



Rapport de stage DUT GEII : Coll Dump Fichier

Tuteur pédagogique :
M. Mohamed TFAZZOLI

Tuteur entreprise :
M. Pascal SAPEY

Année d'étude :
2014-2015

Date du stage :
25/03/2015 – 05/06/2015



Rapport de Stage IUT

GYBELS Hugo

29 mai 2015

Table des matières

Abstract	4
Remerciements	6
Introduction	8
Contexte	10
2.1 Description de l'entreprise	10
2.2 Comptage Paramétrage	11
2.3 Platine	12
2.4 Coll Dump Fichier	14
Missions du stage et cadre de travail	15
3.1 Objectifs	15
3.2 Enjeux	16
3.3 Cadre de travail	16
Travaux réalisés	18
4.1 Optimisation de l'outil	18
4.1.1 Méthodes/environnement de travail	18
4.1.2 Fonctionnement du code	23
4.1.3 Localisation la perte de performance	25
4.1.4 Amélioration de la partie décodage	29
4.1.5 Résultats obtenus	33

4.2	Rédaction de la documentation	35
4.2.1	Documentation utilisateur	35
4.2.2	Documentation développeur	36
	Enseignements	39
5.1	Compétences acquises	39
5.1.1	Langage C++	39
5.1.2	Gestion de configuration	40
5.1.3	Nouveaux logiciels	41
5.1.4	Langage html5 et css3	41
5.1.5	Création des documents de l'historique du stage	42
5.2	Apports personnels	43
	Conclusion	45
	A - Organigramme	48
	B - Planning	49
	C - Diagramme UML	50
	D - Documentation	51

Abstract

Résumé

Depuis de nombreuses années, le monde de télécommunications a une place essentielle dans notre vie quotidienne. Tout le monde utilise les réseaux des différents opérateurs dont Orange qui comptabilise à elle seule 236 millions de clients. Ces opérateurs ont dû mettre en place des installations permettant de traiter un tel volume de données.

J'ai effectué mon stage de fin de cursus de DUT Génie Électrique et Informatique Industriel au sein de l'entreprise Orange à l'Isle d'Abeau. Dans ces locaux, tout le monde s'occupe de faire fonctionner l'application qui traite les comptes-rendus de toutes ces communications (7 milliards de comptes-rendus par jours) : l'application Platine.

En bref, mon stage a consisté, dans un premier temps, à optimiser un outil utilisé comme aide à l'exploitation de Platine : **Coll dump fichier**. Cet outil a pour rôle de décoder et d'afficher un de ces comptes-rendus. Son exécution demande trop de temps : je devais gagner 50% du temps de traitement. Dans un second temps, j'ai dû rédiger une documentation présentant l'outil, son utilisation, son fonctionnement mais aussi le contenu de son code.

Pour finir ce stage m'a amené à utiliser de nombreux outils informatiques plus ou moins complexes qui ont changé les méthode de travail que j'utilisais.

Abstract

For many years now, telecommunications world represents a major part of our daily life. Everyone uses operators networks as Orange which has itself 236 million customers. That's why those operators had to develop tools which can process that much data.

I did my final internship of my DUT GEII's formation in the business Orange Isle d'Abeau. In this place, everyone takes care of the application which processes all the reports of those conversations (7 billion reports per day) : this application is named Platine.

In short, I had two assignments during my internship : the first was to optimize a tool used as an aid to the operation of Platine : **Coll dump fichier**. This tool is used to decode and display one of those reports. Its execution time require too much time : I had to save 50% of execution time. The second was to write a documentation which presents the tool, how it's used, how does it works but also how does the code works.

Finally, this internship leads me to use a lot of tools which learnt me a new way to work.

Remerciements

Avant toute chose, je tiens à remercier toutes les personnes qui ont contribué de près ou de loin au bon déroulement de ce stage et qui m'ont permis la rédaction de ce rapport.

Plus précisément, je tiens à remercier Nadine RUPEK-NGUYEN pour m'avoir accueilli chaleureusement au sein du service Comptage d'Orange à l'Isle d'Abeau, et pour la confiance qu'elle m'a accordé.

Je remercie également vivement Mr Yannis MIKLER, pour les missions qu'il m'a confié, et pour les efforts et le temps qu'il a fournis afin de m'intégrer le mieux possible dans l'ensemble des activités du service, que se soit pour mon arrivée, mes travaux, ma soutenance ou pour mon départ.

J'exprime également mes remerciements à Mr Pascal SAPEY, mon tuteur entreprise, pour ses conseils et ses enseignements qui m'ont permis de comprendre le métier de paramétreur ainsi que son environnement ; ce qui m'a offert la possibilité de progresser sans cesse pendant ces 10 semaines de stage.

Je remercie de même Mr Mohamed TAFFAZOLI, tuteur pédagogique, pour son intérêt et pour le suivi qu'il a apporté à mon stage.

J'adresse de plus mes remerciements à Mr Patrick KHALFA pour m'avoir apporté à tout moment son aide technique et pour le partage quotidien de sa connaissance du sujet sur lequel j'ai travaillé.

Je souhaite également adresser mes remerciements à toute l'équipe pédagogique de l'IUT de Saint-Étienne, en particulier les responsables de la formation "Génie Électrique et Informatique Industriel", ainsi que les enseignants de l'école de Telecom Saint-Étienne pour leurs

enseignements et pour m'avoir apporté les éléments nécessaires à la réalisation de ce travail.

Enfin je remercie les stagiaires et alternants du service qui m'ont agréablement accueilli et qui ont rendu ce stage des plus agréable.

Pour finir, je tiens à remercier toutes les personnes qui m'ont soutenu, conseillé et relu lors de la rédaction de ce rapport de stage : ma famille et mon amie Margot JOUVE.

Introduction

Le monde des télécommunications, force majeure de l'innovation, attire un volume important des compétences dans les domaines de l'informatique, des réseaux mais aussi de la communication.

Étant moi même très intéressé par le développement informatique et les nouvelles technologies, c'est avec plaisir et enthousiasme que j'ai accepté le stage qui m'a été proposé par l'entreprise Orange à l'Isle d'Abeau, dans le cadre de la validation de mon DUT en Génie Électrique et Informatique Industriel.

Malgré la distance quotidienne que cela représentait, j'ai choisi ce stage premièrement pour la tâche qui m'a été proposée : intégré dans une équipe de développeurs, je devais, dans un premier temps, m'approprier un logiciel codé en C++ afin de l'optimiser. Ce projet m'a donc demandé de mettre en pratique beaucoup d'éléments de ma formation. Puis secondement, je devais rédiger une documentation technique de cet outil détaillant son utilisation ainsi que son fonctionnement. Je devais donc, à la fois améliorer un code dans un langage qui m'est familier avec lequel j'ai rapidement été efficace, et dans le même temps effectuer un travail de synthèse et de rédaction ; part importante des métiers vers lesquels je m'oriente. Ce thème correspondait pleinement à mes attentes de par son côté plaisant et formateur pour ma future poursuite d'études en tant qu'ingénieur informaticien.

De plus, l'entreprise Orange, acteur principal des télécommunications et opérateur principal en France, m'offrait l'opportunité d'évoluer dans un cadre professionnel similaire à celui auquel je me destine. Ceci était un point qui m'importait réellement dans le choix de mon stage. Le monde de l'éducation étant très différent du monde de l'entreprise j'attendais sincèrement de ce stage qu'il conforte ou non mon choix d'orientation.

L'élaboration de ce rapport a pour principale source les enseignements tirés de la pratique quotidienne des tâches qui m'ont été confiées. Ces enseignements n'auraient pas été possibles sans l'excellent environnement de travail incluant à la fois le poste et les entretiens que j'ai pu avoir avec les employés du service.

Je vais donc maintenant détailler le déroulement de mon stage, mais pour mieux le comprendre, je commencerai par présenter le contexte dans lequel est venu s'inscrire ce dernier, puis je présenterai les travaux que j'ai réalisés ainsi que les résultats que j'ai obtenu, et pour finir, je ferai un retour sur les enseignements que m'a apporté ce projet.

Contexte

2.1 Description de l'entreprise

Orange est le premier opérateur Français dans le domaine mobile, fixe/voix et haut débit fixe. Anciennement connu pour son statut d'opérateur national sous le nom de France Telecom (changement d'appellation en juillet 2013), Orange est implanté dans le monde entier : principalement en Europe, Afrique, Moyen-Orient et même aux Caraïbes. Elle est leader dans la majorité des pays où elle est implantée.

La figure ci-dessous résume les trois chiffres importants du groupe : Orange emploie 165.000 salariés au service de 236 millions de clients dans le monde entier, et a rapporté 40,9 milliards d'euros de chiffre d'affaires en 2014.

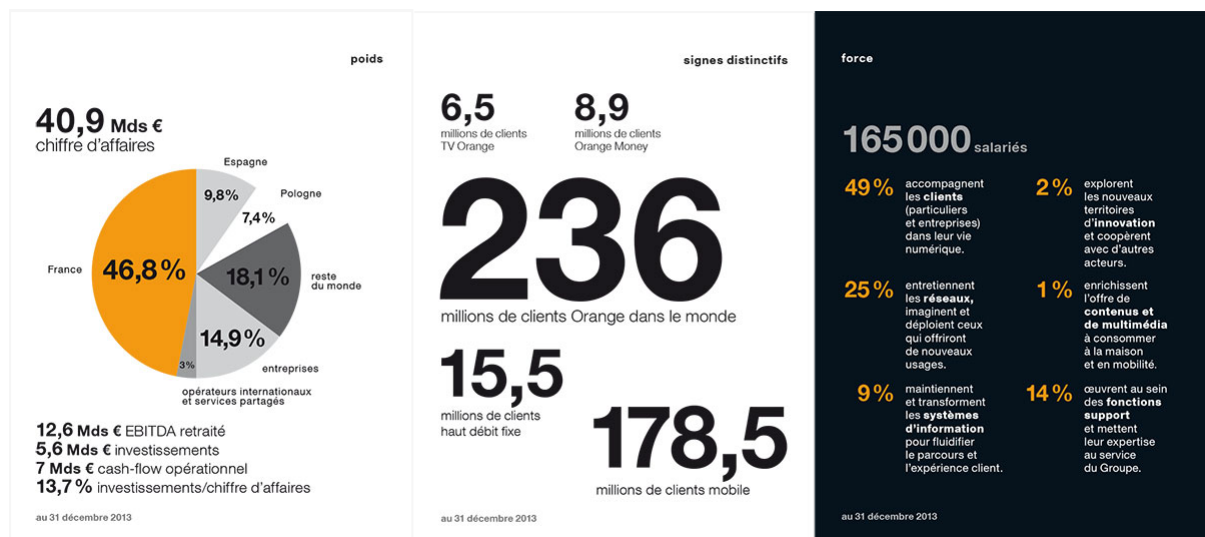


FIGURE 2.1 – Chiffres Significatifs Orange 2014.

Toutefois, Orange France constitue une partie importante du groupe : elle participe à la hauteur de 47% du chiffre d'affaires réalisé. Orange France employait en 2013, seulement en France, plus de 80.000 personnes et récoltait un chiffre d'affaires supérieur à 20 milliards d'euros.

2.2 Comptage Paramétrage

Orange est ensuite divisé en plusieurs directions ayant chacune une tâche précise. Le département du comptage, dans lequel j'ai effectué mon stage, appartient à la Direction des Réseaux (DR). Elle gère l'exploitation de l'ensemble des réseaux d'Orange France : fixe, mobile, wifi et internet sur les mobiles. Elle s'occupe ainsi de la supervision et l'expertise des ces réseaux. La DR appartient elle même à la Direction des Réseaux et Services (DERS) qui est une entité de la Direction Technique et du Système d'Information (DTSI) (*Cf. Annexe A - Organigramme*).

La direction Comptage a trois rôles majeurs. Le premier est de collecter les données d'usage client. C'est à dire les comptes-rendus d'appels émis par toutes les antennes de France. Elle s'occupe notamment d'archiver ces rapports. Son deuxième rôle est de mettre en forme ces données de façon à ce qu'elles soient par la suite traitées extrêmement rapidement par la suite. Son troisième et dernier rôle est donc de redistribuer ces comptes-rendus pré-traités (mis en forme) vers le Système d'Information (SI) d'Orange France : facturation, reversement, infocentre.

Le comptage est encore divisé en trois équipes représentant au total 57 CDI¹ :

L'exploitation : Elle utilise l'application Platine (*Cf. Partie Platine*). Elle est en relation avec les clients émetteurs (réseau) et consommateurs (facturation) du comptage.

Lorsqu'il y a une erreur dans le traitement des données, elle est le premier niveau de maintenance, lorsqu'un problème persiste elle fait appel à l'expertise.

L'expertise : Elle représente un niveau supérieur de connaissance de l'application Platine qui traite les données. Les experts ont deux rôles : ils interviennent lorsqu'un problème

1. Contrat de travail à Durée Indéterminée.

apparaît en exploitation et que les exploitants n'arrivent pas à le résoudre. Ils s'occupent aussi de tester les nouveaux développements avant de les mettre en production. Ils sont également la médiation entre l'équipe du paramétrage et l'exploitation : lorsqu'un nouveau service apparaît, les experts sont chargés de tester les outils développés par le paramétrage avant de le livrer en exploitation. Ensuite ils maintiennent cet outil dans l'exploitation et ces précédents tests leur confèrent une bonne connaissance des outils.

Le paramétrage : Il s'occupe du développement des futurs services (créés par le service de Marketing). Lorsqu'un nouveau service ou une mise à jour va être livré par les autres directions d'Orange, l'équipe du paramétrage développe ou adapte les outils disponibles. Les paramétreurs travaillent en amont des différentes livraisons. Ils produisent tous les développements de la Direction Comptage.

L'ensemble de la Direction Comptage est donc centrée autour du fonctionnement de l'application qui collecte, met en forme et distribue les données du réseau : Platine. Je vais donc détailler le fonctionnement et le rôle de cette application.

2.3 Platine

Le paramétrage fait donc partie du pôle de Médiation qui fait le lien entre le réseau et le système d'information (SI), autrement dit les applications clientes du type facturation ou enquêtes juridiques par exemple. Ce lien est réalisé grâce à l'application Platine. Elle fonctionne principalement sur trois modules : la collecte, le traitement, puis la distribution.

La première fonction de Platine est de collecter les comptes-rendus d'appels (CRAs) envoyés par les antennes du réseau. Ces CRAs peuvent être envoyés dans tout les types de formats. Ils sont conservés tels qu'ils ont été reçus dans le module d'archivage.

La prochaine étape est de mettre en forme ces comptes-rendus. Elle va donc ensuite convertir tous les CRAs dans un format commun : le TLV (Type-Length-Value). Pendant cette étape, on effectue aussi une action de sélection : seuls les champs utiles aux traitements ultérieurs sont conservés. Certains champs sont coupés à cette étape. Les CRAs sont

maintenant mis en forme et sont prêts à être traité efficacement par les applications du SI.

La dernière étape est d'envoyer ces CRAs pré-traités vers le Système d'Information qui va effectuer principalement les opérations de facturation.

Platine prépare donc le traitement des données envoyées par les EEC² aux ECC³ de façon à faciliter et à accélérer par la suite le traitement de ces données. Voici un schéma résumant le rôle de Platine dans le pôle médiation.

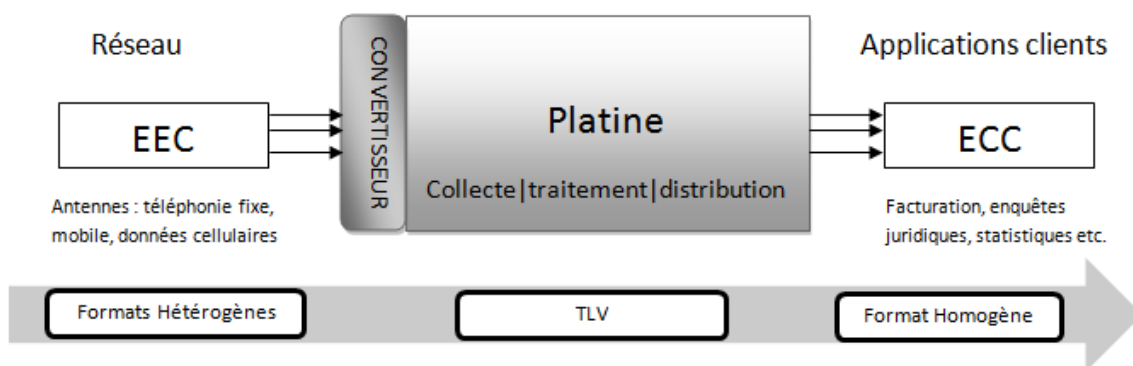


FIGURE 2.2 – Fonctionnement de Platine

L'application Platine traite des milliards de CRAs tous les jours. Pour gérer un tel flux, tous les CRAs sont convertis et surtout allégés par Platine avant d'être transmis, et ce dès leur entrée dans l'application. Ainsi, une fois dans l'application, le compte-rendu ne ressemble plus en rien à celui qui a été reçu.

