TER
Signalysis Plug-in for Orcaflex (SPO)

Maîtrise d'informatique 2003/2004

Etudiants :	**Encadrants :**
Viale Fabien	Michel Buffa Maîtres de conférences en Informatique

Résumé

Les ingénieurs travaillant dans un bureau d'études doivent effectuer tous les jours des calculs, des simulations, des estimations, qui permettent de pouvoir calibrer un produit qu'ils souhaiteraient vendre ou de réaliser une maintenance d'un produit déjà vendu.

Selon les corps de métier, les ingénieurs utilisent plus ou moins de logiciels permettant la réalisation de ces calculs. Il leur faut la plupart du temps jongler entre les logiciels pour pouvoir réaliser des transferts de données d'une application à l'autre.

L'objectif principal de ce projet est de réaliser un transfert de données entre deux applications utilisées par les ingénieurs du département d'hydrodynamique de la Single Buoy Moorings. L'outil réalisé sera ensuite utilisé tous les jours par les ingénieurs, cela doit donc être un produit fini et opérationnel.

Sommaire

Introduction

Le présent TER avait au départ une orientation bien différente de ce qui a finalement été réalisé. Le contexte du TER est le suivant : j'ai travaillé pendant un an dans un bureau d'études basé à Monaco et réalisant des designs de bouées d'amarrages, de tours de pompage, et d'autres équipements offshore.

L'ambiance de la société, le stress des projets, le cadre m'ont toujours plu. C'est pourquoi j'ai eu envie de réaliser un projet en rapport avec cette société : produire un outil pour les ingénieurs.

Au cours de mon séjour dans cette société, j'ai eu le plaisir de rencontrer M. Jean-Loup Isnard, manager du département d'hydrodynamique. Il m'avait entretenu il y a un an et demi d'un projet qu'il avait en tête et qui était la réalisation d'un outil pour centraliser les données hydrodynamiques véhiculées entre leur différents programmes.

Cet outil aurait dû permettre de modéliser des bouées, ensuite de générer un maillage de celles-ci, et enfin de pouvoir produire des fichiers d'entrée pour différents logiciels. Les données produites au fur et à mesure des différentes étapes auraient été stockées dans une base de donnée.

C'est ce projet que j'avais présenté à Olivier Dalle, le coordinateur de la maîtrise d'informatique, pour qu'il puisse faire l'objet d'un TER. Il avait pensé au départ que j'aurais pu réaliser ce projet avec un autre étudiant travaillant comme moi à temps plein : Louis Alessandra.

Après deux mois de recherche personnelles sur la question, en me documentant sur la CAO/DAO, les techniques de Maillages, après la démission de Louis, j'ai finalement discuté avec Olivier Devillers et il m'a fait comprendre que réaliser un mailleur, une des étapes du projet, étaient une tâche bien trop importante pour un simple TER. J'ai été légèrement déçu mais j'ai néanmoins essayé de voir ce que l'on pouvait faire sans réaliser de mailleur. Au fur et à mesure de mes recherches sur les logiciels utilisés par les ingénieurs, j'ai compris que je ne pourrais comprendre leur maniement et les données qu'ils véhiculaient sans les voir fonctionner, sans voir les ingénieurs travailler avec et sans leur poser des questions au jour le jour concernant leur fonctionnement. J'ai donc conclu que ce projet devait être réalisé dans les locaux de la SBM et j'ai pris contact avec eux. C'est au cours de cette prise de contact que j'ai discuté avec Christian Bauduin, un des ingénieurs travaillant dans le département, et il m'a proposé d'orienter le TER sur un autre sujet, qui selon lui seraient bien plus réalisable compte tenu du temps imparti.

Sa proposition était la suivante : réaliser une interface entre une application nommée SIGNALYSIS qu'il a développé lui-même à partir de MATLAB, et un des logiciels utilisés en hydrodynamique nommé ORCAFLEX. Le principe de cette interface serait de pouvoir importer les fichiers de simulation ORCAFLEX directement dans SIGNALYSIS, pour pouvoir faire ensuite du traitement de signal sur les données importées grâce aux fonctionnalités de SIGNALYSIS.
L'idéal serait de réaliser une interface graphique qui permettrait de pouvoir visualiser tous les signaux d'un fichier de simulation et ensuite de sélectionner ceux que l'on souhaite.

J'ai trouvé le sujet à la fois intéressant et surtout réalisable et j'ai fait part de la modification du sujet à Olivier Dalle et à Michel Buffa, qui n'ont émis aucun désaccord.

Dans ce rapport, je rappellerai les points essentiels du cahier des charges, puis je décrirai en détail le travail que j'ai réalisé. J'évoquerai ensuite les améliorations de l'outil qui sont envisageables, le déroulement du projet et les planning prévisionnels et effectifs. Finalement, je conclurai sur l'expérience et l'enrichissement personnel acquis lors de la réalisation du projet.

1 Cahier des charges

1.0 Remarques

Il n'a pas été possible de rédiger un cahier des charges précis et détaillé au début du projet pour deux raisons majeures (il a seulement été établi au début dans les grandes lignes) :

- La nature même du travaille à réaliser demande une connaissance approfondie du logiciel ORCAFLEX et de MATLAB. Il a donc fallu tout d'abord comprendre ces logiciels avant de pouvoir poser les questions nécessaires à l'élaboration d'un cahier des charges. La compréhension de ces logiciels demandait de commencer à faire du développement, la phase de développement a donc commencé avant la fin de l'élaboration du cahier des charges.
- La personne responsable du projet au sein de la Single Buoy Moorings est partie pendant les trois semaines qui ont été la phase principales du développement, au moment où j'étais prêt à rédiger le cahier des charges. J'ai dû me baser sur l'avis d'autres ingénieurs pour rédiger mon cahier des charges.

1.1 Fournitures

Je dois rendre un programme opérationnel et intégré aux ordinateurs du département d'hydrodynamique de la Single Buoy Moorings. Un manuel de maintenance pourrait également être utile pour de futurs développements.

1.2 Organisation du projet

1.2.1 Processus

Le modèle de développement utilisé est la modélisation en cascade :

- Analyse : étude de l'existant, étude de la *spreadsheet* Excel déjà réalisée par ORCINA, la société éditrice d'ORCAFLEX et effectuant une opération similaire.
- Conception : étude des possibilités d'appel de la DLL ORCAFLEX, choix d'un procédé permettant de récupérer les données sous MATLAB, choix du(des) langage(s) de programmation(s) utilisé(s), choix du langage utilisé pour réaliser l'interface graphique, découpage du programme en tâches simples...
- Programmation : programmation des routines d'importations de données, programmation de l'interface graphique, lien entre les modules.
- Tests : installation du programme sur les postes et test effectué par les ingénieurs.
- Maintenance : réalisation d'un manuel de maintenance expliquant l'implémentation, les structures de données, le découpage en tâches simples.

1.2.2 Limites et interfaces

Limites

- Le programme est limité par les possibilités de la DLL OrcFxApi du logiciel ORCAFLEX. On ne peut pas avoir accès au données d'un fichier de simulation autrement. Néanmoins, cette DLL a de très grandes possibilités et est très fonctionnelle.
- Le programme ne peut être utilisé que depuis MATLAB. Il n'est pas possible de l'utiliser comme exécutable ou de l'appeler depuis un autre logiciel. Néanmoins, le principe même du programme est d'être appelé depuis MATLAB et il ne semble pas utile de réaliser d'autres interfaces pour l'instant.

Interfaces

Deux interfaces sont à prévoir au sein du projet. L'une est avec le logiciel ORCAFLEX, à travers sa DLL. L'autre interface est avec l'environnement de MATLAB.

1.3 Gestion

1.3.1 Objectifs et priorités

Objectifs

Je dois fournir une application conviviale et rapide à utiliser. Le but du projet est de remplacer une application développée par ORCINA sous la forme d'une *spreadsheet* EXCEL. Cette application est un peu fastidieuse d'utilisation et oblige l'ingénieur à faire de multiples transferts de formats de fichiers à d'autres. Mon application, doit permettre d'apporter un réel progrès en matière de facilité d'utilisation, tout en intégrant les possibilités d'extraction de données, voire d'automation, que la spreadsheet contient. L'application doit s'intégrer parmi les routines d'importation de SIGNALYSIS (voir ci-dessous).

