UNIVERSITÉ CÔTE D'AZUR

Rapport TER
Semestre Pair du Master 1,
Année Académique 2020/2021

Génération de moniteurs SCADE à partir de scénarios de Safety pour le véhicule autonome

Réalisé par :

MARINO Samuele

LAUBRY Vincent

Encadré par :

MALLET Frédéric

BOUALI Amar

Table de matières

Introduction

Le sujet se place dans le contexte du projet ***PSPC*** (***P**rojets **S**tructurants **P**our la **C**ompétitivité*), de la région ***ADAVEC*** (***A**daptation automatique du **D**egré d'**A**utonomie du **V**éhicule à son **E**nvironnement et au **C**onducteur*), en collaboration avec la *Renault Software Factory*.

Il s'agit de partir de quelques scénarios de sûreté, discutés avec les partenaires industriels du projet. Ces scénarios sont capturés sur la forme de diagrammes de séquences temporisés. Les propriétés temporelles et causales décrites par ces scénarios doivent être traduites dans un langage formel non ambigu qui peut être utilisé pour un traitement automatique.

Le langage formel utilisé est ***CCSL*** (***C**lock **C**onstraint **S**pecified **L**anguage*).
Ce langage intermédiaire est utilisé pour l'analyse formelle et sert de base pour la génération de moniteurs écrits en ***SCADE*** qui doivent être intégrés dans le système ***SCADE*** pour en valider le comportement.

SCADE (***S**afety **C**ritical **A**pplication **D**evelopment **E**nvironment*) est un environnement de développement intégré, basé sur le langage ***Lustre*** diffusé par ***Esterel Technologies***, destiné à la conception de systèmes critiques.

État de l'art

Le projet *ADAVEC* est un projet collaboratif né en juillet 2020 qui a pour objectif un transfert de responsabilité en toute sécurité. Il est donc spécialisé dans le changement de mode entre la conduite autonome et la conduite manuelle dans un véhicule autonome.

Ses partenariats sont :

- La société d'ingénierie logicielle française *AVISTO* ;
- La startup fondée par des ingénieurs du monde de l'innovation et du logiciel automobile *EPICNOC* ;
- La *Renault Software Factory* ;
- L'*Université Nice-Sophia Antipolis.*

Messieurs ANDRÉ Charles et MALLET Frédéric, tous deux membres de l'Université Nice-Sophia, sont des précurseurs concernant les langages synchrones.

En effet, Mr. ANDRÉ, spécialiste dans les langages synchrones comme *SyncCharts*, ancêtre de *SCADE*, est l'inventeur du langage *CCSL*, assisté par Mr MALLET.

Nous avons reçu de leur part une librairie *SCADE* de base, créée en 2010, qui fournissait un ensemble d'opérateurs qu'il fallait implémenter et adapter si nécessaire, pour l'avancement du projet *ADAVEC* en lui-même.

Nous avons aussi eu comme base une librairie de Mr. MALLET, en *Eclipse Modeling Tools* (Java), qui consistait en une génération de codes du langage *CCSL* vers le langage *Java*, ainsi qu'une interface de Générateur *SCADE*.

Pour finir, nous avons eu comme base les scénarios eux-mêmes, conçus par les membres du projet *ADAVEC*, ainsi que les ingénieurs Renault, écrits en Excel.

Nos objectifs pour le projet TER étaient donc multiples :

- Étudier le processus de génération de moniteurs *SCADE* à partir du langage *CCSL* et de l'implémenter ;

- Étudier les scénarios de Safety du projet *ADAVEC* décrits en Excel, en capturer les stimuli et les transformer d'un langage informel et ambigu en un langage formel et non ambigu ;

- Tout cela dans le but de vérifier la bonne conception et viabilité des scénarios donnés.

Le processus de transformation de ces scénarios était donc le suivant :
Excel CCSL Java SCADE.

A travers Excel, nous devions identifier des relations à transcrire en *CCSL*.
De là, ces relations écrites en *CCSL* puis générées en Java, devaient être modifiées de façon à les rendre formelles et compréhensibles à toute l'équipe *ADAVEC*.
Finalement, le résultat attendu était un code *SCADE*.

Travail effectué

Le travail effectué dans un point de vue global peut être décomposé en **trois parties** :

- Transformation des scénarios écrits d'une façon ambigüe sur Excel en une façon non-ambigüe sur Java, en utilisant le langage CCSL comme intermédiaire entre Excel et Java ;

- Étude et adaptation la librairie SCADE donnée par Mr. ANDRÉ afin de comprendre le fonctionnement des opérateurs avec les horloges ;

- Génération du code SCADE en partant du code CCSL, à l'aide de l'environnement Eclipse Modeling Tools (Java) suggéré par Mr. Mallet.

A. Transformation des scénarios

Nous avons commencé cette partie par l'analyse des scénarios.

Ainsi, nous avons reçu 31 scénarios différents à analyser.

Chaque scénario présente :

- Un **Mode de départ**, qui représente le mode de conduite du véhicule (manuel ou autonome) au moment du début du scénario
- Un ensemble d'éléments donnant le contexte du scénario en question ainsi qu'au moins un trigger (un élément clé du scénario) qui détermine le mode final.
  Parmi ces éléments, nous retrouvons :
  - **Weather**, le temps atmosphérique du scénario étudié ;
  - **Road**, le type de route ;
  - **Traffic**, l'intensité du trafic routier ;
  - **Vehicle**, si les capteurs étaient bien fonctionnants ;
  - **Human**, l'état du stress, de santé ou d'attention du conducteur.
- Et un **Mode final**, qui représente l'état attendu du véhicule à la fin du scénario.

Notre première approche consistait en l'énumération de ces scénarios, afin de mieux pouvoir les regrouper et les analyser. Parmi les 31 scénarios, donc :

- 7 étaient en Mode de départ : manuel, mais aucun changement de mode n'était prévu ;
- 8 étaient en Mode de départ : manuel, avec possibilité de changement de mode ;
- 8 étaient en Mode de départ : autonome, mais aucun changement de mode n'était prévu ;
- 8 étaient en Mode de départ : autonome, avec changement de mode prévu.

Nous avons nommé les scénarios en leur donnant un sigle composé d'une lettre et d'un chiffre pour bien les différencier :

A et B pour les modes de départ manuels, C et D pour les modes de départ autonomes, avec un chiffre allant jusqu'à 8 (7 pour les scénarios de type A).

Notre première idée fut de générer un fichier CCSL par scénario, ce qui nous a donc mené à la création de 31 fichiers CCSL.

Après analyse ultérieure des scénarios, nous avons estimé qu'il ne fallait pas tenir en compte de la volonté du conducteur de changer de mode, tout simplement car nous ne pouvons pas intervenir sur la volonté du conducteur lui-même. Nous nous sommes donc focalisés là où le changement de mode était prévu par le véhicule.

C'est là que nous avons dû faire des choix à cause de l'ambiguïté des scénarios :

Nous avons donc ignoré les scénarios dont le mode de départ coïncidait avec le mode final, et nous nous sommes focalisés là où il y avait bien un changement de mode, tout en gardant les sigles pour bien se référer aux scénarios d'origine.

Cependant, nous avons réalisé que certains scénarios avaient des relations en commun.
Cela revenait donc implicitement à copier-coller certaines parties des uns dans les autres.
Cette façon de faire n'était donc pas optimale.

Nous avons donc formulé des relations qui définissent le changement de mode lui-même, sur CCSL, et modifié le code généré en Eclipse d'une façon telle que ces scénarios se révélaient faciles à comprendre par les ingénieurs Renault et ceux qui ne connaissaient pas le langage CCSL.
Cela revient à transformer un morceau de code comme celui-là, écrit en CCSL :

```
Specification DebugMode {
    Clock start finish Trigger ReactionTime end Context
    [
        Let Mode be start + finish
        Exclusion start # finish
        Let Condition be Trigger * Context
        Delay = Trigger $ 1 on ReactionTime
        Precedence Mode < Trigger
        Precedence Trigger < Mode
        Precedence Mode <= Delay
        Precedence Delay <= finish
        Precedence finish <= end
    ]
}
```

En celui-ci :

```
changeMode(simple, "Manual", "Automatic", "Drunk", clockList);
```

Ainsi nous pouvons bien voir que le véhicule passe en mode manuel vers le mode autonome, quand il est ivre. Ici, le trigger est "Drunk".

