode dialog flow chart**

The Table 16: CSApp VUI: Grammar Object HBBTV_MODE reports the information about the HBBTV_MODE grammar object of type LevenshteinDistanceTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| companion screen | COMPANION | COMPANION | REMOTE |
| main screen | MAINSCREEN | MAIN_SCREEN | CATALOG |

**Table 16: CSApp VUI: Grammar Object HBBTV_MODE**

## 7.9. Companion screen remote control

In the following table we report the dialog flow chart for the COMPANION screen of the CSApp.

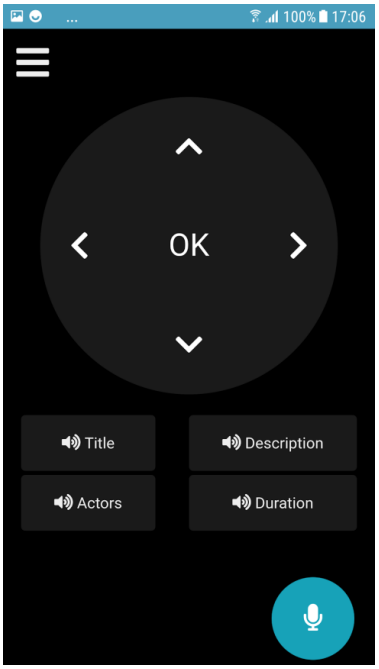
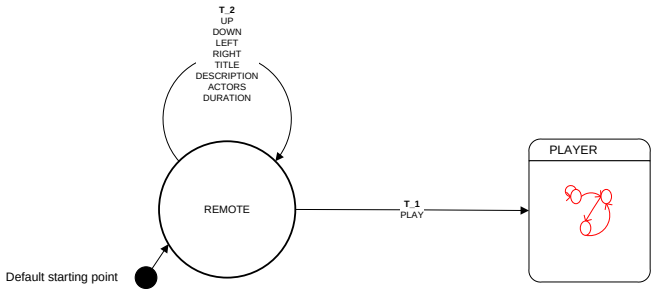
| START SCREEN | DIALOG FLOW CHART |
|---|---|
|  |  |

**Table 17: CSApp VUI: COMPANION dialog flow chart**

The Table 18: CSApp VUI: Grammar Object COMPANION reports the information about the COMPANION grammar object of type LevenshteinDistanceTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| play | PLAY | PLAYER | PLAYER |
| up | UP | COMPANION | REMOTE |
| down | DOWN | COMPANION | REMOTE |
| left | LEFT | COMPANION | REMOTE |
| right | RIGHT | COMPANION | REMOTE |
| title | TITLE | COMPANION | REMOTE |
| description | DESCRIPTION | COMPANION | REMOTE |
| actors | ACTORS | COMPANION | REMOTE |
| duration | DURATION | COMPANION | REMOTE |

**Table 18: CSApp VUI: Grammar Object COMPANION**

## 7.10. Main screen

In the following table we report the dialog flow chart for the MAIN_SCREEN screen of the CSApp.

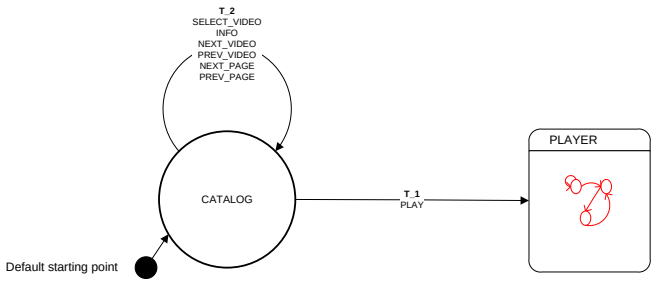
| START SCREEN | DIALOG FLOW CHART |
|---|---|
|  |  |



**Table 19: CSApp VUI: MAIN_SCREEN dialog flow chart**

The Table 20: CSApp VUI: Grammar Object MAIN_SCREEN reports the information about the MAIN_SCREEN grammar object of type LevenshteinDistanceTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| play | PLAY | PLAYER | PLAYER |
| info | INFO | MAIN_SCREEN | CATALOG |
| next video | NEXT_VIDEO | MAIN_SCREEN | CATALOG |
| prev video | PREV_VIDEO | MAIN_SCREEN | CATALOG |
| next page | NEXT_PAGE | MAIN_SCREEN | CATALOG |
| prev page | PREV_PAGE | MAIN_SCREEN | CATALOG |

**Table 20: CSApp VUI: Grammar Object MAIN_SCREEN**

The Table 21: CSApp VUI: Grammar Object MAIN_SCREEN_SELECT_VIDEO reports the information about the MAIN_SCREEN_SELECT_VIDEO grammar object of type RegExprTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| select video number 1 | SELECT_VIDEO | MAIN_SCREEN | CATALOG |
| video 3 | SELECT_VIDEO | MAIN_SCREEN | CATALOG |
| video 6 please | SELECT_VIDEO | MAIN_SCREEN | CATALOG |

**Table 21: CSApp VUI: Grammar Object MAIN_SCREEN_SELECT_VIDEO**

The type of this grammar object is RegExprTextRecognition where the following is a sample of a regular expression included in the grammar:

Regular Expression: [\w\s]*(?<**command**>video)\s*[a-zA-Z\s]*\s*(?<**videoalue**>[\d]*)[\D\w\s]*

The phrases on Table 21: CSApp VUI: Grammar Object MAIN_SCREEN_SELECT_VIDEO, are instead phrases that matches the aforementioned regular expression.

## 7.11. Player screen

In the following table we report the dialog flow chart for the PLAYER screen of the CSApp.

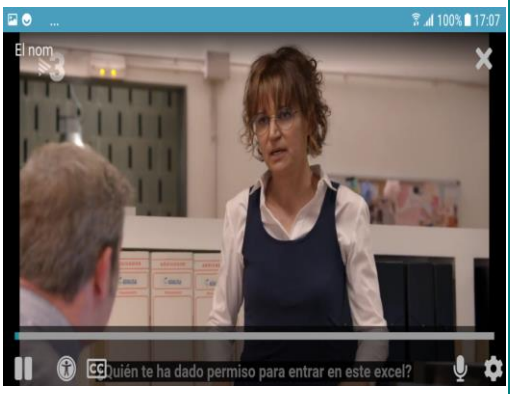
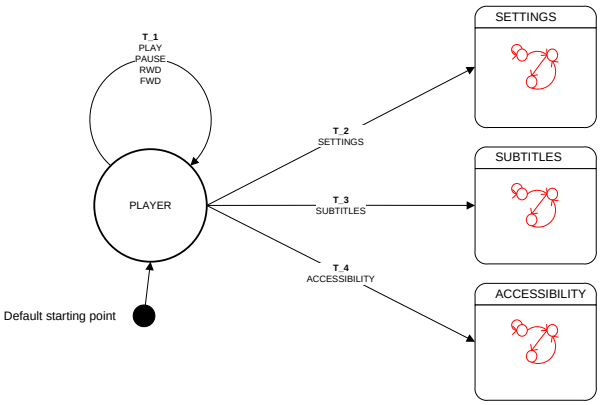
| START SCREEN | DIALOG FLOW CHART |
|---|---|
|  |  |



**Table 22: CSApp VUI: PLAYER dialog flow chart**

The Table 23: CSApp VUI: Grammar Object PLAYER reports the information about the PLAYER grammar object of type LevenshteinDistanceTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| play | PLAY | PLAYER | PLAYER |
| pause | PAUSE | PLAYER | PLAYER |
| forward | FWD | PLAYER | PLAYER |
| rewind | RWD | PLAYER | PLAYER |
| settings | SETTINGS | SETTINGS | CHANGE_SETTINGS |
| subtitles | SUBTITLES | SUBTITLES | SUBS_SETTINGS |
| Accessibility | ACCESSIBILITY | ACCESSIBILITY | ACC_SETTINGS |

**Table 23: CSApp VUI: Grammar Object PLAYER**

## 7.12. Accessibility settings

In the following table we report the dialog flow chart for the ACCESIBILITY_SETTINGS screen of the CSApp.

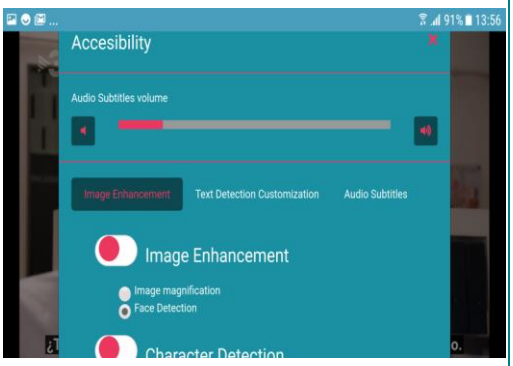
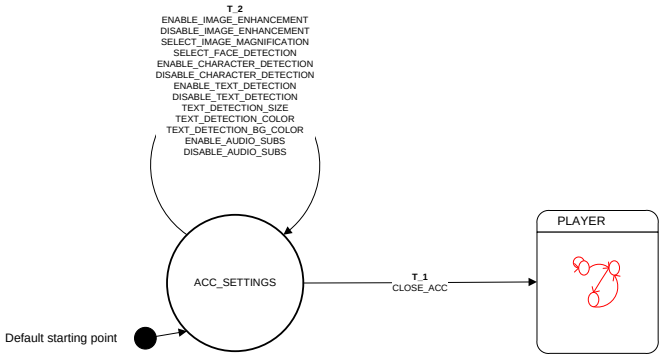
| START SCREEN | DIALOG FLOW CHART |
|---|---|
|  |  |

**Table 24: CSApp VUI: ACCESSIBILITY_SETTINGS dialog flow chart**

The **Table** 25**: CSApp VUI: Grammar Object ACCESSIBILITY** reports the information about the ACCESSIBILITY grammar object of type LevenshteinDistanceTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| enable image enhancement | ENABLE_IMAGE_ENHANCEMENT | ACCESSIBILITY | ACC_SETTINGS |
| disable image enhancement | DISABLE_IMAGE_ENHANCEMENT | ACCESSIBILITY | ACC_SETTINGS |
| enable image magnification | SELECT_IMAGE_MAGNIFICATION | ACCESSIBILITY | ACC_SETTINGS |
| enable face detection | SELECT_FACE_DETECTION | ACCESSIBILITY | ACC_SETTINGS |
| enable character detection | ENABLE_CHARACTER_DETECTION | ACCESSIBILITY | ACC_SETTINGS |
| disable character detection | DISABLE_CHARACTER_DETECTION | ACCESSIBILITY | ACC_SETTINGS |
| enable text detection | ENABLE_TEXT_DETECTION | ACCESSIBILITY | ACC_SETTINGS |
| disable text detection | DISABLE_TEXT_DETECTION | ACCESSIBILITY | ACC_SETTINGS |
| enable audio | ENABLE_AUDIO_SUBS | ACCESSIBILITY | ACC_SETTINGS |

| | | | |
|---|---|---|---|
| subtitles | | | |
| disable audio subtitles | DISABLE_AUDIO_SUBS | ACCESSIBILITY | ACC_SETTINGS |
| close accessibility settings | CLOSE_ACC | PLAYER | PLAYER |

**Table 25: CSApp VUI: Grammar Object ACCESSIBILITY**

The Table 26: CSApp VUI: Grammar Object ACCESSIBILITY_TEXT_SIZE **Table 13: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**reports the information about the ACCESSIBILITY_TEXT_SIZE grammar object of type RegExprTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| text size number 1 | TEXT_DETECTION_SIZE | ACCESSIBILITY | ACC_SETTINGS |
| text size 3 | TEXT_DETECTION_SIZE | ACCESSIBILITY | ACC_SETTINGS |
| text size 6 please | TEXT_DETECTION_SIZE | ACCESSIBILITY | ACC_SETTINGS |

**Table 26: CSApp VUI: Grammar Object ACCESSIBILITY_TEXT_SIZE**

The type of this grammar object is RegExprTextRecognition where the following is a sample of a regular expression included in the grammar:

Regular Expression: [\w\s]*(?<**command**>text size)\s*[a-zA-Z\s]*\s*(?<**sizevalue**>[\d]*)[\D\w\s]*

The phrases on Table 26: CSApp VUI: Grammar Object ACCESSIBILITY_TEXT_SIZE are instead phrases that matches the aforementioned regular expression.

The **Table 27: CSApp VUI: Grammar Object ACCESSIBILITY_TEXT_COLOR Table 13: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**reports the information about the ACCESSIBILITY_TEXT_COLOR grammar object of type RegExprTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| text color green | TEXT_DETECTION_COLOR | ACCESSIBILITY | ACC_SETTINGS |
| text color yellow | TEXT_DETECTION_COLOR | ACCESSIBILITY | ACC_SETTINGS |

**Table 27: CSApp VUI: Grammar Object ACCESSIBILITY_TEXT_COLOR**

The type of this grammar object is RegExprTextRecognition where the following is a sample of a regular expression included in the grammar:

Regular Expression: [\w\s]*(?<command>text color)\s*(?<color>[a-zA-Z\s]*)

The phrases on Table 27: CSApp VUI: Grammar Object ACCESSIBILITY_TEXT_COLOR are instead phrases that matches the aforementioned regular expression.

The Table 28: CSApp VUI: Grammar Object ACCESSIBILITY_TEXT_BG_COLOR **Table** 13**: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**reports the information about the ACCESSIBILITY_TEXT_BG_COLOR grammar object of type RegExprTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| text background green | TEXT_DETECTION_BG_COLOR | ACCESSIBILITY | ACC_SETTINGS |
| text background yellow | TEXT_DETECTION_BG_COLOR | ACCESSIBILITY | ACC_SETTINGS |
| text background none | TEXT_DETECTION_BG_COLOR | ACCESSIBILITY | ACC_SETTINGS |

**Table 28: CSApp VUI: Grammar Object ACCESSIBILITY_TEXT_BG_COLOR**

The type of this grammar object is RegExprTextRecognition where the following is a sample of a regular expression included in the grammar:

```
Regular Expression: [\w\s]*(?<command>background color)\s*(?<color>[a-zA-Z\s]*)
```

The phrases on **Table** 28: CSApp VUI: Grammar Object ACCESSIBILITY_TEXT_BG_COLOR are instead phrases that matches the aforementioned regular expression.

## 7.13. Subtitles settings

In the following table we report the dialog flow chart for the SUBTITLE_SETTINGS screen of the CSApp.

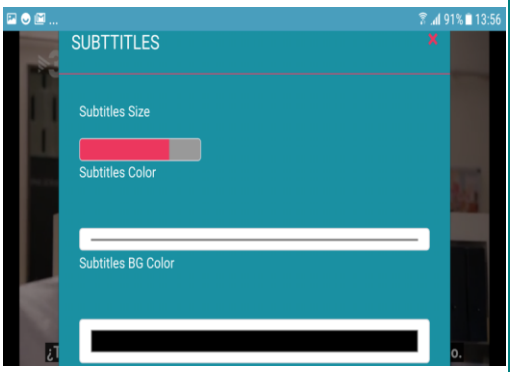
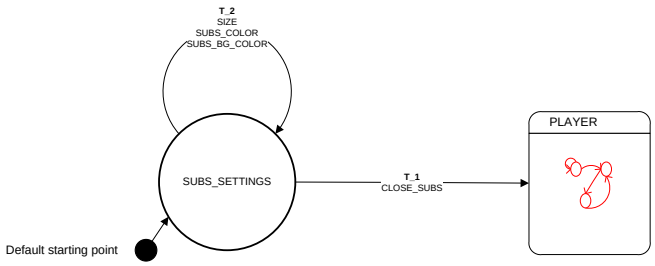
| START SCREEN | DIALOG FLOW CHART |
|---|---|
|  |  |



**Table 29: CSApp VUI: SUBTITLE_SETTINGS dialog flow chart**

The Table 30: CSApp VUI: Grammar Object SUBTITLE_TEXT_SIZE **Table 13: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**reports the information about the SUBTITLE_TEXT_SIZE grammar object of type RegExprTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| text size number 1 | TEXT_DETECTION_SIZE | ACCESSIBILITY | ACC_SETTINGS |
| text size 3 | TEXT_DETECTION_SIZE | ACCESSIBILITY | ACC_SETTINGS |
| text size 6 please | TEXT_DETECTION_SIZE | ACCESSIBILITY | ACC_SETTINGS |

**Table 30: CSApp VUI: Grammar Object SUBTITLE_TEXT_SIZE**

The type of this grammar object is RegExprTextRecognition where the following is a sample of a regular expression included in the grammar:

Regular Expression: [\w\s]*(?<**command**>subtitles size)\s*[a-zA-Z\s]*\s*(?<**sizevalue**>[\d]*)[\D\w\s]*

The phrases on Table 30: CSApp VUI: Grammar Object SUBTITLE_TEXT_SIZE are instead phrases that matches the aforementioned regular expression.

The Table 31: CSApp VUI: Grammar Object SUBTITLE_TEXT_COLOR **Table 13: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**reports the information about the SUBTITLE_TEXT_COLOR grammar object of type RegExprTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| text color green | SUBS_COLOR | SUBTITLES | SUBS_SETTINGS |
| text color yellow | SUBS_COLOR | SUBTITLES | SUBS_SETTINGS |

**Table 31: CSApp VUI: Grammar Object SUBTITLE_TEXT_COLOR**

The type of this grammar object is RegExprTextRecognition where the following is a sample of a regular expression included in the grammar:

```
Regular Expression: [\w\s]*(?<command>subtitles color)\s*(?<color>[a-zA-Z\s]*)
```

The phrases on Table 31: CSApp VUI: Grammar Object SUBTITLE_TEXT_COLOR are instead phrases that matches the aforementioned regular expression.

The Table 32: CSApp VUI: Grammar Object SUBTITLE_BG_COLOR **Table 13: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**reports the information about the SUBTITLE_BG_COLOR grammar object of type RegExprTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| background color green | SUBS_BG_COLOR | SUBTITLES | SUBS_SETTINGS |
| background color yellow | SUBS_BG_COLOR | SUBTITLES | SUBS_SETTINGS |
| background color none | SUBS_BG_COLOR | SUBTITLES | SUBS_SETTINGS |

**Table 32: CSApp VUI: Grammar Object SUBTITLE_BG_COLOR**

The type of this grammar object is RegExprTextRecognition where the following is a sample of a regular expression included in the grammar:

```
Regular Expression: [\w\s]*(?<command>background color)\s*(?<color>[a-zA-Z\s]*)
```

The phrases on Table 32: CSApp VUI: Grammar Object SUBTITLE_BG_COLOR are instead phrases that matches the aforementioned regular expression.

The Table 33: CSApp VUI: Grammar Object SUBTITLES reports the information about the SUBTITLES grammar object of type LevenshteinDistanceTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| close subtitles settings | CLOSE_SUBS | PLAYER | PLAYER |
| close settings | CLOSE_SUBS | PLAYER | PLAYER |
| close please | CLOSE_SUBS | PLAYER | PLAYER |

**Table 33: CSApp VUI: Grammar Object SUBTITLES**

## 7.14. General accessibility settings

In the following table we report the dialog flow chart for the GENERAL_ACC_SETTINGS screen of the CSApp.

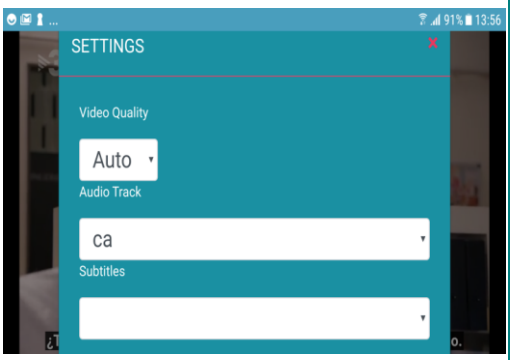
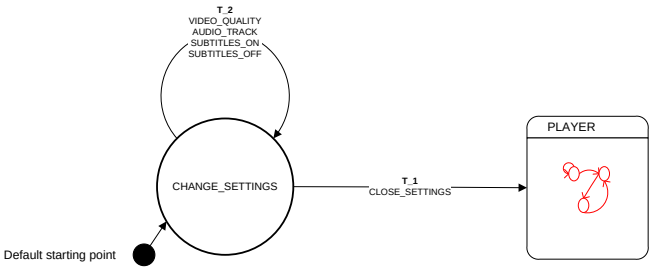
| START SCREEN | DIALOG FLOW CHART |
|---|---|
|  |  |

**Table 34: CSApp VUI: GENERL_ACC_SETTINGS dialog flow chart**

The Table 35: CSApp VUI: Grammar Object SETTING_VIDEO_QUALITY **Table** 13**: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**reports the information about the SETTINGS_VIDEO_QUALITY grammar object of type RegExprTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| video quality number 1 | VIDEO_QUALITY | SETTINGS | CHANGE_SETTINGS |
| video quality 3 | VIDEO_QUALITY | SETTINGS | CHANGE_SETTINGS |
| video quality 6 please | VIDEO_QUALITY | SETTINGS | CHANGE_SETTINGS |

**Table 35: CSApp VUI: Grammar Object SETTING_VIDEO_QUALITY**

The type of this grammar object is RegExprTextRecognition where the following is a sample of a regular expression included in the grammar:

Regular Expression: [\w\s]*(?<**command**>video quality)\s*[a-zA-Z\s]*\s*(?<**qualityvalue**>[\d]*)[\D\w\s]*

The phrases on Table 35: CSApp VUI: Grammar Object SETTING_VIDEO_QUALITY are instead phrases that matches the aforementioned regular expression.