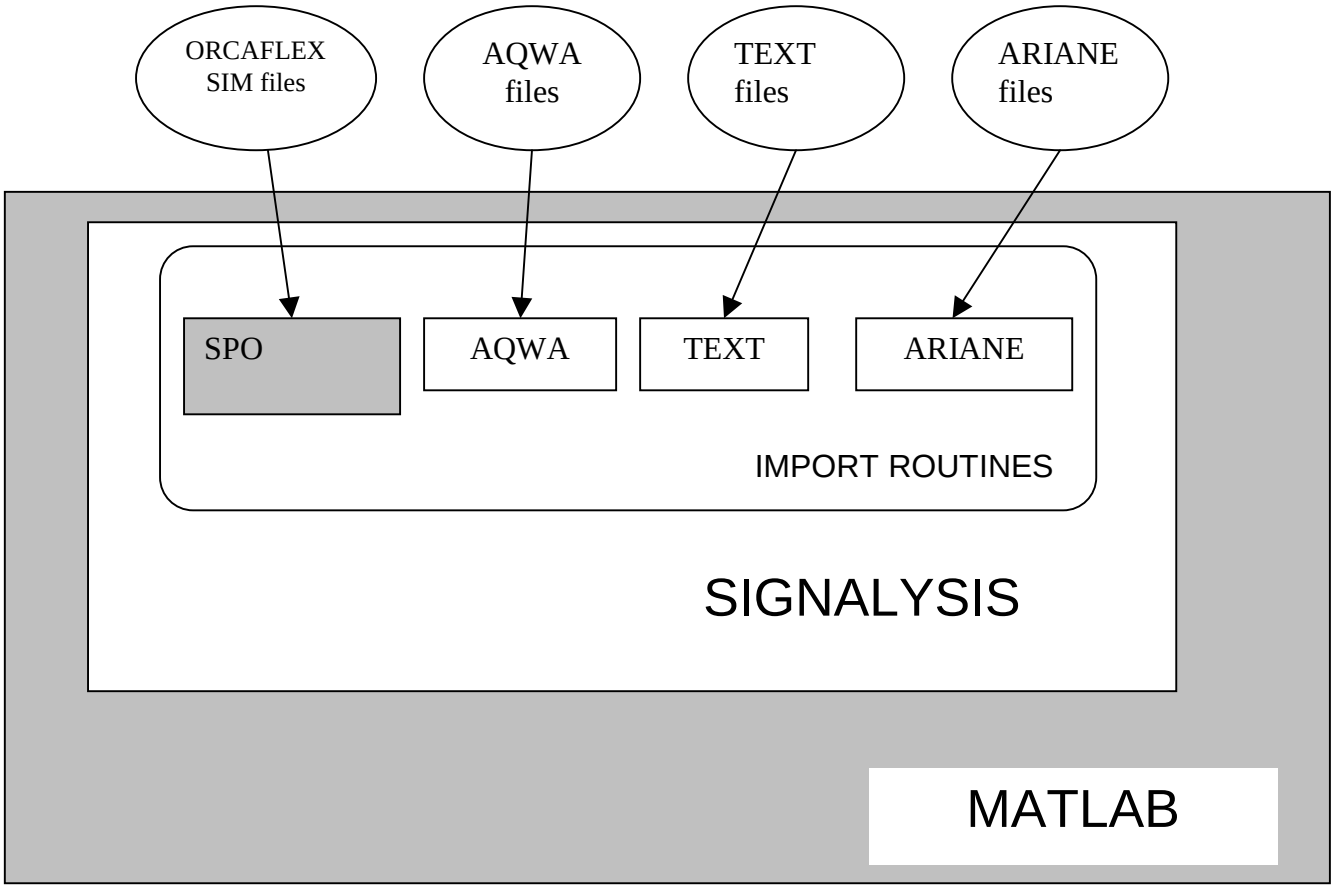


Figure 1 – Intégration d'SPO à Signalysis

Priorités

La tâche prioritaire est le transfert de donnée d'ORCAFLEX à MATLAB. On doit commencer par transférer toutes les données d'une simulation. Ensuite, on peut envisager de ne sélectionner que certaines données.

1.3.2 Hypothèses, dépendances, contraintes

Hypothèses

Les données d'une simulation sont regroupées en objets. Chaque objet représente un élément du système (vaisseau, ligne d'ancrage). Ces objets sont organisés en catégories, on trouve les objets environnement, vaisseaux, bouées, lignes d'ancrage, « winch » , etc …

Selon le type de l'objet, certaines variables sont accessibles et d'autres non. La disponibilité des variables dépend en plus de la position choisie sur l'objet. Ainsi, certaines variables d'une ligne d'ancrage ne sont accessibles qu'au début et à la fin de la ligne par exemple. L'utilisateur doit donc tout d'abord choisir un objet, ensuite une position sur cet objet, et finalement il peut choisir tel ou tel variable à extraire.

L‘intervalle de temps choisi pour importer les données est supposé unique pour toutes les variables à importer. On ne peut pas importer un intervalle de temps pour une variable et un autre intervalle de temps pour une autre variable.

Cette organisation d’objets n’est pas supposée évoluer avec les prochaines versions d’ORCAFLEX (du moins dans les grandes lignes).

Contraintes

- Le code doit utiliser les fonctions de la DLL ORCAFLEX, seul moyen disponible d’extraire les données d’un fichier de simulation.
- Les données à intégrer dans MATLAB doivent être sauvegardées dans les structures utilisées par l’application SIGNALYSIS.
- L’application doit fonctionner sous Windows 2000/XP. Aucune utilisation dans d’autres systèmes d’exploitation n’est à prévoir.
- Le code écrit doit être suffisamment bien commenté pour pouvoir être facilement réutilisé et modifié dans le cas de changements dans la DLL ORCAFLEX.

1.3.3 Gestion du risque

Trois risques majeurs sont à craindre pour l’application :

- L’application devant permettre à MATLAB et ORCAFLEX s’exécutant en parallèle de communiquer, des conflits d’utilisation de ressources systèmes peuvent survenir. L’application doit gérer de la manière la plus stable possible les transfert de données entre les deux applications (en veillant que des ressources ne sont pas partagées par les deux applications).

- Des développements ultérieurs d’ORCAFLEX peuvent modifier les fonctions de la DLL, rendant, mon application inutilisable si une compatibilité descendante n’est pas prévue lors des modifications. Dans ce cas l’application devient inutilisable tant que le code n’est pas mis à jour pour prendre en compte ces modifications. Ce risque inquiétant ne peut être contourné qu’en commentant correctement le code et en fournissant une notice explicative du programme.

- L’application doit fonctionner sous Windows 2000/XP. Aucune utilisation dans d’autres systèmes d’exploitation n’est à prévoir.

1.3.4 Moyens de contrôle

- Communication avec les ingénieurs de la Single Buoy Moorings.
- Echange d’informations avec les encadrants, mais la présence des vacances scolaire pendant la phase de programmation rend la communication difficile.
- Procédures de tests : tests avec les ingénieurs.

1.4 Fonctions du produit

1.4.1 Ouverture d'un fichier de simulation

- Le programme permet d'ouvrir un nouveau fichier de simulation pour extraire des signaux.
- Le programme permet d'ajouter des signaux à un fichier déjà extrait (à faire).

1.4.2 Sélection des signaux

- Le programme affiche la liste des objets présents dans le fichier de simulation.
- Il affiche pour chaque objet une brève description.
- L'utilisateur peut sélectionner les signaux à importer en choisissant une position spécifique sur un objet et en choisissant une ou plusieurs variables dans la liste des variables de l'objet.

1.4.3 Transfert vers Signalysis

- L'utilisateur choisit un nom de fichier de destination et clique sur le bouton terminer.
- Le programme enregistre les signaux choisis dans un fichier au format Signalysis.
- Le programme termine son exécution et retourne dans Signalysis, les signaux enregistrés se retrouvent disponibles dans Signalysis.

1.4.4 Routines accessibles depuis la ligne de commande de MATLAB

- Ces routines permettent d'appeler la DLL ORCAFLEX par des commandes MATLAB. Elles permettent à de futurs développements d'être réalisés en réutilisant ces commandes.
- Elles sont composées de cinq routines : loadsimulation, importobjectlist, importvariablelist, importtimehistory, destroymodel.
- La première et la dernière permettent de charger le modèle en mémoire et de le détruire.
- La deuxième permet d'obtenir la liste des objets du modèle, la troisième la liste des variables pour un objet, et la dernière, d'importer un signal.

1.5 Architecture logicielle

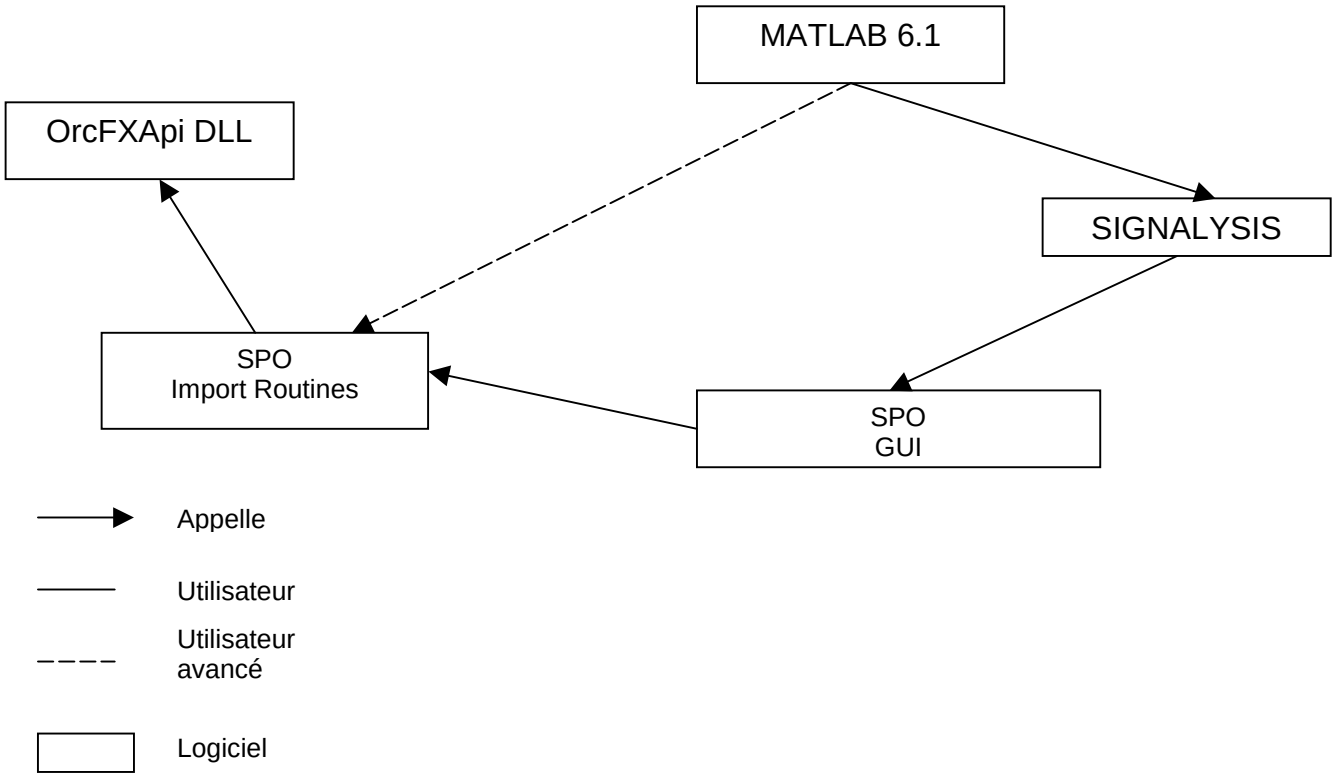


Figure 2 – Architecture Logicielle

2 Descriptif du travail réalisé

2.0 Introduction

Le descriptif du travail réalisé suit d'une manière fidèle les étapes que j'ai dû suivre pour réaliser mon projet. J'ai eu la chance de pouvoir réaliser chaque étape l'une après l'autre, sans avoir à commencer différentes parties de mon application en même temps. C'est pour cela je pense que ce projet était réellement adapté à un travail individuel.

Même si j'ai dû parfois revenir en arrière pour corriger certaines erreurs, les étapes de développement ont été bien respectées.

J'ai dû dans un premier temps comprendre le fonctionnement de cette DLL, voir quelles étaient ses possibilités. Ensuite j'ai dû déterminer comment importer des données extérieures dans MATLAB.

