



## MAT2PTU

Conception et réalisation d'un outil capable de générer un fichier de Test Unitaire à partir de fichiers issus de simulations sur Matlab

**RAPPORT CONFIDENTIEL**

**Tuteur encadrant**  
Stéphane Bride

**Tuteur enseignant**  
Marianne De-Michiel

**Stagiaire**  
Théo Vaucher

**8 avril au 28 juin 2013**  
**Année 2012 - 2013**





**Continental Automotive France SAS - Toulouse**

## **MAT2PTU**

Conception et réalisation d'un outil capable de générer un fichier de  
Test Unitaire à partir de fichiers issus de simulations sur Matlab

**RAPPORT CONFIDENTIEL**

**Tuteur encadrant**  
Stéphane Bride

**Tuteur enseignant**  
Marianne De-Michiel

**Stagiaire**  
Théo Vaucher

**8 avril au 28 juin 2013**  
**Année 2012 - 2013**



## Remerciements

Je remercie, bien évidemment l'entreprise Continental de m'avoir accueilli pour effectuer ce stage.

Je tiens à remercier Mr. **Christophe VANDERDONCKT**, mon tuteur effectif dans l'entreprise, pour sa présence et ses conseils. Merci à Mr. **Stéphane BRIDE**, mon tuteur dans l'entreprise, et Mr. **Philippe AUDOIN** de m'avoir accepté pour ce stage.

Une pensée également à Mme. **Joelle DELCOL** pour son apport de bonne humeur quotidien. De même pour **l'équipe du 3ème étage** ainsi que **l'équipe du 1<sup>er</sup>**.

Et enfin, je remercie les stagiaires **Jeremy PAVAGEAU** et **Lucile GONCALVES**, présents dans l'entreprise durant la durée de mon stage, avec qui j'ai pu discuter sur beaucoup de sujets.



## Sommaire

Remerciements .....	1
Table des illustrations .....	4
Introduction .....	5
I – Continental .....	6
1. L'entreprise .....	6
2. Le site Toulousain .....	7
3. La division Powertrain .....	7
4. Intérêt d'un outil de génération de tests .....	8
II – Présentation du projet MAT2PTU .....	9
III – Travail réalisé .....	11
1. Analyse et conception .....	11
2. Développement de l'outil MAT2PTU .....	15
a. Utilisation de bibliothèques existantes .....	15
b. Création de l'arborescence du projet .....	16
c. Module App .....	16
d. Module Util .....	20
e. Module MatReader .....	23
f. Module Plot .....	24
g. Module PtuWriter .....	25
h. Récapitulatif des flux d'entrées et de sorties de l'application .....	26
i. Manuel utilisateur .....	26
3. Tests de l'application .....	27
a. Performances .....	27
b. Tests unitaires des modules de l'application .....	27
4. Livraison .....	28
5. Etat de l'outil .....	29
a. Fonctionnalités .....	29
b. Evolutions futures .....	29
IV – Bilan .....	30
1. Technique .....	30
2. Humain .....	30
Conclusion .....	31
Glossaire .....	32
Références bibliographiques .....	33
Résumé / Abstract .....	34
Sommaires des annexes .....	35

## Table des illustrations

Figure 1 : Chiffres d'affaires et nombre d'employés .....	6
Figure 2 : Répartition du groupe Continental dans le monde.....	6
Figure 3 : Le site Continental de Toulouse .....	7
Figure 4 : Division Powertrain .....	7
Figure 5 : Visualisation d'un module avec Matlab Simulink.....	8
Figure 6 : Extrait d'un fichier de tests unitaires.....	8
Figure 7 : Un graphique sous Matlab .....	9
Figure 8 : Extrait d'un fichier .PTU .....	9
Figure 9 : Visualisation d'un fichier .PTU avec les structogrammes.....	10
Figure 10 : Champs contenant les informations des signaux.....	11
Figure 11 : Informations d'une entrée .....	11
Figure 12 : Données d'une entrée .....	11
Figure 13 : Classe MatReader .....	13
Figure 14 : Classe Plot.....	13
Figure 15 : Classe PtuWriter .....	13
Figure 16 : Extrait du diagramme de séquence.....	13
Figure 17 : Extrait du diagramme de séquence système .....	14
Figure 18 : Maquette de l'application .....	14
Figure 19 : Résultat d'un test de la librairie JMatIO .....	15
Figure 20 : Test de génération de fichier XML .....	15
Figure 21 : Fichier XML en sortie .....	15
Figure 22 : Exemple d'utilisation de JFreeChart .....	16
Figure 23 : Packages .....	16
Figure 24 : Classe App.....	17
Figure 25 : Classe Action.....	18
Figure 26 : Classe View .....	18
Figure 27 : Test de positionnement des éléments avec le GridBagLayout .....	19
Figure 28 : Interface de l'application.....	19
Figure 29 : Exemple d'une couleur presque invisible.....	20
Figure 30 : Classe Signal .....	20
Figure 31 : Classe Data .....	21
Figure 32 : Classe SelectInfo .....	21
Figure 33 : Classe XmlParser.....	22
Figure 34 : Classe MatReader .....	23
Figure 35 : Classe Plot.....	24
Figure 36 : Affichage de la sélection.....	24
Figure 37 : Classe PtuWriter .....	25
Figure 38 : Ensemble des échanges entre l'outil et les fichiers MAT, XML et PTU .....	26



## Introduction

Ce rapport de stage présente l'ensemble du travail effectué lors du stage de fin d'études d'un D.U.T. Informatique effectué à l'I.U.T. de Blagnac au sein de l'entreprise Continental Automotive à Toulouse. Le stage s'est déroulé sur une durée de 3 mois.

Connaissant peu le domaine de l'automobile, c'était pour moi l'opportunité de découvrir ce secteur et par la même occasion de découvrir l'entreprise Continental. Entreprise, qui, contrairement à ce que pense la plupart des personnes, ne fabrique pas uniquement des pneus.

Le stage avait pour principale mission la réalisation d'un outil permettant la visualisation d'enregistrements Matlab / Simulink (ensemble de fichiers Matlab .mat) avec la possibilité de sélectionner des points ou des plages de valeurs qui définiront des cas de test afin de générer un fichier de Tests Unitaires (.ptu), c'est-à-dire, vérifier la cohérence des valeurs de différents signaux d'entrée et de sortie.

Ce rapport présente dans un premier temps l'entreprise et le groupe Continental. Puis la présentation du projet en lui-même avec le travail effectué et l'état actuel du projet. Pour finir, j'établirai le bilan technique et humain ressenti durant la durée du stage ainsi qu'une conclusion générale.

## I – Continental

### 1. L'entreprise

L'entreprise Continental a été fondée en 1871 à Hanovre en Allemagne. A ses débuts, l'entreprise fabriquait des tissus caoutchouteux, des pneumatiques pour voitures et vélos.

Aujourd'hui, elle est devenue le cinquième plus grand équipementier dans le monde et premier fabricant de pneumatique en Allemagne, deuxième au niveau européen, et quatrième au niveau mondial.

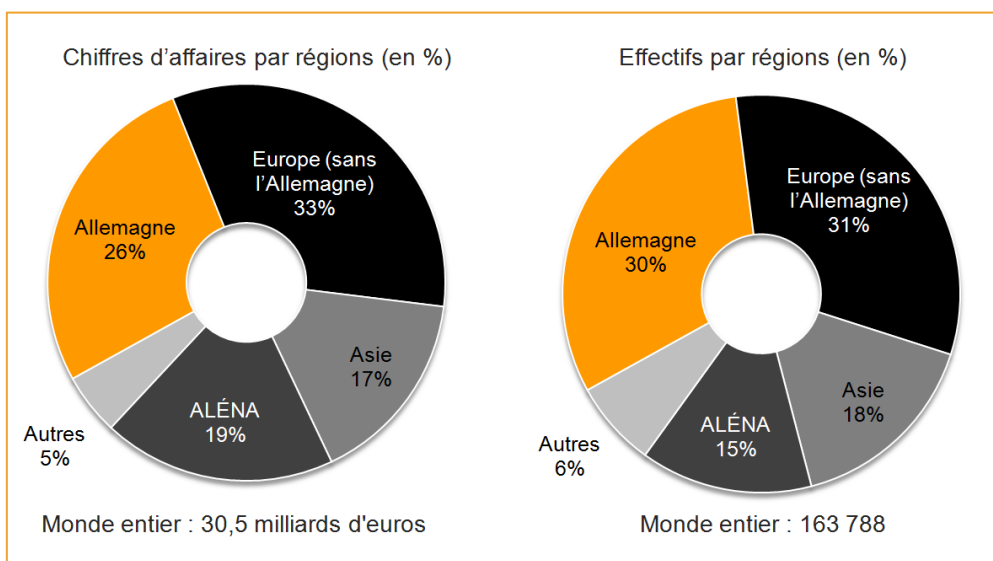


Figure 1 : Chiffres d'affaires et nombre d'employés

L'achat de la division Automotive du groupe Siemens VDO en 2007 a permis d'augmenter le chiffre d'affaires annuel du groupe qui est passé de 13 à plus de 30 milliards d'euros (chiffre 2011). Continental compte maintenant plus de 163 000 employés dans le monde (chiffre 2011).

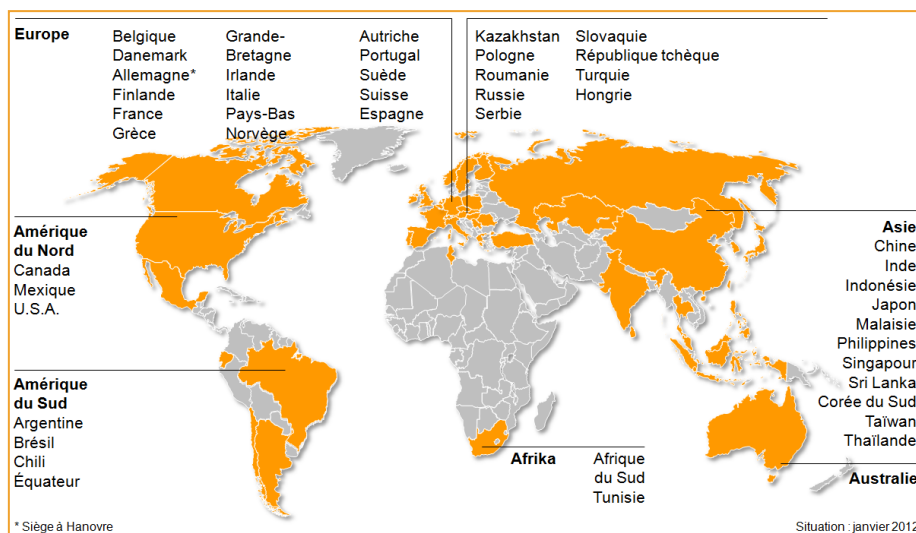


Figure 2 : Répartition du groupe Continental dans le monde

Le groupe est présent dans 269 sites et 46 pays au niveau mondial.

## 2. Le site Toulousain

Le site de Toulouse, appartenant anciennement au groupe Siemens-VDO, a été racheté en 2007 par le groupe Continental.



Figure 3 : Le site Continental de Toulouse

Le site est constitué d'une usine, produisant plus de 200 millions de pièces par mois, d'un centre d'essais capable de tester 7 moteurs, des injecteurs diesel ou essence, ou la réaction du véhicule face à différentes températures et d'un centre de recherche et développement composé de plusieurs divisions<sup>1</sup> : powertrain, chassis & safety et interior.

## 3. La division Powertrain

Powertrain
Engine Systems
Transmission
Hybrid Electric Vehicle
Sensors & Actuators
Fuel Supply

Figure 4 : Division Powertrain

La division Powertrain s'occupe essentiellement du contrôle moteur, au niveau logiciel, matériel (ECU : Engine Control Unit), mise au point et également des systèmes d'injection diesel et essence.

Le service où j'ai effectué le stage est : P-ES-E-SYS-ETV-V

P : Powertrain.

ES-E-SYS : Engine Systems.

ETV : Engineering Tool and Verification.

Engineering Tool : outils de développement informatique.

Vérification : développement des tests automatiques pour le logiciel.

<sup>1</sup> Retrouvez l'ensemble des divisions du groupe Continental en [annexe VI](#).

#### 4. Intérêt d'un outil de génération de tests

Aujourd'hui le processus de développement logiciel utilise plusieurs logiciels.

Pour la partie conception, les développeurs utilisent Matlab/Simulink qui permet de concevoir le module à l'aide de blocs mais aussi de tester (injection de stimuli sur les entrées).

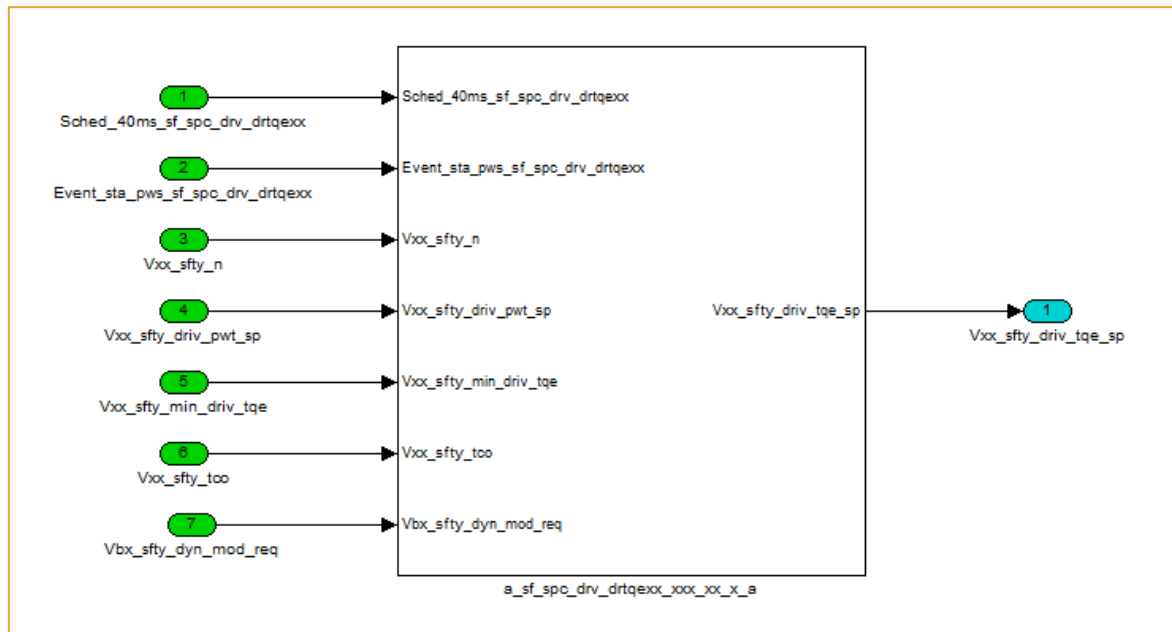


Figure 5 : Visualisation d'un module avec Matlab Simulink

Concernant la phase de codage du module, le code écrit est testé avec le logiciel Rational Test RealTime (RTRT) avec des fichiers de tests unitaires. Ce fichier de tests unitaires comprend les cas de test avec les valeurs à tester. Ces fichiers sont écrits avec des mots clés spécifiques.

```
TEST 1
FAMILY nominal
~T
COMMENT point 0.52
~T
ELEMENT

~T
USE calibration_set_1

~T
~INPORT
VAR Vbx_sfty_dyn_mod_req, INIT = 0, EV = INIT
VAR Vxo_sfty_driv_pwt_sp, INIT = 0p0.25, EV = INIT
VAR Vxo_sfty_min_driv_tqe, INIT = 0p0.0, EV = INIT
VAR Vxo_sfty_n, INIT = 0p2592.0, EV = INIT
VAR Vxo_sfty_tco, INIT = 0p0.0, EV = INIT

~OUTPORT
VAR Vxo_sfty_driv_tqe_sp, INIT = 0p0.0, EV = 0p76.5546875

~INTERN
#SfSpcDrv_Drtqexox_40ms();

END ELEMENT
~T
END TEST
```

Figure 6 : Extrait d'un fichier de tests unitaires

Aujourd'hui la rédaction du fichier de tests unitaires s'effectue à la main, ce qui prend du temps. D'où l'intérêt de cet outil qui permet de sélectionner, graphiquement, les différents cas de test puis de générer le fichier.

