REALISATION DU NOYAU COMPREHENSION-GESTION DE DIALOGUE

Développement et Test d'une application VoiceXML

1- Cadre de l'application

Vous développerez un système de dialogue dédié à la gestion de salles de réunion. Il s'agira de développer et tester la partie de l'application chargée de la **compréhension et de la gestion du dialogue** (écrit) compte tenu de la tâche à réaliser.

L'approche choisie est une approche pas règles (grammaires), orientée par la tâche (frame-based / slot filling)

Les entrées sorties vocales seront simulées et seul le texte supposé sortir du module de reconnaissance sera traité. Il pourra à terme être accompagné d'un score de confiance qui permettra d'agir sur le choix opéré au niveau de la gestion du dialogue.

Le noyau de l'application est constitué d'un ou plusieurs documents VoiceXML (fichiers .vxml dédiés à la gestion de dialogue) et de grammaires appropriées (fichiers .grxml dédiés à la compréhension).

Tâche à réaliser : identifier la requête à exécuter (= transaction à effectuer vers une Base de données de gestion des réservations). En effet à l'issue du dialogue, le système est supposé générer une requête SQL (**`insert, update, select, delete`**) pour mettre à jour la base de réservation. La base de données sera bien sûr simulée, seuls les éléments nécessaires pour effectuer la requête seront extraits des différentes phases du dialogue avec l'utilisateur.

En vous inspirant de la version V0 qui vous sera fournie, vous développerez une version qui permettra de collecter les informations relatives à la réservation à faire.

2- Composants à utiliser

Utiliser le logiciel **`Optimtalk`** (sous Windows) pour l'interprétation d'un énoncé utilisateur (décodage sémantique) et la gestion de dialogue (action à faire).

1) Vous utiliserez principalement 2 outils :
   - **`ot_grammar_tester.exe`** : permet de tester une grammaire écrite en langage SRGS (syntaxe Speech Recognition Grammar Specifications) fichier avec extension `.grxml`.

     `ot_grammar_tester.exe` *`chemin_vers_fichier.grxml`*

   - **`ot_vxml_interpreter.exe :`** permet de lancer une application VoiceXml (fichier extension `.vxml`) pour la tester « offline » c'est-à-dire sans la reconnaissance de la parole et sans la synthèse vocale (interface texte seulement).

     `ot_vxml_interpreter.exe` *`chemin_vers_appli.vxml`*

2) Ces outils se trouvent dans le répertoire **`C:\Optimtalk`**.
   a. Ouvrir une fenêtre de commande DOS (`cmd`)
   b. Se placer dans le répertoire mentionné (problème de licence sinon)
   c. indiquer le chemin menant à la grammaire ou l'application à tester. **NE PAS CREER DE FICHIER SOUS `C:\Optimtalk`**, travailler dans votre home (à voir suivant les machines de la MFJA)

3) Les messages d'erreurs n'étant pas très explicites dans la version basique, vous pouvez consulter les fichiers de log qui sont mis à jour à chaque exécution de l'interpréteur. Ces fichiers se trouvent dans le répertoire de l'application.

4) Un document pdf annexe concernant la syntaxe des fichiers .grxml et .vxml est disponible sous moodle. Vous vous appuierez également sur le jeu d'exemples contenus dans le dossier « Examples » ou via l'URL *`https://help.voxeo.com/go/vxml/elements.overview`* (voir moodle).

3- TP : Cahier des charges

I- **TP n°1**

Objectif n°1 = Récupérer et tester la version V0 fournie.

5) Vous utiliserez d'abord l'outil `ot_grammar_tester.exe` pour tester chacune des grammaires fournies, comprendre ce qu'elles font, comment cela est implémenté et quel est le résultat produit.
   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.
   - `gramaire_nombre_v3.grxml`
   - `grammaire_num_ab.grxml`
   - `grammaire_act_lang_confirmation.grxml`
   - `grammaire_aide.grxml`
   - `grammaire_dates_v3.grxml`

6) Utilisez l'outil `ot_vxml_interpreter.exe` pour exécuter la version V0 de l'application qui vous est fournie et analyser le contenu du fichier vxml.

7) Modifiez la gestion de dialogue pour contrôler la validité du numéro d'abonné compris entre 1 et 2500. Cette phase d'identification simule une phase de connexion au service « tâche connexion ».

Objectif n°2 = Ecrire une nouvelle version V1 à partir de la V0 fournie

En vous inspirant de la version V0, développer une version V1 permettant à l'utilisateur connecté de faire une demande de réservation de salle. On considèrera qu'une réservation est caractérisée par une **`date`**, un **`horaire de début de réservation`**, une **`durée`**, un **`nom de salle`** avec éventuellement un **`nom de bâtiment`** *(salle 108 ou salle 108 du U3)* et éventuellement un **`objet de réunion`** **(***projet Xfree, PGE, ...***)**.

8) Modifiez la gestion de dialogue pour prendre en compte la date de réservation. Utilisez pour cela la grammaire externe fournie et demandez **confirmation explicite** de l'information collectée.