Il a fallu ensuite comprendre comment étaient structurées les fichiers de simulation du logiciel ORCAFLEX. Comme les données sont des données relatives à des simulations d'expériences physiques, avec un vocabulaire très spécifiques au corps de métier, elles sont inaccessibles à un non-initié, de même que les relations entre ces données permettant de maintenir la cohérence de l'ensemble. Il a donc été nécessaire de bien comprendre les données manipulées et de poser tout au long du développement toutes les questions nécessaires aux ingénieurs.

Il a fallu enfin réaliser une interface graphique conviviale et intuitive, permettant par des manipulations simples d'avoir toutes les données que l'on souhaite.

Enfin il a fallu exporter les données de mon application dans un fichier Signalysis.

J'ai découpé mon application en deux composantes majeurs. D'une part, un ensemble de cinq commandes MATLAB permettant de contrôler la DLL directement depuis la ligne de commande de MATLAB et pouvant permettre de futurs développements. D'autres part, l'interface graphique utilisateurs, qui ne peut pas fonctionner sans le premier composant.

2.1 La DLL d'ORCAFLEX

La DLL est fournie avec une interface sous la forme d'un header-file en C. Appeler la DLL depuis un programme écrit en C était donc un choix naturel.

Même si elle compte un grand nombre de fonctions (ses possibilités dépassent en effet le cadre de ce TER). Seules quelques-unes seront utilisées dans ce projet.

ORCAFLEX est un logiciel orienté-objet. Pour représenter un système flottant il crée un objet **model** qui est simplement une structure de données dans laquelle les simulations d'ORCAFLEX et leurs résultats peuvent être chargés. Dans l'objet model, d'autres objets sont ensuite créés pour représenter les différentes parties du système flottant.

La première chose que l'on doit faire en utilisant la DLL est d'appeler la fonction C_CreateModel pour créer l'objet model. Cette fonction crée le modèle et retourne la référence du modèle (model handle).

On peut ensuite charger une simulation dans le modèle en utilisant la fonction C_LoadSimulation. Cela charge dans le modèle tous les objets inclus dans la simulation. Cela charge également tous les résultats de la simulation en mémoire, prêts à être exportés.

Lorsqu'on utilise les fonctions de la DLL, on a besoin de la référence du modèle, mais aussi de la référence des autres objets présents dans le modèle. Car en appelant une fonction de la DLL on doit lui signaler à quel objet du modèle on se réfère. Ces informations peuvent être obtenues en appelant C_EnumerateObjects. On doit établir notre propre structure de donnée dans laquelle stocker toutes les informations recueillies. La fonction C_EnumerateObjects appelle pour chaque objet, une fonction qui lui est fournie en paramètre et dont le prototype est défini dans l'API par EnumerateObjectsProc. C'est cette fonction qui est chargée d'enregistrer les informations sur chaque objet.

Les informations récupérées par défaut par EnumerateObjectsProc ne sont que très sommaires, elles ne permettent d'obtenir qu'une référence à l'objet ainsi que son type.

Pour obtenir des informations supplémentaires on doit utiliser une des fonctions suivantes :

- C_GetDataDouble
- C_GetDataInteger
- C_GetDataString

Ces fonctions permettent d'obtenir n'importe quelle information concernant le modèle. On donne une référence à l'objet en rapport avec l'information et le code de l'information sous forme d'une chaîne de caractères, et on obtient en retour l'information voulue selon que le type de l'information est un Double, un Integer ou une String.

Cela permet d'avoir une connaissance approfondie du modèle, savoir par exemple combien de segments possède une ligne, quel est le temps de début et de fin de la simulation, quel est le pas de temps de la simulation. On peut ainsi construire, en récupérant les informations souhaitées au moment de l'appel de EnumerateObjectsProc, une hiérarchie d'objet plus complète et avoir dans notre application toutes les informations indispensables sur le modèle.

On a ensuite besoin de connaître, pour chaque objet donné, la liste des variables associées avec cet objet. Cette liste de variables s'obtient avec la fonction C_EnumerateVars2. Cependant, la donnée seule d'un objet n'est pas suffisante pour obtenir cette liste de variables, on doit en plus spécifier à partir de quelle position sur l'objet on veut obtenir cette liste. En effet, certaines variables ne sont accessibles qu'à des endroits bien précis d'un objet.

Pour pouvoir connaître la plage des valeurs possibles des positions sur un objet, les fonctions C_GetDataDouble, C_GetDataInteger ou C_GetDataString décrites précédemment sont nécessaires.

Ayant chargé une simulation, et ayant établi notre liste d'objets dans le modèle, on peut extraire des résultats de la simulation. Cela peut être fait en appelant C_GetNumOfSamples, C_GetSampleTimes et C_GetTimeHistory2.

- C_GetNumOfSamples renseigne sur le nombre d'échantillons qu'il y a dans les enregistrements (elles ont toutes le même nombre d'échantillons). Cela permet d'allouer suffisamment de mémoire pour stocker les enregistrement que l'on veut extraire.
- C_GetSampleTimes retourne le tableau des temps des échantillons qui s'applique à tous les enregistrements.
- C_GetTimeHistory2 retourne le tableau des valeurs des échantillons pour un enregistrement donné.

Finalement, quand on a fini d'extraire les résultats, on doit appeler C_DestroyModel, cela détruit le modèle que l'on a créé et tous les objets qu'il contient.

Malgré le nombre important de fonctions dans l'API, je n'ai utilisé finalement qu'une douzaine de fonctions. Le code n'en sera que plus facile à maintenir.

2.2 Les fichiers MEX de MATLAB

Rappels sur MATLAB

MATLAB est un logiciel de calcul matriciel à syntaxe simple.

Avec ses fonctions spécialisées, MATLAB peut être aussi considéré comme un langage de programmation adapté pour les problèmes scientifiques.

MATLAB est un interpréteur: les instructions sont interprétées et exécutées ligne par ligne.

MATLAB fonctionne dans plusieurs environnements tels que X-Windows, Windows, Macintosh.

Il existe deux modes de fonctionnement:

- mode interactif: MATLAB exécute les instructions au fur et à mesure qu'elles sont données par l'usager.
- mode exécutif: MATLAB exécute ligne par ligne un "fichier M" (programme en langage MATLAB).

Définition des fichiers MEX

MEX signifie MATLAB Exécutable. Les fichiers MEX sont des sous-routines liées dynamiquement à partir d'un code source en C ou Fortran qui, une fois compilées, peuvent être lancées à partir de la ligne de commande de MATLAB de la même manière que les fichiers M de MATLAB.

Les fonctions de l'interface externe de MATLAB(accessibles depuis le code C ou Fortran à la manière de fonctions d'une librairie) proposent des fonctionnalités pour transférer des données entre les fichiers MEX et MATLAB et pour appeler des fonctions MATLAB depuis le code C ou Fortran.

Pourquoi utiliser des fichiers MEX dans mon application ?

Les fichiers MEX sont des programmes écrits en C ou Fortran (dans mon cas ce sera du C). Ils ont accès aux mêmes possibilités qu'un programme écrit en C. Ainsi, on peut appeler les fonctions d'une autre librairie depuis un fichiers MEX et le lier à une DLL.

Interface MATLAB vers les DLL génériques

Avant d'utiliser les fichiers MEX, j'avais pensé utiliser l'interface MATLAB vers les DLL génériques. Cette interface n'est accessible qu'à partir de la version 6.5.1 de MATLAB et permet d'appeler des fonctions d'une DLL directement depuis MATLAB.

J'ai dû laisser tomber cette idée après plusieurs requêtes au support technique de MATLAB.
En effet, cette interface ne permet pas d'utiliser ce qu'on appelle les callbacks-fonctions, c'est à dire des fonctions passées en paramètres. Et la DLL ORCAFLEX nécessite l'utilisation de ces callbacks-fonctions avec la fonction EnumerateObjectsProc(voir ci-dessus) lors de l'énumération des objets du modèle.

Programmation des fichiers MEX en C – la structure de données MATLAB array

Le langage MATLAB fonctionne avec seulement un type d'objet : le MATLAB array.
Toutes les variables MATLAB, incluant les type scalaires, les vecteurs, les matrices, les chaînes de caractères, les cellules, les structures et les types objets sont stockés comme des MATLAB array. En C, le MATLAB array est déclaré être du type mxArray. La structure mxArray contient, en autres choses :

- son type
- ses dimensions
- les données associées avec ce tableau
- s'il est numérique, les données sont-elles réelles ou complexes ?
- Si c'est une structure ou un objet, le nombre de champs et les noms de ces champs.

Toutes les transferts de données passées à MATLAB par un fichier MEX se feront donc par l'intermédiaire de cette structure. Cette uniformisation de l'information rend le code bien plus compréhensible.

Compilation des fichiers MEX

MATLAB inclut un compilateur C pour Windows appelé Lcc.

Les premiers tests avec ce compilateur ont montré qu'il ne pouvait pas être utilisé dans mon projet.

La raison en est la suivante : la DLL ORCAFLEX utilise une syntaxe propre à la programmation sous Windows. Le fichier .h de la DLL utilise des directives comme _declspec que le compilateur Lcc ne comprend pas (du moins dans la version de Lcc fournie avec MATLAB).

J'ai donc dû utiliser un autre compilateur que Lcc appelé MinGW, qui est un compilateur quasi identique à gcc et fonctionnant sous Windows.

Pour pouvoir compiler mes fichiers MEX avec MinGW. J'ai dû employer un petit utilitaire écrit en langage MATLAB s'appelant Gnumex permettant de configurer MATLAB pour l'utilisation de MinGW.

2.3 La programmation objet sous MATLAB

La programmation objet sous MATLAB
La programmation orientée-objet sous MATLAB a les fonctionnalités classiques du paradigme objet :

- **Surcharge des opérateurs et des fonctions :** il est possible de redéfinir les opérateurs de base ainsi que les fonctions MATLAB pour prendre en charge une classe d'objet utilisateur. MATLAB cherche d'abord s'il y a une méthode définie pour la classe d'un objet avant de faire appel à la bibliothèque des fonctions standard.
- **Encapsulation des données et des méthodes :** les propriétés des objets ne sont pas visibles à partir de la ligne de commande. On ne peut accéder aux propriétés des objets que par les méthodes.

- **Héritage :** on peut créer des hiérarchies de classe de classes parents et enfants dans laquelle la classe enfant hérite des champs de données et des méthodes du parent.
- **Agrégation :** on peut créer des classes dans lesquelles les objets contiennent d'autres objets.

Les méthodes des classes, tout comme leur constructeur, sont des fichiers M situés dans un répertoire nommé @nom_de_la_classe accessible depuis la ligne de commande (i.e. situé dans le *path* de MATLAB). La programmation des méthodes est exactement la même que celle des fichiers M.