## II – Présentation du projet MAT2PTU

Le but de ce projet est de réaliser une application capable de visualiser les résultats d'enregistrement Matlab / Simulink (.mat) et de pouvoir, en sélectionnant des cas de test sous forme de points ou de plages, générer un fichier de Tests Unitaires (.ptu).

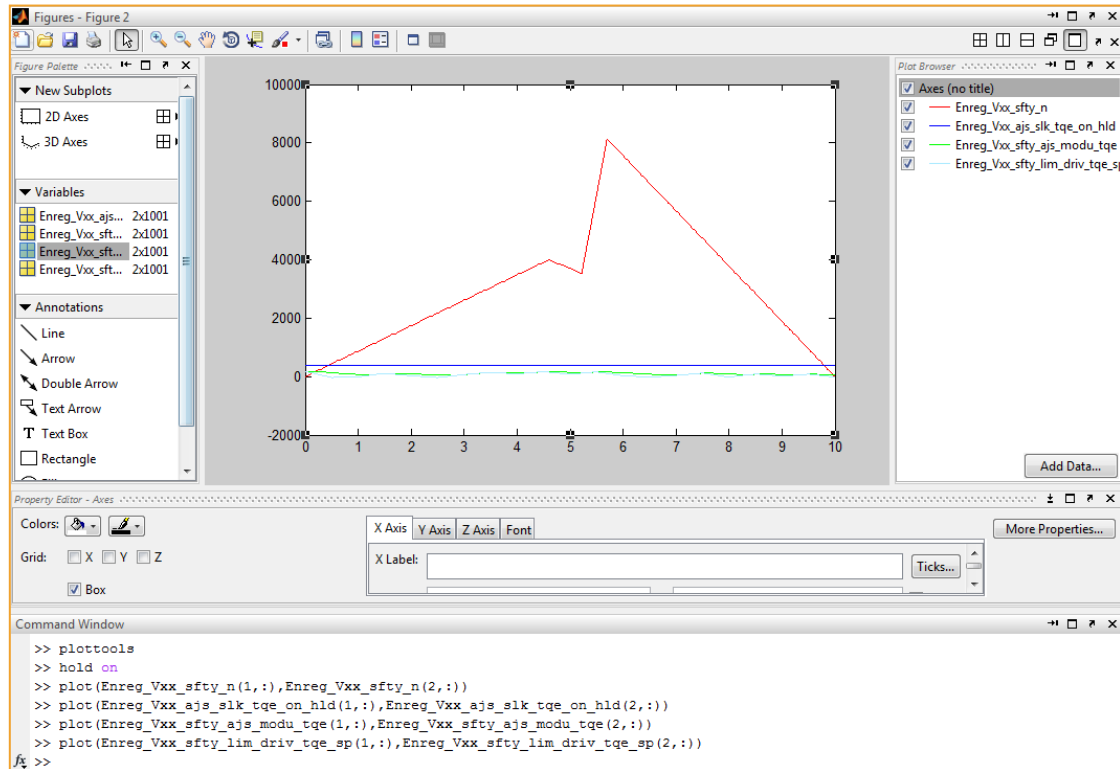


Figure 7 : Un graphique sous Matlab

Les différentes entrées sont injectées dans le module simulé et les sorties sont enregistrés.

Le fichier PTU est un fichier texte qui comporte une notation particulière.

```
~A
~+:Test 1 TestName
~T
TEST 1
FAMILY nominal
~T
COMMENT comment...
~T
ELEMENT

~T
USE calibration_set_1

var Vxx_sfty_lim_driv_tqe_sp,      init = 0p1024, ev = init
var Vxx_sfty_ajs_modu_tqe,        init = 0p512,  ev = 0p1024
var Vxx_sfty_ajs_modu_tqe_prev,   init = 0p0,    ev = 0p512
var Vxx_sfty_ajs_tq_dif,          init = 0p0,    ev = 0p512

#SfSpPaj_Ajmodox_40ms();

END ELEMENT
~T
END TEST
~E
```

Figure 8 : Extrait d'un fichier .PTU

Quelques mots clés d'un fichier de Tests Unitaires :

*TEST ... END TEST* : Permet de définir un test.

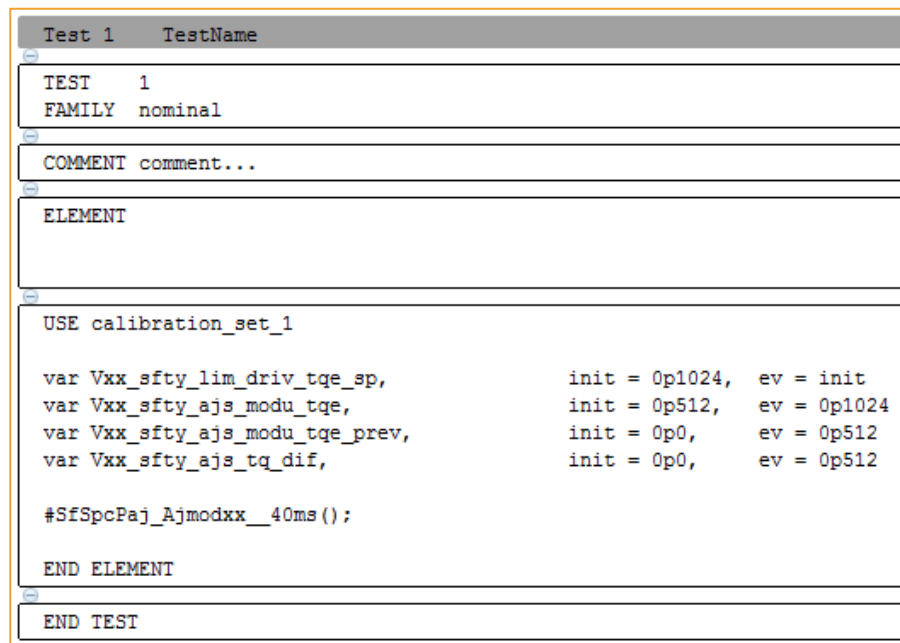
*ENVIRONMENT ... END ENVIRONMENT* : Permet de définir un environnement, utile par exemple pour les calibrations (valeurs fixes définies dans le fichier .mat) qui ne varient pas.

*COMMENT* : Permet d'écrire un commentaire sur une seule ligne.

*COMMENT START ... COMMENT END* : Permet d'écrire un commentaire sur plusieurs lignes.

*#* : Ce caractère permet d'insérer une instruction du langage C.

Il est également composé de structogrammes, ce qui permet de visualiser le fichier de façon structurée.



```

Test 1  TestName
TEST  1
FAMILY nominal

COMMENT comment...

ELEMENT

USE calibration_set_1

var Vxx_sfty_lim_driv_tqe_sp,          init = 0p1024,  ev = init
var Vxx_sfty_ajs_modu_tqe,           init = 0p512,   ev = 0p1024
var Vxx_sfty_ajs_modu_tqe_prev,      init = 0p0,     ev = 0p512
var Vxx_sfty_ajs_tq_dif,             init = 0p0,     ev = 0p512

#SfSpcPaj_Ajmodxx__40ms();

END ELEMENT

END TEST
    
```

Figure 9 : Visualisation d'un fichier .PTU avec les structogrammes

Le fichier de Tests Unitaires est exécuté par le logiciel IBM Rational Test RealTime. Le logiciel va initialiser les variables de calibrations, d'entrées et de sorties puis il va appeler les fonctions du module testé, qui est un fichier écrit en langage C. Le logiciel va ensuite tester les valeurs attendues des variables d'entrées et de sorties par rapport à celles obtenues après passage dans le module.

Le logiciel génère ensuite un rapport de test qui affiche le succès ou non des différents cas de test.

## III – Travail réalisé

### 1. Analyse et conception

En premier lieu, j'ai étudié le fonctionnement de Matlab, qui est un logiciel de calcul numérique, de visualisation et de programmation. J'ai suivi les indications de mon tuteur et suivi l'aide du logiciel, qui dispose d'une bonne documentation. J'ai donc manipulé les matrices, les structures, et l'outil qui permet de tracer les courbes.

Dans un second temps, j'ai analysé la composition d'un résultat d'une simulation Renault. Il y a donc un fichier Matlab contenant diverses informations sur les différentes entrées, sorties, et valeurs des calibrations, et des variables « mémoire non volatile » (mémoire sauvegardée à l'arrêt du calculateur).

ScalingCalibs	<1x10 struct>
ScalingInPorts	<1x4 struct>
ScalingIntern	<1x9 struct>
ScalingNVMY	<0x0 struct>
ScalingOutPorts	<1x1 struct>

Figure 10 : Champs contenant les informations des signaux

Dans l'exemple ci-dessus, on peut voir qu'il y a 10 variables de calibration, 4 entrées, 9 internes et 1 sortie.

Concernant les variables d'entrées, de sorties et internes, les informations sont stockées dans une structure enregistrée dans un fichier Matlab et les vecteurs de données sont sauvegardés dans d'autres fichiers Matlab. Dans le premier fichier Matlab, figurent des informations comme le nom du signal, son unité, son lsb<sup>2</sup> et sa valeur si c'est une calibration :

ScalingInPorts(1,1) <1x1 struct>			
Field ▲	Value	Min	Max
Name	'Event_sta_pws'		

Figure 11 : Informations d'une entrée

Les autres fichiers Matlab contiennent les données d'un signal qui est un tableau à deux entrées. La première ligne pour le temps et la deuxième pour les valeurs :

Enreg_Event_sta_pws <2x1001 double>									
	1	2	3	4	5	6	7	8	9
1	0	0.0100	0.0200	0.0300	0.0400	0.0500	0.0600	0.0700	0.0800
2	1	0	0	0	0	0	0	0	0

Figure 12 : Données d'une entrée

<sup>2</sup> Permet de passer de la valeur physique à la valeur codée et inversement. Explication détaillée en [partie III-2-g](#). Voir [annexe VI](#).



L'étape de conception n'était pas spécifiée dans la réalisation de l'outil, mais c'est pour moi une étape nécessaire pour avoir une application conçue sur une base solide et pour pouvoir, par la suite, maintenir et améliorer l'application dans les meilleures conditions.

J'ai ainsi suivi le cycle de développement en V<sup>3</sup>, qui est un standard dans le développement de logiciel. Il permet une meilleure réactivité dans l'enchaînement des étapes de la réalisation d'une application. J'ai utilisé le logiciel Eclipse<sup>4</sup> et le langage JAVA<sup>5</sup> pour le développement de l'outil.

Je suis donc parti sur l'étude de la spécification<sup>6</sup> fournie qui comprend les exigences fonctionnelles et non fonctionnelles et les différents cas d'usages.

Je me suis basé sur le langage UML<sup>7</sup> en choisissant les différents schémas utiles pour le développement. La conception sera donc constituée d'une étude de l'architecture matérielle et logicielle et d'un diagramme des cas d'utilisation, diagramme de classes, diagramme de séquence, diagramme de séquence système. Concernant l'IHM, il n'y aura qu'une fenêtre principale, j'ai donc choisi de ne réaliser qu'une maquette.

Pour le diagramme des cas d'utilisation<sup>8</sup>, j'ai réalisé le schéma à partir des cas d'usages définis dans la spécification.

Pour le diagramme des classes<sup>9</sup>, j'ai suivi le modèle MVC<sup>10</sup> (Modèle – Vue – Contrôleur) en séparant au maximum la vue des différents contrôleurs, le modèle étant ici, les données issues de la simulation Matlab. Ci-dessous, une partie du diagramme des classes représentant les 3 classes essentielles qui sont :

- Le module MatReader qui permet de lire le résultat d'une simulation Matlab et d'en extraire toutes les données.
- Le module Plot qui permet d'afficher les données des différents signaux à partir des données stockées dans le module MatReader. Il permet aussi la sélection des données que l'on veut tester dans le fichier de tests unitaires.
- Le module PtuWriter qui permet d'écrire le fichier de tests unitaires selon une structure donnée.

<sup>3</sup> Méthode de développement. Voir l'ensemble des étapes en [annexe III](#).

<sup>4</sup> Environnement de programmation. Voir [glossaire](#).

<sup>5</sup> Langage de programmation.

<sup>6</sup> Ensemble des besoins. Voir [annexe I](#).

<sup>7</sup> Langage de modélisation graphique. Voir [glossaire](#).

<sup>8</sup> Diagramme des cas d'utilisation. Voir [annexe II-A](#). Voir [glossaire](#).

<sup>9</sup> Diagramme des classes. Voir [annexe II-C](#). Voir [glossaire](#).

<sup>10</sup> Méthode d'analyse et conception. Voir [glossaire](#).