Cependant, certaines fois, on retrouve des CRAs erronés qui ne passent pas correctement dans l'application. Ces CRAs ne sont donc pas envoyés vers les applications clientes (SI). Il est donc possible que des appels ne soient pas facturés ou non transmis dans le cadre d'une enquête. Lorsque cela arrive, il faut déterminer si l'erreur vient de l'antenne réseau ou de Platine. Ici arrive un problème : les CRAs ayant été convertis et modifiés, on ne peut plus communiquer avec l'équipe qui s'occupe de l'antenne réseau pour savoir si le CRAs reçu est identique à celui envoyé ou encore pour indiquer sur quel champ se situe notre erreur. C'est

2. Équipement Émetteur de Comptes-rendus d'appels (émetteur)

3. Équipement Consommateur de Comptes-rendus d'appels (récepteur)

ici qu'intervient l'outil Coll Dump Fichier sur lequel j'ai travaillé.

2.4 Coll Dump Fichier

Comme je l'ai expliqué précédemment, Platine simplifie les CRAs avant des les traiter. Toutefois, le CRA initial n'est pas perdu : le module de collecte a archivé les comptes-rendus au format natif dans des fichiers conservés 5 jours dans l'application avant d'être envoyés en archive. Platine ajoute un préfixe à chaque fichier (une entête) précisant le numéro de l'EEC, la date de création du fichier etc. L'outil Coll Dump Fichier permet la lecture et le décodage de ces fichiers, quel que soit le format utilisé. **Coll Dump Fichier permet ainsi d'afficher le contenu d'un CRA reçu par platine avant qu'il ne soit converti en format TLV ; cela permet d'observer le message exact ayant été envoyé par l'EEC.** Ceci permet de localiser si le défaut vient en amont de platine ou s'il est dû à un module de l'application. De plus, lorsque l'on contacte l'équipe s'occupant des antennes, on peut facilement communiquer. Voici ci-dessous un schéma résumant l'emplacement et le rôle de l'outil Coll Dump Fichier par rapport à l'application Platine.

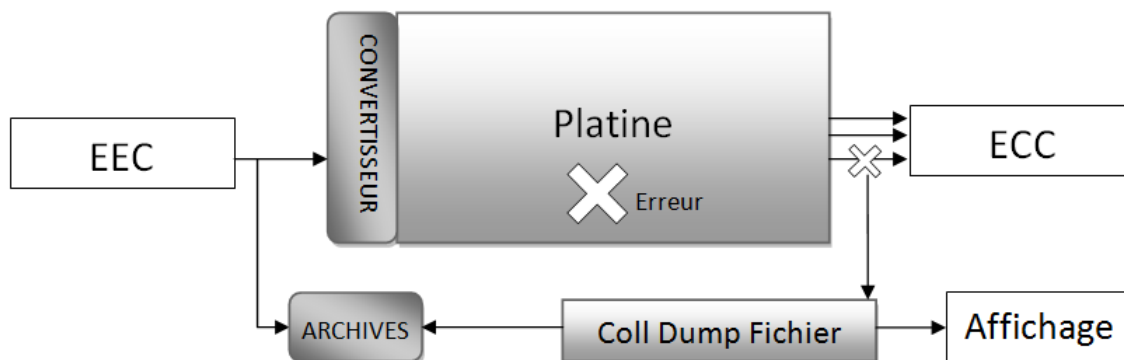


FIGURE 2.3 – Emplacement et rôle de Coll Dump Fichier.

Missions du stage et cadre de travail

3.1 Objectifs

L’outil Coll Dump Fichier est un outil qui a été codé en interne, c’est-à-dire par des développeurs d’Orange. Il était tout d’abord codé en C. La version codée en langage C était performante. Cependant, l’information de traitement étant codée en dur, à chaque modification d’un type de traitement, modification d’un champ, ou tout autre modification, il fallait modifier le code source, recompiler l’ensemble du projet, puis réinstaller sur toutes les machines la nouvelle version de l’outil. Ceci demandait un temps de maintenance conséquent. De plus, étant donné la rapidité à laquelle évoluent et apparaissent les types de traitement, ces opérations devenaient de plus en plus fréquentes. Il est rapidement apparu la problématique de la maintenance de cet outil.

La solution apportée à ce problème a été de recoder l’outil en apportant au code des fichiers externes qui contiennent les données nécessaires au décodage de chaque type de traitement. Ce code a été réalisé, en C++, et en grande partie par un précédent stagiaire en troisième année de licence informatique. N’ayant pas eu le temps de traiter tous les types de traitement, d’autres personnes de l’équipe ont complété le code et ont traité tous les types de traitement. Il est maintenant opérationnel.

Toutefois, deux problèmes majeurs se sont présentés. Premièrement, depuis le passage en C++, on observe une importante perte de performances. Le temps de traitement d’un fichier est beaucoup plus long. Deuxièmement, le code étant long et relativement complexe, son développement a demandé la majeure partie du temps que possédait le précédent stagiaire ; au détriment de sa documentation. La reprise de l’outil par les personnes de l’équipe a été

très complexe et son maintien et optimisation le sont encore aujourd'hui.

C'est dans ce cadre qu'est venu s'inscrire mon stage. En effet, il m'a été confié deux missions :

- Optimiser le temps de traitement de l'outil en C++. L'objectif qui m'a été fixé était de gagner approximativement 50% du temps de traitement d'un fichier de collecte.
- Écrire une documentation qui soit à la fois technique, adaptée pour les développeurs et utilisateurs, qui explique le rôle et le fonctionnement de l'outil.

3.2 Enjeux

Il y a plusieurs enjeux derrière ces missions. Tout d'abord, l'outil Coll Dump Fichier est beaucoup utilisé dans le département. Il est utilisé par les équipes d'expertise et d'exploitation. Il y a donc des attentes quant aux résultats du service entier. Il faudra que le produit final soit toujours fonctionnel et que l'amélioration soit visible de façon à satisfaire ces attentes.

De plus, il arrive que Platine subisse des coupures, prévues ou non, qui peuvent durer plusieurs heures. Dans ces cas, il faut décoder l'intégralité des CRAs reçus par Platine pendant la coupure, grâce à l'outil Coll Dump Fichier. Ce décodage peut durer jusqu'à plusieurs jours depuis la mise à jour C++. Dans ce contexte, une amélioration de 50%, qui me permet gagner quelques secondes sur des tests unitaires, permettrait aux exploitants de gagner un temps considérable.

Pour finir, les réseaux continuent d'évoluer en parallèle de mon travail. Il va donc falloir continuer à maintenir cet outil après mon départ. Cependant, l'organisation de l'équipe est amenée à être modifiée (Temps Partiel Senior, changement de service etc.). Il va donc apparaître la nécessité d'une documentation claire permettant une maintenance par les futurs utilisateurs.

3.3 Cadre de travail

Mon stage s'est déroulé dans les locaux d'Orange implantés à l'Isle d'Abeau. Ces locaux sont modernes, bien équipés et se situent dans un contexte calme, avec beaucoup de verdure, en banlieue de l'Isle d'Abeau. Les employés sont très accueillants, je me suis facilement senti intégré au sein de l'entreprise. Un petit déjeuner avait été prévu dès mon arrivé afin que je puisse me présenter et expliquer ma présence au reste du département. J'avais notamment accès au restaurant d'entreprise dans lequel ce fut un plaisir de manger pendant toute la durée de mon stage pour un tarif voisin des restaurants universitaires. Finalement, Orange recrutant beaucoup d'alternants, j'ai facilement pu trouver ma place auprès d'autres étudiants dans l'entreprise, notamment pour les repas, conseils et autres activités.

Au sein de l'équipe du paramétrage, un poste de travail équipé d'un poste informatique performant et agréable m'a été confié me permettant d'effectuer des recherches, de développer ou de rédiger les documents de ce stage dans des conditions très agréables. De plus, des réunions hebdomadaires étaient organisées avec toute l'équipe. Je me suis réellement senti intégré dans ces réunions. Elles m'ont permis de faire le point régulièrement sur l'avancement de mon projet, d'échanger sur les différents choix à faire, et de connaître les volontés du service quant aux travaux que je réalisais. Pour finir, mon poste était installé dans le bureau d'un autre employé, Patrick KHALFA, qui a lui aussi beaucoup travaillé sur l'outil Coll Dump Fichier et qui a pu m'apporter beaucoup d'aide dès que j'en avais besoin sur des points techniques du code. Pendant ces réunions, il m'a été proposé de présenter mes travaux ainsi que le déroulement de mon stage au reste de l'équipe. Cette présentation était ouvert à l'ensemble du département et notamment à toutes les personnes utilisant Coll Dump Fichier. Ce fut un excellent exercice de communication ainsi qu'une très bonne préparation à la soutenance de mon rapport.

En conclusion, l'environnement dans lequel j'ai effectué mon stage était idéal pour travailler efficacement et dans la convivialité.

Travaux réalisés

La première étape de mon stage a été de prendre en main le code et son environnement de travail, de le comprendre, puis de l'optimiser.

4.1 Optimisation de l'outil

Je vais expliquer, par la suite, par quelles méthodes de travail, de mesure puis d'optimisation j'ai pu améliorer cet outil.

4.1.1 Méthodes/environnement de travail

Pour commencer, pendant la première semaine de mon stage, j'ai installé mon environnement de travail. Elle a aussi été une semaine de formation pendant laquelle on m'a expliqué le fonctionnement des normes et formats que reçoit l'outil. Elle a notamment été l'occasion d'être formé aux méthodes de travaux que je ne connaissais pas forcément à ce moment. J'ai donc appris plusieurs façons de concevoir un outil et d'organiser mon travail que je n'avais pas eu l'occasion d'étudier pendant ma formation. Je vais donc par la suite expliquer ces méthodes qui, dans le cadre de mon stage, m'ont permis de récupérer le code source du logiciel et de travailler ce dernier.

La première nécessité à été de prendre en main la *gestion de configuration*.

Gestion de configuration

La gestion de version est une méthode de conception, principalement utilisée dans la création de logiciel, qui permet de gérer le code source. Elle est utile sur des projets collaboratifs

tels que Coll Dump Fichier. Elle permet de garder un historique de toutes les versions du projet, souvent stockées sur un "dépôt" en ligne. Tout le monde peut récupérer le code de la version de son choix (opération de *checkout*), y apporter ses modifications, puis créer une nouvelle branche ou une nouvelle version de la branche principale qui contiendra ces modifications (opération de *commit*). Le versionning permet donc de conserver un historique de l'outil. On ne risque plus de le détériorer et les données sont accessibles par tout le monde et ne peuvent être perdues.

Le code source de Coll dump fichier est géré grâce à un système de gestion de versions qui utilise un logiciel subversion (SVN). On utilise pour cela un dépôt sous Orange forge⁴ et un client SVN sur notre IDE⁵. L'IDE utilisé sera Éclipse car c'est le plus utilisé dans le service et qu'il est très adapté à la gestion de versions. Ce fut donc une occasion de découvrir un nouvel environnement que je ne connaissais pas.

Sur ce dépôt Orange forge, on trouve toutes les versions du projet Coll Dump Fichier depuis sa création. Les versions principales, qui sont livrées en production, se situent dans le tronc (trunc). Avant tout développement majeur, il est conseillé de créer une branche séparée du tronc que l'on mergera⁶ plus tard si les modifications sont à conserver. Pour cela on utilise un autre client SVN directement intégré à Windows : *Tortoise*. Ce client est plus performant et plus convivial que le client d'éclipse. Il est conseillé de créer une copie de travail locale lorsque l'on effectue un checkout du projet, autrement dit lorsque l'on récupère les données du dépôt.

L'avantage principal de la gestion de configuration sur un projet collaboratif est de conserver tous les développements qui ont été effectué. De cette manière, on peut modifier le code sans crainte de détériorer le code. Si les modifications ne sont pas intéressantes ou si le code ne fonctionne pas, on peut récupérer une ancienne version. Cette manipulation s'appelle un *revert*. Voici la branche SVN que je me suis créé pour mes travaux :

4. Orange Forge est une plateforme collaborative appartenant à Orange

5. Integrated Development Environment = logiciel de développement.

6. fusion de la branches avec la dernière version du tronc

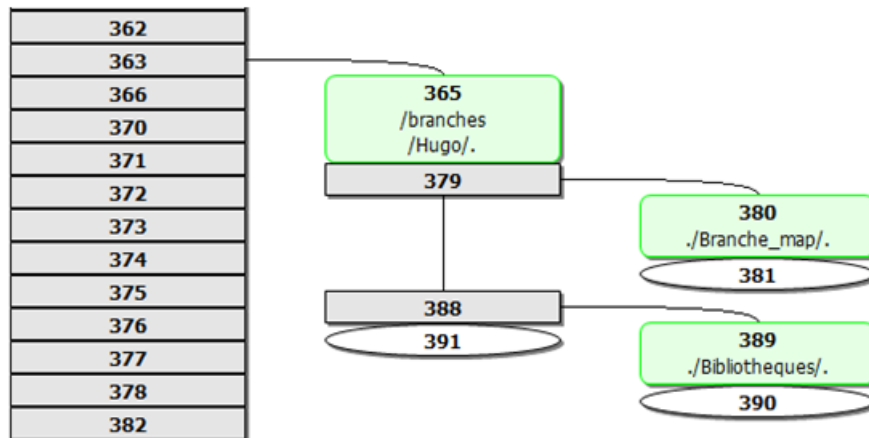


FIGURE 4.4 – Graphique de révision svn.

La gestion de configuration m'a permis de conserver chaque branche de développement. Ainsi, même si certaines n'ont pas apporté de résultat, les codes peuvent être repris si jamais une personne souhaite approfondir la piste explorée. De plus, je garde de cette façon un historique de mes travaux

Une fois que j'avais acquis les bases de la gestion de configuration, je savais comment sauvegarder mes modifications. Cependant, avant de les sauvegarder, il a fallu que je récupère le code source et que je puisse le modifier. Pour cela j'ai installé mon environnement Éclipse.

Création d'un projet eclipse

J'ai ensuite dû créer un projet SVN sous eclipse (l'IDE). Cependant, les machines utilisant cet outil fonctionnent sous Linux et non Windows. Il a donc fallu que je crée ce projet sur une machine utilisant Linux. Nous avons donc effectué un montage Samba utilisant le réseau local pour créer mon projet sur une machine de développement ayant été mise à ma disposition. Cette machine est utilisée par plusieurs personnes de l'étage mais se situe physiquement à Paris.

Éclipse est un logiciel open source beaucoup utilisé et relativement complet. Il permet de base de développer des projets classiques ; cependant, il existe de nombreuses "extensions" que l'on peut installer sous forme de plug-in. C'est notamment de cette manière que j'ai

installé le client SVN. Ce logiciel ressemble relativement à celui que j'utilisais pendant ma formation : Visual Studio 2013. Voici ci-dessous un aperçu de l'environnement d'éclipse.

