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.

D3.4 – EasyTV Remote control with speech recognition preliminary 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 3.4

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:	D3.4
DOCUMENT TITLE:	EasyTV Remote control with speech recognition preliminary development
WORK-PACKAGE:	WP3
DELIVERABLE TYPE:	Report
CONTRACTUAL DATE OF DELIVERY:	30-11-2018
LAST UPDATE:	29-11-2018
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 Table of Contents definition and 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 sw component description
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

Definitions, Acronyms and Abbreviations

ACRONYMS / ABBREVIATIONS	DESCRIPTION
API	Application programming interface
ASR	Automatic Speech Recognition
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
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
W3C	World Wide Web Consortium

Table of Contents

1. LIST OF FIGURES

2. LIST OF TABLES

3. EXECUTIVE SUMMARY

This deliverable describes the preiminary version of the Speech Platform remote control development according to:

- 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).

We organized the document in two main parts (chapter 5 and 6):

In chapter 5 we describe the architecture of the EasyTV Speech Platform and the components that we have implemented in the first prototype according to the software architecture. Among these components we described the Text To Speech, Speech Recognition and the Semantic Interpretation of voice commands modules.

In chapter 6 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 application domains do define its Voice User Interface (VUI).

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

We also reported in the “Addendum – Voice User Interface Guidelines” the internal guidelines used to define such VUIs and their formal representation.

Moreover, 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 that we have considered and used in our definition and implementation of the EasyTV Speech Platform. In this addendum we talk about Speech Recogniton systems, both Dictation and Command and Control Features as well as local and remote systems (Cloud Based ASR and Distributed ones). Features and architecture of Text To Speech Engine are also reprted in the addendum.

4. EASYTV SPEECH PLATFORM

4.1. Introduction

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

4.2. General Architecture

We started analizing 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 visually-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.

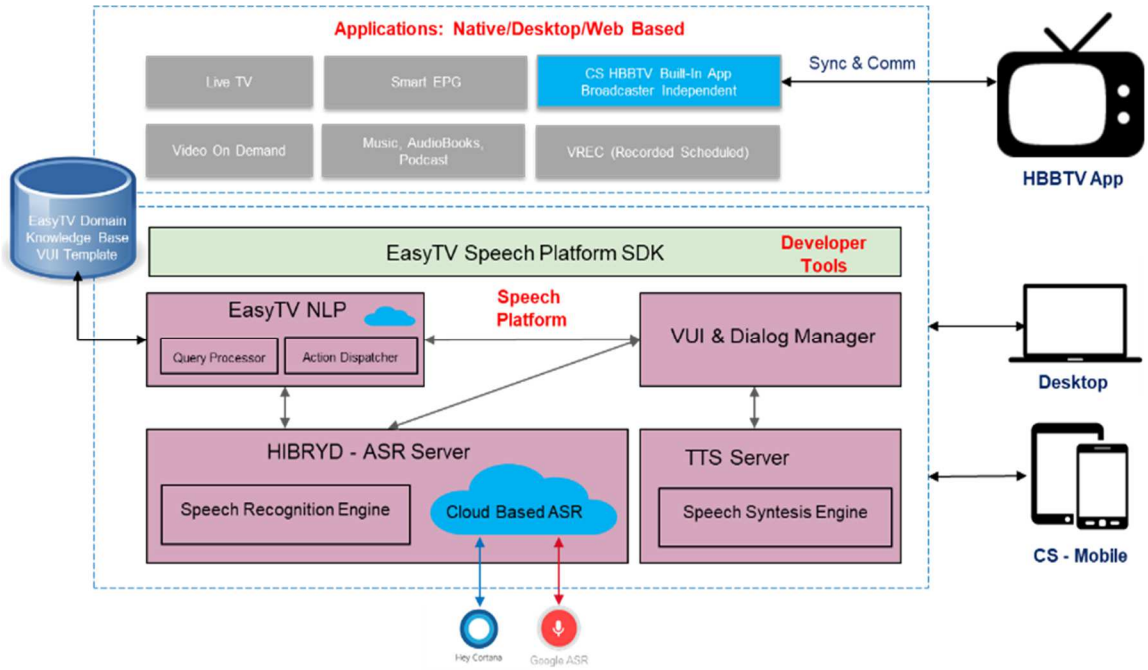


Figure 1: General Architecture of EasyTV Speech Platform

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 ben implemented and experimented usig 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 be 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.

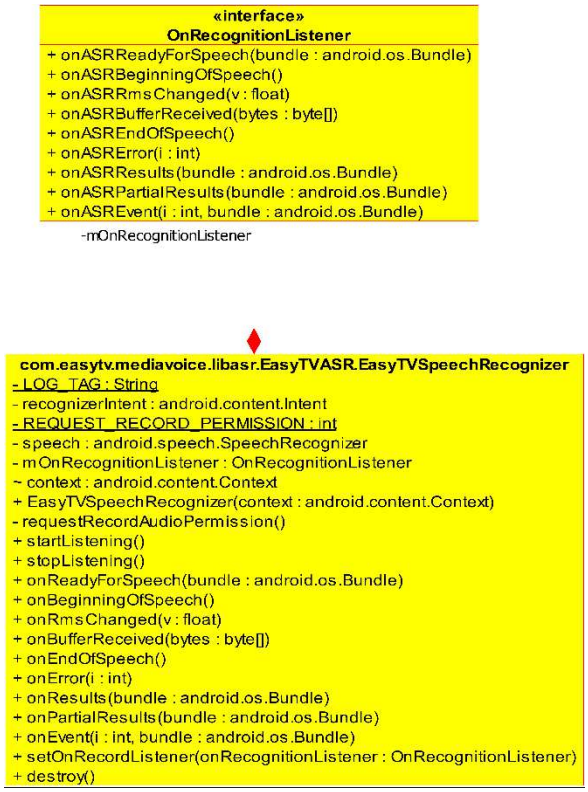


Figure 3: Speech Recognition System - EasyTVSpeechRecognizer Class

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 then 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 aquired from the speech recognition engine.

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

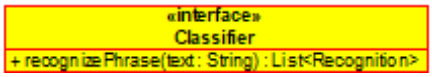


Figure 5: EasyTV NLP Module - Classifier Interface

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.

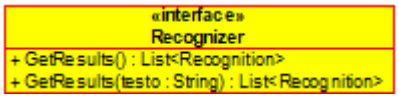


Figure 6: EasyTV NLP Module - Recognition Interface

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

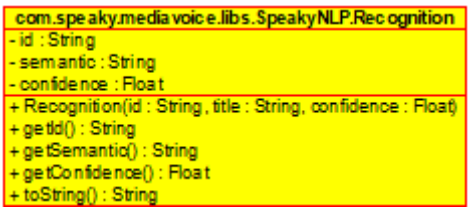


Figure 7: EasyTV NLP Module - Recognition Class

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 bystanding in the BlindMode layer described above. This custom control draws a button that once touched gets animated on the basis of the intensity of the user's voice as a kind of vumeter.

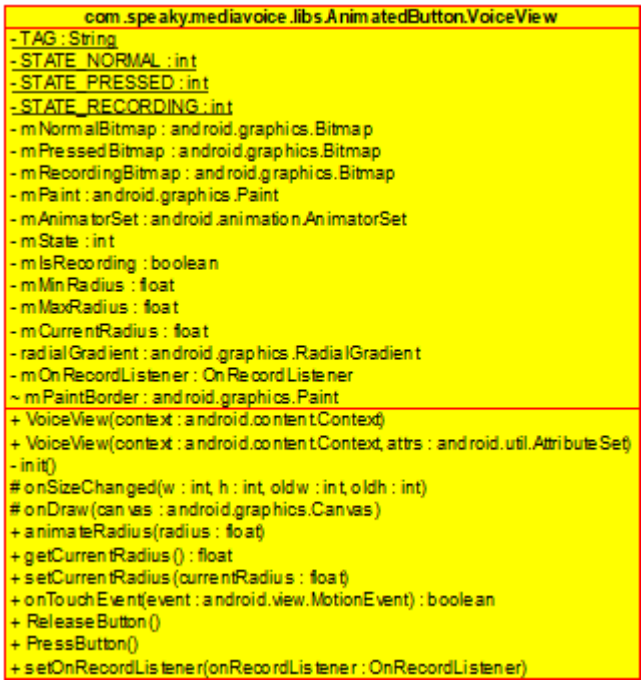


Figure 8: EasyTV Animated Button for Blind Mode

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 at the moment 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 are able to 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.

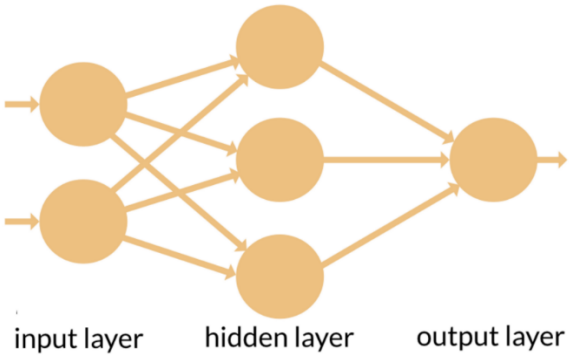


Figure 9: Feed-Forward Neural Networks

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 a 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 a RNN and a Feed-Forward Neural Network.

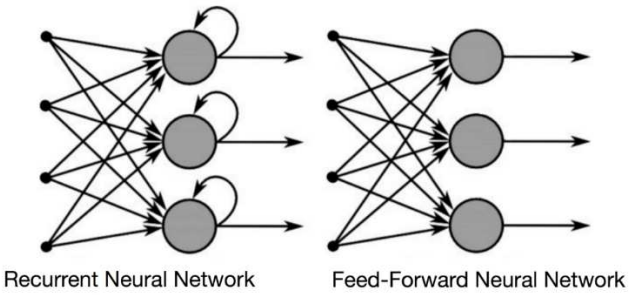


Figure 10: Difference in the information flow between a RNN and a FFNN

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 it’s 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 a 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 also 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).

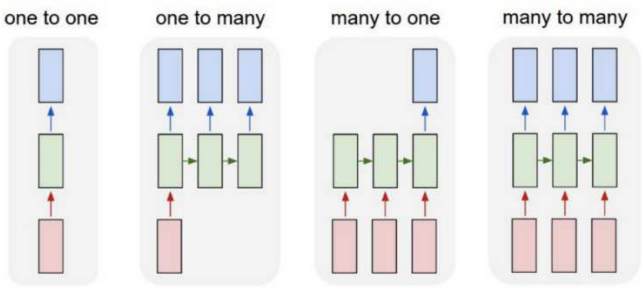


Figure 11: RNN and FFNN input output mapping

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:

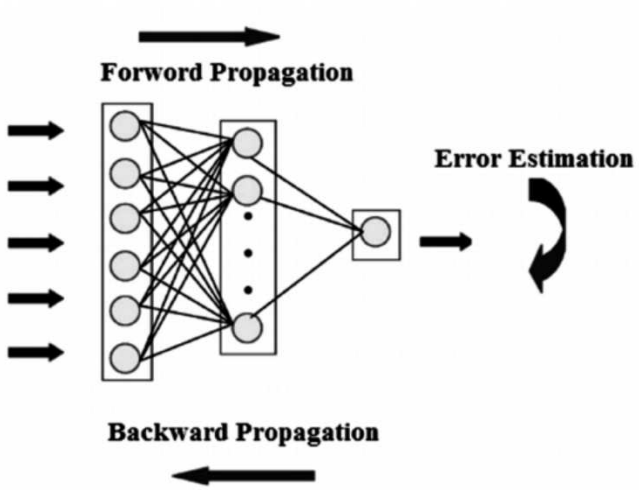


Figure 12: Forward Propagation and Backward Propagation of FFNN

We can view a 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 a RNN can be seen as a sequence of Neural Networks.

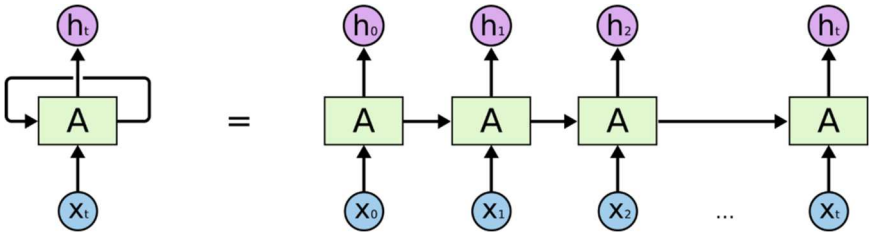


Figure 13: RNN as a sequence of Neural Networks

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 a 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 or not 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 a RNN with its three gates below:

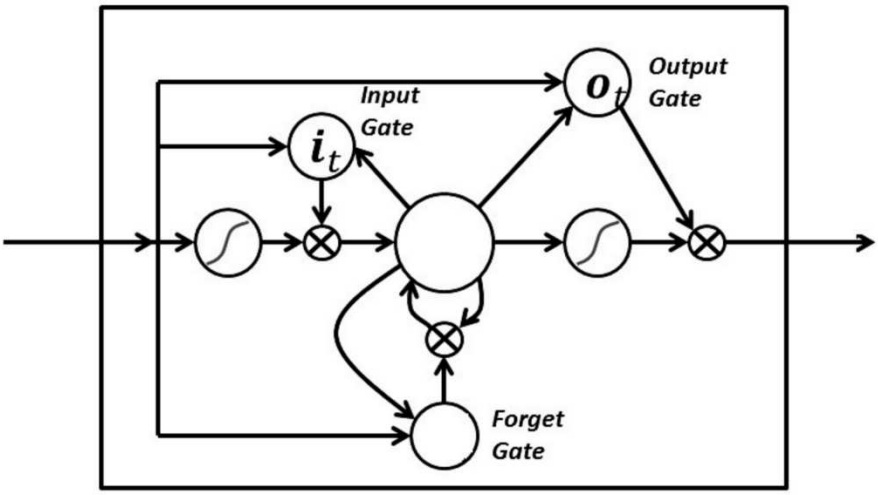


Figure 14: RNN with its three gates

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 is 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 wich 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 learn to categorize.
Input phrases and output intent category have to be transformed before to be passed as input and otuput 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 tecnique 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 cna 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.

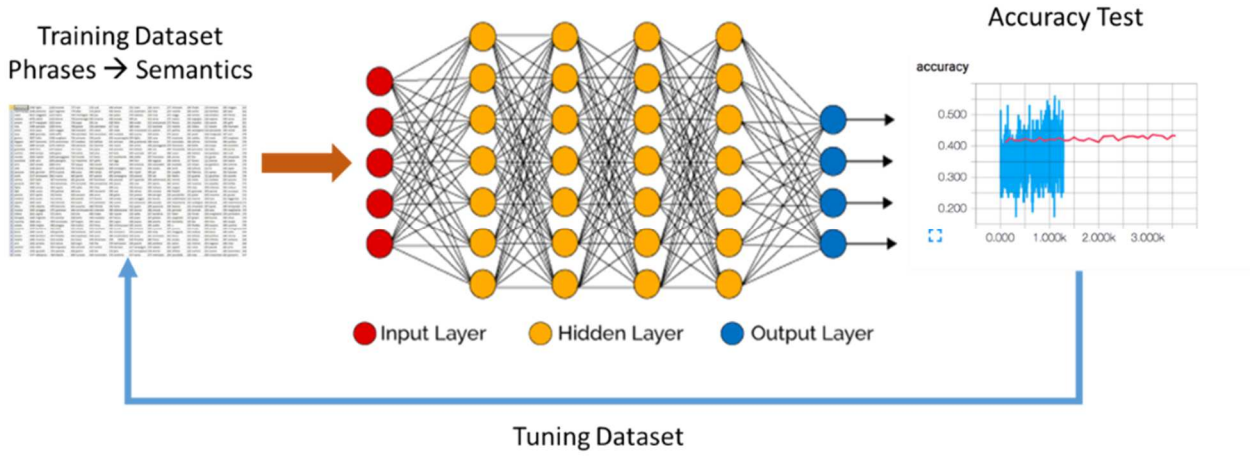


Figure 15: Training Phase of EasyTV RNN

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 informations to extract: for example: rewind video 5 minutes and 30 seconds

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

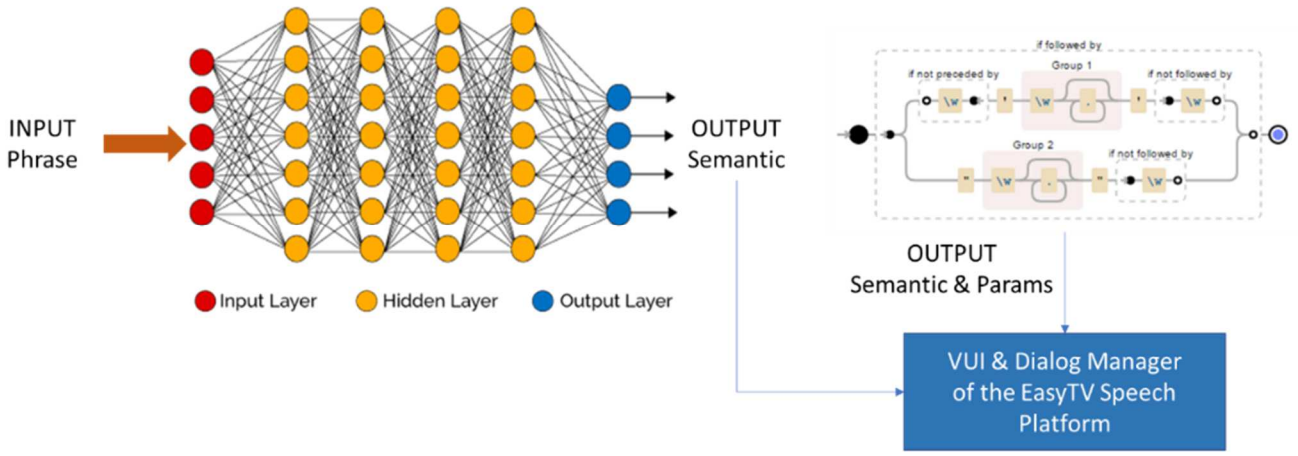


Figure 16: Interpretation Phase of EasyTV RNN

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 in 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 flow defined following the VUI guidelines and formalism described in chapter 8.

5.2.1. **EasyTV Video On Demand Catalogue**

5.2.1.1 Description

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 analysed and defined.

- Searching video.
- Browsing Video by categories.
    - Choosing a category or a sub category 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 want to go directly to a predefined VideoList (suggested video or favourite 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.

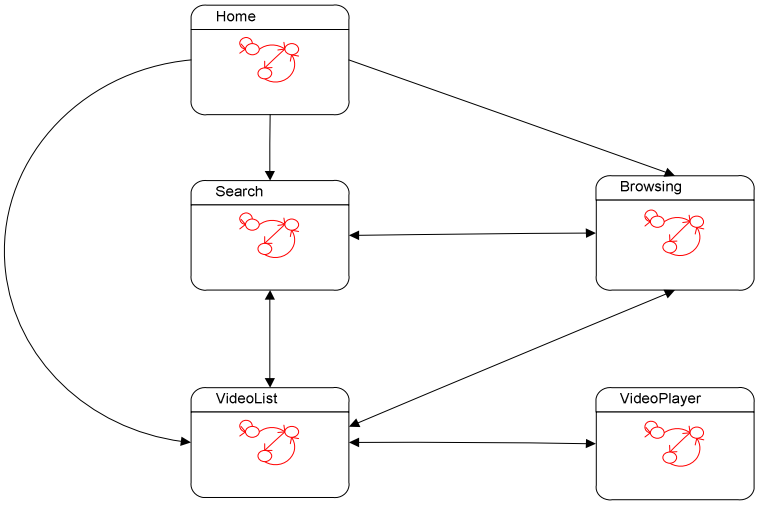


Figure 17: Video On Demand Main Dialog

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 list depending on the user voice command; that is: the list of favourites video, last accessed videos (video history) or the list of video suggested by the EasyTV platform.

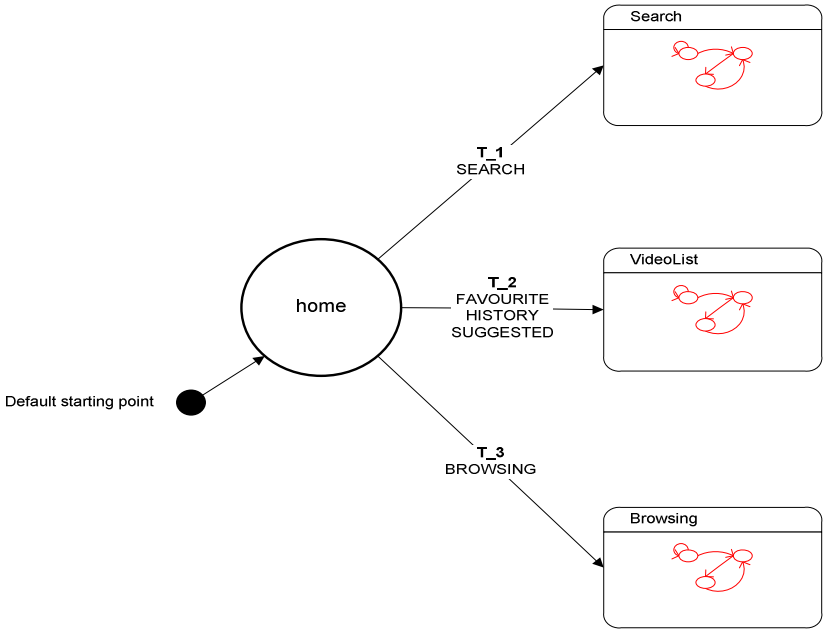


Figure 18: Video On Demand Sub Dialog Home

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 tha scope of this analysis we consider to have 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 change his mind and wants to access videolists browsing by categories.

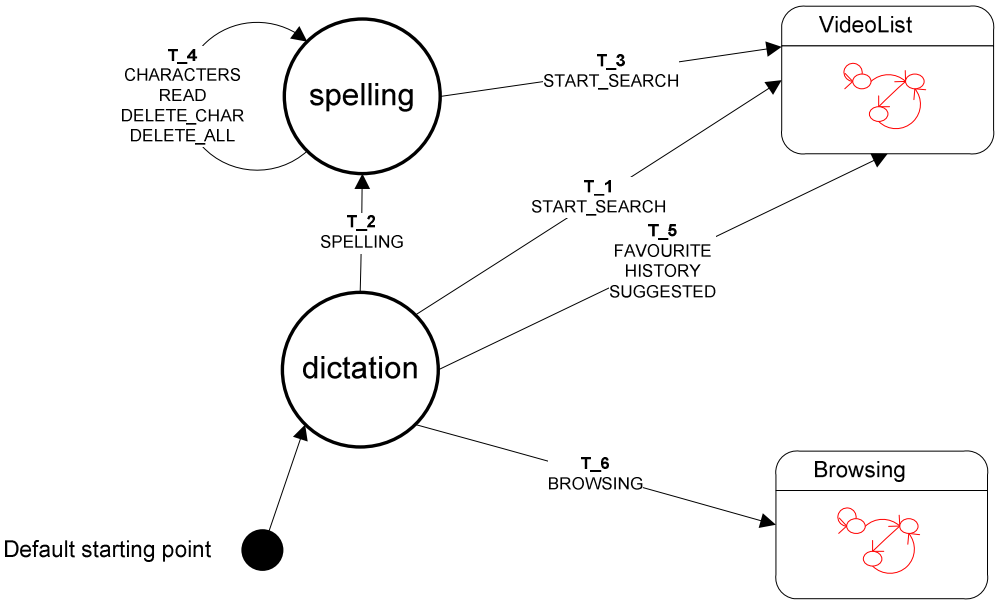


Figure 19: Video On Demand Sub Dialog Search

The Figure 20 is related to the Browsing sub dialog. In this sub dialog the user can browse videos selecting any listed category or sub category 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 videolist browsing directly the category or sub catagory of his chooice. For example the favourite videolist 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.

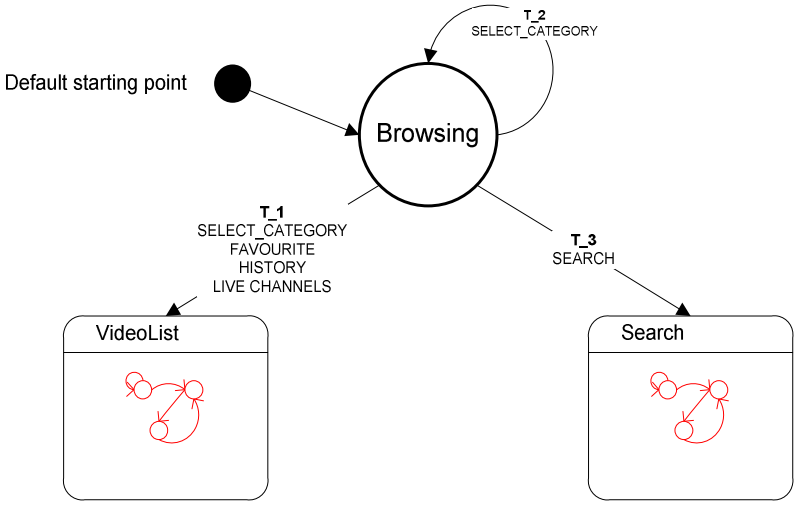


Figure 20: Video On Demand Sub Dialog Browsing

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 videolist.
- Change the videolist going directly to other predefined videolist 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 videolist browsing directly the category or sub catagory of his chooice. For example the favourite videolist 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.

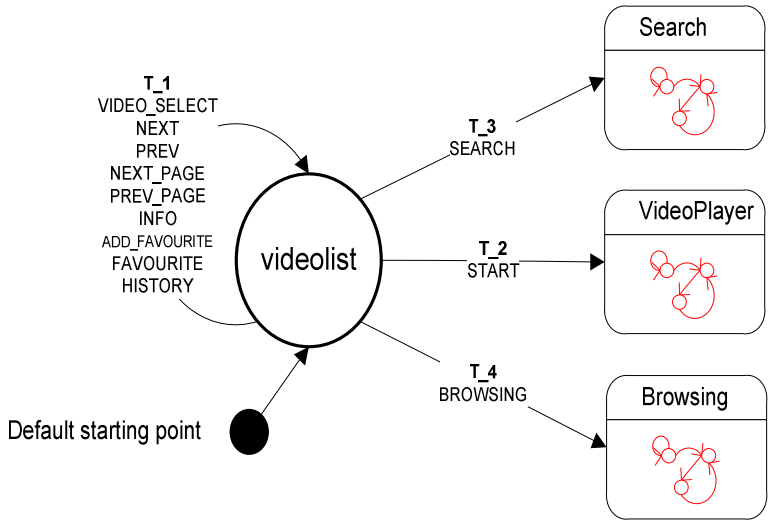


Figure 21: Video On Demand Sub Dialog VideoList

The Figure 22 is related to the Player sub dialog. This is one of the important sub dialog 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 specifing the time frame.
- Activate or deactivate fullscreen.
- Show or hide subtitles.
- Activate or deactivate secondary audio channels
- Add the video to his favourite videolist.
- 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.

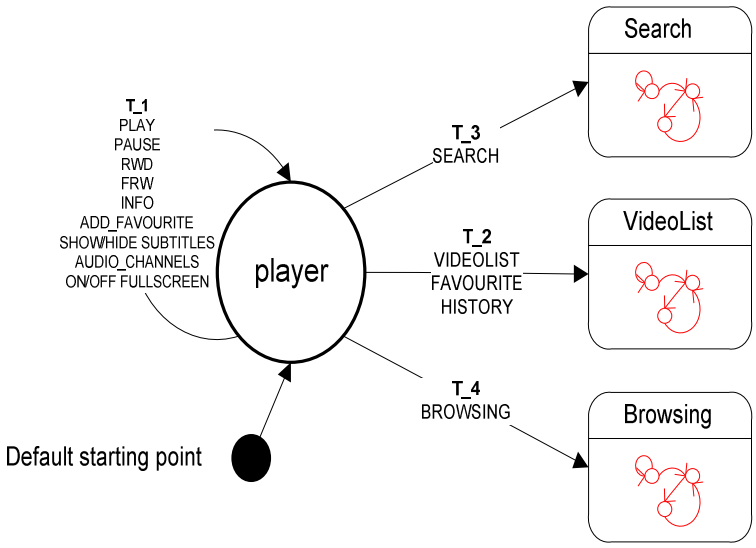


Figure 22: Video On Demand Sub Dialog Player Control

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 analize 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 inlcude 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.

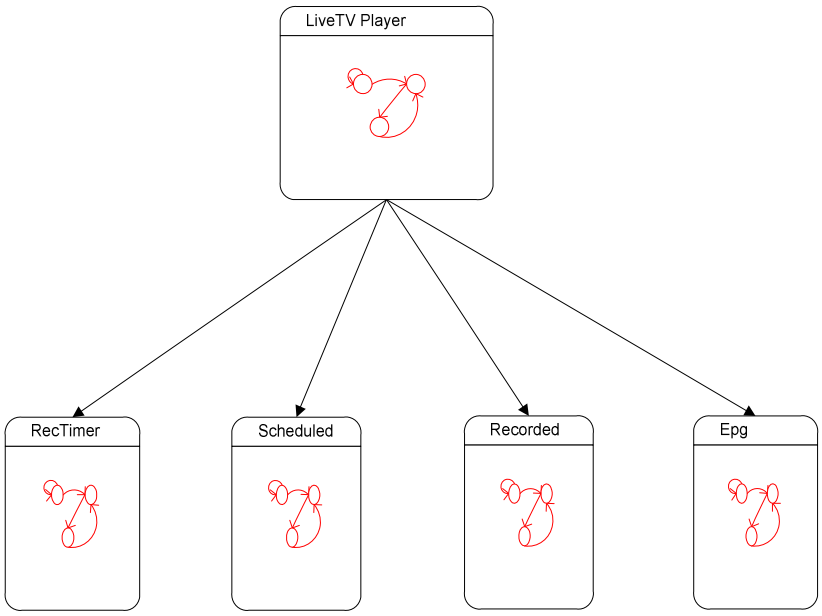


Figure 23: LiveTV Main Dialog

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.

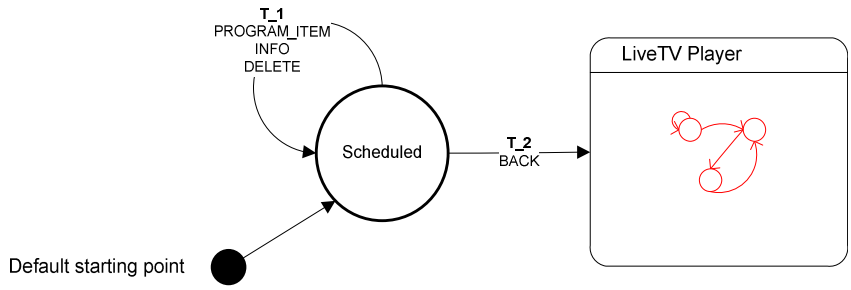


Figure 24: LiveTV Sub Dialog Scheduled Recordings

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.

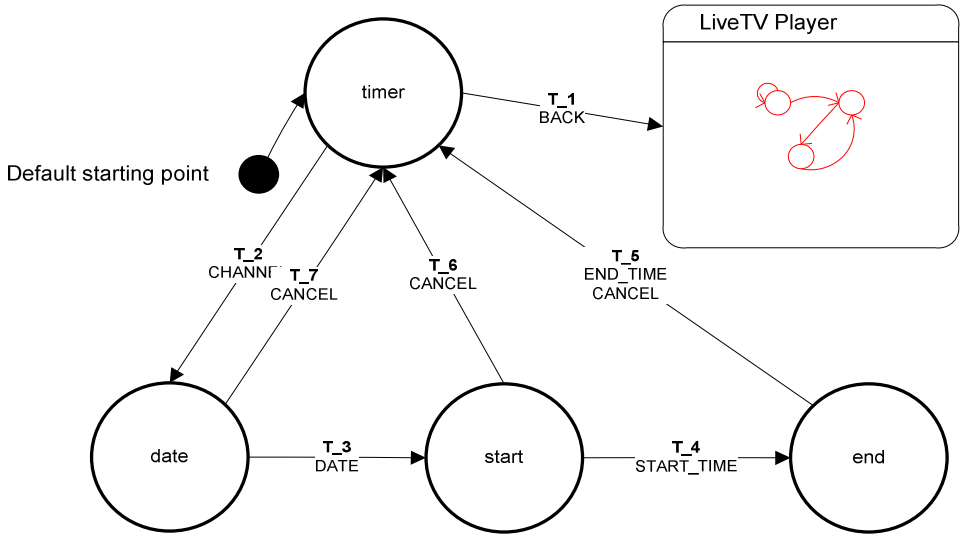


Figure 25: LiveTV Sub Dialog Recording Timers

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.

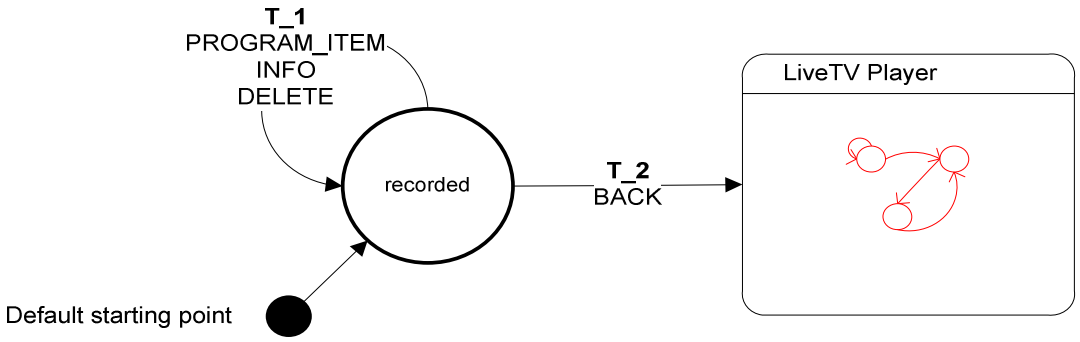


Figure 26: LiveTV Sub Dialog Recorded Programs

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.

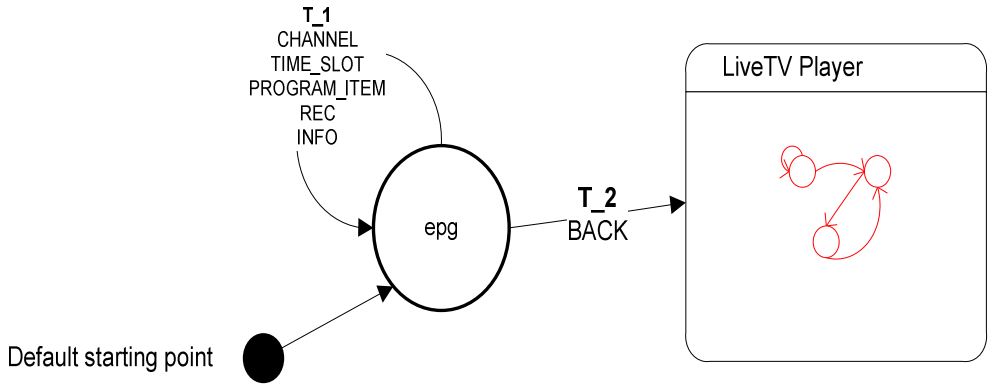


Figure 27: EPG Sub Dialog

Figure 28 is related to the LiveTV Player sub dialog. This is one of the important sub dialog 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 Sceduled or Recorded sub dialog, RecTimer and EPG sub dialogs.

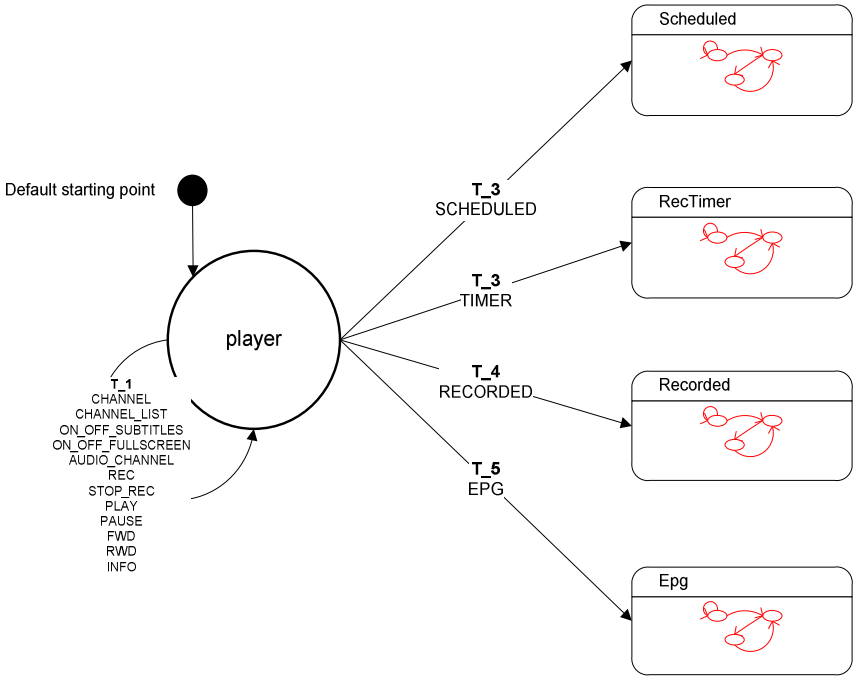


Figure 28: Player Sub Dialog

6. CONCLUSIONS

In this document we described and defined the EasyTV Speech Platform component that aims to create an effective and easy voice user interface system for blind and visually impaired users. The EasyTV Speech Platform has been defined and a first prototype also has been deveoloped. A first dialog flow module including NLP techniques based on Recurrent Neural Network for classification and interpretation of Voice Commands has been defined and developed along with speech recognition and speech synthesis modules. Finally, the first prototype has been tested using a simple Voice User Interface for YouTube Web Application.
To complete and finalize the EasyTV Speech Platform component we will carry out the next following activities:

- Implementation of the Voice User Interface for the TV App Domain based on the HBBTV applications developed by the EasyTV Partners;
- Improving Speech Platform Prototype and integrating in the software architecture of the EasyTV ecosystem applications;
- Extracting from the prototype and Developing the Speech Platform SDK for EasyTV Applications (HBBTV);
- Improving the Classification and Interpretation phase (Query Processor) as well as the Dialog Manager system.

