# Utilizzare la libreria

Cervella Davide

November 2019

La libreria fornisce gli strumenti per poter integrare alcune delle funzionalità tipiche di Embodied Conversational Agent. Attraverso alcuni file di configurazione sarà possibile inserire ed utilizzare strumenti quali Speech To Text, Text To Speech, Natural Language Understanding.
Attualmente queste funzioni sono implementate attraverso i servizi cognitivi di **Azure** per STT e TTS e **LUIS** per quanto riguarda la parte di NLU. I moduli che si occupano di questo possono essere facilmente sostituiti con altre tecnologie in quanto trattati in maniera isolata dal resto (purchè rispettino il modo di interfacciarsi con le altri componenti dell'architettura).

Se si desidera invece utilizzare la libreria con i servizi attualmente offerti è necessaria una veloce fase di **configurazione**.

## 1 Configurazione

La fase di configurazione avviene principalmente tramite la creazione/customizzazione di alcuni file XML presenti nella cartella **Resources/Ita**. Possono essere create nuove cartelle per creare configurazioni diverse a seconda della lingua. Per settare la lingua da utilizzare basta impostarlo nello script **Configuration.cs** tramite il metodo *SetLanguage* (la stringa che definisce la lingua deve essere uguale al nome della sottocartella in Resources). Le principali componenti da configurare riguardano:

1. indicare le credenziali Azure e Luis necessarie per comunicare con i servizi online.
2. definire quali messaggi necessitano dei servizi di TTS e il loro contenuto;
3. (opzionale) definire quali azioni (tipicamente per la Gamification) si vogliono aggiungere all'applicazione e dunque impostarne i paramentri tipici;
4. (opzionale) definire quali messaggi sono abilitati e quali no in base alla tipologia di gioco prevista (es. training, rehearsal o examination);
5. (configurazione di default fornita) definire un Emotion Model che determina come le appraisal variables modifichino le emozioni dell'ECA;
6. registrare gli **Intent** previsti per la propria applicazione (definiti sul portale LUIS).
7. (opzionale) definire le regole del gioco in termini di quali Nodi vadano eseguiti prima di altri;

## 2 Credenziali per accesso ai servizi cognitivi

Permette di definire le credenziali necessarie per l'utilizzo dei servizi cognitivi Microsoft. Di seguito viene riportata la struttura del file XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  !-- By default, LUIS recognizes intents in US English(en-us).
  Other examples: "de-de", "it-it" -->
  <luisAttributes>
    <luisAppId>il tuo AppID</luisAppId>
    <luisAppKey>il tuo AppKey</luisAppKey>
    <luisRegion>region dell'App luis</luisRegion>
    <language>linugua dell'app luis</language>
  </luisAttributes>

  <ttsParameters>
    <serviceId>il tuo service ID</serviceId>
    <serviceZone>region</serviceZone>
  </ttsParameters>

