European Clearing House
for Open Robotics Development
www.echord.info

ECHORD

Technical Report D3.2:

Robotic Voice Development Kit Release

E. Bastianelli*, F. Giacomelli**, N. M. Manes**, D. Nardi*, V. Perera*

Proposal full title: **SPEAKY for Robots**

Proposal Acronym: **S4R**

Name of the Coordinating Person: **Daniele Nardi**

Institution of the Coordinating Person (*): **Sapienza Università di Roma, Dipartimento di Ingegneria Informatica, Automatica e Gestionale**

Other participating institution (**): **Mediavoice S.r.l.**

Preface

SPEAKY for Robots (S4R) aims at fostering the definition and deployment of voice user interfaces (VUIs) in Robotic applications where human-Robot interaction is required. More in depth, S4R promotes speech technologies transfer towards manufacturing processes, to provide semi-automatic speech-based interface development for Robotic platforms. This in turn will boost up the Robot presence in manifold activities, by supporting a natural interaction with humans.

S4R specific goal is a novel **Robotic Voice Development Kit** (RVDK), namely a framework that supports Robotic developers in designing voice user interfaces with little effort. RVDK is conceived as an interactive environment aiding designers to define the voice interface according to the desired application requirements; hence, it is adaptable to different application fields. In order to design and implement the RVDK, state of the art solutions about lexical vocabularies and knowledge representation to capture the semantics of the domain, and natural language processing technologies will be integrated to build the input for the speech processing system SPEAKY, that is currently commercialized by Mediavoice partner of the consortium.

S4R experiment targets two possible application domains for RVDK. The first scenario deals with **home services**, where a user controls a humanoid Robot through a voice user interface within a domestic environment. The second scenario consists of an **outdoor Robotic surveillance**, where the user is controlling the action of a wheeled Robot capable of navigating in rough terrain**.** The goal of the experiments is to assess the performance of the voice interface implemented though RVDK. This project is thus a **joint enabling technology development** effort, whose scope falls within the **human-Robot co-worker** scenario, addressing the **human-Robot interfacing and safety** research focus.

In this report, we describe the Robotic Voice Development Kit, which is the prototype for the design and implementation of customized voica user interfaces, that is one of the key results of the Speaky for Robots Experiment. The development of the prototype is part of the activities of Task 3.2.

1 Introduction

The Robotic Voice Development Kit is a software tool whose aim is to support the development of Vocal User Interfaces, that are specialized for a given robotic platform a given operational environment and the tasks that the robot is able to accomplish. This prototype is the industrial target of our project, since it is been conceived with the goal of supporting MediaVoice in the development of specialized Vocal User Interfaces for robots.

The prototype Robotic Voice Development Kit has been devised after several experiments of Vocal User Interfaces that address specific instances of robots and operational environments. In particular, we have addressed the design of Vocal User Interface for the NAO Humanoid Robot, in principle to be used in a home environment [D2.1]. Given the richness in terms of mobility actions and other types of motion and feedback for the user available, this prototype explore a rich command language. However, given the limited sensing capabilities of the robot, most of the commands implemented do not consider the perception of the environment.

Later, we shifted our attention to a small Videre wheeled robot, operating in a home environment. To this end, we built a semantic map of a home environment and made it available on the robot. While the semantic mapping is not part of the design addressed by the Robotic Voice Development Kit, it is worth emphasizing that it is a requirement, when the system is expected to perform tasks that require the perception and understanding of the operational environment.

At present, a third instantiation of the Vocal User Interface is under development, targeting again the Videre wheeled robot, deployed for a surveillance application in an indoor office environment.

As a result of the design of the above mentioned Vocal User Interfaces and system prototypes, we have been able to identify a run-time environment that is common to all the Vocal User Interfaces and a number of specific inputs for creating customized instances of the interface. In particular, the customization takes into account the type of platform and the operational environment. Given these information as input, the customization process amounts to define a specification of the grammars used by the ASR, a specification of the knowledge base of the commands and a specialization of the dialog states, as preliminarily described in [D2.1]. The overall specification process is implemented through a wizard, following a design approach well established in Mediavoice.

The structure of this document is thus organized in two sections: the next one describes the architecture, the components and the information flow of the run-time environment; the subsequent one describes the wizard that allows for the creation of specific instances of the interface.

2 Run Time Environment

In this section we describe the runtime environment of the Voice User Interface developed within the Speaky for Robots framework: the different modules that implement the architecture, together with the protocol implemented for the communication among them and the overall information flow.

2.1 Architecture

The runtime environment of VUI is shown in Fig. 2.1. The figure shows the system components, the communication flow among them and their input specification.

More specifically, the “Broker” component is in charge of handling the communication with the robot; since the module does not deal with the content of messages, it is the same for every specialized VUI and it will not be described in more detail. For a more detailed specification of the communication with the robot, see [SPEAKY SDK PG, 2009] and section

The “Speaky Platform”, basically includes all the features of Speaky Development Tool[SPEAKY SDK PG, 2009] plus a number of additional functionalities described in detail in the next subsection. The input specification of the Speaky Platform component is given by one or more grammars to be used by the ASR and the description of the different states to be used by the Dialog Manager, including specialized grammars to be loaded dynamically and the sentence(s) to be uttered to the user in each state.

The “Semantic Analyzer” component takes care of the selection and interpretation of the results of the ASR. The input specification of the “Semantic Analyzer” is given by a knowledge base of frames, that characterize the meaning of the commands executable by the robot. Additional details on the semantic interpretation are given in subsection 2.3.

