Tutorial
Vocal Interaction

(Ph. Truillet)
September 2023

1. A Voice-activated application

We wish to design and implement a **non-visual** application (input and output including speech and possibly sound - music, recorded messages, etc.) allowing a user to **add, remove, manipulate** foods displayed on a screen in **order** to compose the content of a "gourmet" plate of dessert(s). (e.g. coffee, tea, orange juice, “creme brulee”, profiteroles, etc.). **There will be exactly two desserts and one drink in the mixed dish.**
The physical layout is part of the problem!
You will code your application **in the language you want** (using Processing.org can be a good alternative).

It should be possible to perform all the requested actions in a purely vocal way in input and output.

In order to develop our multimedia application, we will primarily use the ivy middleware (software bus) [https://github.com/truillet/ivy/blob/master/README.md], support for the future project on multimodality.

Nota: If you are on Linux or MacOS, you will need to find alternative solutions for speech recognition and synthesis (for example, use **MaryTTS** for speech synthesis or **STT** or **SpeechRecognition**, a python library for speech recognition - see links below)

2. Work expected from this session (2 h)

After the learning phase about speech recognition and synthesis agents running on the ivy software bus, the purpose of this session is to:

1. **define the recognition grammar** (speech commands or "pseudo-natural" language) that will be used by your application, manage the semantic output (i.e. the concepts associated with the spoken words) and the confidence rate.
2. define **voice and sound feedbacks** to be synthesized and used by your application.
3. develop an application to display shapes on the screen (in java, Processing, python ... or another language).
4. and finally develop **the dialogue controller** using a state machine (which can either be separate or included in the display application). The controller will rely on an ivy message exchange with at least the speech recognition and synthesis modules.

At the end of the session, you will have produced a **testable high-fidelity prototype** of the required system.

Nota: you can use some already coded ivy agents (presented in annex)

3. Links

- **ppilot5** (Text-to-Speech), **sra5** (Automatic Speech Recognition), **…** voice interaction agents:
  `https://github.com/truillet/upssitech/tree/master/SRI/3A/IHM`

- **ivy libraries:**
  `https://github.com/truillet/ivy/blob/master/README.md`

- You may also use
  - **MaryTTS** (`https://github.com/marytts/marytts`), Test-to-Speech server written in Java
  - **ElevenIO** (`https://elevenlabs.io`), TTS accessible through a REST API (`https://api.elevenlabs.io/docs`)

- **STT** : Speech Recognition for Java/Processing based onr Google Chrome and websockets:
  `http://florianschulz.info/stt`
  Nota: You can use the webpage `https://www.irit.fr/~Philippe.Truillet/stt.html` in order to launch the voice-recognition server.

- **SpeechRecognition**, library written in Python: `https://pythonprogramminglanguage.com/speech-recognition/`

Don't hesitate to ask if an agent exists: it may already be the case! And then, you can **CODE** your own agents according to **YOUR** desires! ☺

Annex 1 – how to use sra5

sra5 is an ivy agent that uses native Windows SAPI 5.x Speech Recognizer and is able to send back **two kinds of solutions** provided by the Speech Recognizer under **two different formalisms**:

Launch the agent with CLI

```
sra5 -b 127.255.255.255:2010 -p on -g grammar.grxml
```

By default, sra5 uses the local grammar file `grammar.grxml`

- **-b** IP adress + port
- **-p** parse result option (parse[1] mode **on** or **off**)
- **-g** grammar file used (grXML file – cf. `http://www.w3.org/TR/speech-grammar`)

[1] *The parsing mode consists of returning the semantic output rather than the spelling string as the result*

Feedbacks (ONLY provided on the ivy bus)

- **sra5 Text=** orthographic_output **Confidence=**confidence_rate (if *parse* flag is off)
- **sra5 Parsed=**result **Confidence=** confidence_rate **NP=**xx **Num_A=**xx where NP is current result number since sra5 activation and Num_A, the alternative number (if *parse* flag is on)
- **sra5 Event=**{Grammar_Loaded | Speech_Rejected}: events output provided by Speech Recognizer.

Commands (ONLY provided on the ivy bus)

- **sra5 -p** {on | off} sra5 switches from one mode to the other one (*on* → concepts feedback or *off* → orthographic feedback)
- **sra5 –g** **sra5** uses a new speech recognition grammar (given on a local path)

Annex 2 – how to use ppilot5

ppilot5 allows the use of SAPI5 TTS compatible synthesizers.

Launch the agent with CLI

```
ppilot5 -b 127.255.255.255:2010 -r Hortense -o "Microsoft Hortense"
```

By default, ppilot5 uses the first TTS found and appears on the ivy bus under "ppilot5" name

- **-b** IP adress + port
- **-r** agent's name (in the previous example, "*Hortense*")
- **-o** name of the TTS engine used (here, "*Microsoft Hortense*" TTS is the default French voice on Windows)

Commands (ONLY provided on the ivy bus)

* ***Synthesis***

- **ppilot5 Say=**hello **ppilot5** pronounces via the TTS the sentence "hello"
- **ppilot5 SaySSML=**<*SSML_sequence*> **ppilot5** pronounces the SSML sequence and returns **ppilot5 Answer=Finished** when the buffer is empty. Tags <*speak*> and </*speak*> are automatically added

  Example with SSML:

  ```
  ppilot5 SaySSML=I want to speak <emphasis level="strong">louder</emphasis>
  ```

* ***Commands***

- **ppilot5 Command=Stop** TTS is stopped. **ppilot5** returns **ppilot Answer=Stopped**
- **ppilot5 Command=Pause** TTS is paused. **ppilot5** returns **ppilot5 Answer=Paused**
- **ppilot5 Command=Resume** TTS is resumed. **ppilot5** returns **ppilot5 Answer=Resumed**
- **ppilot5 Command=Quit** the agent is closed

* ***Parameters***

- **ppilot5 Param=Pitch:**value Pitch is changed according the given value. **ppilot5** returns **ppilot5 Answer=PitchValueSet:value**
- **ppilot5 Param=Speed:**value Speed is changed according the given value. **ppilot5** returns **ppilot5 Answer=SpeedValueSet:value**
- **ppilot5 Param=Volume:**value Volume is changed according the given value. **ppilot5** returns **ppilot5 Answer=VolumeValueSet:value**