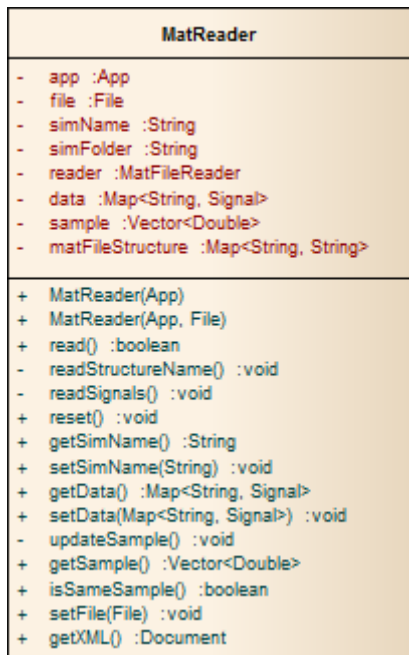


Figure 13 : Classe MatReader

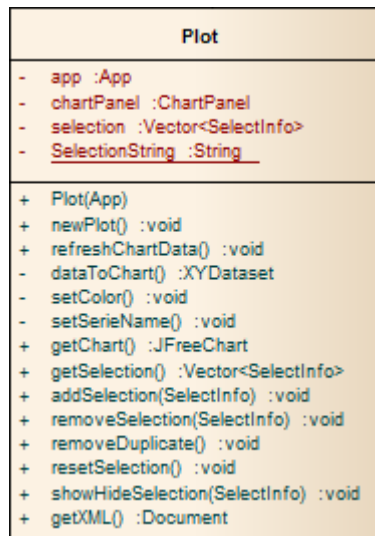


Figure 14 : Classe Plot

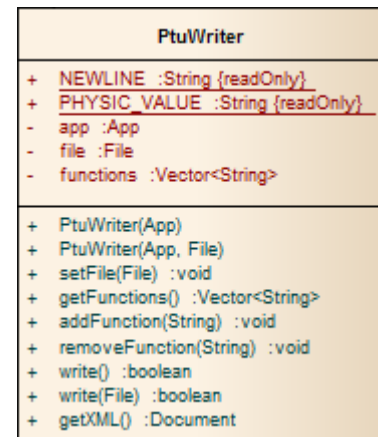


Figure 15 : Classe PtuWriter

Ci-dessus, nous pouvons voir un descriptif des 3 modules principaux, avec leurs noms, leurs attributs, et leurs méthodes.

Dans le diagramme de séquence<sup>11</sup>, j'ai détaillé le déroulement de l'utilisation de l'application avec les interactions entre, principalement, la vue et les modules MatReader, Plot et PtuWriter.

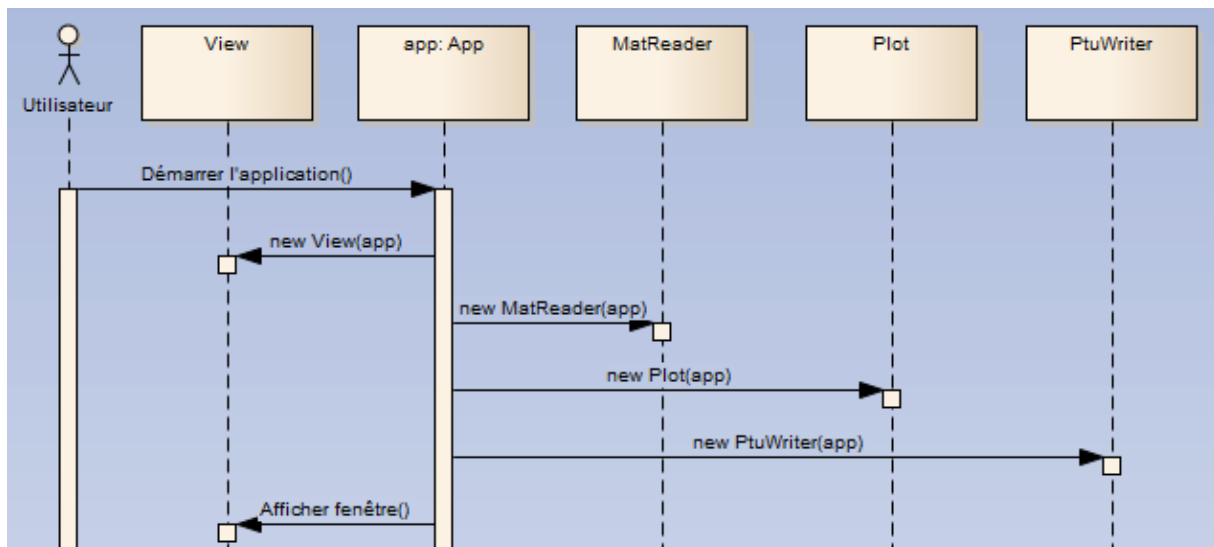


Figure 16 : Extrait du diagramme de séquence

Ci-dessus, nous pouvons voir que la première action de l'utilisateur va permettre d'initialiser l'application et afficher la fenêtre principale.

<sup>11</sup> Diagramme de séquence. Voir [annexe II-D](#). Voir [glossaire](#).

Ensuite nous avons le diagramme de séquence système<sup>12</sup> qui permet d'encapsuler l'application au travers d'un seul élément : le système. Nous avons ainsi toutes les interactions entre l'utilisateur et le programme.



Figure 17 : Extrait du diagramme de séquence système

Et enfin, la maquette<sup>13</sup> qui permet de donner une idée de l'interface graphique de l'application.

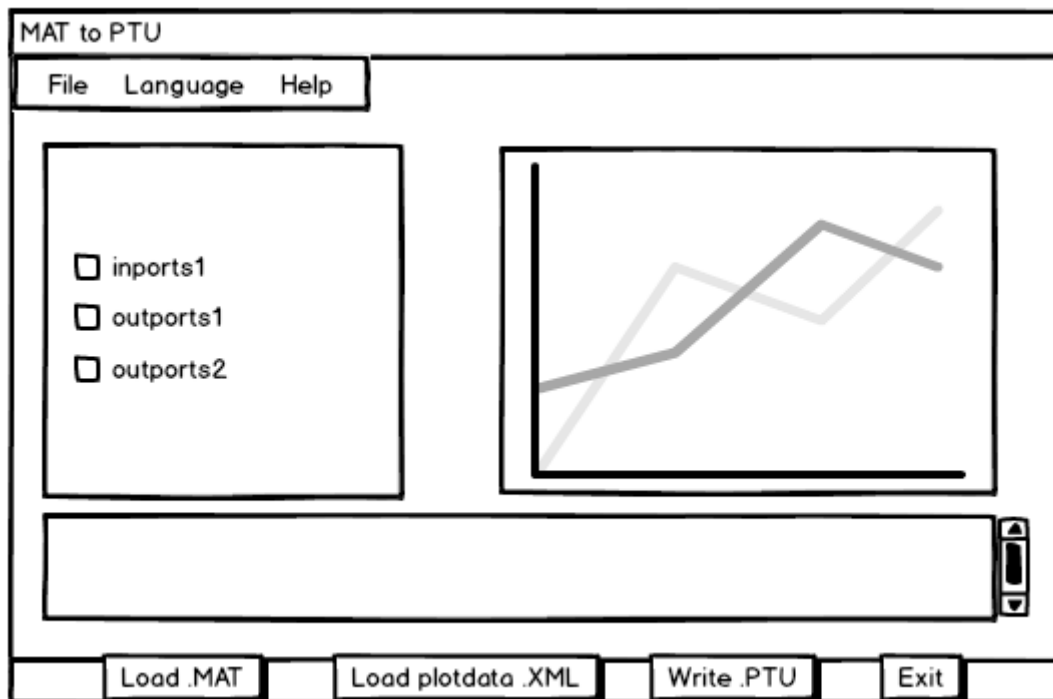


Figure 18 : Maquette de l'application

<sup>12</sup> Diagramme de séquence système. Voir [annexe II-E](#). Voir [glossaire](#).

<sup>13</sup> Maquette. Voir [glossaire](#).

## 2. Développement de l'outil MAT2PTU

### a. Utilisation de bibliothèques existantes

#### JMatIO

Cette bibliothèque permet de lire un fichier Matlab, elle permet également d'en écrire mais ce ne sera pas nécessaire dans ce projet. Après de nombreux tests, je me suis aperçu que la bibliothèque stoppait la lecture lorsqu'elle rencontrait un caractère UTF-16 / UTF-32 (par exemple : °). J'ai finalement réussi, après étude du code source de la bibliothèque, à le modifier afin qu'elle prenne en charge ces caractères. Ci-dessous, nous pouvons voir un des tests de prise en main de la bibliothèque :

```
Lecture du fichier : Results_Sim_SfSpcDrv_drtqexx__01_drtqexx.mat
Nombre de variables : 40

Lecture des structures inports :
Event_sta_pws
Vbx_sfty_dyn_mod_req
Vxx_sfty_driv_pwt_sp
Vxx_sfty_min_driv_tqe
Vxx_sfty_n
Vxx_sfty_tco

Lecture Vxx_sfty_n : Enreg_Vxx_sfty_n.mat
{Enreg_Vxx_sfty_n=[2x126 double array]}
0.0      0.04      0.08      0.12      0.16      0.2       0.24      0.28      0.32      0.36      0.4       0.44      0.48
2848.0   2784.0   2752.0   2688.0   2624.0   2560.0   2496.0   2496.0   2528.0   2528.0   2560.0   2560.0   2592.0
```

Figure 19 : Résultat d'un test de la bibliothèque JMatIO

Dans cet exemple, le programme lit le fichier Matlab spécifié en entrée, affiche le nombre de variables qu'il contient et lit la structure qui contient tous les noms des entrées. Puis, il lit le fichier Matlab contenant le vecteur de données correspondant à une entrée.

#### JDOM

Cette bibliothèque permet de manipuler facilement les fichiers XML<sup>14</sup> en JAVA.

```
Document document = new Document();
Element parent = new Element("parent");
Element enfant = new Element("enfant");
enfant.addContent("contenu");
enfant.setAttribute("attribut", "valeur");
parent.addContent(enfant);
document.addContent(parent);
try {
    XMLOutputter sortie =
        new XMLOutputter(Format.getPrettyFormat());
    sortie.output(document, System.out);
} catch (java.io.IOException e){}
```

Figure 20 : Test de génération de fichier XML

```
<?xml version="1.0" encoding="UTF-8"?>
<parent>
  <enfant attribut="valeur">contenu</enfant>
</parent>
```

Figure 21 : Fichier XML en sortie

<sup>14</sup> XML pour eXtensible Markup Language, langage de balisage extensible. Voir [glossaire](#).

## JFreeChart

Cette librairie permet de créer des graphiques de plusieurs types.

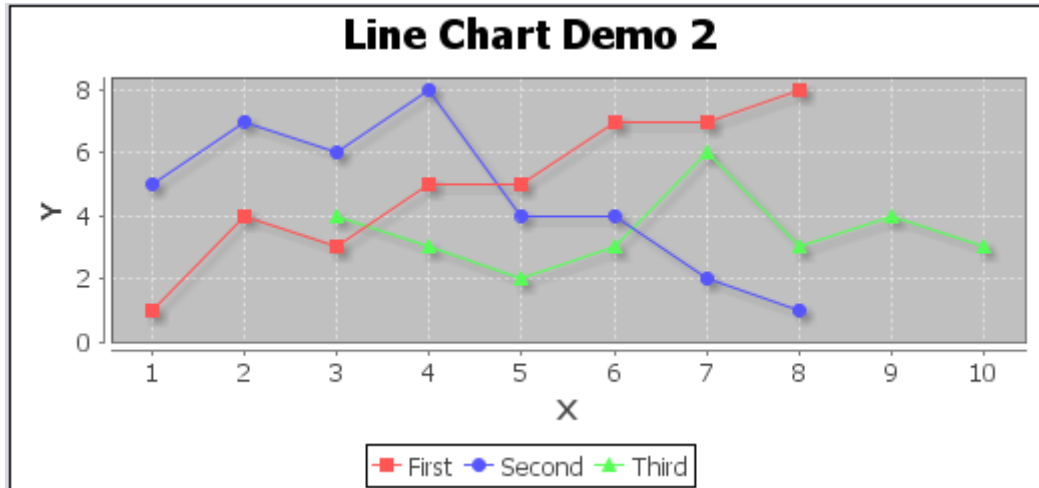


Figure 22 : Exemple d'utilisation de JFreeChart

Ces 3 librairies ont été trouvées grâce à des recherches sur internet :

- JMatIO : <http://sourceforge.net/projects/jmatio/>
- JDOM : <http://www.jdom.org/>
- JFreeChart : <http://sourceforge.net/projects/jfreechart/>

## b. Création de l'arborescence du projet

Séparation des modules à l'aide de packages<sup>15</sup>.

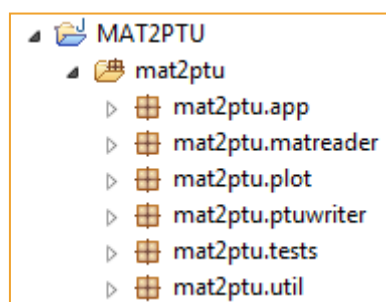


Figure 23 : Packages

## c. Module App

Ce module comprend les classes :

- App : Permet de communiquer d'un module à un autre.
- Action : Contient des fonctions permettant de réaliser les actions déclenchées par la vue.
- View : Fenêtre principale avec la gestion des événements.

<sup>15</sup> Un package est un groupe contenant une ou plusieurs classe.

### Classe App

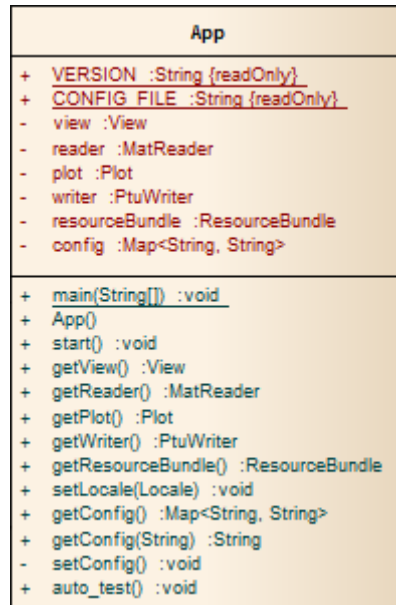


Figure 24 : Classe App

Cette classe permet de garder le lien entre la vue et les modules principaux de l'application. Elle permet d'obtenir des chaînes de caractères traduites selon une langue définie, et les différentes variables de configuration lues dans un fichier XML.

A l'heure actuelle, la définition de la langue est implantée uniquement pour les menus. Par manque de temps, le choix de la langue est donc désactivée, la langue par défaut est l'anglais. Il reste possible cependant de définir une méthode dans chaque module, de l'exécuter et de rafraichir la fenêtre pour ainsi avoir une application multi-langues.

L'application dispose d'un fichier de configuration<sup>16</sup> pour permettre une certaine flexibilité et modifier les paramètres. Le fichier de configuration est un fichier dans un format XML reposant sur la structure suivante :

```

<MAT2PTU type="MAT2PTU_CONFIG">
  <CONFIG>
    <CONFIG1>Valeur1</CONFIG1>
    <CONFIG2>Valeur2</CONFIG2>
  </CONFIG>

  <STRUCTURE type="MATLAB_RENAULT">
    ...
  </STRUCTURE>
</MAT2PTU>
    
```

Il sera possible, dans une future version, de créer un menu pour modifier ces valeurs directement dans l'application.

<sup>16</sup> Retrouvez le fichier de configuration en entier en [annexe V](#).

### Classe Action

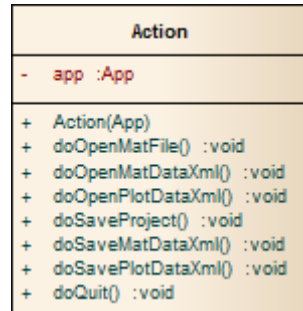


Figure 25 : Classe Action

Cette classe permet d'effectuer principalement les actions suite à l'action d'un bouton du menu principal. Elle permet par exemple d'ouvrir un fichier Matlab en appelant différentes méthodes sur les modules MatReader, View et Plot.

### Classe View



Figure 26 : Classe View

La classe View permet de construire l'architecture de la fenêtre principale. Elle respecte des contraintes selon sa dimension.

Elle permet également de répondre à tous les événements, puisque, pour un événement en JAVA, il faut associer une classe pour que son action puisse être exécutée. Par exemple, les actions du menu de l'application sont exécutées par une méthode de la classe View.

En JAVA, les interfaces graphiques avec Swing (Bibliothèque graphique intégrée à JAVA), ne sont pas toujours très pratiques à définir lorsqu'on commence à avoir des interfaces complexes. Pour réaliser l'interface graphique de l'application, j'ai, en premier lieu, mis en place le positionnement avec un système de contraintes (le GridBagLayout) pour finalement changer vers un positionnement avec des cadres redimensionnables (JSplitPane).



Figure 27 : Test de positionnement des éléments avec le GridBagLayout

Le cadre gris clair contient les noms des différents signaux avec leurs types (calibration, entrée, sortie, interne). Avec la possibilité d'afficher ou non ces signaux.

Le cadre gris foncé contient le graphique représentant l'ensemble des variables d'entrées, de sorties et internes en fonction du temps.

Le cadre blanc contient les informations de sélections, avec possibilité de signaler l'emplacement de cette sélection, de la supprimer et de la configurer.

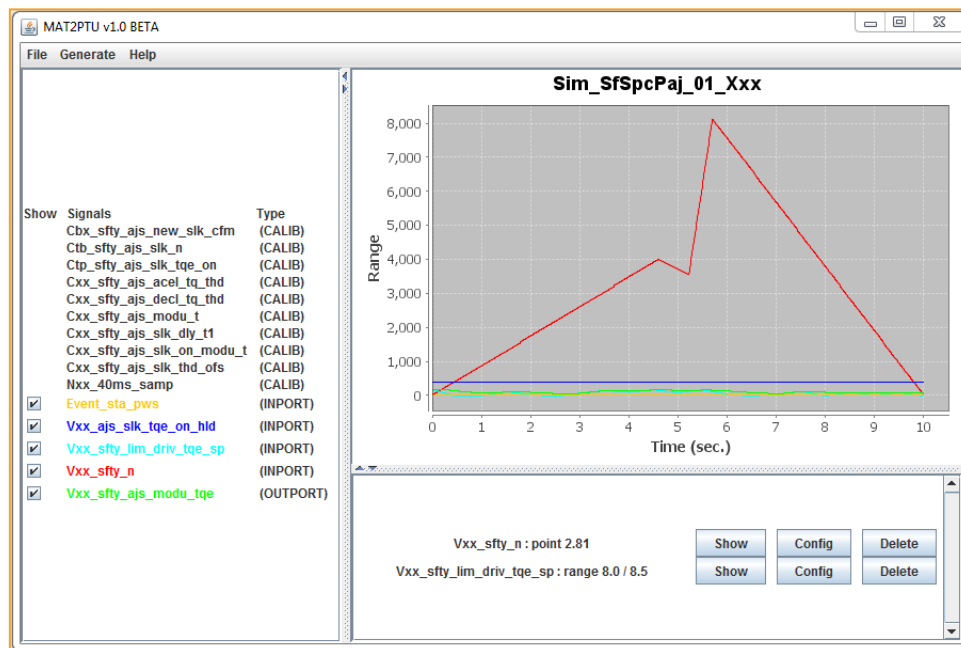


Figure 28 : Interface de l'application

Le système de positionnement avec des cadres redimensionnables est beaucoup plus facile à mettre en place que le système de contraintes. Il n'y a que deux paramètres à modifier pour le système retenu, la position d'origine par rapport au composant du dessus ou de sa gauche et la proportion d'extension de chaque cadre quand on redimensionne la fenêtre.

#### d. Module Util

Ce module comprend les classes utiles au bon fonctionnement du programme, il comprend :

- ChartColor : Contient les couleurs à assigner pour le graphique.
- Signal : Contient les informations d'un signal.
- Data : Contient des couples de temps et valeurs.
- SelectInfo : Utile pour la sélection des données et la configuration des cas de test.
- XmlParser : Permet de lire et écrire des fichiers XML.

#### Classe ChartColor

J'ai créé cette classe pour définir les couleurs qui peuvent être utilisées par le graphique car, de base, la librairie JFreeChart, qui permet d'afficher des graphiques, définit elle-même les couleurs, et certaines ne sont pas très visibles.



Figure 29 : Exemple d'une couleur presque invisible

Cette classe ne contient qu'une seule méthode intToColor qui prend un entier compris entre 0 et 21, pour chaque entier la méthode retourne une couleur différente. Pour les entiers supérieurs à 21, la classe retourne une couleur aléatoire.

#### Classe Signal

Signal	
+	UNKNOWN :String {readOnly}
+	INPORT :String {readOnly}
+	OUTPORT :String {readOnly}
+	CALIB :String {readOnly}
+	NVMY :String {readOnly}
+	INTERN :String {readOnly}
-	name :String
-	type :String
-	unit :String
-	lsb :double
-	data :Data
-	color :Color
+	Signal(String, String, String, double)
+	Signal(String, String, String, double, Color)
+	getName() :String
+	getType() :String
+	getUnit() :String
+	getLSB() :double
+	getData() :Data
+	length() :int
+	getColor() :Color
+	setColor(Color) :void
+	toString() :String

Figure 30 : Classe Signal

Toutes les informations d'un signal, qu'il soit entrée, sortie, interne ou calibration sont stockées dans cette classe, son nom, type, unité, lsb (explication dans la [partie III-2-g](#)), ainsi que les données (temps et valeur associés) à l'aide de la classe Data.



### Classe Data

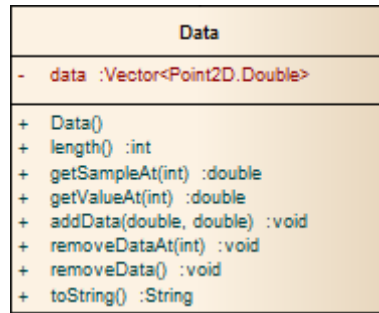


Figure 31 : Classe Data

Utilisée par un signal, elle permet de stocker les données (temps et valeurs) sous forme d'un vecteur de points x et y.

### Classe SelectInfo

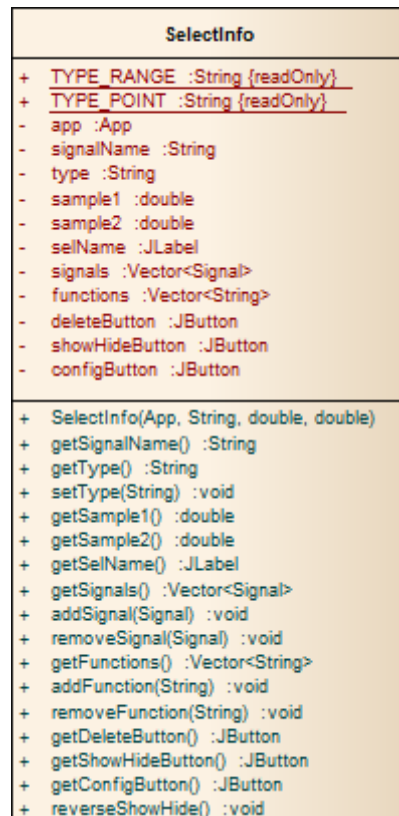


Figure 32 : Classe SelectInfo

Permet d'obtenir des informations pour une sélection donnée, qu'elle soit de type point ou de type plage.

Les événements des boutons qui la composent : suppression, affichage et configuration sont associés à cette classe pour facilement identifier la sélection et interagir avec elle.

Ces trois classes, Signal, Data, SelectInfo comportent uniquement des accesseurs et mutateurs simples, qui permettent, respectivement, d'accéder aux variables et de modifier ces variables.

### Classe XmlParser

XmlParser
<ul style="list-style-type: none"> <li>- app :App</li> <li>- file :File</li> <li>- data :Document</li> </ul>
<ul style="list-style-type: none"> <li>+ XmlParser(App)</li> <li>+ XmlParser(App, File)</li> <li>- read() :void</li> <li>+ getConfig() :Map&lt;String, String&gt;</li> <li>+ readSimName() :String</li> <li>+ readMatFileStructure(String) :Map&lt;String, String&gt;</li> <li>+ readMatFileData() :Map&lt;String, Signal&gt;</li> <li>+ readPlotData() :Vector&lt;SelectInfo&gt;</li> <li>+ getData() :Document</li> <li>+ setData(Document) :void</li> <li>+ setFile(File) :void</li> <li>+ write(File) :boolean</li> <li>+ write(File, Document) :boolean</li> </ul>

Figure 33 : Classe XmlParser

Permet de lire et écrire des fichiers au format XML à l'aide de la librairie JDOM.

Les fichiers XML comme un projet MAT2PTU, les valeurs des fichiers Matlab et les valeurs de sélection reposent sur un même schéma :

```
<MAT2PTU type="MAT2PTU_PROJECT" simName=" " version=" ">
  <READER>
    <SignalName1 type=" " unit=" " lsb="">
      <SAMPLE>...</SAMPLE>
      <VALUE>...</VALUE>
    </SignalName1>
    <SignalName2 type=" " unit=" " lsb="">
      <SAMPLE>...</SAMPLE>
      <VALUE>...</VALUE>
    </SignalName2>
  </READER>
  <PLOT>
    <SELECTION signalName="SignalName1" type="POINT" sample1="x" sample2="x" />
    <SELECTION signalName="SignalName2" type="RANGE" sample1="x" sample2="y" />
  </PLOT>
  <FUNCTION>
    ...
  </FUNCTION>
</MAT2PTU>
```

### e. Module MatReader

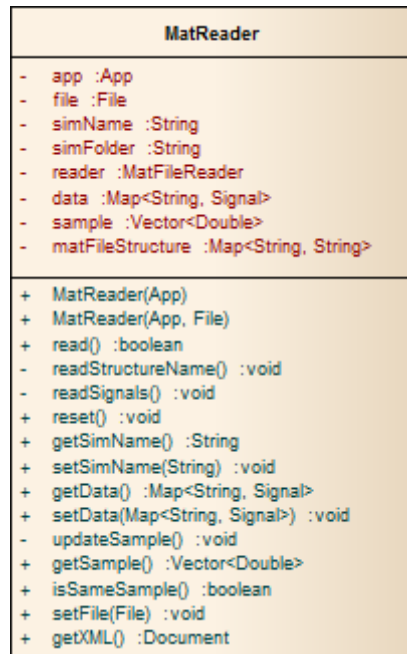


Figure 34 : Classe MatReader

Un des modules essentiels, capable de lire des fichiers Matlab grâce à la librairie jMatIO. Une simulation provenant de Renault dispose d'une architecture particulière, avec par exemple un emplacement des valeurs des signaux dans un dossier différent, c'est pourquoi j'ai rajouté dans le fichier de configuration, la description de la structure Renault<sup>17</sup>, elle se présente sous cette forme :

```
<STRUCTURE type="MATLAB_RENAULT">
  <INPUTS>ScalingInPorts</INPUTS>
  <OUTPUTS>ScalingOutPorts</OUTPUTS>
  <CALIBS>ScalingCalibs</CALIBS>
  <NVMY>ScalingNVMY</NVMY>

  <PREFIX_SIM>Results_</PREFIX_SIM>
  <SUFFIX_SIM />

  <FOLDER_SIGNAL>MIL</FOLDER_SIGNAL>
  <PREFIX_SIGNAL>Enreg_</PREFIX_SIGNAL>
  <SUFFIX_SIGNAL />
</STRUCTURE>
```

Cela permet de rajouter une structure, par exemple une simulation provenant de Continental, dans le fichier de configuration de l'application sans avoir à modifier l'application elle-même. Cette propriété n'a pas encore été testée par l'absence de projet non issu de Renault.

<sup>17</sup> Explication de l'utilisation de fichier XML en [annexe VII](#).

## f. Module Plot

Plot
<ul style="list-style-type: none"> <li>- app :App</li> <li>- chartPanel :ChartPanel</li> <li>- selection :Vector&lt;SelectInfo&gt;</li> <li>- <u>SelectionString :String</u></li> </ul>
<ul style="list-style-type: none"> <li>+ Plot(App)</li> <li>+ newPlot() :void</li> <li>+ refreshChartData() :void</li> <li>- dataToChart() :XYDataset</li> <li>- setColor() :void</li> <li>- setSerieName() :void</li> <li>+ getChart() :JFreeChart</li> <li>+ getSelection() :Vector&lt;SelectInfo&gt;</li> <li>+ addSelection(SelectInfo) :void</li> <li>+ removeSelection(SelectInfo) :void</li> <li>+ removeDuplicate() :void</li> <li>+ resetSelection() :void</li> <li>+ showHideSelection(SelectInfo) :void</li> <li>+ getXML() :Document</li> </ul>

Figure 35 : Classe Plot

Classe essentielle également puisqu'elle construit le graphique avec l'utilisation de la librairie JFreeChart pour pouvoir ainsi l'afficher. Elle gère aussi tout ce qui concerne la sélection grâce à la classe SelectInfo vue précédemment.

Au niveau de la sélection, j'ai vérifié l'intégrité de celle-ci par une fenêtre temporaire qui s'actualise toutes les secondes et qui affiche la sélection stockée dans le module Plot.