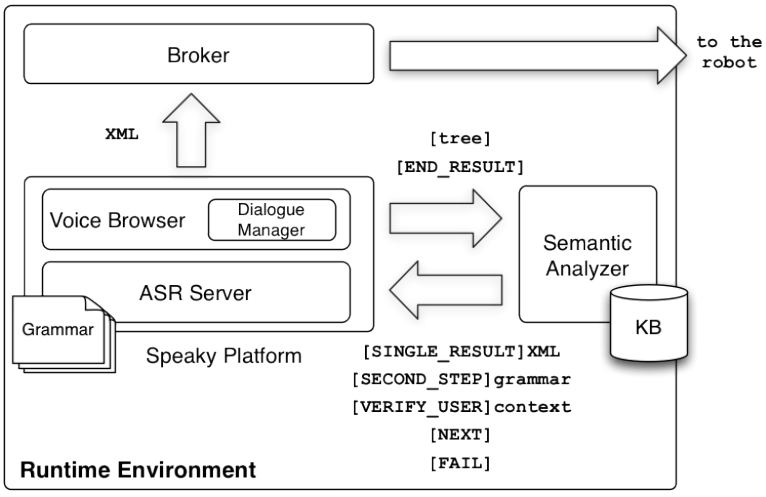


Fig. 2.1 S4R VUI runtime architecture

The communication among the modules is implemented using TCP/IP sockets, asynchronous message exchange and simple message formats. These choices allow for an efficient use of the computational resources. The components are implemented using the basic technology adopted for Speaky, namely C# programming language and the .NET development framework.

2.2 Speaky Platform

The Speaky Platform is the main component of the runtime of the S4R VUI and includes several subcomponents.

Voice Browser

The Voice Browser is the central module of the platform as it manages the interaction and the communications between the other modules. More specifically, it takes the string representing the result of the ASR Server and passes it to the Semantic Analyzer; it forwards the result of the semantic analysis, that is an XML file containing the semantic interpretation of the command, to the Broker, which then sends the command to the robot; finally, it handles the Dialog Manager and the interaction with the Semantic Analyzer, that is better described in the sequel.

ASR Server

The automatic speech recognition server is the core of the recognition process. It takes the sound wave of the utterance captured through the microphone and outputs the result of the recognition. The recognition process is mainly driven by the SR grammars produced and selected by the user during the wizard process. The result of the recognition process is a string that represents a parse tree with different kinds of information about the syntax and the semantics of the utterance, and the different confidence values of the words that form the sentence. The output of the ASR server is sent to the Voice Browser (VB) that, in turn, sends it to the Semantic Analyzer.

Here is an example of the string representing the tree yielded by the recognition process.

```
[[s,[motion,[impVP,[[target,[verb,[go,0.7384719]]],[goal,[pp,[[prep,[to,0.001149209]],[np,[[art,[the,0.6600994]],[noun,[kitchen,0.9073448]]]]]]]]],0.7553115]
```

In addition, the ASR server can request the recognition of the user sentence to the Microsoft built in Speech Recognition (Dictation Engine) included in Windows 7 operating system and send also this result to the Semantic Analyzer. We have not yet exploited this capability, that has been included to deal with sentences that are outside the grammar.

Dialog Manager

The Dialog Manager controls the interaction with the user by changing the state and the behaviour of the system, according to the response of the semantic analysis process, which provides an interpretation to the utterance of the user. More specifically, every state of the dialog is characterized by a set of grammars to be loaded when a transition into that state is triggered, and by a set of utterances that the system will verbalize to the user in correspondence of that same transition. This information is encoded in some XML files that the system takes as input. The Dialog Manager module is implemented as a part of the Voice Browser.

In section 2.4 a complete analysis of the communication between the Semantic Analyzer and the Dialogue Manager is provided.

2.3 Semantic Analyzer

The Semantic Analyzer selects the best interpretation of the user utterance, given the output of the ARS, that is it realizes the semantic analysis. Thus, the parse tree produced by the ASR Server is taken as input and the semantics is extracted from it, by matching it with a frame and its frame elements. We recall that the result of the ASR contains three kinds of information: semantic information, syntactic information and the confidence value of the entire sentence together with the confidence values of the individual words. The first kind of information is used to instantiate the frame structure, starting from a description of it contained in a knowledge base. This KB contains information about the structure of all the frames corresponding to the commands executable by the robot (and selected during the wizard process see section 3.2.2. These “frame descriptions”

contain also information about constraints on the frame elements that must be satisfied in order to determine an executable command from the sentence (e.g. the GOAL frame element for a motion command. The KB is implemented as a set of XML files written in the OWL formalism [W3C, OWL-REF 2004], one for each frame, and is organized in a hierarchical structure reflecting the frame hierarchy. This hierarchy reflects the fact that some frames have a general meaning and cannot correctly specify the action to be executed by the robot; the right action to perform could depend on the verb and other particles used in the given command, enriching the frame meaning. Thus, starting from the general frame and the verb used, we can instantiate a frame that is a child of the general one, but that can be mapped in a clear robot action. For example, the *Change_operational_state* frame has two children, the one corresponding to the "turn on" action, *Change_operational_state_on*, and the one corresponding to the "turn off" action, *Change_operational_state_off*. As you can see, the correct instantiation may depend on the verb used (it could be "start", "deactivate", and so on) plus, as in this case, the adverb (e.g. "on", "off"). Using a frame hierarchy, one can infer the right action for the robot. Once the semantics has been extracted and the appropriate frame structure has been instantiated, the Semantic Analyzer must compare different interpretations that can be associated to the alternative results computed by the ASR. To this end, it first checks the consistency of the frame instance against the constraints described in the KB. Moreover, it uses the confidence values to validate the trustworthiness of the sentence, or of a piece of it (i.e. of a frame element). These values are also used together with the syntactic information to choose the best interpretation when more than one speech result is available.

Additional consistency checks on the semantics of the requested action could be performed by the Semantic Analyzer, by exploiting domain knowledge. For example, in the case of a command for transporting objects the feasibility of the operation can be verified. At this stage, we have not developed this aspect.