Voici donc les scénarios où il y avait bien changement de mode :

```
// CHANGE MODES
changeMode(simple, "Manual", "Automatic", "Drunk", clockList);                                                    // Scenario A4
changeMode(simple, "Manual", "MRM", "Unconscious", clockList);                                                    // Scenario A5
changeMode(simple, "Manual", "Automatic", new String[]{"HeavyRain", "NotFocused"}, clockList);                    // Scenario A6
changeMode(simple, "Manual", "MRM", new String[]{"HeavyRain", "HeavyTraffic", "FaultySensors", "LongDriving"}, clockList);  // Scenario B3
changeMode(simple, "Manual", "MRM", new String[]{"CountrySide", "Influenced"}, clockList);                        // Scenario B4
changeMode(simple, "Manual", "Automatic", "Influenced", clockList);                                               // Scenario B5
changeMode(simple, "Manual", "Automatic", new String[]{"HeavyRain", "LongDriving"}, clockList);                   // Scenario B8
changeMode(simple, "Automatic", "MRM", new String[]{"HeavyRain", "Drunk"}, clockList);                            // Scenario C2
changeMode(simple, "Automatic", "Manual", new String[]{"FaultySensors", "Stress"}, clockList);                    // Scenario C3
changeMode(simple, "Automatic", "MRM", "Death", clockList);                                                       // Scenario C4
changeMode(simple, "Automatic", "Manual", "HighwayExit", clockList);                                              // Scenario D1
changeMode(simple, "Automatic", "Manual", new String[]{"HighwayExit", "FaultySensors"}, clockList);               // !  // Scenario D3
changeMode(simple, "Automatic", "Manual", "Inactive", clockList);                                                 // Scenario D4
changeMode(simple, "Automatic", "MRM", "Unconscious", clockList);                                                 // Scenario D5
```

Une fois ces scénarios bien définis, nous nous sommes rendu compte de différentes choses qui encore une fois génèrent de l'ambiguïté.

- Par exemple, vous aurez remarqué que les scénarios D1 et D3 se ressemblent particulièrement. :
    - Dans le scénario D1, le véhicule passe du mode Autonome au mode Manuel lorsqu'il souhaite sortir à la prochaine sortie de l'autoroute.
      Le scénario D3 fait de même, lorsque le véhicule doit sortir à la prochaine sortie et que les capteurs sont défectueux.
      On comprend donc que la partie "capteurs défectueux" n'est pas nécessaire :
      Dès que l'on doit prendre à la prochaine sortie de l'Autoroute, et peu importe les autres facteurs, le véhicule passera en mode Manuel.
    - Nous ne pouvons pas savoir si les ingénieurs Renault, qui ont conçu ces scénarios, ont remarqué cette ambiguïté ou non, mais cela montre qu'il est plus facile de comprendre les scénarios quand ils sont écrits de façon formelle.

- Une autre remarque concerne les scénarios où il y a plus d'un élément trigger, comme les scénarios B3 et B4 :
    - Selon le scénario B3, le véhicule devra passer du mode Manuel au mode MRM.
      Ce mode, dont nous n'avons pas parlé avant, est un mode d'arrêt d'urgence.
      *Il signifie **M**inimal **R**isk **M**anœuvre.*
      Dans le scénario B3 donc, ce mode s'active quand il y a une forte pluie **ET/OU** un trafic intense **ET/OU** les capteurs abîmés **ET/OU** que le conducteur conduit depuis longtemps.
      Vous l'aurez compris, c'est une nouvelle ambiguïté !
      On ne sait pas si c'est un "**ET**" ou un "**OU**".

      ⇒ Nous pourrions donc supposer que ce n'est pas le **ET** logique qui est l'opération qui lie les différents triggers, et qu'il pourrait donc s'agir d'un **OU** logique.
      En effet, il est quasiment impossible que tous ces événements se produisent en même temps et que le changement de mode ait donc lieu.

    - Mais cela pose problème pour le scénario B4.
      En effet, le véhicule devra passer du mode Manuel au mode MRM si le conducteur est influencé ou bien qu'il arrive en campagne.
      *Il n'y aurait pas de raisons particulières pour lesquelles le conducteur devrait s'arrêter s'il arrive en campagne **OU** s'il est influencé, mais cela pourrait avoir son sens s'il arrive dans la campagne **ET** qu'il est influencé.*

      ⇒ Selon nous, ce scénario n'est pas très explicite. En effet, il aurait été plus compréhensible si la campagne était remplacée par la ville, car être en campagne est moins dangereux qu'être en ville.

Ces suppositions provoquent donc des contradictions d'un scénario à l'autre :

Encore une fois, donc, l'ambiguïté nous amène à supposer et à faire des choix. Choix auxquels les concepteurs des scénarios n'avaient pas pensé ou qui, pour eux, étaient évidents.

⇒ Supposer ou omettre des détails n'est pas quelque chose de souhaitable pour les concepteurs de systèmes critiques.

B. La librairie SCADE

Une autre tâche de ce projet TER fut d'étudier la librairie de **Mr. ANDRÉ**. Nous avons remarqué que la librairie en question avait été implémentée en 2010, avec la version 2 du logiciel *SCADE*. De nos jours, nous utilisons la version 6, et c'est la version qu'il nous a présentée lors du cours de *Safety Critical Systems* du premier semestre cette année. Par conséquent, certains opérateurs ne pouvaient plus être utilisés tels quels pour la version actuelle. Cela a impliqué, de notre part, une réadaptation de ces derniers afin de pouvoir être à nouveau utilisés.

Une fois ce travail effectué, nous nous sommes rendu compte qu'il y avait des opérateurs qui n'étaient pas implémentés et qu'il fallait par conséquent créer de toute pièce.

Au fur et à mesure que nous avancions dans nos modifications et créations, la taille du projet ne faisait qu'augmenter… jusqu'au jour où il ne fût plus possible pour nous de pouvoir le sauvegarder. En effet, nous utilisions, au début du projet, la version étudiante de SCADE.

Cette version étant limitée, il nous fallait impérativement obtenir une version complète de ce logiciel, ainsi que la licence associée.
Nous avons pour cela fait appel à Mr. **BOUALI Amar** et son équipe d'ingénieurs SCADE.
Fort heureusement, une installation de cette version complète était déjà en cours sur les serveurs de la faculté.

Afin de pouvoir utiliser cette version SCADE complète, il était aussi nécessaire d'utiliser le VPN signé Unice.

⇒ **Nous n'avons pu obtenir ces deux éléments qu'au dernier mois de notre temps imparti pour le TER.**

Comme dit précédemment, il a été nécessaire d'implémenter des opérateurs supplémentaires, notamment pour les horloges qui s'activent entre deux moments donnés.

C'est pour cela que nous avons construit un opérateur *InitMax*, dont l'horloge en question prend en sortie la même valeur que l'horloge ***Vin*** donnée en entrée, à partir du moment où le l'évènement ***Init*** s'est produit***, et*** cela jusqu'à ce que l'évènement ***Max*** se produise, de la façon suivante :

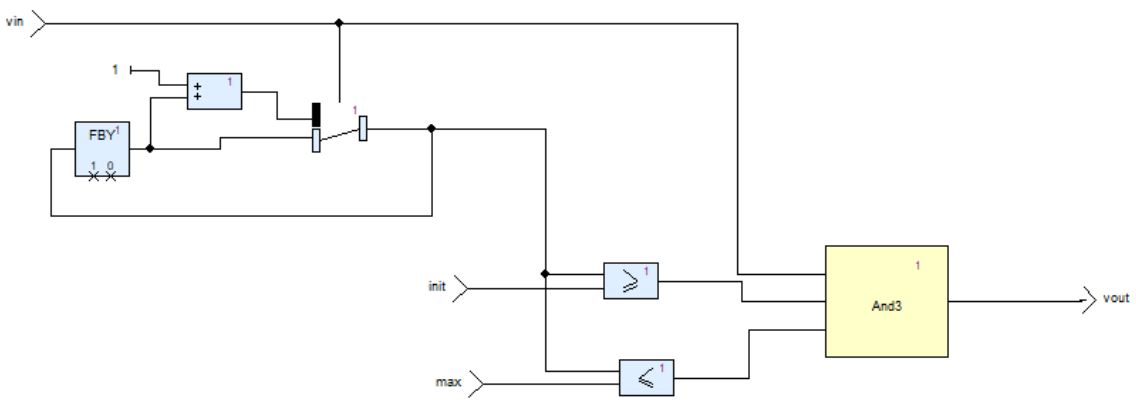

Schéma SCADE de l'opérateur InitMax

Rappelons que les horloges ont trois états possibles :

- **PRESENT,** pour définir une horloge qui est bien active à l'instant donné ;
- **ABSENT**, pour définir une horloge qui n'est pas active à l'instant donné ;
- **DEAD**, pour définir une horloge qui est ABSENT et ne sera jamais plus PRESENT.

Nous avons traité **PRESENT** et **ABSENT** comme des valeurs booléennes, en y ajoutant une sorte de troisième "valeur" pour **DEAD**.
Cela nous donne :

- PRESENT = TRUE ;
- ABSENT = FALSE ;
- DEAD = "REALLY FALSE".

Parmi les opérateurs (ils seront listés dans la partie Génération du code SCADE), l'Union et l'Intersection peuvent être traitées de façon booléenne, respectivement comme un ***OU logique*** et un ***ET logique***.

- Ainsi, comme dans un ***OU logique*** où le TRUE est la valeur décisive, ***PRESENT*** est la valeur décisive dans l'Union des horloges.
- De la même manière, comme dans un ***ET logique*** où le FALSE est la valeur décisive, la valeur décisive pour les horloges est la valeur ***ABSENT***.
  Cependant, nous avons fait un choix :
  Étant donné que la valeur ***DEAD*** est "encore plus FALSE" de la valeur ***ABSENT*** elle-même, nous avons décidé de définir cette valeur DEAD comme étant justement la valeur décisive de l'Intersection des horloges.

(Valeur décisive : valeur qui peut d'emblée définir le résultat final de l'opération en question)

<u>Petite précision :</u> Nous avons enrichi la libraire avec cette notion de "mort" d'une horloge.
Cependant, nous n'avons pas eu à l'utiliser dans le cadre de notre projet.
Nous espérons que cette implémentation pourra servir aux ingénieurs ADAVEC dans le futur.

C. Génération du code SCADE

Dans la librairie donnée par Mr. MALLET, nous avons hérité d'une Interface Java, appelée *MyScadeGenerator*, qui présentait des méthodes dont le corps n'était pas donné.
Notre travail fut de concevoir ces méthodes d'une façon telle à pouvoir générer un fichier SCADE à l'exécution du programme Java qui l'appelait.

Les méthodes étaient les suivantes :

- AddClock ;
- SubClock ;
- Exclusion ;
- Precedence ;
- Causality ;
- Inf ;
- Sup ;
- Union ;
- Intersection ;
- Minus ;
- Periodic ;
- DelayFor.

Nous avons longuement réfléchi pour le corps de ces méthodes. Après avoir réalisé qu'il existe deux façons pour réaliser un fichier SCADE, une visuelle et une textuelle, nous avons adopté l'approche textuelle.
En effet, un fichier SCADE textuel présente une syntaxe qui se rapproche du langage ***Lustre*** que nous avons étudié avec Mr. MALLET pendant le cours de *Programmation Synchrone* de ce semestre.

Voici donc comment un fichier SCADE textuel, qui représente l'Union entre deux horloges, est écrit :

```
L1 = a;
L2 = b;
L3 = Union(L1, L2);
o = L3;
```

Ici, "**a"** et "**b"** sont des horloges, "**o"** est l'horloge résultante, "**L1"**, "**L2"**, et "**L3"** sont les fils qui lient les horloges.

(Un fil est "tuyau" qui permet de rediriger les entrées/sorties des opérateurs vers les entrées/sorties d'autres opérateurs)

Union(L1, L2) représente l'action de l'opérateur ***UNION*** sur les deux horloges liées par les fils L1 et L2 (dont l'information des horloges **a** et **b** est passée respectivement).

Notre projet étant basé sur la génération automatique, nous avons pensé que cette représentation textuelle devrait être automatisée et écrite dans un fichier.
Nous avons donc utilisé la méthode *PrintWriter* pour cela.
Nous avons aussi utilisé des compteurs pour numéroter les fils, ainsi que pour les outputs.
En effet, le fichier final contiendra de nombreuses lignes, dans lequel il faudra pouvoir différencier les fils et outputs les uns des autres.

Prenons comme exemple la méthode de l'exclusion, dont voici sa composition :

```
// Method descriptor #8 (Ljava/lang/String;Ljava/lang/String;)V
public void exclusion(java.lang.String arg0, java.lang.String arg1){
    outputfile.println("L" + ++countL + " = " + arg0 + ";");
    outputfile.println("L" + ++countL + " = " + arg1 + ";");
    outputfile.println("L" + ++countL + " = Exclusion(L" + (countL - 2) + ", L" + (countL - 1) + ");");
    outputfile.println("O" + ++countO + " = L" + countL + ";");
}
```

L'objet, de type PrintWriter, *outputfile* et la méthode *println*, font en sorte que dans le fichier dont le nom est donné par *outputfile* soient écrites les relations qui composeront le fichier lui-même.
Le compteur *countL* sera incrémenté avant d'être écrit, et *arg0* et *arg1* correspondront aux horloges définies par le fichier Java qui appelait cette interface.
Pour les Exclusions en elles-mêmes, on revient en arrière avec *countL* autant de fois que les arguments de la méthode.
Finalement nous donnons l'output comme dernier *countL*, exactement comme dans un fichier SCADE textuel.

Parmi les méthodes, certaines étaient récurrentes, comme l'Union et l'Intersection.
En effet, il pourrait très bien y avoir un OU et un ET avec plus de deux éléments.
Sachant aussi que nous avons appris dans les cours de *Programmation Synchrone* comment traiter la récursivité d'une façon parallèle et plus rapide, notre encadrant nous a suggéré d'adopter la même approche aussi dans ces opérateurs. Voici donc l'Union :

```
// Method descriptor #14 (Ljava/lang/String;[Ljava/lang/String;)V
public void union(java.lang.String arg0, java.lang.String... arg1){
    int n = arg1.length;
    if (n == 1) {  // Cas terminal: n == 2
        outputfile.println("L" + ++countL + " = " + arg0 + ";");
        outputfile.println("L" + ++countL + " = " + arg1[0] + ";");
        outputfile.println("L" + ++countL + " = Union(L" + (countL - 2) + ", L" + (countL - 1) + ");");
        outputfile.println("O" + ++countO + " = L" + countL + ";");
    } else {
        String[] left = Arrays.copyOfRange(arg1, 0, n/2);
        String[] right = Arrays.copyOfRange(arg1, n/2, n);
        union(arg0, left);
        union(arg0, right);
    }
}
```

(Il faut donner une image encore meilleure, à modifier, récursivité non-optimale)

Nous pouvons voir que le cas d'arrêt est quand n = 1, la longueur du tableau de String donnée en arg1.
Nous pouvons voir ici comment la récurrence est effectuée de façon parallèle, à gauche et à droite du tableau, ce qui permet de résoudre la récurrence en un temps logarithmique, au lieu d'un temps linéaire.

Afin de pouvoir vraiment appeler cette interface, il fallait modifier légèrement la classe Main du programme Java généré par CCSL, en ajoutant le morceau qui permettait la liaison entre le fichier Java et l'interface *MyScadeGenerator*.
Une fois ce morceau ajouté, il suffit juste d'exécuter le programme Java pour que le fichier SCADE textuel soit généré.

Gestion de projet

Premièrement, la compréhension de notre sujet ainsi que la découverte des langages et des ressources associées a occupé une grande partie du début de notre travail.

En effet, comprendre la librairie donnée et le langage CCSL et se les approprier ne fut pas évident.

Dès la première semaine, nous avons commencé les appels hebdomadaires avec Mr. MALLET, que nous avons défini comme chaque jeudi de 16h00 à 18h00. Cela nous a permis, dès le début, de très vite réussir cette phrase d'adaptation et de compréhension de la base de notre travail.

Afin de ne perdre aucun "morceau" du TER, nous avons constamment travaillé à deux, en appel Zoom, chaque semaine, du jeudi au dimanche, car nous avions cours du lundi au mercredi.

Concernant le travail en lui-même, nous avons travaillé sur les trois parties décrites précédemment en même temps. En effet, cela nous permettait de continuer à travailler, même lorsque nous étions bloqués et que nous avions besoin de Mr. MALLET alors qu'il n'était pas présent au même moment.

La fréquence de ces appels avec Mr MALLET a donc continué jusqu'à la fin du semestre, nous apportant chaque semaine des réponses à nos questions, des pistes d'améliorations de notre travail ainsi que de nouvelles idées d'ajouts possibles au projet.

Nous avons sauvegardé notre projet sur Github, donc voici le lien.

Conclusion

Au moment de la rédaction de ce rapport, nous avons donc bien avancé sur ces trois parties qui composent notre projet dont le but final est la génération de moniteur SCADE :

1. Transformation des scénarios :
    1.1. Transcription des scénarios Excel en CCSL ;
    1.2. Clarification de ces derniers pour les rendre compréhensible ;

2. Librairie Scade de Mr. ANDRÉ :
    2.1. Modification des opérateurs déjà présents ;
    2.2. Ajout de nouveaux opérateurs ;

3. Génération du code SCADE :
    3.1. Création du corps des méthodes concernant les horloges
    3.2. Génération des fichiers SCADE textuels.

Cependant, il reste une dernière étape pour laquelle nous n'avons pas eu assez de temps pour réellement comprendre comment le faire :

Faire la liaison de ces trois parties pour qu'elles "collaborent" ensemble, dans le but de partir d'un scénario Excel et d'obtenir le moniteur SCADE associé.

Le fait d'avoir tardivement reçu la licence professionnelle pour SCADE ainsi que le VPN universitaire pour y accéder nous a amené à perdre du temps, que nous avons fait en sorte de pallier en avançant sur les autres parties qui ne nécessitaient pas SCADE ou qui pouvaient attendre afin d'être testées

Perspectives et réflexions personnelles

Nous nous sommes rendu compte rapidement que ce projet ne correspondait pas à ce que nous pensions faire. En effet, nous pensions faire un projet entièrement sur SCADE, comme celui que nous avions fait avec Mr. BOUALI dans la matière *Safety Critical Systems*.
Ce projet consistait en l'implémentation d'un ascenseur, répondant aux normes de sécurité imposées.

Nous nous sommes retrouvés à apporter une contribution sur ce grand projet qu'est le projet ADAVEC, projet étant d'autant plus actuel que les véhicules autonomes ont pour vocation à devenir une must-have dans notre quotidien.

Humainement parlant, ce TER a été exigeant quant à la nécessité de gérer notre temps pour se dédier efficacement au travail, car il venait se greffer à côté des autres matières dans lesquelles nous avions aussi des projets ou des TPs à rendre régulièrement.

De plus, tout ou presque dans ce projet était nouveau pour nous. Nous ne connaissions que brièvement le logiciel SCADE que nous avions vu au premier semestre, ainsi que le langage Java.
Nous avons dû nous approprier la librairie de Mr. ANDRÉ, comprendre le langage CCSL et transformer les scénarios dans ce langage.
Heureusement, durant notre semestre 2, nous avons suivi des matières comme Programmation Synchrone, qui nous ont permis d'obtenir ces connaissances pour le TER. Timing Parfait.

Ce projet fût pour nous une expérience nouvelle, compliquée, mais très satisfaisante !
Notre encadrant nous a patiemment montré la bonne voie à adopter sur certaines parties de ce projet.
Pour la première fois, nous avons eu le sentiment de travailler sur un projet en relation avec l'extérieur de l'université et plus précisément dans le vrai monde du travail.
Pouvoir travailler dans le but d'apporter quelque chose à une entreprise et pas seulement pour une note fût vraiment motivant et satisfaisant.

Nous devions avoir un appel avec les équipes ADAVEC, expérience qui se serait avérée fort enrichissante, mais il n'a malheureusement pas eu lieu.

Ce projet représente ce à quoi nous devrons nous attendre lorsque nous sortirons de cette université.

Nous voici à la fin de ce rapport concernant notre projet de TER.
Nous espérons avoir réussi à vous expliquer clairement notre contribution au grand projet ADAVEC !

Bibliographie

Projet ADAVEC : http://adavec.fr/fr/

CCSL : https://hal.inria.fr/hal-03135428

SyncCharts : https://www.i3s.unice.fr/~andre/SyncCharts/

SCADE : https://www.ansys.com/products/embedded-software/ansys-scade-suite

Lustre : https://www-verimag.imag.fr/The-Lustre-Programming-Language-and?lang=en

Lien GitHub de ce projet : https://github.com/Viince06/TER_SafetyCriticalSystem

MARINO Samuele,
LAUBRY Vincent,
M1 Informatique,
Année 2020/2021