7. 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/)

8. ADDENDUM – VOICE USER INTERFACE GUIDELINES

8.1. What is a VUI

Voice User interfaces (VUIs) allow the user to interact with a system through voice or speech commands. In this section we define the guidelines to analize and design a VUI for any kind of Application and hence for our EasyTV applications. In these guidelines we define a VUI as a set of elements that represents all the dialog between the user and the software application equipped with a vocal interface. The scheme of the VUI has to formally represent all the dialog elements as well as the flow of the dialog itself. Therefore it will have to include the texts dispensed by the system, the actions carried out by the system, the voice commands and a diagram representing the flow of the dialog between the user and the application.
In any case, the design of the interface for a speech 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).

8.2. The Elements of a VUI

The VUI design has to be expressed formally through two graphical instruments:

- The flow diagramas, providing an immediate vision of the dialog flow.
- The descriptive tables detailing the characteristics of the graphic objects in the flow diagram.

8.2.1. The dialog process

A dialog process identifies a portion of VUI that can be considered an independent logical unit. The VUI subdivision in dialog processes allows a modular engineering and comes from an attentive analysis of the application one wants to voice. Tipically, one can map the dialog processes with the macro functionalities or with the application graphical sections. A VUI can be made of a single process or of multiple ones, the identification of the right process number depends from the single application. One or more dialog states can be impemented within the dialog process to allow a structured dialog management.

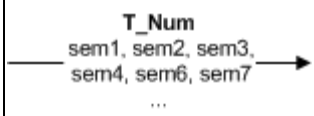
8.2.2. Dialog States

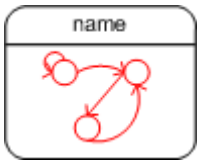
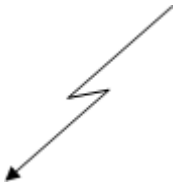
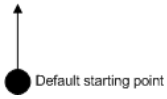
A dialog state identifies an interactive request with a distinctive sign and an active grammatical context. The number of the essential states to the management of a dialog process has to be established taking in consideration two interactive characteristics: which voice commands have to be understood and what type of help must be summarized. When the user input recognition leads to an interaction with the VUI, whose active grammars and or help dispensed have to be different from the ones in the previous interaction, it might be necessary to add a new dialog state. The effective addition of a new state can be replaced by a condition associated with the transition. The choice between a new state and the use of the condition is left to the engineer, it has to be evaluated in terms of the clarity both of the VUI and of its managing code.
The states of the dialog support the “Import” functionality. When a B state imports within its inner state an A state, in the B state all the grammars of the B state will be activated and all the rules for the semantic matchings of the A state will be inherited. The B state can also overwrite a semantics in the A state, since the parsing algorithm associates a greater priority to the rules of the state doing the import that to the one being imported. A state of dialog imported in all the VUI states of dialogs is called Universal.

8.3. The flow diagrams

In this paragraph, the guidelines for the formalization of the flow diagrams describing the VUI will be stated.

8.3.1. Legend of the graphic elements

	State of dialog: it identifies a dialog state. The lebel StatoId has to be an abbreviation describing exhaustively the implemented functionalities of the element. It does include by default the import of the universals state Tables: Identification Table, Grammar Table, Import Table. Object: Flow diagram – Other forms – Process (circle)
	State of dialog (exception): it identifies a dialog state. The lebel StatoID has to be an abbreviation describing exhaustively the implemented functionalities of the element. It DOES NOT include the import of the universals state Tables: Identification Table, Grammar Table, Import Table. Object: Flow diagram – Other forms – Process (circle)
	State transition of a process: it states the transitions between two states or within the same state, caused by the matching of a semanthics. The lebel shows the reference T Num to the transition group that the symbol represents within the Voice Transation Table, with Num being an whole value. As per the VUI designer discretion, it is possible to add the semathics generating the transition. Tables: Voice Transitions Table at a start state Object: Connectors – Curve connector 2
	State transition in the flow: it states the transitions between states or processes. It is used in the general flows diagrams. Tables: Voice Transitions Table at a start state

	Object: Connectors – Curve Connector 2
	Synthetic dialog process (Voice Template): it identifies a dialog process where one or more dialog states are involved. It is used as a logical container of an under-dialog. Each dialog process has to be detailed in a dedicated form. The "name" identifying the process has to match the xml file implementing it. It is used in the diagram of the single process to identify generically the dialog processes towards which a transition can direct the interaction starting from a process state of the current dialog. Object: specially made
	Dialog container Process: It is used in the "flow" diagram as a container of the dialog states involved in a dialog process, it has to include at least one state of dialog. It is used as a logical container of an under-dialog. Each dialog process in the flow has to be detailed in a dedicated form. The "name" identifying the process has to match the xml file implementing it. Object: flow diagram – Other forms – Divided process 2
	Event: It indicates the presence in the state towards where the arrow is pointing, of transitions caused by events not having a corresponding voice command. These can be either specific events of the current state and or general events that require a particular specialization for the specific state. Tables: Event Transition Table in the state towards where the arrow is pointing. Object: Connectors – Communication 1
	Import: it joins two states connected by an import link. The state generating the link inherits all the rules and grammars available in the state towards which the link is pointing. It has to be used only to indicate graphically specific import operation and not general ones. Tables: Import Table in the state generating the link Object: Connectors – Curve connector 2
	Default Starting Point: it points towards the initial state of the application, meaning the starting configuration of the application launch. Object: Diagram IDEF0 – Shapes – Knot+Connectors – Strait line 1
	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. Object: Diagram IDEF0 – Shapes – Knot+Connectors – Straight line 1

8.3.2. **The Flow Diagrams**

The VUI project must contain the following dialog flow diagrams:

- One with the general flow diagram reporting all the Voice Templates (dialog processes) engaged in the VUI stating the states involved in each Voice Template, the transitions between the different states, the external events regarding each state, the non-universals import relationships, the default starting point and where applicable the unique exit point. This representation may be replaced in case of extremely complex processes, with a synthetic flow diagram, where all processes are indicated in a synthetic manner. Figure 29 and Figure 30 show are examples of dialog flow diagrams.