The result of the Semantic Analyzer is sent to the Voice Browser, that manages the interaction with the user and with the robot. More specifically, the outcome of the Semantic Analyzer is used to determine the next state of the dialog. From a more operational viewpoint, when a frame is for example correctly instantiated, the corresponding command is serialized into an XML message that is sent to the Voice Browser; the receipt of this message will enable the transition to a state where the command is executed and the user is informed accordingly. A detailed description of the management of the interaction among the components of the interface is provided in the next subsection.

2.4 Interaction between the Speaky Dialog Manager and the Semantic Analyzer

As previously mentioned, the interaction with the user is handled by the Dialog Manager, which keeps track of the interaction state and implements the corresponding actions, towards the user and the robot. The state transitions are determined by the results of the semantic analysis, as well as by inputs provided by the other components. Below we characterize the generic states that are defined for the Dialog Manager.

Initial_state: the initial state of the system. When the system is in this state, it awaits for the user to utter a command, which is interpreted using the grammar defined for this state.

Command_execution: corresponds to a correctly recognized and analysed command. The result of the semantic analysis is sent to the robot via the Broker. When the interface is in this state, the command is executed by the robot.

Verify_user: corresponds to situations where an interaction with the user needed in order to complete the voice command; this may happen either because a user utterance is incomplete or because some part of the voice command is incorrect (e.g. does not make sense or is not applicable). In this case the Speaky Platform will load the appropriate grammars for this context and prompt the user to provide the missing or incorrect information. Several instances of this kind of states corresponding to different types of requests can be defined in the voice template files (see section 2.2 – Dialog Manager).

Second_step: corresponds to situations where the Semantic Analyser fails in the first attempt to provide an interpretation of the user utterance; in this case the system may try an alternative processing of the user input

(either through a different grammar or through the use of another ASR such as the Dictation Engine); in this state there is no interaction with the user.

Fail: this state is triggered when the system completely fails in determining an executable command for the user utterance. In this case the system will inform the user of the failure and go to the initial state.

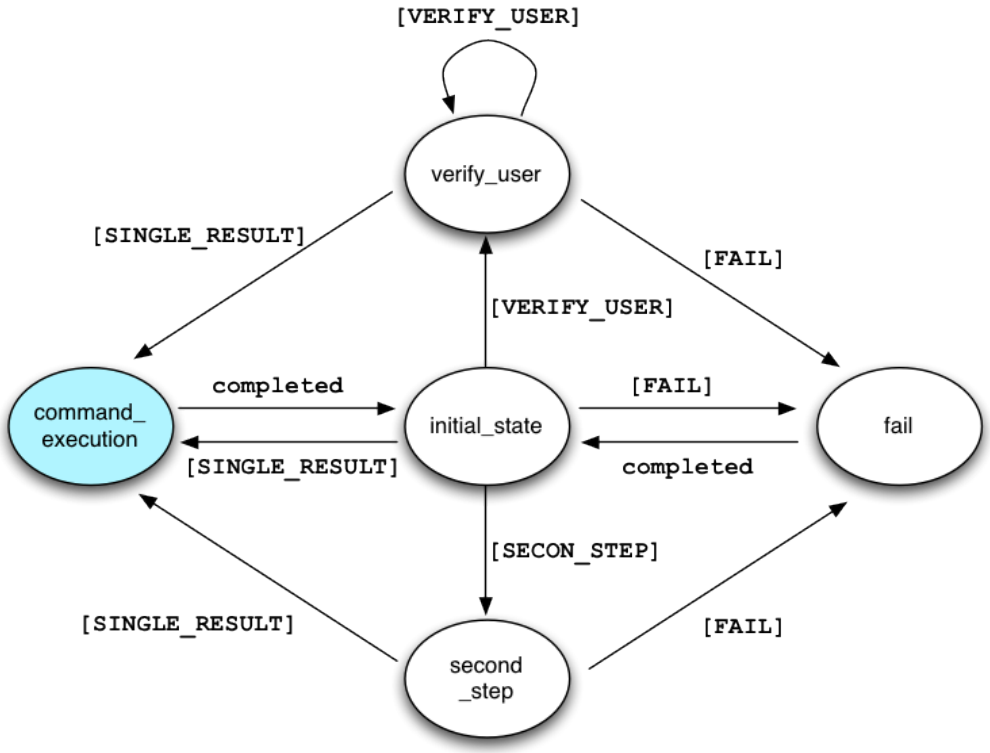


Fig. 2.2 S4R VUI Dialog Manager diagram

The evolution of the Dialog Manager state machine is driven by the messages generated by the Semantic Analyzer. The figure above represents the Dialog Manager with the different states previously described and the transitions associated with the messages of the Semantic Analyzer triggering them (the ones between square brackets). For a brief description of these messages, see Appendix B. The “completed” label denotes an internal signal of the Speaky Platform that resets the system at the end of the recognition process (with good or bad outcome), putting the platform in the initial state, ready to accept new commands. Moreover, in this picture the “verify_user” state can be viewed also as a set of user defined states (i.e. each instance of this state represents a particular step in the whole interaction process).

In the next section we focus on the implementation of the wizard process depicting the various steps involved in the process itself.

3 Wizard Development

In this section, we describe the implementation of the tool to support the definition of a dedicated vocal user interface for a given robotic platform and a given operational environment. The goal of this process is to specialize the input for the S4R runtime environment presented in the previous section, namely the grammar specification for the ASR and the frame knowledge base used by the Semantic Analyzer. Through the wizard, the operator can define a specific voice interaction for the application at hand, thus decoupling between the Speaky voice enabled platform from the Robot platform.

3.1 Repository Architecture

Before addressing the process, we start by presenting the organization of the data structures that are used by the S4R runtime environment. The data used by the wizard is organized like a directory tree, structured according to the following hierarchy: Robot Platform (e.g. NAO Humanoid, Videre wheeled) is specified at the first level; then, at the second level of the tree, we place different Application Domains is along with the Knowledge Base of the Robot platform and finally, under each application domain the Knowledge Base of the Robot platform and the actual data of each instance of the Vocal User Interface are placed. Since some VUI elements or Application Domain Elements appear in different places in the hierarchy, we keep a main copy and create appropriate shortcuts to the original data.

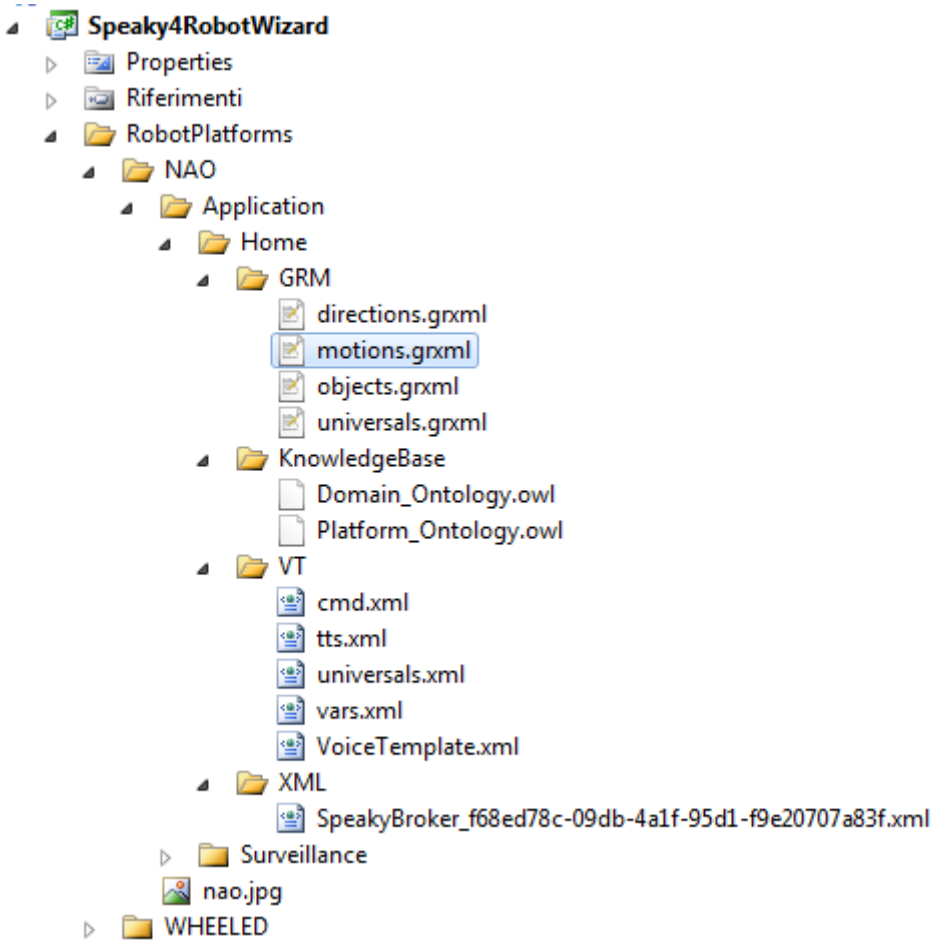


Fig. 3.1 The tree structure for storing the data used by S4R

The data corresponding to each Voice User Interface are arranged into three subfolders, the GRM folder where with all the grammars for the Speech recognition engine, the VT folder and the XML folder, whose contents are described below. All the files and components of the Voice User Interface are defined using the Speaky Platform and the Speaky SDK Programming Guide [SPEAKY SDK PG, 2009].

The VT folder contains instead the following files:

- Tts.xml, where we define all the prompts, feedbacks for the speech synthesis engine used by the platform to interact with the user;
- Cmd.xml, which contains all the action strings to be parsed by the Speaky Broker, before sending them to the Robot;
- Vars.xml, which contains string variables in a key:value style that can be used in voice template files in the cmd.xml file and in tts.xml file to build text to speech prompts at run time, to fill values in action strings or to calculate grammar names in voice template contexts.

All the other files included in the VT folder are Voice Templates, containing the definition of the dialog flow for the specific application.

The xml file under the XML folder is used by the Speaky Platform to start the Speaky Broker that is the interface component between Speaky Platform and the Robot platform. In this file we also define the voice commands used by the end user to start the Speaky Broker bye voice.

All the files described above are defined for the specific domain where the Robot will be used for and can be customized by the operator appropriately during the wizard process.

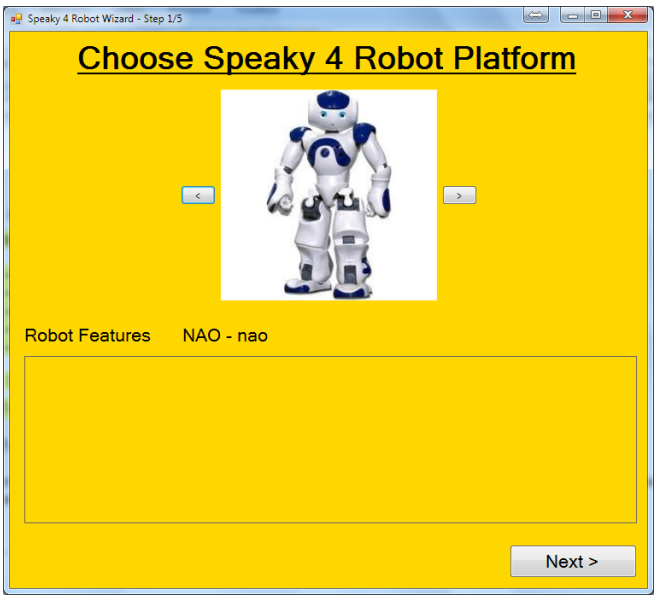
The system uses a knowledge base to drive the building process of the grammars. This knowledge base contains hierarchical structures of concepts about the robot capabilities and the environment of the application domain. The knowledge base is built as a set files written using the OWL formalism.