The Table 36: CSApp VUI: Grammar Object SETTINGS_AUDIO_TRACK **Table 13: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**reports the information about the SETTINGS_AUDIO_TRACK grammar object of type RegExprTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| audio track number 1 | AUDIO_TRACK | SETTINGS | CHANGE_SETTINGS |
| audio track 3 | AUDIO_TRACK | SETTINGS | CHANGE_SETTINGS |
| audio track 6 please | AUDIO_TRACK | SETTINGS | CHANGE_SETTINGS |

**Table 36: CSApp VUI: Grammar Object SETTINGS_AUDIO_TRACK**

The type of this grammar object is RegExprTextRecognition where the following is a sample of a regular expression included in the grammar:

Regular Expression: [\w\s]*(?<**command**>audio track)\s*[a-zA-Z\s]*\s*(?<**audiovalue**>[\d]*)[\D\w\s]*

The phrases on Table 36: CSApp VUI: Grammar Object SETTINGS_AUDIO_TRACK are instead phrases that matches the aforementioned regular expression.

The Table 37: CSApp VUI: Grammar Object MAIN_SETTINGS **Table 13: CSApp VUI: Grammar Object DEVICES_SELECT_TERMINAL**reports the information about the MAIN_SETTINGS grammar object of type LevenshteinDistanceTextRecognition.

| Voice Command sample | SEMANTIC | DEST SCREEN | DEST STATE |
|---|---|---|---|
| enable subtitles | SUBTITLES_ON | SETTINGS | CHANGE_SETTINGS |
| disable subtitles | SUBTITLES_OFF | SETTINGS | CHANGE_SETTINGS |
| close settings | CLOSE_SETTINGS | PLAYER | PLAYER |

**Table 37: CSApp VUI: Grammar Object MAIN_SETTINGS**

# 8. CONSIDERATIONS RELATED TO THE VOICE ASSISTANTS

A voice assistant is a digital assistant that uses voice recognition, natural language processing and speech synthesis to help users to perform tasks and interact with applications using their voice. Examples of how assistant can be used are: set reminders and alarms, reading news, ask for weather conditions, find information about restaurants, answer questions via Google search and many more. Popular voice assistants currently include Apple's Siri, Amazon's Alexa, Google Assistant and Microsoft's Cortana.

We have been working and testing the Google and Amazon Alexa assistants and developed some actions for google and some Alexa skills. Doing this, we understood well their architectures and business models and did some considerations comparing them with our ESP-SDK. We also verified the integration possibilities with HBBTV platforms, limits and advantages of these technologies compared with the EasyTV project and particularly with ESP-SDK.

All issues related to the use of voice assistants with other external applications and platforms, and with HBBTV applications, are related to the need for deep integration with their respective platforms. Both for Alexa and google home, voice interfaces and dialogue management must be developed and managed in the cloud from their respective platforms while applications can be resident on third-party servers. This mechanism does not allow companies to have their platform within their infrastructure. Furthermore, as regards HBBTV applications that synchronize with companion screen applications, it is necessary to implement, by voice assistants, the component that acts as a link between the HBBTV application and the companion screen application. This component has not yet been implemented by the voice assistants analyzed so far, so, to ensure that smart speakers, such as Alexa and google home, reach the HBBTV terminal running in the user's home network, it is necessary to perform voice authentication with Google or Amazon platforms and connect the cloud application with a companion screen application resident in the same network as the terminal.

In the following picture (Figure 32: Voice assistants and HBBTV logic architecture), we show the Google Home and Nest Hub logic architecture and how they can fit in the Home network along with a HBBTV Terminal and a Companion Screen Application device.

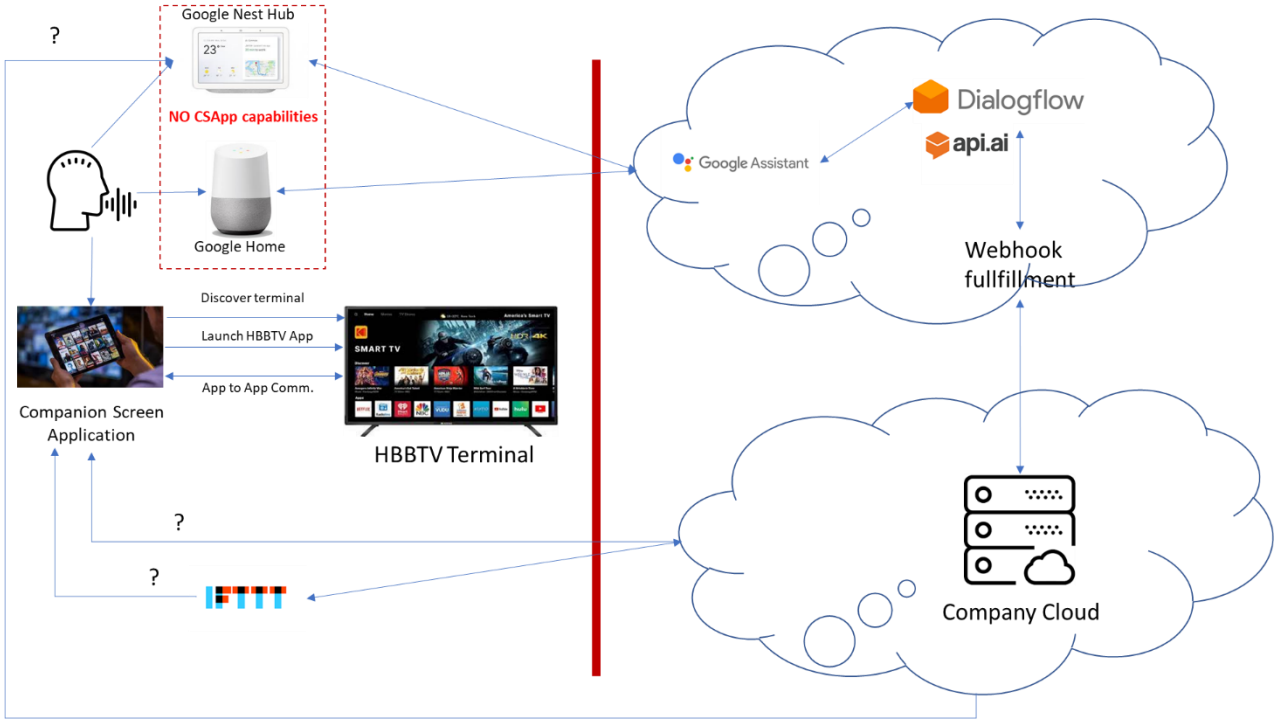



**Figure 32: Voice assistants and HBBTV logic architecture**

As we can see, when the interactions starts from the user to one of the smart devices (Google Home or Google Nest HUB) through the Google Assistant, the interaction can come back to the smart speaker through a voice prompt or, from the Company Cloud applications, to the companion screen device or a IFTTT[3] (IF This Then That) device. The connection with the companion screen applications seems doesn't make sense since the user already interact with the companion screen device which already has voice capabilities. The IFTTT device instead must be programmed with the set of actions in order to communicate with a companion screen application or directly with the HBBTV Terminal. Finally, Google Home and Google Nest Hub doesn't have yet Companion Screen Capabilities.

Regarding the business model also, both Google Assistant and Amazon Alexa are based on a pay per use model which is charged per transactions for the Google DialogFlow components or per transactions for AWS (Amazon Web Service). This requires the company business model to fit the Google or Amazon business model instead of defining their own and have all the EasyTV platform in their own premises. ESP-SDK is a platform built on top of third-party speech engines and hence has a common interface for multiplatform and multilingual applications. Moreover, Broadcasters and other companies can build their own speech engines in case they want to be completely independent and manage their own business model and all the platform in their own premises.

[3] https://ifttt.com/

# 9. CONCLUSIONS

In this document we described the work done to define and develop the EasyTV Speech Platform component that aims to create an effective and easy voice user interface system for blind and visually impaired users. In the first release of this document we started defining the EasyTV Speech Platform and developed a first prototype of this component. Furthermore, we defined a preliminary dialog flow module which includes NLP techniques for the classification and interpretation of Voice Commands based on Recurrent Neural Network. Also, the integration of speech recognition and speech synthesis modules have been included in the Platform. A first dialog analysis for the TV App domain has been carried out and reported in this document in order to help the implementation and integration of the speech interaction in the TV domain applications. Finally, we developed a first prototype of application to test the EasyTV Speech Platform using a simple Voice User Interface for YouTube Web Application.

Afterwards, we defined the final architecture of the EasyTV Speech Platform and its Software developer Kit (SDK) to complete and finalize the work done in the first prototype. Particularly, we developed the final version of the main components of the architecture, the NLP and Dialog Manager components, extracted and implemented the EasyTV Speech Platform SDK (6.3 Final EasyTV Speech Platform SDK) and developed a sample application to execute the technical test of the whole ESP-SDK. With the availability of the ESP-SDK, developers can integrate any Voice User Interface in any multilingual application and particularly in companion screen applications for EasyTV App domain used for the interaction with HBBTV compliant applications.

# 10. REFERENCES


[1] EasyTV – Annex I "Description of Work"
[2] D1.1 User scenario and requirements definition (http://easytvproject.eu)
[3] D1.2 EasyTV system requirements specification (http://easytvproject.eu)
[4] D1.3 First release of the EasyTV system architecture (http://easytvproject.eu)
[5] D1.4 Final release of the EasyTV system architecture (http://easytvproject.eu)
[6] Speech Recognition Grammar Specification (SRGS) (https://www.w3.org/TR/speech-grammar/)
[7] Semantic Interpretation for Speech Recognition (SISR) (https://www.w3.org/TR/semantic-interpretation/)
[8] Speech Synthesis Markup Language (SSML) (https://www.w3.org/TR/speech-synthesis11/)
[9] Voice Extensible Markup Language (VoiceXML) 3.0 (https://www.w3.org/TR/voicexml30/)
[10] VoiceXML Forum (http://www.voicexml.org/)
[11] World Wide Web Consortium (W3C) (https://www.w3.org/)
[12] Google TensorFlow (https://www.tensorflow.org/)
[13] Keras (https://keras.io/)

# 11. ADDENDUM - SPEECH TECHNOLOGY

## 11.1. Speech Recognition and Text To Speech Systems

Voice and Speech Recognition is a software that helps users to control computer functions and convert voice input into text. In a Speech Recognition System there are two main components: the first component is for processing signal which is captured by microphone and the second component is to translate the processed signal into words. A speech system consists of a speech recognition engine called Automatic Speech Recognition (ASR) and of a system for delivering voice feedback, called Text To Speech (TTS).

The areas of use of speech technology are different and in principle, in every task where a user is required to interface with the computer, it is possible to use the ASR and TTS systems.

However, the applications that most commonly use speech recognition systems are:

- **Dictation:** The dictation is certainly the application making greater use of ASR systems
- **Command and Control:** Command and Control (C&C) systems are ASRs that are designed to perform particular functions and actions on the system, such as "open notepad".
- **Telephony:** Some voicemail/PBX systems allow callers to vocalize the commands they want to perform rather than require the corresponding buttons to be pressed.
- **Medical Field/Disability:** Many people have trouble using the keyboard due to injuries induced by repetitive strain injuries (RSI), muscular dystrophy and other causes. For example, those with hearing impairments could connect and ASR system to their phone to convert the caller's voice to text format.
- **Embedded Applications:** Some cell phones have C&C voice recognition and recognize expressions such as "call home".
- **Speech synthesis** technologies are very useful in the field of disabilities because they enable blind people for example, to listen to the feedback received without owning expensive Braille devices.

## 11.2. Automatic Speech Recognition (ASR)

Speech recognition is the process of converting acoustic signals, consisting of voice and noise, into a corresponding set of words. A voice recognizer receives an audible signal and returns the corresponding text string.

The output of the speech recognition process can then be used to enter data in the system, for example the dictation of a text to a word processor, to control computer systems or as an input for more complex systems such as text comprehension systems, which use sophisticated artificial intelligence algorithms to presume the meaning of a sentence or a text.

### 11.2.1. Historical background

The first speech recognition systems have been developed last century by both independent and university researches centers and are the result of studies in the field of Artificial Intelligence.

The aim was the creation of machines able to converse without the need to learn new languages and whose behavior emulated human intelligence as much as possible.

The first attempts to realize a voice recognition date back to the 40s, when the US Department of Defense financed a project of automatic translator. The goal was the translation of the unverified

Soviet transmissions, and therefore required the realization of systems for voice recognition and automatic translation. The project did not lead to acceptable results but had the merit of making scientists more aware of the conceptual difficulties linked to this objective.

The following are some major landmarks.

In 1952, at the Bell Laboratories, David, Biddulph and Balakesh built a system for the recognition of isolated figures for a single speaker. In 1959, at the University College of England, a system was built to recognize four vowels and nine consonants.

A new great effort for the creation of speech recognition systems for continuous speech was produced in the late 70's. The DARPA (Defense Advanced Research Projects Agency) project was launched and specialized research centers were founded with the aim of creating recognition systems for large vocabularies (>1000 words) with high accuracy.

The first results, limited to the recognition of few and simple sentences, clashed with the very limited computational capabilities, compared to the current ones, of the computers available at the time: to have real-time recognition, 50 computers had to be run in parallel.

Since then, there has been an extraordinary increase in the performance of recognition systems, passing from the simple recognition of single digits in discrete speech (it was necessary to insert a short pause between one word and another) up to the current systems for large vocabularies (tens of thousands of words) in continuous speech, capable of great accuracy even on a telephone line.

The older systems are not very flexible because they are built on a language model based on finite state automation and because they are based on the recognition of single words according to a very small vocabulary in their possession. Obviously, the accuracy of the recognition is greatly reduced, for example when dictating phrases that do not contain words present in the dictionary or not completely corresponding. The newer systems, on the other hand, in addition to being equipped with a wider vocabulary (even up to 50000 words) use a context-sensitive approach.

Thanks to complex algorithms of artificial intelligence, these speech recognition systems are able to identify words do not present in the dictionary or to reconstruct a sentence based on the presumed meaning of the same. They also offer a greater tolerance to environmental noise, a multi-language functionality, the recognition of continuous speech and even the speaker recognition.

### 11.2.2. Types of Speech Recognition

One first important distinction is between **speaker-dependent** and **speaker-independent** recognition systems.

- The first ones mostly used for dictation (in which accuracy must be very high) can recognize commands and words pronounced only by the person who has performed an initial phase of training within the system. This so-called acoustic training consists in the reading of a pre-defined text and allows the acoustic models found in the recognition software to be adapted to a particular user. If the user changes, the system will have to be trained once again to create a new acoustic profile.
- On the other hand, the situation is different in the speaker-independent systems that are able to recognize commands pronounced by any person without requiring any preliminary training from the user. For this reason, they reach lower levels of accuracy than speaker-dependent systems and are therefore better suited for information analysis systems than for dictation systems.

It is important to note the different sensitivity to errors in the two systems. In the dependent recognition engines, mostly used for dictation of texts, the so-called word error is relevant. In the speaker-independent ones, on the other hand, the semantic mistake is the relevant one, i.e. it is important that the meaning recognized by the system corresponds to the one dispensed.

A second distinction is to be made between **discrete speech recognition** and **continuous speech recognition** (or natural language) systems.

- With the systems belonging to the first category the speaker has to make a small pause between one word and the other and the recognition happens word by word, i.e. the system analyzes the sounds between the pauses and then tries to build the corresponding textual forms. It is evident that such systems are better suited to impart, for example to the pc, simple voice commands corresponding to the most common operations than for dictation of texts. The recognition engines belonging to this category also works with dated hardware, as the amount of processing required is modest.
- The second category, that of the continuous speech, allows a voice interaction with the system through natural speech (therefore without forced pauses). Therefore, individual words are not recognized, but only whole sentences. In this case it is important to train the system to recognize the user's voice because the speech recognition is based on the comparison between a specific voice profile for each user and the words that make up the specified phrase. The efficiency of the system therefore increases with its ability to recognize inflections or particular accents in the specific user's pronunciation.

The last differentiation is between **contest-independent** recognition systems, therefore able to understand any type o question or request, and **contest-dependent** ones, able to understand users' requests within a defined contest (medical, meteorological etc.).

### 11.2.3. The recognition process

A typical speech recognition system has a structure as depicted in the Figure 33:

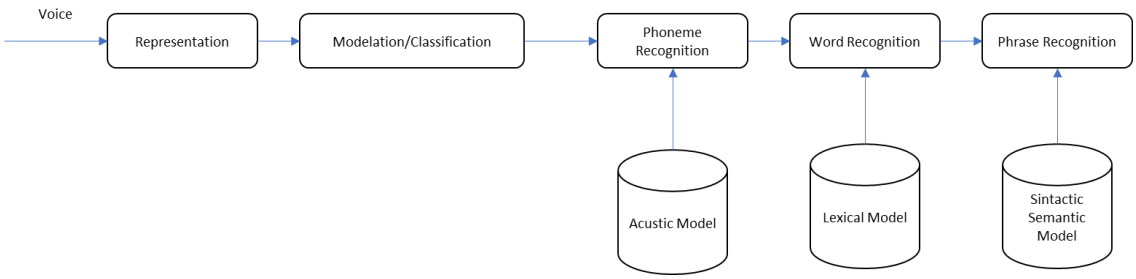



**Figure 33: structure of a speech recognition system**

The recognition process takes place through a series of consecutive steps, which can be integrated into a single logical chain, from which one can decide to go out at any stage, thereby settling for the result obtained until that passage.

First of all, the user delivers a series of sentences that are acquired by the system through a microphone for example, in order to have a digital representation of the sentence pronounced.

One then moves on to an operation indicated in the diagram as "modelling/classification" consisting in the system scanning the sentence in order to separate the individual words, which will then be subdivided into individual phonemes. Thus, the phase of recognition actually starts: after a period of recognition of the previously decoded phonemes, the system takes care, in succession, of the recognition of the words they form and of the entire sentence pronounced by the user, in order to generate the corresponding text string.

### 11.2.4. Recognition of phonemes and words

As mentioned, the first step of the actual speech recognition consists in the recognition of the phonemes, in other words, a translation to a sound level of what the system has previously extracted from the speech pronounced by the speaker. There is a great variability of the acoustic forms (sounds) associated with the individual phonemes, a phenomenon due to numerous factors such as, for example, the sex and the accent of the speaker, the background noise. It is therefore necessary to represent the acoustic phenomena related to the voice with complex statistical models (trained on large amounts of data of real recordings) as it is clear that the identification of a phoneme (and therefore the identification of a particular letter or sequence of letters) it is almost never a certain event, but rather a more or less probable event.

For example, the sounds produced by different speakers in pronouncing the vowel "a" and the vowel "i" will constitute two classes representing the set of acoustic achievements of the two phonemes that can be associated with the respective vowels. Since it is possible to distinguish the sound of an "a" from that of an "i", there must therefore be some "average" acoustic characteristics which make it possible to distinguish the two classes and to associate all the sounds of a class with the same phoneme.

An acoustic model describes these average statistical properties and is compared with the phonemes generated in the previous phase of the recognition so as to proceed to a possible identification of these. To increase the likelihood that the identification is accurate, it is also necessary to analyze the phonemes preceding and following the one in question, and verify that the set of probable identifications is itself probable; in this regard, the vocabulary of words to be recognized helps to restrict the number of possible combinations, since the subsequent phoneme must be combined in an appropriate manner with the previous ones so as to constitute a valid word. In other words if, for example, the letter "o" is recognized with a certain probability and it is in an identification sequence that proposes "d" - "o" - "g", since the word "Dog" exists in the vocabulary, the probability of a correct recognition increases.

One then moves on to words recognition, a phase that makes use of a lexical model of the terms used by the speaker to improve the final result. Numerous alternative hypotheses can therefore be constructed which are then evaluated on the basis of their consistency with the words present in the model: the hypotheses that have no relevance to the words contained in it are eliminated, even if they are the result of a composition of phonemes that had obtained a very high single score in the previous phase of the recognition process.

### 11.2.5. Problems affecting speech recognition

The speech recognition technologies are mainly affected by errors due to the use of probabilistic recognition algorithms, which sometimes resort to inappropriate grammars, and to the presence of external factors that introduce noise in the analyzed signals. The so-called environmental noise (people talking in the same room, passing vehicles, background music, etc.) is difficult to isolate due to the impossibility of obtaining its footprint before the sampling and to its variability over time. On the other hand, the noise introduced by the instruments (for example the classical background noise of the microphones) is different and can be eliminated by performing a preventive sampling of the noise and subsequently subtracting it from the signal being processed.

Another problem is the so-called acoustic variability: the phonemes produce different acoustic effects depending on the context in which they are pronounced. The sound footprint of a phoneme varies depending on the environment in which the sound is emitted (for example, in the mountains the sounds may be subject to the echo effect) and the quality of the instrument used for sampling.

## 11.3. Text To Speech (TTS)

A speech synthesizer or Text To Speech Engine (TTS) is a so called “text reader”, namely a system capable of reproducing in output a vocal version of any text supplied to it, either directly by the operator or by scanning, using an OCR (Optical Character Recognizer).

There are therefore many uses of a speech synthesizer: vocalization of textual / visual contents, for example web pages, of database contents, books, documents, bibliographic references, etc.

The hardware used by any synthesizer is normally constituted by the simple sound card present in personal computers, which, in most cases, it provides acceptable results, but you can also buy sound cards specialized in voice production, to be included in addition to the supplied sound card.

A speech synthesizer should not be confused with a so-called voice response device, such as the automatic system used at some railway stations for announcements. Such devices are based, in fact, on the simple concatenation of isolated words or parts of sentences and are used when a very limited vocabulary is required (in the order of hundreds of words) and when sentences must be pronounced respecting a fixed and defined syntactic structure.

A **speech synthesizer**, on the other hand, must be able to reproduce vocally any text regardless of its nature and origin and it is therefore unthinkable to memorize all the words of the spoken language. This is why it is more than legitimate to define a speech synthesizer an automatic speech producer.

There are two factors that characterize the quality of a TTS system: the comprehension (intelligibility) of the produced speech and the naturalness with which it is pronounced.

It is therefore necessary to obtain a vocal reproduction that is as close as possible to that of a human being: not only clear from the point of view of pronunciation and syntax, but also natural and fluid in reaching the listener's ear. From this point of view, giant steps have been made in the field of synthesizers: the most dated ones, characterized by the metallic sound of the voice, offered a poor quality synthesis, especially when it came down to naturalness, while the modern ones are getting closer and closer to human speech, so much as to replicate its emphasis and pauses in pronunciation.

Modern synthesizers also offer multilingual support and multi-voice support. The first consists simply in the possibility of vocally synthesizing texts written in any language, even with mixed-language functionality, while the second consists in the possibility to choose and customize the voice used for the synthesis: one will have the option of choosing the speaker as per a well-defined voice profile (type of voice, inflection, hue, emphasis, etc.) Thus, selecting for example "John", the synthesizer will provide a voice output as similar as possible to that of a male figure with a low and calm voice, while selecting, for example, "Sara" you will hear a female and maybe even sharp voice.

### 11.3.1. Historical background

Attempts to realize systems for the generation of human voice - speech synthesis systems - date back to 1779 in St. Petersburg, when the Russian professor Kratzenstein built acoustic resonators capable of producing the sounds of the five vowels. Wolfgang von Kempelen who built the “mechanical-acoustic speaking machine” made another attempt in Vienna five years later.

This machine consisted of a bellows that produced a flow of air that, through a suitably deformed leather tube, generated the sounds of the vowels. Beyond the outcome, these studies were very important because they showed that the oral cavity is the fundamental element for the co-articulation. Before that, one believed that the production of the sounds related to the voice occurred at the larynx level.

The first voice synthesis system of the modern era - always mechanical - was the VODER (Voice Operating DEmonstratoR), presented by the Bell Laboratories in 1939.

VODER simulated the voice production mechanism, now well known in its fundamental dynamics:

when we speak, a basic sound, produced by the air of the lungs passing through the vocal cords, is modulated by the oral cavity, the nose and the mouth. The position of the different parts of the tongue and the position of the lips are responsible for the different sounds of the voice.

VODER was a kind of musical instrument. A vibrating bar generates the fundamental frequencies, variable through a pedal mechanism. The sound produced was modulated using fingers controlled acoustic filters. The quality of the synthetic voice was obviously very poor, but it proved that a synthetic voice was possible.

Subsequent attempts to synthesize the voice used digital technology, thus opening a new era for the construction of automated systems. The research focused on three different approaches to automatic voice generation:

- **Articulatory synthesis**: one tries to build accurate mathematical models of the human phonatory system. It is the theoretically most appropriate and flexible method, but also the one that involves the greatest difficulties and the greatest computational load.

- **Formants synthesis**: where basic acoustic forms are identified - the formants. The voice is synthesized through a combination of formants. This technique leads to the generation of a large quantity of voices, guaranteeing great flexibility. However, the quality of the voices is not particularly high.

- **Synthesis by concatenation**: where dynamically linked pre-recorded voice fragments are used. This method currently guarantees the best quality synthesis (almost indistinguishable from the original item).

### 11.3.2. Architecture of a speech synthesizer

A speech synthesizer consists of two main modules, the NLP (Natural Language Processing) and the DSP (Data Signal Processing).

The NLP module has the task of "reading" the text input into the synthesizer and then, through a phoneme generator, of producing a corresponding phonetic transcription and finally associate the information related to the desired prosody (the set of characteristics concerning intonation, intensity and duration of speech).

The DSP module, on the other hand, has the task of producing the actual speech, according to the information received from the NLP module.

The algorithms and formalisms used can speed up some phases of the speech synthesis process and sometimes involve restrictions on the pronounced text or the reduction of the expressiveness of the synthetic voice (obviously compared to the human voice), but allow completing the synthesis operation using a limited amount of memory resources.

## 11.4. Distributed Speech Recognition (DSR)

### 11.4.1. The DSR in synthesis

Given the increase in network performance and the increasingly widespread use of low-level mobile computing devices, such as Smartphone, several software houses have started to include voice commands to operate applications, replacing or integrating the navigation using the classic graphical interface and the icon navigation system.

In many cases, distributed systems are adopted in which the computational burden is left to one or more servers with software devoted to processing voice signals; the devices receive only the result of the processing as a reply to the sending of the recorded audio.

To prevent the low **bitrate** of some communication channel from affecting the quality of the recognition, the recorded audio is compressed and packaged with an error protection system and then sent to the server that will be responsible for decompressing the packages and rebuilding the audio. It will then be sent to the speech recognition engine.

The block diagram below shows the difference between a DSR and a traditional **remote recognition system.**

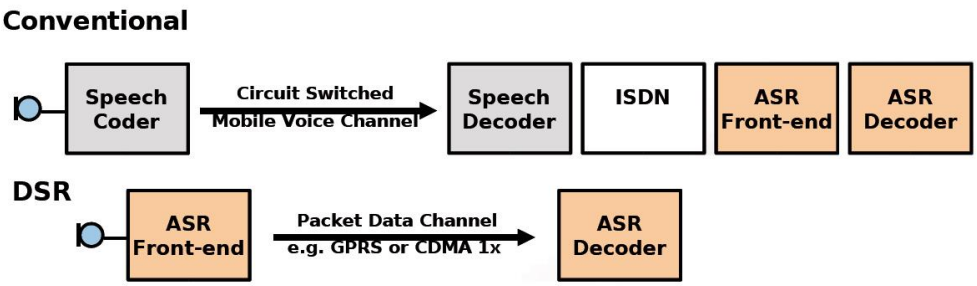



**Figure 34: Schema of Distributed and traditional ASR Systems**

### 11.4.2. DSR Architecture

From recent studies, it is clear that the distributed recognition has reached levels of reliability equal to the most widespread local ASR engines.

The diagrams of two distributed voice systems are depicted in Figure 35 and Figure 36

In addition to the audio packages, in the first diagram (Figure 35) additional information to optimize the performance of speech recognition is also sent.

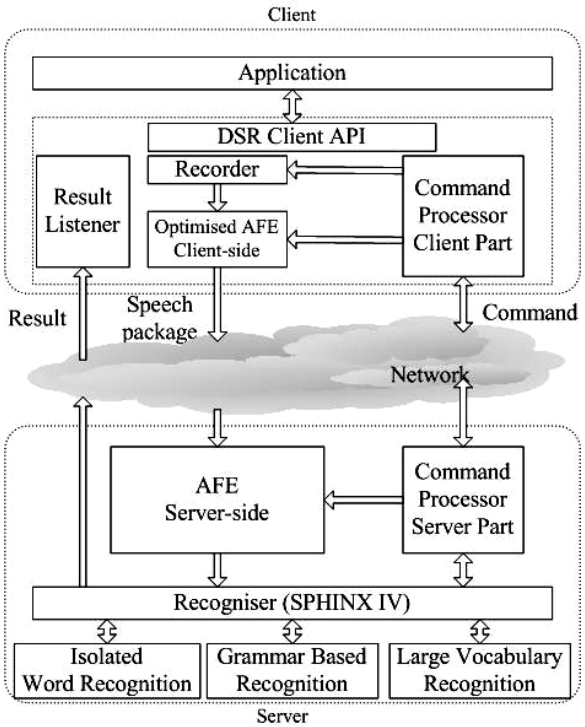



**Figure 35: Architecture 1 of a DRS system**

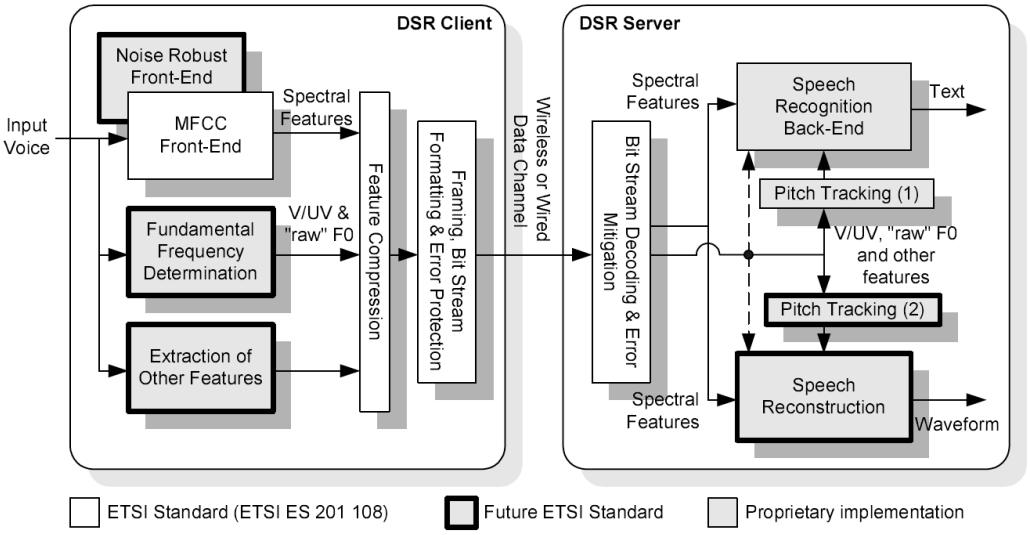



**Figure 36: Architecture 2 of a DRS system**

In most of the current DSR systems, the application interacts at a high level and via API with the client-side recognition system that will be responsible for compressing and sending to the server all the information for processing the data and for giving the feedback, often in the form of text string.

## 11.5. Dialogue and VUI

#### *11.5.1.1* introduction

Voice user interfaces (VUIs) allow the user to interact with a system through voice or speech commands. Virtual assistants, such as Siri, Google Assistant, and Alexa, are examples of VUIs applications in everyday life. The primary advantage of a VUI is that it allows for a hands-free, eyes-free way in which users can interact with a product while focusing their attention elsewhere.

Applying the same design guidelines to VUIs as to graphical user interfaces is impossible. In a VUI, there are no visual affordances; so, when looking at a VUI, users have no clear indications of what the interface can do or what their options are. When designing VUI actions, it is thus important that the system clearly state possible interaction options, tell the user what functionality he/she is using, and limit the amount of information it gives out to an amount that users can remember.

Because individuals normally associate voice with interpersonal communication rather than with person-technology interaction, they are sometimes unsure of the complexity to which the VUI can understand. Hence, for a VUI to succeed, it not only requires an excellent ability to understand spoken language but also needs to train users to understand what type of voice commands they can use and what type of interactions they can perform. The intricate nature of a user's conversing with a VUI means a designer needs to pay attention to how easily a user might overstep with expectations. This is why designing the product in such a simple, almost featureless form is important—to keep the user mindful that a two-way "human" conversation is infeasible. Likewise, the user's patience in building a communication "rapport" will help improve satisfaction when the VUI, becoming increasingly acquainted with the speaker's voice (which the speaker will use more effectively), rewards him/her with more accurate responses.

Once the spoken message has been transformed into text, it is necessary to equip it with a "semantic", namely a meaning. This is because the process of transforming the voice into text simply provides an alphabetic sequence, but the sequence has no meaning.

This is comparable to the case in which a person perfectly knows a language from a phonetic-lexical-syntactic (non-semantic) point of view and is therefore able to write exactly what he listens to, not understanding the meaning of what he writes. The language models simply serve to specify the sequences of admissible phonemes, but no semantics is yet associated with the text. An additional application layer is dedicated to the "understanding" of the text.

This is what distinguishes dictation programs - where no text comprehension is carried out - from dialogue systems, which must be able to "dialogue" with the user in a sensible way, attributing "a meaning" to what the user says. Understanding the meaning of the text is evidently a crucial element for building systems able to dialogue using natural language.

The creation of automatic dialogue systems able to manage complex services, such as automatic help desk systems, must take into account the interaction of the application layers, voice recognition, analysis of the meaning, processing of the response to be provided in the subsequent dialogue step.

It is extremely complicated to make complex dialogues in natural language using the usual programming rules. In the traditional approach, the flow scheme that describes exhaustively the behavior of the system is normally defined.

A program that implements the automaton corresponding to this flowchart is then created. This approach works for simple sequential dialogues but is very inefficient or even unmanageable for complex dialogues, for example a free dialogue for flight reservations or a help desk system.

A much more effective approach is to define the set of rules that describe (according to some formalism) the behavior that the dialogue system must have.

This set of rules is subsequently interpreted by a universal dialogue manager (i.e. not dependent on the particular dialogue) that will perform a series of actions as specified by the rules themselves.

For the creation of interactive dialogues between the user and the computer, a series of languages have been defined, such as VoiceXML (VXML), [9] which stands for Voice eXtensible Markup Language.

Traditionally the VoiceXML platform works in a similar way to an HTML browser: VoiceXML documents are downloaded from a web server and interpreted and transformed into voice by a Voice Gateway found on the end-user's computer.

The Voice User Interface (VUI) is nothing more than the voice interface of an application or of a speech service and it is the voice equivalent of the GUI that we normally use for interaction with programs. While in the GUI we press a button, with the VUI one can give a voice command.

#### *11.5.1.2* VoiceXML

VoiceXML is a language for creating interactive voice interfaces, especially in the telephone environment. It uses voice recognition and tone keyboards as input, pre-recorded audio and TTS synthesis as output.

You can access a VoiceXML application using a simple telephone through the VoiceXML interpreter (browser) of a telephone server.

VoiceXML is an XML-based language that combines several features of the most common and widespread languages:

- *It is declarative* (as HTML): on a simpler level, VoiceXML applications describe a relatively static sequence of interactions with the user as it happens for HTML web pages.
- *It is procedural*: creating interactive applications using a purely declarative language often does not guarantee the necessary flexibility. VoiceXML provides full support for ECMAScript (JavaScript), which allows you to use a significant number of dynamic variables, procedures and functions within a single document.

- *It is driven by events*: voice applications are not linear; therefore, developers must take into account the fact that at any moment an event can occur. VoiceXML takes into consideration that developers can define a series of additional events that are used to manage particular events when they occur.
- I*t is object-oriented* (like C ++): VoiceXML includes a number of constructs in which objects have a set of properties that can be verified and manipulated.

VoiceXML is simply a possible syntax to describe a conversation between an automated system and a caller. An application must exhaustively describe the behavior of an automated system in response to an unpredictable audio input of a caller. The VoiceXML language is therefore essentially an attempt to provide developers with all the necessary tools to express this interface.

