REALIZATION OF THE COMPREHENSION-DIALOGUE MANAGEMENT CORE

Development and testing of a VoiceXML application

1- Scope of the application

You will develop a dialogue system dedicated to the management of meeting rooms. You will develop and test the part of the application in charge of **understanding and managing the dialogue** (written) given the task at hand.

The chosen approach is a frame-based / slot filling approach, based on rules (grammars)

The voice inputs and outputs will be simulated and only the text supposed to come out of the recognition module will be processed. Eventually, a confidence score may be added to the text to influence the choice made in the dialogue management.

The core of the application consists of one or more VoiceXML documents (.vxml files dedicated to dialogue management) and appropriate grammars (.grxml files dedicated to comprehension).

Task to be carried out: identify the query to be executed (= transaction to be carried out towards a reservation management database). Indeed, at the end of the dialogue, the system is supposed to generate an SQL query (**`insert, update, select, delete`**) to update the reservation database. The database will of course be simulated, only the elements necessary to perform the query will be extracted from the different phases of the dialogue with the user.

Based on the V0 version that will be provided to you, you will develop a version that will allow you to collect the information related to the reservation to be made.

2- Components to be used

Use the **Optimtalk** software (under Windows) to interpret a user statement (semantic decoding) and manage a dialogue (action to be taken).

1) You will mainly use 2 tools:
   - **ot_grammar_tester.exe**: allows to test a grammar written in SRGS (Speech Recognition Grammar Specifications) language file with `.grxml` extension.

     `ot_grammar_tester.exe` *`path_to_file.grxml`*

   - **ot_vxml_interpreter.exe :** allows to launch a VoiceXml application (`.vxml` file extension) to test it "offline", i.e. without speech recognition and without speech synthesis (text interface only).

     `ot_vxml_interpreter.exe` *`path_to_appli.vxml`*

2) These tools are located in the **C:\Optimtalk** directory.
   a. Open a DOS command window `(cmd)`
   b. Go to the mentioned directory (license problem otherwise)
   c. indicate the path to the grammar or application to be tested.
      DOES NOT NOTCREATE OF FILE UNDER C:\Optimtalk,
      work in your home (to see according to the machines of the MFJA)

3) As the error messages are not very explicit in the basic version, you can consult the log files which are updated at each execution of the interpreter. These files are located in the application directory.

4) An attached pdf document concerning the syntax of .grxml and .vxml is available in moodle. You can also use the set of examples in the "Examples" folder or via the URL *`https://help.voxeo.com/go/vxml/elements.overview`* (see moodle).

3- TP : Specifications

I- **TP n°1**

Objective n°1 = Retrieve and test the V0 version provided.

5) You will first use the `ot_grammar_tester.exe` tool to test each of the grammars provided, understand what they do, how it is implemented and what the result is.
   You will first use the `ot_grammar_tester.exe` tool to test each of the grammars provided, understand what they do, how it is implemented and what the result is.
   - `gram_number_v3.grxml`
   - `grammar_num_ab.grxml`
   - `grammar_act_lang_confirmation.grxml`
   - `grammar_help.grxml`
   - `grammar_dates_v3.grxml`

6) Use the `ot_vxml_interpreter.exe` tool to run the V0 version of the application provided to you and analyze the contents of the vxml file.

7) Change the dialog maintenance to check the validity of the subscriber number between 1 and 2500. This identification phase simulates a logon phase for the "logon task" service.

Objective n°2 = Write a new V1 version from the V0 provided

Based on the V0 version, develop a V1 version allowing the connected user to make a room reservation request. We will consider that a reservation is characterized by a **`date`**, a **`time of beginning of reservation`**, a **`duration`**, a **`name of room`** with possibly a **`name of building`** *(room 108 or room 108 of the U3)* and possibly an **`object of meeting`** **(***project Xfree, PGE, ...***)**.

8) Change the dialog maintenance to take the reservation date into account. To do this, use the external grammar provided and request **explicit confirmation of** the information collected.

9) Predict how the system will behave when the user requests help. The help message must be relevant.

10) Modify the dialog management to be able to exit the application at any time (= make the interaction flexible).

II- **TP n°2** :

Objective: continue the modifications to obtain the V1 version

11) Test the grammar to recognize any **time schedule** (assignment 1) stated in different ways: *quarter to twelve, eleven forty-five...* and extract the meaning (corresponding numerical values) consisting of two fields ***H*** and ***MN*** as well as the field ***text*** and the total number of minutes ***NTMN*** :

```
Schedule {H: 11
          MN : 45
          text : quarter to twelve NTMN
          : 705}
```

12) Change the dialog maintenance to include this new information, maintain the help request and explicitly ask for confirmation.

13) You will then check the validity of the information collected by the system and you will take into account the application constraints concerning the time slots for room reservations:
    - a valid time is between 00:00 and 23:59 (75 hours is not a valid time)
    - in this application, rooms can only be booked between 7:30 and 19:30

    Messages must therefore be appropriate to the situation

III- **TP n°3 :**

Objective: finalize the V1 version

Define the grammars needed to identify a **duration**, a **room name** with possibly a **building**, and a **meeting object** (whose mention is optional) and extract in each case the relevant information needed for the application. Test these grammars by making a test report (copy and paste the screen results). This test report will be part of the renderings.

14) Integrate the corresponding information requests in the dialogue management part. The requests will be made according to a **directive interaction** strategy **and an explicit confirmation strategy**. In each case the error or help messages will be adapted to the context.

IV- **TP n°4 :**

Objective: finalize the **V2** version

15) Duplicate your vxml application (not the grammars) to make a second **V2** version

16) Modify the date confirmation mode by integrating an **implicit confirmation**. This mode involves giving the value to be confirmed and querying the user for the next piece of information (in this case the time).

```
You want to reserve a room for November 2, 22. At what time?
```

The user can then answer :

- `At ten o'clock` which implicitly validates the date
- `No,` which cancels the date and must provoke a new request
- `No on November 12` which cancels the date and gives it a new value; the system does not ask again and continues.

Write the grammar `confirm_Date_give_Time.grxml` which allows to manage the 3 types of answers above.

Integrate this grammar into the dialog management and modify the system behavior accordingly.

17) Transform the remaining explicit confirmations to pool the confirmation request via a **new form**
`<form>` which will be called as a subdialog (see poly VXML for more details heading **`<subdialog>`** and examples on moodle).

18) Show the task tree associated with this application (see course), as well as the dialog model of the **V2** version.

OUTPUT: Make a clean archive (no bulk files) with your name and containing

1- the complete **code** of **V1** and **the** associated **test report** (grammars and dialog).
2- the complete **code** of V2 and the associated **test report** (new grammars and dialog).
3- the various **diagrams requested** (Q18 scan of a clean diagram)

IV- **TP n°5 :**

Objective: implement **a mixed interaction strategy**

Retrieve the `Pizza VXML` example from the `Examples` folder available in moodle.

19) Test the `pizza.grxml` grammar
20) Represent the associated automaton
21) Testing the `pizza.vxml` application
22) Represent the associated dialogue model
23) `Using the pizza.grxml` grammar as a guide, write the `RESERVATION.grxml` grammar that recognizes statements like :

```
I would like to reserve room three hundred and fourteen at the MFJA
for the EMP on Monday, October 1 at ten o'clock

Request to reserve room 108 of the U3 at two o'clock on October 10,
2002
```

24) Test this grammar and make a test report to put on moodle.

OUTPUT: Put an archive on moodle containing the code of the reservation grammar (Q24), the corresponding tests (Q25), and the diagrams made to the automaton (Q21) and to the dialog model (Q23).