3.2 Wizard Implementation Description

The development of the wizard relies on Visual studio 2010 as the development Environment and C#, as the Programming language of the .Net Framework. Moreover, the wizard has been developed using windows form technology and the standard components for choosing graphically the features of the platform to deploy. Also the steps involved on the personalization of the knowledge base and the voice user interface have been developed using standard an well known windows form components.

3.2.1 Step By Step Description of the Wizard Forms

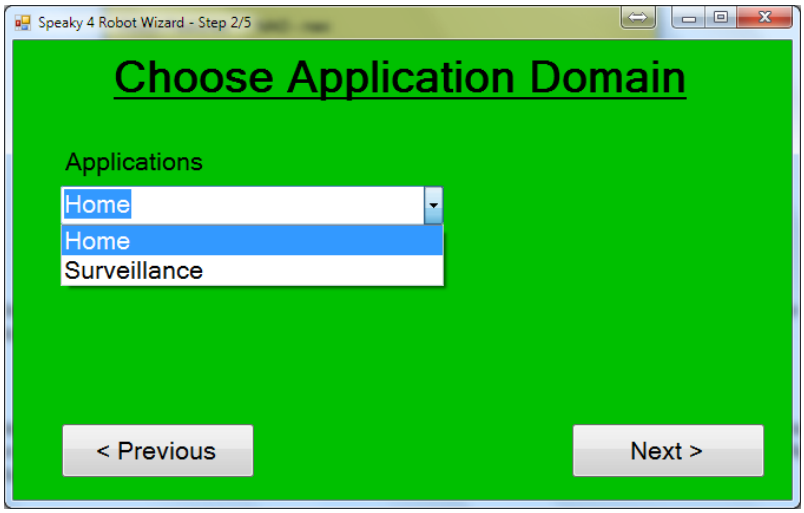
The first step of the development wizard (**WZ1**) is choosing the Robot platform to use; for example, using a humanoid Robot or a wheeled Robot:



In this step all the Robot specific data are selected by the wizard and will be used in the subsequent steps of the process. The available domains are specific to the Robot selected in this step; therefore, this step filters the list of applications available and its voice user interface as well as the knowledge base of the specific Robot.

The choice of the robot platform corresponds to a selection in the Platform Knowledge Base of the frames related to the robot capabilities. For example, if a robot able to move in an environment (e.g. a wheeled robot) is selected, all the frames implying a motion action will be selected (e.g. motion, bringing).
The frames so far selected will be used together with all the information about frame elements and relative lexicon to populate the language structure definition step (WZ3).

The second step (**WZ2**) requires the choice of the application domain where the Robot will be used: for example, the Home service domain or the outdoor- surveillance;



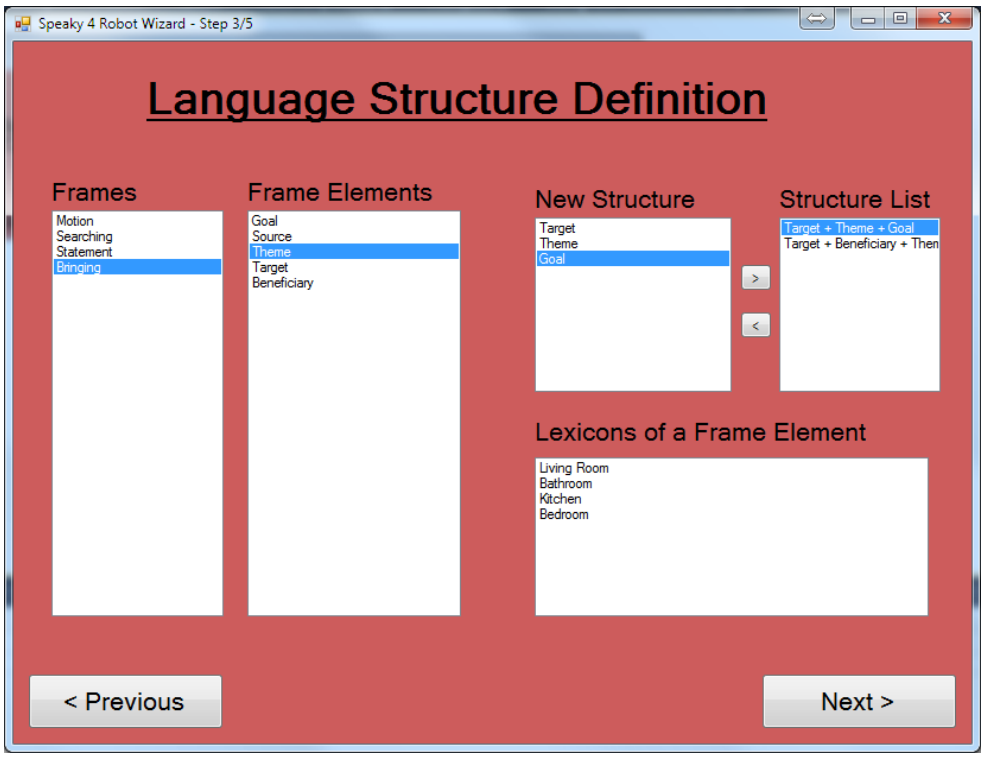
The first two steps altogether determine the appropriate platform and application grammars to be personalized in the next steps of the wizard; specifically, the “Platform Specific Core Grammar” and “Application Domain Specific Core Grammar” as described in report D2.1.
In this step also the Domain Knowledge Base related to the specific Application Domain will be selected. As for the Platform Knowledge Base also the Domain Knowledge Base will be used to populate the language structure definition step (WZ3).

The third step (**WZ3**) consists in generating the Recognition Grammars [W3C, SRGS 2004] for the Speaky Platform and the XML files that represent the frame hierarchy used by the Semantic Analyzer (see section 2.3).

This step is accomplished starting from the command specification of the platform, basic core grammars, the domain lexicon/grammar and the available linguistic resources. The Language Structure Definition step includes the definition of the syntactic structures combining the frames and the frame elements that are specific to the robotic platform and to the application domain as filtered in the WZ1 and WZ2 steps.

The operator will be able to define these structures as well as add and delete lexicon phrases and tokens for each frame element. This operation can be accomplished for each structure that the operator will define and each lexicon set of a frame element can be personalized based on the use of the defined structure.

The operator can't define or add new frames or frame elements but only combine frame elements to define new language structure and modify the lexicons of the frame elements combined for the defined structures.



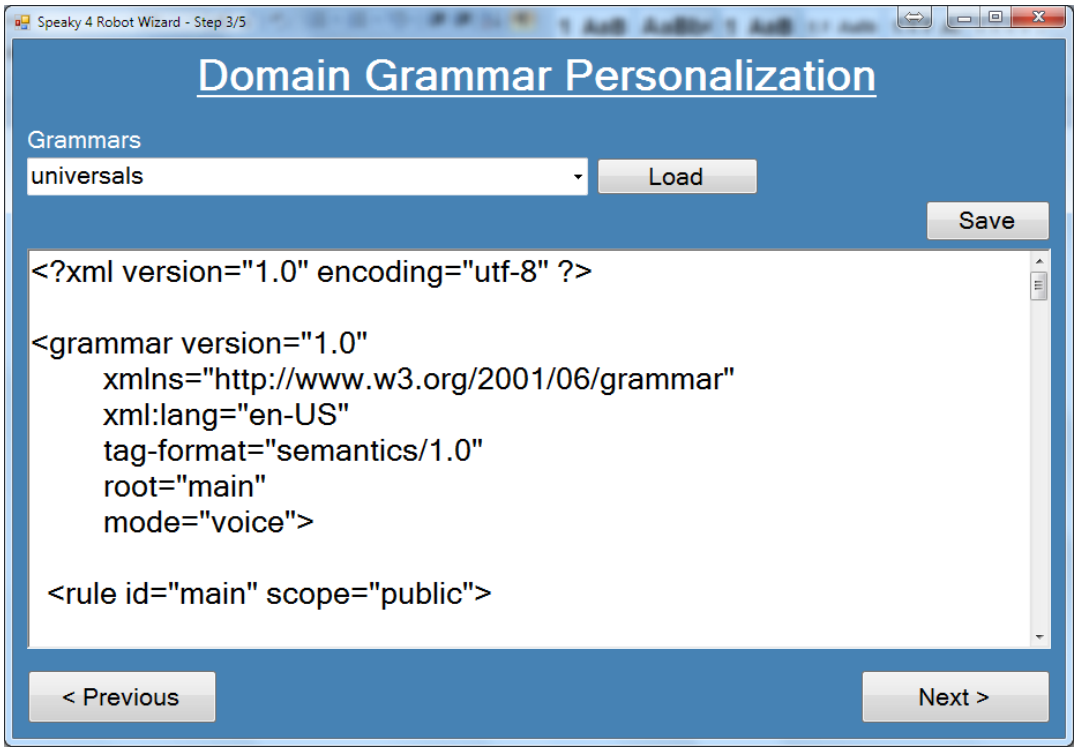
The operator selects the frame it is interested to add to the final application ad all the frame elements of the selected frame will be viewed. The screenshot shows an example where the "Bringing" frame is selected, so that the "Frame Elements" list box can be filled with all the frame elements belonging to the "Bringing" frame.

To create a new structure the operator double clicks the frame elements he wants to add in the order that the various frame elements will be combined as a sequence of frame elements. For example, to create a structure of the frame elements: Target, Theme and Goal, where in the first position of the structure we have the Target, in the second position we have the Theme and in the third position the Goal, the operator executes the following: first he double click in the frame element "Target", so it goes in the first position of the "New Structure" list box, then he double click on "Theme" and finally on the "Goal" frame element so that the new language structure will be any lexicon combination of the sequence {Target, Theme, Goal} for example {bring the magazine in the living room} where Target=bring, Theme=the magazine, Goal=in the living room.

Selecting an element on the New Structure list box the operator can also customize the phrases or tokens of that element in the “Lexicon of a Frame Element” list box. The modification of these lexicons only affects the actual structure that the operator is defining. For example, to customize the lexicons of the “Goal” frame element, one can click on the “Goal” frame element in the “New Structure” list box and the lexicons appear in the “Lexicons of a Frame Element”. Lexicons can then be managed (deleted, modified or added) through an edit Dialog Form.

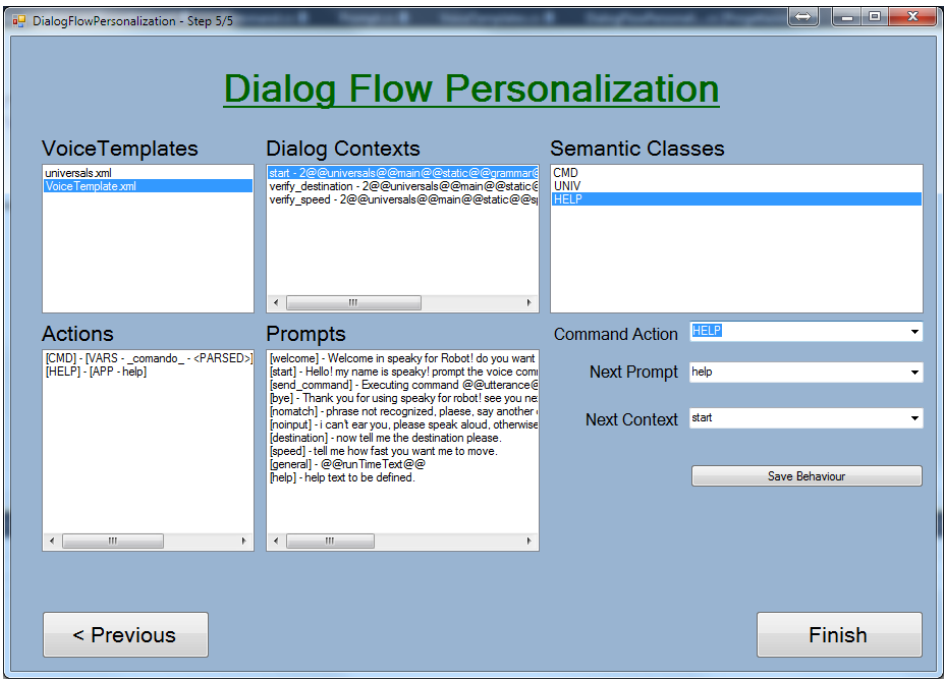
Once the customization is completed, the operator adds the new structure in the “Structure List” by clicking the button with the right arrow. This operation should be accomplished for each new syntactic structure that the operator wants to add in the final system. To modify a new created structure the operator can simply select it in the “Structure List” list box and repeat the previous operations. When all the structures have been defined the operator can click the “Next” button to complete the step. The result is the creation of all the grammars for the system and customize them in the next step as described later.