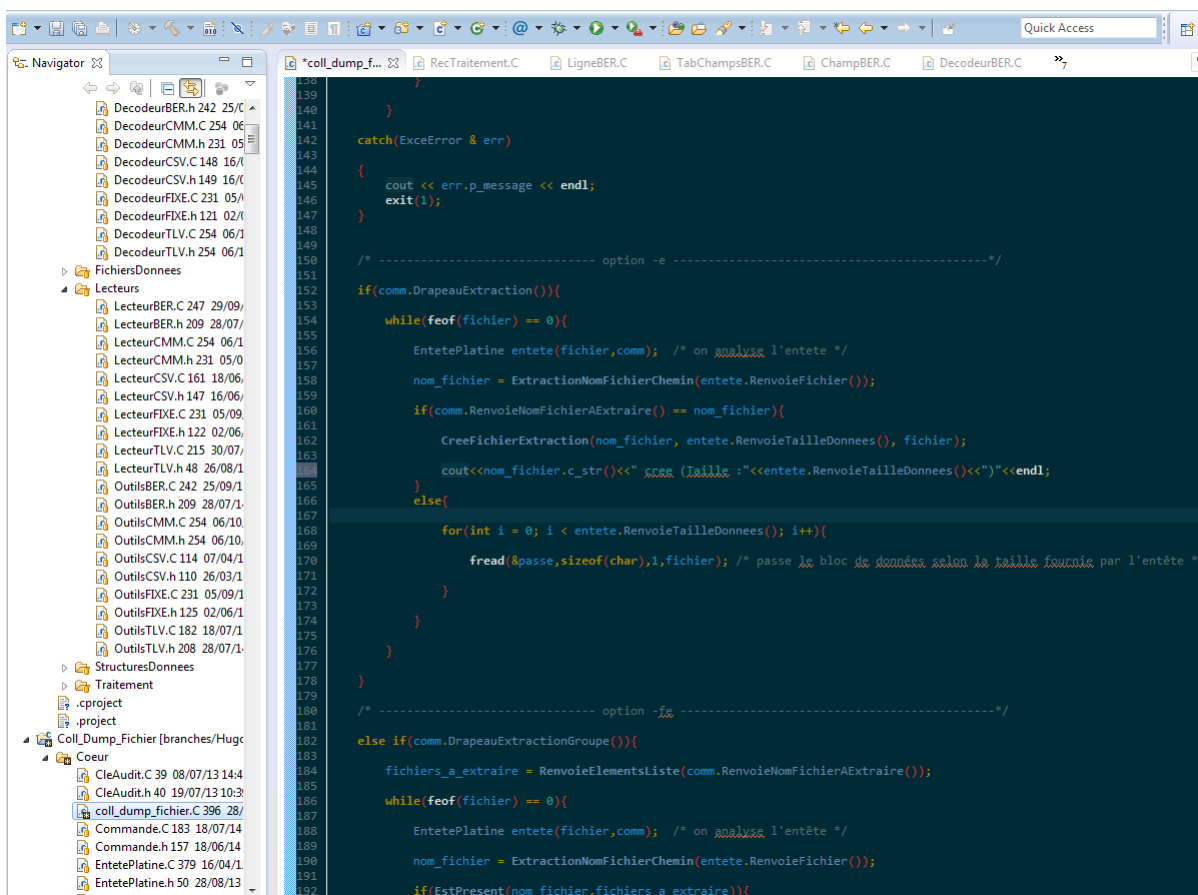


FIGURE 4.5 – Fenêtre d'accueil du logiciel Eclipse.

Une fois le projet créé, j'ai pu récupérer les fichiers sources. Il faut ensuite compiler le projet sous eclipse de façon à générer le fichier Makefile. Cependant, une fois ce fichier généré, on ne compilera presque plus jamais sous eclipse car cela demande beaucoup de temps. On compilera directement sur la machine distante. Le projet compilé, on peut maintenant passer sur la machine distante pour exécuter le programme.

*Remarque : J'ai utilisé un logiciel sous Windows, cependant l'exécutable sera utilisé sous Linux, il a donc fallu modifier le compilateur par défaut de façon à réaliser une **compilation croisée**. Le compilateur utilisé s'appelle Cross GCC.*

Exécution sur la machine distante

Pour exécuter le programme, j'ai dû accéder à distance à la machine de développement. Pour cela j'ai utilisé un terminal grâce à l'outil Putty qui permet d'ouvrir un terminal sur une machine distante. L'ensemble des actions effectuées par la suite seront exécutées sous forme de lignes de commandes. Par chance, j'avais déjà utilisé le logiciel Putty lors de ma formation à l'IUT ainsi que la navigation en lignes de commande sur une machine Linux, ce qui m'a permis de très vite comprendre et d'être autonome sur ce fonctionnement.

Une fois ce plus, cela me confirme la pertinence de ce stage quant à ma formation. Cette dernière semble m'avoir correctement préparé aux technologies utilisées dans le monde de l'entreprise actuel.

La figure ci-dessous montre la configuration du logiciel Putty :

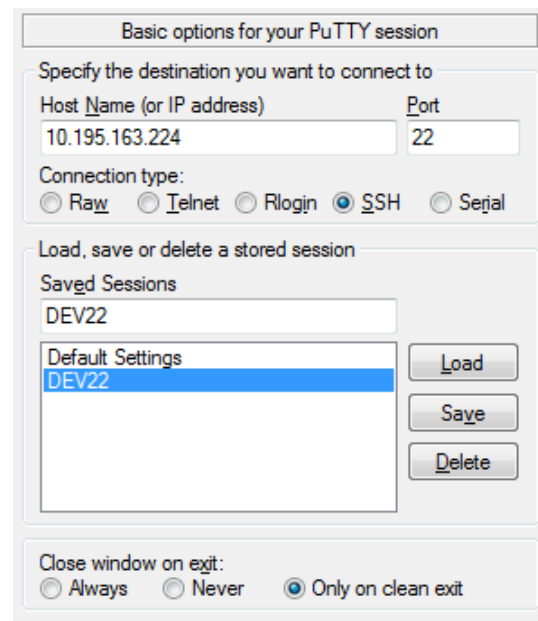


FIGURE 4.6 – Configuration du logiciel Putty.

Une fois le terminal ouvert, on ne compilera plus sous Éclipse mis à part lorsque l'on souhaite actualiser le fichier Makefile (création de nouveaux fichiers sources). Pour compiler et observer les modifications que j'apportais sous Éclipse, je lançais la commande make qui compilait le projet complet.

Je pouvais dès à présent lancer l'exécutable. Je ne détaillerai pas le lancement de la commande "coll_dump_fichier" car cette dernière est relativement technique : il faudrait expliquer chacune des options et le fonctionnement des fichiers de collecte. Cependant, tout le détail de l'utilisation de l'outil est disponible dans la documentation en Annexe (*Cf. Annexe D - Documentation*).

4.1.2 Fonctionnement du code

Prise en main du code

Avant de pouvoir proposer des axes d'améliorations, une grande étape de l'optimisation a été de comprendre le fonctionnement du code. Il a fallu que j'étudie la façon dont sont stockées les données des fichiers externes, les appels des différentes classes qui sont parfois même récursives ou qui font appel à d'autres classes dans le constructeur, le rôle de chaque variable et de chaque instance. Le déroulement du code est assez particulier car la plupart des actions se déroulent lors de la déclaration des variables, c'est à dire dans les constructeurs des classes. De plus, je n'avais jamais vu le traitement des options d'une commande Linux.

Pour comprendre le fonctionnement de ce code, j'ai eu accès au diagramme UML créé par le précédent stagiaire (*Cf. Annexe C - Diagramme UML*). Ce diagramme m'a beaucoup aidé sur la compréhension du cœur du code, cependant, de nombreuses parties restaient floues et se sont éclaircies petit à petit en plaçant des traces dans le code, en utilisant le débogueur GDB pour dérouler le code pas à pas. A chaque étape j'effectuais des premières modifications, encore un peu hasardeuses, qui m'ont tout de même permis de voir les réactions du code et de comprendre son déroulement.

Pendant cette période j'ai notamment commencé à prendre le maximum de notes de façon à me rappeler les points qui freinent le plus la compréhension ; de façon à l'expliquer clairement dans la documentation que je devrai rédiger.

Cette étape m'a demandé un temps relativement important ; cependant, elle était essentielle avant de pouvoir prétendre optimiser l'outil. Une fois que je pensais connaître clairement le fonctionnement du code, j'ai voulu le segmenter de façon à mesurer le temps d'exécution

de chaque étape. Cela me permettra de focaliser mon travail sur les parties problématiques (lecture des fichiers externes, décodage du DC, affichage etc.).

Fonctionnement du code

Avant d'expliquer concrètement mes travaux, il me semble essentiel de présenter en quelques mots le fonctionnement du code. En effet, j'ai déjà expliqué dans quel contexte il est utilisé, mais je vais expliciter brièvement le fonctionnement du code. Bien sur, je garderai toujours une vue très globale avec beaucoup de recul. Je ne donnerai pas chacun des détails du code mais simplement les grandes étapes.

Pour commencer, un rappel sur le principe du code : il doit ouvrir un fichier de collecte (qui contient des comptes-rendus d'appels, ou CRAs). Il va ensuite récupérer le type de traitement de l'EEC, autrement dit, le format ou la norme dans laquelle est codé le compte-rendu. Suite à cela, il va isoler chaque compte-rendu, qu'il va ensuite afficher dans la console avec le détail des champs qu'il contient. Cette étape est justement automatique dans Platine, qui elle transmet automatiquement les DC sans même les afficher.

Ces comptes-rendus sont donc composés d'une listes de champs correspondants à des valeurs. Par exemple, on peut retrouver le champ "Durée de l'appel" avec la valeur 60 secondes. Cependant, dans le fichier on aura pas le nom du champ "durée de l'appel", mais un numéro (ou tag) par exemple le tag 10 qui identifie ce champ. Il faut donc aller chercher la signification du numéro du champ dans le fichier csv qui contient la correspondance entre les tag et le nom des champs. Toujours dans notre exemple, l'outil va chercher, dans le fichier csv du type de traitement, la case avec le tag 10 qui correspondra au nom de champ "durée de l'appel" et le type de données sera "secondes". Voici un schéma qui résume ce fonctionnement :

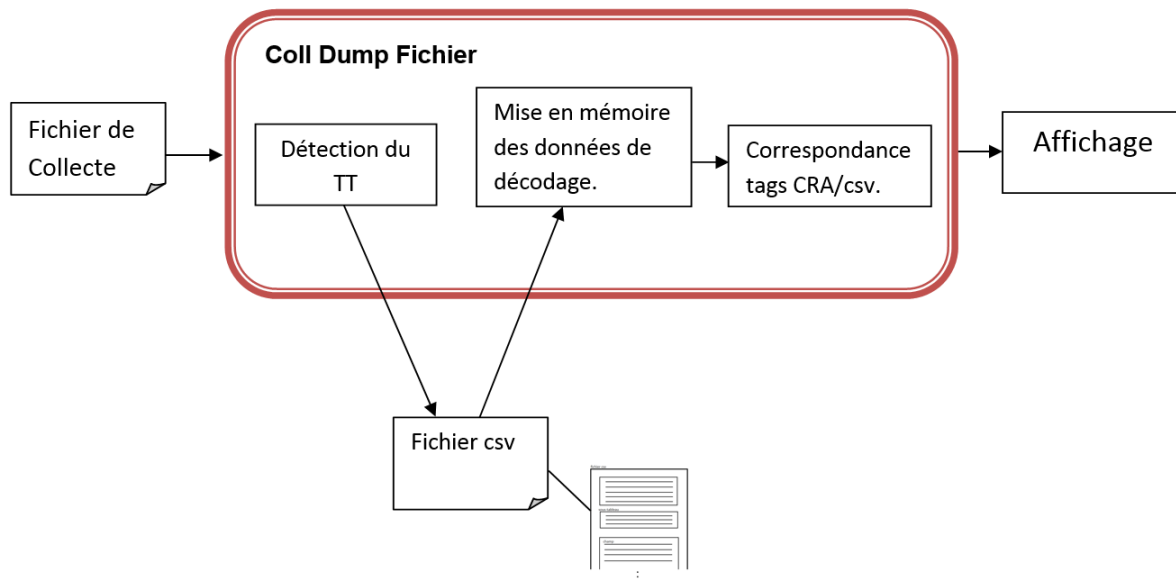


FIGURE 4.7 – Fonctionnement du code de Coll Dump Fichier.

Pour rendre cela peut-être un peu plus concret, je vais vous présenter un exemple d'utilisation. Bien que l'interface ne soit pas graphique, l'affichage dans la console peut être rebutant, c'est cependant réellement de cette manière que l'outil est utilisé par les différentes équipes de la Direction Comptage que ce soit experts ou exploitants. J'ai donc jugé intéressant d'inclure dans ce rapport un exemple d'utilisation classique, quotidienne, de l'outil (*Cf. Figure 4.8 ci-dessous*).

4.1.3 Localisation la perte de performance

Une fois que j'avais correctement analysé le code, mon plus grand problème pour le moment était de localiser la partie du code demandant le plus de temps. En effet, je ne pouvais pas rechercher en parcourant le code fichier par fichier la partie de code qu'il fallait modifier. Il m'a été nécessaire de trouver les outils adaptés pour mesurer le temps d'exécution de chaque étape décrite dans la *figure 4.7* ; et par la même occasion découvrir comment les utiliser. En effet, la mesure du temps est parfois complexe, tout particulièrement dans notre cas car l'exécution du programme se fait via un réseau dont les performances fluctuent selon

```

$ cdf_Hugo -f /deci/pood5325/collecte/fichiersCollecte/coll_0030_010414_15_07_003 -v -d | more
: Importation des donnees de traitement...
-----bloc: #0001-----
version      : 2
eec          : 30
numero       : 0
horodate     : 1396357292 (01/04/2014 13:01:32 GMT)
drapeaux     : 4
taille       : 866013
fichier      : /data/COL1B/coll/data/travail/eec/30/.colI30/TTFILE00_NGEMONAC_8226_f6fedd7f
-----
a180a081c1a181be8c0300002789030e0401940f3333231393030203133412f30312f8a030a140e9f4302558c860853394420
10f085339442037846042850802080101001633f49d01119f2f01059b0702f810313ae11b8707113366106957f59f4501069f5
a515800102810400000065850a92001104338609607600a01aaf188102558c8302000a980543e09c558c9907113386091059f0
300002788030e0401930f3333231393030203133412f30312f89030a140e9f2d02558c9f3207113366106957f59507484c523
064116890000f0960642534330334984064186805634f08507113366106957f58701000000
Composite(1)
  UMTSGSMPLMNCallDR(0)
  mSOriginatingC(1)
    chargeableDuration_MSO(12)=000027 -> 00:00:39
    dateForStartofCharge_MSO(9)=0e0401 -> 01/04/2014
    callingSubscriberIMSI_MSO(5)=02080101001633f4 -> 208010100061334f
    teleServiceCode_MSO(29)=11
    iNMarkingOfMS_MSO(47)=05
    firstCallingLocInfo_MSO(27)=02f810313ae11b -> 208f0113a31eb1
    calledPartyNumber_MSO(7)=113366106957f5 -> 1133660196755f
--More--

```

FIGURE 4.8 – Exemple d'utilisation classique Coll Dump Fichier.

l'occupation du réseau et la charge de la machine. J'ai donc été amené à utiliser plusieurs outils.

Scripts Linux

Pour commencer, on m'avait présenté des tests à mon arrivée comparant les performances de l'outil codé en C et en C++. Ces tests utilisaient des scripts Linux dont le principe est simple. On stocke la date dans un fichier avant d'exécuter une ou plusieurs lignes de commandes. Puis on récupère à nouveau la date à la fin de l'exécution. Il fallait ensuite ouvrir le fichier contenant les dates et effectuer la différence entre les deux données. Cette méthode fonctionnait plus ou moins. C'est-à-dire que le résultat était très affecté par la charge du réseau ; et ce même dans le cas où les deux exécutions étaient très proches. De plus, le résultat était long à interpréter et générait vite beaucoup de fichiers (un script par version et par test avec un fichier de résultat etc.). En résumé, cette méthode présentait de nombreux inconvénients.

Cependant, j'ai continué à l'utiliser dans un cas où elle présentait un net intérêt : elle

permet d'exécuter et de mesurer plusieurs lignes de commandes consécutives. Elle est donc utile pour exécuter des traitements très longs. De plus, sur ces traitements, l'erreur absolue devenait minime par rapport au temps total d'exécution. J'ai donc conservé cette méthode pour mesurer précisément le gain en performance à la fin d'un développement.

Toutefois, j'avais besoin d'une méthode de mesure pour mes tests unitaires afin d'observer au cas par cas l'impact de mes modifications.

Commande Time

La seconde méthode que j'ai utilisé a donc été la commande *time*. Le plus gros avantage de cette méthode est sa rapidité et sa simplicité d'utilisation. En effet, il suffit d'exécuter la commande précédée du terme *time* sous Linux et cette fonction nous affiche la durée qu'a demandé l'exécution en temps réel, en temps utilisateur ainsi qu'en temps processeur (*voir la figure ci-dessous*). Pour ne pas être influencé par l'affichage, on peut rediriger la sortie de la commande vers un fichier null. Cette méthode était très efficace pour avoir un ordre de grandeur du temps d'exécution.

```
$ time coll_dump_fichier -f /deci/pood5325/collecte/fichiersCollecte/coll_0058_04714_14_04_013 -v -d > /dev/null
0m47.45s real      0m40.73s user      0m3.32s system
```

FIGURE 4.9 – Exemple d'utilisation de la commande time.

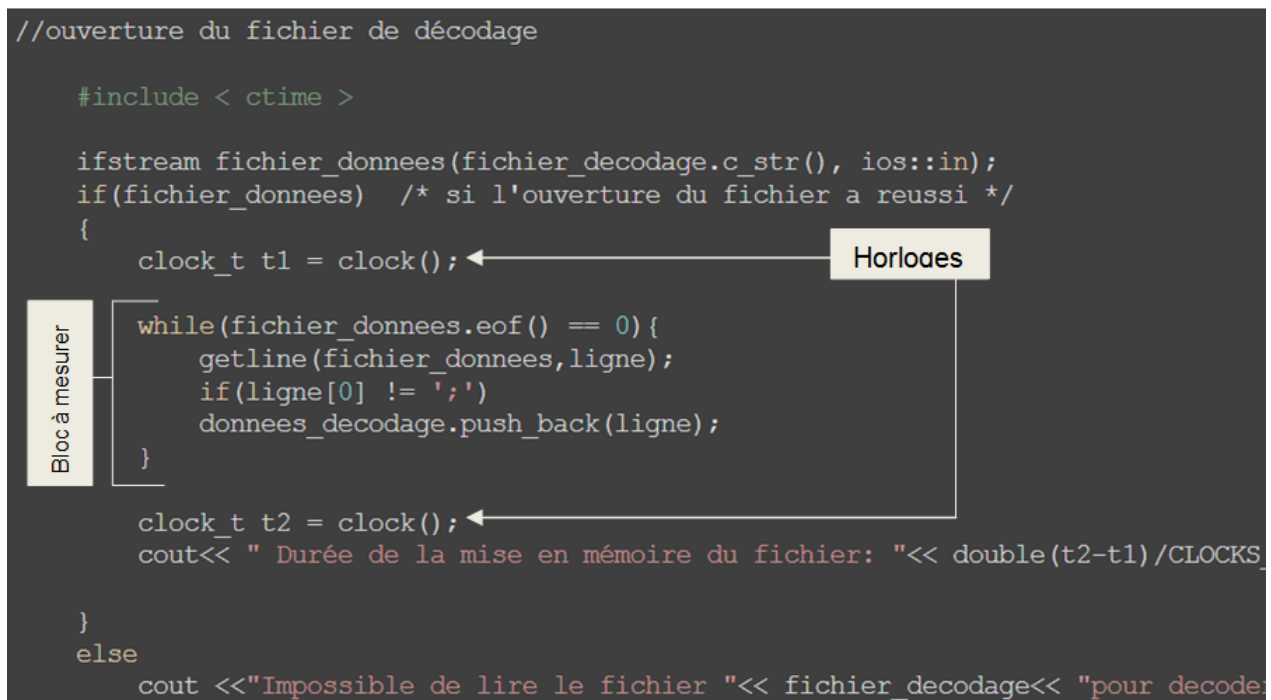
Malgré tout, il demeurait un problème un problème de taille : je ne pouvais mesurer que la durée totale de l'exécution. Pourtant, mon but premier était de localiser, à l'intérieur du code, la partie demandant le plus de temps afin de cadrer mes futurs travaux.

Bibliothèque "ctime"

J'ai donc trouvé une dernière méthode me permettant de mesurer le temps d'exécution d'un code. Cette méthode m'a permis d'encadrer une partie seulement du code et de n'afficher seulement le temps d'exécution de cette partie du programme. Pour cela j'ai utilisé une bibliothèque mentionnée par un membre de mon équipe : la bibliothèque "*ctime*". Cette

bibliothèque permet de placer dans le code des objets de type `clock_t` qui sont en réalité des horloges. on récupère pour chaque objet l'horloge du processeur. En effectuant la différence entre les deux horloges, on obtient le nombre de cycles CPU demandé par la partie de code séparant les deux horloges. On convertit ensuite ce chiffre en secondes en le divisant par le nombre de coups d'horloge par seconde.

Cette méthode ne mesure que les cycles du processeur attribués au programme. On s'affranchit donc en plus des fluctuations dues au réseau. On obtient donc une mesure précise de chacune des étapes du code. Voici ci-dessous un exemple d'utilisation de la bibliothèque `ctime`.



```
//ouverture du fichier de décodage

#include < ctime >

ifstream fichier_donnees(fichier_decodage.c_str(), ios::in);
if(fichier_donnees) /* si l'ouverture du fichier a reussi */
{
    clock_t t1 = clock(); ← Horloges

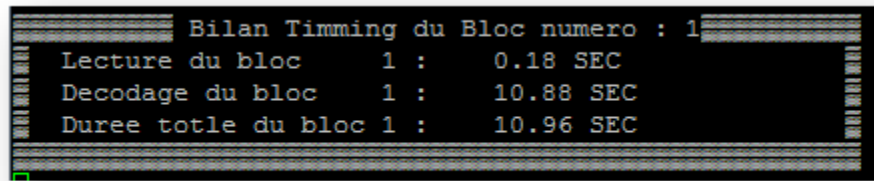
    while(fichier_donnees.eof() == 0){ ← Bloc à mesurer
        getline(fichier_donnees,ligne);
        if(ligne[0] != ';')
            donnees_decodage.push_back(ligne);
    }

    clock_t t2 = clock(); ← Horloges
    cout<< " Durée de la mise en mémoire du fichier: "<< double(t2-t1)/CLOCKS_

}
else
    cout <<"Impossible de lire le fichier "<< fichier_decodage<< "pour decode
```

FIGURE 4.10 – Exemple d'utilisation de la bibliothèque `ctime`.

On voit bien sur ce code que les objet de type `clock_t` me permettent d'encadrer une partie de code afin de mesurer le temps de son exécution. Après une utilisation complète de cette bibliothèque sur le code j'ai obtenu les résultats ci-dessous :



Bilan Timming du Bloc numero : 1			
Lecture du bloc	1 :	0.18	SEC
Decodage du bloc	1 :	10.88	SEC
Duree totle du bloc	1 :	10.96	SEC

FIGURE 4.11 – Résultats temps d'exécution.

Ces résultats ont été très surprenants. En effet, la lecture des fichiers de décodage était la plus grande différence avec le code en C. La plupart des employés pensaient donc que la grande perte de performances qui a accompagné le passage au C++ provenait de la lecture de ces fichiers. Hors ces fichiers étaient aussi la plus grande motivation du passage du C au C++ car ils permettent une maintenance plus simple de l'outil. Les abandonner aurait donc été extrêmement décevant pour les programmeurs qui ont passé une bonne partie des deux dernières années à créer ce code et ces fichiers. Ces résultats confirment donc la pérennité du projet qui était pourtant mise en question.

Dans le doute, j'ai tout de même codé une bibliothèque dynamique pour un type de traitement remplaçant le fichier csv qui était compilée avec le reste du code. Après de nouvelles mesures les résultats sont similaires aux précédents. Tout semble donc montrer que la lecture du fichier de décodage ne pose pas de problème de performance.

Cette nouvelle est donc apparue comme une très bonne nouvelle pour le reste de l'équipe : il est donc possible de retrouver les performances initiales avec des fichiers externes. Le problème vient donc du code de décodage du fichier en C++.

4.1.4 Amélioration de la partie décodage

J'étais désormais sûr que la partie à optimiser n'était dans la partie du code qui permet de stocker les données de décodage en mémoire. Le problème se situe dans le code qui effectue le décodage du CRA en entrée.

Pour résumer le fonctionnement de cette partie, les champs du CRA que l'on reçoit sont identifiés par un numéro de champ (appelé tag). Le CRA contient les numéros de champs et la valeur qu'il contiennent. Cependant, pour l'afficher à l'utilisateur, on a besoin du nom du champs et non de son tag. Pour cela, le fichier de décodage contient un tableau qui effectue la correspondance entre les numéros des champs, leur nom et le type de la valeur qu'ils contiennent. Le logiciel va donc, pour chaque champs du CRA, rechercher le nom du champs dans le fichier avant de l'afficher.

Je ne voyais donc que deux endroits où pouvait se situer le problème :

- Soit la perte de performance est due à la façon de stocker les données de décodage (tableau des champs).
- Soit le problème vient de l'algorithme effectuant la correspondance tags reçus et ceux du tableau.

Stockage des données

L'algorithme de décodage me semblant relativement trivial (boucle for comparant le tableau de tags), j'ai décidé de poursuivre mes travaux en modifiant la façon dont sont stockées les données en mémoire. J'ai donc recherché des conteneurs qui soient plus adaptés que la classe *vector* utilisée jusqu'à présent. La classe qui m'a semblé la plus adaptée a été les conteneurs associatifs de la STL, autrement appelés tableaux associatifs ou encore tableaux map.

Les tableaux map permettent de stocker une valeur et de l'identifier par une "clé". Grâce à cette clé, les tableaux map permettent d'effectuer des recherches extrêmement performantes. Ceci me permettrait d'effectuer la correspondance entre les tags du CRA et du fichier de décodage très rapidement. Ils semblent donc parfaitement adaptés à notre problème.

J'ai ainsi mis en place ce conteneur dans l'intégralité du code. Cependant, à ma grande surprise, les performances n'étaient pas meilleures ; si ce n'est pires. Cet outil semblait pourtant parfait pour ma situation. Afin de comprendre ce phénomène, j'ai recherché des informations plus détaillées sur ces conteneurs, et j'ai trouvé une explication sur le site *www.developpez.com* qui a remis en question ce projet :

Les conteneurs associatifs permettent d'effectuer des recherches d'éléments extrêmement rapides. En effet, les opérations de recherches de recherche se font avec un coût logarithmique [...], ils sont donc particulièrement adaptés lorsque le nombre d'éléments stockés devient grand.

Cependant, dans notre cas, nous n'avons pas un seul tableau map qui contient tous les champs mais plusieurs sous-tableaux qui correspondent aux sous-tableaux du fichier csv. En effet, le fichier csv est composé de sous-tableaux car les champs peuvent être composés de plusieurs sous-champs : par exemple, on pourrait avoir un champ "numéros de téléphone" qui pointerait sur un tableau contenant "numéro appelant" et "numéro appelé". Voici un extrait de ce type de fichier (ici on a seulement trois sous-tableaux) :

bCSMTDPDat		2			
0	ServiceKey	3	0	NULL	0
1	gsmSCFAddr	3	0	NULL	0
bCSMTDPDat					
0	ServiceKey	3	0	NULL	0
1	gsmSCFAddr	3	0	NULL	0
CallDataRecc		15			
0	transit	1	0	TransitExt	0
1	mSOriginatir	1	0	mSOriginatir	0
2	roamingCalli	1	0	roamingCalli	0
3	callForwardi	1	0	callForwardi	0

FIGURE 4.12 – Structure d'un fichier csv.

On n'effectue donc pas de nombreuses recherches sur un très grand tableau ; mais à l'inverse plusieurs recherches très courtes sur des petits tableaux (maximum entre 20 et 30 lignes). Les performance, dites "logarithmiques", perdent donc tout leur intérêt : sur des petits tableaux il est plus intéressant de conserver la classe vecteur.

Ceci nous amène à un problème : si on est sûr que le problème vient de la façon dont sont stockées les données du fichier csv, il faudra soit revoir l'intégralité du fonctionnement du code, soit refaire tous les fichiers csv qui peuvent faire quelques milliers de lignes. Dans les deux cas, ce ne sont pas des projets adaptables à la durée de mon stage.

Algorithme de décodage

Suite à ce constat, j'ai décidé d'explorer le dernier axe de développement qu'il me restait : le décodage. Toutes mes mesures semblaient indiquer que la plupart du temps d'exécution se déroule dans cette partie, peut-être est-il possible de l'optimiser sans avoir à modifier la façon dont sont stockées les données. Cet algorithme de décodage a pour rôle de faire la correspondance entre le numéro des champs du fichier de collecte que l'on veut décoder (tags), que comprend Platine, et le nom du champ de manière à pouvoir l'afficher de façon compréhensible par l'utilisateur.

Dans ce code, j'ai retrouvé à plusieurs endroits des parties de code **redondantes**. En effet, voici un exemple ci-dessous :

```
for(int i = 1; i <= taille; i++){ /* on initialise un tableau contenant les LigneBER né  
  
    tag = atoi(RenvoieElementLigne(donnees_decodage[i],1).c_str());  
    nom_champ = RenvoieElementLigne(donnees_decodage[i],2);  
    suivant = RenvoieElementLigne(donnees_decodage[i],3);  
    flag = atoi(RenvoieElementLigne(donnees_decodage[i],4).c_str());  
  
    p_lignes.push_back(new LigneBER(tag,flag,suivant,nom_champ,donnees_decodage));  
}
```

FIGURE 4.13 – Code lecture original redondant.

Ici, il utilise 4 fois la fonction `RenvoieElementLigne` qui va récupérer une ligne entière du fichier, puis n'en retourner qu'un seul élément. C'est donc un bon exemple de code qui se répète. Au lieu de récupérer quatre fois la ligne on peut simplement la lire une seule fois et retourner la ligne complète. Pour cette partie de code j'ai donc écrit une fonction retournant une ligne complète de façon à ne l'appeler qu'une seule fois :

```

for(int i = 1; i <= taille; i++){ /* on initialise un tableau contenant les LigneBER néces
    ligne_complete = RenvoieLigneTag(donnees_decodage[i]);

    p_lignes.push_back(new LigneBER(tag,flag,ligne_complete[2],ligne_complete[1],donnee
    }

```

FIGURE 4.14 – code lecture final.

Sur cette partie de code, le gain de performances n'était pas exceptionnel. En effet, le code servait à la lecture du fichier externe. Or, nous avons vu précédemment que la lecture ne demandait au total que quelques dixièmes de seconde. J'ai choisi cet exemple car il est facilement compréhensible. Cependant, on retrouvait ce même problème dans de nombreuses parties du code dont la partie de décodage. Dans cette partie il récupérait pour chaque champ à afficher, le tableau entier des champs (fichier csv) avant de pointer sur la valeur qu'il souhaitait récupérer. J'ai donc modifié cet algorithme de façon à ne récupérer qu'une seule fois ce tableau, de le conserver dans une variable locale, puis d'accéder pour chaque champ directement à la case qui nous intéresse. Ceci permet deux avantages : pour commencer, on alloue et désalloue moins de mémoire vive ; mais surtout, ce tableau était un attribut d'une classe du code. Son accès était donc très lourd car, un des grand désavantage du C++, est que l'instanciation de classes ainsi que l'accès aux méthodes et attributs demandent beaucoup de ressources et donc beaucoup de temps. Le langage C++ n'est donc pas très adapté au traitement de données importantes telles que le Big Data.

Lorsque j'ai effectué cette modification dans le décodage, le gain de performance fût beaucoup plus important. J'étais donc très satisfait de cette modification à la vue des temps d'exécutions unitaires. Toutefois, afin de mesurer l'amélioration il a encore fallu que j'effectue des tests de mesure.