9) Prévoir le comportement du système en cas de demande d'aide de la part de l'utilisateur. Le message d'aide doit être pertinent.

10) Modifiez la gestion de dialogue pour pouvoir sortir de l'application à tout moment (= rendre l'interaction flexible).

II- **TP n°2** :

Objectif : poursuivre les modifications pour obtenir la version V1

11) Tester la grammaire permettant de reconnaître un **horaire** (devoir n°1) quelconque énoncé de différentes façons : *midi moins le quart, onze heures quarante-cinq*… et d'en extraire le sens (valeurs numériques correspondantes) constitué de deux champs ***H*** et ***MN*** ainsi que le champ ***text*** et le nombre total de minutes ***NTMN*** :

```
Horaire {H : 11
         MN : 45
         text : midi moins le quart
         NTMN : 705}
```

12) Modifiez la gestion de dialogue pour intégrer cette nouvelle information, gérer la demande d'aide et demander explicitement sa confirmation.

13) Vous veillerez ensuite à vérifier la validité de l'information recueillie par le système et vous prendrez en compte les contraintes d'application portant sur les plages horaires de réservation des salles :
    - un horaire valide est compris entre 00h00 et 23h59 (75 heures n'est pas un horaire valide)
    - dans cette application, les salles ne peuvent être réservées qu'entre 7h30 et 19h30

    Les messages doivent donc être appropriés à la situation

III- **TP n°3 :**

Objectif : finaliser la version V1

Définir les grammaires nécessaires pour identifier une **durée**, un **nom de salle** avec éventuellement un **bâtiment**, et un **objet de réunion** (dont la mention est facultative) et extraire dans chaque cas l'information pertinente nécessaire à l'application. Testez ces grammaires en faisant un rapport de test (copier-coller les résultats écran). Ce rapport de test fera partie des rendus.

14) Intégrer les demandes d'informations correspondantes dans la partie gestion de dialogue. Les demandes se feront suivant une stratégie d'**interaction directive et une stratégie de confirmation explicite**. Dans chaque cas les messages d'erreur ou d'aide seront adaptés au contexte.

IV- **TP n°4 :**

Objectif : finaliser la version **V2**

15) Dupliquer votre application vxml (pas les grammaires) pour en faire une seconde version **V2**

16) Modifier le mode de confirmation de la date en intégrant une **confirmation implicite**. Ce mode implique de donner la valeur à confirmer et d'interroger l'utilisateur sur l'information suivante (ici l'horaire).

```
Vous voulez réserver une salle pour le deux novembre deux
mille vingt-deux. A quelle heure ?
```

L'utilisateur peut alors répondre :

- `A dix heures` → ce qui valide implicitement la date
- `Non` → ce qui annule la date et doit provoquer une nouvelle demande
- `Non le douze novembre` → ce qui annule la date et lui donne une nouvelle valeur ; le système ne redemande pas et continue.

Ecrire la grammaire `confirmer_Date_donner_Horaire.grxml` qui permet de gérer les 3 types de réponses ci-dessus.

Intégrer cette grammaire dans la gestion de dialogue et modifier le comportement du système en conséquence.

17) Transformer les confirmations explicites restantes de façon à mettre en commun la demande de confirmation via un **nouveau formulaire `<form>` qui sera appelé comme un sous-dialogue** (voir poly VXML pour plus de détails rubrique **`<subdialog>`** et les exemples sur moodle).

18) Représenter l'arbre des tâches associé à cette application (cf cours), ainsi que le modèle de dialogue de la version **V2**.

RENDU : Faire une archive propre (pas de fichiers en vrac) portant votre nom et contenant

1- le **code** complet de la **V1** et **le rapport de test** associé (grammaires et dialogue).
2- le **code** complet de la V2 et le **rapport de tests** associé (nouvelles grammaires et dialogue).
3- les différents **schémas demandés** (Q18 scan d'un schéma propre)

IV- **TP n°5 :**

Objectif : implémenter une **stratégie d'interaction mixte**

Récupérer l'exemple `VXML Pizza` dans le dossier `Examples` disponible sous moodle.

19) Tester la grammaire `pizza.grxml`
20) Représenter l'automate associé
21) Tester l'application `pizza.vxml`
22) Représenter le modèle de dialogue associé
23) En vous inspirant de la grammaire `pizza.grxml`, écrire la grammaire `RESERVATION.grxml` qui permet de reconnaître des énoncés du type :

```
Je voudrais réserver la salle trois cent quatorze à la MFJA pour le
PGE le lundi premier octobre à dix heures

Demande de réservation de la salle 108 du U3 à deux heures le dix
octobre deux mille vingt-deux
```

24) Testez cette grammaire et faire un rapport de test à déposer sur moodle.

RENDU : Déposer une archive sur moodle contenant le code de la grammaire réservation (Q24), les tests correspondants (Q25), et les schémas réalisés à l'automate (Q21) et au modèle de dialogue (Q23).