Stage - Ingénieur d'études développeur dans le domaine du Contrôle aérien

Antoine ROCQ

2023-2024

Remerciements : Je tiens à tout particulièrement remercier Claudine SIMONIN pour son aide et son soutien tout au long de mon apprentissage et de la réalisation de mes missions.

Sommaire

1. Introduction

Dans un monde où la sécurité et l'efficacité du trafic aérien reposent sur des systèmes de plus en plus sophistiqués et où les exigences en sécurité sont de plus en plus élevées, le développement d'outils robustes et performants est essentiel. Pour arriver à un résultat combinant sécurité et performance, le travail sur la génération de configurations pour un gestionnaire de configuration et de supervision est nécessaire. Cependant, peu de solutions adaptées aux nombreuses montées en version et modifications nécessaires au sein de projets exigeants, en collaboration avec l'armée française, ont été mises en place.

C'est précisément le défi qui m'a été présenté : créer des solutions pour faciliter la manipulation et l'utilisation des nombreux fichiers XML, utilisés dans divers aspects du contrôle aérien, et notamment pour les projets SRSA et CLA, au sein desquels des missions m'ont été confiées. La mission de ce stage était donc de créer des programmes de parseurs XML ayant diverses fonctions, à adapter selon les besoins des projets auxquels j'ai été assigné et selon les demandes et retours du client.

Il s'agissait également d'un travail de maintenance et de correction du logiciel CMoIP, afin de me familiariser avec un travail plus pratique et moins théorique, où je pouvais manipuler graphiquement les fichiers XML que je créais avec mon programme, observer les erreurs, les corriger, mais aussi effectuer des mises à niveau. CMoIP (Configuration an Management over IP) est un logiciel développé par CS Group en java, qui assure 2 fonctions : configuration d'un système et supervision en utilisant le protocole standard SNMP(Simple Network Management Protocol). Sur cet aspect, le temps et les missions à effectuer variaient en fonction de l'avancée de la mission, mais aussi de l'intervention des autres équipes. En effet, chaque équipe de chaque projet communique étroitement et l'action d'une équipe peut avoir des répercussions sur une autre. Il est donc assez commun de voir des changements et des erreurs sur CMoIP créés par l'intégration du travail d'une autre équipe.

Ce contexte exigeant m'a permis de plonger dans des environnements complexes où la rigueur et la précision sont primordiales. Mon stage au sein de CS Group s'est articulé autour de plusieurs axes principaux, tous visant à optimiser les processus et à garantir la robustesse des systèmes de contrôle aérien. Le développement de parseurs XML, la maintenance et la correction de configurations dans CMoIP, le développement de modules d'affichage personnalisés de fichiers XML, mais aussi de fichiers YAML, ont fait l'objet de missions tout au long de mon stage.

Cependant, d'autres aspects majeurs de ce stage sont communs à toutes ces missions et ne sont pas nécessairement mentionnés de manière explicite dans l'énoncé des tâches à réaliser. Ces aspects incluent, par exemple, la collaboration étroite avec les autres équipes de CS Group. Chaque équipe, bien que spécialisée dans des domaines distincts, contribue au projet global, et toute modification apportée par une équipe peut entraîner des répercussions sur les autres. J'ai donc appris à naviguer dans cet environnement collaboratif, à communiquer efficacement avec mes collègues et à anticiper les impacts de mes modifications sur le travail des autres.

Cette expérience m'a permis d'acquérir des notions de gestion de projet dont la coordination,

essentielles pour assurer la cohésion et l'efficacité des équipes dans un contexte de développement de systèmes critiques. Il y a aussi l'adaptation et la réactivité. En effet, la nature de ce stage m'a appris à être flexible et réactif face aux changements constants des besoins du projet et des demandes des clients. Le fait de devoir adapter régulièrement mes outils et mes méthodes de travail en fonction des retours des utilisateurs finaux et des évolutions du projet m'a permis de comprendre l'importance de l'agilité dans le développement logiciel. Cela m'a également sensibilisé à l'importance de l'innovation continue pour répondre aux exigences croissantes en matière de sécurité et de performance dans le domaine du contrôle aérien. En raison de la nature sensible des projets de CS Group, j'ai été tenu de signer une clause de confidentialité stricte. Cette clause vise à protéger la propriété intellectuelle, les informations stratégiques et les données sensibles auxquelles j'ai eu accès pendant mon stage.