4.1.5 Résultats obtenus

Après toutes les modifications précédentes, et au vu de la durée des exécutions, je pensais avoir atteint les 50% d'amélioration du temps de traitement. J'ai donc recompilé le projet tel

que je l'avais lorsque je suis arrivé grâce au gestionnaire de versions afin de pouvoir comparer les temps d'exécutions. Le premier aperçu grâce à la commande `time` semble confirmer mon pressentiment. Toutefois, avant de présenter ces résultats, je les ai vérifiés grâce à un script qui exécutait plusieurs lignes de commandes pour les deux versions et qui inscrit le résultat de cette comparaison dans un fichier nommé *compare* afin d'avoir une résultat plus affiné. Voici ci-dessous les résultats de ces deux tests.

```
$ time cdf_Hugo -f /deci/pood5325/collecte/fichiersCollecte/coll_0058_04714_14_04_013 -v -d > /dev/null
0m41.77s real    0m37.02s user    0m2.52s system
[poood5325@dv57gas1 - /deci/pood5325]
$ time cdf_original -f /deci/pood5325/collecte/fichiersCollecte/coll_0058_04714_14_04_013 -v -d > /dev/null
1m38.17s real    1m26.83s user    0m4.08s system
```

FIGURE 4.15 – Comparaison avec la version originale - Commande `time`.

```
[poood5325@dv57gas1 - /deci/pood5325/TEST_COLL_DUMP_HUGO]
$ compare
[poood5325@dv57gas1 - /deci/pood5325/TEST_COLL_DUMP_HUGO]
$ more resu
/*Nouveau Test */

Coll_dump_fichier_Hugo :      |      Coll_dump_fichier_Original :
Tue May 26 10:13:23 CEST 2015 |      Tue May 26 10:24:42 CEST 2015
Tue May 26 10:19:47 CEST 2015 |      Tue May 26 10:36:51 CEST 2015

=> 6m24s      |      => 12m9s
```

FIGURE 4.16 – Comparaison avec la version originale - Script `compare`.

D'après les mesures ci-dessus, j'ai bien obtenu une amélioration supérieure à 50% du temps d'exécution pour le type de traitement BER. Bien sûr, ces résultats peuvent varier selon la charge de la machine ou du réseau au moment de l'exécution. Cependant, j'ai utilisé les méthodes les moins influencées par le réseau pour ces mesures, et sur de telles durées la différence n'est pas conséquente. Pour finir, sur la moyenne des résultats on est bien au dessus de 50% d'optimisation du temps de traitement. L'objectif est donc atteint pour ce type de traitement. Le type de traitement BER, que j'ai traité, est un des plus complet. Chacune des modifications peuvent être reportées sur les autres types de traitement.

J'ai présenté ces résultats à mon manager, et pour ce dernier les résultats étaient sa-

tisfaisants. De plus, la personne travaillant dans mon bureau a suivi mes modifications et connaît très bien l'outil. Il m'a donc conseillé de passer à la rédaction de la documentation en priorité, puis de revenir sur les autres types de traitement plus tard si j'en ai le temps. Dans le cas contraire, cette personne pourra apporter les modifications sur les autres formats.

4.2 Rédaction de la documentation

La prochaine étape était donc d'écrire la documentation. Pour cela, j'ai été forcé de faire certains choix. En effet, une documentation s'adresse toujours à un public particulier : elle peut être axée conception, marketing, utilisateur, technique etc. Après discussion avec le reste de l'équipe lors des réunions hebdomadaires, il est ressorti deux besoins différents : une documentation utilisateur et une développeur. J'ai donc décidé de créer deux documentations séparées qui auraient des rôles différents.

4.2.1 Documentation utilisateur

Pour la documentation utilisateur, qui aura un rôle plus général, j'ai choisi comme format un document classique qui sera accessible à toutes les personnes utilisant l'outil. Son rôle est de présenter l'outil et d'expliquer dans quel contexte il est utilisé. Elle devra ensuite décrire comment il est utilisé, lister l'ensemble des options possibles et présenter quelques exemples d'utilisation. Une partie présentera brièvement le fonctionnement du code mais sans rentrer dans les détails, le but est réellement de garder beaucoup de recul et de simplement décrire les différents modules du code. Pour finir, une partie de la documentation sera dédiée à la maintenance de l'outil : comment modifier les fichiers lorsqu'un type de traitement est modifié ou comment le créer lorsqu'un nouveau type apparaît.

Cette documentation pourra être destinée aux personnes qui découvrent l'outil pour la première fois, qui utilisent régulièrement l'outil (elle servira de manuel), ou simplement qui souhaiteraient plus de détails sur le fonctionnement de l'application. Voici un extrait de son contenu : le tableau des options de la ligne de commande (la documentation complète est aussi jointe en annexe (*Cf. Annexe D - Documentation*))

-h + -fr/-en	Affiche l'aide
-f	Obligatoire pour toutes les options ci-dessous.
-t	Force le type de traitement. Doit être suivi du type de traitement.
-v	Affiche les DC seulement.
-d	Affiche le détail du DC (ne peut être utilisé qu'avec l'option -v).
-i	Permet d'ajouter l'identifiant d'audit (ne peut être utilisé qu'avec l'option -v).
-e	Extraction du fichier spécifié.
-s	Analyse un fichier sans entête platine (le fichier contient seulement un DC). On doit préciser le type de traitement avec -t.
-stat	Affiche le nombre de CRA dans le fichier.

FIGURE 9 – Liste des options de commande.

Voici deux exemples de ligne de commande. Dans la première, le type de traitement n'est pas précisé, il le trouve donc grâce au numéro d'ECC ; et dans la 2^{ième} c'est un fichier sans entête qui n'est pas forcément un fichier de collecte dont on précise le type de traitement manuellement.

```
coll_dump_fichier -f /deci/pood5325/collecte/fichiersCollecte/coll_0005_120214_09_06_001 -v -d
coll_dump_fichier -f /deci/pood5325/collecte/fichiersCollecte/CACIVCMFRAF103275 -t TTcrae -v -d -s
```

FIGURE 10 – Exemples de ligne de commande.

Voici le résultat que l'on obtient sur ce type de commande :

FIGURE 4.17 – Extrait de la documentation utilisateur.

Un axe possible d'évolution que je développerai s'il me reste du temps après la date de rendu de ce rapport, serait de rédiger cette documentation logiciel sous la forme d'un wiki qui serait associé au projet sous la plateforme OrangeForge. Cela aurait pour but d'augmenter et de faciliter sa disponibilité de façon à ce que tout le monde puisse y avoir accès depuis le dépôt de l'outil.

4.2.2 Documentation développeur

La documentation développeur est pour moi plus compliquée à rédiger. En effet, sous forme de document, elle risquerait de ressembler à un vaste catalogue, une liste illisible dans laquelle il serait difficile de trouver les informations, souvent précises, que l'on recherche. J'ai donc décidé d'utiliser un outil qui avait déjà été utilisé par le précédent stagiaire : **Doxygen**. Mon travail sera donc de compléter cette documentation, de l'enrichir (graphiques) et de la

rendre plus facilement disponible. Pour cela, il faut installer le plug-in Doxygen d'eclipse (Cf. *Création d'un projet eclipse*).

Doxygen est un outil qui permet de générer automatiquement une documentation développeur sous forme d'une page html. Elle se présente donc comme une forme de site "web" statique mis à part qu'il est hébergé localement. Pour cela, il faut commenter le code dans une syntaxe particulière et propre à Doxygen. Doxygen récupère ensuite les données sur toutes les variables, fonctions et classes afin de créer le "site web" dans lequel on peut retrouver facilement toutes ces informations. Il génère de plus des diagrammes d'héritage et de dépendance des classes. Il nous reste ensuite plus qu'à taper dans la barre de recherche le nom de l'instance sur laquelle on veut obtenir des informations. Voici un aperçu du résultat final :

The screenshot shows a web browser displaying the Doxygen-generated documentation for a project named 'Coll Dump Fichier'. The page title is 'Documentation1.1'. The navigation bar includes 'Main Page', 'Classes', and 'Files'. The 'Classes' tab is active, showing a sidebar with a tree view of the project structure. The main content area is titled 'Lecteur Class Reference' and includes the following sections:

- #include <Lecteur.h>**
- Inheritance diagram for Lecteur:** A diagram showing 'Lecteur' as the base class, with five derived classes: 'LecteurBER', 'LecteurCMM', 'LecteurCSV', 'LecteurFIXE', and 'LecteurTLV'.
- Public Member Functions:**
 - `Lecteur ()`
 - `Lecteur (std::string nom, std::vector< std::string > donnees_decodage)`
 - `std::string RenvoieNomLecteur ()`
 - `std::vector< std::string > RenvoieDonneesDecodage ()`
 - `std::string RenvoieFichierDecodage ()`
 - `void Afficher (std::ostream &out) const`
 - `virtual void Lecture (Commande comm, EntetePlatine entete, FILE *fichier, int num_entete)`
 - `virtual ~Lecteur ()`
- Private Attributes:**
 - `std::string p_nom`
 - `std::vector< std::string > p_donnees_decodage`
 - `std::string p_fichier_decodage`
- Constructor & Destructor Documentation:**
 - `Lecteur::Lecteur ()`

FIGURE 4.18 – Aperçu de la documentation générée par Doxygen.

Le dossier contenant toutes les ressources html est inclus dans le projet, ainsi lorsqu'un développeur récupère le code source il récupère obligatoirement cet outil avec le projet. Cette documentation est essentielle et efficace pour toute personne qui ne connaît pas le code et qui souhaite apporter une modification.

Enseignements

5.1 Compétences acquises

En plus du travail que j'ai effectué, ce stage m'a permis d'acquérir de nombreuses compétences, et même bien plus que ce à quoi je m'attendais. J'ai été amené à utiliser de nouveaux outils quotidiennement et à voir des méthodes de travail très efficaces que je n'utilisai pas jusqu'ici. J'ai découvert avec beaucoup de plaisir chacun de ces outils qui m'ont énormément apporté en culture générale. J'ai pu développer un tout nouveau point de vue grâce à des langages que je ne connaissais pas, des façon de gérer les projets ou même grâce à des nouveaux éditeurs de documents.

En effet, j'ai eu beaucoup de cours en informatique, bien que ce ne soit pas non plus la spécialisation de ma formation, qui a une vocation généraliste et relativement théorique ; et pourtant, j'ai réalisé pendant ce stage qu'il y avait encore de nombreuses choses très utiles que je ne connaissais pas de ce domaine. L'informatique mais aussi l'organisation des projets, la production de documents propres, réguliers et rigoureux étant des domaines qui me passionnent, j'ai été réellement heureux de découvrir tous ces outils qui ont amené beaucoup de nouveautés dans ma façon de travailler.

Je vais donc détailler avec un peu plus de précision les différentes "compétences" que ce stage m'a permis d'acquérir.

5.1.1 Langage C++

Pour commencer, le langage C++ est le langage dans lequel je suis le plus à l'aise. En effet, c'est ce dernier que j'ai utilisé dans presque l'intégralité de ma formation. Je pensais

avoir presque vu toutes les bases de ce langage. Cependant, même ici ce stage m'a apporté des connaissances essentielles. Par exemple, je n'avais jamais vu l'aspect interactions avec des fichiers externes qui ouvre énormément les possibilités du langage. En effet, un programme qui n'utilise pas de fichier peut-être lancé une fois mais on ne garde aucune trace de l'exécution, il n'y a pas de sauvegarde possible, pas de configuration, pas d'utilisateurs etc. Cela relève plus de l'utilitaire que du logiciel. Et l'interaction avec les fichiers étant un point essentiel du fonctionnement de l'outil sur lequel j'ai travaillé, j'ai été forcé de correctement comprendre la lecture et l'écriture, la déclaration des différents flux etc. Je pense maintenant être capable d'utiliser des fichiers externes pour des développements personnels.

De plus, je n'avais encore jamais vu une réelle application de la programmation orientée objet, ce qui est une part importante du langage. En effet, j'avais vu le fonctionnement de la POO⁷, ainsi que quelques exemples ; mais je n'avais pas vu toutes les possibilités que cela induit. Ce logiciel était un très bon exemple qui montrait à la fois la rigueur nécessaire pour réaliser un projet de cette envergure, et ce fut à la fois un très bon rappel reprenant toutes les connaissances que j'avais acquises jusqu'ici.

5.1.2 Gestion de configuration

J'ai aussi découvert une toute nouvelle façon de gérer un code ou même n'importe quel type de projet : la gestion de configuration. Je ne réexpliquerai pas le fonctionnement de cette dernière en détail (*Cf. Gestion de configuration*), mais elle permet une réelle collaboration entre plusieurs développeurs. Mais surtout, elle permet une organisation très propre d'un projet, avec un historique de ces évolutions, et avec une unique dernière versions sans qu'il risque d'y avoir des copies sur tous les postes que l'on utilise.

C'est un outil qui est très utilisé en entreprise et qui est, à mon sens, essentiel de connaître avant d'entrer dans le monde du travail.

7. Programmation Orientée Objet

5.1.3 Nouveaux logiciels

J'ai aussi été amené à travailler dans un environnement que je ne connaissais pas, ce qui implique des logiciels que je n'avais pas forcément utilisés. Cela demande certes un temps d'adaptation, mais amène aussi des nouvelles possibilités qui n'étaient pas forcément disponibles sur les anciens logiciels.

Pour commencer, au cours de tous les développements que j'ai réalisés, j'ai découvert un nouvel IDE : *eclipse*. C'est un IDE qui n'a pas du tout la même "mentalité" que Visual Studio 2013 que j'utilisais jusqu'ici. En effet, Visual contient toutes les fonctionnalités possibles, les compilateurs de tous les langages et est très lourd à installer et à lancer, tant dit qu'*eclipse* est très léger de base et on ajoute ensuite seulement les outils nous intéressant.

Ensuite, lorsque l'environnement de développement ne suffisait, pas j'ai eu à utiliser des outils permettant de déboguer des codes complexes. En effet, parfois le code se compile car il n'y a pas d'erreur de syntaxe, mais le fonctionnement n'est pas le bon. Il faut alors utiliser un déboguer pas à pas tel que le déboguer *gdb* qui m'a beaucoup servi, ou encore *Valgrind* qui permet de détecter les fuites mémoires.

5.1.4 Langage html5 et css3

Le domaine dans lequel je pense avoir le plus appris est le web, et plus précisément les langages html5 et css3. En effet, je n'avais auparavant jamais étudié du développement web ni même le langage html tout simplement. A mon arrivée à Orange, j'ai très vite été forcé de constater à quel point ce langage peut s'avérer utile. Il est présent presque partout et le fait d'avoir simplement quelques bases dans ces langages permet très vite de modifier ou de produire quelques documents. Une personne du service a proposé au reste de l'équipe de réaliser, pendant mon stage, une petite initiation au langage html car il maîtrisait bien ce langage et que peu de personnes de l'équipe ne l'utilisent. Je me suis donc rendu à cette présentation, ce qui fût mon premier contact avec le langage. Ensuite, j'ai vite eu besoin de ce dernier dans plusieurs de mes travaux. Pour commencer, la documentation technique était générée en html, si je souhaitais modifier quelques petits détails, ajouter une image

ou un schéma, il fallait modifier le code html. Ces modifications n'étaient pas compliquées mais demandaient tout de même quelques bases. De plus, un autre alternant travaillant au comptage m'a présenté un outil très pratique permettant de générer une présentation claire et uniforme à partir d'un code en html. Il m'a montré comment me servir de cet outil ce qui m'a permis de pratiquer et de mieux comprendre ce langage qui était nouveau pour moi. J'ai donc eu un peu de théorie avec une illustration très concrète des différentes utilisations possibles de ces langages.

5.1.5 Création des documents de l'historique du stage

Pour finir, un point très important de ce stage a été de produire un historique clair, précis, rigoureux qui trace et présente mes travaux. Pour cela j'ai utilisé de nombreux nouveaux outils, principalement basés sur la génération du document final. Ainsi, on décrit le contenu du document grâce à une syntaxe particulière, et le logiciel produit la mise en page et l'organisation du document final. Ceci permet de se concentrer sur le fond du document, grader une vue globale de l'organisation sans se préoccuper de la mise en page, de cette façon on est certain de garder une rigueur tout au long du document ; et même pour les futurs documents qui garderont la même forme.

En premier pour ce rapport ainsi que la documentation utilisateur, pour lesquels j'ai utilisé le langage $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ qui permet de créer des sections et sous sections, puis le logiciel créé une table des matières, numérote les figures, les titre et garde une mise en page cohérente. Malgré un certain temps d'apprentissage, je pense que ce langage m'a permis de gagner un temps considérable sur la rédaction de mon rapport et de la documentation utilisateur. De plus, il permet de créer des documents uniformes qui gardent toujours la même mise en page et la même structure.

Ensuite, j'ai pu utiliser deux outils utilisant le langage HTML dont j'ai parlé ci dessus. Tout d'abord l'outil Doxygen que j'ai présenté précédemment (*Cf. Documentation développeur*). Puis la rédaction de mes présentations (notamment pour les équipes d'Orange), j'ai utilisé l'outil mentionné précédemment : reveal.js. Pour ce dernier, il faut coder un squelette en HTML qui utilise des niveaux de titres, des listes numérotées ou non, des images et toutes

les autres balises classiques du langage. On place ce code dans un squelette fourni avec la bibliothèque. Puis le moteur de l'outil génère une présentation interactive, claire, originale et dont le format est adapté au support sur lequel elle est lue. Il suffit de posséder un navigateur web. Cette bibliothèque permet cette fois-ci de gagner énormément de temps dès la première présentation : le langage est simple à prendre en main, les thèmes de base sont largement suffisants, mais si on en a l'utilité on peut créer son thème qui sera à chaque fois réutilisable ; et en suite, si on a le contenu de notre présentation, il suffit d'une dizaine de minutes pour l'écrire et la présentation est déjà prête.

Je pense donc continuer par la suite à utiliser ces outils dans le cadre de mes études car je suis très satisfait de la mise en page finale de ces documents. De plus, ces outils apportent une cohérence et une continuité dans la création des différents documents qui m'accompagneront au cours de mes études.

5.2 Apports personnels

Finalement, ce stage m'a certes apporté beaucoup de nouvelles connaissances, mais ce fût avant tout une grande expérience personnelle, tout particulièrement sur le plan social. En effet, c'était pour moi la première fois que j'ai réellement été intégré au sein d'une entreprise. J'ai eu la chance de travailler en équipe, de participer aux réunions tant comme spectateur qu'en tant qu'acteur. J'ai pu présenter le déroulement de mon stage ainsi que mes travaux lors d'une présentation ouverte à tout le service (plus d'une trentaine de personnes se sont présentées), et j'ai réellement apprécié l'intérêt qu'a porté chacun des employés sur mes travaux. Cela a été pour moi de très nombreux exercices de communication : apprendre à trouver le besoin, demander de l'aide mais aussi être force de proposition et convaincre des personnes bien plus expérimentées de nos idées. De plus, la présentation de mes travaux a donné un réel sens à mon travail car, si on prend l'exemple des documentations, c'est une chose de les réaliser, mais si personne ne connaît leur existence elles perdent tout leur intérêt.

J'ai eu des collègues de travail très accueillants et très agréables à qui j'ai pu demander de

l'aide à n'importe quel moment et qui ont toujours été disponibles et à l'écoute. Ce stage fût donc un très bon entraînement de communication auquel on n'a pas forcément l'occasion de se confronter pendant les cours et qui est un point essentiel de tout projet. J'ai pu présenter mes travaux devant un public très réceptif, j'ai dû expliquer mes idées de développement, cibler les besoins des personnes de l'équipe, et tout ceci m'a forcé à aller communiquer avec tout le monde afin de devenir force de proposition d'amener une part d'amélioration au reste du service.

Je garde en très bon souvenir cette expérience pendant laquelle j'ai beaucoup appris des autres, et j'espère avoir fourni un travail mais aussi des connaissances.

Conclusion

Pour conclure, je garderai en mémoire ce stage qui restera une des expériences les plus enrichissantes de ma formation. Chacune des missions qui m'ont été confiées m'ont permis de participer concrètement à la vie d'une entreprise, de découvrir en détail le métier de développeur, ses contraintes et ses avantages. Tout particulièrement, il m'a permis d'effacer des craintes que je conservais par rapport à ce métier : la peur de l'ennui ou de la redondance. Cette expérience m'a montré tout le contraire.

La réalisation des missions qui m'étaient proposées m'a permis de confronter les connaissances acquises au cours de ma formation à celles exigées dans le monde du travail. Elles m'ont de plus permis d'approfondir ces connaissances si ce n'est d'en acquérir de nouvelles. La rédaction des documents d'historique de ce stage (rapport, soutenance, journal de bord etc.) m'a demandé une méthode de travail rigoureuse et organisée. En effet, il m'aurait été difficile de fournir cette quantité de travail le temps imparti tout en conservant un historique sans un travail méthodique et planifié. La mise en place de ces méthodes de travail, qui me resteront désormais comme culture générale, a donc aussi été un point important de ce stage.

Pour finir, je suis fier des travaux que j'ai réalisés et qui m'ont permis, je pense, d'apporter quelque chose de concret, bien que modeste, à une entreprise de la taille de Orange. Plus que ces travaux, ce stage m'a permis d'apprendre beaucoup, notamment sur le plan personnel : confronté à de nombreux exercices de communication j'ai du faire l'effort d'être force de proposition dans un cadre parfois contraignant, et j'ai eu à travailler avec une équipe à laquelle j'ai du montrer ma motivation quotidienne.

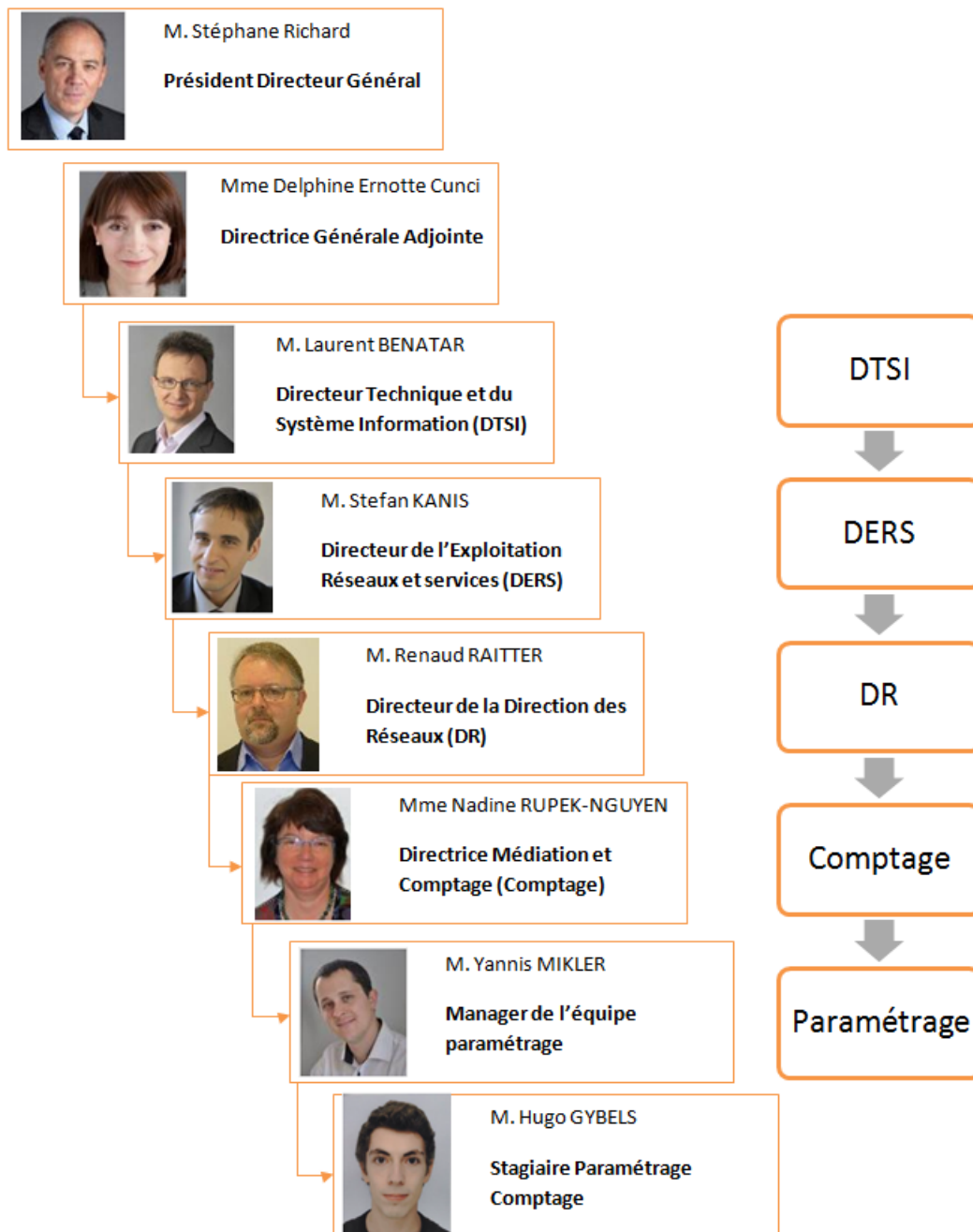
Fort de cette expérience, c'est motivé et sans hésitation que j'intégrerai l'école d'ingénieur

de Telecom Saint-Étienne dans le but de m'orienter par la suite vers de secteur informatique.

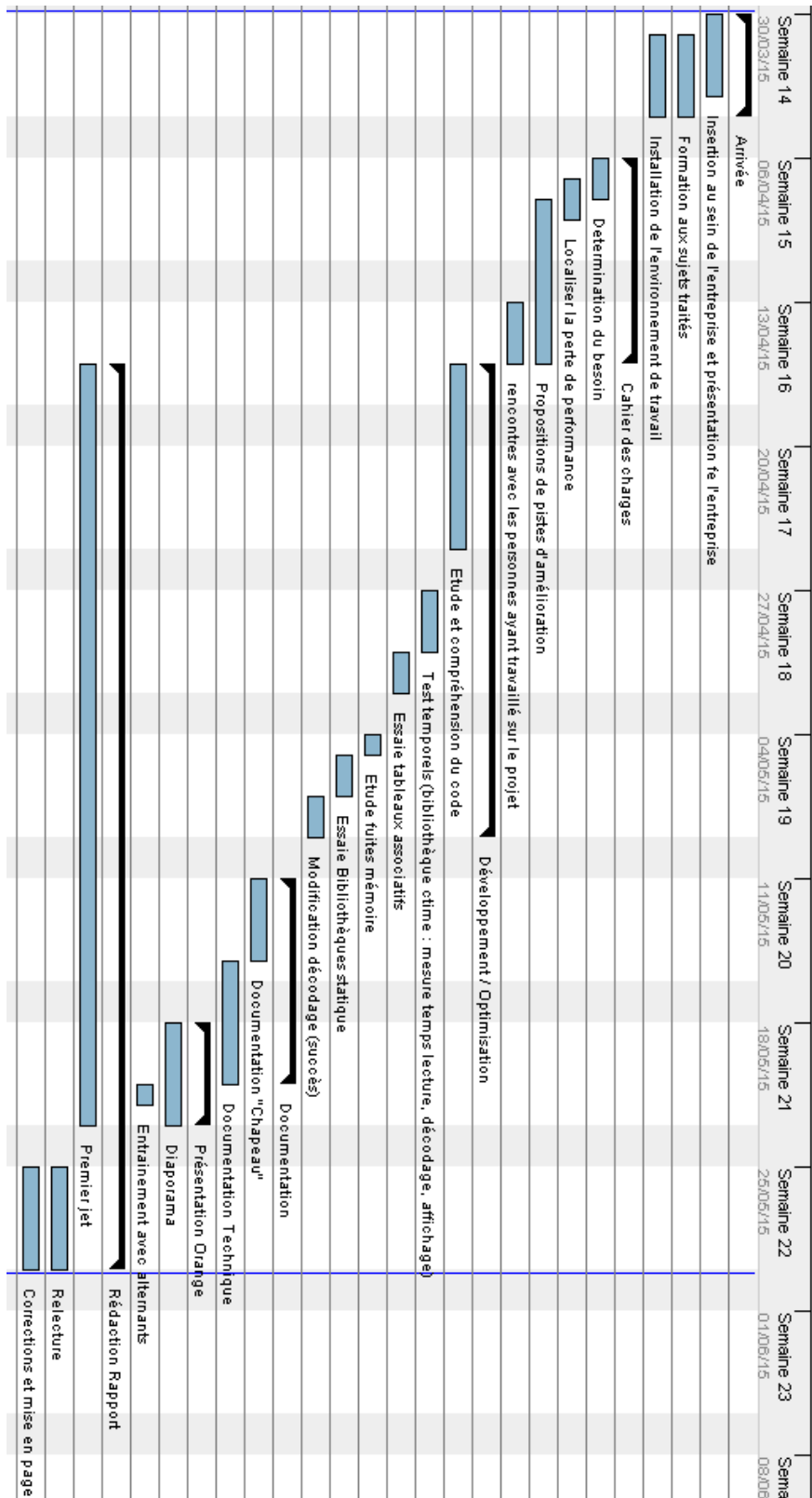
Table des figures

2.1	Chiffres Significatifs Orange 2014.	10
2.2	Fonctionnement de Platine	13
2.3	Emplacement et rôle de Coll Dump Fichier.	14
4.4	Graphique de révision svn.	20
4.5	Fenêtre d'accueil du logiciel Eclipse.	21
4.6	Configuration du logiciel Putty.	22
4.7	Fonctionnement du code de Coll Dump Fichier.	25
4.8	Exemple d'utilisation classique Coll Dump Fichier.	26
4.9	Exemple d'utilisation de la commande time.	27
4.10	Exemple d'utilisation de la bibliothèque <i>ctime</i>	28
4.11	Résultats temps d'exécution.	29
4.12	Structure d'un fichier csv.	31
4.13	Code lecture original redondant.	32
4.14	code lecture final.	33
4.15	Comparaison avec la version originale - Commande time.	34
4.16	Comparaison avec la version originale - Script compare.	34
4.17	Extrait de la documentation utilisateur.	36
4.18	Aperçu de la documentation générée par Doxygen.	37

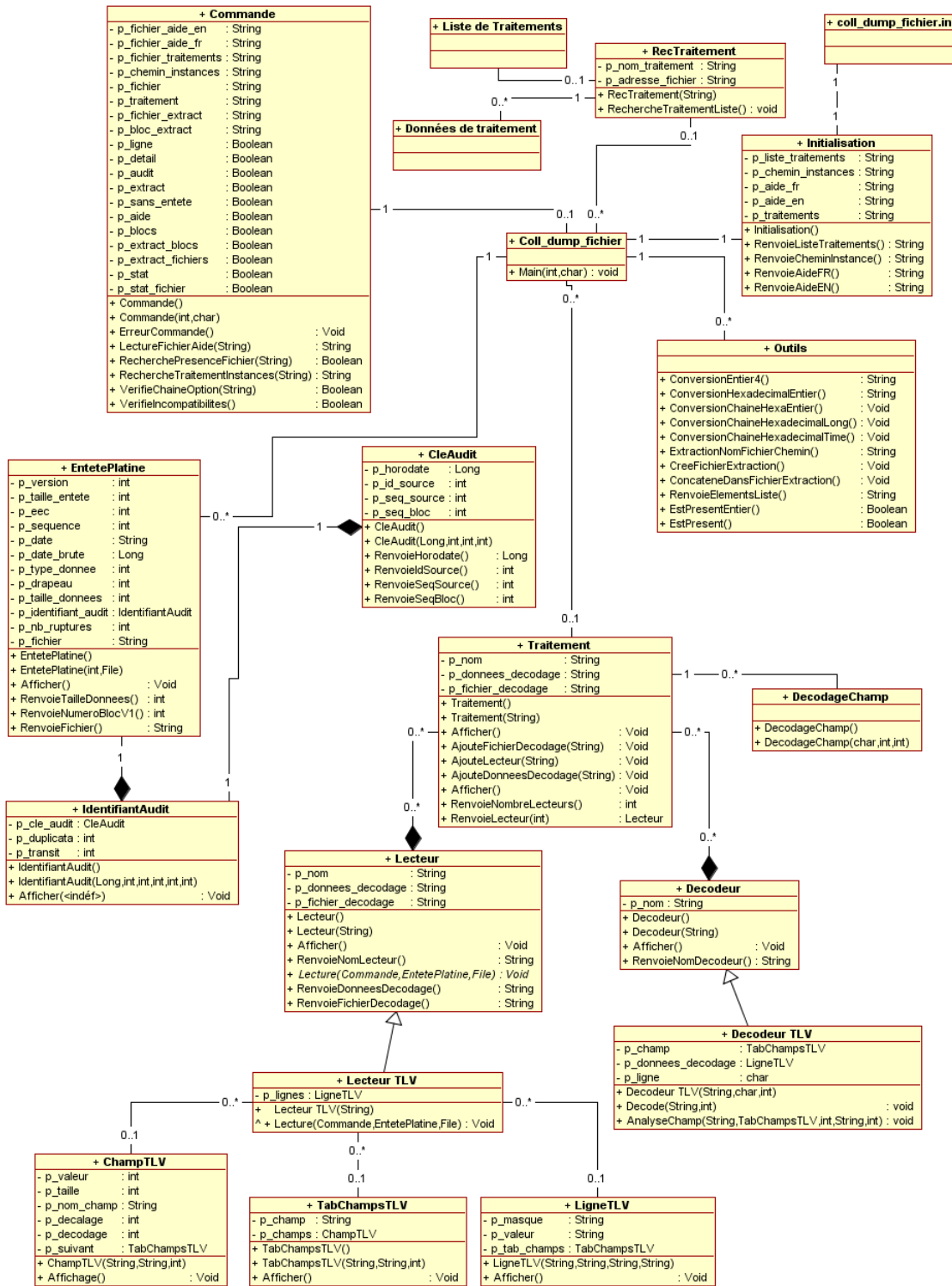
Annexe A - Organigramme



Annexe B - Diagramme de Gantt



Annexe C - Diagramme UML



Annexe D - Documentation

Documentation Coll Dump Fichier

GYBELS Hugo

28 mai 2015

Table des matières

1	Présentation générale	2
2	Installation	3
2.1	Coll_dump_fichier.ini (Obligatoire)	3
2.2	Coll_dump_traitements.csv (Obligatoire)	3
2.3	Fichiers de décodage : traitement.csv (Obligatoires)	4
2.4	Dossier Instances (Facultatif)	5
2.5	Les fichiers d'aide (Facultatifs)	6
3	Utilisation	6
3.1	Environnement de Travail	6
3.2	Commandes	7
4	Fonctionnement du code	9
4.1	Fonctionnement général	9
4.2	Lecture du fichier csv	10
4.2.1	Instanciación de la clase Lecteur	10
4.2.2	Structures de stockage des données	11
4.3	Decodage du fichier de collecte	13
5	Maintenance	14