</configuration>
```

## 3 Definizione dei messaggi e SmartAction

Il file permette di definire sia messaggi di carattere generico che messaggi legati allo stato in cui si trova una determinata SmartAction. Riguardo le SmartActions è possibile settare ulteriori parametri che influenzano il comportamento dell'ECA in relazione all'andamento delle attività svolte dall'utente.

tag **tts**
viene utilizzato per definire dei messaggi di tipo generico che possono essere richiamati in qualsiasi punto del codice. Per aggiungere un messaggio di questo tipo è sufficiente aggiungere un nuovo tag (es. Presentation) che potrà dunque essere richiamato tramite la stringa "Presentation" ed il metodo:
*ECA.Speech("Presentation");*

Tramite l'attributo **ecaID** di tale tag è possibile specificare a quale ECA appartiene. Tale ID dovrà essere lo stesso utilizzato nell'enum **Ecas** definito nello script *ECAManager*. Nel caso in cui non dovesse essere specificato (null) il messaggio sarà associato a tutti gli eventuali ECA dell'applicazione.

**actions**
Utilizzato per definire le diverse SmartActions presenti nell'applicazione. In particolare il nome dell'azione dovrà essere lo stesso utilizzato nell'Enum SmartActions.

tag **actions/tts**
Permette di definire messaggi legati ad una specifica azione. Ci sono due tipi di messaggi:

- di tipo generico (sempre legati all'azione): che possono essere utilizzati ad esempio per dare una descrizione, per fornire aiuto, un messaggio di chiusura ecc. É possibile aggiungere quanti messaggi si vuole di questo tipo.
- legati all'andamento dell'azione: si tratta dei messaggi associati alla variazione dei valori relativi ad Accuracy e Staging. Tali messaggi vengono lanciati nel momento in cui si ha un **cambiamento di stato** in Bad, Good o Normal. É possibile definire per quali valori numerici tale passaggio avviene ("xLimit").

Attributo **weight**
indica l'importanza dell'azione ed influenza la probabilità che un messaggio venga riprodotto dall'ECA oppure no (in base al livello di presenza dell'ECA stesso). Valori possibili: Low, Medium, High.

Di seguito viene riportata la struttura del file XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<game>
  <tts>
    <Presentation></Presentation>
    <Misunderstood></Misunderstood>
  </tts>

  <actions>
         <action name="Same action name in SmartActions Enum">
    <tts weight="High">
      <Description>Descrizione del Task</Description>
      <EndTask> meg finale </EndTask>
      <Help>msg di aiuto</Help>
      <Criteria>
        <Accuracy badTxt ="" badLimit="0.4"
             normalTxt ="" normalLimit ="0.8"
             goodTxt = "" goodLimit ="1">
        </Accuracy>
        <Staging badTxt ="" badLimit ="0.8"
                normalTxt ="" normalLimit ="0.4"
                goodTxt ="" goodLimit="0.0">
        </Staging>
      </Criteria>
    </tts>
    </action>
  </actions>
</game>
```

# 4 Abilitare o disabilitare messaggi

Nel caso in cui si prevedano diverse modalità di gioco è possibile che in alcune di queste non tutti i messaggi debbano essere riprodotti mentre in altre si. Nel caso in cui questa funzionalità non dovesse essere necessaria è possibile omettere tale file di configurazione. In quest'ultimo caso tutti i messaggi saranno sempre abilitati di default.
Di seguito viene presentata la struttura del file XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<activatedMsgs>

  <gameType name ="Training">
      <EndTask isActive ="YES"> </EndTask>
      <Help isActive ="YES"> </Help>

      <Misunderstood isActive ="YES"> </Misunderstood>
      <Presentation isActive ="YES"> </Presentation>
  </gameType>

  <gameType name ="Rehearsal">
      <EndTask isActive ="NO"> </EndTask>
      <Help isActive ="YES"> </Help>

      <Misunderstood isActive ="YES"> </Misunderstood>
      <Presentation isActive ="YES"> </Presentation>
  </gameType>

  <gameType name = "Examination">
      <EndTask isActive ="NO"> </EndTask>
      <Help isActive ="NO"> </Help>

      <Misunderstood isActive ="NO"> </Misunderstood>
      <Presentation isActive ="YES"> </Presentation>

  </gameType>
</activatedMsgs>
```

# 5 Configurare l'Emotion Model

L'Emotion model definisce come le Appraisal Variables vanno ad influenzare le emozioni percepite dall'Embodied Conversational Agent. Tali variabili possono essere utilizzate in qualunque parte del codice per andare ad etichettare degli eventi come:

- Good;
- Bad;
- UnexpectedPositive;

- UnexpectedNegative;
- Nice;
- Nasty;

Ognuna di queste etichette/variabili influenza una o più delle seguenti emozioni:
**Joy, Trust, Fear, Surprise, Sadness, Disgust, Anger, Anticipation**.
Ogni emozione ha un valore compreso tra -1.5 e +1.5 ed un livello (che dipende dal valore stesso) tra **Low, Medium, High** al quale possono essere associate delle emozioni secondarie così come definito nella **Plutchik's Wheel**.
É fornito un file di configurazione di default che può essere personalizzato per generare dei comportamenti diversi.
Di seguito viene riportata la struttura del file XML.