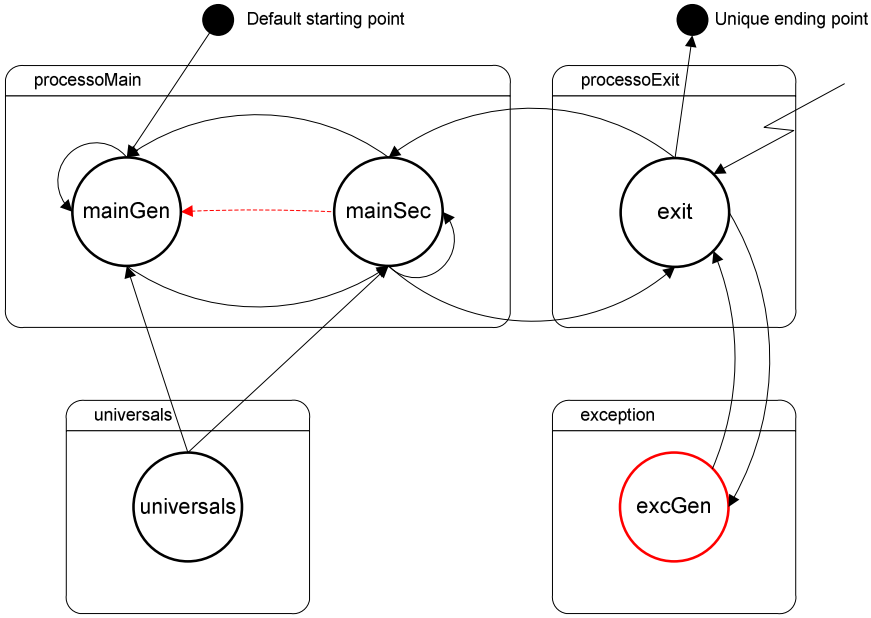


Figure 29: Example of a dialog flow diagram

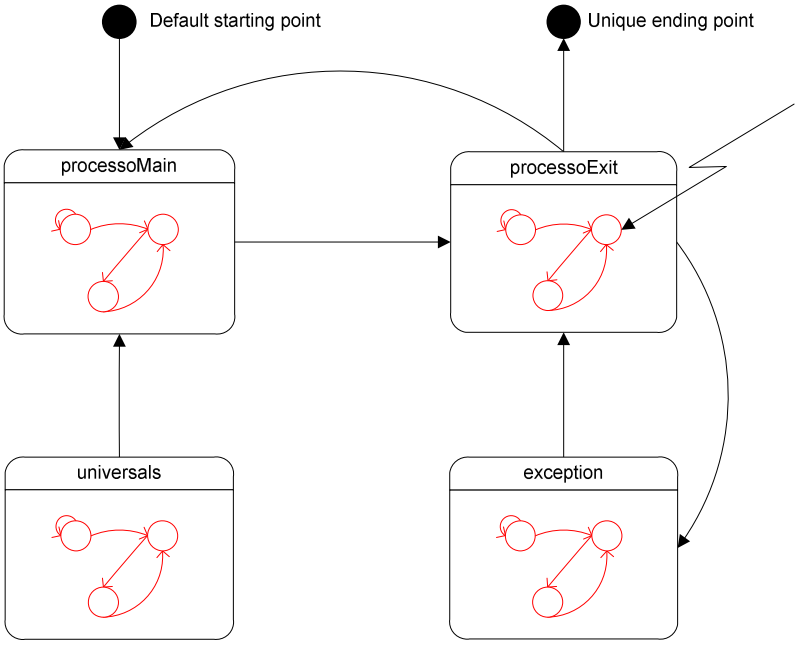


Figure 30: example of a process flow diagram

- One with the detailed flow diagram for each Voice Template involved in the general dialog flow, where the exit transitions from the different states in the described dialog process are clearly stated. Where these transitions entail a transition towards a state being part of a dialog process different from the one taken in consideration, the arrow will point generically towards the destination dialog process. The transitions will therefore be tagged; the transition's tags displayed graphically in the same dialog flow have to follow an ascending order. The Figure 31 schematises 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. The tag identifies the transitions group in the Voice Transitions Table.

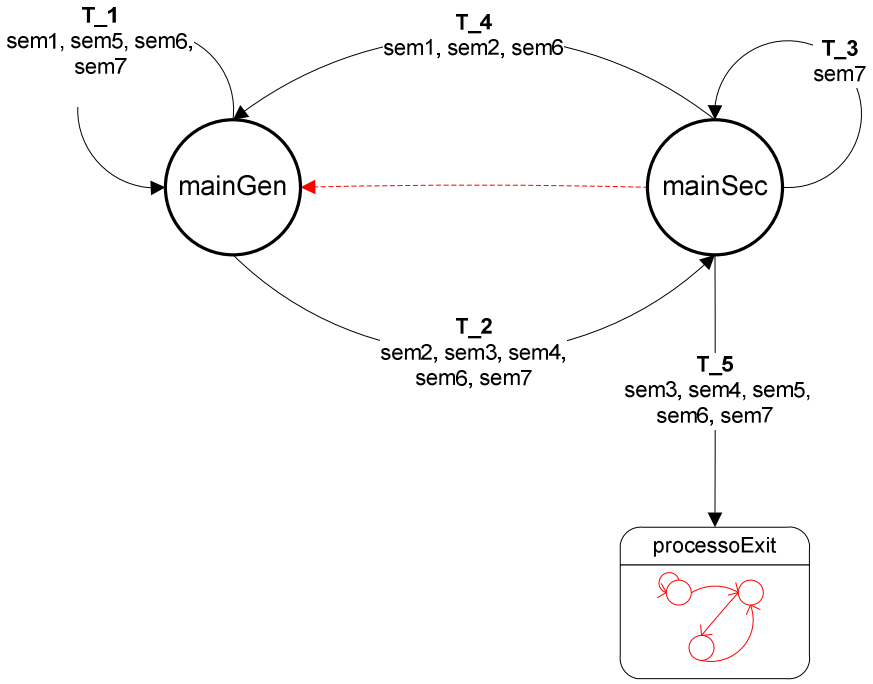


Figure 31: Dialog Flow - Main Process

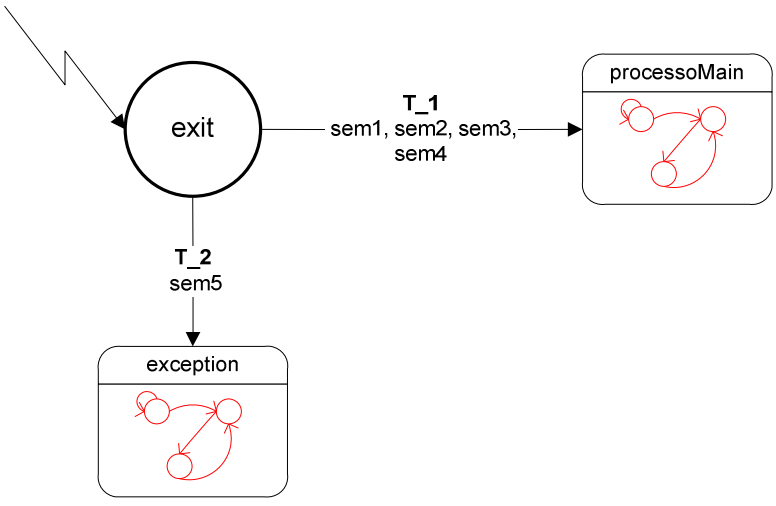


Figure 32: Dialog Flow - Exit Process

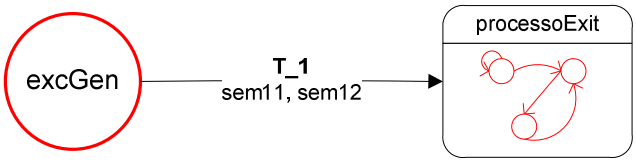


Figure 33: Dialog Flow - Exception Process

- One with a detailed dialog flow diagram for the Universals sdialog state to formalize the universally imported states. In the Universals dialog state we will indicate only the transitions caused by the universal semanthics identified for the specific application. The lebel of such transitions must be written in the following way Tuniv_Num (see Figure 34).

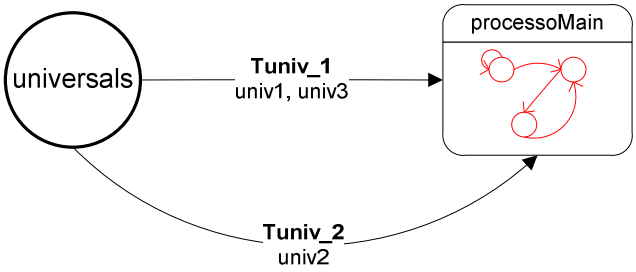


Figure 34: Dialog Flow - Universals dialog state

8.4. Tables used to define VUI elements

In this paragraph we define the guidelines to compile the tables that represents the VUI depicted in the state diagrams describet before.
The VUI elements to consider when writing the information tables are the following:

- Dialog Flow also known as Voice Template. They identify a specific portion of the dialog flow and will contain all the dialog states information of that portion. A Voice Template can be thought like a specific sub dialog that can be also imported in any other Voice Template.
- Dialog States: a dialog state identifies a specific moment of the dialog. Each dialog state identifies the grammars (phrases that the user can say in that specific momento of the dialog), and the imported states, if any, from other Voice Template defined in the VUI.
- State Transitions: they specify the behavour 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 apropriately.
- Actions: are representation of the actions to be managed by the application and related to a specific voice command and to the specific Transition that have been rised by the voice command itself.
- Voice prompt: are the voice phrases that the application will prompt to the user as a response to a voice command. They can depend also on the result of the actions executed by the application.

8.4.1. Voice Templates

The dialog flow is describet through Voice Templates files. For each dialog or sub dialog defined for the VUI there will be its corresponding Voice Template file.
For the sample reported in above in Figure 30 four Voice Template files must be defined: processoMain, processoExit, exception and universals. Each VT file will contain a table for each dialog state. The ID of each table will be the unique ID name of the dialog state defined in that dialog flow.

8.4.2. Dialog States

The dialog state represents a specific moment of the dialog flow. It identifies a context in which the user can say specific voice commands and the application is aware of it in order the corresponding acctions can be managed appropriately. The dialog state is identified with the following information:

- The STATE_ID, should be a short name that can be intuitive to identify the dialog context.
- DESC: a short description which summarize the specific voice interaction in that conext.
- The Grammar Set (if command and control type) which identify all the possible phrases that the user can say in that context.
- A list of imported dialog states from other Voice Template.

The dialog states are represented by tables inside the Voice Template file which they belong. The table of the dialog state will contain the following information:

- Description Table
- Grammar Table
- Import Table

Description Table

The description table will contain few information, the STATE_ID, the name of the Voice Template to which it belong and a short description as depicted below.

STATE_ID:	mainGen
Voice Template:	processoMain
Description:	<description>

Table 1: Dialog State Description

Grammar Table

The grammar table report the list of active grammars for a specific dialog state. Each row of the table includes the grammar name, the first role of the grammar to activate and the type of the grammar (static or dinamic). A static grammar is predefined and doesn't change during the voice user interaction. A dinamic grammar instead is a grammar that can be created run-time depending on the content of the application.

Grammar Table		
Name	**Role**	**Type**
grammar1	main	static
grammar2	second	dinamic

Table 2: Dialog State Grammar

Import Table

The Import table report the list of the dialog states, from the same VT from a second VT, that will be activated along with the importing dialog state. Each row of this table includes the STATE_ID and the VT name to which it belongs.