The fourth step of the wizard (**WZ4**) is the personalization and customization of the grammars; this operation involves the addition/deletion of grammar chunks, phrases, names, objects and specific words depending on the specific application requirements.



this step gives to the operator the possibility to look at the source code [W3C, SRGS 2004] of each grammar, selecting it in the “Grammar” combo box, so that he/she can add, remove or customize phrases or tokens in the grammar depending on the specific domain on which the Robot will be used and particularly the personal end user domain.

The fifth and last step of the wizard (**WZ5**) is the personalization of the Dialog Flow starting form a generic Dialog Flow. In this step, the developer can add or remove dialog states, define new feedback for the end user, personalize predefined feedbacks and even change the actions to be performed by the robot after a particular utterance has been recognized.



The user selects the voice template he is interested in and so can look all the dialog context belonging to it. Dialog context are list of grammars activated by Speaky on that particular context/moment of the dialog.The operator can select a context in order to list all the semantic classes managed in that context. At this stage the user can customize the behavior of that semantic class and save that behavior.

The semantic class is referred as the string representation of the interpretation of the voice command uttered by the end user.

The behavior of a particular semantic class is built of three elements:

- The action to perform as a result of the given voice command
- The Text To Speech feedback to be prompted to the user
- The Next Dialog Context to be loaded by Speaky Platform in order to make the dialog proceed.

For example when the user says any phrase that is interpreted as “HELP” we want the system to remain in the same context (context are defined in the voce template files), to execute the action corresponding to the “HELP” identifier (cmd.xml file) and to give the text to speech feedback (the Prompt in the tts.xml file) identified with the “help” “id”.

So as depicted in the figure above we selected the “HELP” semantic class of the first context (start) and defined the behavior by selecting the “Command Action” identified by “HELP” string, the “Next Prompt” identified by “help” string and the “start” context that is the same starting context.

All the available actions can be viewed in the Action list Box. This is the list obtained reading the cmd.xml file mentioned in the previous section. All the available feedbacks can be viewed in the Prompts list Box. This is the list obtained reading the tts.xml file as described in the previous chapter. The destination dialog context is a context identifier included in the Voice Template Files available for the application.

References

[D2.1]	E. Bastianelli, G. Castellucci, F. Giacomelli, N. M. Manes, L. Iocchi, D. Nardi, V. Perera, “Robotic Voice Development Kit Design”, Echord S4R Technical Report D2.1, April 2012.
[Framenet]	Framenet. URL https://framenet.icsi.berkeley.edu/fndrupal/home .
[W3C, SRGS 2004]	W3C. Speech recognition grammar specification version 1.0 w3c recommendation, March 2004. URL http://www.w3.org/TR/2004/REC-speech-grammar-20040316/
[W3C, SISR 2006]	W3C. Semantic interpretation for speech recognition (sisr) version 1.0 candidate recommendation, January 2006. URL http://www.w3.org/TR/2006/CR-semantic-interpretation-20060111/ .
[W3C, SSML 2004]	W3C Speech Synthesis Markup Language (SSML) Version 1.0 W3C Recommendation 7 September 2004 http://www.w3.org/TR/speech-synthesis/
[SPEAKY SDK PG, 2009]	Speaky Sdk Programming Guide Version 1.0 – Mediavoice S.r.l.
[W3C, OWL-REF 2004]	OWL Web Ontology Language Reference W3C Recommendation 10 February 2004 http://www.w3.org/TR/owl-ref/

Task 3: System Development, Implementation, Integration and Testing [M 6-12]

Participant	Role	Person-months
MV	Leader	14
UR1	Participant	3

Objectives:

This task describes the activities necessary for the development of the Robotic Voice Development Tool (starting from design process):

- Implementation and testing of the communication protocol (Integration System) between Speaky and the Robot environment, to transmit the voice commands received by the user to the Robot, and, conversely, to transmit the results of operations performed by the Robot to the operator (by the TTS).
- Implementation and testing of the wizard, which will guide the operator in the definition of the VUI, and then, must produce the data structures for the TTS and for ASR, called:
  - Grammars, set of allowed words;
  - Vocal Prompts, set of sentences to be read after an action;
  - Vocal States, set of states that specify which grammars should be active and when.
- Implementation and test of a reader, which can detect semantic knowledge, for example, contained in an ontology, and provide the wizard with the necessary data, to help the operator in making his choices (e.g. regarding the names of objects…).

The output of this task will support the activities conducted in T4.

Description of work, and role of participants:

T3.1: Integration System Development
In this first part of the task the participants, Mediavoice and DIS, integrate their respective systems.

The work will be split up into different stages:

- Development and implementation of the communication methods (physical and logical protocols) and of the selected formalisms (descriptive language and related features), which the subsystem exchange information with, using dimensional parameters coming out of the previous phase of the analysis.
- Development of the physical Database, repository of the data shared between the subsystems.