```
<?xml version="1.0" encoding="utf-8"?>
<model>
  <Good>
    <Joy>+0.3</Joy>
    <Sadness>-0.3</Sadness>
    <Surprise>-0.3</Surprise>
    <Anger>-0.1</Anger>
  </Good>

  <Bad>
    <Joy>-0.3</Joy>
    <Sadness>+0.3</Sadness>
    <Surprise>-0.3</Surprise>
    <Anger>+0.1</Anger>
  </Bad>

  <UnexpectedPositive>
    <Surprise>+1</Surprise>
    <Joy>+0.4</Joy>
    <Sadness>-0.3</Sadness>
    <Anger>-0.2</Anger>
  </UnexpectedPositive>

  <UnexpectedNegative>
    <Surprise>+1</Surprise>
    <Joy>-0.3</Joy>
    <Sadness>+0.4</Sadness>
    <Anger>+0.2</Anger>
  </UnexpectedNegative>

  <Nice>
    <Surprise>+0.2</Surprise>
    <Joy>+0.2</Joy>
    <Sadness>-0.4</Sadness>
    <Trust>+0.2</Trust>
```

```
    <Anger>-0.6</Anger>
  </Nice>

  <Nasty>
    <Disgust>+1</Disgust>
    <Joy>-0.5</Joy>
    <Sadness>+0.3</Sadness>
    <Surprise>+0.2</Surprise>
    <Trust>-0.2</Trust>
    <Anger>+0.1</Anger>
  </Nasty>

</model>
```

# 6 Registrazione degli Intents creati attraverso il portale LUIS

Una volta creati i vari Intents attraverso il portale LUIS, ogni script che necessita di reagire al riconoscimento di uno uno di essi dovrà estendere l'interfaccia *IIntentHandler* ed implementare i metodi *SubscribeHandlerToIntentManager* ed *Handle*. Il primo è utilizzato per fare in modo che, nel momento in cui venga riconosciuto un particolare Intent si possa invocare il metodo Handle appropriato. Il secondo è utilizzato per reagire all'Intent appena riconosciuto.
Di seguito vengono mostrati due esempi di override dei metodi appena menzionati.

```
public void SubscribeHandlerToIntentManager()
    {
        // Add intents defined in LUIS portal in this way:
        // IntentManager.Instance.AvailableIntents.Add(
             //"Intent NAME", this);

        //EXAMPLE:
        IntentName = new List<string> { "None", "Help"};
        IntentManager.Instance.AddIntentHandler(IntentName[0], this);
        IntentManager.Instance.AddIntentHandler(IntentName[1], this);
    }
```

Dove *Intent name* (string) deve **necessariamente** corrispondere al nome utilizzato sul portale LUIS.

```
public void Handle(Intent intent)
    {
        switch (intent.IntentName)
        {
            case "Presentation":
                SendMessage("Presentation");
                break;
```

```
            case "Other":
                // some method
                break;
        }
    }
```

# 7 Influenzare emozioni dell'ECA

Per influenzare le emozioni dell'Embodied Conversational Agent è necessario avere un EmotionModel.xml. Questo permette di agire sulle Appraisal Variables piuttosto che direttamente sui valori delle singole emozioni. Per fare questo è necessario accedere al campo EmotionManager dello specifico Embodied Conversational Agent ed aggiornare le sue emozioni classificando l'evento appena avvenuto con una specifica variabile. Un esempio di utilizzo:

```
//get one of available ECAs from EcaManager
ECA myEca = EcaManager.Instance.AvailableEcas[Ecas.Default];
//update emotion of myEca
myEca.Emotion.updateEmotion(AppraisalVariables.Bad);
```

# 8 ECAAnimator

Si tratta di una classe (sempre associata al GameObject che rappresenta l'ECA) che permette di agire a livello grafico sulle azioni svolte dall'Agente. Per cui permette ad esempio di riprodurre l'audio, settare i parametri dell'audio, di far spostare l'ECA nello spazio, di svolgere azioni come "indicare", "guardare" ecc.
Queste funzionalità andranno implementate dal programmatore andando ad effettuare un override dei metodi interessati.