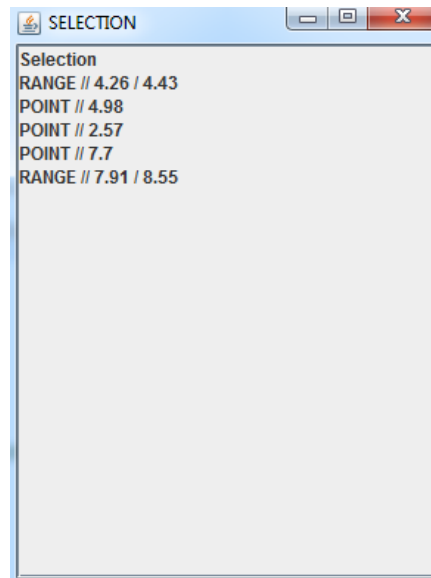


Figure 36 : Affichage de la sélection

Ce qui m'a permis de corriger la sélection en corrigeant certains points comme par exemple, la sélection de mêmes points qui crée des sélections en double.

Toujours au niveau de la sélection, j'ai écrit un petit algorithme exécuté après chaque insertion de sélection permettant de supprimer les sélections en double.

```
POUR i DE 1 à nombre(selection) FAIRE
    POUR j DE 0 à i FAIRE
        Selection1 <- selection[ i ]
        Selection2 <- selection[ j ]

        SI type(Selection1) = type(Selection2) ALORS // type : POINT ou PLAGE
            SI sample1(Selection1) = sample1(Selection2)
                ET sample2(Selection1) = sample2(Selection2) ALORS
                    supprimer(Selection1)
                    j <- i
                    i <- i - 1
            FIN_SI
        FIN_SI
    FIN_POUR
FIN_POUR
```

### g. Module PtuWriter

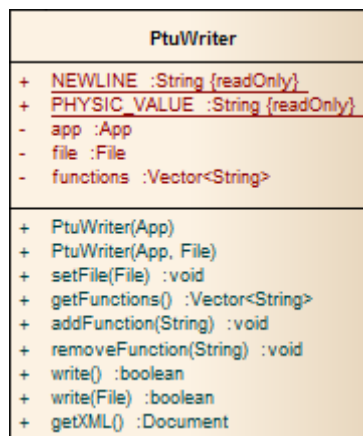


Figure 37 : Classe PtuWriter

Ce module gère la dernière étape, la génération du fichier de Tests Unitaires (.ptu).

La génération du fichier s'appuie sur un modèle de fichier de Tests Unitaires, l'application remplace certaines parties du fichier par des données permettant de créer les différents cas de test avec les données présentes dans le module MatReader et les cas de test à partir des sélections dans le module Plot.

Avant la fin du stage, une des exigences fonctionnelles sera implantée. Elle concerne la marge d'erreur permise à l'exécution des tests. Cette marge est calculée à partir du lsb (Low Significant Bit, pour bit de poids faible). Ce lsb est une valeur qui permet de passer de la valeur physique à la valeur codée.

Exemple avec le signal N, le régime moteur. Il peut être dans la plage de valeurs suivante : 0 tours/minute à 8160 tours/minute.

Pour coder ces valeurs sur un octet (8 bits), un octet pouvant coder un entier de 0 à 255, il faut passer par la formule suivante :  $x_{physique} = a(x_{codé} + b)$ . Avec a : la valeur du lsb et b : la valeur de décalage si le signal a des valeurs négatives.

Pour l'exemple, a = 32 et b = 0. Ainsi on obtient bien  $\frac{8160}{32} = 255$ .

### h. Récapitulatif des flux d'entrées et de sorties de l'application

Ci-dessous, le schéma résume l'ensemble des échanges entre l'application et les fichiers de type MAT, XML et PTU :

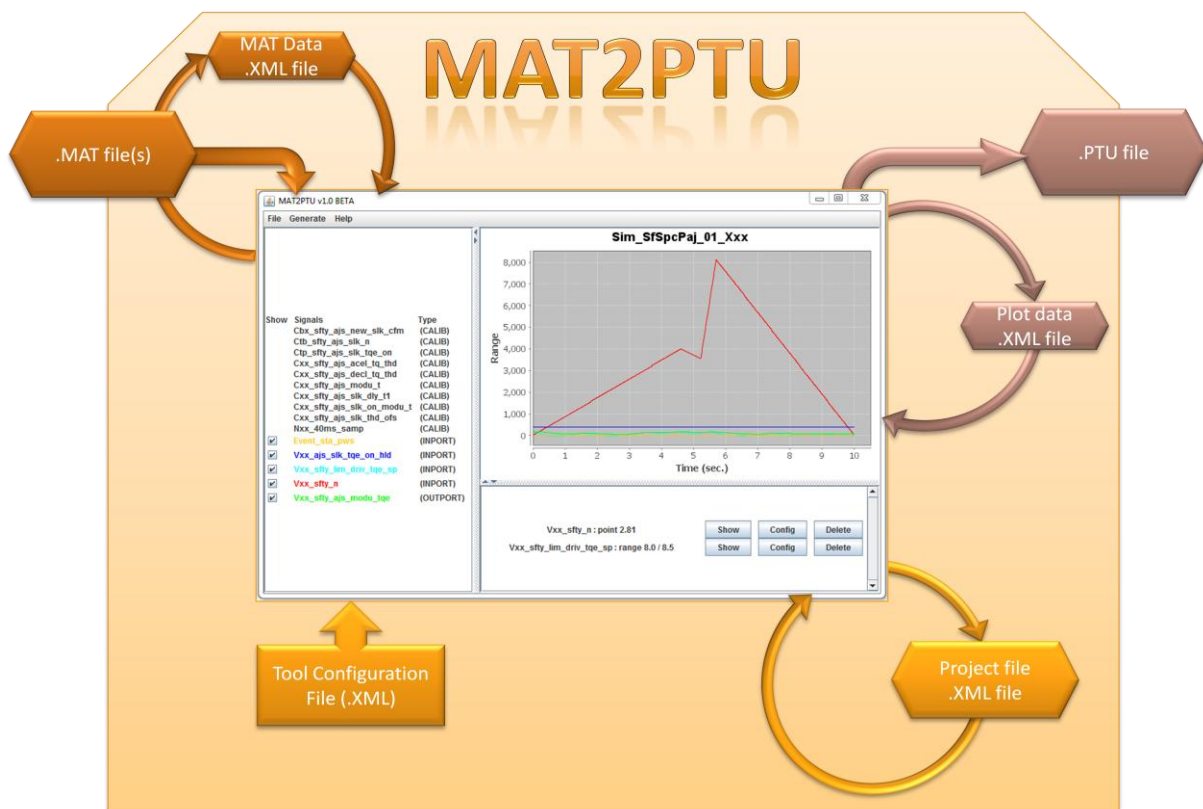


Figure 38 : Ensemble des échanges entre l'outil et les fichiers MAT, XML et PTU

### i. Manuel utilisateur

Le manuel utilisateur doit être rédigé le plus clairement possible pour permettre aux utilisateurs d'utiliser toutes les fonctionnalités d'une application. De plus, la langue officielle du groupe Continental est l'anglais, le manuel devra donc être rédigé dans cette langue.

*Note* : ce manuel n'est pas encore écrit, il le sera avant la fin du stage.

### 3. Tests de l'application

#### a. Performances

L'application, après de nombreux tests, obtient de très bonnes performances.

En temps normal, les enregistrements d'une simulation peuvent durer 5 ou 10 secondes. L'ouverture des fichiers Matlab et l'affichage du graphique se fait quasi instantanément, la sélection des différents points/plages de valeur également, de même pour la génération du fichier .ptu.

Avec un extrême de 50 000 secondes d'enregistrements, ce qui correspond à 13 heures, l'ouverture et l'affichage du graphique prend 40 secondes à 1 minute. C'est un exploit comparé au logiciel Matlab qui ne peut pas afficher graphiquement cette simulation. L'application prend environ 4 à 5 Go de mémoire ce qui est énorme comparé à une utilisation d'environ 60 Mo en temps normal. Sachant que sans la création du graphique, l'application utilise 1 Go de mémoire.

#### b. Tests unitaires des modules de l'application

Le test unitaire permet de vérifier le bon fonctionnement d'une partie d'un programme, communément appelée module. Ces tests sont exécutés pour vérifier le bon fonctionnement d'un programme. Ils doivent être rejoués après chaque modification du code de l'application afin d'éviter une régression de celle-ci.

Les tests unitaires reposent sur les assertions, une assertion est une expression qui doit être évaluée à vrai. Un exemple : `assertMatch(valeur-attendue, valeur-obtenu)`.

Ces tests doivent couvrir un maximum de modules d'une application, mais tout n'est pas testable, comme par exemple, les interfaces graphiques. On parle alors de couverture de tests. Un test unitaire couvre un module en particulier, après ces différents tests unitaires, on réalise ce qu'on appelle des tests d'intégrations, ces tests regroupent plusieurs modules, et permet de vérifier, à partir des données d'entrée, qu'on obtient bien le résultat attendu.

*Note* : ces tests ne sont pas encore écrits à l'heure actuelle, ils seront normalement écrits avant la fin du stage.

#### 4. Livraison

Avant la fin du stage, je vais effectuer, ce qu'on appelle en informatique, la recette. La recette est la dernière étape du cycle en V, elle assure la conformité de l'application vis-à-vis des exigences définies dans la spécification.

Pour valider la recette, le client procède à sa propre série de tests de validation, ces tests vont permettre de vérifier que les exigences fonctionnelles fournies dans la spécification ont bien été respectées.

Je dois également présenter l'application devant une dizaine de personnes qui vont par la suite utiliser cette application.



## 5. Etat de l'outil

### a. Fonctionnalités

L'outil est pleinement fonctionnel, aucun bug majeur n'est connu à ce jour. Il respecte toutes les requêtes requises et certaines optionnelles fournies dans la spécification.

L'application a été conçue pour être totalement fonctionnelle avec des fichiers XML. Ainsi, si une évolution des fichiers Matlab survenait et que Continental manquait d'effectif pour corriger l'application, il suffirait de créer un script Matlab qui génère un fichier XML avec les données de la simulation et l'application pourra de nouveau fonctionner.

### b. Evolutions futures

Quelques évolutions possibles pour apporter des nouvelles fonctionnalités ou également pour améliorer l'application.

Sont possibles par exemple :

- Laisser la possibilité à l'utilisateur de choisir la couleur de chaque signal par une palette de couleur.
- Mise en place d'un menu pour pouvoir modifier la configuration de l'application avec une option pour restaurer la configuration par défaut.
- Possibilité d'un outil annexe pour fusionner plusieurs projets MAT2PTU afin de produire un seul fichier .ptu par la suite.
- Une intégration en tant que plugin dans Eclipse (Requête optionnelle de la spécification).

## IV – Bilan

### 1. Technique

L'aspect technique est très convenable, ma formation m'ayant pleinement préparé pour concevoir entièrement une application.

Ce stage confirme l'idée qu'une étude d'analyse et conception est essentielle dans le développement d'une application, qui est de toute manière obligatoire lorsqu'on suit les étapes d'un cycle en V. D'une part pour être organisée avec les différents niveaux d'abstraction que comportent des méthodes telles que MERISE ou UML, mais également pour faciliter le développement du logiciel, sa maintenabilité, son utilisation ainsi que son évolution. L'utilisation de langage graphique tel que l'UML a d'ailleurs été très bien accueillie.

J'ai rencontré quelques difficultés notamment au niveau de l'utilisation d'une librairie qui ne prenait pas en charge les caractères UTF-16 / UTF-32. Finalement, après récupération du code source et modification, j'ai pu corriger la lecture de ces types de caractères ce qui m'a permis d'acquérir quelques notions sur la lecture de fichier bas niveau avec le langage JAVA.

### 2. Humain

Cette première expérience m'a beaucoup apporté humainement. La bonne ambiance qui règne dans le service a facilité mon intégration au sein de l'entreprise.

J'ai été autonome sur l'ensemble du projet, j'ai également été force de proposition, en apportant de nombreuses idées accueillies avec beaucoup d'intérêt. J'ai pu appliquer la méthode du cycle en V, tout en utilisant l'UML pour la partie conception, ce qui permet d'accorder une confiance sur la qualité du travail fourni et ainsi être bénéfique à l'entreprise.

Ce stage était ma première expérience professionnelle, et l'avoir effectué dans une grande entreprise est un vrai plus. J'ai appris beaucoup de nouvelles choses sur le domaine de l'automobile qui est un secteur très intéressant.

## Conclusion

Ce stage a été une réussite, c'est pour moi une très bonne expérience professionnelle.

J'ai choisi Continental pour la grandeur du groupe et pour le projet auquel j'ai porté beaucoup d'intérêt. J'ai pu échanger sur de nombreux sujets avec les employés et stagiaires et ainsi obtenir de précieux conseils.

Cette expérience m'a beaucoup apporté sur le niveau professionnel, technique et humain. Elle permettra, avec les autres stages à venir, de préparer l'insertion professionnelle, de préférence dans un grand groupe comme Continental.

L'objectif de ce stage correspond parfaitement à mon choix d'orientation : le développement de logiciel. Pour cela, j'envisage d'intégrer une Licence L3 Informatique et de poursuivre par un Master Développement Logiciel.

## Glossaire

**Eclipse** : logiciel permettant de développer un projet. C'est un EDI, Environnement de Développement Intégré, il possède tous les outils nécessaires pour développer un projet.

**XML** : pour eXtensible Markup Language, langage de balisage extensible. Permet de stocker des informations à l'aide de balises. Exemple : <ANNEE>2013<ANNEE>

**UML** : pour Unified Modeling Language, langage de modélisation unifié. C'est un langage de modélisation graphique à l'aide de différents diagrammes.

**MVC** : pour Modèle-Vue-Contrôleur, est une façon de structurer une application en séparant le modèle (données, le plus souvent une base de données) de la vue (interface utilisateur) du contrôleur (événements, synchronisation).

**Diagramme des cas d'utilisation** : donne une vision globale du fonctionnement de l'application à travers un ensemble d'actions entre l'utilisateur et l'application.

**Diagramme de classes** : présente les différentes classes et leurs relations. Une classe est un ensemble d'attributs (variables) et de méthodes (fonctions). Une classe peut ensuite être instanciée et devenir, ce qu'on appelle en programmation, un objet avec lequel on peut dialoguer à travers ses fonctions.

**Diagramme de séquence** : il représente les interactions entre l'utilisateur et le système décomposé.

**Diagramme de séquence système** : il représente les interactions entre l'utilisateur et le système.

**Maquette** : Représente l'interface graphique de l'application telle qu'elle doit être développée afin de donner une idée de la finalité de l'application au client.