1 Présentation générale

L'outil Coll Dump Fichier est un outil appartenant à l'entreprise Orange utilisé dans le pôle médiation. Il est utilisé comme aide à l'exploitation de Platine. Platine est une application collectant des comptes-rendus d'appels (CRA), qu'elle va traiter grâce à différents modules (collecte, traitement, distribution), puis de les envoyer vers les Systèmes d'Informations (SI) qui traiteront ces données (facturation, enquêtes juridiques, etc.). Platine prépare donc le traitement des données envoyées par les EEC¹ aux ECC² pour faciliter ensuite le traitement des données ; principalement la facturation.

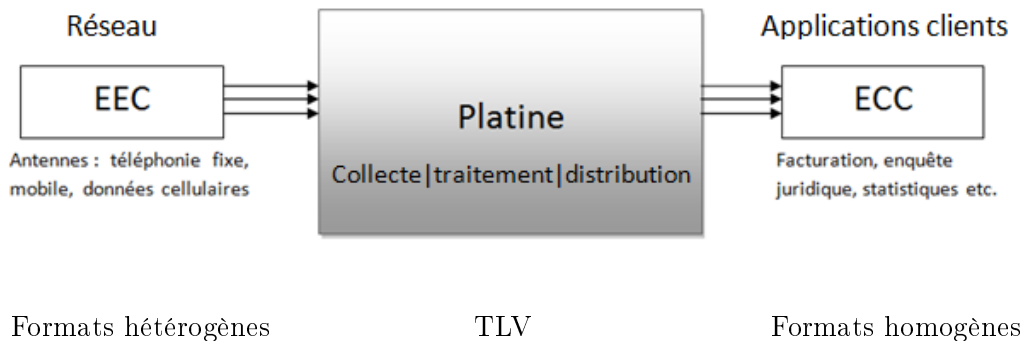


FIGURE 1 – Fonctionnement de Platine

L'application Platine traite des milliards de CRAs tous les jours. Pour gérer un tel flux, tous les CRAs sont convertis dans le format TLV (Type-Length-Value) avant d'être traités par Platine, et seul les champs importants du CRA sont conservés de façon à pouvoir traiter par la suite ces CRAs le plus rapidement possible.

Cependant, certaines fois, on retrouve des CRAs erronés. Lorsque cela arrive, il faut déterminer si l'erreur vient de Platine ou de l'émetteur. Pour cela, il nous faut donc décoder et afficher le message tel que Platine l'a reçu, avant même que le CRA ne soit converti au format TLV. C'est ici qu'intervient l'outil Coll Dump Fichier : il permet de décoder et d'afficher un fichier de collecte pour chaque type de traitement. **Coll Dump Fichier permet ainsi d'afficher le contenu d'un CRA reçu par platine avant qu'il ne soit converti en format TLV ; cela permet d'observer le message exacte ayant été envoyé par l'EEC.** Ceci permet de localiser si le défaut vient en amont de platine ou s'il est du à la collecte ou aux convertisseurs de Platine. Voici ci-dessous un schéma résumant l'emplacement de l'outil Coll Dump Fichier par rapport à l'application Platine.

1. Equipement Emetteur de Comptes-rendus d'appels (émetteur)
2. Equipement Consomateur de Comptes-rendus d'appels (récepteur)

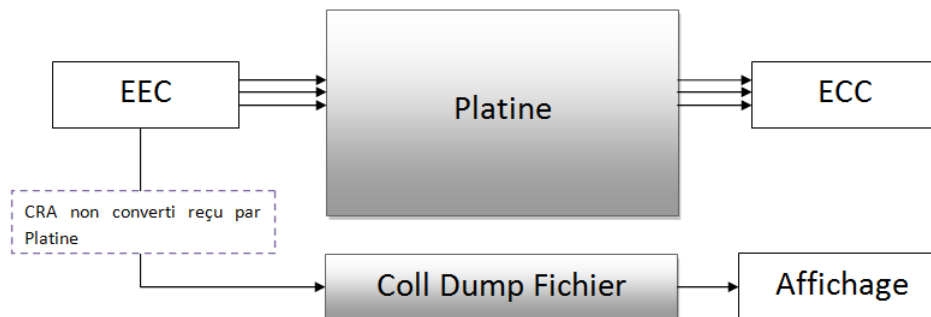


FIGURE 2 – Emplacement de Coll Dump Fichier par rapport à Platine

2 Installation

L'outil Coll Dump Fichier a obligatoirement besoin de trois fichiers pour fonctionner :

2.1 Coll_dump_fichier.ini (Obligatoire)

Le fichier `Coll_dump_fichier.ini` contient les ressources nécessaires au bon fonctionnement de l'outil. C'est-à-dire l'emplacement des fichiers `Coll_dump_traitements.csv`, `instances.csv`, et de l'aide en Français et en anglais. Ci-dessous un exemple du fichier `Coll_dump_fichier.ini` :

```

Liste des traitements :/deci/pood5325/cdf_h/Debug/coll_dump_traitements.csv
Chemin instances :/deci/pood5325/EEC/instances/
Fichier aide Francais :/deci/pood5325/cdf_h/Debug/help_FR/coll_dump_fichier_aide_fr.txt
Fichier aide anglais :/deci/pood5325/cdf_h/Debug/help_EN/coll_dump_fichier_aide_en.txt

```

FIGURE 3 – Coll_dump_fichier.ini

2.2 Coll_dump_traitements.csv (Obligatoire)

Le fichier `Coll_dump_traitements.csv` doit se situer à l'emplacement indiqué dans le fichier `Coll_dump_fichier.ini`. Il effectue la correspondance entre les différents types de traitements (TTnss9, TTprims, etc.) et la norme qu'ils utilisent (BER, CMM, FIXE, TLV, etc.). Il indique également l'emplacement du fichier contenant le détail des champs du type de traitement (les fichiers de décodage). Il doit être mis à jour pour chaque nouveau type de traitement. Voici un extrait du fichier :

Nom de Traitement	Fichier de decodage	Nombre de lecteurs
trait1	/deci/pood5325/Outil/coll_dump_traitements/trait1.csv	1 LecteurTLV
TT02F_HP	/deci/pood5325/Outil/coll_dump_traitements/TT02F_HP.csv	1 LecteurFIXE
TTbgw_gsn	/deci/pood5325/Outil/coll_dump_traitements/TTbgw_gsn.csv	1 LecteurBER
TTbgw_gsn	/deci/pood5325/Outil/coll_dump_traitements/TTbgw_gsn.csv	1 LecteurBER
TTbgw_gsnN	/deci/pood5325/Outil/coll_dump_traitements/TTbgw_gsn.csv	1 LecteurBERN
TTcb	/deci/pood5325/Outil/coll_dump_traitements/TTcb.csv	1 LecteurTLV
TTcitR4	/deci/pood5325/Outil/coll_dump_traitements/TTcitR4.csv	1 LecteurTLV
Ttcræe	/deci/pood5325/Outil/coll_dump_traitements/Ttcræe.csv	1 LecteurBER
TTgourou	/deci/pood5325/Outil/coll_dump_traitements/TTgourou.csv	1 LecteurTLV
TTincc	/deci/pood5325/Outil/coll_dump_traitements/TTincc.csv	1 LecteurCMM
TTmib	/deci/pood5325/Outil/coll_dump_traitements/TTmib.csv	1 LecteurCSV
TTmmm	/deci/pood5325/Outil/coll_dump_traitements/TTmmm.csv	1 LecteurTLV

FIGURE 4 – Coll_dump_traitements.csv

2.3 Fichiers de décodage : traitement.csv (Obligatoires)

Chaque type de traitement doit correspondre à un fichier de décodage de la forme **Traitement.csv**. Ce fichier doit se situer à l'emplacement indiqué dans la ligne correspondante du fichier Coll_dump_traitements.csv. Ce fichier contient l'intégralité des champs du DC ainsi que le numéro de la fonction de décodage du champ (exemple : date, ACSII, numéro de téléphone etc.) et le décalage de l'affichage. Pour finir il contient la forme du champ : si le champs est un primitif (une valeur à afficher) le champ contient la valeur "NULL", si c'est un constructed (un autre sous tableau) il contient le nom du sous tableau suivant.

Voici un tableau qui résume la structure d'un fichier csv :

Tag	Nom du champ	Décalage de l'affichage	Numéro de la fonction de décodage	Nature du champ suivant	0
-----	--------------	-------------------------	-----------------------------------	-------------------------	---

Les éléments de chaque ligne du fichier csv doivent être séparés par des ";". Ci-dessous un exemple de fichier csv (pour le TTnss9) avec quelques sous tableaux.

\$Id: TTnss9.c	2				
0 UMTSGSMPL	CallDataRecc	1			
1 Composite	CompositeEx	1			
CallDataRecc	15				
0 transit	1	0	TransitExt	0	
1 mSOriginatir	1	0	mSOriginatir	0	
2 roamingCallf	1	0	roamingCallf	0	
TransitExt	35				
4 callingPartyM	2	16	NULL	0	
5 calledPartyN	2	16	NULL	0	
51 bCSMTDPDat	2	0	bCSMTDPDat	0	
bCSMTDPDat	2				
0 ServiceKey	3	0	NULL	0	
1 gsmSCFAddr	3	0	NULL	0	
Tag	Nom Champs	Décodage	Décalage		Primitives

FIGURE 5 – Exemple de fichier csv (TTnss9.csv)

2.4 Dossier Instances (Facultatif)

Le dossier instances contient les descriptifs de toutes les EEC. Il permet de retrouver le type de traitement d'une EEC sans qu'on ait à le préciser dans la commande. Pour cela il faut juste qu'on retrouve le numéro de l'EEC dans le nom du fichier de collecte. Il doit se trouver à l'emplacement indiqué dans Coll_dump_fichier.ini.

Ce dossier est donc **obligatoire sans l'option -t** cf. *Partie 3.2 Commandes*.

```
[global]
AX_IDENT_EEC      = "WIFI"
AX_EEC_PROFIL     = "PUSH_CFT"
AX_EEC_PROTO      = "PUSH-CFT"
AX_EEC_CDATE      = 1257172699
AX_EEC_VALID      = 1
AX_LIBELLECOMPL_EEC = "Fusion Work WIFI"
AX_EEC_FREQTRANSFERT = 100
```

FIGURE 6 – fichier instances\1.ini : description de l'EEC 1

L'EEC 1 envoie donc des CRAs sous le type de traitement WIFI.

2.5 Les fichiers d'aide (Facultatifs)

Les fichiers d'aides en Français comme en Anglais sont des fichiers text (.txt). Il doivent avoir le nom et l'emplacement indiqué dans le fichier Coll_dump_fichier.ini. Leur rôle est de résumer l'utilisation de l'outil ainsi que des différentes options (détail plus tard dans la doc).

3 Utilisation

3.1 Environnement de Travail

Gestion de configuration

Le code source de Coll dump fichier est géré grâce à une gestion de versions sous eclipse. On utilise pour cela un dépôt sous Orange forge et un client SVN sous eclipse. Avant tout développement majeur, il est conseillé de créer une branche séparée du tronc que l'on mergera plus tard si les modifications sont à conserver. Pour cela on utilise un autre client directement intégré à Windows : *Tortoise* plus performant et plus convivial que le client eclipse. Par exemple, voici la branche SVN que je me suis créé pour mes travaux :

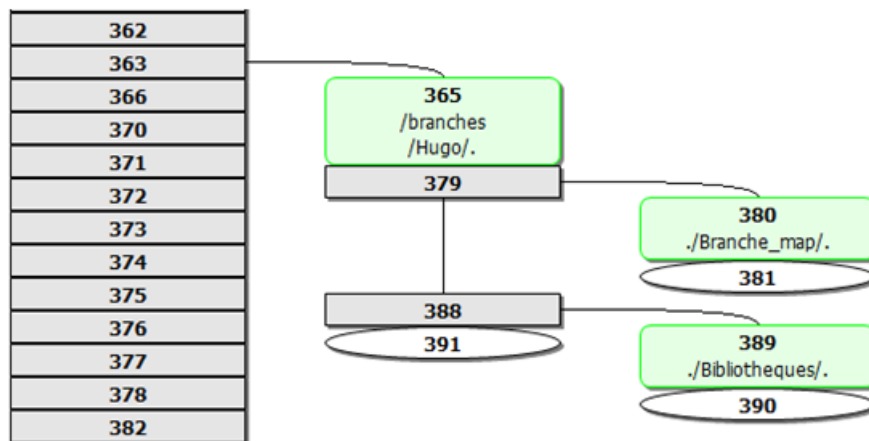


FIGURE 7 – Graphique de révision svn de CDF.

Il est conseillé de créer une copie de travail locale lorsque l'on effectue un checkout du projet.

Création d'un projet eclipse

Il faut ensuite créer un projet SVN sous eclipse. Pour cela, il faut créer un nouveau projet sur la machine distante grâce au menu *File -> New -> Other... -> SVN -> Checkout Projects from SVN* (le client SVN doit être installé). Le lien UML du dépôt est le suivant :

<https://www.forge.orange-labs.fr/svnroot/coll-dump-fichier>

On peut ensuite récupérer les fichiers sources. Il est ensuite nécessaire de compiler une fois sous eclipse de façon à générer les fichiers Makefile. ATTENTION : on va exécuter le programme sous linux, on utilise donc pas le compilateur par défaut mais bien **Cross GCC** disponible dans le compilateur MinGW32.

Exécution sur la machine distante

Une fois les Makefiles générés, on peut compiler sous la machine distante. Pour cela on ouvre un putty. Ci-dessous une capture d'écran de la configuration de la machine distante (DEV22).

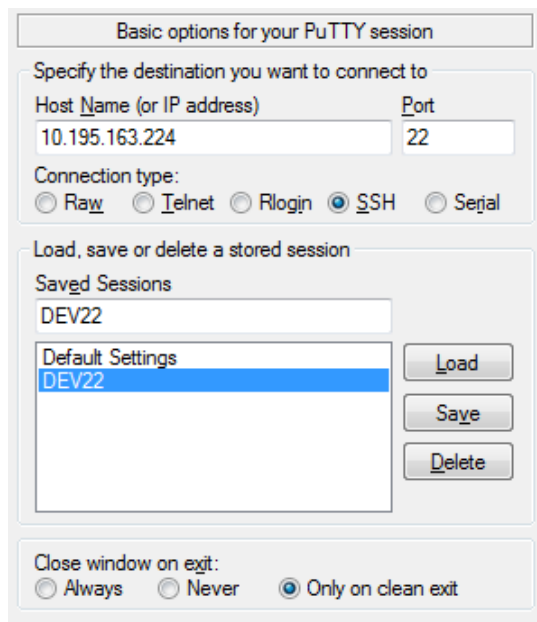


FIGURE 8 – Configuration de la machine DEV22.