Pourquoi utiliser la programmation objet ?

Il est toujours possible d'éviter d'utiliser la programmation objet dans un programme. Cependant, lorsque des critères comme la réutilisabilité sont importants, lorsque les données à manipuler ont une organisation qui s'apparente à des objets, la programmation objet permet une meilleure lisibilité et organisation du code.

La structure même du modèle d'un fichier de simulation ORCAFLEX s'apparente à une organisation en objets. La documentation de l'API souligne d'ailleurs bien ce point. Il était donc tout naturel de reprendre dans mon programme une structure en objets.

Cette section et les deux suivantes décrivent la structure de l'application :

2.4 Les routines d'importation de SPO

Ces routines sont des fichiers MEX. On peut donc les appeler depuis la ligne de commande de MATLAB. Elles sont au nombre de cinq, chacune effectuant une tâche simple auprès de la DLL ORCAFLEX.

LoadSimulation :

Cette routine permet de charger un fichier de simulation en mémoire. Elle retourne une référence au modèle créé. La routine se résume à l'utilisation successive de C_CreateModel et de C_LoadSimulation.

ImportObjectList :

Cette routine permet, à partir d'une référence à un modèle de connaître la liste des objets présent dans le modèle. La routine retourne la liste d'objets du modèle décrits sous la forme d'objets MATLAB. La hiérarchie précise de classes utilisée est décrite dans la section suivante. La routine correspond à l'utilisation de C_EnumerateObjects.

ImportVariableList :

Cette routine permet, à partir de la référence d'un objet et d'informations relatives à la position sur l'objet (cf. section suivante) d'obtenir la liste des variables disponibles.
La routine correspond à l'utilisation de C_EnumerateVars2.

ImportTimeHistory :

Cette routine permet, à partir de la référence d'un objet, d'un identificateur de variables, d'informations relatives à la position sur l'objet , et d'informations sur l'intervalle de temps souhaitée (la période), d'obtenir le tableau des temps des échantillons, et des valeurs des échantillons. La routine correspond à l'utilisation de C_GetNumOfSamples, C_GetSampleTimes et C_GetTimeHistory2.

DestroyModel :

Cette routine permet, à partir de la référence d'un modèle de détruire celui-ci et tous les objets qu'il contient de la mémoire.

Conclusion :

La douzaine de fonctions de la DLL ORCAFLEX sont donc résumés, au niveau de la ligne de commande de MATLAB, à seulement cinq fonctions. La routine-clé est ImportObjetList car celle est qui récupère toutes les informations nécessaires au niveau du modèle. Je l'ai programmée de telle manière à obtenir les informations indispensables au développement de l'application. Mais elle pourrait être étendue dans le futur pour obtenir toutes les informations ou presque sur un modèle.

2.5 La hiérarchie d'objets de SPO

Voici un schéma de la hiérarchie d'objets MATLAB de l'application, avec les conventions de la notation UML. Les types des variables sont les types MATLAB:

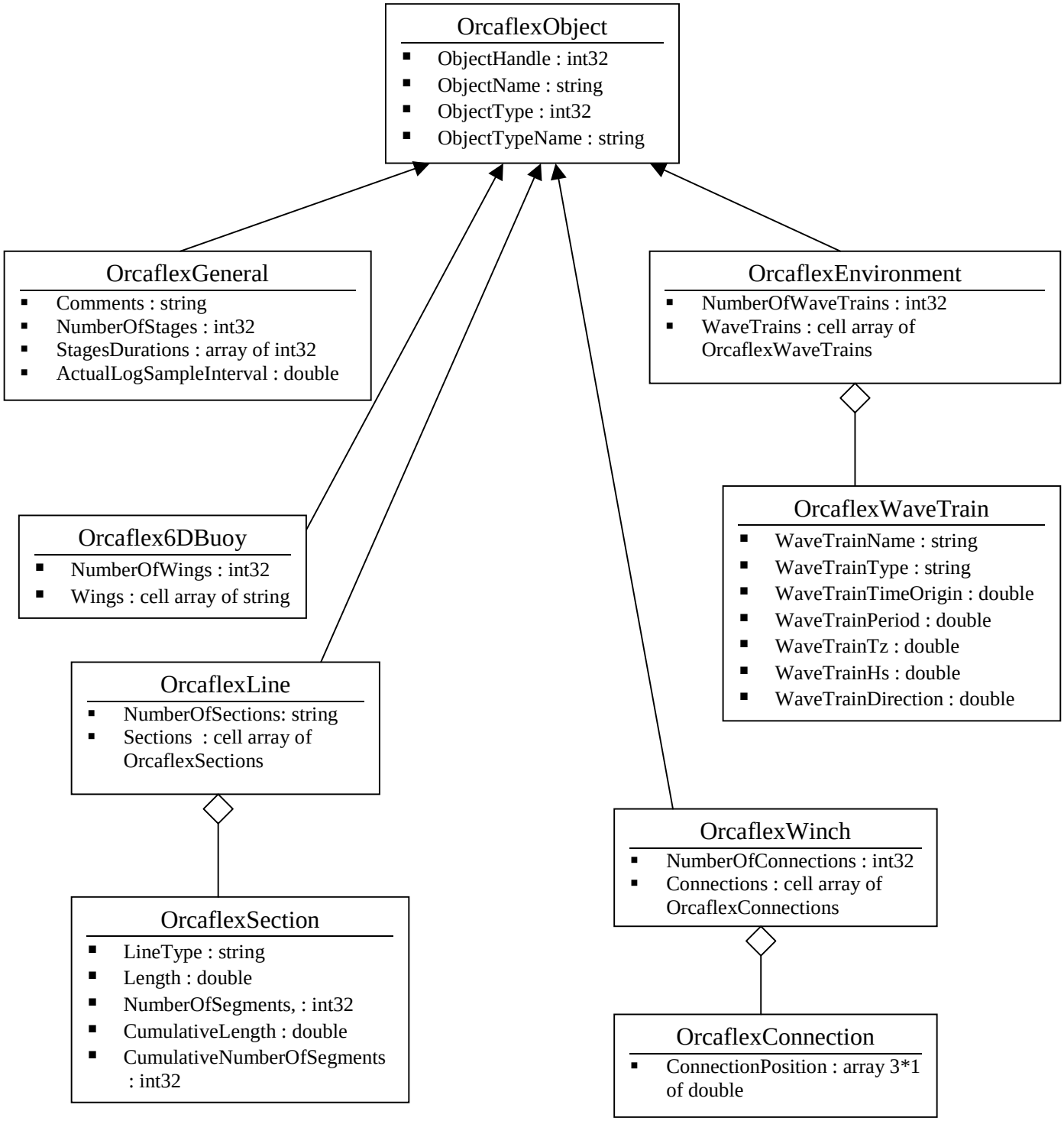


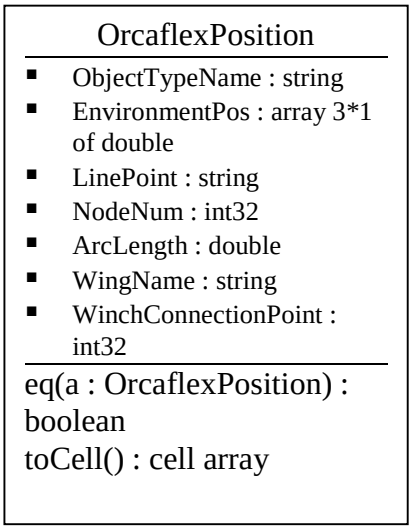
Figure 3 - Hierarchie d'objets de SPO

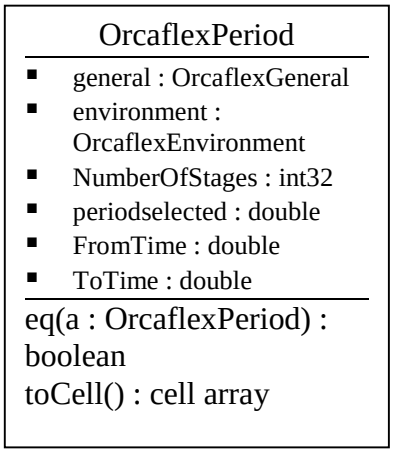
ImportObjectList construit une liste d'objets MATLAB qui sont membres de la classe OrcaflexObject ou une de ses sous-classes.

Chacune de ces classes définit les méthodes suivantes :

constructeur : permet de créer un objet de ce type, et est identifié par le nom de la classe.
get : permet d'obtenir un des attributs de l'objet.
set : permet de modifier un des attributs de l'objet.
subsref : redéfinit l'opérateur « . » pour la syntaxe du type « toto = mon_objet.mon_attribut »
subsasgn : redéfinit l'opérateur « . » pour la syntaxe du type « mon_objet.mon_attribut = toto »
toString : retourne une chaîne de caractères décrivant l'objet.

En plus de cette structure d'objets principale, deux classes supplémentaires sont utilisées pour mon application, il s'agit des classes OrcaflexPosition et OrcaflexPeriod :





OrcaflexPosition définit la position sur un objet en fonction de son type.

OrcaflexPeriod définit un intervalle de temps pour une simulation donnée. Il contient également la liste de tous les intervalles de temps possibles pour une simulation donnée.

2.6 L'interface Graphique(GUI) de SPO

SIGNALYSIS, qui selon le cahier des charges doit fonctionner avec mon application est une application entièrement développée sous MATLAB, grâce aux possibilités de création d'interface graphique de MATLAB. Il était donc tout naturel que je réalise mon interface graphique également avec MATLAB.

2.6.1 La création d'une interface graphique avec MATLAB

MATLAB permet de développer des applications grâce à un environnement de développement appelé GUIDE. Cet environnement permet de dessiner des interfaces graphiques et de sauvegarder un fichier FIG contenant la disposition de l'interface. Il ne reste plus ensuite qu'à programmer un fichier M qui va gérer cette interface graphique. Ce fichier M contient généralement tout le code nécessaire à l'interface graphique. Néanmoins, il est possible de partager ce code entre plusieurs fichiers M, ainsi que de gérer par un seul fichier M plusieurs interfaces graphiques. Cela permet d'avoir une grande flexibilité pour développer l'interface graphique.

La structure de donnée utilisée par l'interface graphique est une structure appelée « handles » contenant toutes les informations concernant l'interface graphique (i.e. les différents composants de l'interface graphique). Grâce à cette structure, on peut modifier l'aspect de chaque composant de l'interface graphique et même rajouter au supprimer des composants.

Les actions de l'utilisateur sont enregistrées par des fonctions « Callbacks » définies dans le fichier M de l'interface graphique (mais qui peuvent aussi être définies dans un autre fichier M). Ces fonctions permettent à l'interface graphique de répondre aux actions de l'utilisateur.

Le programmeur peut aussi ajouter n'importe quelle information à la structure « handles ». On peut ainsi par exemple enregistrer dans la structure un champ contenant le nombre de cases cochées, etc.…

Cela permet d'utiliser la structure comme un moyen de communication très efficace entre les différents « callbacks », chaque fonction « callback » pouvant obtenir, par le biais de la structure « handles », toute l'information gérée par l'interface graphique.

Voici un petit schéma décrivant la gestion d'une interface graphique sous MATLAB :

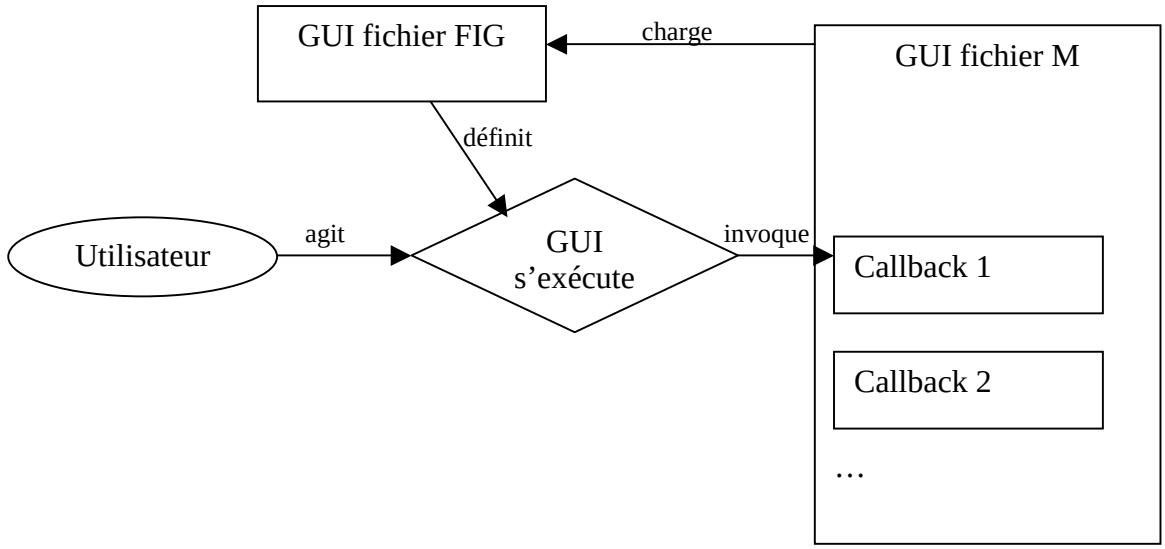


Figure 4 - gestion d'une interface graphique sous MATLAB

2.6.2 L'interface graphique de SPO

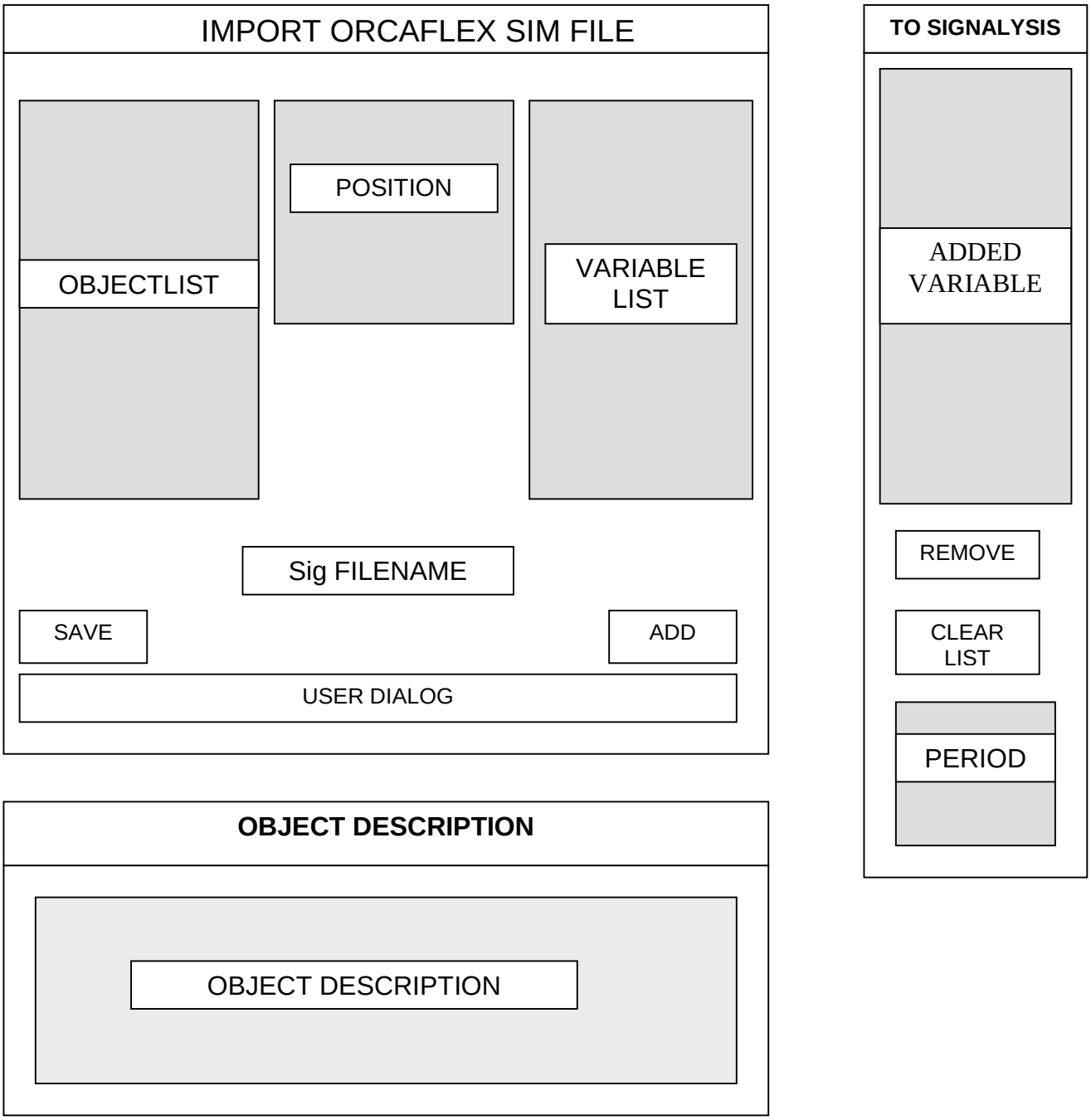


Figure 5 - Interface Graphique de SPO

L'interface graphique est divisée en trois fenêtres. La fenêtre principale permet à l'utilisateur de naviguer dans le modèle. En sélectionnant un objet dans « objectlist », la zone position se modifie selon le type de l'objet sélectionné pour pouvoir choisir la position sur l'objet. La liste des variables disponibles pour cet objet et cette position apparaît dans « variablelist ».

La fenêtre « Object Description » affiche les informations concernant l'objet sélectionné.

La fenêtre « To Signalysis » montre la liste des variables choisies pour être importée dans Signalysis et permet de choisir l'intervalle de temps (period) de l'importation. Le bouton « Add » de la fenêtre principale permet d'ajouter des variables dans cette liste et les boutons « Remove » ou « ClearList » d'en enlever.

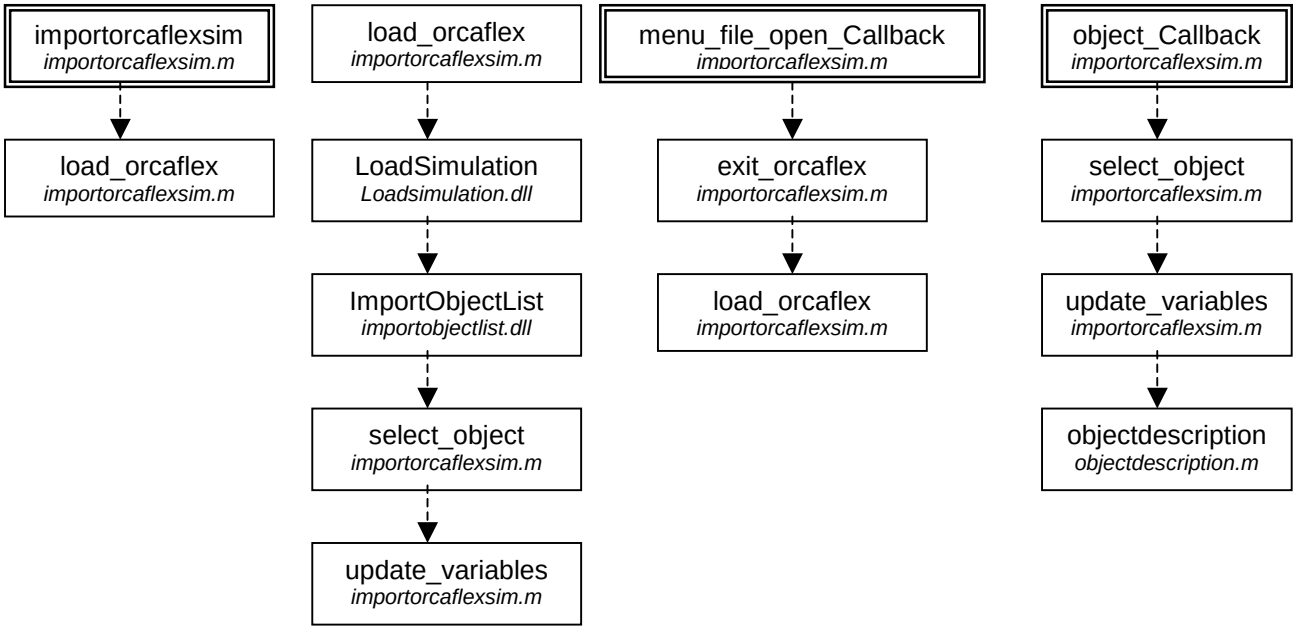
La zone de texte « user dialog » affiche des messages à l'utilisateurs comme « chargement en cours », « fichier enregistré avec succès ».

La zone de texte modifiable « Sig Filename » permet à l'utilisateur d'entrer le nom du fichier Signalysis destination (fichier SIG).

Finalement, le bouton « Save », permet d'écrire le fichier SIG et de retourner à Signalysis.

2.6.3 Détails de l'implémentation

Le schéma suivant décrit les appels dans le fichier M de l'interface graphique :



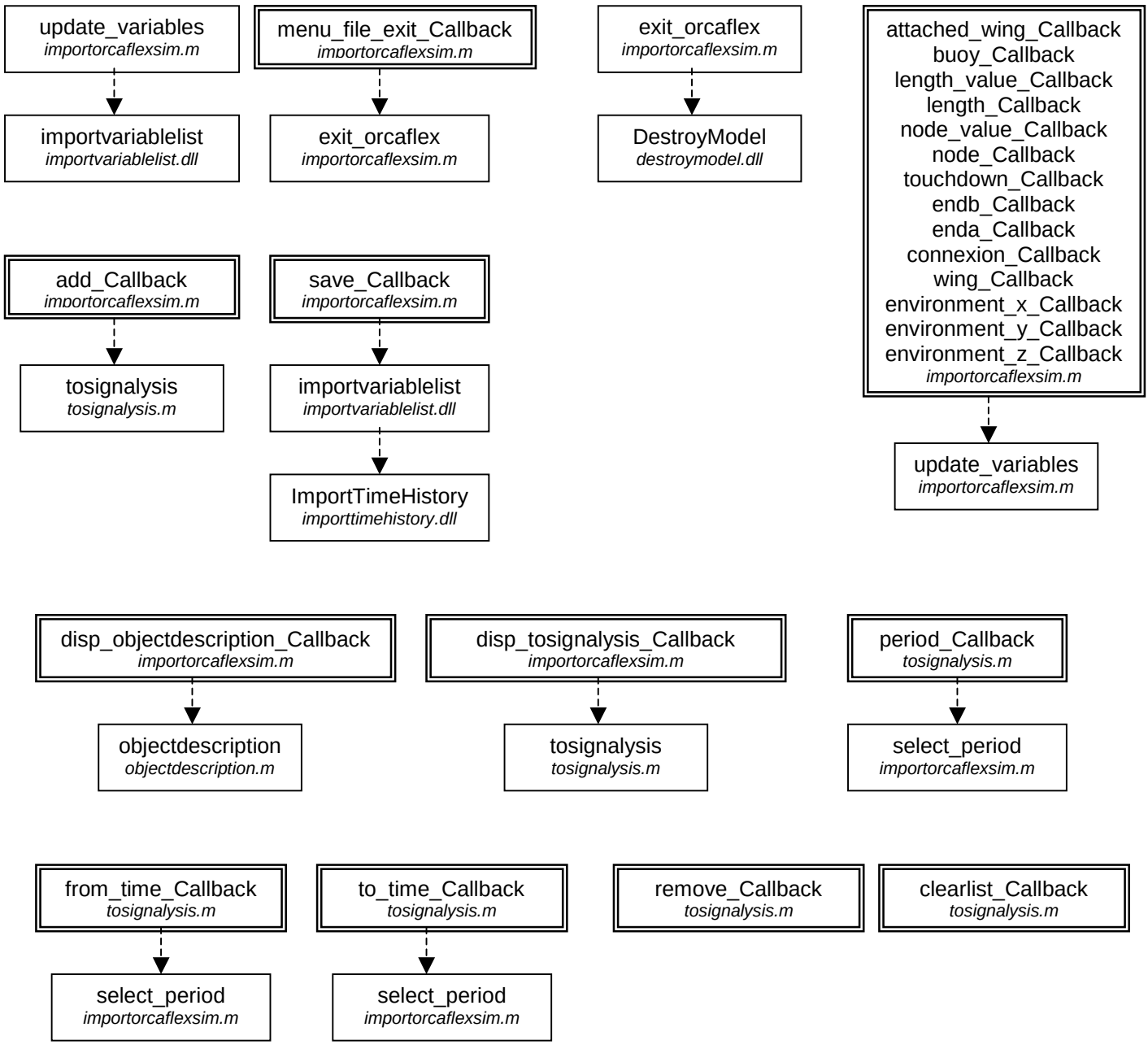


Figure 6 - Structure de la GUI

Description des fonctions :

Nom	Quand	Description
importorcaflexsim	L'application est lancée	Ouvre l'interface graphique
load_orcaflex	n/a	Charge un fichier de simulation ORCAFLEX
menu_file_open_Callback	L'utilisateur choisit Open dans le menu file	Détruit le modèle en cours et appelle load_orcaflex
object_Callback	L'utilisateur sélectionne un objet	Appelle select_object puis update_variables. Ouvre la fenêtre objectdescription
select_object	n/a	Enregistre la sélection de l'objet en interne
update_variables	n/a	Affiche la liste des variables
menu_file_exit_Callback	L'utilisateur choisit Exit dans le menu File ou l'utilisateur ferme la fenêtre principale	Appelle exit_orcaflex et ferme toutes les fenêtres.
exit_orcaflex	n/a	Détruit le modèle ORCAFLEX
disp_objectdescription_Callback	L'utilisateur clique sur le bouton d'ouverture de la fenêtre objectdescription	Ouvre ou ferme la fenêtre objectdescription
disp_tosignalysis_Callback	L'utilisateur clique sur le bouton d'ouverture de la fenêtre tosignalysis	Ouvre ou ferme la fenêtre tosignalysis
attached_wing_Callback	L'utilisateur sélectionne le bouton « attached wing »	Appelle update_variables
buoy_Callback	L'utilisateur sélectionne le bouton « buoy »	Appelle update_variables
length_value_Callback	L'utilisateur entre une valeur dans la case « Arc Length »	Appelle update_variables
length_Callback	L'utilisateur choisit le bouton « Arc Length »	Appelle update_variables
node_value_Callback	L'utilisateur entre une valeur dans la case « Node Num »	Appelle update_variables
node_Callback	L'utilisateur choisit le bouton « Node Num »	Appelle update_variables
touchdown_Callback	L'utilisateur choisit le bouton « Touchdown »	Appelle update_variables
endb_Callback	L'utilisateur choisit le bouton « EndB »	Appelle update_variables
enda_Callback	L'utilsateur choisit le bouton « EndA »	Appelle update_variables
connexion_Callback	L'utilisateur choisit lune connexion dans le menu « Winch Connection »	Appelle update_variables
wing_Callback	L'utilisateur choisit un item dans le menu « Attached Wing »	Appelle update_variables

environment_x_Callback	L'utilisateur entre une variable dans la case « Environment Position X »	Appelle update_variables
environment_y_Callback	L'utilisateur entre une variable dans la case « Environment Position Y »	Appelle update_variables
environment_z_Callback	L'utilisateur entre une variable dans la case « Environment Position Z »	Appelle update_variables
add_Callback	L'utilisateur appuie sur le bouton Add	Ouvre la fenêtre tosignalysis et ajoute les variables sélectionnées à la liste « add_variables »
save_Callback	L'utilisateur appuie sur le bouton Save	Enregistre le fichier Signalysis SIG.
remove_Callback		Efface les variables sélectionnées de la liste « add_variables »
clearlist_Callback		Efface la liste « add_variables »
period_Callback	L'utilisateur sélectionne une période	Appelle select_period
select_period	n/a	Enregistre la période choisie
from_time_Callback	L'utilisateur entre une valeur dans la case « From Time »	Appelle select_period
to_time_Callback	L'utilisateur entre une valeur dans la case « To Time »	Appelle select_period

Figure 7 - Description des fonctions de la GUI

Quelques remarques sur l'interface graphique :

- L'interface réagit aux actions erronées de l'utilisateur, comme par exemple entrer des valeurs fausses dans les champs de texte.
- La liste des variables ajoutées tient compte des variables déjà présentes pour ne pas créer de doublon.
- La zone de texte « user_dialog » avertit l'utilisateur des actions effectuées ou des erreurs détectée par la DLL Orcaflex (clé de protection absente, erreur de lecture ou d'écriture). Il l'avertit également de l'état d'avancement d'un processus long. Néanmoins, l'état d'avancement du chargement du fichier de simulation n'a pas été implémenté par manque de temps. La DLL Orcaflex fournit le moyen de connaître l'état d'avancement d'une fonction de la DLL employée.

2.7 Signalysis

SIGNALYSIS est un puissant outil permettant de faire du traitement du signal.

Il a été développé par Christian Bauduin, ingénieur en hydrodynamique à la Single Buoy Moorings.

SIGNALYSIS utilise un format de fichiers sur mesure : les fichiers SIG.

Les fichiers SIG sont en fait des fichiers de données standards MATLAB ou fichiers MAT qui ont une structure particulière que l'application SIGNALYSIS reconnaît. Ces fichiers peuvent être crées directement et simplement à l'aide d'un fichier M de MATLAB.

Il suffit simplement de créer une structure « user_data » ou ud qui va contenir les champs nécessaires à SIGNALYSIS.

Les champs de la structure user_data sont les suivants :

filename : contient le nom du fichier SIG
filepath : contient le chemin du fichier SIG
originfile : contient le nom du fichier d'origine
testname : contient des commentaires sur le fichier, quelle simulation est réalisée …
nbts : contient le nombre d'échantillons des signaux (un seul nombre pour tous les signaux)
dt : contient l'intervalle de temps entre deux échantillons
signal : est le tableau de tous les signaux
initial : contient le temps de début de la simulation

Ensuite il faut enregistrer cette structure dans un fichier MAT de MATLAB que l'on appellera SIG. Le fichier peut alors être lu par SIGNALYIS.

2.

