Automate de tests VOIP V. béta

Desription logicielle

Résumé :
Ce document donne une description non détaillée de l'architecture logicielle de l'automate de tests VOIP version béta. Il est destiné au programmeur pour pouvoir avoir une vue d'ensemble du logiciel avant une éventuelle modification du code source.

Sommaire

1.GENERALITES SUR L'AUTOMATE DE TESTS VOIP....3

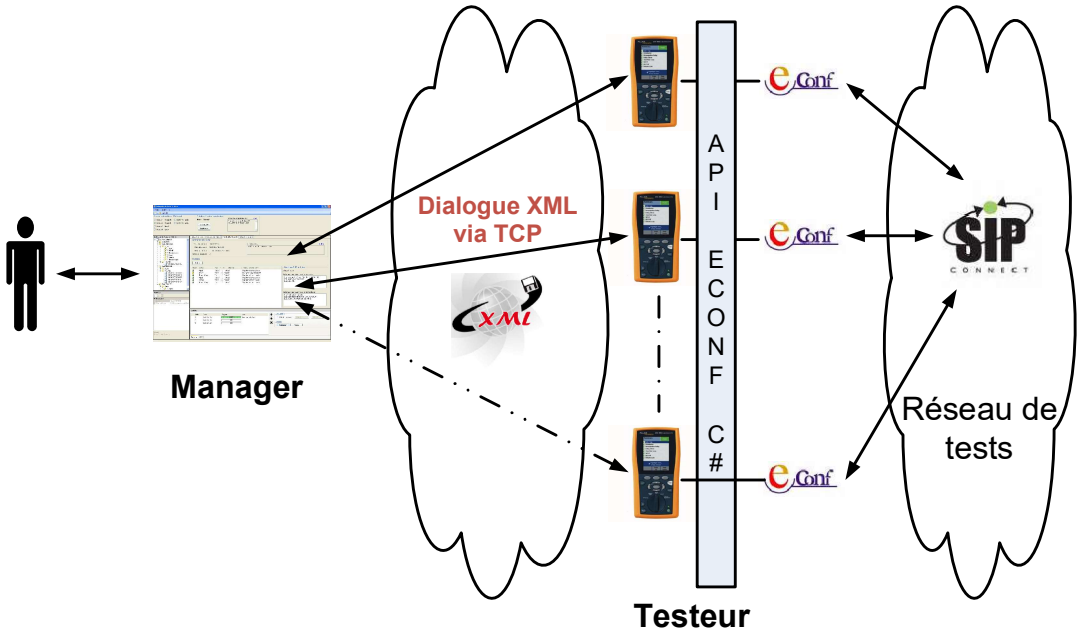
2.DEVELOPPEMENT DE L'APPLICATION....3

2.1.LE PROJET COMMUN....4
2.2.LE PROJET MANAGER....4
2.3.LE PROJET TESTER....6

3.LE DIALOGUE ENTRE LE MANAGER ET LES TESTEURS....6

1. Généralités sur l'automate de tests VOIP

Ce logiciel, comprenant une partie "Manager" et une partie "Testeur", est basé sur un model Client / Serveur TCP. Le schéma ci-dessous décrit l'architecture fonctionnelle du logiciel :



Sur chaque testeur, un serveur TCP est hébergé. Le manager est vu comme client d'un point de vue du réseau Ethernet, il déploie donc autant de clients TCP qu'il n'y aura de testeurs à connecter.

La version d'Econf pilotée par chaque testeur est la v5.0.16.

2. Développement de l'application

L'intégralité de l'application a été développée sur Microsoft Visual Studio 2005 sous licence étudiante (licence non disponible après mon départ) en utilisant le langage C#.

Il y a possibilité d'utiliser une version gratuite du logiciel, appelé Visual C# Express. Cette version peut très bien convenir pour un futur développement du logiciel, la version payante comportant quelques outils d'aide en plus et la visualisation UML entre autre.

L'application comporte quatre projets, dont deux rattachés aux autres projets. Les deux projets que vous serez amenés à charger sont le projet "Manager" et "Tester". Le projet commun est rattaché avec les deux projets principaux. Enfin le projet XPTable est un ensemble de sources récupérées sur internet servant pour l'implantation d'un élément graphique bien spécifique. Ce projet est rattaché seulement au projet Manager.

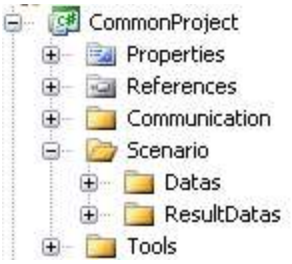
L'intégralité du code sources est présente sur le cd dans l'archive *sources.rar* (un répertoire par projet).

2.1. Le projet commun

Pour simplifier le développement, un projet commun utilisé par le projet Manager et Tester a été créé. Celui-ci regroupe toutes les données communes entres ces deux entités. Ces données sont dites génériques car elles peuvent être redéfinies.

Les données communes sont organisées en 3 catégories :

- **Communication** : contient les classes génériques pour la communication entre testeur et manager.
- **Scenario** : Données génériques concernant exclusivement la définition d'un scenario et d'un résultat.
- **Tools** : Outils communs.