Import Table	
STATE_ID	**VT**
universals	universals
mainGen	processoMain

Table 3: Dialog State Import

8.4.3. Dialog State Transitions

Dialog state Transitions are related to the specific dialog state and describe the Transitions to other dialog state/context and the actions performed when executing this Transition.
Transitions are identified in the dialog flow graph with arrows named with labels like T_Num, where Num is an integer number. Each dialog state Transition is identified by a list of possible behaviour that rises that Transition. The following table is a representation of part of the Transitions for the dialog flow sample in Figure 29

Voice Transition Table								
T_ID	**SEM**	**COND**	**CMD_ID**	**TTS_ID**	**VT**	**STATE_ID**	**PARAMETER**	**NOTE**

T_1	sem1	-	cmd1	TTS_txt1	processoMain	mainGen	$$_var1_$$	-
	sem5	-	cmd5	TTS_txt5	processoMain	mainGen	$$_var1_$$	-
	sem6	A	cmd6	TTS_txt6	processoMain	mainGen	$$_var1_$$	A = download completato
	sem7	A	cmd8	TTS_txt9	processoMain	mainGen	$$_var1_$$	
T_2	sem6	B	cmd7	TTS_txt7	processoMain	mainSec	$$_var1_$$	B = download in corso
	sem2	-	cmd2	TTS_txt2	processoMain	mainSec	$$_var1_$$, $$_var2_$$	-
	sem3	-	cmd3	TTS_txt3	processoMain	mainSec		-
	sem4	-	cmd4	TTS_txt4	processoMain	mainSec	$$_var3_$$	-
	sem7	B	cmd9	TTS_txt10	processoMain	mainSec	$$_var1_$$	
Tuniv	AIUTO	-	cmd3	TTS_aiutoMainGen	processoMain	mainGen		overload
Tuniv_1	univ1	-	cmd1	TTS_txt11	processoMain	mainGen	$$_var1_$$	overload

Table 4: Dialog State Transition

The Table 4Table 4: Dialog State Transition reports the Transitions of the dialog state mainGen of the processoMain VT. Each row connects the mainGen dialog state to the correspondig dialog state, inside or outside the VT to which it belongs.
So we represent this Transitions on the table with a group of rows with the following information:
We use a group of row for each Transition because it can be rised by different voice commands and their corresponding actions and application conditions.

- T_ID: is the label of the Transition reported in the dialog flow graph.
- SEM: is the semantic interpretation of the voice command from the user input coming from the grammar matched for that user input.
- CONDITION: is a specific condition on the application under which the Transition will be executed.

- CMD_ID: Action ID that represents the action executed by the application according with the semantic interpretation.
- TTS_ID: is the identifier of the voice prompt to reply to the user after the execution of the Transition.
- VT: is the name of the Voice Template of the destination dialog state.
- STATE_ID: identifier of the next dialog state (it could be starting dialog state) from where the Transition begin.
- PARAMETER: contains the variables where to store values useful during the dialog state Transition.
- NOTE: any description or note to explain the Transition.

NOTE: Transitions named Tuniv, are not represented in the dialog flow graph because are Transitions rised by universals voice commands and are reported on the universal dialog flow graph.

8.4.4. Global Events

Global events are reported in a table named “event where all events of the application are reported. Global events usually cause a dialog state Transition and are not related to any voice command. The Table 5 is an example of an event transition table:

Event Transition Table							
EVENT	**COND**	**CMD_ID**	**TTS_ID**	**VT**	**STATE_ID**	**PARAMETER**	**NOTE**
evt_1	-	cmd_1	tts_1	processoMain	mainGen	$$_var1_$$	evt_1 = connessione interrotta
evt_5	-	cmd_5	tts_5	processoMain	mainSec	$$_var1_$$	-
evt_6	A	cmd_6	tts_6	processoMain	mainGen	$$_var1_$$	A = download completato; evt_6 = connessione interrotta
evt_6	B	cmd_6	tts_7	processoMain	mainSec	$$_var1_$$	B = download in corso; evt6 = connessione interrotta

Table 5: Event transition- global events

Each row of the table represent a specific global event that can occur in any moment and in any dialog state/context of the dialog flow. Events are managed by the speech platform and are sent back to the application when needed. Gobal Events can also be overloaded for a specific dialog state when needed and it must be reported in the “Event Transition Table” of the specific dialog state. Moreover each event can rise different Transitions and lead to different dialog state, so every Transition must be reported in the table specifying the different conditions and information.

Event rows in the table contains the following information:

- EVENT: name of the event. Is a good rule to use mnemonic names for example “connectionDown” or an acronim to explain on the column “note”.
- CONDITION: is a specific condition on the application under which the event Transition will be executed.
- CMD_ID: Action ID that represents the action executed by the application according with the semantic interpretation.
- TTS_ID: is the identifier of the voice prompt to reply to the user after the execution of the Transition.
- VT: is the name of the Voice Template of the destination dialog state.
- STATE_ID: identifier of the next dialog state (it could be starting dialog state) from where the Transition begin.
- PARAMETER: contains the variables where to store values useful during the dialog state Transition.
- NOTE: any description or note to explain the Transition.

8.4.5. Context Events

Context events are reported in a table where all events related to a specific dialog state are reported. In the same way as global events, context events cause a dialog state Transition and are not related to any voice command. Moreover context events are managed only when the dialog flow is in the dialog state where the event is reported.
We define the vent table as follow:
Context events are represente in the dialog flow graph with a lightning symbol touching the dialog state symbol to which it belongs. The Table 6 is an example of an event transition table:

Event Transition Table							
EVENT	**CONDITION**	**CMD_ID**	**TTS_ID**	**VT**	**STATE_ID**	**PARAMETER**	**NOTE**
evt_1	-	cmd_2	tts_2	processoMain	mainGen	$$_var1_$$, $$_var2_$$	evt_1 = connection broken, overload
evt_{11}	-	cmd_5	tts_1	processoMain	mainSec	$$_var1_$$	evt11 = end of speech

Table 6: Event transition- context events

Event rows in the table contains the following information:

- EVENT: name of the event. Is a good rule to use mnemonic names for example “connectionDown” or an acronim to explain on the column “note”.
- CONDITION: is a specific condition on the application under which the event Transition will be executed.

- CMD_ID: Action ID that represents the action executed by the application according with the semantic interpretation.
- TTS_ID: is the identifier of the voice prompt to reply to the user after the execution of the Transition.
- VT: is the name of the Voice Template of the destination dialog state.
- STATE_ID: identifier of the next dialog state (it could be starting dialog state) from where the Transition begin.
- PARAMETER: contains the variables where to store values useful during the dialog state Transition.
- NOTE: any description or note to explain the Transition.

8.4.6. Actions

Actions are executed by the application following the associated voice command given by the user. All actions related to the application are reported in a unique and specific file named “cmd”. The Table 7 shows the information of the actions:

CMD Table			
CMD_ID:	**Type**	**Var**	**Value**
cmd_id	VARS	var_name	valore
	APP	-	text $$var_name$$

Table 7: VUI Actions

Actions are listed by named strings containing coded names that represent the action to be executed by the application. The VUI designer and the Application designer will list the actions and their coded string as needed by the application and VUI. Each string can also contain variable information that are managed by the speech application so each action can hence be just filling a variable with some value and then use this variable in a coded string for a secon action.
Every action is defined using few information

- CMD_ID is the ID of the action.
- Type is the type of the action and can be VARS to assign a value to a specific varable. APP indicate that the coded string is sended to the application for its interpretation and execution.
- Var is the var name to fill with the value on the Value field.
- Value: is the value to fill in the variable if the Type field is VARS otherwise will contain the coded string of the action to send to the application. The coded string can refer also a variable using the naming reported in the table. In that case the the portion af the coded string will be replaced with the value of the referenced variable.

8.4.7. Voice Prompts

Voice prompts are the voice phrases that the application will prompt to the user as a response to a voice command. They can depend also on the result of the actions executed by the application.
Voice prompts are reported in a unique and specific file named “tts”. The Table 8 shows the information of the voice prompts with some tipical samples:

TTS Table					
TTS_ID	**Type**	**Content**	**Parameter**	**Parameter value**	**Note**
TTS_txt1	text	Bla bla bla @@var1@@ bla bla bla @@var2@@	@@var1@@	Value var1	
			@@var2@@	Value var2	
TTS_txt2	text	Bla bla bla			
TTS_txt3	file	Path/URI			
TTS_txt4	text	Static text @@var1@@ static text @@var2@@	@@var1@@	insieme di valori	
			@@var2@@	insieme di valori	

Table 8: VUI - Voice Prompts

Every voice prompt in the table is identified by a unique id named tts_id and will contain the following information:

- Type: is the tipe of the voice prompt that can be a static text or a dinamic text or mixed static nd dinamic. Its value will be “text” or “file”
- Content: is the text to be prompted to the user or the path/URI of the file containing the text. The voice prompt content can also be compiled at run-time using variables value as reported in the table. So it can be all static all dinamic or a mixed tet filled with static parts and dinamic parts.
- Parameter and Parameter Value are the variable name to be included in the voice prompt and its value whenever the voice prompt will be dinamic.
- Note: is the description of the voice prompt if needed.

8.5. 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 are included in a specific universals grammar or managed processing the utterance of the dictation grammar. When designing the VUI all universals command should be identified and reported to the specific grammar.