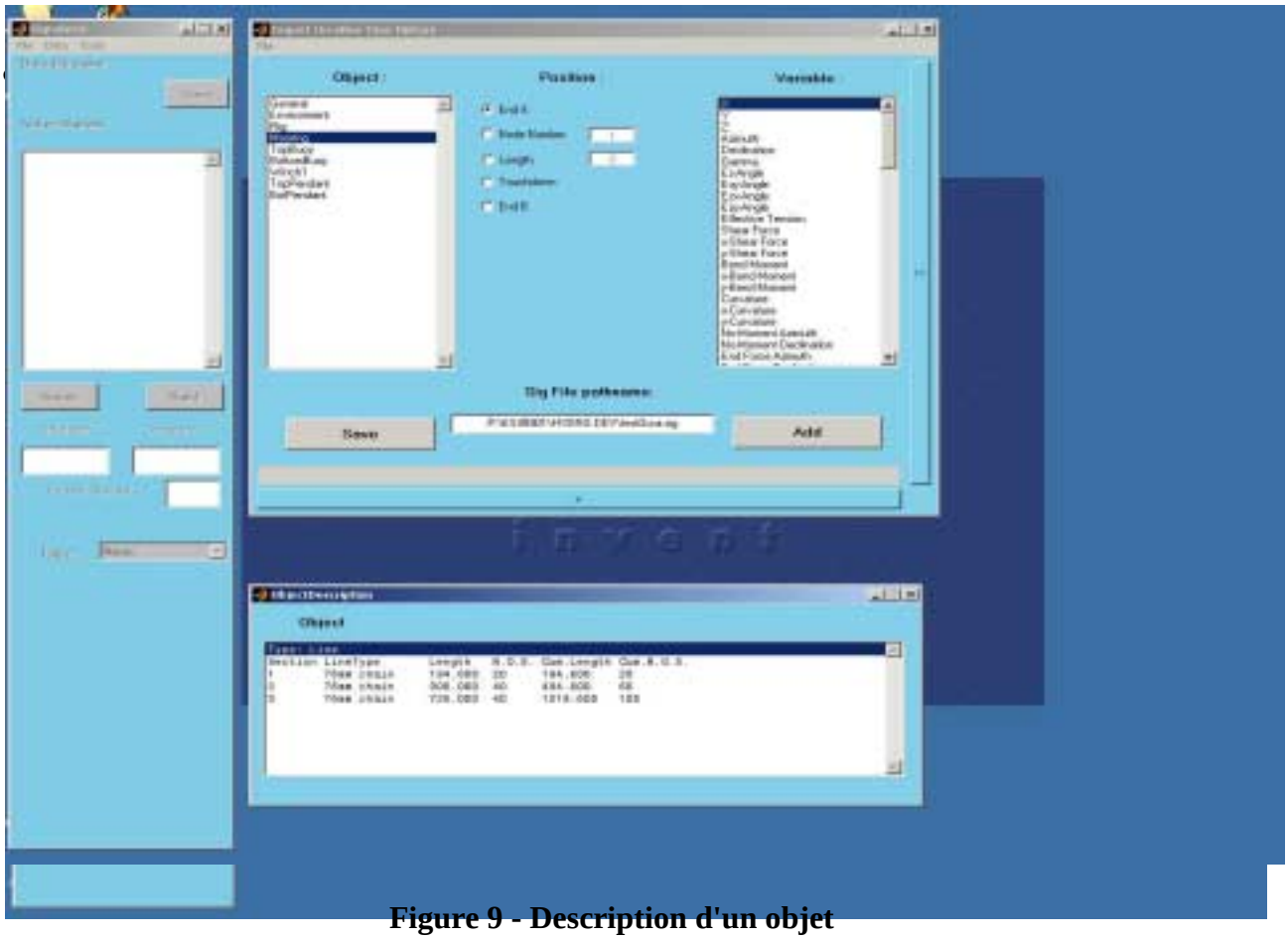


Figure 9 - Description d'un objet

Figure 8 – Ouverture du modèle

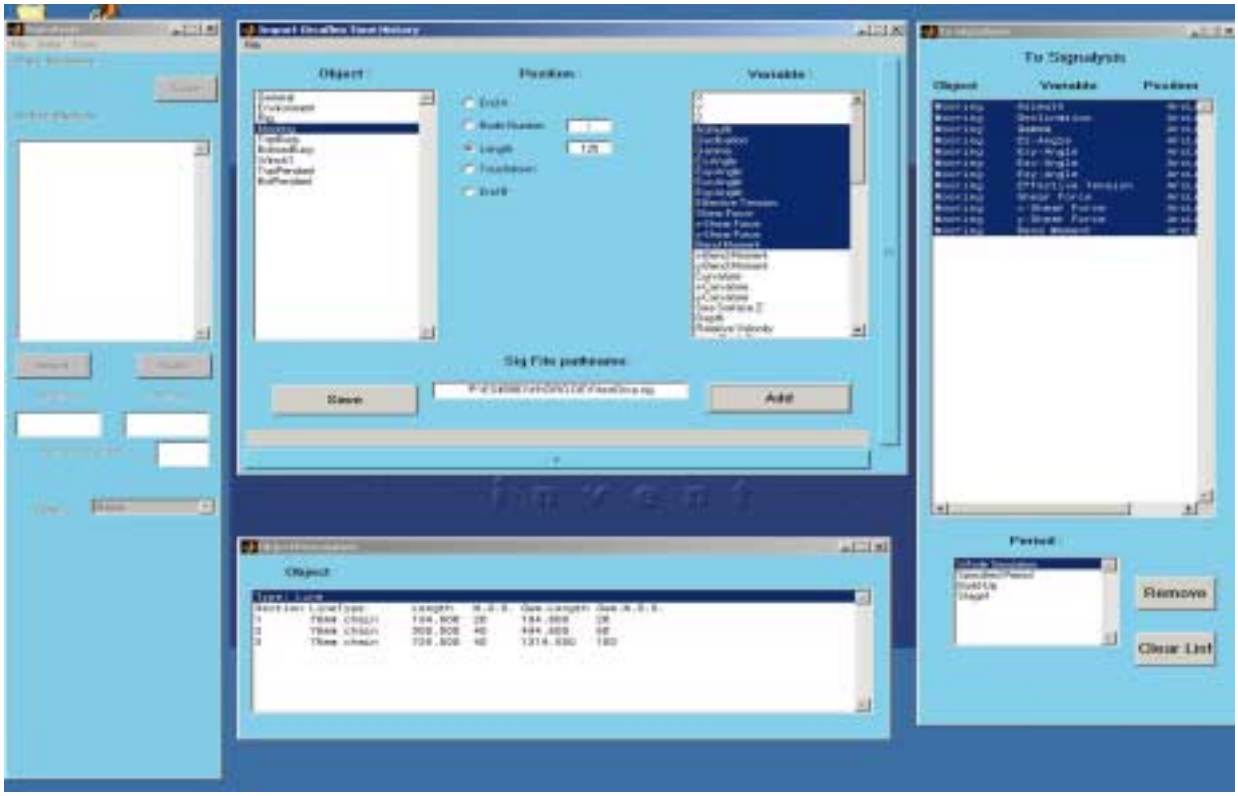


Figure 10 - Ajout de variables

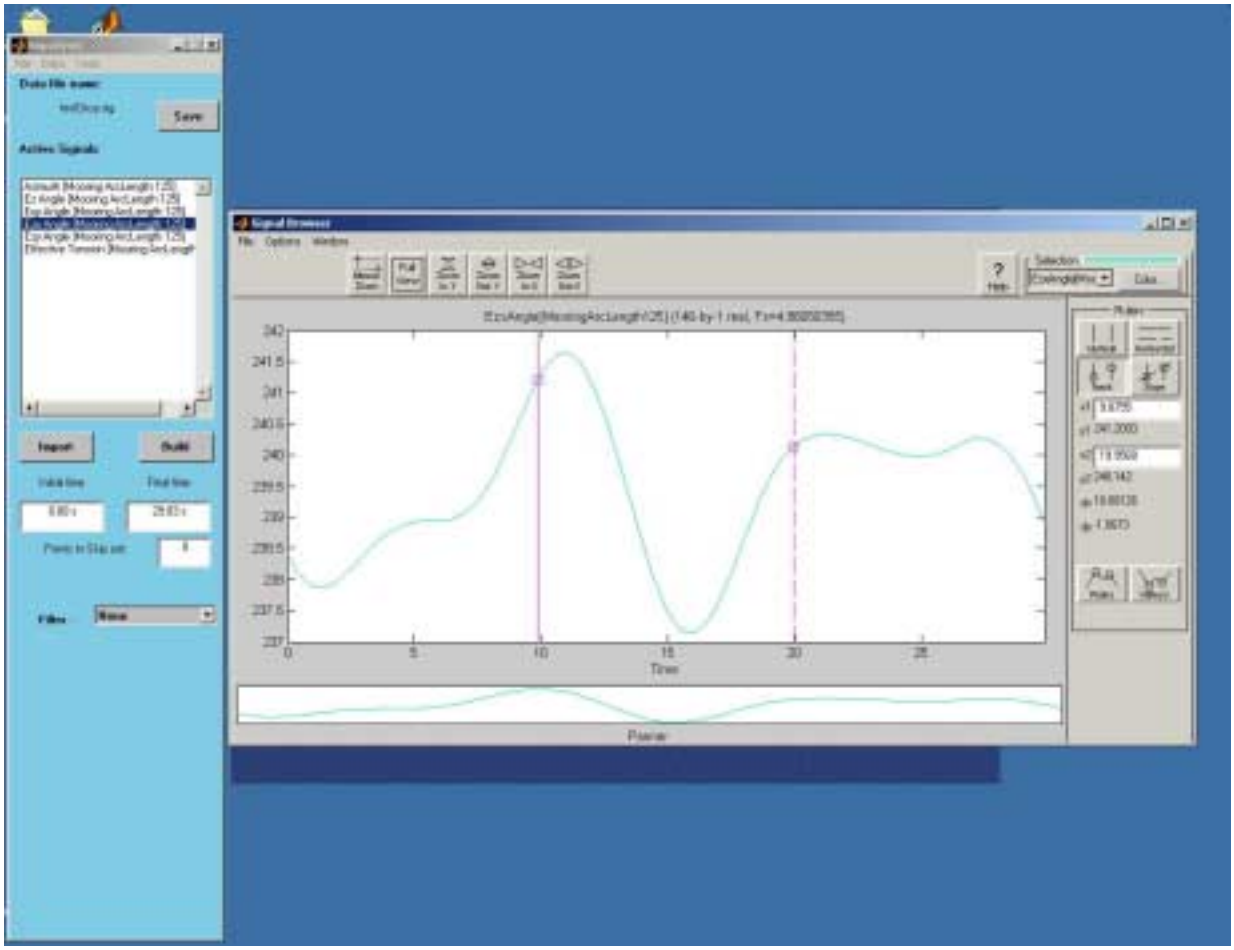


Figure 11 - Résultat dans SIGNALYSIS

2.9 Les extensions possibles

N'ayant eu finalement qu'un mois et demi pour réaliser le projet à cause du changement de sujet, je me suis vraiment concentré sur l'essentiel du cahier des charges : réaliser le transfert de données vers SIGNALYSIS de la manière la plus juste possible, pour que les ingénieurs aient à leur disposition un outil qu'ils peuvent utiliser dès maintenant et qui leur permettra de gagner du temps. Néanmoins, je suis parfaitement conscient que cette application a ses limites et peut être largement améliorée. Je vais décrire dans cette section les améliorations possibles que j'entrevois pour cette application.