Une fois le terminal ouvert, il ne sera plus nécessaire de compiler sous éclipse mis à part lors de la création de nouveaux fichiers. Pour compiler, on se place donc dans le répertoire Debug créer dans le dossier contenant notre projet. On exécute ensuite la commande **make clean** puis la commande **make** qui lancera la compilation du projet.

Le lancement de l'exécutable sera détaillé par la suite.

3.2 Commandes

Une fois installé, nous pouvons décoder différents types de fichiers de collecte. Dans tous les cas, la syntaxe est la même pour la ligne de commande :

```
coll_dump_fichier [-f] 'emplacement_du_fichier_de_collecte' [-options]
```

Il existe ensuite plusieurs options permettant de décoder différemment le fichier de collect. Voici ci-dessous un tableau résumant ces options.

-h + -fr/-en	Affiche l'aide
-f	Obligatoire pour toutes les options ci-dessous.
-t	Force le type de traitement. Doit être suivi du type de traitement.
-v	Affiche les DC seulement.
-d	Affiche le détail du DC (ne peut être utilisé qu'avec l'option -v).
-i	Permet d'ajouter l'identifiant d'audit (ne peut être utilisé qu'avec l'option -v).
-e	Extraction du fichier spécifié.
-s	Analyse un fichier sans entête platine (le fichier contient seulement un DC). On doit préciser le type de traitement avec -t.
-stat	Affiche le nombre de CRA dans le fichier.

FIGURE 9 – Liste des options de commande.

Voici deux exemples de ligne de commande. Dans la première, le type de traitement n'est pas précisé, il le trouve donc grâce au numéro d'ECC ; et dans la 2^{ième} c'est un fichier sans entête qui n'est pas forcément un fichier de collecte dont on précise le type de traitement manuellement.

```
coll_dump_fichier -f /deci/pood5325/collecte/fichiersCollecte/coll_0005_120214_09_06_001 -v -d
```



```
coll_dump_fichier -f /deci/pood5325/collecte/fichiersCollecte/CACIVCMFRAF103275 -t TTcrae -v -d -s
```

FIGURE 10 – Exemples de ligne de commande.

Voici le résultat que l'on obtient sur ce type de commande :

```

-----bloc: #0001-----
version      : 2
eec          : 30
numero       : 0
horodate     : 1396357292 (01/04/2014 13:01:32 GMT)
id audit     : 16777217_47336_58959_512_0 (date=16777217, idfSource=47336, seqSource=58959,
type        : 256
drapeaux     : 4
nbRuptures   : 31
taille       : 866013
fichier      : /data/COL1B/coll/data/travail/eec/30/.colI30/TTFI000_NGEMONAC_8226_f6fedd7f
-----;-----
a180a081c1a181be8c0300002789030e0401940f3333231393030203133412f30312f8a030a140e9f4302558c8
80101001633f49d01119f2f01059b0702f810313ae11b8707113366106957f59f4501069f550543e09c558c9f56
84064186805634f0bf4b0f8004000000658107113386096076f09f5b064253433033499f4a07113366106957f59
543e09c558c9907113386091059f0a515800102810400000065850a92001104338609607600a01aaf188102558c
0300002788030e0401930f3333231393030203133412f30312f89030a140e9f2d02558c9f3207113366106957f
34984064186805634f08507113366106957f58701000000
Composite(1)
  UMTSGSMPLMNCallDR(0)
  mSOriginatingC(1)
    chargeableDuration_MSO(12)=000027 -> 00:00:39
    dateForStartofCharge_MSO(9)=0e0401 -> 01/04/2014
    exchangeIdentity_MSO(20)=3333231393030203133412f30312f -> 3321900 13A/01/
    timeForStartofCharge_MSO(10)=0a140e -> 10:20:14

```

FIGURE 11 – Exemples de résultat de Coll Dump Fichier.

4 Fonctionnement du code

Cette partie a pour but d'expliquer le fonctionnement global du code : quelles sont les grandes étapes, comment les données sont stockées, décodées et affichées. Encore une fois, on gardera beaucoup de recul sur le fonctionnement du code, on n'explicitera pas les détails du fonctionnement.

4.1 Fonctionnement général

Nous avons déjà vu dans quel contexte l'outil Coll Dump Fichier est utilisé, mais avant de parler du code lui même, voici un résumé du fonctionnement de ce dernier.

Pour commencer, un rappel sur le principe du code : il doit ouvrir un fichier de collecte (qui contient des comptes-rendus d'appels, ou CRAs). Il va ensuite récupérer le type de traitement de l'EEC, autrement dit, le format ou la norme dans laquelle est codé le compte-rendu. Suite à cela, il va isoler chaque compte-rendu, qu'il va ensuite afficher dans la console avec le détail des champs qu'il contient. Cette étape est justement automatique dans Platine, qui elle transmet automatiquement les DC sans même les afficher.

Ces comptes-rendus sont donc composés d'une listes de champs correspondants à des valeurs. Par exemple, on peut retrouver le champ "Durée de l'appel" avec la valeur 60 secondes. Cependant, dans le fichier on aura pas le nom du champ "durée de l'appel", mais un numéro (ou tag) par exemple le tag 10 qui identifie ce champ. Il faut donc aller chercher la signification du numéro du champ dans le fichier csv qui contient la correspondance entre les tag et le nom des champs. Toujours dans notre exemple, l'outil va chercher, dans le fichier csv du type de traitement, la case avec le tag 10 qui correspondra au nom de champ "durée de l'appel" et le type de données sera "secondes". Voici un schéma qui résume ce fonctionnement :

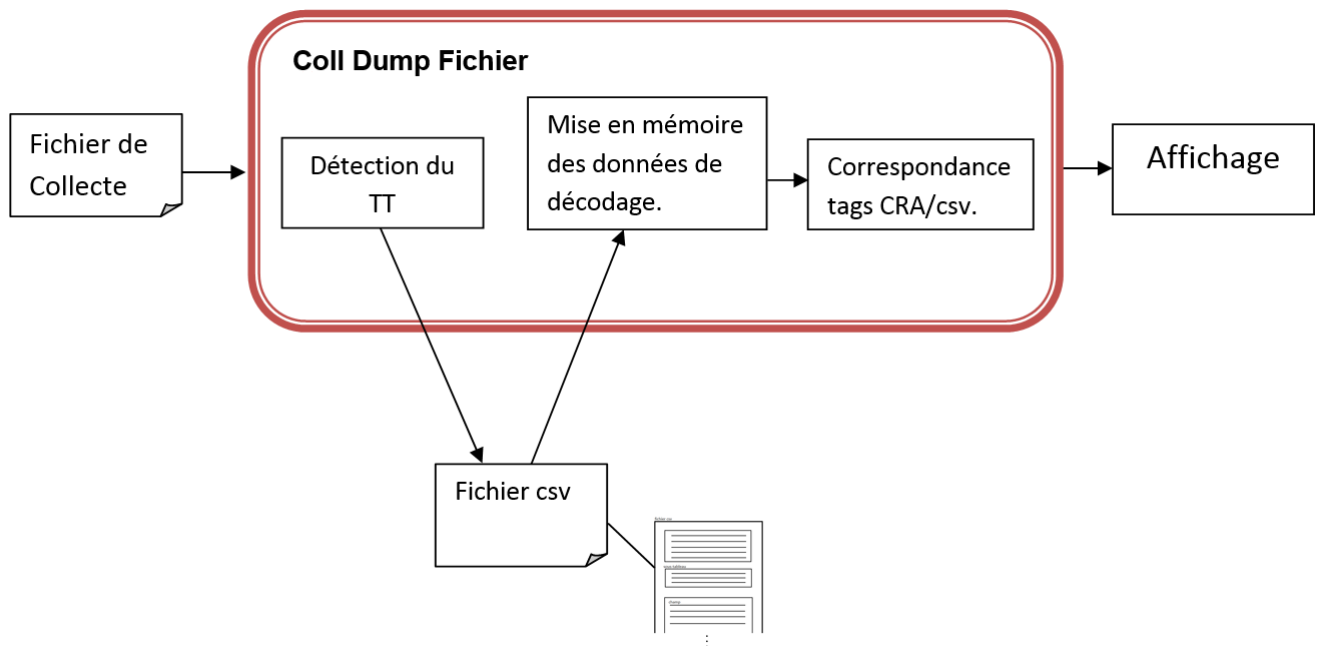


FIGURE 12 – Fonctionnement du code de Coll Dump Fichier.

Pour rendre cela plus concret, voici un exemple d'utilisation quotidienne de l'outil. La ligne de commande peut être combinée avec un ou plusieurs `grep`, ou encore sa sortie peut être redirigée vers un autre fichier (Cf. figure 13).

4.2 Lecture du fichier csv

4.2.1 Instanciation de la classe Lecteur

L'instanciation de la classe `Lecteur` se fait grâce aux classes `RecTraitement` et `Traitement` appelées dans le `main()`. En effet, la classe `RecTraitement` récupère le nom du ou des lecteur(s) dont nous allons avoir besoin pour le décodage. Il a deux façon de récupérer ce type de traitement : soit il est indiqué dans la ligne de commande à la suite de l'option `-t` ; soit le nom du fichier de collecte contient le numéro de l'EEC et il trouvera le type de traitement correspondant grâce au fichier `instances.csv`. La déclaration des lecteurs se fait dans une instance de type `traitement` à laquelle on ajoute tous les lecteurs dont nous avons besoin.

```

$ cdf_Hugo -f /deci/pood5325/collecte/fichiersCollecte/coll_0030_010414_15_07_003 -v -d | more
: Importation des donnees de traitement...
-----bloc: #0001-----
version      : 2
eec          : 30
numero       : 0
horodate     : 1396357292 (01/04/2014 13:01:32 GMT)
drapeaux     : 4
taille       : 866013
fichier      : /data/COL1B/coll/data/travail/eec/30/.colI30/TTFILE00_NGEMONAC_8226_f6fedd7f
-----
a180a081c1a181be8c0300002789030e0401940f33333231393030203133412f30312f8a030a140e9f4302558c860853394420
10f085339442037846042850802080101001633f49d01119f2f01059b0702f810313ae11b8707113366106957f59f4501069f5
a515800102810400000065850a92001104338609607600a01aaf188102558c8302000a980543e09c558c9907113386091059f0
300002788030e0401930f33333231393030203133412f30312f89030a140e9f2d02558c9f3207113366106957f59507484c523
064116890000f0960642534330334984064186805634f08507113366106957f58701000000
Composite(1)
  UMTSGSMPLMNCallDR(0)
  mSOriginatingC(1)
    chargeableDuration_MSO(12)=000027 -> 00:00:39
    dateForStartofCharge_MSO(9)=0e0401 -> 01/04/2014
    callingSubscriberIMSI_MSO(5)=02080101001633f4 -> 208010100061334f
    teleServiceCode_MSO(29)=11
    iNMarkingOfMS_MSO(47)=05
    firstCallingLocInfo_MSO(27)=02f810313ae11b -> 208f0113a31eb1
    calledPartyNumber_MSO(7)=113366106957f5 -> 1133660196755f
--More--

```

FIGURE 13 – Exemple d'utilisation classique de Coll Dump Fichier.

4.2.2 Structures de stockage des données

Le stockage des données contenues dans le fichier *.csv* se fait dans le constructeur de la classe fille du Lecteur appelé (par exemple dans LecteurBER). On prendra par la suite l'exemple du LecteurBER afin de mieux comprendre, mais la structure est la même pour les autres lecteurs. Les données du csv sont stockées dans plusieurs tableaux dynamiques (vector) appartenants à plusieurs classes. Le graphique ci-dessous résume l'appel de ces classes par ordre chronologique :

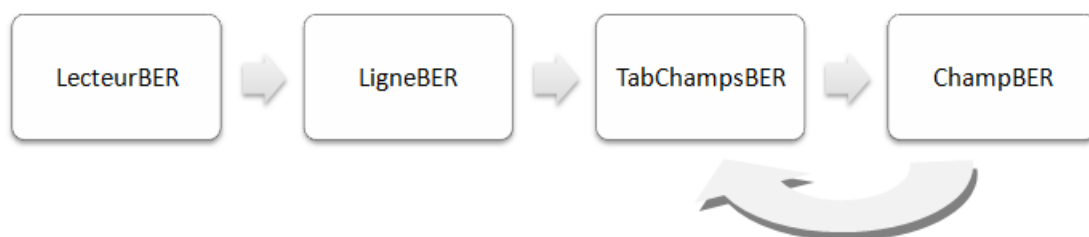


FIGURE 14 – Appel des classes de stockage du csv.

Chacune de ces classes contient un tableau dynamique correspondant à une partie du fichier csv, le tableau ci-dessous résume ce à quoi correspond chacune des classes :

LecteurBER	LecteurBER contiendra donc l'intégralité du fichier csv dans p_ligne .	p_ligne : Tableau de LigneBER.
LigneBER	Chaque LigneBER correspond à une "nature" du DC. Il y aura autant de LigneBER que de ligne de 4 colonnes au début du fichier csv.	p_tab_champs : Tableau de TabChampsBER.
TabChampsBER	Chaque champ de forme <i>constructed</i> correspond à un TabChampsBER. TabChampsBER correspond donc à un sous tableau du fichier csv.	p_champs : Tableau de ChampBER
ChampBER	ChampBER contient les données de chacune des cases du fichier csv. ChampBER correspond donc à une ligne du csv.	p_tag , p_flag , $p_decodage$, etc.

0	UMTS GSM PL CallDataRec	1		LigneBER	
1	Composite	CompositeEx	1		
CallDataRec	15				TabChampsBER
0	transit	1	0	TransitExt	
1	mSOriginatir	1	0	mSOriginatir	0
2	roamingCallI	1	0	roamingCallI	0
3	callForwardi	1	0	callForwardi	0
4	mSTerminati	1	0	mSTerminati	0
5	mSOriginatir	1	0	mSOriginatir	0
7	mSTerminati	1	0	mSTerminati	0
9	SSProcedure	1	0	SSProcedure	0
17	iSDNOrigina	1		LigneBER	0
18	iSDNCallForv	1			0
11	sSIInvocation	2	0	SSIEventMod	0
12	serviceSwitc	2	0	serviceSwitc	0
16	iNServiceDat	2	0	iNServiceDat	0
23	iSDNSSInvo	2	0	iSDNSSInvo	0
25	handOverEv	2	0	handOverEv	0
CompositeEx	1				
0	UMTS GSM PL	1	0	CallDataRec	0
CallDataRec	15				
0	transitC	1	0	TransitComp	0
1	mSOriginatir	1	0	mSOriginatir	0
2	roamingCallI	1	0	roamingCallI	0
3	callForwardi	1	0	callForwardi	0
4	mSTerminati	1	0	mSTerminati	0

FIGURE 15 – Structuration d'un fichier csv.

4.3 Decodage du fichier de collecte

Une fois toutes les données nécessaires au décodage des DC chargées en mémoire ainsi que le fichier de collecte, on peut commencer le décodage ainsi que l’affichage des différents DC. L’intégralité du décodage se déroule dans le fichier `Decodeur` (par exemple `DecodeurBER`). La fonction **Decode** analyse une ligne et appelle la fonction **AnalyseChamp** qui va soit afficher la valeur du champs si c’est une primitive, soit s’appeler à nouveau jusqu’à tomber sur un champs primitif. C’est une fonction récursive qui va nous permettre de parcourir l’intégralité du fichier csv stocké en mémoire en effectuant des comparaisons sur le tag des lignes du DC et du fichier csv. Pour finir, la fonction `AnalyseChamp` fait appel à une dernière fonction : la fonction **DecodeChamp** pour mettre en forme la valeur du champ avant de l’afficher ; par exemple, si c’est une date il la met au format `aaaa.mm.jj`, de même pour les IPv6, etc.

Remarque : On ne passe par cette partie seulement si on à choisi l'option -d correspondant au décodage.