- A list of the standard Voice Commands that usually should be considered in any VUI are the following:
  - REPEAT: usually this universals command promts 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 canceling 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.

Dialog state Transitions caused by universals voice commands are represented with Transitions label named Tuniv_NUM and included in the voice transition table (Table 9) of the universals dialog state.

Voice Transition Table								
T_ID	**SEM**	**COND**	**CMD_ID**	**TTS_ID**	**VT**	**STATE_ID**	**PARAMETER**	**NOTE**
Tuniv	RIPETI			-	-	-		last_tts, same_vt, same_state
	AIUTO			TTS_aiuto	-	-		same_vt, same_state
	INDIETRO			???	???	???		
	ESCI			TTS_esci	-	-		same_vt, same_state
	MENU			TTS_menu	universals	universals		
	WHEREAMI			TTS_dovesono	-	-		same_vt, same_state
Tuniv_1	univ1		cmd1	TTS_univ1	processoMain	mainGen	$$_var1_$$	
	univ3		cmd3	TTS_univ3	processoMain	mainGen		
Tuniv_2	univ2		cmd2	TTS_univ2	processoMain	mainSec	$$_var1_$$, $$_var2_$$	

Table 9: Voice transition- Universals

9. ADDENDUM - SPEECH TECHNOLOGY

9.1. 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.

9.1.1. 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.

9.1.1.1 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 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.

9.1.1.2 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.).

9.1.1.3 The recognition process

A tipical speech recognition system has a structure as depicted in the Figure 35:

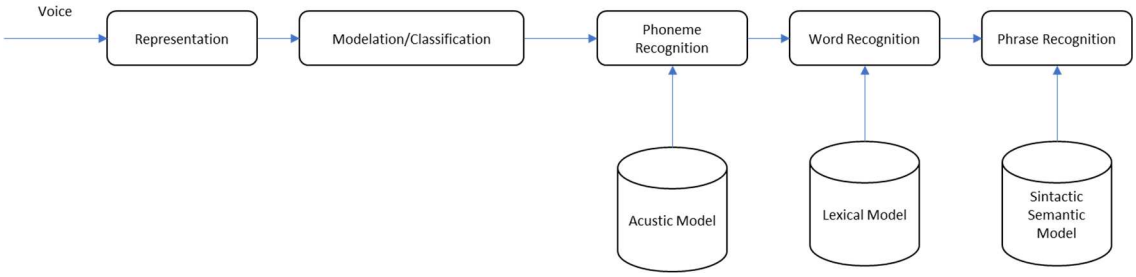


Figure 35: structure of a speech recognition system

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 "modeling / 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.

9.1.1.4 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 move 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.

9.1.1.5 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.

9.1.2. **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, encyclopedias, 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.

9.1.2.1 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). Its limit is the lack of flexibility: the synthetic voice is that of the donor speaker, and is poorly modifiable.

9.1.2.2 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.

9.1.3. **Distributed Speech Recognition (DSR)**

9.1.3.1 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.

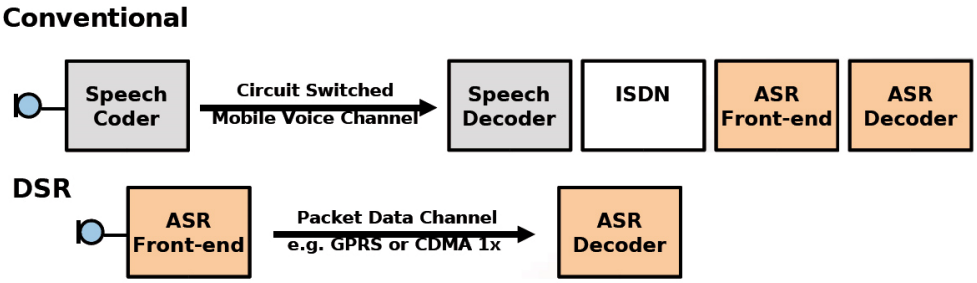


Figure 36: Schema of Distributed and traditional ASR Systems

9.1.3.2 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 37 and Figure 38

In addition to the audio packages, in the first diagram (Figure 37) additional information to optimize the performance of speech recognition is also sent.

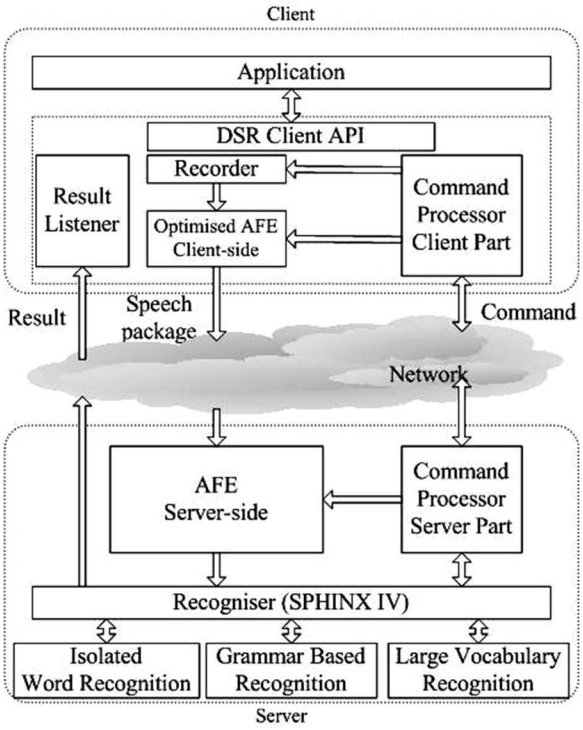


Figure 37: Architecture 1 of a DRS system

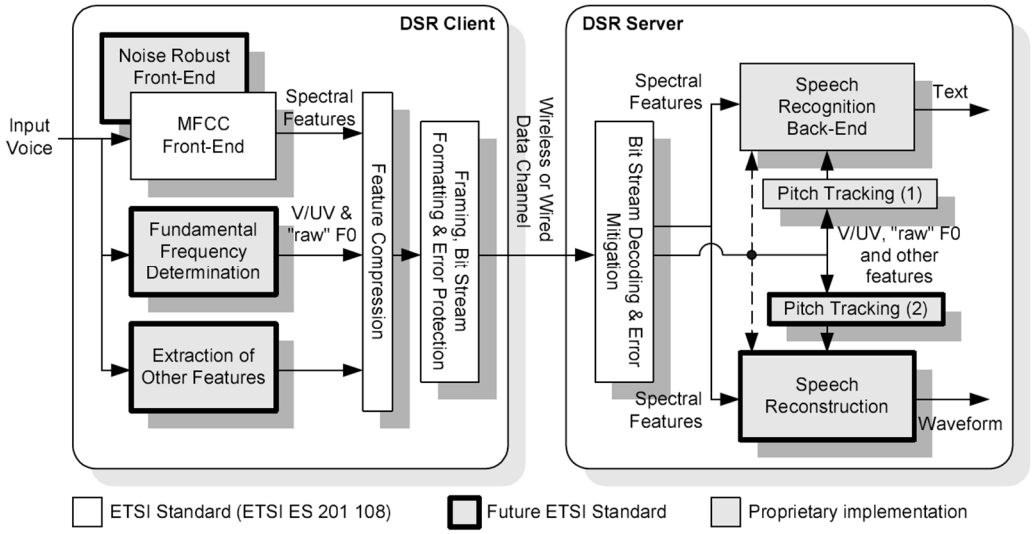


Figure 38: Architecture 2 of a DRS system

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.

9.1.4. **Dialogue and VUI**

9.1.4.1 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.

9.1.4.2 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 amount 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>
```

9.1.4.2.1 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.

9.1.4.2.2 VoiceXML architecture

The VoiceXML architecture is structured as detailed below:

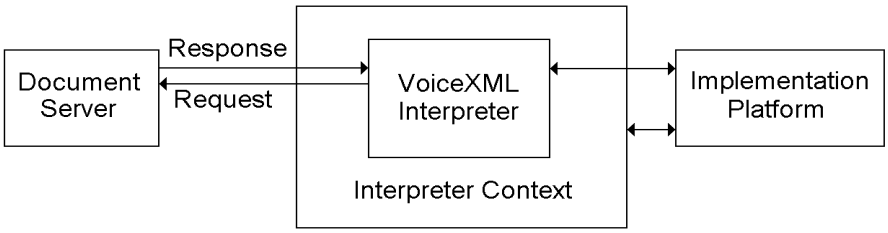


Figure 39: VXML Architecture

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.

9.1.4.2.3 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.

9.1.4.2.4 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.

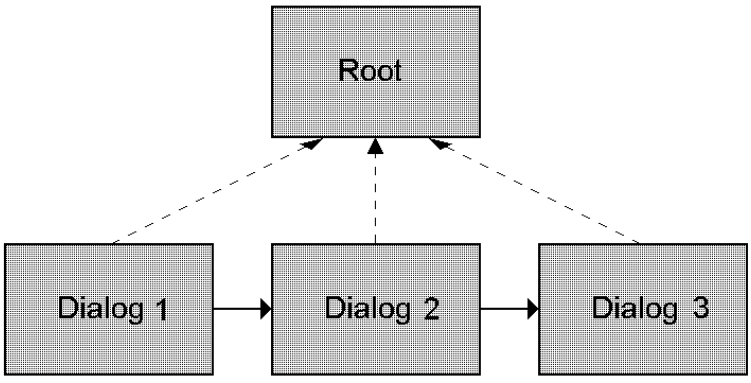


Figure 40: Structure of a VoiceXML application

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 behavior 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.