An example of a VXML file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
   http://www.w3.org/TR/voicexml20/vxml.xsd"
   version="2.0">
 <meta name="author" content="John Doe"/>
 <meta name="maintainer" content="hello-support@hi.example.com"/>
 <var name="hi" expr="'Hello World!'"/>
 <form>
  <block>
     <value expr="hi"/>
     <goto next="#say_goodbye"/>
  </block>
 </form>
 <form id="say_goodbye">
  <block>
     Goodbye!
  </block>
 </form>
</vxml>
```

#### 11.5.1.2.1 VoiceXML evolution

On March 1999, AT&T, Lucent Technologies, IBM and Motorola announced the launch of the VoiceXML Forum, an IEEE Industry Standards and Technology Organization program. Prior to the adoption of VoiceXML, all the groups participating in the Forum [10] had developed various proprietary speech technologies.

The VoiceXML Forum has the following basic objectives:

- Develop a specification opened to standardization;
- Demonstrate to the industries the need for a single standard for vocal makup languages;
- Attract industries to participate and support the Forum;
- Promote the widespread use of the resulting standard to create innovative applications and services.

On March 2000, the VoiceXML 1.0 specification was submitted for approval by the World Wide Web Consortium (W3C), which approved it in May. On October 2001, W3C [11] announced the first release of VoiceXML version 2.0.

#### 11.5.1.2.2 VoiceXML architecture

The VoiceXML architecture is structured as detailed below:

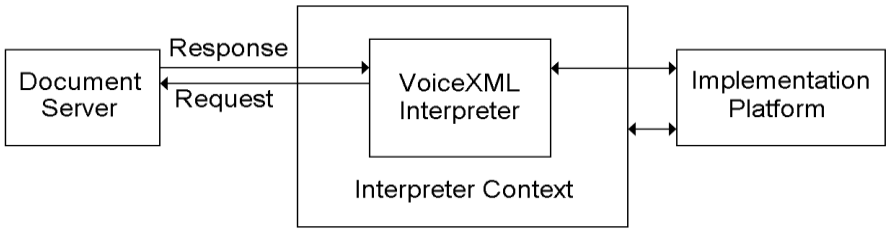



**Figure 37: VXML Architecture**

The main part of a VoiceXML architecture is the Voice browser. This is the software that transforms the VoiceXML markup into a sequence of two types of dialog between system and user. It consists of an integrated VoiceXML interpreter with software components for TTS and audio file output and for speech recording and recognition.

A document server (such as a web server) processes requests from a client application through the VoiceXML interpreter. The server produces a VoiceXML document that is in turn processed by the VoiceXML interpreter. The VoiceXML Interpreter Context has the task of monitoring, in parallel with the VoiceXML interpreter, the input coming from the user. Both the VoiceXML interpreter and the VoiceXML Interpreter Context control the implementation platform. For example, in an interactive voice application, the VoiceXML Interpreter Context can be responsible for accepting an incoming call, acquiring the initial VoiceXML document and answering the call, while the VoiceXML interpreter takes care of the dialogue after the call is answered. The implementation platform generates events in response to particular user actions or particular system events.

The VoiceXML browser refers to the dialogue between the user and the VoiceXML application, proceeding with the interactions specified by the application itself. It can go to the next dialog, switch to a new VoiceXML document or provide information to be processed to a Web application server. A Web-based voice application can be implemented through a large set of different technologies such as Java servlets, JSPs, ASPs and other server-side scripts.

#### 11.5.1.2.3 Advantages and disadvantages of VoiceXML

The use of VoiceXML has many advantages with some disadvantages. Below are some positive aspects of its use:

- It is equipped with a free interface that provides a wide range of messages and possible applications;
- It keeps the user interaction code (markup) separate from the server logic (php, ASP.NET etc.);
- It defines a markup language that can handle simple basic cases without precluding the management of complex cases;
- Most companies today have a Web infrastructure and an IVR infrastructure, VoiceXML eliminates this separation;
- VoiceXML solutions allow you to leave the applications, the platform and the speech part separate and to choose the best solution for each context;

Disadvantages include:

- People with strong accents or speech difficulties probably do not see their voice recognized. The same problem occurs if the user speaks in an extremely informal way;
- Delays in TTS and ASR can be high in some cases, especially if the network is working at its maximum capacity;
- Using this technology in a very noisy environment can cause problems because the ASR device may not be able to filter too much noise. In this context, we are often satisfied with having the data entered through the telephone keypad.

#### 11.5.1.2.4 Structure of a VoiceXML application

Each VoiceXML document consists of one or more dialogs. The peculiarities of the dialogs concern the collection of inputs, the generation of audio outputs, the management of asynchronous events and the progression of the dialogs themselves.

Applications use a root folder for shared data and dialogs. The sub-dialogs are used for the decomposition of the dialogs and their possible re-use (e.g. library dialog).

An application is a collection of VoiceXML documents that share the same root application. This remains active as the user moves from one document to another in the same application and is deactivated when the user changes to a document belonging to another application. When it is active, the root application variables are available as application variables, and its grammars can also be set to remain active for the duration of the application.

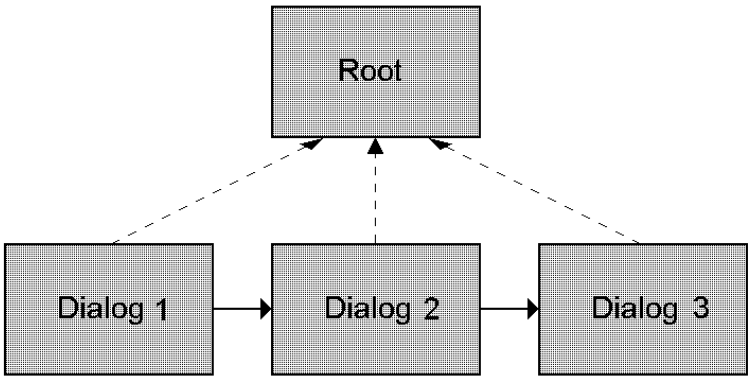



**Figure 38: Structure of a VoiceXML application**

The user is always in a single dialog at the time. Each dialog determines what will be the next. The transitions are specified using the Uniform Resource Identifier (URI) which define the next document or the next dialog to be used. If a URI does not refer to any document then the current document is kept, similarly, if a URI does not refer to any dialog then the first one is chosen. The execution of a dialog ends when it does not refer to any successor or if it contains an element that explicitly terminates the conversation.

There are two main types of dialog: *directed dialog* and *mixed initiative dialog*. The first consists simply of a dialog that contains a certain number of elements that are inserted in a sequence, each of which requires a single operation at a time. A mixed initiative dialog, on the other hand, serves to implement patterns that simulate a more natural conversation. They provide a more complex initial application to allow the user to provide more information in a single response.

The basic concepts referred to by VoiceXML are described in more detail below:

- **Dialog:** There are two types of dialog: form and menu. The first one defines an interaction that collects the values for the input variables. Each field must specify a grammar that defines which are the acceptable inputs for that field. A menu instead presents the user with a series of options, the choice of the user determines the transition to another dialog.
- **Sub-Dialog:** They are subjected to dialogs and can be reused perhaps for acquisitions of standard data (e.g. the collection of credit card numbers, etc.). At the end of its execution, the control returns to the dialogue that had invoked it.
- **Grammar:** Each dialog is associated with one or more grammars, each of which is active only when the user is in that dialog. Some dialogs can make their grammars active even when the user is in another dialog within the same document or in another document of the same application. In this case, if the user says something that has correspondence in the active grammar of another dialog, the application passes into the new dialog and the user's expression is treated as if it came from this.
- **Events:** VoiceXML provides mechanisms (forms) to manage normal user input. In addition, it defines a mechanism for handling events not covered by the forms. The events are launched by the platform in various circumstances, for example if a user does not answer, if he responds in an incomprehensible way, if he asks for help, etc. The management of events consists in being able to establish at each level what behaviour to adopt when they arise.
- **Variables:** The scope of a variable represents the lifetime of that variable in a given application. VoiceXML supports five different types of scopes:

    - *Session scope*: it contains the variables declared by the VoiceXML interpreter and belonging to the current user's session. Session variables are available for the duration of the session.
    - *Application scope*: it contains the global variables visible for the duration of the application. To have a variable with this scope, it must be declared within the root application.
    - *Document scope*: it contains the visible variables within the document in which they are declared. To create a variable of this type, you must declare it as an element originated from the vxml element in the chased document. The variable is initialized each time the VoiceXML interpreter loads the document in which it is declared. When the VoiceXML interpreter passes to another document the variable exits the scope, while moving between different dialogs of the same document does not cause leaving the scope.
    - *Dialog scope*: it contains variables visible only within the dialog in which they are declared. To create a variable of this type it must be declared as an element originating from the form element. These variables are initialized each time the VoiceXML interpreter enters the dialog; switching to another dialog causes the variable to exit the scope.

- *Anonymous scope*: it contains variables visible within the block in which they are declared. The VoiceXML interpreter initializes them when it enters the block and destroys them when it leaves the block.

# 12. ADDENDUM – VOICE USER INTERFACE GUIDELINES

## 12.1. What is a VUI and its elements

Voice User Interface (VUI) defines how the application will interact with the user by voice. The user will use voice commands or dictate word and phrases while the system execute actions through the application and responds to the user using voice prompts. In this section we define the guidelines to analyze and design a VUI for a generic application and hence for EasyTV applications.
Typically, a VUI is represented as a set of elements that describes the dialog between the user and the application. Therefore, it will have to include voice prompts, actions carried out by the system, voice commands and a chart diagram representing the flow of the dialog between the user and the application. The chart diagram of a VUI is a representation of a dialog flow where each item of the chart represents a specific dialog state. The dialog state is the most important part of the dialog flow and identifies a specific moment of the interaction between the user and the application. A dialog flow can be organized as sub dialogs each identifying a portion of the VUI and that that can be considered an independent logical unit. The VUI subdivision in sub dialogs allows a modular engineering and comes from an attentive analysis of the application to vocalize. Typically, the dialog can be mapped with the macro functionalities or with the application graphical sections. A VUI can be made of a single dialog flow or of multiple ones, the identification of the right number of sub dialogs depends from the single application. One or more dialog states can be implemented within the dialog flow to allow a structured dialog management

As a general guideline, the design of the VUI for a speech enabled application should follow general usability principles briefly summarized below:

- *Feedbacks*: The user needs feedback from the system to know what is going on during the interaction. When a user issues a speech command, the system should acknowledge the reception. Users also must be given feedback when the system is busy.
- *Confirmation*: Confirmations are the questions that the system could ask to the user to resolve possible ambiguities during the dialog
- *User Expectation*: Users expect the system to understand more than it is capable of, and to be able to provide more information than they can produce. The interface design can help the user to set appropriate expectations.
- *Prompting*: Choose the appropriate words for your text. For example, use the word “say” when you want the user to speak, rather than “enter”, use “enter” for inviting the user to press a key. Moreover, provide different ways of phrasing the same information/request to keep the user interested in the dialog.
- *Non-Speech Audio / Auditory Icons*: Preserve standard sounds with their usual meanings (Siren: Emergency, Beep: Error or Attention and so on).

The present guidelines want to give just some recommendations and instruments to the VUI developers in order to define and keep track of the whole voice interaction scheme and all its core elements that defines the dialog with the application. The following guidelines hence can be used partially or modified based on developer needs.

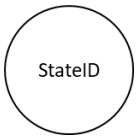
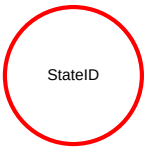
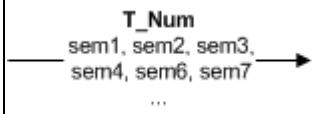
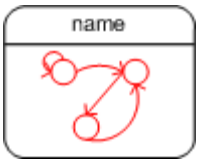
We define the following elements to represent the VUI:

- A chart diagram providing an immediate vision of the dialog flow.
- A set of tables detailing the characteristics of the VUI elements used in the flow diagram.

## 12.2. Elements of a chart diagram

In this section we report the graphical elements that can be used for the representation of a dialog flow diagrams.

### 12.2.1. Legend of the graphic elements

| | |
|---|---|
|  | Dialog State: it identifies a dialog state. The label StateID should be an abbreviation describing exhaustively the implemented functionalities of the state. |
|  | Dialog State: it identifies a dialog state to manage exceptions like error handling or application errors. The label StateID should be an abbreviation describing exhaustively the implemented functionalities of the state. |
|  | State transition: it represents the transition between two states or within the same state, caused by the matching of a semantics. The label shows the reference number (T_Num) of the transition and includes the list of the semantics related to the voice commands that raise the transition. |
|  | State transition in the flow: it states the transitions between states or processes. It is used in the general flow diagrams. |
|  | Synthetic dialog process: it identifies a dialog process where one or more dialog states are involved. It is used as a logical container of a sub dialog. The label name is used to identify generically the dialog processes. |
|  | Event: It indicates an external event that can happen when the application is running and can be either a specific event related to the current state and or general event. |
|  | Import: it joins two states connected by an import link. The state generating the link inherits all the gramma objects of the imported state and the behaviour of their semantics. |
|  | Default Starting Point: it points towards the initial dialog state of the application. |

| | |
|---|---|
|  | Unique Ending Point: it exits from a state of the "flow" diagram if this is to be identified as the only state in which the application can be turned off vocally. |

**Table 38: 12.2.1. VUI chart diagram - graphic elements**

### 12.2.2. Dialog Flow diagrams

In this section we report a list of the dialog flow diagrams that can be used to represent the dialogs and sub dialogs of the VUI.

- The **general dialog flow** diagram is an overview of sub dialogs involved in the VUI. Each sub dialog will contain the dialog states, transitions between different states, events, a default starting point and where applicable the unique exit point. This representation may be replaced in case of complex dialog processes, with a synthetic flow diagram, where all processes are indicated in a synthetic way. Figure 39: Example of a dialog flow diagram and Figure 40: Example of a process flow diagram, are examples of general dialog flow diagrams.

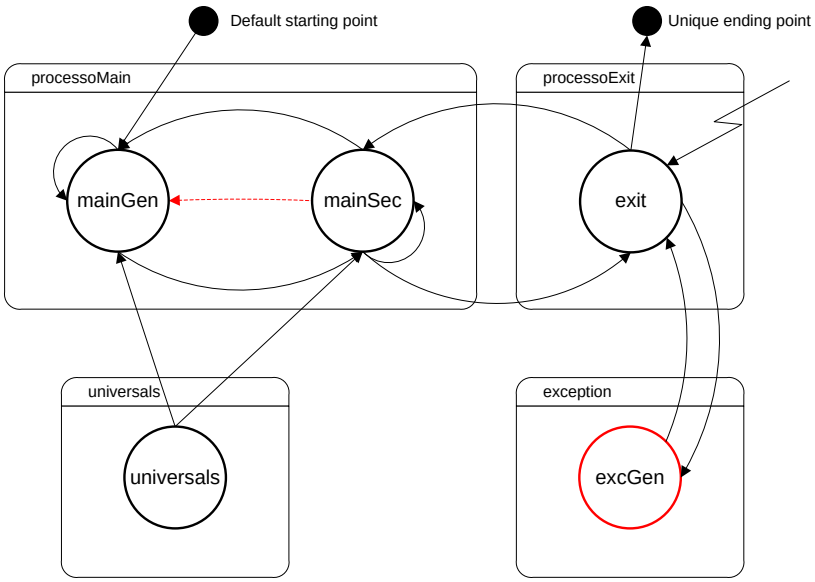



**Figure 39: Example of a dialog flow diagram**

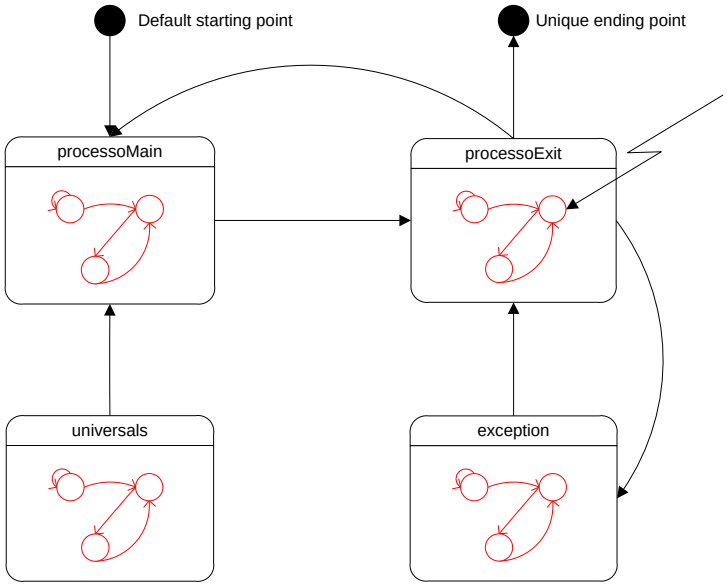



**Figure 40: Example of a process flow diagram**

- A **detailed dialog flow** diagram for each sub dialog involved in the general dialog flow, is a chart diagram where the exit transitions from the different states are clearly stated. The transition arrows will point generically to a next dialog state or to a next sub dialog process. Transitions are typically labeled using group names (T_1, T_2, … T_n) and include the semantic strings that will cause the transition itself during the user interaction. The Figure 41: Dialog Flow - Main Process represents the Main Process with its 5 transitions, these will therefore be labelled with the strings T_1, T_2, T_3, T_4 e T_5. Figure 42: Dialog Flow - Exit Process and Figure 43: Dialog Flow - Exception Process are respectively the diagram of the Exit process and the Exception process depicted in the Figure 40: Example of a process flow diagram.

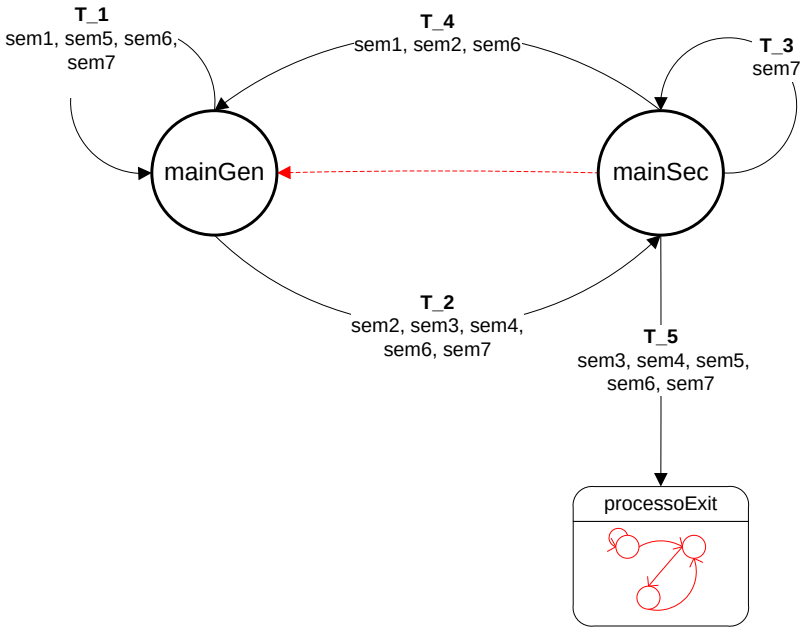



**Figure 41: Dialog Flow - Main Process**

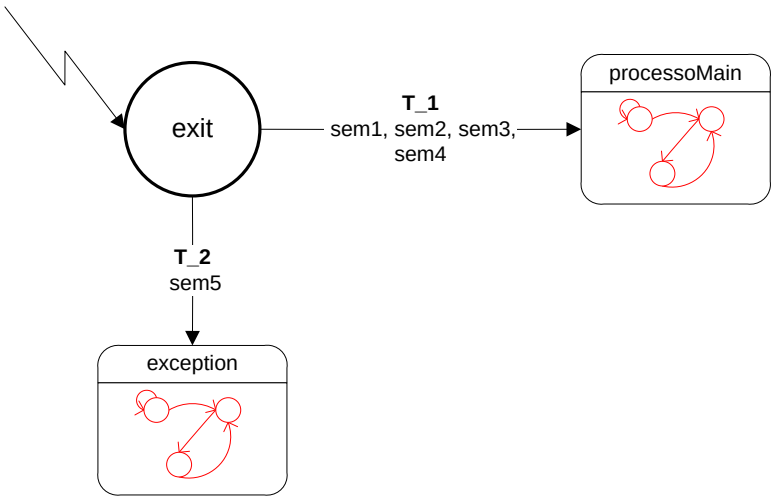



**Figure 42: Dialog Flow - Exit Process**

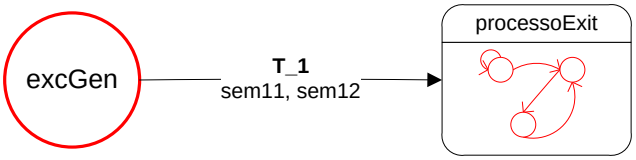



**Figure 43: Dialog Flow - Exception Process**

- The dialog flow diagram for the **Universals sub dialog** can be reported in the VUI to define the general behaviour of the UNIVERSALS grammar objects imported in all dialog states. In the universals dialog state, we will indicate only the transitions caused by the universal semantics identified for the specific application. The labels of such transitions can be written in the following way Tuniv_Num to be distinguished by those relate to the detailed dialog flows (Figure 44: Dialog Flow - Universals dialog flow).

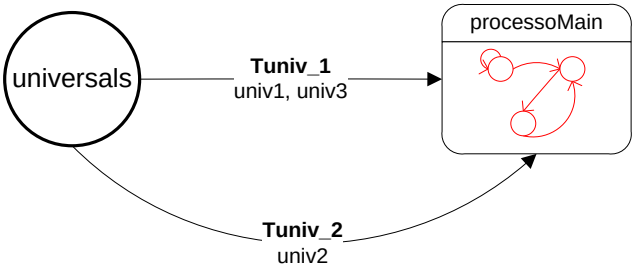



**Figure 44: Dialog Flow - Universals dialog flow**

## 12.3. Tables used to define VUI elements

In this paragraph we define the guidelines to compile the tables that represents the VUI depicted in the dialog state diagrams described above.

The elements to consider when writing the information tables are the following:

- Dialog Flow: identifies a specific portion of the dialog flow (sub dialog) with its dialog states information.
- Dialog States: a dialog state identifies a specific moment of the dialog. Each dialog state identifies also the grammars (phrases that the user can say in that specific context of the dialog), and the imported states, if any, from other sub dialogs.
- State Transitions: they specify the behaviour of the progress of the dialog flow for every voice command interaction or event that happen during the dialog.
- Events: events identify behaviours that happen in the application without a voice interaction. Events can happen in any dialog state or in a specific dialog state. For each event a specific VUI behaviour should be described and formalized in order to manage the progress of the dialog appropriately.
- Semantics: are strings (static or dynamic) of the semantic interpretation of voice commands to be managed by the application that brings to the execution of actions performed by the application and transitions through the dialog states.
- Voice prompts: are the voice phrases that the application will prompt to the user as a response to a voice command.

### 12.3.1. Grammar Object Table

In the grammar object table, we identify all the behavior of the voice commands, their semantics and the transitions to next dialog state of the dialog flow. Information related to the Graphical User Interface (GUI), useful to the developer, can be included in order to facilitate the integration of the VUI in the application.

| Voice Command sample | SEMANTIC | GUI Info | DEST STATE |
|---|---|---|---|
| Example of words or phrase for the voice command | Semantic string | Information about the application screen or other GUI elements | Destination state |

**Table 39: Grammar Object Table structure**

For each voice command (word, phrase, regular expression etc.) the following information should be defined:

- The semantic string of each used to identify the actions that must be taken by the application.
- The destination state of where the VUI will fall after the actions are executed by the application.
- Information about the GUI element or screen (optional) where the actions will take place.

### 12.3.2. Dialog State table

The list of states for a specific dialog flow or sub dialog can be represented in the following table (Table 40: Dialog State Table).

| STATE_ID | GRAMMAR_OBJECT | SPEECH_IN | SPEECH_OUT |
|---|---|---|---|
| Unique Identifier of the dialog state | List of the grammar objects defined for the dialog states. | List of voice prompts when the dialog state is activated | Optional list of voice prompts when the dialog state is left for a new state activation. |

**Table 40: Dialog State Table**

For each dialog state the following information should be defined:

- The list of grammar objects when the dialog state becomes active.
- The list of voice prompts used by the system when the dialog state becomes active
- The list of voice prompts used by the system when the dialog state is left and a new one becomes active.

A grammar object can belong to more than one state since it can be activated in more than one context of the dialog. A single list of dialog state can also be defined in a VUI since the dialog state should have a unique identifier and transitions for each semantic in the grammar object is defined in the grammar object table.

### 12.3.3. Event Table

In the event object table, we identify all the behavior of the application VUI when the event happens. In the Event Table (Table 41: Event Table) we define, for each event, its unique identifier, the event name, the information related to the Graphical User Interface (GUI) and finally the dialog state that should be activated when the event happens.

| EVENT_ID | EVENT NAME | GUI Info | STATE_ID |
|---|---|---|---|
| Unique identifier of the event | Name of the event that is raised | Information about the application screen or the GUI elements | State id of the dialog state to be activated. |

**Table 41: Event Table**

## 12.4. UNIVERSALS Voice Command

UNIVERSALS voice commands are voice commands always available in the whole dialog flow of the application and in every dialog state. They can be included in specific grammar objects. When designing the VUI all UNIVERSALS command should be identified and reported in these grammar objects.

A list of standard Voice Commands that usually should be considered in any VUI are the following:

- REPEAT: usually this UNIVERSALS command prompts the last voice feedback to the user;
- HELP: gives general help or context help to the user to go on with the conversation;
- BACK: usually go back to the previous task executed during the conversation cancelling the last operation performed.
- EXIT: this command will close the application to start a new voice interaction;
- MAIN_MENU: resets the current conversation and starts with a new one from the starting point of the application.