**Fichier Matlab .mat** : fichier contenant des variables de plusieurs types, comme une chaîne de caractères ou encore des structures.

## Références bibliographiques

Sites internet consultés :

- *JavaDoc* : Documentation en ligne sur le langage JAVA  
<http://docs.oracle.com/javase/6/docs/api/>
- *StackOverflow* : Répond aux questions sur différents problèmes que l'on peut rencontrer pendant le développement  
<http://stackoverflow.com/>
- *Balsamiq* : Permet de réaliser des maquettes en ligne  
<http://www.balsamiq.com/>
- *Normes de développement JAVA* :  
<http://jmdoudoux.developpez.com/cours/developpons/java/chap-normes-dev.php>

## Résumé / Abstract

J'ai effectué mon stage dans l'entreprise Continental. Le projet qui m'a été confié est la réalisation d'un outil capable de visualiser des simulations Matlab. Une simulation Matlab est composée de plusieurs variables, entrées, sorties, calibrations, internes, et mémoire non-volatile. Chaque variable d'entrée, de sortie, interne est évaluée toutes les 10 ou 40 millisecondes et c'est à partir de ces données qu'est composé le graphique. A partir de cette visualisation, l'utilisateur peut sélectionner des points ou des plages de valeurs pour créer des cas de test. L'utilisateur peut également configurer quelles fonctions et quels signaux seront inclus dans chaque cas de test. Après cette étape, l'application génère le fichier de Tests Unitaires (.ptu).

Ce projet débute à partir d'une spécification déjà réalisée, j'ai suivi le cycle en V, donc j'ai réalisé une étude d'analyse et conception, avec des diagrammes UML, le développement de l'application en elle-même puis prochainement les tests et la recette.

Cette expérience est une très bonne chose dans ma carrière professionnelle. J'ai pu réutiliser beaucoup de méthodes apprises en D.U.T., cette formation m'a apporté une base solide, ce qui m'a permis d'accomplir complètement ce projet. De plus, j'ai découvert une entreprise et le secteur de l'automobile. L'équipe qui m'a accueilli était très sympathique, et l'atmosphère autour du stage était agréable.

I did my internship in the worldwide company Continental. The project which was entrusted to me is an application which can visualize Matlab simulations. A Matlab simulation is composed by some variables, inputs, outputs, calibrations, interns, and non-volatile memory. Each input, output, intern is sampled every 10 or 40 milliseconds and it is from these data that the chart is composed. From this visualization of the chart, the user can select some point or some range of values to create test cases. The user can configure which functions and which signals are included in each test case. After this step, the application generates a .ptu file, which is a Tests Units file.

This project started from a specification already made, I followed the V-Model so I realized a design, with UML diagram, the development of the application itself and soon testing and recipe.

This experience was a very good thing in my professional experience. I have re-used a lot of methods learned in the D.U.T., this formation brought me to have a solid base which allowed me to fully complete the project. In other part, I discovered a company and automotive sector. The team that welcomed me was great, and the atmosphere throughout the internship was pleasant.

## Sommaries des annexes

I – Spécification .....	I
II – Diagrammes de la partie analyse et conception .....	IV
A. Diagramme des cas d'utilisations .....	IV
B. Diagramme de classes initial .....	V
C. Diagramme de classes actuel .....	VI
D. Diagramme de séquence .....	VII
E. Diagramme de séquence système .....	VIII
F. Maquette .....	IX
III – Méthode de développement : le cycle en V .....	X
IV – Planification du projet : Gantt .....	XI
V – Fichier de configuration .....	XIII
VI – Utilisation de la valeur du Isb .....	XIV
VII – Configuration d'une simulation .....	XV
VIII – Divisions du groupe Continental .....	XVI

## I – Spécification



Continental Automotive France SAS  
1, avenue Paul Ourliac – BP 83649 – F-31036  
Toulouse Cedex 1

11 Février 2013

### SPECIFICATION

Objet : Génération de fichiers de Test Unitaire (.PTU) à partir d'enregistrements MATLAB/SIMULINK (.MAT) utilisation .MAT fourni par Renault pour générer des fichiers de test unitaire .PTU à Conti

#### But :

Développer un outil permettant de générer un fichier de Test Unitaire (.PTU) à partir de fichiers issus de simulations sur MATLAB / SIMULINK (.MAT).

Le fichier .MAT contient des échantillons de signaux enregistrés lors d'une simulation du modèle MATLAB représentant le design du module SW que l'on vise à tester par la suite en Test Unitaire. Il contient donc les signaux d'Entrée du modèle, ainsi que les signaux de Sortie.

Cet outil s'inscrit dans le processus de vérification des développements BCS (bon de commande spécifique) : livraison de spec/modèle par Renault – codage et intégration par Conti.

#### Exigences fonctionnelles :

##### REQ F1 :

Visualiser graphiquement les signaux enregistrés dans les fichiers .MAT (plusieurs formats structure dans les .MAT) avec des possibilités de zoom

##### REQ F2 :

Sélectionner, parmi la liste de signaux disponibles (liste définie dans les fichiers .XML ou des .MAT), les signaux d'Entrée et les signaux de Sortie qui devront être pris en compte pour la génération des cas de test unitaires

- Les cas de tests pourront être de deux types : de type "unique", c.-à-d. un seul échantillon de données d'entrées ou de type "suite", c.-à-d. une suite de quelques dizaines d'échantillons

##### REQ F3 :

Prendre en compte les éventuels décalages temporels entre Entrées et Sorties pour la génération des cas de test unitaires

##### REQ F4 :

Permettre une gestion de la marge d'erreur dans les calculs, dûe aux conversions de types entre les signaux enregistrés en Float Double précision (dans le .MAT) et le cas de test en Virgule Fixe (.PTU)



**REQ OPTIONNEL F5 :**

Sélectionner automatiquement les signaux d'Entrée et de Sortie, en se basant sur le modèle Simulink lui-même (potentiellement à l'aide d'un rapport HTML du modèle Simulink)

**REQ OPTIONNEL F6 :**

Identifier automatiquement des plages "fixes" pendant lesquelles les signaux sont stables, et pourraient être transformés en un cas de test unitaire "unique"

**REQ OPTIONNEL F7:**

L'outil devra être compatible aussi avec les fichiers .MAT générés par Conti SDA (en plus de ceux générés par Renault targetlink ( ?))

**REQ OPTIONNEL F8 :**

Sur la base d'un .MAT et d'un PTU existant, afficher en mode superposition par donnée sur une vue « oscilloscope » les valeurs présentes dans les fichiers

**Exigences non fonctionnelles :**

**REQ NF1 :**

L'outil devra pouvoir s'intégrer facilement dans la plateforme de développement logiciel (basée sur Eclipse) ==> idéalement, il pourrait se présenter sous la forme d'un plugin Eclipse.

**REQ NF2 :**

L'outil devra être implémenté dans un langage agréé par ETV (JAVA ?), et suivant les règles de développement ETV (ETV : Engineering Tools and Validation group at CONTINENTAL)

**REQ NF3 :**

L'outil devra pouvoir supporter les éventuelles évolutions de format des fichiers .MAT générés par les différentes versions de MATLAB/SIMULINK (potentiellement en utilisant des DLL du runtime de Matlab)

**REQ NF4 :**

L'outil devra être indépendant de MATLAB/SIMULINK (il ne devra pas être lié à une licence MATLAB)

**REQ NF5 :**

L'objectif en termes de date serait de disposer d'un prototype fonctionnel fin JUIN 2013.

Questions et précisions à statuer :

Q1 :

Le fichier .MAT contient des données en représentation flottante ou virgule fixe ?

### Cas d'usages :

#### *UC1 :*

Réception d'un modèle Matlab associé à un enregistrement .MAT des entrées et des sorties du modèle

L'outil permet la visualisation sur un mode « oscilloscope » des valeurs des données enregistrées. Grâce à un « pointeur / curseur » l'utilisateur sélectionne par donnée les valeurs à extraire pour constituer le jeu de test du fichier de test unitaire. Ces valeurs sont sélectionnées en « point unique » : une valeur, soit en « suite » : plusieurs valeurs consécutives. Une action utilisateur lance la génération du fichier texte PTU associé.

#### *UC2 :*

Dans la sélection des données « en point unique », l'utilisateur choisit des données d'entrées et de sortie sur un même phasage temporel « t »

#### *UC3 :*

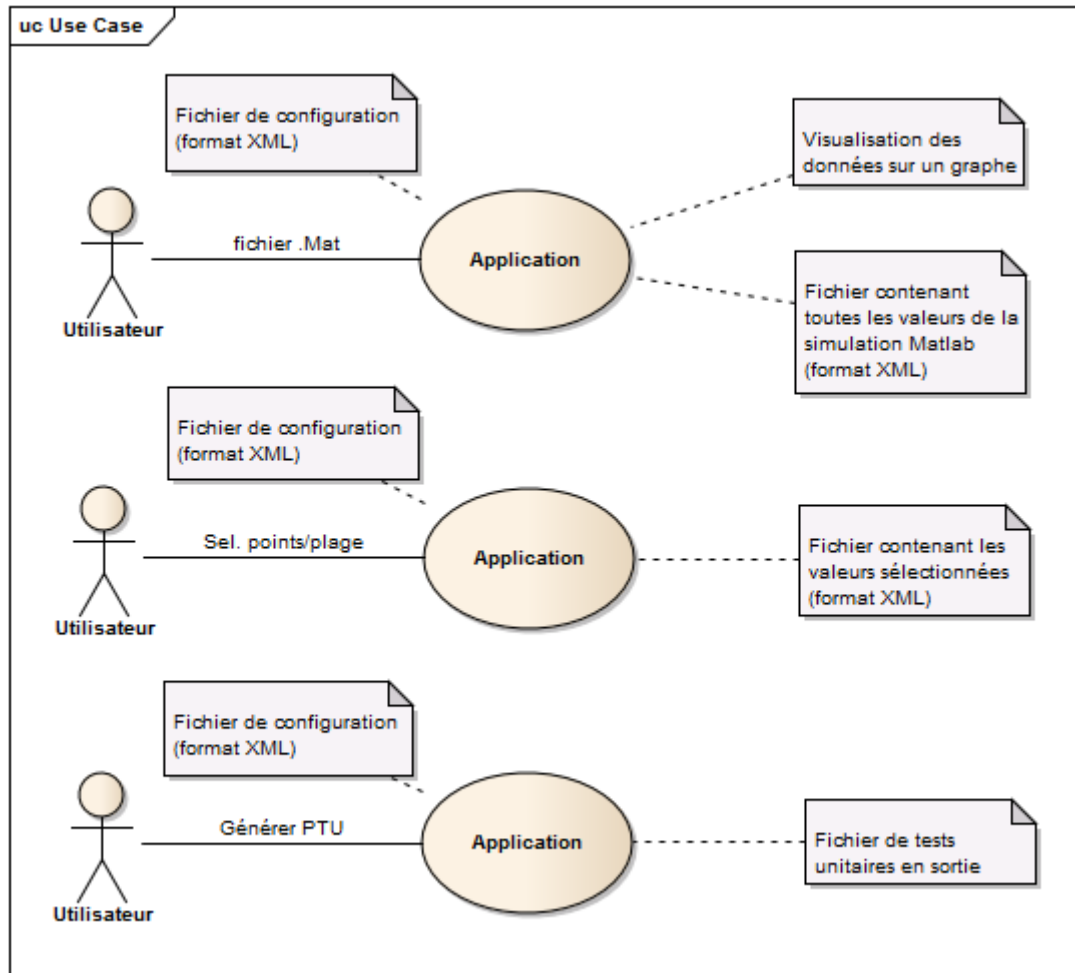
Dans la sélection des données « en point unique », l'utilisateur choisit des données d'entrées sur un phasage temporel « t » et le sorties sur un phasage « t+x »

#### *UC4 :*

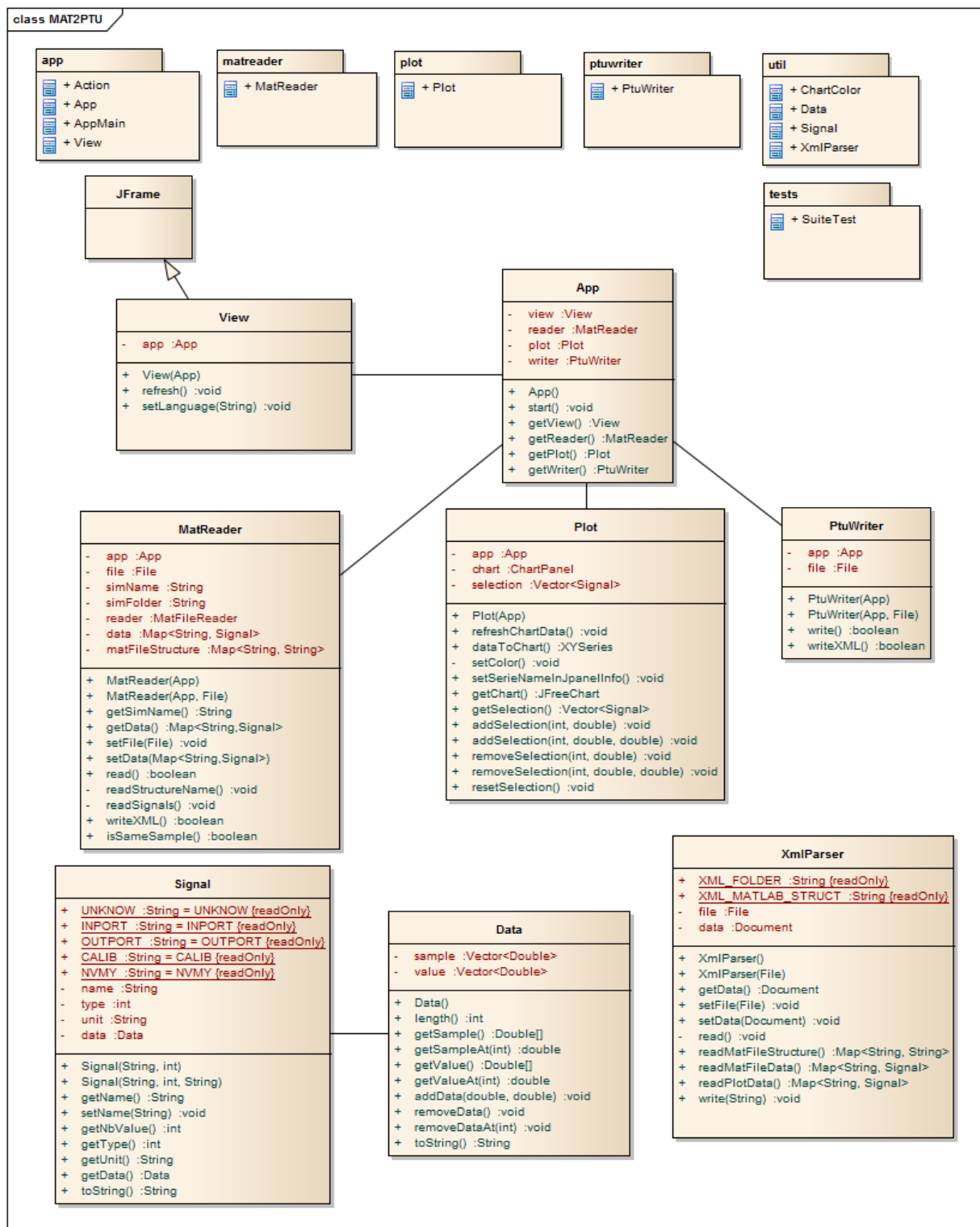
Dans la sélection des données « en suite », l'utilisateur choisit des données temporelles à retenir à l'intérieur de la suite, avec éventuellement un décalage temporel entre entrées et sorties.