Ouverture d'un fichier SIG depuis SPO :

Cette fonctionnalité était présente dans le cahier des charges mais n'a pas pu être terminée. Cette fonctionnalité rendrait possible l'ouverture depuis SPO d'un fichier SIG qui a été produit par SPO.

On pourrait ainsi avoir la liste des variables qui ont été déjà importées et on pourrait ajouter d'autres variables au fichier SIG.

Cette fonctionnalité très pratique pourrait être implémenter de la sorte :

- Il faudrait rouvrir le même fichier SIM qu'à l'origine et vérifier que le modèle n'a pas changé : même liste d'objet, mêmes listes de variables, mêmes informations contenues dans les objets. Sans ces vérifications des erreurs pourrait éventuellement survenir sous le nez des ingénieurs.
- Il faudrait ensuite afficher la liste des variables déjà importées et empêcher l'utilisateur d'effacer ces variables ou de changer la période. Enfin, il faudrait recréer un fichier SIG contenant les anciennes valeurs et les nouvelles).

Cependant pour pouvoir vérifier que le modèle n'a pas changé, il faut pouvoir stocker des informations dans le fichier SIG créé ou alors créer un autre fichier contenant ces informations ou encore créer un autre standard, par exemple SPO. Ces choix nécessite l'avis de Christian Bauduin qui n'était pas présent à la fin du développement.

Ouverture de plusieurs fichiers dans SPO :

On pourrait également imaginer la possibilité d'ouvrir plusieurs fichiers dans SPO, on aurait ainsi dans la liste des objets ceux de plusieurs fichiers de simulation et on pourrait construire dans un seul fichier SIG des valeurs provenant de plusieurs fichiers de simulation (pour faire des synthèse par exemple).

A priori cette fonctionnalité n'apparaît pas très difficile à implémenter. La structure de données au sein de l'interface graphique serait un peu plus complexe pour prendre en compte la présence de plusieurs modèles mais, dans les grandes lignes, le fonctionnement de l'application resterait à peu près identique.

Sélection de plusieurs positions :

L'application ne permet de sélectionner qu'une seule position à la fois. Car la liste des variables affichée dépend d'une position précise.

Néanmoins, on pourrait imaginer pouvoir sélectionner plusieurs positions pour un seul objet, tout en faisant bien la distinction entre les positions qui affichent des listes de variables différentes.

Pour une ligne par exemple, on n'aurait qu'une seule possibilité de sélectionner le début ou la fin de la ligne, car ces deux positions ont des variables spécifiques qui ne sont pas accessibles par les autres points de la ligne, mais par contre on pourrait sélectionner d'un seul coup un ensemble de points autres de la ligne, comme par exemple un intervalle de points ou même toute la ligne sauf les extrémités. Cela permettrait d'ajouter d'un seul coup à la liste des variables tous les choix de positions effectués.

Une autre alternative serait, pour un ensemble de choix donnés, ne montrer que la liste des variables qui sont communs à tous les choix. Et pouvoir ainsi sélectionner d'un seul coup n'importe quel ensemble de points de la droite.

Affichage d'un indicateur de progression :

L'application affiche déjà sa progression pour les tâches qu'elle exécute. Néanmoins, elle n'affiche pas la progression des tâches exécutées par la DLL. Cet affichage pourrait s'avérer bien commode lors de l'appel de loadsimulation, qui peut demander plusieurs dizaines de minutes lors de l'ouverture d'un fichier de 3 Mega-octets.

Cette fonctionnalité est assez simple à implémenter. La DLL fournit une fonction C_SetProgressHandler qui permet de définir une fonction qui sera avertie de la progression d'une des fonctions coûteuse en calcul de la DLL. Cette fonction devra envoyer des informations à MATLAB alors que le fichier MEX

continue de s’exécuter. Cela peut se faire en invoquant par exemple une fonction d’affichage de MATLAB depuis le fichier MEX.

Maintenance de l’application :

Que faire si les futures version d’ORCAFLEX modifient par exemple le nom des variables du modèle ? L’application deviendrait alors totalement inutilisable. On pourrait éventuellement définir ces variables par des macros au sein du code des fichiers MEX. Il n’y aurait plus ensuite qu’à changer ces macros pour que l’application fonctionne toujours.

3 Analyse de la réalisation

3.0 Planning

La première réunion avec la Single Buoy Moorings a eu lieu pendant la semaine 9. Comme expliqué dans l'introduction, le projet avait alors une toute autre direction. La période entre la semaine 9 et la semaine 18 a consisté en une phase d'analyse du projet alors proposé. Pour aboutir, à la suite d'une réunion avec Olivier Devillers, à la conclusion que le projet proposé était trop important pour un simple TER de maîtrise d'informatique et qu'il fallait revoir le sujet. La réunion avec Christian Bauduin a eu lieu quelque temps après les partiels, lors de la semaine 27, c'est elle qui a permis de réorienter le projet sur le sujet actuel.

Tâche	***Date de début prévue***	***Date de fin prévue***	***Date de début effective***	***Date de fin effective***
Analyse et spécification (lecture des documentations, lecture du code de la spreadsheet Excel…)	01/07/04	26/07/04	01/07/04	26/07/04
Installation dans les locaux de la SBM	26/07/04	01/08/04	26/07/04	01/08/04
Phase de conception, premiers tests sur la DLL	01/08/04	13/08/04	01/08/04	13/08/04
Rédaction du cahier des charges en français	13/08/04	17/08/04	13/08/04	15/08/04
Installation des outils nécessaires à la programmation	15/08/04	17/08/04	15/08/04	15/08/04
Rédaction d'une version du cahier des charges en anglais pour la SBM	19/08/04	22/08/04	19/08/04	15/09/04
Ecriture des routines d'importation	17/08/04	25/08/04	17/08/04	30/08/04
Test sur des gros fichiers pour vérifier l'occupation mémoire	25/08/04	25/08/04	30/08/04	30/08/04
Réalisation d'une interface graphique	25/08/04	5/09/04	30/08/04	17/09/04
Période des partiels de Septembre	06/09/04	13/09/04	/	/
Réalisation d'une application indépendante en C produisant des fichiers MATLAB .mat ou SIGNALYSIS .sig à partir de fichiers de simulation ORCAFLEX .sim	13/09/04	18/09/04	abandonnée	
Phase de tests, améliorations et paramétrages	18/09/04	19/09/04	20/09/04	27/09/04
Intégration du programme dans l'entreprise	20/09/04	20/09/04	27/09/04	
Finalisation du rapport	21/09/04	25/09/04	18/09/04	19/09/04
Soutenance du TER	Fin septembre 2004		21/09/04	

La réalisation du projet pendant la phase de production a été plutôt éprouvante physiquement. Le fait de travailler trente-cinq heures par semaines en plus de mon projet a évidemment rendu la tâche plus difficile. J'ai travaillé en moyenne 4-5 heures par jour sur le projet les jours ou je travaillais à côté ; j'ai dû à plusieurs reprise échanger mes horaires avec quelqu'un d'autre pour pouvoir aller travailler le matin à la Single Buoy Moorings et avoir une journée complète de travail par semaine à la SBM. Cette journée là, je travaillais en moyenne douze heures.

Le planning du cahier des charges a été faussé car je pensais au départ que ma soutenance allait avoir lieu autour du 28 Septembre alors qu'elle a lieu le 21. Par conséquent, je n'ai pas pu tester l'application avec les autres ingénieurs. Néanmoins, j'ai testé l'application moi-même tout au long du projet. S'agissant d'une interface graphique, il était facile de voir ce qui s'affichait correctement et ce qui ne s'affichait pas.

L'écriture des routines d'importations a duré plus longtemps que prévu car je n'avais pas pensé au départ faire une hiérarchie complète d'objets. Je pensais n'avoir qu'à extraire une ligne d'objets et une liste de variables. J'ai découvert certaines subtilités de l'API d'ORCAFLEX au fur et à mesure. Ce planning a également été faussé par le départ en vacances de Christian Bauduin, avec qui je pensais pouvoir finaliser mon application. J'ai pu néanmoins obtenir les renseignements que je souhaitais auprès des autres ingénieurs.

Conclusion

Bilan du TER

Afin de dresser un bilan de mon TER, je vais d'abord récapituler le travail effectué. J'ai réalisé un ensemble de routines permettant d'importer des données depuis un fichier de simulation ORCAFLEX. J'ai réalisé une interface graphique utilisant ces routines pour permettre à un utilisateur d'ouvrir un fichier, de sélectionner les signaux qu'il souhaite et d'obtenir le résultat sous SIGNALYSIS.

Ce TER m'a permis de réaliser une application complète, de la phase de conception à la mise en place. Cette application sera utilisée demain par les ingénieurs de la SBM. Le plus grand enrichissement que j'ai retiré de ce projet, et c'est pour cela que je l'ai choisi, et d'avoir eu le sentiment de réaliser quelque chose pour quelqu'un d'autre et non pas pour moi-même, de réaliser quelque chose d'utile. C'est ce que je pense être une des caractéristique du travail d'un développeur d'application ou d'un ingénieur.

J'ai trouvé également l'organisation du projet, toutes les différentes étapes de la conception au développement enrichissantes car j'ai pu aller au bout de mon objectif et voir le projet avancer étapes par étapes.

Si c'était à refaire

J'ai bien sûr perdu du temps au début du projet à cause de l'hésitation sur le sujet. Peut-être aurait-il fallu réfléchir davantage avant de me lancer dans un projet trop difficile à réaliser. Heureusement j'ai pu me rendre compte à temps de cette erreur et l'exprimer à la SBM. J'ai passé plus de temps que prévu à la rédaction du cahier des charges, mais il était nécessaire de bien maîtriser le sujet pour pouvoir le rédiger correctement sans rester trop vague, je ne pouvais comprendre le logiciel ORCAFLEX sans avoir véritablement travaillé dessus, c'est à dire sans avoir commencé à développer.

Je n'ai pas pu réaliser de phase de test avant la rédaction du rapport, mais l'application fonctionne correctement et on obtient bien le résultat escompté.

Références bibliographiques

Sites Web

Le site de la Single Buoy Moorings :

- http://www.singlebuoy.com/

Le site d'Orcina :

- http://www.orcina.com/

Le site de MATLAB :

- http://www.mathworks.com/

Les outils de compilation

- *MinGW* : http://www.mingw.org/
- *GnuMex* : http://gnumex.sourceforge.net/ - setup