T3.2: Wizard Development
Mediavoice will mainly carry out this part. The work will be split up into different stages:

1. Development of the voice templates, related to the grammars, the voice prompts and the vocal states, that will be used and filled by the wizard, depending on the operator's needs;
2. Implementation of the wizard in accordance with the designed diagrams resulting from the analysis, so that as the user chooses his preferences, the systems fills the templates necessary for the voice interface and gives the reader instructions to select the data necessary for the following choices;

3. Consistency/operational test and wizard calibration;

T3.3: Reader Development
Mediavoice will mainly carry out this part.

The work will be split up into different stages:

- Analysis of the designed diagrams and selection of the best environment for the implementation, taking into consideration the formalisms used in the ontology representation.
- Implementation of the functionalities observed in T2 related to all the reader's components with a particular focus on the consistency of the emerging scenarios (relationship between ontology and voice);
- Implementation of the integration between the reader and the wizard;
- Operational test of the reader.

Technical Support

For any issue or support please contact manes@mediavoice.it

Appendix A: Xml Result Message for the Robot

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.7.0" class="java.beans.XMLDecoder">
<object class="it.uniroma1.nlp.robotic.models.Frame">
  <void property="command">
<string>GOTO</string>
  </void>
  <void property="frameElements">
   <object class="java.util.Vector">
    <void method="add">
     <object class="it.uniroma1.nlp.robotic.models.FrameElement" id="FrameElement0">
      <void class="it.uniroma1.nlp.robotic.models.FrameElement" method="getField">
       <string>roleFiller</string>
       <void method="set">
        <object idref="FrameElement0">
         <void class="it.uniroma1.nlp.robotic.models.FrameElement" method="getField">
          <string>core</string>
          <void method="set">
           <object idref="FrameElement0">
            <void property="elementName">
             <string>goal</string>
            </void>
            <void property="position">
             <int>1</int>
            </void>
            <void property="words">
             <object class="java.util.ArrayList">
              <void method="add">
               <object class="it.uniroma1.nlp.robotic.models.Word">
                <void property="confidence">
                 <float>0.001149209</float>
                </void>
                <void property="surface">
                 <string>to</string>
                </void>
               </object>
              </void>
              <void method="add">
               <object class="it.uniroma1.nlp.robotic.models.Word">
                <void property="confidence">
                 <float>0.6600994</float>
                </void>
                <void property="surface">
                 <string>the</string>
                </void>
               </object>
              </void>
              <void method="add">
               <object class="it.uniroma1.nlp.robotic.models.Word">
                <void property="confidence">
                 <float>0.9073448</float>
                </void>
```

```
              <void property="surface">
               <string>kitchen</string>
              </void>
             </object>
            </void>
           </object>
          </void>
         </object>
         <boolean>true</boolean>
        </void>
       </void>
      </object>
      <string>to(0.001149209) the(0.6600994) kitchen(0.9073448)</string>
     </void>
    </void>
   </object>
  </void>
  <void method="add">
   <object class="it.uniroma1.nlp.robotic.models.FrameElement">
    <void property="elementName">
     <string>speed</string>
    </void>
    <void property="position">
     <int>2</int>
    </void>
    <void property="words">
     <object class="java.util.ArrayList"/>
    </void>
   </object>
  </void>
 </object>
</void>
<void property="lu">
 <object class="it.uniroma1.nlp.robotic.models.Lu" id="Lu0">
  <void class="it.uniroma1.nlp.robotic.models.Lu" method="getField">
   <string>words</string>
   <void method="set">
    <object idref="Lu0">
     <void class="it.uniroma1.nlp.robotic.models.Lu" method="getField">
      <string>roleFiller</string>
      <void method="set">
       <object idref="Lu0"/>
       <string>go(0.7384719)</string>
      </void>
     </void>
    </object>
    <object class="java.util.ArrayList">
     <void method="add">
      <object class="it.uniroma1.nlp.robotic.models.Word">
       <void property="confidence">
        <float>0.7384719</float>
       </void>
       <void property="surface">
        <string>go</string>
       </void>
      </object>
     </void>
    </object>
```

```
     </void>
    </void>
   </object>
  </void>
  <void property="name">
   <string>motion</string>
  </void>
  <void property="totalConfidence">
   <float>0.7553115</float>
  </void>
</object>
</java>
```

Appendix B: Messages Exchanged between Semantic Analyzer and Dialog Manager

In this appendix is described the message system that realizes the communication between the Semantic Analyzer and the Dialog Manager. Each of this command is sent from the Semantic Analyzer to the Dialog Manager in order to change the state of the system and drive the interaction with the robot and with the user (see section 2.4 for a scheme of the Dialog Manager).

`[SEMANTIC_ANALYZER:SINGLE_RESULT]` followed by an XML String (in the Appendix A we report a sample XML message mentioned in this type of response). This message triggers the **single_result** state of the Dialog Manager.

`[SEMANTIC_ANALYZER:SECOND_STEP]` followed by the grammar name that has to be loaded by the Speaky Platform in order to execute the recognition phase. This message triggers the **second_step** state.

`[SEMANTIC_ANALYZER:VERIFY_USER]` followed by the dialog state that has to be loaded by the Dialog Manager. In this case the user is asked to utter a new voice command that can be part of the previous one, in order to complete the command or a completely new one.

`[SEMANTIC_ANALYZER:FAIL]` this message coming from the Semantic Analyzer triggers the **fail** state, that correspond to a wrong recognition for the Speaky Platform.

Finally, a simple message protocol has been implemented to collect all the different result of the recognition process.

`[SEMANTIC_ANALYZER:NEXT]` this message coming from the Semantic Analyzer is used to ask for the next result returned by the Speech Recognition System, when more than one speech result is available.

`[END_RESULTS]` is the message sent to the Semantic Analyzer from the Voice Browser when no more result are available.