## II – Diagrammes de la partie analyse et conception

### A. Diagramme des cas d'utilisations



## B. Diagramme de classes initial

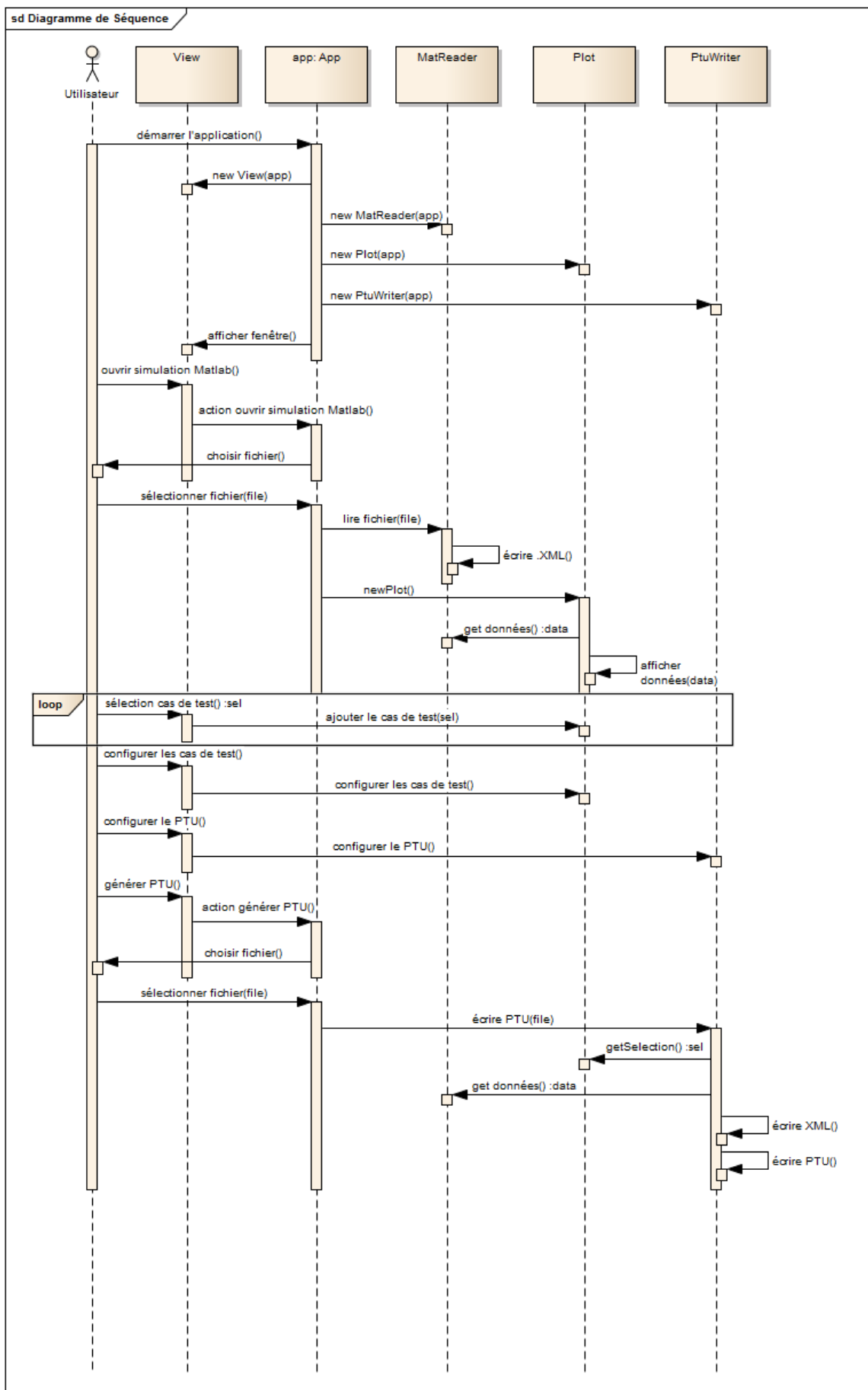


Ceci est la version initiale du diagramme de classes, l'annexe suivante présente le diagramme de classes actuel.

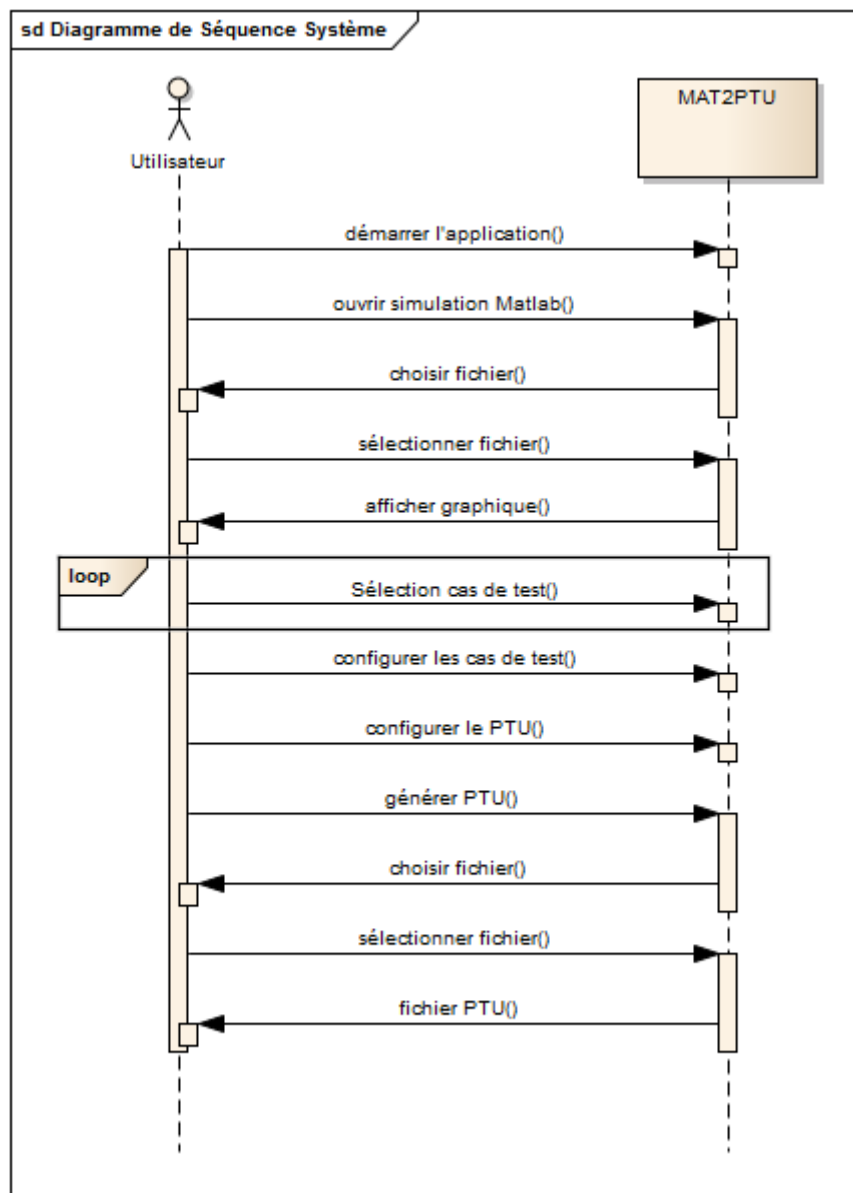
## C. Diagramme de classes actuel



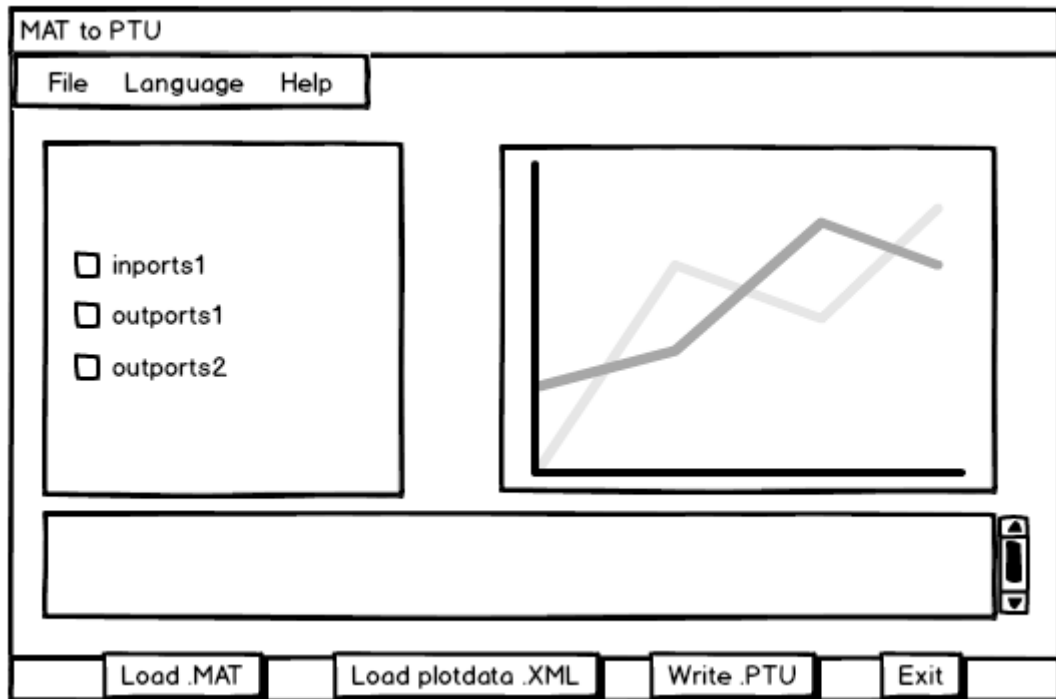
## D. Diagramme de séquence



## E. Diagramme de séquence système

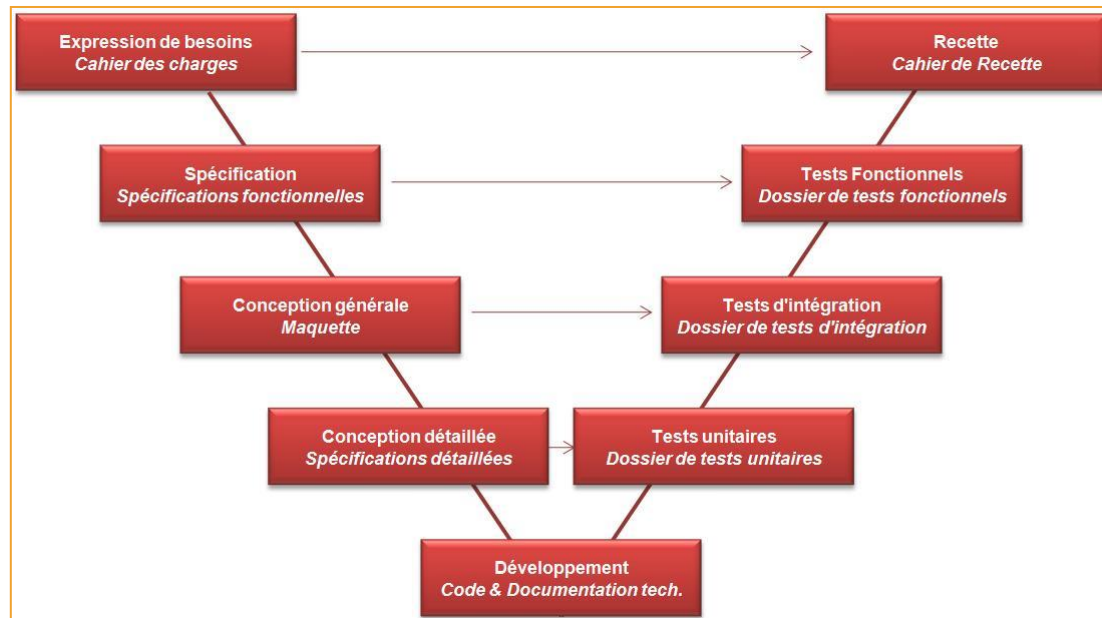


## F. Maquette





### III – Méthode de développement : le cycle en V



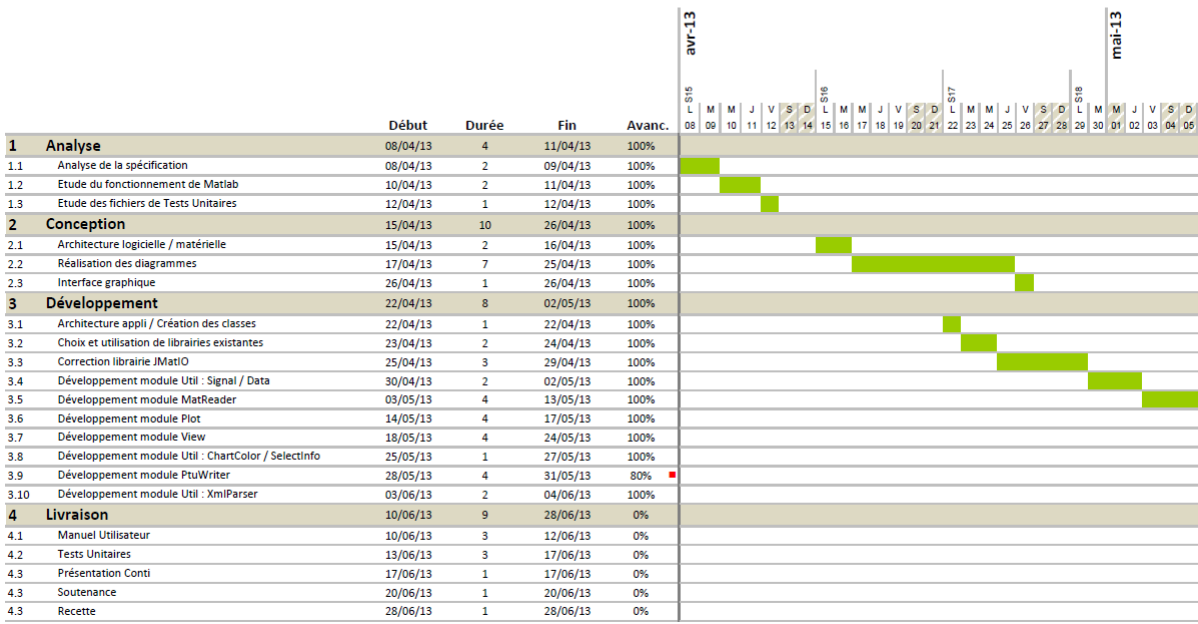
Ce graphique montre toutes les étapes qui s'enchaînent pour réaliser un projet : de l'expression de besoins à la recette.

En axe des abscisses, il y a le temps et en ordonnée la complexité. Plus on descend dans la branche du V et plus il y a de détails.

## IV – Planification du projet : Gantt

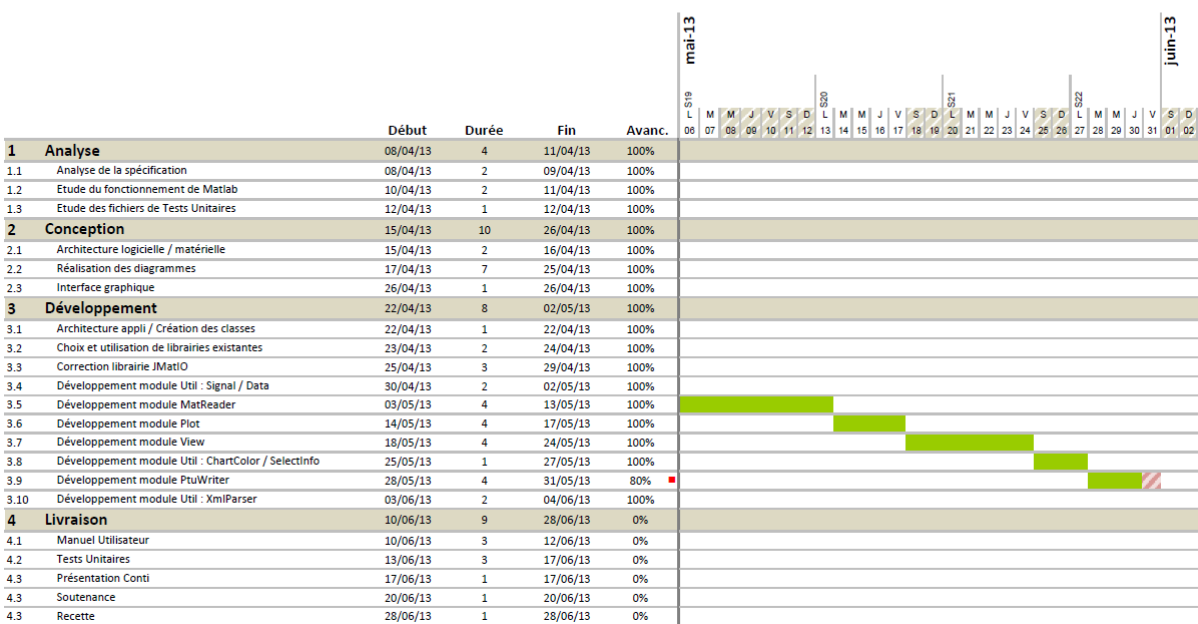
### Diagramme de GANTT

Détails du projet	
Produit	MAT2PTU
Chef de projet	Théo Vaucher
Révision	1
Commentaires	



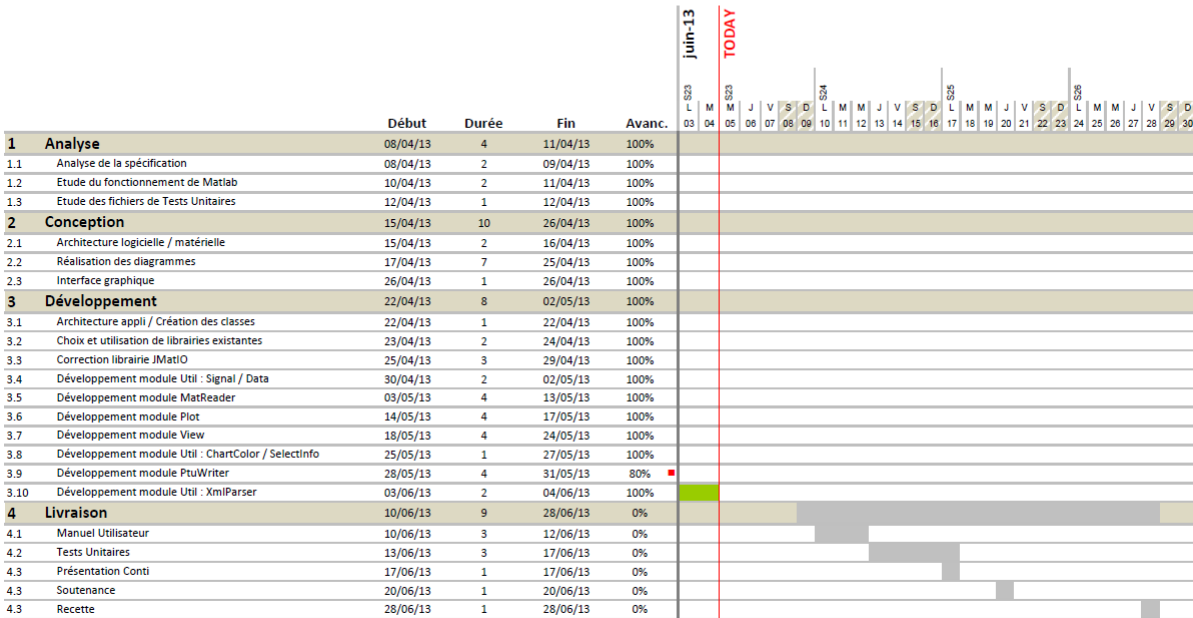
### Diagramme de GANTT

Détails du projet	
Produit	MAT2PTU
Chef de projet	Théo Vaucher
Révision	1
Commentaires	



### Diagramme de GANTT

Détails du projet	
Produit	MAT2PTU
Chef de projet	Théo Vaucher
Révision	1
Commentaires	



## V – Fichier de configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<MAT2PTU type="MAT2PTU_CONFIG">
  <CONFIG>
    <PTU_SAMPLE_SHIFT>1</PTU_SAMPLE_SHIFT>
    <WARN_TOO_MANY_VALUE_SELECTED>200</WARN_TOO_MANY_VALUE_SELECTED>
  </CONFIG>

  <STRUCTURE type="MATLAB_RENAULT">
    <INPORTS>ScalingInPorts</INPORTS>
    <OUTPORTS>ScalingOutPorts</OUTPORTS>
    <CALIBS>ScalingCalibs</CALIBS>
    <NVMY>ScalingNVMY</NVMY>
    <INTERN>ScalingIntern</INTERN>

    <PREFIX_SIM>Results_</PREFIX_SIM>
    <SUFFIX_SIM />

    <FOLDER_SIGNAL>MIL</FOLDER_SIGNAL>
    <PREFIX_SIGNAL>Enreg_</PREFIX_SIGNAL>
    <SUFFIX_SIGNAL />
  </STRUCTURE>

  <!--
  <STRUCTURE type="SDA">
  </STRUCTURE>
  -->
</MAT2PTU>
```

## VI – Utilisation de la valeur du lsb

Cette annexe explique pourquoi utiliser le lsb. Il faut savoir que les nombres dans les simulations Matlab sont codés en virgule flottante double précision tandis que dans les modules sources et les PTU, ils sont codés en virgule fixe.

Précision sur les nombres à virgule flottante et à virgule fixe :

- Virgule flottante (double précision)

Ces nombres sont stockés en mémoire sous forme de mantisse et d'exposant où la mantisse est le nombre sans la virgule et l'exposant encode la position de la virgule.

Par exemple :  $1.23 = 123 * 10^{-2}$

123 représente la mantisse

$10^{-2}$  représente l'exposant

Représentation en simple précision (32 bits):



Représentation en double précision (64 bits):



Le fait d'être double précision augmente la taille de la mantisse et de l'exposant, on obtient ainsi 16 chiffres significatifs (précision du nombre).

- Virgule fixe

En virgule fixe, le nombre de chiffres après la virgule est défini pour tous les nombres.

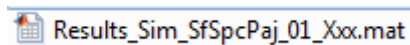
Il peut y avoir une perte de précision entre les deux notations, c'est pour cela qu'on définit une tolérance dans le PTU sur la valeur attendue.

## VII – Configuration d'une simulation

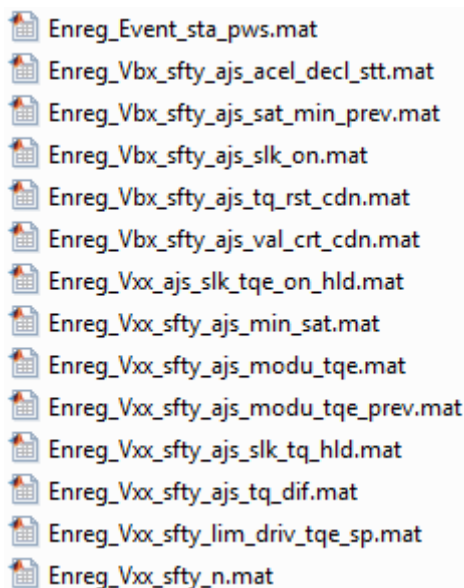
Une simulation Renault est constituée de plusieurs fichiers répartis dans différents dossiers. Pour des besoins d'évolution et d'utilisation de l'outil MAT2PTU avec des simulations provenant de Continental, j'ai fait en sorte que l'application puisse accepter différentes configurations d'architectures (sous réserve de quelques modifications éventuelles).

Pour cela, j'ai créé, dans le fichier de configuration de l'application, une partie pour stocker des descriptions de simulations. Ci-dessous, un descriptif de la composition d'un résultat de simulation :

Dossier : MIL\_SfSpcPaj\_AJMOD\Results



Dossier : MIL\_SfSpcPaj\_AJMOD\Results\MIL\Sim\_SfSpcPaj\_01\_Xxx



- Enreg\_Event\_sta\_pws.mat
- Enreg\_Vbx\_sfty\_ajs\_acel\_decl\_stt.mat
- Enreg\_Vbx\_sfty\_ajs\_sat\_min\_prev.mat
- Enreg\_Vbx\_sfty\_ajs\_slk\_on.mat
- Enreg\_Vbx\_sfty\_ajs\_tq\_rst\_cdn.mat
- Enreg\_Vbx\_sfty\_ajs\_val\_crt\_cdn.mat
- Enreg\_Vxx\_ajs\_slk\_tqe\_on\_hld.mat
- Enreg\_Vxx\_sfty\_ajs\_min\_sat.mat
- Enreg\_Vxx\_sfty\_ajs\_modu\_tqe.mat
- Enreg\_Vxx\_sfty\_ajs\_modu\_tqe\_prev.mat
- Enreg\_Vxx\_sfty\_ajs\_slk\_tq\_hld.mat
- Enreg\_Vxx\_sfty\_ajs\_tq\_dif.mat
- Enreg\_Vxx\_sfty\_lim\_driv\_tqe\_sp.mat
- Enreg\_Vxx\_sfty\_n.mat

L'application récupère le dossier à l'ouverture du premier fichier, celui contenant les informations de la simulation et les noms des signaux (grâce à la configuration fournie). Puis, toujours à l'aide de la configuration, remplace le préfixe dans le nom du fichier, ce qui permet d'obtenir le nom du sous-dossier et ainsi, avec l'ajout d'un préfixe, avoir accès aux différents signaux.

## VIII – Divisions du groupe Continental

Chassis & Safety	Powertrain	Interior	Pneus	ContiTech
Electronic Brake Systems	Engine Systems	Instrumentation & Driver HMI	Première Monte	Air Spring Systems
Hydraulic Brake Systems	Transmission	Infotainment & Connectivity	Remplacement "EMEA"	Benecke-Kaliko Group
Passive Safety & Sensorics	Hybrid Electric Vehicle	Body & Security	Remplacement "Les Amériques"	Conveyor Belt Group
Advanced Driver Assistance Systems (ADAS)	Sensors & Actuators	Comm. Vehicles & Aftermarket	Remplacement "Asie Pacifique"	Elastomer Coatings
Chassis Components	Fuel Supply		Pneus Utilitaires	Fluid Technology
			Pneus deux-roues	Power Transmission Group
				Vibration Control
				Autres activités





# TRAITEMENT DOCUMENTAIRE

**NOM ETUDIANT : Théo VAUCHER**

**Date de Naissance : 03/09/1993**

Signature

**NOM TUTEUR IUT : Marianne DE-MICHIEL**

**DEPT. – ANNEE – PROMO : Département Informatique**

**Année 2013 - Promo 2012**

**NOM ENTREPRISE : Continental**

**Ville- Pays : Toulouse - France**

**NOM TUTEUR ENTREPRISE : Stéphane BRIDE**

Signature

☐ **Confidentiel \***

**SUJET DE STAGE : Conception et réalisation d'un outil capable de générer un fichier de test unitaire à partir de fichiers issus de simulations sur Matlab**

**MOTS CLES (5) : JAVA, conception, développement, tests unitaires, cycle en V**

## **RESUME :**

J'ai effectué mon stage d'une durée de 12 semaines dans l'entreprise Continental. J'ai conçu et réalisé l'outil MAT2PTU, capable de générer un fichier de test unitaire à partir de fichiers issus de simulations sur Matlab. L'outil permet de visualiser graphiquement un résultat de simulation Matlab / Simulink, un ensemble d'entrées et de sorties. L'utilisateur sélectionne des cas de test, les configure et l'application génère le fichier de test unitaire.

**NB :** Votre signature valide et donne droit à la bibliothèque de l'IUT de Toulouse II Blagnac, de diffuser ce document dans le catalogue commun du réseau des bibliothèques des universités de Toulouse.

Portail du réseau : [http://bibliotheques.univ-toulouse.fr/38310688/0/fiche\\_pagelibre/&RH=](http://bibliotheques.univ-toulouse.fr/38310688/0/fiche_pagelibre/&RH=)

**\* Mettre une croix si confidentiel**