Ce projet crée une librairie dll, nommée "CommonProject.dll" qui sera obligatoirement présent dans le répertoire d'exécution du manager et des testeurs. Attention à bien mettre à jours cette librairie après chaque modification de code! De plus, vérifiez que le projet Manager et Tester compile bien avec ce nouveau projet commun avant l'exécution.

2.2. Le projet Manager

Pour concevoir la partie Manager du logiciel, un projet séparé du testeur a été créé. Ce projet est dépendant de deux autres projets; le projet commun appelé "CommonProject" et le projet "XPTable" contenant un composant graphique utilisé par le Manager.

Lors de son chargement dans Visual Studio 2005 ou Visual C# Express, l'ensemble des projets seront ouvert, soit 3 projets au total. Le projet Manager est bien entendu le projet principal, la compilation et la génération de l'exécutable se fera pour le Manager.

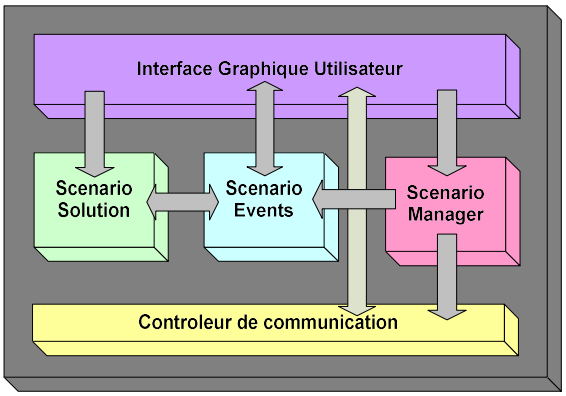
Le projet Manager est divisé en quatre branches:

- **Communication** : Contient les méthodes pour la communication entre le manager et les testeurs.
- **IHM** : Contient toutes les éléments de l'interface graphique.
- **Ressources** : Rassemble la totalité des fichiers extérieurs, essentiellement images, utilisés pour l'IHM.
- **Scenario** : Objets et méthodes utilisés pour la gestion, la lecture, et le stockage des informations des scenarios.
- **Tools** : Outils propres au manager (chargement du fichier de configuration,checking scénario).

La branche Communication et Scenario contient en grande partie des objets implémentant les classes définit dans le projet commun. Dans un souci de compréhension, les noms des répertoires sont identiques entre le projet commun et le projet Manager.

Le fichier "MainEntry.cs" est le fichier de point d'entrée de l'application. Il contient toutes les définitions des variables globales de l'application.

Une description UML des classes est présent dans le répertorie "UML" du projet (nécessite Visual Studio pour les visualiser).
Sans rentrer dans les détails, la conception et le développement du manager a été réalisée de façon à respecter le plus possible l'architecture MVC (Modèle-Vue-Contrôleur), modèle de conception logiciel répandu. La figure ci-dessous résume les différents éléments du manger avec leurs interactions entre eux.

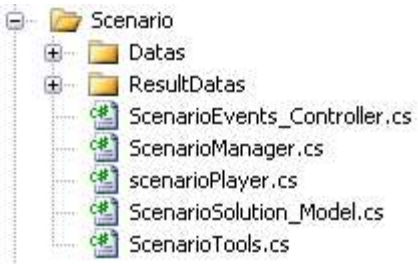


La vue correspond à la partie IHM, le modèle à l'entité "Scenario Solution", et le contrôleur à l'entité "Scenario Events". L'entité "Scenario Manager" permet la gestion de la lecture de scenario, tandis que la partie "Contrôleur de communication" correspond à l'ensemble des méthodes pour le dialogue vers les Testeurs.

Pour plus d'information sur l'architecture MVC, se référer aux liens suivant :

- http://www.zdnet.fr/builder/architecture/conception_integration_si/0,39021041,2130206,00.htm
- http://fr.wikipedia.org/wiki/Mod%C3%A8le-Vue-Contr%C3%B4leur

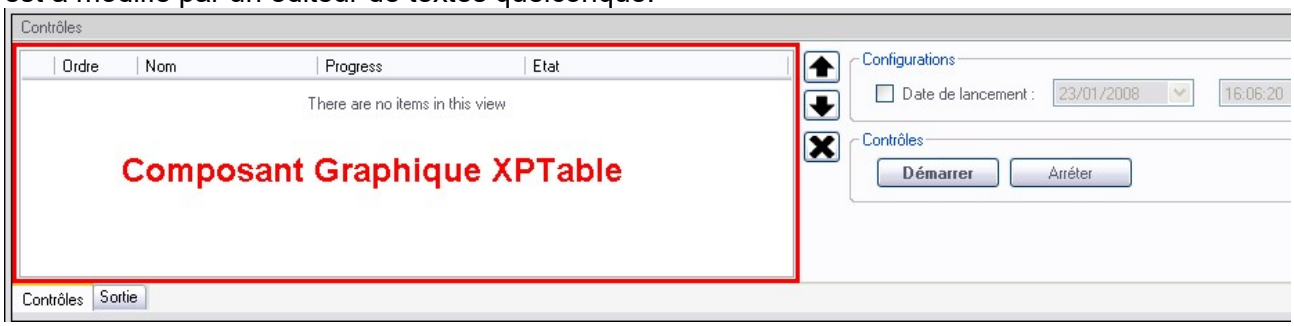
Le cœur du logiciel et les principales fonctions se situe essentiellement dans le sous-arbre "Scenario" du projet. Voici un détail des éléments le constituant:



- **Répertoires Datas** : Contient les données relatives à la définition d'un scenario. Chaque objet est la redéfinition de l'objet générique lui correspondant dans le projet commun.
- **Répertoire ResultDatas**: Contient les données relatives pour la définition du résultat d'un scenario.
- **ScenarioEvents_Controller**: Centralise tous les événements pouvant être générés dans par un scenario. Cet objet joue le rôle de contrôleur.
- **ScenarioManager**: Objet gérant la lecture d'un ensemble de scenario.
- **ScenarioPlayer**: Objet et fonctions propres à la lecture d'un scenario et à la génération du son résultat.
- **ScenarioSolution_Model**: Contient toutes les données de scenarios. Cet objet joue le rôle de model.
- **ScenarioTools** : Fonctions statiques utilisés pour traiter des données de type scenarios.

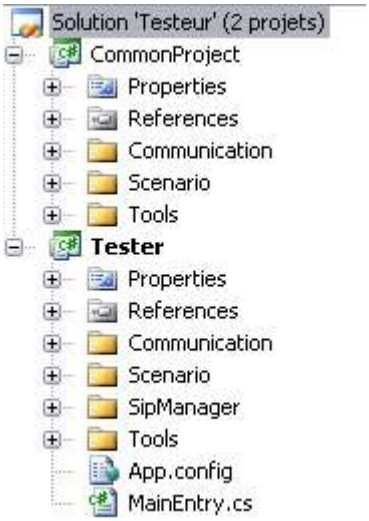
A propos du projet XPTables, il comprend un ensemble de composant graphiques concernant les listView. Comme dit-précédemment, l'IHM du manager implémente un de ces composants pour la gestion des scenarios à jouer.

Pour finir, un fichier de configuration destiné à renseigner les testeurs à contrôlé par le manager est présent dans le répertoire d'exécution du programme. Nommé "ConfigfManager.xml", ce fichier est à modifié par un éditeur de textes quelconque.



2.3. Le projet Tester

Ce projet rassemble le code source écrit pour la partie testeur de l'application. Il implémente de même le projet commun.



De plus, que ce soit pour exécuter l'application Testeur ou la modification du code source, il faut impérativement installer un Sipphone Econf (de préférence la version v.5.0.16) sur la machine en question. En effet, le projet Tester implémente l'API Econf (SDK COM) pour pouvoir faire la connexion au Sipphone. Les dll ne sont disponibles qu'après l'installation d'EConf.

L'organisation du projet Tester est similaire au projet Manager. Il n'y cependant pas tous les éléments nécessaire à une interface graphique (simple console utilisé). Le répertoire" communication" contient tous les codes sources pour le dialogue avec le Manager, le répertoire "scenario" tous les sources utiles pour la lecture d'un scenario et le répertoire "Tools" contient des fonctions statiques propres au testeur (chargement du fichier de configuration). Un dernier répertoire, nommé "SipManager" regroupe toutes les données, méthodes et objets permettant le dialogue avec le Sipphone Econf.

Le fichier "App.config" est le fichier de configuration de l'application Tester. Il contient les paramètres relatifs à un testeur (Nom, port d'écoute, adresse du manager) et est chargé systématiquement lors de l'exécution de l'application. Il sera transformé en "Tester.exe.config" lors de la génération du projet.

Le fichier "MainEntry.cs" est le fichier de point d'entrée de l'application. Il contient toutes les définitions des variables globales de l'application.

Une description UML des classes est présent dans le répertorie "UML" du projet (nécessite Visual Studio pour les visualiser).

3. Le dialogue entre le manager et les testeurs

Le dialogue entre le manager est les testeurs est uniquement XML, utilisant pour certains messages la technologie de sérialisation des classes pour transformer les objets en format XML.

Les messages transitant sont prédéfinis et respectent un format bien spécifique.

Format des messages :
<MS><Type>TypeMessage</Type><Name>NomMessage</Name><From>NomEmetteur</From><Datas>DonnéesDuMessage</Datas></MS>

Elément	Description
MS	Elément racine du message
Type	Définis le type d'informations envoyé
Name	Définis le nom du message envoyé
Source	Définis l'émetteur du message
Data	Elément contenant les informations propre au message

Les informations envoyées peuvent être de types différents :

- **ordre :** envoyé uniquement du manager aux testeurs, il ne contienne aucunes données. L'élément "Datas" du message XML est inexistant.
- **message** : information contenant des données spécifiques, transitant essentiellement du testeur au manager. L'élément "Datas" contient des données.

Pour plus d'informations, se référer à la classe *TCPDatas* contenu dans le projet commun. Cette classe définit tous les messages.