Développement d'outils OSS & ENM

Amaury MAILLÉ
Licence Informatique – Université Claude Bernard

2 Mai 2017 – 28 Juillet 2017

Tuteur universitaire	Rapporteur universitaire	Maître de stage
Jacques BONNEVILLE	**Matthieu HEITZ**	**Emmanuel CLERC**

Remerciements

Je souhaite commencer par remercier les différents membres du RAN Support Ericsson pour leur accueil au sein de l'équipe. Leur ouverture et leur enthousiasme a permis de faire de ce stage une expérience particulièrement plaisante et enrichissante.

Je remercie également Mr Emmanuel CLERC pour m'avoir proposé un stage adapté à mes compétences et à mes préférences. Cela m'aura permis de m'améliorer, ainsi que de faire des découvertes intéressantes. Par ailleurs, je remercie Mr Philippe GABARD, responsable du RAN Support Ericsson, pour m'avoir accueilli dans son équipe et m'avoir fait confiance pour ce poste.

De plus, je remercie mon tuteur de stage, Mr Jacques BONNEVILLE pour sa bonne humeur, sa disponibilité et ses conseils.

Sommaire

Table des illustrations

En conclusion de la licence informatique, il nous est demandé de réaliser un stage en entreprise d'une durée minimale de trois mois, afin de découvrir le monde professionnel. J'ai choisi d'effectuer ce stage au sein de l'entreprise Orange. La mission qui m'a été confiée est la suivante : développer des outils back-end pour l'OSS Ericsson.

L'OSS Ericsson est un ensemble de serveurs, tournant sous Solaris, permettant d'administrer les différents composants du réseau radio d'Orange ; diverses personnes sont amenées à s'y connecter afin de résoudre des problèmes survenus sur le réseau radio. La gestion de ces utilisateurs passe par l'application web « User Management ». Celle-ci génère des fichiers qui sont ensuite récupérés par l'une des machines de l'OSS, l'OMINF, qui les traite via divers scripts pour gérer les demandes de créations de compte, de changements de mot de passe, ou d'autres opérations administratives, comme le verrouillage ou la suppression de comptes.

Mon premier objectif était de réorganiser les différents fichiers manipulés par les scripts de l'OMINF afin de leur faire respecter la hiérarchie standard[1]. Dans la version initiale, le script d'Orange, ainsi que les bibliothèques et fichiers de configuration, étaient placés au milieu de scripts administratifs fournis par Ericsson, ce qui n'était pas extrêmement pratique ni organisé.

Mon second objectif était de transposer l'application en langage Python afin de la rendre plus robuste, et lui apporter de nouvelles fonctionnalités sur lesquelles je reviendrai plus tard. La raison principale de ce changement est liée au fait que les scripts initiaux étaient écrits en langage Bash, ce qui rendait certaines opérations complexes, comme l'implémentation de tests unitaires, la vérification de la conformité des fichiers produits par l'application web, ou encore une meilleure gestion des erreurs.

Enfin, mon troisième objectif était de compléter le script Python précédemment produit pour que les opérations de création de compte, changement de mot de passe, verrouillage, déverrouillage et suppression de compte soient également fonctionnelles sur ENM, plateforme destinée à remplacer les OSS dans l'avenir. Ici, il s'agissait de communiquer avec une interface REST, et non plus de communiquer avec des scripts Bash fournis par Ericsson.

Cette mission a été amenée à évoluer au fil du stage. Ainsi, je vais par la suite contribuer à plusieurs systèmes back-end, par exemple l'implémentation d'une « message queue » dans un système de logs.

J'ai réalisé ce stage au côté de Mr Valentin ROUSSEL, également étudiant à l'université. Il s'est entre autres occupé de l'IHM de l'application web « User Management », ainsi que des scripts PHP chargés de la génération des fichiers exploités par l'OMINF (et, par la suite, par ENM). Nous avons été amenés à travailler ensemble à plusieurs reprises, nous aidant mutuellement lorsque nous rencontrions des difficultés.

Après avoir présenté plus en détail l'entreprise ainsi que l'unité dans laquelle j'ai été intégré, j'approfondirai les missions qui m'ont été confiées, pour terminer en revenant sur ce que le stage m'a apporté.

[1] A savoir : fichiers de configuration sous /etc/opt/ORANGE, fichiers temporaires sous /var/opt/ORANGE/<nom de l'appli>, fichiers exécutables sous /opt/ORANGE/bin et bibliothèques sous /opt/ORANGE/lib.

I. Orange & RAN Support Ericsson

1. Orange

Tout d'abord, présentons rapidement Orange. Il s'agit de l'un des principaux opérateurs de télécommunications dans le monde. Au 31 décembre 2016 l'entreprise compte 156 000 salariés, dont 96 000 en France, et un chiffre d'affaire s'élevant à 40,9 milliards d'euros. Implémenté dans 29 pays, le groupe Orange sert 263 millions de clients dans le monde, dont 202 millions de clients dans le réseau mobile, et 18 millions de clients dans le haut débit fixe.

Ses domaines d'activités se répartissent dans le développement la commercialisation de trois familles de services différentes : la téléphonie fixe, l'Internet bas débit, haut débit (via l'ADSL), et très haut débit (via la fibre optique), la téléphonie IP, la télévision numérique et les contenus multimédias ; cet ensemble forme la première famille de service, à savoir les services de communication résidentiels (SCR, un terme beaucoup trop compliqué pour désigner l'accès à l'Internet, au téléphone fixe et à la télévision). La seconde famille de service est l'ensemble des services de communication personnels (ou SCP), c'est-à-dire les communications mobiles (2G, 3G, 4G etc…). Enfin, la dernière famille de services est l'ensemble des services de communication d'entreprise (ou SCE) sous la marque Orange Business Services.

A ce jour, le réseau mobile d'Orange repose sur des équipements fournis par des constructeurs externes, comme par exemple Ericsson. Je reviendrai plus en détail sur ce point lorsque j'aborderai le rôle du RAN Support Ericsson, l'unité dans laquelle j'ai été intégré. Présentons maintenant l'historique de l'entreprise.

Le 1er Janvier 1988, la direction générale des télécommunications (DGT, fondée en 1941 sous le régime de Vichy en tant que direction des télécommunications, puis renommée DGT en 1946 par le GPRF) devient France Télécom, afin de répondre à une directive européenne (impossible de trouver précisément laquelle malheureusement) et de réorganiser le secteur des télécommunications. En 2000, France Telecom rachète Orange plc, auparavant Microtel, consortium fondé en 1990 pour développer une offre concurrente à celle de Vodafone, groupe de télécommunications britannique. Ce rachat devait permettre à France Télécom de rattraper son retard sur son développement à l'international. Le rachat d'Orange plc sera complété en 2003, et France Télécom commence progressivement à commercialiser ses produits sous le nom Orange ; ainsi, en 2006, Wanadoo deviendra Orange. A partir de 2011, France Télécom devient Groupe France Télécom – Orange, puis, en 2012, commercialise ses services de téléphonie fixe sous la marque Orange. Enfin, en 2013, le changement de nom est voté au cours d'une assemblée générale et le changement de nom devient effectif au 1er juillet 2013.

Maintenant, parlons de l'un des services composant Orange, le RAN Support Ericsson.

2. RAN Support Ericsson

J'ai été intégré au sein du RAN Support Ericsson 2G 3G 4G, équipe dirigée par Mr Philippe GABARD, et dont fait partie Mr Emmanuel CLERC, mon maître de stage. Le RAN Support est une sous-branche du RAM[2], lui-même dirigé par Mr Florent GROMENIL.

Contrairement à ce à quoi je m'attendais lorsque je suis arrivé, il ne s'agit pas d'une équipe de développeurs, bien que certains des membres soient auteurs d'une partie des applications que j'ai manipulées (par exemple, Mr CLERC a développé l'outil Python

[2] RAM : Réseau d'Accès Mobile

permettant d'effectuer des logs, ainsi que l'outil Python de packaging). Pour mieux expliquer le rôle du RAN Support, il me semble nécessaire d'élaborer en premier lieu ce que sont les OSS.

Les OSS, au nombre de huit, sont des ensembles de onze serveurs, répartis à travers la France, dénommés selon les régions qu'ils couvrent (Nord-Ouest, Sud-Ouest, Île-De-France etc…). Les OSS permettent, comme j'ai l'ai rapidement indiqué en introduction, d'administrer les différents composants du réseau radio Orange, par exemple les antennes (BTS[3], NodeB[4] ou eNodeB[5] dans le jargon technique). Les OSS permettent d'obtenir des informations en temps réel sur ces composants, comme leur connectivité, leur activité ; ils permettent également de les configurer et d'effectuer des opérations de maintenance. Par exemple lorsqu'une antenne tombe en panne (s'arrête de fonctionner, a un taux d'inactivité anormalement élevé etc…), l'OSS lève une alarme qui est transmise à l'équipe de supervision ; celle-ci fait quelques manipulations sur l'OSS pour tenter de résoudre le problème. Si elle n'y arrive pas, elle transmet un ticket Océane[6] à l'équipe Pilotage, qui envoie un ou plusieurs techniciens sur place pour tenter de résoudre le problème (leur spécialité est de changer les cartes). Si ceux-ci n'y parviennent pas, l'équipe Pilotage transmet le ticket Océane au RAN, qui prend la relève. Si le RAN ne parvient pas à résoudre le problème à son tour, le constructeur du matériel est directement contact. Ici, au RAN Support Ericsson, c'est le matériel fourni par Ericsson qui est administré.

Le RAN a donc pour objectif de résoudre les problèmes qui surviennent sur le réseau mobile. Néanmoins, cela ne représente pas la totalité de leur domaine d'expertise. Les membres du RAN sont également chargés de donner des méthodes et des fiches consignes aux équipes de déploiement (de matériel comme les NodesB / eNodesB) pour que le déploiement se passe bien. Ils sont également chargés de tester le matériel avant sa mise en place afin de s'assurer que tout est conforme.

[3] BTS : Base Transceiver Station, base d'émission du réseau 2G, contrôlé par un BSC

[4] NodeB : comme un BTS, mais pour le réseau 3G, contrôlé par un RNC

[5] eNodeB : evolved NodeB, comme un NodeB pour le réseau 4G ; se contrôle d'elle-même

[6] Un document électronique qui permet d'indiquer qu'il y a panne et quelle est la panne (et les mesures prises)

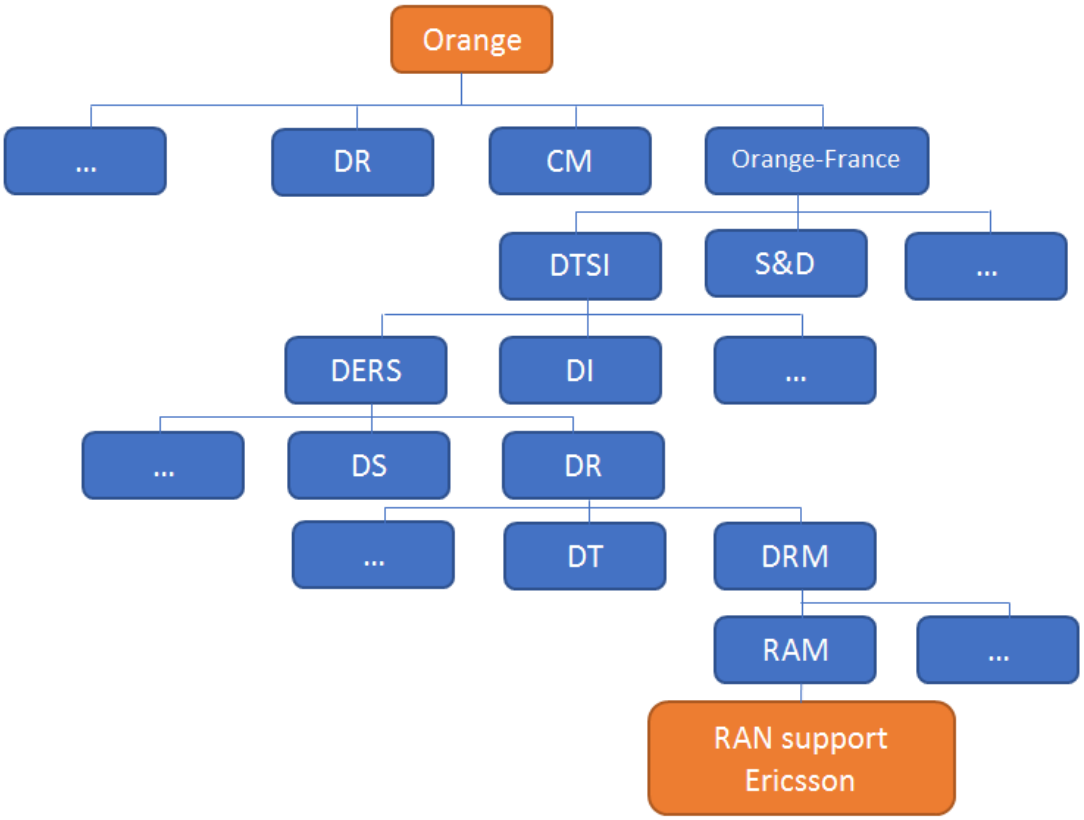

Figure 1: Organigramme partiel d'Orange

- DTSI : Direction Technique du Système d'Information
- DERS : Direction de l'Exploitation du Réseau et des Services
- DR (DERS/DR, pas Orange/DR) : Direction des Réseaux
- DRM : Direction du Réseau Mobile
- RAM : Réseau d'Accès Mobile
- RAN : Radio Access Network

II. Développement d'outils

Au cours du stage, j'ai été amené à réaliser plusieurs missions en lien avec les OSS. A l'avenir, les tâches qui me sont proposées vont se diversifier, bien que toujours centré autour de la réalisation de scripts back-end. Je vais commencer par revenir sur les tâches que j'ai déjà accomplies.

1. Le point

a) Environnement de travail

Un bon environnement de travail est fondamental pour un développeur. Avant de commencer quoi que ce soit, je préparai tout ce qui allait me servir par la suite. Les machines sur lesquelles je travaillais dans les locaux d'Orange tournaient sous Windows ; étant donné que j'allais faire du scripting, je me tournais vers Cygwin pour obtenir la majorité des programmes qui me serviraient dans l'avenir : cvs[7], git, python, bash, emacs, et surtout ssh qui allait me permettre d'accéder aux différentes machines des OSS. Sous Windows, j'ajoutai également Mozilla Firefox, PuTTY[8] et WinSCP[9], le dernier pour avoir une interface quand la situation deviendrait un peu délicate en ligne de commandes.

Etant donné que j'allais être amené à effectuer une très grande quantité de transferts de fichiers entre ma machine et les OSS, je me familiarisai avec la commande *ssh-agent*[10] afin d'automatiser la connexion aux OSS, et j'implémentai plusieurs scripts Bash qui me permettraient à l'avenir d'effectuer rapidement les transferts de fichiers.

La phase la plus complexe fut de configurer l'environnement sur les machines composant les OSS. Celles-ci tournent sous Solaris[11], un changement par rapport à l'environnement Debian / Xubuntu dont j'avais l'habitude. La plupart des commandes et programmes installés sur les OSS sont datés, certains programmes très pratiques (tels qu'emacs) ne sont pas présents et cela me força à me rabattre sur des alternatives pas toujours très pratiques (comme *vi*, que je considère comme l'un des pires éditeurs de texte avec lesquels il m'ait été donné de travailler).

Il me fallait également réécrire une grande partie des fichiers de configuration des terminaux afin de me connecter dans un shell Bash et non dans un shell tcsh, et charger automatiquement les clés SSH pour me connecter sur les autres machines. En effet, la seule machine accessible directement depuis mon poste dans les locaux d'Orange est la machine « Admin ». Toutes les autres machines de l'OSS ne sont accessibles que depuis la machine « Admin ».

La mise en place de l'agent SSH (la mémorisation des clés) sur la machine « Admin » fut difficile. La commande *ssh-agent* refusait d'exporter les variables d'environnement nécessaires pour que *ssh-add* fonctionne, il m'a donc fallu mettre en place un script complet pour récupérer la sortie de *ssh-agent* et en extraire les informations nécessaires. Cette opération fut rendue plus complexe en raison des outils datés que j'avais à ma disposition, et j'ai dû utiliser des stratégies complexes pour résoudre ce problème simple.

[7] Gestionnaire de versions, permettant de faire revenir un fichier ou un dossier à un état antérieur

[8] Un utilitaire Windows qui permet d'utiliser SSH pour se connecter sur une machine distante

[9] Un utilitaire permettant de transférer des fichiers entre plusieurs machines avec une interface graphique

[10] Une commande qui permet de charger une clé SSH en mémoire pour se connecter sur une machine distante sans saisir la phrase secrète de la clé

[11] Un système d'exploitation dérivé d'UNIX, développé par Oracle (précédemment par Sun)

b) Scripts Bash & packaging

La toute première mission qui me fut confiée concernait la réorganisation des fichiers utilisés par l'application User Management sur l'OMINF. J'avais déjà évoqué cette machine précédemment, voici quelques précisions.

Parmi les onze machines de chaque OSS, une seule est dédiée à l'OMINF (contrairement à, par exemple l'« Admin », qui est répartie sur deux machines pour augmenter la tolérance aux pannes). Cette machine agit comme annuaire et gestionnaire d'utilisateurs. C'est là que se trouvent les scripts fournis par Ericsson permettant de créer, supprimer, verrouiller ou déverrouiller un compte sur un OSS, changer un mot de passe de compte etc...

L'une des particularités de cette organisation est que si un utilisateur demande un compte sur plusieurs OSS, plusieurs machines OMINF seront interrogées. Les interactions entre les utilisateurs et l'OMINF se font via User Management.

User Management est une application Web ; à mon arrivée, son fonctionnement était le suivant :

- Un utilisateur se rend sur l'application et effectue sa demande de création de compte, ou de changement de mot de passe (sans aucune vérification de la cohérence des données saisies).
- La requête est envoyée au serveur Web, qui attend que Mr CLERC ou Mr MIOR la valide, en la stockant dans une base de données.
- Une fois validées, les requêtes sont placées dans un fichier « export.txt », que l'OMINF récupère tous les quarts d'heure via une *crontab*[12].
- Un script Bash traite le fichier pour récupérer les différentes demandes effectuées au cours du précédent quart d'heure ; il crée les comptes, effectue les changements de mots de passe, puis génère un fichier récapitulatif (import.txt)
- Le serveur Web récupère le fichier « import.txt » afin de voir si l'OMINF a réussi les requêtes et notifie les utilisateurs en cas de succès.

La création de compte sur un OSS ainsi que le changement de mot de passe sont des opérations complexes. L'OMINF doit ajouter les entrées dans l'annuaire LDAP[13] pour que l'utilisateur ait accès aux dix autres machines de l'OSS grâce à un seul et unique compte, de nombreuses vérifications doivent être effectuées (contraintes sur le format du mot de passe, contraintes sur le login, vérification qu'il n'y a pas de doublons, gestion des groupes, des profils etc…). Afin de simplifier la vie des administrateurs, Ericsson leur fourni plusieurs scripts, disponibles sur l'OMINF.

La personne qui s'était chargée de développer la version initiale d'User Management avait remanié ces scripts afin de retirer leur aspect interactif (ne plus demander à l'utilisateur quoi faire, mais lire les opérations depuis un fichier extérieur, export.txt) ; cela allait me permettre d'avancer un peu plus rapidement, les scripts fournis par Ericsson étant très complexes (Bash n'étant pas, selon moi, un langage très souple ou pratique pour programmer).

La conception de cette version initiale s'était faite sans respect des standards concernant l'organisation des fichiers et toute l'application se retrouvait au milieu des scripts d'Ericsson.

[12] Un ensemble de programmes qui s'exécutent de façon périodique (et configurable)

[13] Un annuaire qui stocke des utilisateurs ; par exemple, pour donner accès sur dix machines à un même utilisateur, on l'ajoute dans l'annuaire LDAP et chaque machine cherche dans l'annuaire quand l'utilisateur essaye de se connecter. Cela permet à l'utilisateur de n'utiliser qu'un seul couple « nom / mot de passe » sur les dix machines.

Les composants de l'application incluaient, entre autres, une paire de clés SSH, un fichier de données de configuration, quatre scripts Bash et un fichier de log, rempli par le script principal. Mon but était de réorganiser tout cela selon une hiérarchie plus pertinente : scripts de cœur dans */opt/ORANGE/bin*, données de configuration dans */etc/opt/ORANGE* et données temporaires dans */var/opt/ORANGE/iumgt* (internal user management, voir Figure 2 plus bas).

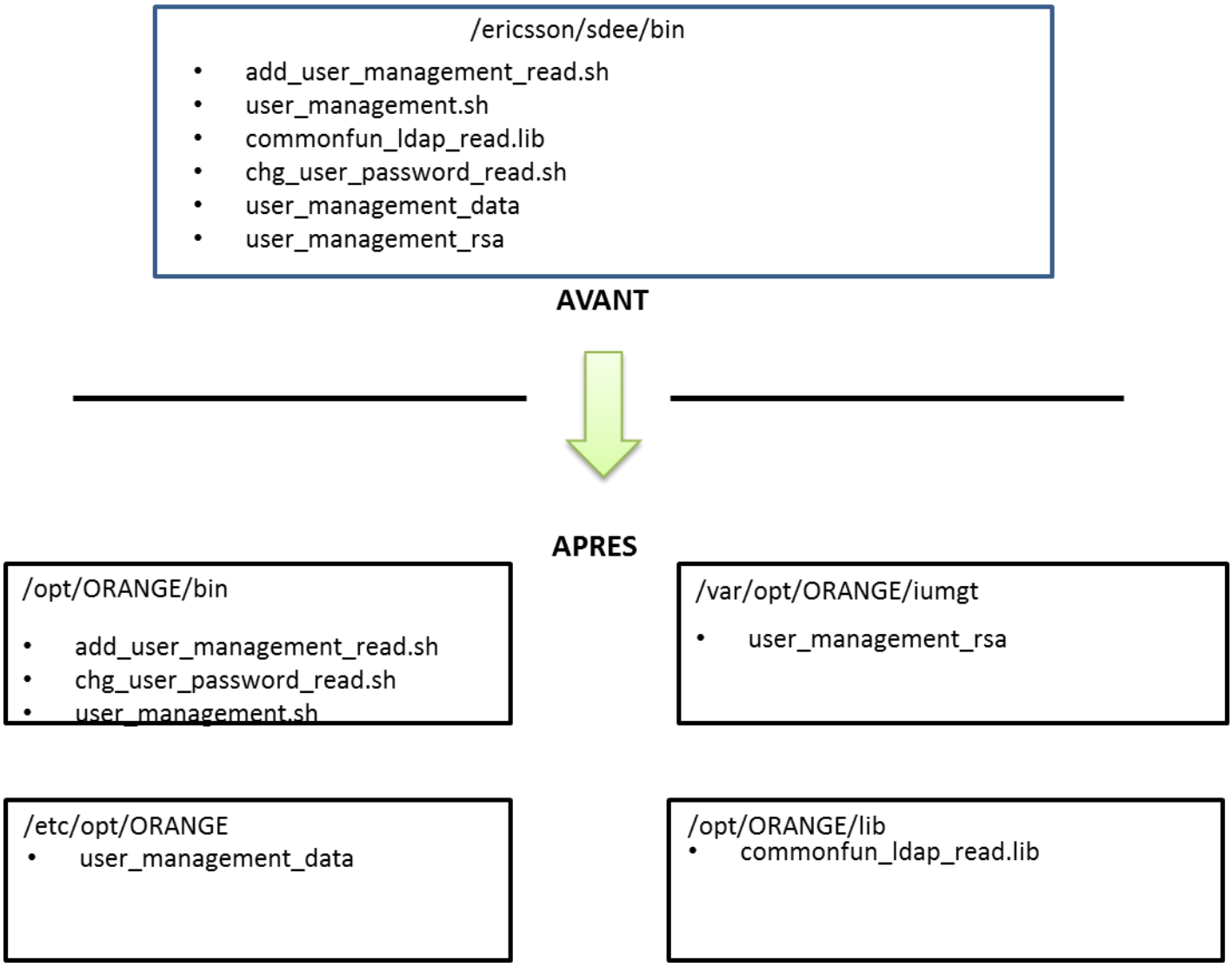

Figure 2: Organisation du script avant et après mon travail

Cette opération n'était pas compliquée, il fallait corriger les chemins dans les différents scripts appelés afin que les références fonctionnent toujours. J'ai du apprendre à travailler avec le système de packaging[14] d'Emmanuel CLERC pour packager l'application en un fichier package qui serait ensuite déployé sur chaque OSS, afin de ne pas refaire les modifications plusieurs fois. J'ai également dû me familiariser avec un environnement Solaris daté, et au langage Bash.

Jusqu'alors, j'avais utilisé Bash pour automatiser quelques opérations répétitives, comme rassembler plusieurs fichiers, les compresser et envoyer l'archive sur une machine distante. Je ne l'avais jamais utilisé en tant que langage de programmation.

[14] Globalement, on rassemble tous les fichiers dans une archive, et on rajoute des informations indiquant comment les fichiers doivent en être extraits (dans quels répertoires, avec quels droits d'accès…)

Ici, il s'agissait de réaliser un script complet, qui devait lire un fichier ligne par ligne, éclater chaque ligne selon un délimiteur, puis, en fonction du premier élément de la ligne, appeler un certain script externe.

La gestion des erreurs du script externe passait par la lecture de sa sortie standard, car Ericsson avait fait le choix de n'utiliser que deux codes de sortie pour leurs scripts de création de compte et de changement de mot de passe : 0 si tout s'était bien passé, et 1 en cas d'erreur. Il était impossible de savoir facilement si, par exemple, la création de compte avait échoué car le mot de passe ne respectait pas les contraintes sur les mots de passe, si le login était déjà pris, si le profil demandé n'existait pas, ou autre.

Après une semaine je finis par compléter la réorganisation des fichiers, et la gestion des erreurs.

c) Protocoles, tests unitaires et Python

L'étape suivante consistait à porter l'application en Python, et à l'enrichir : ajouter de nouvelles commandes, telles que le verrouillage, le déverrouillage et la suppression de comptes. Il allait également falloir mettre en place un nouveau format de trames pour les fichiers échangés entre l'application Web et l'OMINF ; format qu'il allait falloir documenter, pour le standardiser. De plus, je devais rendre l'application plus robuste, plus résistante aux erreurs de formatage dans le fichier source, et plus claire concernant les erreurs déclenchées dans les scripts d'Ericsson. Enfin, je devais implémenter des séries de tests unitaires[15], afin de vérifier que les modifications apportées à l'avenir ne provoqueraient pas de régression ailleurs dans le code.

Je commençai par l'écriture des nouvelles trames à échanger entre l'application Web et l'OMINF. Cela revenait à définir un protocole : un certain format pour les messages et une explication des différentes valeurs pouvant se trouver dans le message.

Le format que Mr CLERC et moi-même décidions de mettre en place consiste en ceci : une chaîne de caractères en « lower camel case[16] » indiquant le type du message, suivie du caractère pipe ('|'), suivi des différentes valeurs associées au message, chacune séparée par le caractère pipe.

Par exemple, pour créer un compte, la trame est définie comme :

createAccount|nom_du_compte|identifiant_web|description|liste,de,profils

L'écriture du document me prit plusieurs jours, et je revins plusieurs fois dessus afin de le compléter : ajout de codes d'erreurs, comme par exemple les codes de LDAP indiquant la raison d'un échec (entité déjà existante, violation de contrainte sur un certain champ). J'accompli tout cela sous la supervision d'Emmanuel CLERC, qui revenait vers moi pour pointer ce qu'il estimait qui n'allait pas ou qui pouvait être amélioré.

J'ai choisi d'utiliser Python car pour ce type de tâches, il propose beaucoup plus d'outils que Bash, et est également beaucoup plus souple. Par exemple, mettre en place des tests unitaires se fait simplement via le module « unittest », alors qu'en Bash il serait certainement nécessaire de développer une grande quantité d'outils.

Enfin, il faut prendre en compte le fait que, d'une part, ces tests vont également tourner sur des machines de production, qui ne possèdent pas de compilateur C++ ou Java, pour des raisons de sécurité, et d'autre part, il faut que les membre du RAN Support soient à même de

[15] Un ensemble de tests où chaque test est conçu pour évaluer une et une seule fonctionnalité bien précise.
[16] Une séquence de mots est écrite ainsi : uneSequenceDeMots

pouvoir reprendre ces scripts par la suite. Les membres du RAN utilisent essentiellement Bash, Perl et Python, il fallait donc rester sur un langage que tout le monde connaisse et que tout le monde puisse exploiter.

Je commençai par décider quel paradigme[17] utiliser pour réaliser le script (orienté objet, programmation procédurale, fonctionnelle). Compte tenu de l'expressivité de Python, de son typage dynamique, et plus généralement qu'il n'impose de contrainte sur le style de programmation, je me rabattais sur un mélange de programmation générique et procédurale, ne voyant pas l'intérêt d'abstraire le code dans une approche objet.

Par acquis de conscience, je m'efforçais de rendre le code aussi souple que possible, appliquant au mieux le principe de responsabilité unique[18]. Cela me conduisit à découper le code en multiples modules, créant une grande quantité de classes statiques (les énumérations natives manquent cruellement en Python) pour obtenir une organisation pratique.

La réalisation de la première version fonctionnelle du programme me prit environ une semaine, représentant un portage pur du script Bash, et comportait une gestion passive des erreurs (on attrape toutes les exceptions et on les ignore avant de passer à la ligne suivante dans le fichier). Mr CLERC m'invita à assouplir l'ensemble pour pouvoir implémenter plus facilement les tests unitaires.

Je m'intéressais alors au module « unittest » de Python, permettant de réaliser des tests unitaires, ainsi qu'au module « argparse », afin de mieux gérer l'entrée en ligne de commande, pour pouvoir assouplir l'exécution de façon dynamique (par exemple, intégrer un mode debug).

Implémenter les tests unitaires se révéla assez complexe. Leur nature invite à rendre le code aussi souple que possible, et je pris peu à peu conscience de certaines règles tacites en programmation (par exemple, chaque référence en dur à une constante rend le programme moins paramétrable, étant donné qu'il devient contraint par la valeur de cette constante).

J'ai rapidement été confronté à un problème que je ne parvenais pas à résoudre, ce qui me demanda d'analyser le code en profondeur pour essayer de mieux le comprendre. A terme je compris que le problème venait de l'algorithme de vérification de la syntaxe des commandes placées dans le fichier d'export. Plutôt que d'énoncer clairement dans le code comment chaque ligne devait être formatée, je partais du principe qu'elle était bien formatée et rattrapais le programme si jamais il y avait un problème.

En découplant cet ensemble de vérifications dans un module séparé que je pouvais tester de façon indépendante, je pouvais enfin tester l'ensemble du programme. Je profitais également de ce changement d'architecture pour reconsidérer certains choix effectués précédemment. Je déplaçai ainsi le système de logging dans un module séparé du module des utilitaires.

C'est au bout d'une nouvelle semaine que je finis par implémenter l'intégralité des tests unitaires nécessaires pour vérifier création de compte et changement de mot de passe. Cela constituant la première milestone[19] dans le portage de l'application en Python, Mr CLERC me proposa de commencer le listing des comptes présents sur un OSS, en attendant que Valentin

[17] Style de programmation

[18] Single Responsibility Principle, ou SRP : chaque classe, chaque fonction, chaque variable d'un programme doit effectuer ou correspondre à une et une seule action ou idée, indivisible.

[19] Une étape (un jalon)

finisse son remaniement de l'application Web. Une fois cela fait, nous pourrions tester si les fichiers qu'il produisait à présent étaient bien lus par ma nouvelle version des scripts.

En parallèle de tout cela, je mettais régulièrement à jour la référence concernant le format des messages, lorsque je réalisais que l'on pouvait ajouter certains codes d'erreurs, ou simplifier certains messages.

d) Des listes, du Python et du Bash

Après avoir réalisé une application Python assez conséquente, revenir sur un script plus simple était plus évident. Une fois familiarisé avec LDAP, dont j'avais entendu parler, mais que je n'avais jamais eu l'occasion de manipuler, je pouvais commencer.

La réalisation du script fut rapide, m'occupant une demi-journée, car j'avais déjà implémenté des wrappers[20] autour de certaines fonctionnalités manquantes en Python 2.6 par rapport à Python 2.7 lors du portage des scripts Bash d'User Management (qui ont formé une partie de mon module d'utilitaires).

Par exemple, Python 2.6 ne possède pas de fonctionnalité permettant d'appeler une commande shell et de récupérer simplement la sortie standard de celle-ci (il est nécessaire de passer par un ensemble de classes différentes pour y arriver, alors que Python 2.7 propose directement une fonction pour faire ça).

Je vais maintenant expliquer l'intérêt de ce listing et justifier le choix de Python comme langage de programmation.

Par rapport au listing, la plupart des utilisateurs demandent un accès aux OSS pour une correction rapide, et n'ont généralement pas besoin d'y revenir régulièrement. Parfois, plusieurs mois s'écoulent, et les comptes inutilisés s'accumulent. L'idée du script est de lister tous les utilisateurs et de leur associer leur date de dernière connexion, ainsi que la méthode utilisée (ssh / login physique) et l'IP source (dans le cas de ssh).

Le script peut ainsi indiquer si un compte n'a pas été utilisé depuis extrêmement longtemps, et notifier Mr CLERC et Mr MIOR afin qu'ils puissent le verrouiller. Si l'utilisateur ne réagit pas à ce verrouillage au bout d'un certain temps, alors le compte est supprimé, afin de libérer un peu d'espace.

Ce système diffère quelque peu de ce qui est mis en place pour les échanges entre User Management et l'OMINF. Ici, les machines qui interviennent sont les machines « Admin » et « UAS » (les machines qui offrent des interfaces graphiques pour permettre une administration plus simple du réseau radio, avec des applications comme Citrix, qui permettent de manipuler les antennes etc…). Le script s'exécute sur celles-ci, extrait la liste des comptes et les différentes informations, les place dans un fichier qui est ensuite récupéré par la machine hébergeant le serveur Web pour que l'application Web User Management puisse afficher ces informations.

Concernant le choix de Python par rapport à Bash, voilà pourquoi : en Bash, une combinaison de *getent*, *cut* et *awk*[21] permettrait d'extraire la liste très rapidement. Néanmoins, le fait que les machines tournent sous Solaris rendait le tout un peu plus compliqué que prévu. Les architectures Linux possèdent par exemple *lastlog* (commande) qui permet de connaître la

[20] Utilitaire destiné à corriger un problème (workaround ⇔ travail autour).
[21] Des commandes qui font plein de choses bien pratiques

méthode, la source et la date de dernière connexion de tous les utilisateurs, et contient en particulier l'année de dernière connexion. Solaris ne possède pas *lastlog*, mais possède *last* (autre commande), qui fait globalement la même chose, sauf qu'elle n'indique pas l'année de dernière connexion, qu'il faut déduire à partir de la cohérence entre le mois, le jour, l'heure et la minute indiquée et la date actuelle. Effectuer ce genre d'opération en Bash est assez… crispant, compte tenu du manque de souplesse de Bash. Enfin, réaliser tout cela en Bash aurait conduit à une baisse de performances.

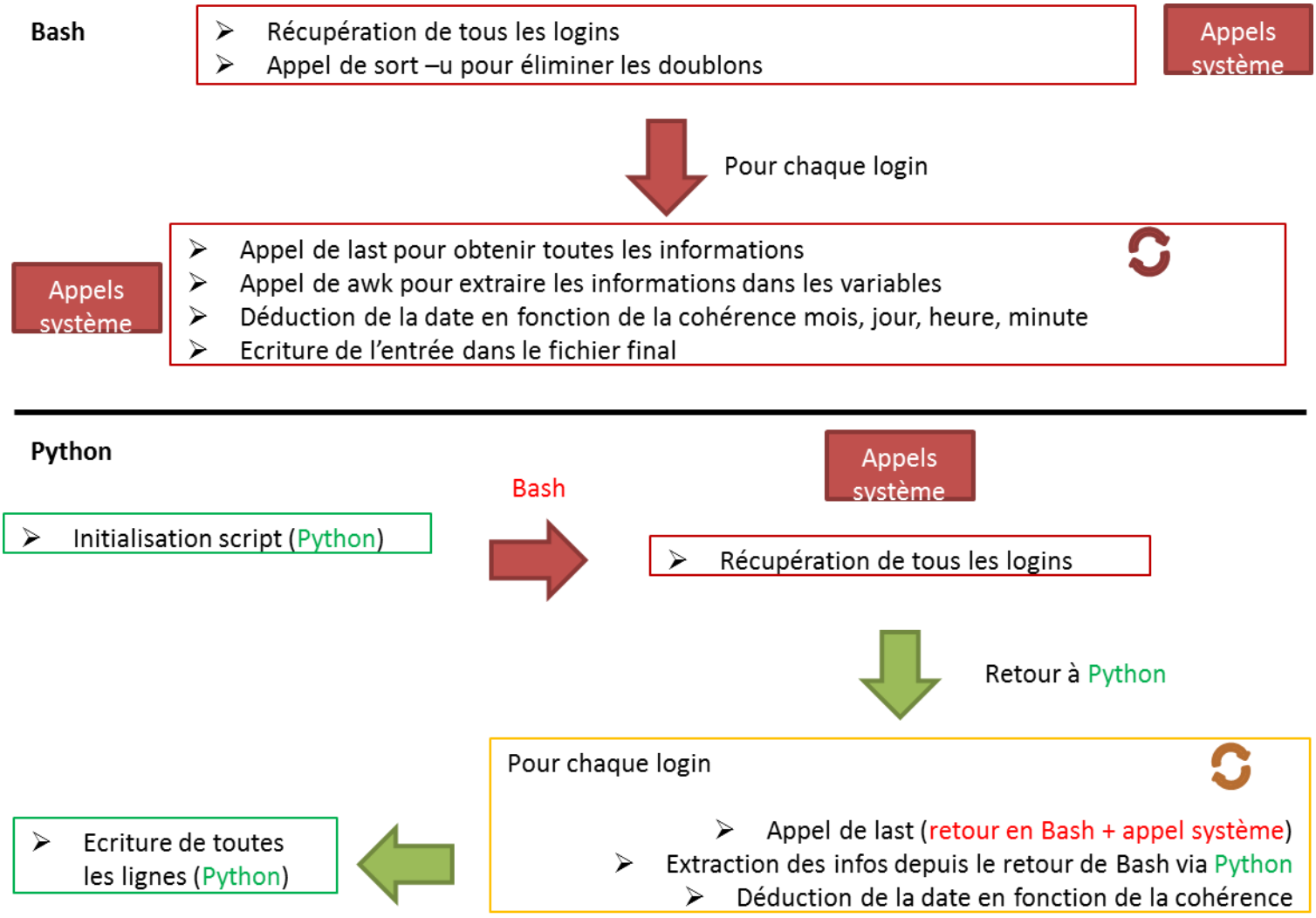

Figure 3: Comparaison Bash / Python pour l'extraction des dates de dernière connexion

Pour effectuer cela en Bash il aurait fallu commencer par faire appel à *getent* pour obtenir tous les utilisateurs enregistrés dans LDAP, ouvrir */etc/passwd* et *cut* la première colonne. Il aurait ensuite fallu supprimer les doublons dans le cas où *getent* liste également les comptes UNIX (donc un appel à *sort* en plus), puis itérer sur le retour de sort pour appeler *last* sur chaque nom, puis utiliser *awk* pour extraire les informations intéressantes, puis lire le retour d'*awk* pour déduire l'année si besoin est, puis écrire dans un fichier.

La quantité de commandes utilisées, en plus de la complexité générale de l'ensemble, m'ont fortement découragé.[22]

En Python, on conserve l'appel à *getent*, on conserve la lecture de */etc/passwd*, mais une fois que c'est fait, on reste en Python. Toutes les opérations sur les chaînes sont faites via la bibliothèque standard de Python, et on peut profiter de sa syntaxe nettement plus souple que celle de Bash. On peut objecter, légitimement, que le gain de temps n'est pas optimal, sachant que le script va s'exécuter une fois par jour. Mais je trouve Python infiniment préférable quand il s'agit de faire du scripting. Bash est très bon pour automatiser des suites de commandes, mais quand il s'agit de gérer un script complet, un autre langage est le bienvenu.

Une fois le script réalisé, et afin de ne pas pêcher par orgueil, j'ajoutai également des tests unitaires pour vérifier des cas extrêmes. J'en profitais ainsi pour apporter quelques corrections, majoritairement au niveau de la déduction de la date. Enfin, je mis à jour la documentation pour indiquer le format des messages générés et supprimai quelques codes d'erreurs que j'avais estimés utiles, mais qui finalement se révélaient dénués d'intérêt. Une fois cela fait, je partais compléter le script Python d'User Management pour gérer le verrouillage, le déverrouillage et la suppression de compte. Cela revient fondamentalement à ce que j'évoquais en première partie, aussi je ne reviendrai pas dessus. Une fois l'implémentation faite, je mettais en place de nouveaux tests unitaires. Après m'être assurés que les tests se déroulaient sans problèmes, je déployais sur la machine de pré-production pour tester l'application. Quelques corrections plus tard, je commitais la première version stable du script Python sur le dépôt CVS et partait donner un coup de main à Valentin sur l'application Web.

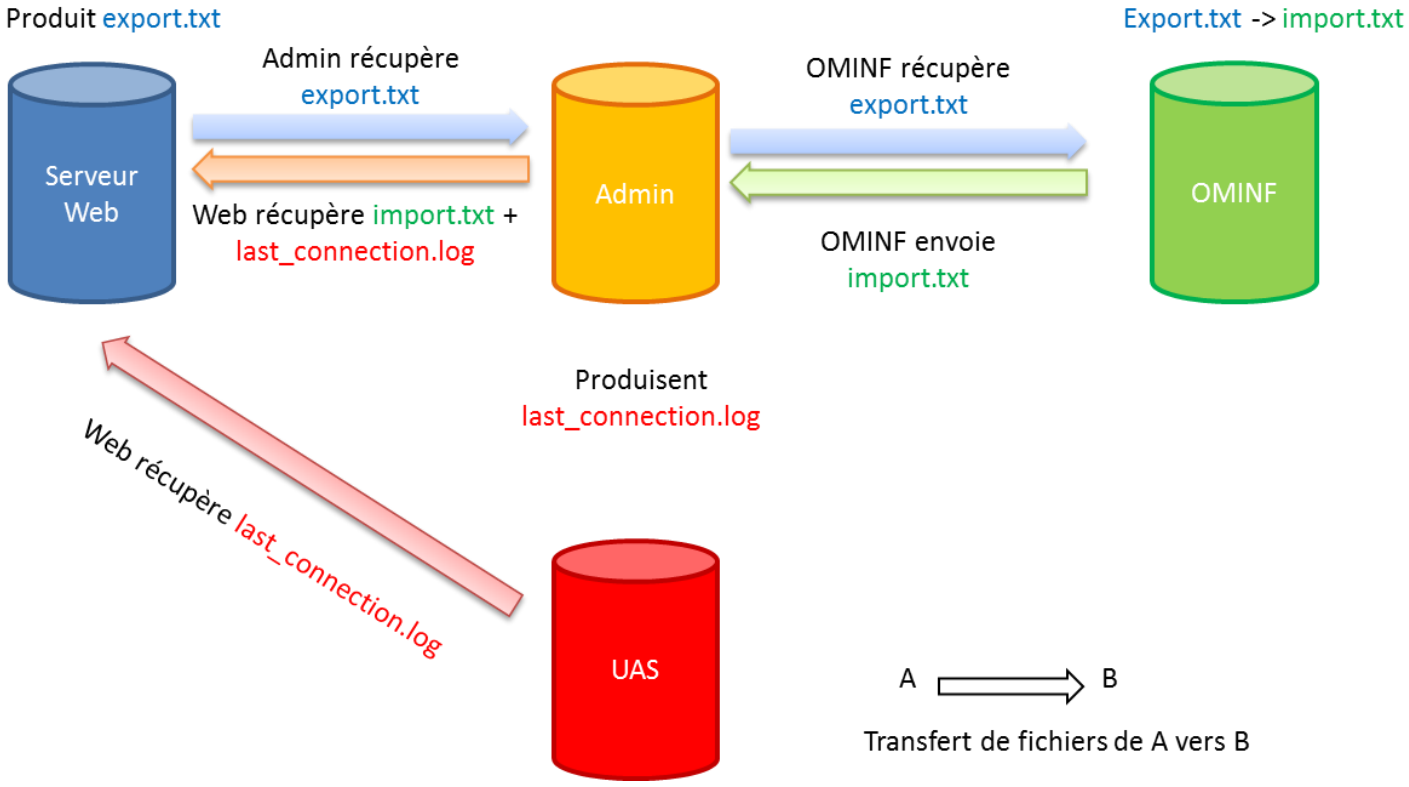

Figure 4: Schéma des échanges de fichiers pour User Management

[22] Le facteur le plus gênant est la grande quantité d'appels système dans le cas du script Bash. Chaque appel est coûteux, puisqu'il faut passer en mode noyau, effectuer des opérations d'entrée – sortie très souvent etc…

e) PHP & MySQL

En parallèle de tout cela, j'intervenais occasionnellement pour aider Valentin en PHP / MySQL. Je lui apportais ainsi quelques conseils et corrections de code pour éviter les failles les plus habituelles, que ce soit les injections SQL[23] ou les failles XSS[24] en JavaScript ; je lui donnais également quelques astuces pour corriger le comportement capricieux de PHP, que ce soit au niveau de la gestion des inclusions de fichiers ou du logging des erreurs ; enfin, je lui venais en aide sur certaines requêtes SQL particulièrement complexes quand elles se présentaient.

Occasionnellement, je donnais mon avis sur l'ergonomie de l'IHM qu'il concevait, afin de pointer ce qui me semblait être des problèmes ou des constructions peu intuitives.

f) REST & ENM[25]

Après avoir constaté ma progression dans le script Python, Emmanuel CLERC me donna accès à ENM, la nouvelle plateforme d'Ericsson destinée à remplacer les OSS, dont je parlais en introduction. Le fonctionnement d'ENM est quelque peu différent de celui des OSS, aussi je vais développer un peu plus le sujet.

ENM fonctionne sur une API REST, et propose également une interface web qui est un miroir et une extension de cette API. Autrement dit tout ce que l'on peut faire via l'API REST est faisable via l'interface Web, mais l'interface Web permet de faire plus. Toutefois, l'API REST permet d'automatiser facilement les opérations, contrairement à l'interface Web.

Une API REST est un ensemble de fonctionnalités offert à l'utilisateur et qui respecte certaines contraintes sur lesquelles je ne vais pas m'attarder. Pour simplifier, imaginez que vous pouvez envoyer une requête HTTP sur un certain fichier de l'API, et que cela déclenche une action en conséquence (création de compte, changement de mot de passe etc…).

REST repose sur les verbes de HTTP : GET pour obtenir une ressource, POST pour ajouter une ressource, PUT pour modifier une ressource existante et DELETE pour supprimer une ressource. En utilisant ces différents verbes, on peut créer différentes requêtes qui auront différents effets, dépendant de quel fichier est interrogé[26].

Mon travail ici était de mettre en place un programme qui dialoguerait avec ENM afin d'effectuer les mêmes opérations que sur l'OMINF : créer des comptes sur ENM, changer le mot de passe, verrouiller, déverrouiller ou supprimer un compte. Contrairement à la version OMINF qui appelait des scripts Bash de façon claire, ici le programme allait dialoguer avec

[23] Une personne mal intentionnée remplit un formulaire avec le contenu d'une requête SQL pour interagir avec la base de données à l'insu des développeurs. Par exemple, remplir le champ « Nom » d'un formulaire avec « Amaury OR 1 = 1 » peut être ajouté à la clause WHERE d'une requête SQL destinée à récupérer les informations sur un utilisateur. En évaluant la clause, SQL verrait la condition 1 = 1, qui est tout le temps vrai, et donc renverrait toutes les informations de tous les utilisateurs.

[24] Une personne mal intentionnée remplit un formulaire avec du code JavaScript pour qu'il soit intégré (potentiellement) à la page Web générée par PHP. Ce code peut faire tout ce que l'utilisateur veut, donc potentiellement voler des données importantes. Echapper les balises « script » en PHP est une méthode simple pour prévenir à ce genre d'attaque.

[25] Ericsson Network Management, la plateforme destinée à remplacer les OSS.

[26] Je simplifie grandement ce qui fait qu'une API est bien une API REST, mon but n'est pas de faire un cours entier sur le sujet.

ENM au travers de l'API REST, et cela allait inévitablement avoir des conséquences au niveau du code.

Une partie du script Python réalisé sur l'OMINF allait me servir. Toute la logique du code était la même, les seules parties différentes allaient être les constantes représentant les chemins de fichiers et les fonctions qui allaient interroger les scripts.

Par conséquent, il me semblait plus logique de continuer en Python plutôt que de revenir en Bash et faire des appels à Bash depuis Python.

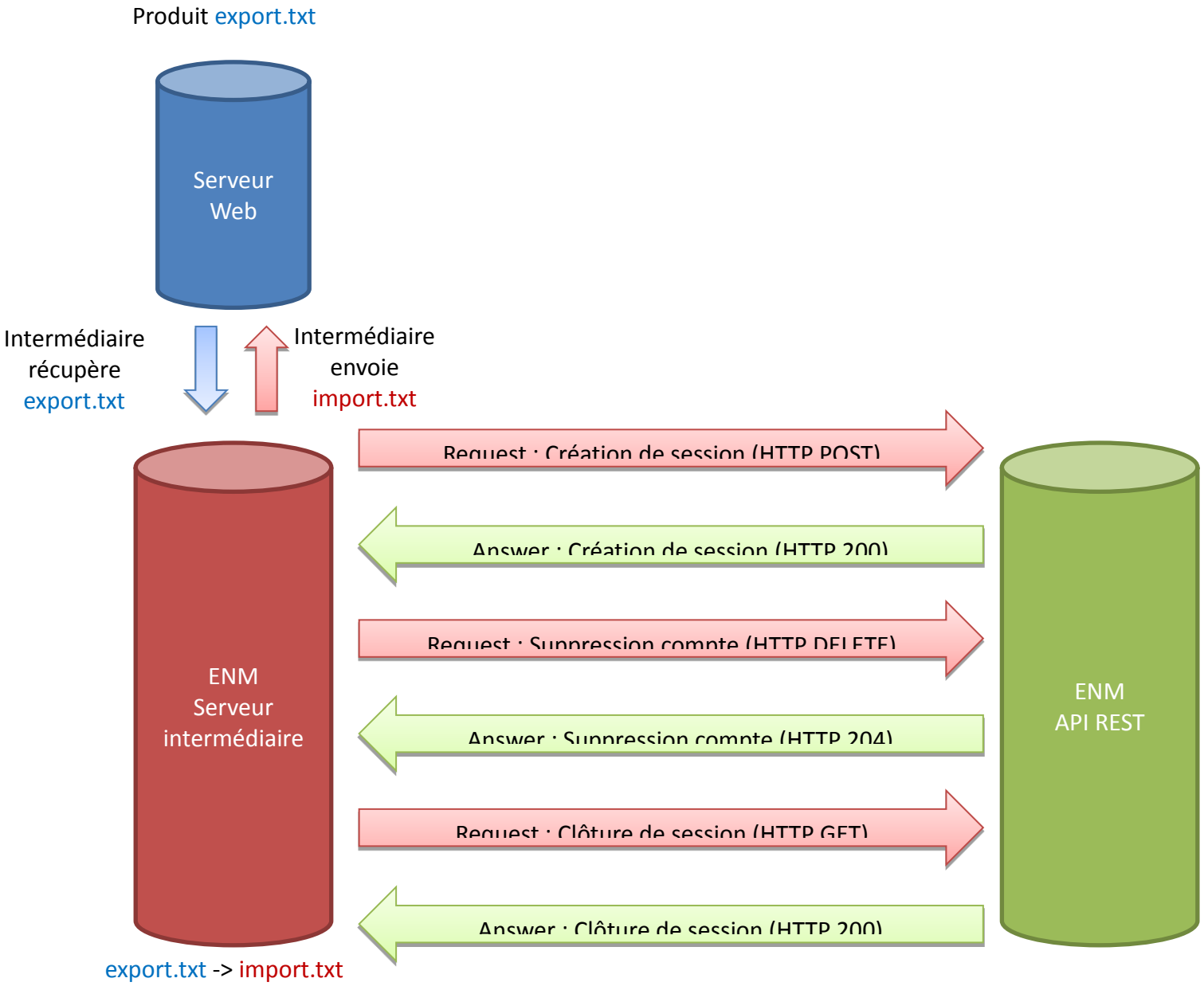

Figure 5: Echanges entre ENM et le serveur Web via REST

Une fois ce choix effectué, il me fallait trouver un moyen de communiquer avec ENM. Python 2.6 possédait le module « urllib2 » qui permettait de dialoguer avec une API REST, mais de façon assez laborieuse. J'avais entendu parler de la bibliothèque « requests », mais je ne l'avais jamais vraiment regardée en profondeur.

« requests » est probablement l'une des bibliothèques les plus élégantes qu'il m'ait été donné de manipuler depuis que j'ai commencé à programmer il y a cinq ans. Souple, extrêmement bien conçue, intuitive, bien documentée. Elle proposait également une gestion des sessions qui allait m'éviter de devoir renseigner certificats SSL[27] et cookies[28] à chaque requête.

Je commençai par me familiariser avec « requests », interrogeant ENM à la fois en curl[29] et en « requests », pour m'assurer que ce que j'obtenais était cohérent. Je pris la bibliothèque en main très rapidement. Le lendemain je commençais à coder les fonctions qui allaient me permettre d'interroger ENM.

Contrairement aux scripts Bash sur l'OMINF, l'API REST était, elle, très bien conçue, et la documentation fournie par Ericsson presque impeccable (quoiqu'un peu simpliste par moments). Cela me permit de réaliser l'ensemble des fonctions dans la journée. Il ne me restait plus qu'à extraire les fonctions communes du code de mon script pour User Management pour pouvoir compléter l'application.

g) Une première bibliothèque (en quelque sorte)

Une bibliothèque est un outil aussi pratique qu'il est complexe à créer. Il s'agit d'un ensemble de fonctions et classes qui permettent d'effectuer des opérations, le tout mis à la disposition des développeurs. Un outil de rêve, par exemple pour concevoir des interfaces graphiques. Si Qt, GTK ou encore SDL n'existaient pas concevoir des interfaces graphiques serait surement plus complexe.

Concevoir une bibliothèque est bien plus délicat que réaliser une application. Créer une bibliothèque, c'est mettre en place quelque chose qui ne pose pas de contraintes à l'utilisateur.

Autant certaines parties du script Python pouvaient fonctionner toutes seules (le module des utilitaires, le module de logging ou encore le module de vérification des lignes), autant d'autres étaient particulièrement liées entre elles. C'est ce couplage entre modules qui est justement à éviter, et qui a été problématique.

J'ai choisi de créer une bibliothèque car le fonctionnement de l'application allait être le même sur l'OMINF et sur ENM, à quelques différences près. On allait toujours récupérer des fichiers, les lire ligne par ligne, valider chaque ligne puis la traiter.

Deux approches s'offraient à moi :

- Une approche orientée objet avec polymorphisme, qui me semblait délicate à mettre en place (car créer les objets appropriés aurait demandé de déterminer la machine sur laquelle l'application se trouve, ce qui aurait rendu l'application dépendante) ;
- L'approche « bibliothèque » : extraire le code commun aux deux applications dans une petite bibliothèque et réaliser deux applications se reposant sur elle. Cela permet également d'étendre le fonctionnement à n'importe quelle machine : il

[27] Un protocole qui complète HTTP (on parle alors de HTTPS) en chiffrant les données échangées entre le client le serveur. On parle aussi de SSL / TLS (TLS a aujourd'hui supplanté SSL).

[28] Des données mises en cache, rééchangées entre le client et le serveur à chaque requête HTTP.

[29] Une bibliothèque qui permet d'envoyer des requêtes HTTP.

suffit d'enregistrer les bonnes fonctions (que l'utilisateur peut lui-même créer) et rédiger le bon fichier de packaging. J'ai été confronté à de multiples difficultés.

J'avais intégré le code de lancement de l'application dans le module « run ». Le module principal se contentait de quelques opérations internes puis appelait ce module « run ». Néanmoins, je me suis rendu compte, après plusieurs journées de réflexion, que cette idée n'était pas bonne. La fonction de lancement du module « run » effectuait un certain nombre d'opérations qui étaient propres au fonctionnement de l'application sur l'OMINF. Je ne pouvais donc pas conserver ce mode de lancement.

Je m'étais aperçu de cette erreur dès le début de ma réflexion, mais je n'avais pas envisagé la solution immédiatement. J'avais par exemple envisagé de simuler les templates de C++ en Python, mais cela était particulièrement compliqué autant pour moi que pour l'utilisateur.

Je finis par décider de laisser l'utilisateur réaliser sa propre phase d'initialisation avant de lancer le module « run ». En d'autres termes, cela revenait à découpler le module application « defines » et le module bibliothèque « run », de façon à ce que la dépendance soit application ➔ bibliothèque et non plus application ⇔ bibliothèque.

Une fois ce découplage fait, il ne me restait plus qu'à m'assurer que le code fonctionnait toujours sur l'OMINF. Après quoi je pouvais achever l'application User Management pour ENM.

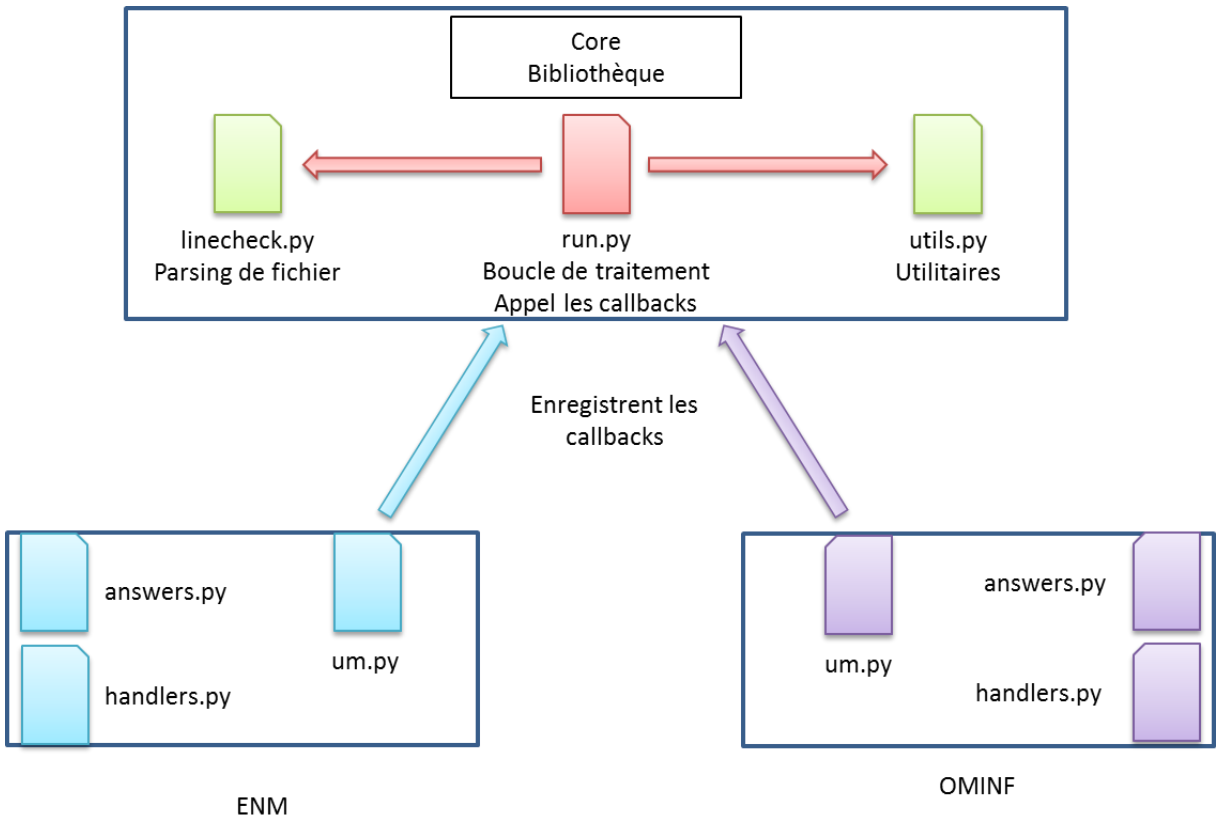

Figure 6: Diagramme "d'activité" de la bibliothèque

2. Futurs projets

Ce rapport étant écrit après un mois et demi de stage, il ne peut représenter l'intégralité de son contenu ; c'est pourquoi je vais également évoquer les différentes tâches qui m'attendent à l'avenir, et les stratégies que j'ai prévu d'utiliser pour les réaliser.

a) Sélénium, des tests unitaires et du PHP

Selenium est un framework permettant d'implémenter des tests unitaires dans divers langages, notamment en Python. Il permettra de tester l'interface web d'User Management.

Pour réaliser cela, je travaillerai avec Valentin ROUSSEL. Nous n'avons pas encore décidé de la répartition des tâches, étant donné qu'il nous reste encore quelques points à améliorer dans nos applications respectives. Je pense m'occuper de la partie consistant à configurer Selenium via la ligne de commande.

En ce qui concerne l'implémentation des tests, ce sera la première fois que Valentin sera confronté à cette problématique, aussi je pense pouvoir lui apprendre ce que j'ai découvert dans ce domaine en Python (rendre le code souple pour pouvoir tester chaque fonctionnalité individuellement, réfléchir aux cas extrêmes etc…). Nous nous répartirons l'écriture des tests afin d'éviter qu'un seul d'entre nous ne s'occupe de tout.

b) Message queue : des logs et de la communication réseau

Une message queue est une structure utilisée pour la communication entre processus. L'idée de fonctionnement est la suivante : la file d'attente est créée, un processus devient producteur et écrit dans la file, un autre processus devient consommateur et lit les messages dans la file.

Le fonctionnement de la file est asynchrone, c'est-à-dire que l'émetteur et le récepteur n'ont pas besoin d'interagir en même temps. Le récepteur peut très bien faire plusieurs opérations avant de regarder s'il y a quelque chose dans la file. Le fonctionnement est un peu différent de celui d'un socket[30].

Un socket permet, lui aussi, d'échanger des messages entre deux processus (qui peuvent être situés sur la même machine ou sur deux machines différentes). En revanche, les sockets TCP (fiables) ont la particularité de posséder un timeout de fermeture. C'est-à-dire, quand l'un des deux processus n'a plus besoin d'envoyer de message, il ferme le socket de son côté. Un message indiquant cette fermeture est automatiquement envoyé à l'autre extrémité du socket. A cause de TCP, l'autre socket doit s'assurer qu'il n'y a pas de messages qui seraient coincés quelque part dans le réseau, donc il va attendre un certain temps (via un timeout). Quand le timeout expire, le socket estime qu'il n'y a plus aucun message en attente et se ferme.

Ce mécanisme est problématique pour la raison suivante : supposons que l'outil qui envoie les messages soit un programme Bash. Chaque appel du programme Bash ouvrirait un socket pour envoyer un message, puis le fermerait. La machine distante, sur laquelle les messages sont envoyés, commencerait alors à accumuler des sockets en attente de fermeture, ce qui est problématique. L'utilisation de message queues permet de s'affranchir de ce problème.

[30] Un socket (ou BSD socket) est un outil utilisé en communication entre processus, permettant de faire communiquer deux processus situés sur deux machines différente. Pour faire simple, un couple de sockets représente les deux extrémités d'un tunnel de communication, chaque processus peut écrire sur le socket sur sa machine, et le message est ensuite envoyé à l'autre socket, et l'autre processus peut lire sur son socket.

En effet, les sockets et les messages queue opèrent à deux niveaux différents[31], les messages queue étant situées plus haut que les sockets. Pour que deux processus communiquent via une message queue, ils n'ont pas besoin d'avoir connaissance de l'existence de l'autre et n'ont même pas besoin d'exister au même moment pour que la message queue fonctionne. Les sockets, elles, requièrent que les deux processus existent et aient connaissance de l'autre.

Par conséquent, si on reprend le programme en Bash, il suffirait qu'il ait connaissance de la queue pour envoyer son message, sans se soucier de considérations techniques[32].

J'ai donc pour objectif d'implémenter une message queue en Python, afin de compléter le système de logging développé par Emmanuel CLERC, pour que tous les logs soient envoyés sur un serveur centralisé pour conserver une trace. Je ne sais pas encore très bien comment je vais implémenter tout cela, je pense qu'il doit très certainement y avoir une bibliothèque en Python qui propose une implémentation ou un début d'implémentation. Il me faudra commencer mes recherches une fois que j'aurai fini de travailler sur ENM.

[31] Plus un système est bas niveau, plus il est proche du fonctionnement de la machine.

[32] Un message queue utilise très probablement des sockets pour communiquer à travers le réseau. La différence majeure est que seuls deux sockets (dans le cas où seules deux machines communiquent) sont mis en place et restent actifs en permanence, au lieu de s'accumuler.

III. Rétrospective

L'objectif du stage n'est pas uniquement de nous permettre de découvrir le monde de l'entreprise, c'est aussi une occasion pour nous de découvrir qui nous sommes, et ce que nous voulons faire plus tard, ce qui nous plairait. Je vais maintenant présenter ce qu'il m'a apporté.

1. Professionnellement

Sur le plan professionnel, le stage a été particulier. J'ai évoqué plusieurs fois le fait que l'équipe dans laquelle j'ai été intégrée n'est pas une équipe de développeurs, aussi cela a eu un impact sur le stage. Valentin et moi ne formions pas réellement une équipe travaillant sur un même projet avec des tâches bien réparties ; fondamentalement le seul point sur lequel nous devions faire en sorte que nos travaux soient harmonisés était le format des messages échangés entre nos deux applications, ce qui représente une part minime du travail accompli.

J'estime donc que le stage n'aura pas permis de vraiment découvrir le fonctionnement de l'informatique en entreprise. Certes, Emmanuel CLERC nous aura parlé du cycle de vie d'un logiciel en entreprise, nous aurons également assisté à quelques phases d'intégration, mais le contexte ne permettait pas une « vraie » immersion. J'aurais par exemple aimé voir comment fonctionne la répartition des tâches entre les développeurs, ou découvrir comment un chef de projet gère un projet. Le stage aura permis une introduction, et je pense qu'il serait profitable de l'approfondir.

Le fait que j'ai été intégré dans une équipe de personnes qui ne sont pas des développeurs ne m'a pas énormément gêné, mais je trouve que cela m'a rendu disjoint des autres membres. User Management, et encore moins sa partie backend, n'est pas une application qui est au sommet des priorités du RAN. Ses membres ont leurs propres missions, j'ai les miennes, et cela n'a pas permis beaucoup d'interactions professionnelles (au sens où il n'y avait aucune raison que j'interagisse avec leur travail ou qu'eux interagissent avec le mien) ; elles seront restées plus personnelles.

Toutefois, j'ai pu observer les interactions internes au RAN, et j'ai constaté qu'elles se rapprochent assez de ce que j'avais imaginé ; chacun vient en aide à l'autre quand il y a un problème un peu pointu. Cela a d'ailleurs été le cas entre Valentin et moi quand nous étions confrontés à des problèmes complexes, et qu'un regard externe était utile.

Toutefois, le stage n'aura pas été inutile. J'ai pu apprendre à me débrouiller en autonomie, à chercher à résoudre des problèmes par moi-même. J'ai aussi eu un aperçu de la cohésion d'une équipe avec Valentin, étant donné que nous nous sommes soutenus mutuellement, même si nos différentes tâches n'avaient pas grand-chose à voir les unes avec les autres.

Par exemple, nous avons passé une journée entière sur Selenium, qui refusait obstinément de marcher, jusqu'à arriver à le faire fonctionner, alors que normalement seul Valentin aurait dû s'en occuper (et j'aurais dû me consacrer à mes propres tâches), mais je ne pouvais pas le laisser tout seul devant ce problème (qu'au final nous avons résolu ensemble).

J'ai aussi eu l'occasion de découvrir plusieurs modules utiles en Python, comme par exemple « unittest » qui m'a permis de redécouvrir les tests unitaires et leur utilité. J'ai également pu approfondir mes connaissances en PHP et redécouvrir MySQL que je n'avais pas utilisé en dehors de l'université depuis deux ans.

Le stage a également été pour moi l'occasion d'améliorer ma façon de programmer. J'ai ainsi attaché plus d'importance à la modularité du code, et au découpage en fonctions (les

tests unitaires aidant beaucoup à ce niveau). Cela m'a conduit à réaliser pour la première fois une bibliothèque, aussi minime soit-elle.

J'ai aussi pu découvrir des outils comme Selenium qui me seront surement utiles dans l'avenir (notamment si on doit réaliser un site web en master). Également, l'outil « coverage » qui permet de regarder quelles zones de code ont été parcourues au cours du l'utilisation d'un script Python ; combiné avec des tests unitaires cela permet de voir qu'est-ce qui a été effectivement testé et quelles zones n'ont pas été atteintes. Enfin, ce que je considère comme l'apprentissage le plus marquant, celui de « requests » en Python, et le fonctionnement de REST.

Professionnellement, le stage aura donc été très profitable individuellement, mais un peu moins dans l'aspect « informatique en entreprise ».

2. Personnellement

Sur le plan personnel, le stage m'aura permis de me poser des questions sur mon avenir et sur qui je suis.

En réfléchissant attentivement aux moments durant lesquels j'étais particulièrement intéressé par ce que je faisais, et les moments où j'étais un peu plus lassé ou ennuyé, j'ai fini par constater que ce qui m'intéresse le plus, au-delà de simplement programmer et voir le résultat de plusieurs centaines de lignes de code, c'est avant tout la réflexion autour de ces lignes de code. Comment organiser le code ? Quels outils utiliser ? Prendre une feuille de papier, faire un dessin de l'architecture, regarder où elle peut aller, estimer où elle doit aller, tout cela me fascine, car c'est un nouveau challenge à chaque fois.

Plus globalement, j'aime réfléchir à des problèmes, et à comment les résoudre. J'aime partir de quelque chose qui semble flou, et progressivement construire une solution. L'implémentation de la solution est la partie finale, mais n'est pas nécessairement la plus intéressante (même si c'est toujours une immense satisfaction que de voir son programme fonctionner). A voir vers quoi cela pourrait me mener.

Un autre point que j'ai pu confirmer est le fait que j'aime énormément apprendre aux autres ; j'ai passé beaucoup de temps à conseiller Valentin, que ce soit lors de nos discussions sur comment implémenter tel ou tel concept, lorsque je lui parlais des principes fondamentaux de conception (qui, selon moi, ne sont pas assez enseignés à l'université, comme par exemple le fondamental principe de responsabilité unique, ou l'importance de logs) ou simplement comment bien rédiger son code (consistance du nommage des variables, des fonctions, limite des 80 caractères par lignes, espaces autour des opérateurs et autres conseils en typographie). Le code qu'il écrit aujourd'hui est beaucoup plus agréable à lire et peut être debugué beaucoup plus facilement[33], et cette évolution me fait extrêmement plaisir. Peut-être qu'un avenir dans l'enseignement pourrait être envisagé.

[33] Je ne cherche pas à rabaisser Valentin ou à me placer au-dessus de lui. Il a produit du code fonctionnel, et il a rempli toutes les demandes de son tuteur, sans que je fasse quoi que ce soit à sa place. Je n'ai fait que lui donner des conseils, et je constate une évolution dans sa manière de programmer.

J’ai déjà effectué une rapide rétrospective du stage, sur ses apports professionnels et personnels. Je vais maintenant dresser un bilan.

Je n’avais pas réellement d’attente en commençant le stage. Pour moi, c’était une UE obligatoire, potentiellement intéressante car permettant de découvrir le monde de l’entreprise ; mais ce n’est pas non plus l’UE que j’attendais avec le plus d’impatience. Je n’avais aucune idée de comment tout allait se passer, une vague idée de ce que j’allais faire et n’attendais pas le stage avec impatience.

Le fait de travailler chez Orange, au sein d’une équipe, même si mes travaux étaient quelque peu disjoints de ceux de ses membres, place le stage en entreprise, et j’ai pu observer comment tout cela fonctionnait, donc j’ai découvert le monde de l’entreprise.

Néanmoins, si l’on prend le stage sous l’aspect « découverte de l’informatique en entreprise », la conclusion est moins tranchée. Le fait que je ne me sois pas retrouvé dans une équipe de développeurs ne m’a pas réellement permis de voir comment se déroule un projet informatique en entreprise. Certes, Emmanuel CLERC m’a présente le cycle de vie du logiciel, nous avons appliqué ce cycle, mais dans un contexte bien particulier. L’application que je réalisais était destinée à fonctionner en interne, et était relativement simple. Ce n’était pas un projet de plusieurs mois répondant aux besoins d’un client externe, mené par une équipe sous la responsabilité d’un chef. En ce sens, et pour moi, le stage ne valide pas l’aspect « découverte de l’informatique en entreprise ».[34]

Toutefois, je ne dirais pas que le stage (ou du moins sa première moitié) est raté. J’ai eu un aperçu de l’entreprise, j’ai pu découvrir de nouveaux outils, approfondir mes connaissances et mon expérience dans les domaines de la programmation et de la conception, j’ai pu poser un regard plus critique sur moi-même, tout cela je le considère comme positif.

Le bilan est donc mitigé. D’un côté le stage aura été profitable sur bien des aspects, d’un autre il lui aura manqué l’aspect « informatique en entreprise ». Néanmoins, j’en suis globalement satisfait, et il m’aura conforté dans ma décision de continuer mes études en master informatique.

[34] Je ne blâme pas Orange. Emmanuel CLERC m’avait prévenu lors de mon entretien que je ne serai pas dans une équipe de développement. J’aurais pu alors ne pas donner suite et chercher un autre stage que j’aurais estimé plus pertinent. Néanmoins, j’ai choisi de privilégier le contenu du stage par rapport à l’aspect « découverte de l’informatique en entreprise ».

ANNEXES

I. Ressources utilisées (documentation et outils)

Pour toutes mes questions concernant Python, je me suis en premier lieu tourné vers la documentation officielle de Python 2.7.13 (https://docs.python.org/2/), puis sur le site de questions / réponses StackOverflow (https://stackoverflow.com/) pour toutes les questions concernant certaines mécaniques internes au langage que la référence ne parvenait pas à clarifier. Concernant le fonctionnement de « unittest » et de « coverage » en Python, les scripts réalisés par Emmanuel CLERC ont servi de référence.

La documentation de « requests » en Python : http://docs.python-requests.org/en/master/. Egalement, les pages manuels de « curl » pour les tests initiaux avec ENM.

En ce qui concerne Bash, StackOverflow, à nouveau, pour l'aspect programmation en Bash, ainsi que Unix & Linux StackExchange (https://unix.stackexchange.com/), pour l'aspect manipulation (utilisation de commandes etc…). Et, bien évidemment, la documentation officielle de Bash (https://www.gnu.org/software/bash/manual/bash.html).
Pour les questions relevant d'administration système, comme le fonctionnement de Cygwin ou ssh-agent, Super User (https://superuser.com/). Enfin, pour les questions concernant la sécurité, Security StackExchange (https://security.stackexchange.com/).

Au niveau de la documentation de l'API REST d'ENM, la documentation « Alex » fournie par Ericsson s'est avérée très complète et très bien réalisée (bien que, comme je l'ai pointé précédemment, un peu simpliste par moments). Pour des raisons évidentes, je ne peux pas la placer ici.

Concernant les commandes Solaris, leurs pages manuels, et également la documentation disponible sur le site d'Oracle (http://docs.oracle.com/cd/E26505_01/index.html). Le site d'OpenLDAP a également été très utile pour le fonctionnement de LDAP (https://www.openldap.org/), de même qu'OpenClassrooms pour une introduction au protocole LDAP (https://openclassrooms.com/courses/presentation-du-concept-d-annuaire-ldap).

Sur le fonctionnement de Base64, le site de l'IETF a été extrêmement pratique pour l'accès aux RFC originaux (https://tools.ietf.org/html/rfc3548).

Lors des moments où j'aidais Valentin en PHP / MySQL, leurs références respectives (http://php.net/manual/en/ et https://dev.mysql.com/doc/refman/5.7/en/) ont résolu énormément de problèmes, ainsi que StackOverflow (encore et toujours).

Enfin, pour diverses questions d'organisation de code, Emmanuel CLERC, qui a toujours su me fournir des explications claires et précises.

Et également d'autres ressources sur Internet, beaucoup trop nombreuses pour être citées, comme des tutoriels sur le fonctionnement de getopts en Bash, des explications sur les serveurs proxy et beaucoup d'autres… Au niveau outils, PuTTY, Python, Bash, Cygwin, WinSCP…