La problématique est alors la suivante :

Comment développer des outils de génération et de manipulation de fichiers XML qui automatisent et optimisent les processus de configuration et de supervision des systèmes de contrôle aérien, tout en répondant aux exigences de sécurité et de performance des projets collaborant avec l'armée française ? Comment assurer également le bon fonctionnement et le maintien de la plateforme CMoIP afin de garantir son fonctionnement et la collaboration efficace entre les différentes équipes du projet ?

Nous débuterons par une présentation de l'entreprise, subdivisée en plusieurs sections : un aperçu global de CS Group, de ses activités et de son organisation ; l'histoire de l'entreprise depuis sa création et les étapes majeures de son développement ; les changements et opportunités à la suite de l'acquisition par Sopra Steria ; et les initiatives et projets innovants de l'entreprise. Ensuite, nous explorerons les différentes missions que j'ai réalisées durant mon stage : découverte et gestion du logiciel CMoIP ; extraction et génération de nouveaux fichiers XML ; méthodes de comparaison et présentation des différences entre fichiers XML ; gestion des fichiers YAML pour les systèmes ; et travail sur les modèles graphiques et leur visualisation. Nous conclurons cette section par un bilan des missions et des compétences acquises. Enfin, la conclusion abordera plusieurs aspects : l'importance de la responsabilité individuelle et de la collaboration au sein de l'équipe de développement ; les effets sociétaux positifs de l'activité de CS Group ; et les initiatives de l'entreprise en matière de durabilité et de réduction de l'empreinte écologique. Le rapport se terminera par une liste des sources utilisées pour la rédaction.

2. L'entreprise

A. Présentation générale

CS Group est une entreprise française spécialisée dans la création et la gestion de systèmes critiques dans les domaines de la défense et de la sécurité, de l'espace, de l'aéronautique et de l'énergie. Son siège social se trouve à Paris, et le groupe compte également 12 autres sites en France, ainsi que des filiales à l'étranger dans des pays comme l'Allemagne, le Royaume-Uni, les Etats-Unis ou encore les Pays-Bas. L'activité du groupe s'organise autour du développement et de l'intégration de logiciels, de la maîtrise d'œuvre et du déploiement d'applications industrielles, scientifiques et techniques, ainsi que de la conception des architectures techniques de systèmes d'information et de réseaux. CS Group est un leader notamment dans

la défense et la sécurité, et plus particulièrement dans le domaine du contrôle aérien, où elle est en concurrence directe avec Thalès. C'est également dans ce domaine que j'ai réalisé mon stage.

B. Historique et évolution

CS Group a été créé en 1902, son nom étant alors Compagnie des Signaux et d'Entreprises Électriques. En 1999, ce nom change temporairement pour devenir ombrelle CS Communication et Systèmes, avant de finalement devenir CS Group en 2019. Son slogan est « *La force de l'innovation* », et sa forme juridique est une société par action simplifiées (SAS), dans le conseil en systèmes et logiciels informatiques. Son chiffre d'affaires était de 238 millions d'euros en 2022, et elle compte plus de 2100 collaborateurs en France et à l'étranger.

En 2022, un rapprochement avec Sopra Steria est annoncé, et en 2023, Sopra Steria finalise l'acquisition d'une participation majoritaire au capital de CS Group. Sopra Steria est une entreprise de services numériques et de conseil en transformation numérique des entreprises française, qui a été fondée en 2015, dont le chiffre d'affaires était de 5,8 milliards d'euros en 2023, et qui compte 56000 collaborateurs dans près de 30 pays. C'est un acteur majeur de la tech en France et en Europe.

En 2023, à la suite de l'acquisition majoritaire par Sopra Steria, CS Group a renforcé sa position sur le marché des systèmes critiques et de la transformation numérique. Cette intégration permet à CS Group de bénéficier des ressources et de l'expertise de Sopra Steria, tout en continuant à se concentrer sur ses domaines de prédilection tels que la défense, la sécurité, l'aéronautique, l'espace et l'énergie. L'intégration dans le groupe Sopra Steria ouvre également de nouvelles perspectives pour CS Group en termes de développement international et de capacité d'innovation. Les synergies entre les deux entreprises permettent d'optimiser les compétences et les solutions proposées aux clients, renforçant ainsi leur compétitivité sur le marché global.

C. Perspectives post-acquisition

CS Group continue de développer des solutions technologiques avancées, notamment dans le domaine du contrôle aérien où elle reste un acteur clé face à des concurrents comme Thales. En parallèle, l'entreprise investit dans la recherche et le développement pour maintenir son avance technologique et répondre aux besoins évolutifs de ses clients. L'engagement de CS Group envers l'innovation se manifeste également par sa participation à divers projets européens et internationaux, visant à développer des technologies de pointe dans les secteurs de la défense, de la sécurité et de l'aéronautique. Par exemple, l'entreprise collabore avec des institutions de recherche et des partenaires industriels pour développer des solutions basées sur l'intelligence artificielle et le big data, améliorant ainsi la performance et la fiabilité des systèmes critiques.

D. Innovation et recherche et développement

En termes de responsabilité sociale et environnementale, CS Group s'efforce de mener ses activités de manière durable. L'entreprise met en œuvre des initiatives pour réduire son

empreinte carbone, promouvoir la diversité et l'inclusion, et soutenir des projets communautaires et éducatifs.

Ainsi, après avoir dressé le portrait détaillé de l'entreprise et exploré ses divers aspects organisationnels, il est désormais pertinent d'approfondir les missions concrètes qui m'ont été confiées. Cette étape permettra d'analyser comment j'ai appliqué mes compétences et connaissances acquises, tout en évaluant l'impact de ces missions sur les objectifs stratégiques de l'entreprise.

3. Les missions

Les missions sont multiples : d'un côté, développer un parseur XML complet avec plusieurs utilités, à adapter selon les besoins des projets sur lesquels je travaille. De l'autre, travailler directement depuis des clients sur la plateforme CMoIP pour effectuer des changements de configuration de travail, créer des plans de remplacement, corriger des modèles graphiques, ou encore remplir les différents objets. CMoIP signifie « Configuration & Management over IP ». Pour ma part, j'allais simplement m'occuper de l'aspect « Management » de CMoIP, et pas de la partie « Supervision ». L'ensemble de mes missions vont donc avoir pour but de faciliter l'utilisation de CMoIP, et de proposer des solutions qui répondent au besoin des utilisateurs de ce logiciel. Pour faire cela, je vais créer des outils de manipulation de fichiers XML, fichiers qui sont utilisés pour les configurations qui seront ensuite entrées et utilisées sur CMoIP. La problématique était alors la suivante : comment créer des outils efficaces afin de faciliter la gestion des configurations ainsi que le travail sur des fichiers XML, tout en effectuant des tâches de maintenance et de correction sur le logiciel CMoIP, en m'assurant que les logiciels XML que mon parseur a créé sont conformes et ne posent aucun soucis aux différentes tâches que les autres équipes et que le client va devoir effectuer lors de son utilisation ?

A. Première mission : Manipulation, supervision et correction de CMoIP

Introduction à la mission

Contexte et objectif de la mission

Comme énoncé précédemment, un aspect très important de la mission qui m'a été confiée était de faciliter le travail sur la génération, modification et utilisation des fichiers XML, et notamment dans le cadre de leur intégration puis du travail sur la plateforme CMoIP. En effet, pour travailler sur ce logiciel, il faut utiliser une configuration de travail. Cette configuration est un fichier XML qui doit être importé puis chargé comme « configuration applicable », qui une fois chargée, va être utilisée par tous les clients. Ainsi, le parseur XML que j'allais réaliser devait être rigoureux dans son traitement des objets, car la moindre erreur empêcherait le travail sur CMoIP. Il était alors nécessaire pour moi d'avoir une idée plus précise du fonctionnement de ce logiciel, afin de pouvoir identifier les objets, observer comment ils interagissent les uns avec les autres, remarquer les problèmes dont on m'a parlé, réfléchir à un plan d'approche lors de la réalisation de mon programme.

Familiarisation avec CMoIP

Rencontre avec la tutrice de stage

Ainsi j'ai débuté mon stage avec la manipulation de CMoIP et l'observation des différentes interfaces avec lesquelles j'allais devoir interagir. J'ai donc commencé par faire la connaissance de Claudine, ma maîtresse de stage, qui allait me guider dans mon apprentissage des bases du contrôle aérien et de CMoIP, avant de pouvoir me lancer sur des projets plus techniques, notamment avec d'autres équipes d'autres projets.

Découverte des interfaces de CMoIP

Tout d'abord, le logiciel fonctionne avec du « Drag and Drop », et utilise un OS sécurisé, avec le moins de fonctionnalités possibles. Un autre défi se posait alors : la version python présente sur les clients n'était pas la dernière disponible. Mon programme devait pouvoir fonctionner sur n'importe quelle machine, il fallait donc que je m'adapte et que je n'utilise que des fonctionnalités de python 3.4.5 ou antérieur. Ensuite, pour accéder au logiciel, il faut s'identifier depuis un poste client. Il existe alors différents identifiants et mots de passe qui donneront accès à des fonctionnalités de CMoIP différentes selon les identifiants utilisés. En effet, dans le cadre du contrôle aérien, un officier fonction n'aura pas besoin d'avoir accès aux mêmes fonctionnalités, ou de pouvoir voir les mêmes informations qu'un gestionnaire opérationnel. Il existe donc différentes sessions, mais celle que j'ai utilisé, et que d'une manière générale la plupart des personnes réalisant des tests ou des opérations sur CMoIP utilisent est la session « indus » (pour industriel), qui permet d'avoir accès à toutes les fonctionnalités, d'avoir accès à tous les onglets et d'effectuer toutes les opérations souhaitées sans problèmes d'accès.

Exploration des fonctionnalités de CMoIP

Onglet Supervision

J'ai ensuite découvert les différents onglets du logiciel, et donc ses fonctionnalités et son fonctionnement. Il y a tout d'abord l'onglet « Supervision », un onglet d'observation qui permet de voir l'état de tous les sites et de tous les objets présents dans ces sites. Par exemple, on peut voir que pour le site se trouvant à Mont-de-Marsan, le switch 1 présente une erreur critique. Si on double clique sur le switch 1, on peut observer que l'erreur critique viens d'un manque de liaison avec le client 4 de la baie 3 dans la salle C-303. On peut également observer la vue SO, qui correspond à la vue opérationnelle de la zone Sud-Ouest donc pour toutes les personnes qui ont un rôle dans l'opérationnel. On peut alors observer que le Site de Mont-de-Marsan s'occupe de la supervision de la zone Sud. En effet, la supervision du contrôle aérien se découpe en 3 zones. La zone Sud-Ouest, Sud Est, et la zone Nord. Selon les situations, chaque site se verra attribuer une ou plusieurs zones dans lesquelles il devra effectuer le contrôle. Cette séparation de la France en plusieurs zones permet une optimisation de la supervision et du contrôle aérien, afin d'assurer une plus grande efficacité et sécurité, ainsi qu'une meilleure adaptabilité à toutes les situations et tous les éventuels problèmes qui peuvent survenir.

Onglet Graphique

Il y a ensuite l'onglet « Graphique », qui contient tous les modèles graphiques des objets. Toutefois, ces modèles graphiques ne sont pas directement liés aux objets, mais chaque modèle graphique est associé à un seul type d'objet. L'onglet « Graphique » ne présente dans un premier temps que le modèle en soit, par exemple pour une baie, on verra toutes les propriétés de cette baie … et ce n'est que dans le sous-onglet « Groupes de synoptiques » de l'onglet « Graphique » que l'on pourra regrouper sur une seule vue différents modèle graphiques de différents types d'objets. Par exemple pour le synoptique baie, on pourra afficher les modèles graphiques des différents équipements présents dans la baie. C'est aussi dans l'onglet «

Groupes de synoptiques », que, en reprenant l'exemple précédent, on pourra depuis le modèle graphique de la baie créer les liens avec les switches, routeurs, postes clients ou serveurs avec du drag and drop. Si je drag and drop le modèle graphique du routeur vers la case qui lui correspond dans le modèle graphique de la baie, cela créera un lien, et quand j'observerai cette même baie depuis l'onglet « Supervision », je pourrais cliquer sur le routeur et cela m'affichera son état ainsi que l'état des différents éléments qui le composent et avec lesquels il a un lien. Voici un second élément très important afin de comprendre le fonctionnement de CMoIP, chaque objet et onglet sont liés. Les objets forment des sortes d'arborescence. Par exemple, il y a plusieurs sites. Chaque site possède plusieurs salles. Chaque salle possède plusieurs baies et autre équipement. Ces mêmes baies et équipements seront composés d'autres baies et équipements, et ainsi de suite. Il existe de nombreux liens entre chaque objet ou équipement.

Onglet Applicable

Un troisième onglet est l'onglet « Applicable ». Cet onglet est extrêmement important puisque c'est depuis cet onglet qu'on peut importer, exporter, modifier ou appliquer des configurations. Ces configurations sont les fichiers XML que je vais être amené à manipuler dans la deuxième partie de mon stage. J'aurais donc l'occasion de parler plus en détails de cet onglet dans la suite de la description de mes missions.

Onglets Applications et Plans de remplacement

Le dernier onglet est en fait un couple d'onglets : ce sont les onglets « Applications » et « Plans de remplacement ». La raison pour laquelle je présente ces deux onglets comme un seul et même onglet est qu'ils fonctionnent ensemble : l'un peut agir sur la création, suppression, modification et application de plans radios et téléphoniques (qui sont par ailleurs eux aussi des fichiers XML que je vais être amené à manipuler), tandis que l'autre permet d'appliquer un plan de remplacement et d'ainsi le transmettre à tous les autres clients. Un élément important à retenir est que si l'onglet « Plans de remplacement » est ouvert depuis un poste client 1, un poste client 2 ne pourra ouvrir ni « Plan de remplacement », ni « Applications » tant que l'onglet n'est pas fermé depuis le client 1.

Missions réalisées sur CMoIP

Navigation et observation initiale

Ainsi, ma première mission était principalement centrée sur la manipulation de CMoIP afin de me familiariser avec les différentes structures de données utilisées ainsi qu'avec le fonctionnement général du contrôle aérien, pour obtenir une meilleure compréhension des termes utilisés pour me garantir une meilleure compréhension du projet sur lequel je travaille, avoir une meilleure vue d'ensemble sur les autres projets et missions, avec lesquels j'allais potentiellement être amené à travailler, et enfin, cela me permettait de mettre du sens sur les tâches que j'allais réaliser.

J'ai donc passé mes premières journées de stage en plateforme. Ce terme désigne des salles aménagées pour héberger des baies de serveurs, avec tous les équipements nécessaires à la manipulation, l'observation et la supervision. C'est donc un endroit très bruyant avec de nombreux postes clients CMoIP, centre d'émission / réception, serveurs, routeurs, switch, chiffreurs, qui tournent en permanence. Cela allait me permettre d'agir à la fois sur CMoIP, et directement sur les équipements concernés.

Je me plaçais alors sur un client et dans un premier temps, je naviguais entre les différents onglets, observant l'état des différents équipements, découvrant les liens entre chacun d'entre eux.

Manipulation directe des équipements

Dans un deuxième temps, j'ai commencé à manipuler les équipements directement, afin d'observer comment mes actions étaient visibles directement sur CMoIP. Par exemple, en débranchant un câble, ou en déplaçant voyant, certains états passaient de « fonctionnels » à « critiques », pour diverses raisons. Débrancher le câble empêchait toute liaison avec les autres équipements auquel il était lié, mais empêche aussi le fonctionnement de certaines parties internes à l'équipement concerné. Déplacer un voyant peut faire passer ce voyant en « critique » ou « non connu », selon l'importance de l'équipement concerné et la rigueur en matière de sécurité qui a été mise en place pour lui. J'ai également pris en notes certaines des erreurs pour les équipements les plus importants et communs à de nombreux projets, comme cela me l'a été demandé par ma maître de stage.

Correction des erreurs

Et cela s'est révélé utile puisque dans un troisième temps, il m'a été demandé de corriger les erreurs observées. C'est cette étape qui était la plus difficile à réaliser puisqu'elle me demandait de comprendre suffisamment précisément les équipements et leur fonctionnement afin de pouvoir analyser l'erreur, comprendre d'où elle venait puis la corriger. C'est donc une étape qui m'a pris beaucoup de temps, mais j'ai pu, comme il me l'avait été demandé, corriger entièrement les erreurs de tout un équipement.

Traitement d'un Fait Technique (FT)

Enfin dans un quatrième temps, on m'a demandé de m'occuper d'un FT qui avait été rapporté par une autre équipe concernant les onglets « Applications » et « Plan de remplacement ». Un FT correspond à un « Fait Technique ». C'est le terme utilisé chez CS Group pour parler d'un problème qui a été relevé dans n'importe quel aspect de n'importe quel projet. Tous ces FT sont signalés et consignés sur une plateforme, qui permet de suivre l'évolution de ceux-ci, de leur création, puis leur application (qui correspond à l'acceptation de celui-ci), catégorisation (selon plusieurs niveaux de priorité), et leur correction. Cela permet également si on rencontre un problème de vérifier si celui-ci n'a pas déjà été relevé. Il fallait donc que je m'occupe d'un FT qui avait été remarqué par un membre de l'équipe, mais qui n'avait pas encore été rédigé, par manque de test. En effet, chaque FT doit respecter une certaine syntaxe et surtout un ensemble de test pour mieux guider celui qui va réaliser la correction. Cette quatrième étape était la plus concrète. L'erreur qui avait été relevée était la suivante. Comme je l'ai expliqué précédemment, l'onglet « Application » permet de créer ou de modifier des plans de remplacement déjà existants. Chaque plan de remplacement correspond à une situation donnée, et indique quels centres auront la charge de quelles zones. Par exemple, pour le plan NORMAL, le CDC MLV aura la zone du Sud et le CER LVI aura la zone Nord. Ces plans de remplacement sont concentrés dans des « scénarios ». L'utilisation de ces scénarios sont ensuite prévues à l'avance selon les vols planifiés, et il est prévu que pour telle journée, tel « scénario » soit appliqué puisqu'il contient tel plan et tel plans. Le FT survient lors du changement de « scénario »», afin de passer d'un scénario test, à un nouveau scénario ne contenant aucun plan en commun avec le premier. CMoIP ne connait donc plus le plan d'origine, puisqu'un nouveau scénario a été créé, avec d'autres plans. Il affiche alors depuis son interface : Plan : « … » avec un espace vide. Toutefois, après cela, tout changement de plan de remplacement est rendu

impossible, puisque pour se faire, CMoIP a besoin de remplacer quelque chose par quelque chose, il ne peut pas remplacer rien par quelque chose, ou en l'occurrence par quelque chose qu'il ne connait pas ou plus. Ce qui m'était demandé était alors de créer des plans de tests, afin de déterminer les actions à effectuer pour se retrouver dans telles ou telles situation, dans quels cas y a-t-il un bug et dans quel cas n'y a-t-il pas de bug. Pour ce faire, j'ai commencé par réfléchir à toutes les possibilités. Concrètement, elles pouvaient se résumer aux suivantes : la situation où aucun plan en commun n'est trouvé par CMoIP. La situation où tous les plans sont retrouvés par CMoIP (on mélange simplement les noms). Et enfin, le cas où chaque plan change sauf un seul, dans ce cas spécifique, il est alors intéressant de tester deux situations, celle où lors du changement du « scénario », on se place tout d'abord sur le plan en commun entre les deux scénarios, et le cas où on se place sur un plan que les deux scénarios n'ont pas en commun. J'ai donc effectué tous ces tests, en me plaçant sur deux clients différents pour reproduire le plus possible une situation réelle. J'ai alors identifié le schéma suivant : il existe deux cas où une erreur rend toute opération ou changement de plan de remplacement impossibles : celle où il n'y a aucun plan en commun, et celle où il y en a un en commun, dans le cas où on se place avant de changer de scénario sur un plan qui ne se trouve pas en commun (demandant par ailleurs un redémarrage du serveur pour réintégrer manuellement un plan de remplacement et pouvoir agir à nouveau).

Après cela, on m'a appris à rédiger un FT correctement, en respectant le plan et la syntaxe, et j'ai pu moi-même le rédiger. Cette première mission m'a permis d'avoir une bonne compréhension du contrôle aérien, des différents équipements présents sur site, et de la plateforme CMoIP, des connaissances nécessaires à la réalisation des tâches plus complexes qui allaient ensuite m'être confiées.

B. Deuxième mission : Parcours, traitement, et création de fichiers XML

Introduction à la mission

Contexte et objectif de la mission

Je me suis alors vu confier ma première tâche complexe, puisque j'allais directement travailler sur la génération de configurations de travail pour CMoIP. L'objectif affiché était de résoudre un problème rencontré par l'ensemble des équipes du projet SRSA et qui les empêchait de travailler efficacement. On m'a décrit le problème comme étant le suivant. Afin de générer et d'utiliser une configuration de travail sur CMoIP, il faut charger un fichier XML. Ce fichier XML est très volumineux puisqu'il contient tous les équipements d'un ou de plusieurs sites, ainsi que leurs caractéristiques. La structure d'un fichier XML fonctionne un petit peu comme un arbre. Au premier niveau, on a les informations propres au fichier XML lui-même, son nom, la version vers laquelle il pointe, ou encore sa description. Au deuxième niveau on a les différents objets et équipement. Chaque objet a différents attributs. Dans le cas des objets de configuration de travail, ces attributs sont « id », « class-name », et « nom ». Parmi ces attributs, seul « id » est un attribut unique. « class-name » fait référence au « type » d'objet en quelque sorte. « nom » permet d'identifier plus précisément l'objet dont il est question. Ensuite, au troisième niveau il y a les informations de ces objets. Différents informations et valeurs sont attribuées à ces objets.

Ainsi, le fait que ce fichier soit aussi volumineux rend difficile les opérations telles que l'application de la configuration de travail, qui met plusieurs minutes à charger, temps durant lequel CMoIP n'est pas utilisable ce qui est problématique. Également, la modification ou

encore la supervision sont complexes, à cause de la taille du fichier, et de l'absence de solution optimisée. Chaque recherche et correction d'erreur ou tout autre opération de ce type doit alors se réaliser à la main, et constitue donc une tâche longue et pénible. Toutefois, parmi tout le grand volume de données de la configuration de travail, seule une petite partie est réellement nécessaire au fonctionnement de CMoIP et à la réalisation d'opérations de test, maintient, supervision ou encore modification. Bien évidemment, lors du rendu du projet au client, la configuration de travail devra être complète, mais actuellement, le projet est encore en phase de développement, et requiert diverses opérations, et notamment des opérations de tests, modifications et corrections.

Le défi qui m'a été présenté était alors le suivant : créer un programme de traitement de fichier XML, facilitant leur manipulation et leur utilisation tout en permettant leur utilisation en tant que configuration de travail fonctionnelle, n'empêchant la réalisation d'aucune opération.

Ma mission était donc de récupérer tous ces équipements dans différents fichiers XML et de créer un nouveau fichier XML ne contenant que ces différents équipements, et étant le résultat de la concaténation des fichiers XML de configurations radios et téléphone, ainsi que des fichiers XML des CER (Centre d'Emission et de Réception), CDC (Centre de Contrôle) et CEX. Cela permettrait de réduire grandement la taille du fichier XML utilisé en tant que configuration de travail, afin de faciliter les opérations. Comme seuls certains équipements présents dans le fichier, la gestion d'erreur est facilitée, tout comme la modification. Le fichier met aussi moins de temps à charger ou à appliquer sur l'ensemble des clients.

Confrontation aux difficultés de la mission

Les défis étaient multiples : tout d'abord, il fallait que mon programme fonctionne sur tous les postes clients. Je devais donc le réaliser en tenant compte de leur version de python installée. Un autre défi était le temps d'exécution de mon programme. En effet, il fallait qu'il puisse être utilisé plusieurs fois, pour générer différents programmes selon les besoins des personnes l'utilisant, le but à long terme étant de mettre ce programme à la disposition des autres équipes pour que tous les projets puissent en profiter, en laissant à l'utilisateur la liberté de spécifier les équipements qu'il souhaite intégrer au fichier XML. De cette manière, le programme pourrait être utilisé par n'importe qui souhaitent créer une configuration de travail ou simplement effectuer du traitement sur un fichier XML.

A ces différents défis se sont ajoutés des problèmes et difficultés que j'ai rencontrés tout le long du processus de réflexion puis de la réalisation de ma mission. Tout d'abord, j'ai toujours effectué mes programmes depuis un IDE directement sans passer par le terminal, je n'étais donc pas très familier avec la notion d'arguments en python, même si mon apprentissage de Java m'a aidé à comprendre le fonctionnement général. J'ai donc trouvé un modèle, le module « sys », qui permet de récupérer chaque information renseignée dans le terminal. Il me suffisait alors de récupérer les arguments par leur index (en commençant par 1 car l'argument 0 correspond au nom du programme qui sera lancé). Également, en me renseignant sur les fichiers XML et leur traitement dans python, j'ai découvert un module très utile. J'ai alors commencé mon code en me basant presque uniquement sur les méthodes de ce module, puis lors d'un premier test, j'ai eu un message d'erreur m'indiquant l'absence de ce module. D'habitude, rien de plus simple, il me suffit d'installer le module en lignes de commandes. Mais comme j'ai pu l'expliquer plus tôt dans mon rapport, les clients fonctionnent avec un OS sécurisé, optimisé et simplifié au maximum nommé « Seducs » (Système d'Exploitation Durci

CS). Il fallait donc que je travaille avec les ressources disponibles, sans installer de module supplémentaire. Je me suis donc rabattu sur un autre module. J'ai également été confronté avec le changement de système d'exploitation. D'habitude, je travaillais directement sur mac, sur un IDE comme Visual Studio Code. Toutefois, pour cette mission, comme les fichiers de Conf étaient hautement confidentiels, je ne pouvais pas les mettre sur une clé USB sans autorisation afin de tester le code directement depuis mon ordinateur. Il me fallait donc coder directement depuis le client, et pour cela, il me fallait utiliser des lignes de commandes linux. J'ai été, lors de mon apprentissage, amené à apprendre ces commandes mais jamais à les utiliser couramment et avec un cadre exigeant une certaine aisance et rapidité comme celui-ci. J'ai donc dû apprendre certaines commandes ou raccourcis afin de pouvoir mieux travailler, notamment pour le test de mon code, nécessitant d'ouvrir et de visualiser les fichiers crées. Cet apprentissage m'a beaucoup plu et j'ai acquis, à l'issue de cette mission une certaine aisance et une meilleure compréhension du fonctionnement et de l'utilisation de Linux. J'ai également rencontré des difficultés directement liées au code et à l'objectif de la mission. En effet, il était uniquement possible de récupérer les éléments en fonction de leurs attributs. Toutefois, lorsqu'il s'agissait de récupérer tous les éléments d'un fichier XML en ne supprimant que la racine, à savoir les éléments d'identification du fichier pour ne garder que son contenu, j'ai rencontré des difficultés avec la version du module que j'ai importé (xml.Etree.ElementTree) qui ne permettait pas de désigner « tous les éléments étant des objets et leur descendants », fonction qu'on trouvait dans un module plus récent, lxml, avec « iterdescendants ».

Discussion avec ma tutrice de stage et détermination d'une solution à appliquer

Lorsque j'ai évoqué mes difficultés à Claudine, elle m'a proposé l'utilisation d'un nouveau langage, le langage Bash. Mon programme était alors fonctionnel jusqu'à un certain point : il prenait un fichier XML en argument et renvoyait un nouveau fichier XML avec seulement les éléments voulus, identifié selon leur attribut « class-name ». Le problème était la concaténation avec les fichiers de configurations Radios et Téléphones, où il fallait garder tout le contenu sauf la racine. Elle m'a alors proposé de réaliser ces étapes avec un script Bash, puis depuis ce même script Bash, de réaliser mon programme python et de récupérer le programme renvoyé, puis dans le script de concaténer les 3 fichiers avant de les renvoyer. