Rapport d'apprentissage n°1
2020-2021

19 SEPTEMBRE

Thales DMS
Créé par : Cyril FARID A1MSI
Maître d'apprentissage : Maxence COTIN

Remerciements

Je tiens à remercier en premier lieu l'ensemble du site Thales DMS pour la confiance qu'il m'a accordé et son accueil notamment M Antoine Preyssas responsable du site Thales DMS.

Je tiens particulièrement à exprimer toute ma gratitude à M David COLIN chef du service Logiciel.
Je souhaite également remercier mon maître d'apprentissage M Maxence COTIN qui a su me confier des tâches, me conseiller en cas de besoin et m'encourager.

Je souhaite remercier M Philippe LEJEUNE Architecte Logiciel pour ses éclaircissements sur l'avenir du projet.
Je suis très reconnaissant du partage de Mme Emilie ORIOL sur sa vision de la gestion de projet et des équipes.

Je souhaite exprimer toute ma reconnaissance à M Thierry VERGUETHEN Ingénieur développement Logiciel pour son aide et ses éclaircissements.
Je tiens également à remercier l'ensemble des développeurs au sein du service Logiciel pour avoir suivi le déroulement de mon apprentissage.

Enfin, je remercie l'ensemble du service Logiciel en charge du développement du logiciel SPECTRA/Brouillage pour l'accueil et l'intégration.
Leur soutien et leur expertise m'ont également beaucoup aidé.

Table des matières

Introduction

Dans le cadre de mon cycle ingénieur par apprentissage à l'ESME Sudria, j'ai eu l'honneur d'effectuer la phase pratique de ma formation chez Thales Defense Missions System (Thales DMS).

Lors de ses différentes périodes en entreprise, j'ai pu découvrir le monde du travail. Ces périodes m'ont permis de participer à un projet dès son commencement. Elles m'ont permis d'en apprendre davantage sur les métiers d'ingénieur en développement logiciel, d'architecte logiciel, chef de projet et de service mais également le déroulement d'un projet.

Cette première année d'apprentissage m'a permis également d'acquérir de nouvelles compétences techniques et d'approfondir les compétences techniques acquises lors de mon cycle préparatoire à l'ESME Sudria.

Elle m'a également permis d'apprendre à travailler en équipe, à être autonome et responsable à travers de nombreuses tâches réalisées lors de mes périodes entreprise.

Ce rapport a pour objectif de réaliser un bilan de cette première année d'apprentissage. La première partie du rapport sera dédiée à la présentation du service et de l'environnement de travail. Une deuxième partie abordera le projet mis en œuvre et les différentes tâches réalisées. Pour finir, la dernière partie analysera les apprentissages acquis en entreprise.

I-Présentation du service et de l'environnement de travail

Thales est un groupe né en 2000 par la fusion de Thomson CSF et Dassault Electronique, le groupe est présents sur 5 domaines d'activités : l'Aéronautique, l'Espace, le Transport, la Défense et Sécurité et l'Identité et Sécurité numériques.

1.Présentation de Thales DMS et du service Logiciel Brouillage

Le groupe Thales possède plusieurs filiales notamment Thales LAS France qui est une filiale dans le domaine « Land and Air Systems » (aéronautique et transport) et Thales DMS France dans le domaine « Defense Missions Systems » (Défense et Sécurité).

L'activité de Thales DMS (Defense Missions Systems) fournit des équipements, des solutions et des services liés aux systèmes de combat électroniques, de surveillance et de reconnaissance, de combat naval, de surface et de lutte sous la mer.

Le pôle Logiciel du département Guerre Electronique intervient sur tous les projets intégrant des sous-ensembles logiciels en collaboration avec les services systèmes de guerre électronique navale et aéroportée. Le pôle logiciel est en charge maîtriser les techniques, technologies et méthodologies des logiciels temps réel embarqués, distribués.
Situé sur les sites de Brest et d'Elancourt, le pôle est composé de plus de 80 personnes couvrant l'ensemble des métiers du Logiciel.

Le service Logiciel Brouillage auquel j'appartiens est en charge du développement du logiciel SPECTRA/Brouillage en charge de lancer les actions de contre-mesures adéquates.

Le service est composé de la manière suivante :

- Un chef de service.
- Une cheffe de projet.
- Un architecte Logiciel.
- Un groupe d'ingénieurs en développement logiciel.
- Un apprenti ingénieur.

Les membres du service participent de près ou de loin à mon apprentissage du métier d'un ingénieur et à l'acquisition des compétences requises pour ce métier.

2.Les liens avec le service Logiciel

Au début de ma première année d'apprentissage, je travaillais essentiellement avec mon maître d'apprentissage M Maxence COTIN qui me donnait des tâches à réaliser et m'expliquait le projet auquel je participe.

Au cours de l'année, j'ai pu m'intégrer au service ce qui m'a permis de pouvoir en apprendre plus sur les différents métiers composant le service Logiciel et sur leur différent point de vue de la vie professionnelle et de leurs conseils.

En cas de difficulté, je peux me diriger vers mes collègues ingénieurs qui peuvent m'aider ou m'épauler sur les difficultés qu'ils ont déjà rencontrées au cours de leurs différentes missions.

Au cours de ma mission, j'informe mon chef de service de l'état d'avancement du projet lorsque nous avons des réunions administratives notamment pour le semestre à l'international.

Je réalise une réunion toutes les 2 semaines avec l'architecte du service : Architecte logiciel.
Pendant cette réunion, j'informe l'architecte et mon maître d'apprentissage sur l'avancement de la mission, des difficultés que j'ai pu rencontrer. Nous discutons également des retards que ces difficultés et rectifications engendrent sur le projet . L'architecte m'informe également des futurs changements à réaliser afin de discuter de la possibilité de ses changements.

Je participe également à une réunion quotidienne avec tout le service (ingénieurs, architecte logiciel, chef de projet et chef de service) ayant pour but de parler des tâches réalisées la veille et les difficultés rencontrées. Cette réunion permet au service d'être informé des avancées de chacun et de pouvoir aider les personnes en difficultés.

Nous réalisons également d'autres réunions avec des membres de différents services afin de partager nos travaux et nos avancées.

3.La hiérarchie du service

Comme présenté précédemment, le service est composé de plusieurs corps de métiers :

- le chef de service M David COLIN qui gère l'ensemble des équipes et qui est responsable de l'évolution du service.

Également, un architecte logiciel, M Philippe LEJEUNE, qui a pour rôle d'anticiper l'impact du code sur les futurs changements à réaliser afin d'innover le logiciel. Il prend également en compte les actions réalisables par les développeurs afin de faciliter les changements futurs à l'aide de la réalisation d'un code modulaire.

Cela permettra également de faciliter l'implémentation du code dans le cas où la programmation composant logiciel pourrait utilisée pour un autre composant. Si le code est générique alors l'implémentation sera possible et sans impact.

Il collabore avec le chef de projet afin d'estimer le temps nécessaire pour subvenir à une offre. L'offre peut être un changement à réaliser sur le composant. L'architecte consolide les offres avec les ingénieurs en développement et réalise une estimation en fourchette d'heures.

Après les estimations réalisées par l'ensemble des architectes, les offres sont transmises à la cheffe de projet.

La cheffe de projet, Mme Émilie ORIOL, a pour rôle d'organiser le projet. Il lui est donné un budget pour réaliser l'offre et son rôle est d'organiser et répartir les tâches selon les priorités par rapport aux autres projets du service pour pouvoir livrer en temps et en heures.

L'équipe de développeurs a pour rôle de réaliser les tâches données par la cheffe de projet. Les développeurs ont pour rôle d'analyser les changements et leurs impacts afin d'en prévenir l'architecte logiciel.

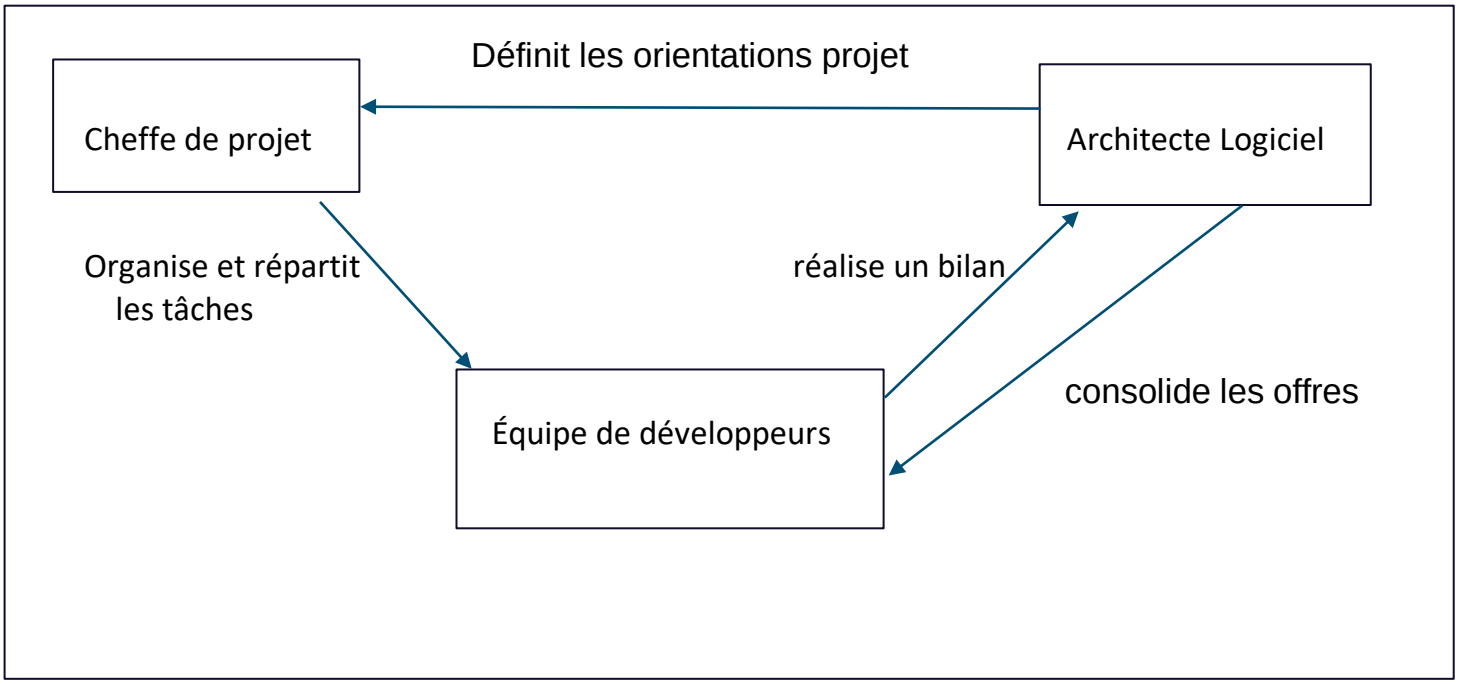


Figure 1 : Les interactions pendant le déroulement d'un projet dans le service

II-Mission Portage Linux

La mission Portage Linux a pour but d'améliorer les composants logiciels de brouillage présents dans le Rafale, rendre le code générique et faciliter les tests.

1.Présentation de la Mission

La mission Portage Linux est une mission ayant pour but de bâtir le futur du brouillage. Ce portage consiste à rendre le code générique permettant de pouvoir changer les données plus facilement, utiliser les programmes pour un autre appareil que le Rafale.
La mission comprend également le changement du système d'exploitation. Le portage sous Linux est fait pour faciliter les tests du logiciel (maintenabilité) et réduire la diversité du parc de machines de la DSI.

La compilation Linux permettrait de :

- ne plus être dépendant des serveurs SOLARIS
- avoir une compilation 2-3 fois plus rapide que sous Solaris
- avoir des exécutions de test nettement plus rapide
- avoir une facilité d'utilisation et d'exécutions.

Cette mission est une mission à part entière, elle nécessite une bonne organisation et plusieurs compétences techniques.

2. En attente d'une habilitation défense

Pour réaliser cette mission, il faut être doté d'une habilitation défense ce qui permettrait de pouvoir avoir accès à des informations dite « sensible ». L'attente d'une telle habitation est longue, elle peut prendre plusieurs mois voire une année.

En attente de ma future habilitation défense, Maxence a décidé de me préparer à ma future mission en commençant par une formation théorique sur le langage **Ada** et le **Big\Little Endian**.

Le **Big Endian** est un ordre de bits de stockage dans lequel le bit le plus important (= valeur la plus significative dans la séquence) est placé en première position de stockage.

Le **Little Endian** est un ordre de bits de stockage dans lequel la valeur la moins importante (=valeur la moins significative dans la séquence) est en première position de stockage.

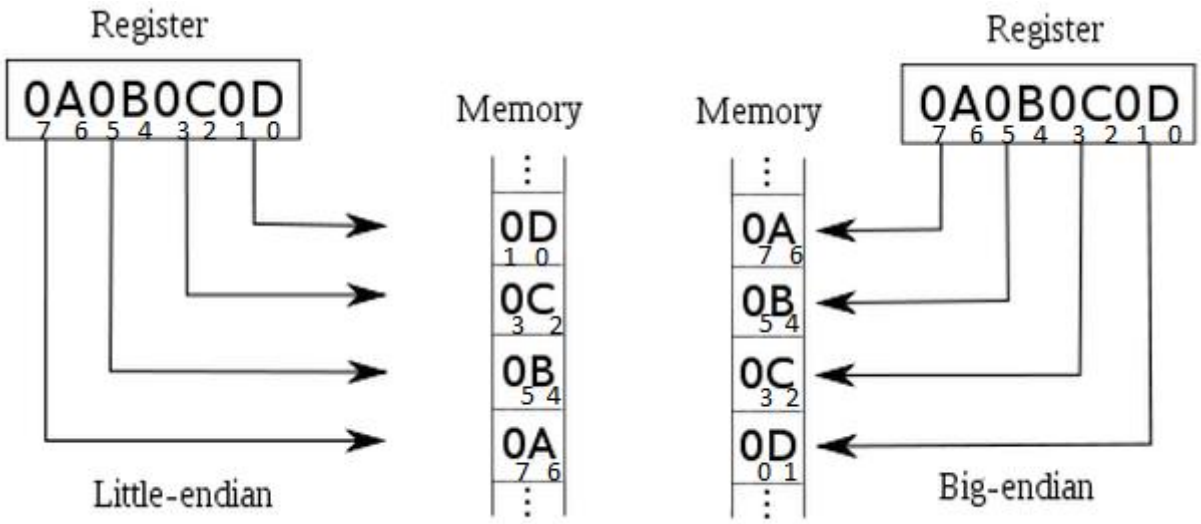


Figure 3 : Schéma décrivant le Big/Little Endian

Après cette partie théorique, nous sommes passés à la pratique avec un premier projet contenant différentes notions qui seront utilisées lors de ma future mission.

Ce premier projet consistait en la réalisation d'un programme qui prend en entrée un nombre en hexadécimale et renvoie une date de naissance.

Par exemple :

110B07D0 (= en hexadécimale => décimale : 17112000) renverrait 17/11/2000 et le programme comprendrait que 17 est le jour de naissance, le 11 est le mois et 2000 l'année.

La première étape était de créer un type de variable "record" « T_Date » qui contenait 3 types de variables filles : le jour un entier allant de 1 à 31 coder sur 4 bits de stockage, le mois allant de 1 à 12 coder aussi sur 4 bits de stockage et l'année allant de 1900 à 2020 coder sur 8 bits de stockage.

Un type de variable "record" est un type de variable mère qui contient plusieurs autres types de variables filles.

```
Package train is
   type T_Date is
      record
         Jour : Integer(1..31);
         Mois : Integer(1..12);
         Annee : Integer(1500..2022);
      end record;

   for T_Date use
      record
         Jour at 0 range 0..7;
         Mois at 1 range 0..7;
         Annee at 2 range 0..15;
      end record;

end train;
```

Figure 4 : Type Variable T_Date de type record

À travers cette première étape, j'ai appris à répartir des bits d'une variable "record".

La deuxième étape était de comprendre comment convertir un entier de 32 bits en une date de naissance de 32 bits (la variable crée lors de la première étape).

Pour cela, je devais avoir recours à une fonction ADA appelé **unchecked conversion** permettant une transmutation des types.

```
type T_Message_01 is array (1 ..7) of T_Date; -- Tableau de Date en fonction du nombre de données rentrés

function Trier(Tab :in out T_Message_01) return T_Message_01;
function lecture(Tableau: in out T_Donnee_Tableau) return T_Message_01;
procedure Affichage(T:T_Message_01);

function conv_donnee_en_date is new Ada.Unchecked_Conversion(Source => T_Mot_Long,
                                                             Target => T_Date); --Unchecked conversion de Donnée
```

Figure 5 : Extrait de déclaration d'une fonction Ada.Unchecked_Conversion

La dernière étape de ce projet était de pouvoir utiliser ce programme avec un tableau d'entiers en les convertissant puis triant du plus vieux au plus jeune.

```
With ppt; Use ppt;
With ada.Text_IO; Use Ada.Text_IO;
With Ada.Integer_Text_IO; Use ada.Integer_Text_IO;
procedure Main14 is
   Tableau:ppt.T_Donnee_Tableau;
   Tableau_converti,Tableau_trie:ppt.T_Message_01;
begin

   Tableau:=(16#170B07C8#,16#110B07D0#,16#140A07C8#,16#140C07BC#,16#1A0C07A9#,16#100907B0#,16#120707A8#);
   Tableau_converti:=ppt.lecture(Tableau);
   Put("Conversion en cours ................");
   New_Line;
   Tableau_trie:=ppt.Trier(Tableau_converti);
   Put("Conversion Terminée avec succès");
   New_Line;
   ppt.Affichage(Tableau_trie);


end Main14;

Main14
Messages  Locations  Run: main14.exe
C:\Users\t0246252\MyApp\Projet14\obj\main14.exe
 23/ 11/ 1992 , 17/ 11/ 2000 , 20/ 10/ 1992 , 20/ 12/ 1980 , 26/ 12/ 1961 , 16/ 9/ 1968 , 18/ 7/ 1960 ,
Conversion en cours ................
Conversion Terminée avec succès
 18/ 7/ 1960 , 26/ 12/ 1961 , 16/ 9/ 1968 , 20/ 12/ 1980 , 20/ 10/ 1992 , 23/ 11/ 1992 , 17/ 11/ 2000 ,
[2021-09-09 11:25:17] process terminated successfully, elapsed time: 00.22s
```

Figure 6 : Programme finale avec classement des dates de naissances

À la fin de ce projet, une nouvelle notion a été abordée : les **tests unitaires** permettant de tester le projet « dates de naissance »

Lors de ses tests, on vérifie le fait qu'avec une valeur donnée en entrée, on obtienne la valeur attendue en sortie.

```
   end Name;

   procedure Test_001(T: in out AUnit.Test_Cases.Test_Case 'Class) is
      pragma Unreferenced(T);
      Tableau:T_Donnee_Tableau :=(16#170B07C8#,16#110B07D0#,16#140A07C8#,16#140C07BC#,16#1A0C07A9#,16#100907B0#,16#120707A8#)
      Message:T_Message_01;

   begin
      Message:=lecture(Tableau);

      assert(Message(1).Jour=23, "Jour 1 est incorrect");
      assert(Message(1).Mois=11,"Mois 1 est incorrect");
      assert(Message(1).Annee=1992,"Année 1 est incorrect");

      assert(Message(2).Jour=17, "Jour 2 est incorrect");
      assert(Message(2).Mois=11,"Mois 2 est incorrect");
      assert(Message(2).Annee=2000,"Année 2 est incorrect");

      assert(Message(3).Jour=20, "Jour 3 est incorrect");
      assert(Message(3).Mois=10,"Mois 3 est incorrect");
pt_test.Test_001
Messages  Locations  Run: test_all.exe
C:\Users\t0246252\MyApp\TU\obj\test_all.exe
 23/ 11/ 1992 , 17/ 11/ 2000 , 20/ 10/ 1992 , 20/ 12/ 1980 , 26/ 12/ 1961 , 16/ 9/ 1968 , 18/ 7/ 1960 ,
 23/ 11/ 1992 , 17/ 11/ 2000 , 20/ 10/ 1992 , 20/ 12/ 1980 , 26/ 12/ 1961 , 16/ 9/ 1968 , 18/ 7/ 1960 ,

OK Test Case : Test n°1 Lecture
OK Test Case : Test n°2 Tri
OK Test Case : Test n°3 Lecture et Tri

Total Tests Run:   3
Successful Tests:  3
Failed Assertions: 0
Unexpected Errors: 0
[2021-09-09 13:39:27] process terminated successfully, elapsed time: 00.31s
```

Figure 7 : Exécution des tests unitaires du projet « dates de naissance » et résultats obtenus

Pour poursuivre ma formation, Maxence a décidé de me donner un projet en lien avec ma future mission, ce projet consiste en l'utilisation d'un outil interne appelé **BIE-BIM**. Cet outil permet de saisir un data-model contenant des structures de données qui seront par la suite généré en un langage donné.

Après avoir généré le code, il fallait que je développe un programme qui déclare un type « Message » et l'utilise.

Comme pour le programme gérant les dates de naissance, la prochaine étape était de réaliser un programme qui permet la conversion d'une série d'entiers en brut en la structure de données « Message 01 » à l'aide de deux méthodes :

-la première en utilisant le code développé en ada

-la deuxième en utilisant le code généré par BIE-BIM.

Après vérification des deux méthodes, il m'a été demandé de les associer afin de tester si pour un même « Message » donné, ils me renverraient le même « Message » en sortie (Message = Structure de donnée).

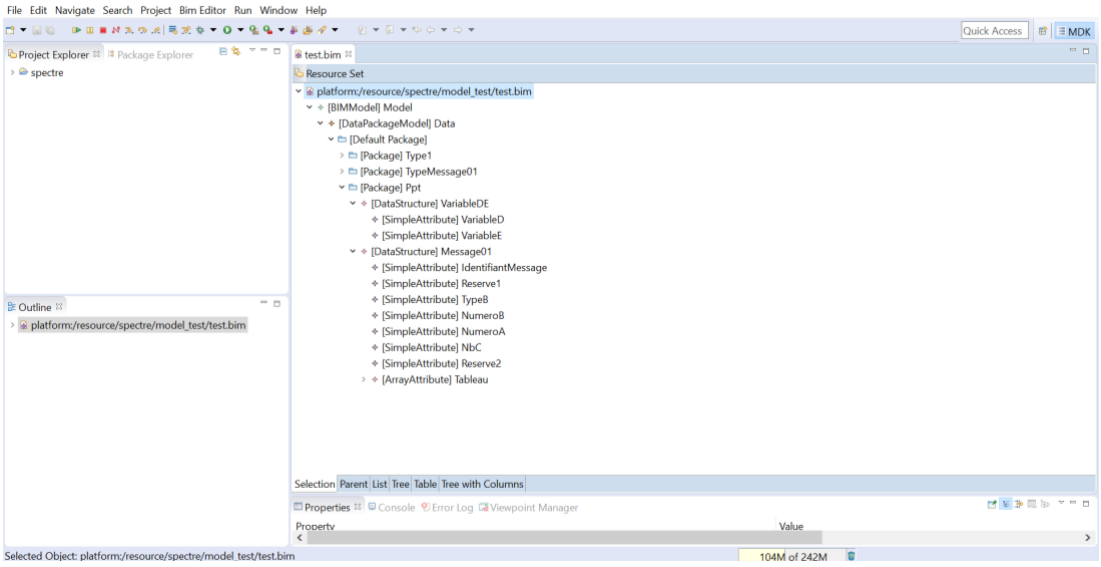


Figure 8 : DataModel contenant la structure de donnée « Message01 »

3. Début de la mission Portage Linux

1.Initialisation de la mission Portage Linux

Ayant obtenu mon habilitation Défense en janvier 2021, le chef de service a pu commander l'équipement nécessaire pour que je puisse commencer à participer à la mission Portage Linux.

En attente du basculement du service de **ClearCase** à **GitHub**, j'ai pu me former aux deux gestionnaires de configuration.

GitHub et **ClearCase** sont des gestionnaires de configuration destinés aux développeurs de logiciels.

Ces gestionnaires de configuration permettent de :

- faciliter le travail en équipe.
- partager son travail sans avoir d'impact sur le travail en cours des autres développeurs (travailler ensemble sans manque de coordination).
-consulter le travail d'une autre équipe de développeurs.
-partager sans travail en toute sécurité.

Pour me former sur **GitHub**, j'ai utilisé un simulateur trouvé sur internet : www.learngitbranching.js.org .

Maxence m'a formé sur **ClearCase** afin de pouvoir utiliser la plateforme le plus rapidement possible sur mes missions.

Pour pouvoir commencer le portage Linux du composant, je me suis également formé sur **Jira**.

Jira *est un système de gestion de projets développé par Atlassian permettant de voir les tâches (= User Story) pour les projets à réaliser.*

2. Les débuts du Portage

Maxence a commencé à me publier des **US** (**User Story**) à réaliser lors du portage Linux du composant. Ces US publiées sur l'application **Jira** sont reliées à **GitHub**.

Ma première US a été de compiler le composant logiciel sous Linux à l'aide de modifications à réaliser sur le fichier **Makefile** du composant et de créer un fichier **bash**.

Pour cette première tâche, j'ai été aidé par un ingénieur Thierry VERGUETHEN qui a pu me guider sur la méthode à suivre.

Après avoir réalisé cette mission, j'ai appris à générer un code à l'aide d'un data-model (comme réalisé auparavant lors du premier semestre mais cette fois-ci en ligne de commande afin de générer le data-model pendant la compilation sous Linux).

Pour cela, j'ai réalisé un fichier **bash** qui permettait cette conversion à l'aide de fichiers réalisés par le service.

Avant de modifier le **data-model** (ce qui n'aurait aucun intérêt avant le basculement **ClearCase** – **GitHub**), je réalise un changement d'**endianité** pour un premier message afin de tester le bon fonctionnement de cette méthode.

Pour cela, j'ai utilisé le fichier que j'ai réalisé lors du premier semestre expliquant la méthode à suivre afin de réaliser ce changement d'**endianité** avec un **data-model**.

Après avoir réalisé le changement d'**endianité**, j'ai réalisé un **test unitaire** de mes fonctions réalisées pour ce premier message.

L'étape suivante a été de réaliser la compilation des tests unitaires en Linux et au Solaris afin de pouvoir réaliser la compilation du composant et de ses tests sous le même système d'exploitation.

3. Le basculement Big/ Little Endian

Le basculement se résume au changement des fonctions permettant la transcription des messages afin qu'elles soient adaptées aux deux systèmes d'exploitation (Solaris et Linux). Pour cela, il faut passer par un fichier stub (=Bouchon) qui va détecter le système d'exploitation dans lequel on compile afin de convertir les messages en Big ou Little Endian selon le système d'exploitation utilisé.

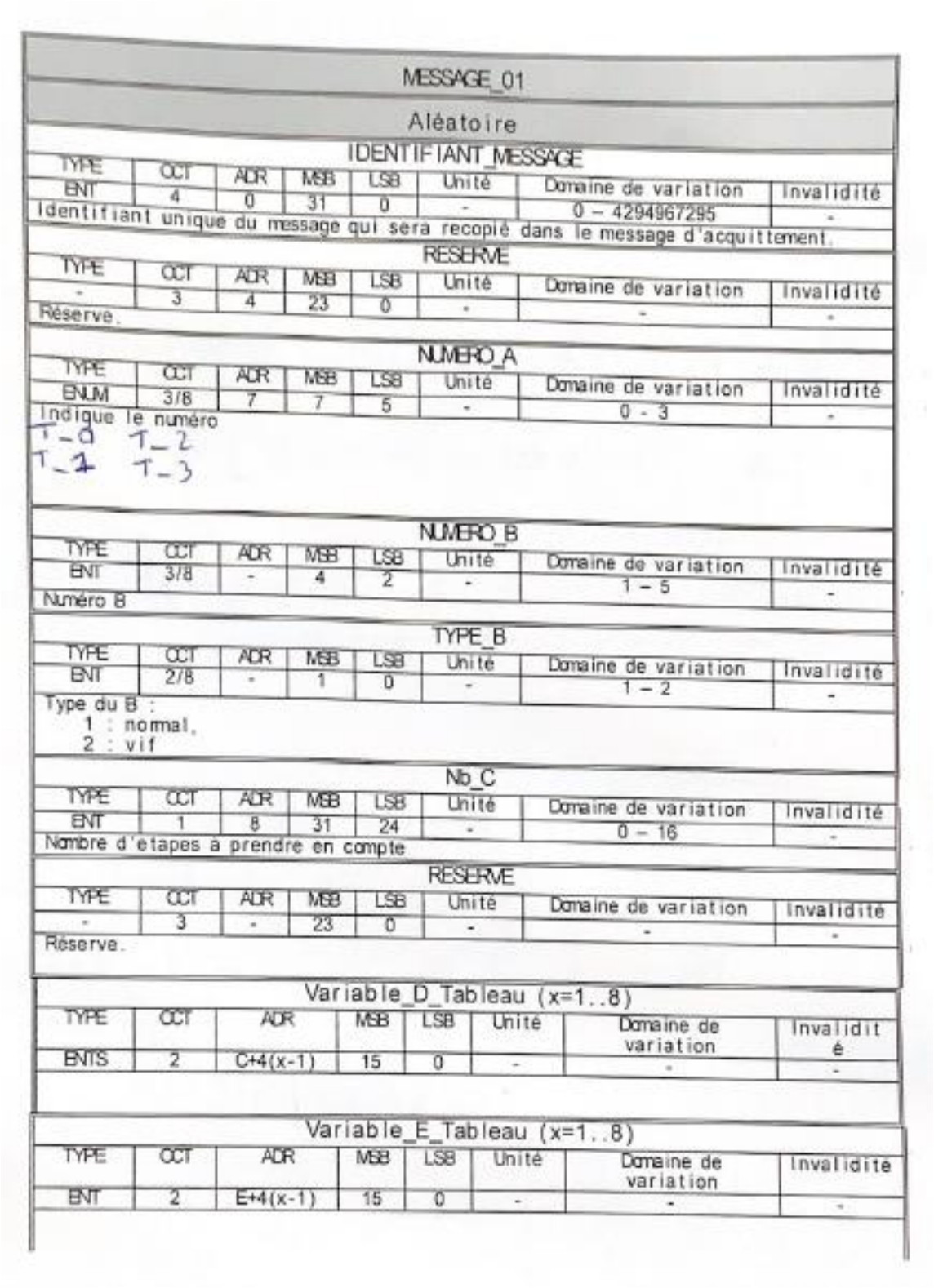
Le but de cette étape a été de faciliter le travail du développeur afin qu'il puisse compiler et lancer ses tests sous Solaris ou Linux sans problème. Le composant devait pouvoir détecter à l'aide du fichier stub le système d'exploitation utilisé par l'utilisateur.

Actuellement, je réalise :

- la comparaison entre les structures de données présentes dans le data-model et celles présentes dans les fichiers internes
- je remplace les anciennes fonctions utilisées par le service, par les fonctions générées par le logiciel BIE_BIM.

Ces fonctions générées permettent la conversion des données de Big Endian à Little Endian et inversement.

Après avoir réalisé ce basculement, j'effectue des tests unitaires permettant de vérifier le bon fonctionnement des fonctions générés. Cela permet également de tester les différents paramètres de la structure de données et le basculement Big/Little Endian.



MESSAGE_01

Aléatoire

IDENTIFIANT_MESSAGE

TYPE	OCT	ADR	MSB	LSB	Unité	Domaine de variation	Invalidité
ENT	4	0	31	0	-	0 – 4294967295	-

Identifiant unique du message qui sera recopié dans le message d'acquittement.

RESERVE

TYPE	OCT	ADR	MSB	LSB	Unité	Domaine de variation	Invalidité
-	3	4	23	0	-	-	-

Réserve.

NUMERO_A

TYPE	OCT	ADR	MSB	LSB	Unité	Domaine de variation	Invalidité
ENUM	3/8	7	7	5	-	0 - 3	-

Indique le numéro
T_0 T_2
T_1 T_3

NUMERO_B

TYPE	OCT	ADR	MSB	LSB	Unité	Domaine de variation	Invalidité
ENT	3/8	-	4	2	-	1 – 5	-

Numéro B

TYPE_B

TYPE	OCT	ADR	MSB	LSB	Unité	Domaine de variation	Invalidité
ENT	2/8	-	1	0	-	1 – 2	-

Type du B :
1 : normal,
2 : vif

Nb_C

TYPE	OCT	ADR	MSB	LSB	Unité	Domaine de variation	Invalidité
ENT	1	8	31	24	-	0 – 16	-

Nombre d'étapes à prendre en compte

RESERVE

TYPE	OCT	ADR	MSB	LSB	Unité	Domaine de variation	Invalidité
-	3	-	23	0	-	-	-

Réserve.

Variable_D_Tableau (x=1..8)

TYPE	OCT	ADR	MSB	LSB	Unité	Domaine de variation	Invalidité
ENTS	2	C+4(x-1)	15	0	-	-	-

Variable_E_Tableau (x=1..8)

TYPE	OCT	ADR	MSB	LSB	Unité	Domaine de variation	Invalidité
ENT	2	E+4(x-1)	15	0	-	-	-

Figure 9 : Exemple de fichier interne d'un Message (structure de données)

III-Analyse des apprentissages

1.Compétences acquises

1.Le Langage ADA

À travers ses différents projets, j'ai assimilé plusieurs notions. Principalement le langage Ada qui a été utilisé pendant tout le semestre à travers plusieurs projets en lien avec la future mission à réaliser après l'acquisition de l'habilitation Défense.

Lors du 1er projet (projet « Dates de naissance » convertissant des entiers en une date de naissance, de nouvelles notions complexes ont été abordés notamment les **unchecked conversions,** une conversion que je n'avais jamais abordée en cours et qui est très rigoureuse.

*Les **unchecked conversions** sont des fonctions ADA permettant la transcription d'un type de taille n en un autre type de même taille.*

Pendant ces différents projets, plusieurs notions ont été abordées notamment les tests unitaires qui sont utilisés par les développeurs afin de vérifier leur programme. La notion interne « Message » a été également abordée et utilisée.

Lors du déroulement de la mission, des progrès ont été observées dans le développement des programmes en ada, les notions complexes comme les **unchecked conversions** et les **tests unitaires** sont assimilés.

2.Les outils internes

La formation sur les outils **internes** a été utile tout au long du semestre afin de :

- **prendre connaissance des missions à réaliser** (Jira).
- **livrer et comparer avec le travail des autres** (GitHub et ClearCase).

J'ai pu également utiliser des notions acquises lors du premier semestre comme le changement d'**endianité**, les **tests unitaires** et la modification d'un **data-model** réalisé sur **biebim**.

A travers le déroulement de la mission, j'ai pu acquérir de nouvelles notions telles que l'utilisation et la génération d'un data-model. Egalement, la création et la modification de fichier **bash**.

Par la suite, cela m'a permis :

-d'harmoniser le data-model utilisé par le service.
-d'établir des règles de nommage et de saisie afin de veiller à une organisation claire et précise des structures de données.

Lors de ces missions, j'ai compris le métier d'ingénieur en développement logiciel et comment travailler en autonomie sur des notions que je ne maitrisais pas.

2.Difficultés rencontrées

1.L'outil interne BIE-BIM

Lors du 2e projet, plusieurs difficultés ont été rencontrées Principalement l'outil interne BIE-BIM, sa compréhension fût complexe.

Pendant l'utilisation de cet outil, aucune aide n'était possible car le service n'utilisait pas cet outil et l'outil est interne à Thales donc les recherches internet ne menaient à rien. Cependant, j'ai pu obtenir de l'aide par un ingénieur d'un autre service qui m'a expliqué son fonctionnement.

Cela m'a permis d'apprendre à résoudre des problèmes en autonomie et de pouvoir les expliquer.

Lors de la génération du code en ADA, plusieurs problèmes ont été rencontrés :

-L'association entre le programme développé et généré a été compliqué car l'outil n'est pas adapté au PC non habilité ce qui a été reflétée par quelques bugs.

- Trouver et comprendre la méthode à suivre.

Etant donné que le service n'utilise pas le data-model et le code généré en ADA, le modèle de données n'était pas à jour, des problématiques se sont donc présentés :

- Comment réorganiser ce data-model ?
- Quelles sont les règles de structure et de nommage à suivre ?

J'ai donc dû prendre des responsabilités et établir moi-même les règles à suivre et la manière d'organiser le data-model afin d'être cohérent lors de futures modifications.

2.Le basculement ClearCase-GitHub et le fichier Makefile

Au début de la mission, une autre problématique s'est présentée : Sur quelles plateformes je devrais livrer mon travail ? (Le service était en plein basculement de **ClearCase** à **GitHub**). J'ai donc dû travailler avec les 2 dispositifs.

Après avoir assimilé la notion de **Makefile** , j'ai effectué quelques modifications sur celui réalisé par le service afin de pouvoir compiler sous Linux. Comme je n'avais jamais vu ce type de fichier auparavant, la compréhension du fichier produit par plusieurs ingénieurs fût compliquée.

Après avoir réussi la compilation, un ingénieur du service avait modifié en parallèle le fichier donc j'ai dû comparer son travail par rapport au mien et implémenter mon travail en tenant compte des modifications effectuées sur ce même fichier (notion de **Merge**) et de pouvoir livrer mon travail.

Rendre compatible le composant sous SOLARIS et Linux fut compliqué : des modifications conséquentes ont été demandées. De plus, il fallait réaliser cette compatibilité en très peu de temps car le basculement ClearCase-GitHub était proche.

3.Les tests unitaires en ADA du passage Big/Little Endian

Des difficultés ont été rencontrées sur les tests unitaires en ADA permettant de tester le passage du Big Endian au Little Endian. Le langage ADA est un langage compliqué dû à son fort typage mais également à son stockage mémoire qui n'est pas facile à comprendre. Cette difficulté se reflète lors de la réalisation des tests unitaires en ADA.

3.Progrès à réaliser

Au cours de cette première année d'apprentissage, j'ai pu assimiler plusieurs notions et acquérir de nouvelles compétences comme évoqué précédemment.

Pour l'aspect technique, je devrais approfondir mes connaissances des fichiers **bash**, **Makefile** afin d'être plus à l'aise dans l'avancement de la mission.
Prendre plus de recul sur les difficultés rencontrées afin de pouvoir les résoudre plus rapidement est aussi un autre axe d'amélioration.

Pour l'aspect social, je voudrais prendre plus de responsabilités en menant les réunions liées à ma mission et être force de proposition.
Je voudrais également être à 100% autonome sur les tâches que l'on me demande de réaliser.

Conclusion

Dans l'ensemble, cette première année d'apprentissage m'a permis :

- d'assimiler plusieurs compétences techniques : l'apprentissage d'un langage de programmation, modification de fichiers conséquents et assimilation de règles de nommage.

-avoir une première expérience du travail en équipe au sein d'une grande entreprise telle que Thales DMS.

-acquérir mes premières relations en entreprise.

-comprendre le fonctionnement d'une grande entreprise.

J'ai pu découvrir le monde de l'aéronautique et les coulisses du déroulement d'un projet comme le Portage Linux. J'ai pu voir de plus près le travail d'un ingénieur : son quotidien, les difficultés qu'il peut rencontrer, sa manière de faire face à un problème.

Je tiens à remercier toutes les personnes qui m'ont permis d'avoir tout ce contenu à vous partager.

Bibliographie

www.thalesgroup.com/fr/candidat/decouvrir-thales

www.lesechos.fr/2017/12/thales-un-geant-tres-diversifie-190070

https://fr.wikipedia.org/wiki/Thales_LAS_France

https://en.wikipedia.org/wiki/File:Big-little_endian.png

Glossaire

Ada : Langage de programmation ayant un fort typage permettant de faciliter le traitement de données.

Tests unitaires : Procédé permettant de vérifier le bon fonctionnement d'un programme

Big Endian : Ordre de bits de stockage dans lequel le bit le plus important (= valeur la plus significative dans la séquence) est placé en première position de stockage.

Little Endian : Ordre de bits de stockage dans lequel la valeur la moins importante (=valeur la moins significative dans la séquence) est en première position de stockage.

GitHub et **ClearCase** : gestionnaires de configuration destinés aux développeurs de logiciels.

Jira : Système de gestion de projets développé permettant d'organiser les différentes tâches d'un projet.

Data-model : Modèle contenant des structures de données d'un système d'informations.

Message : Structure de données d'un système d'information.

BIE-BIM : Outil interne contenant le data-model permettant sa saisie, modification et génération.

Makefile: Fichier contenant un ensemble de conditions et règles afin d'organiser les parties d'un programme à compiler selon les conditions présentes.

Bash: Fichier contenant un ensemble de commande Linux à exécuter.

Merge: Comparaison du travail réalisé avec le travail précédemment publier afin de tenir en compte les modifications réalisés en l'implémentant.