# REALISATION DU NOYAU COMPREHENSION-GESTION DE DIALOGUE

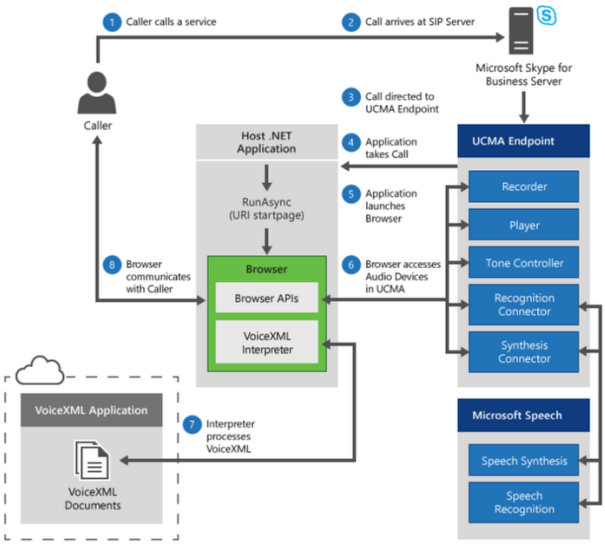
## Développement et Test d'une application VoiceXML

***Development and test of a VoiceXML application***

## 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**) développé en VXML.

**VXML** est un langage de balise (Norme W3C) dédié au développement d'assistants vocaux téléphoniques, utilisé par exemple chez Microsoft (Skype : https://learn.microsoft.com/fr-fr/skype-sdk/ucma/diagram-of-a-voicexml-call - Figure ci-dessous) .




Le noyau de l'application est constitué d'un ou plusieurs documents **VoiceXML** et de grammaires appropriées

- fichiers `.grxml` dédiés à la compréhension
- fichiers `.vxml` dédiés à la gestion de dialogue

**Tâche à réaliser dans la cadre des TP** : interagir avec l'utilisateur pour identifier et collecter les éléments qui caractérisent la réservation d'une salle de réunion. Ces éléments seraient à terme, les constituants de la requête SQL à exécuter pour mettre à jour la 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`). On se limitera ici à l'insertion d'une nouvelle 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 et récapitulés en fin du dialogue.

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 créer.

## Composants à utiliser :

Utiliser le logiciel **`Optimtalk`** (sous Windows, récupérable sur moodle), qui est un interpréteur VXML (VoiceXML).

Vous utiliserez principalement 2 outils :

- **`ot_grammar_tester.exe`** : cet outil permet de tester une grammaire écrite en langage SRGS (syntaxe Speech Recognition Grammar Specifications) fichier avec extension `.grxml`.

  **`ot_grammar_tester.exe`** `C:/chemin…/ma_grammaire.grxml`

- **`ot_vxml_interpreter.exe`** : cet outil 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 (interface texte).

  **`ot_vxml_interpreter.exe`** `C:/chemin…/mon_appli.vxml`

**Attention :**

**Mode ligne de commande** : ouvrir une fenêtre de commande DOS `(avec cmd)` et se placer dans le répertoire Optimtalk (après téléchargement du dossier disponible sur moodle) et indiquer le chemin menant à la grammaire ou l'application à tester.

**Messages d'erreur et débogage :** les messages 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 (`horodatage_session.log`).

**Documentation** : 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` » et sur le site https://evolution.voxeo.com/ (création d'un compte nécessaire).

# TP : Cahier des charges

## I- **TP n°1** :

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

1) Utiliser 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.
   - `gramaire_nombre_v3.grxml`
   - `grammaire_num_ab.grxml`
   - `grammaire_act_lang_confirmation.grxml`
   - `grammaire_aide.grxml`
   - `grammaire_dates_v3.grxml`

2) Utiliser 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 (éditer le fichier en parallèle).

3) Modifier la gestion de dialogue pour contrôler la validité du numéro d'abonné compris entre 1 et 2500. Cette phase d'identification correspond à 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 (sous-tâche de connexion) 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, ...***)**.

4) Modifier la gestion de dialogue pour prendre en compte la **date de réservation**. Utiliser pour cela la grammaire externe fournie et demander **confirmation explicite** de l'information.

5) Prévoir le comportement du système en cas de demande d'aide. Le message d'aide (vocal) doit être pertinent.

6) Modifier la gestion de dialogue pour pouvoir sortir de l'application à tout moment.

## II- **TP n°2** : Poursuivre les modifications pour obtenir la version V1

7) Tester la grammaire permettant de reconnaître un **horaire** quelconque énoncé de différentes façons : *midi moins le quart, onze heures quarante-cinq*… et d'en extraire le sens (valeurs numériques correspondantes) constitué de 4 champs : nombre d'heures, nombre de minutes, nombre total de minutes, texte correspondant au concept (ou entité) horaire.

```
Horaire { H : 11
          MN : 45
          NTOT_MIN : 705
          text : « midi moins le quart »}
```

8) Modifier la gestion de dialogue pour intégrer cette nouvelle information, gérer la demande d'aide et **demander explicitement la confirmation** de l'horaire extrait.

9) Vérifier la **validité de l'information recueillie** par le système. Prendre 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.

   Un message approprié sera communiqué à l'utilisateur suivant le cas.

Travailler en binôme (partagez-vous le travail) pour définir les grammaires nécessaires pour identifier une **durée**, un **nom de salle** avec éventuellement un **bâtiment**, et un **objet de réunion** (fourni par l'utilisateur s'il le souhaite) et extraire dans chaque cas l'information pertinente nécessaire à l'application. **Testez ces grammaires en faisant un rapport de test.**

10) 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.

## III- **TP n°3 :**

Objectif : finaliser la version V1_bis

11) Dupliquer votre application vxml (pas les grammaires) pour en faire une seconde version **V1_bis**.

12) 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-cinq. 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.

13) 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).

14) Représenter l'arbre des tâches associé à cette application (cf cours). A rendre avec le dépôt du code V1 et V1_bis.

## IV- **TP suivants (modification possible) :**

Objectif : implémenter une stratégie d'interaction mixte

15) Récupérer l'exemple `VXML Pizza` dans le dossier `Examples` disponible sous moodle.
16) Tester la grammaire `pizza.grxml`
17) Représenter l'automate associé
18) Tester l'application `pizza.vxml`
19) Représenter le modèle de dialogue associé
20) 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 Embiez de l'AIP pour le PGE le lundi
premier octobre à 10h

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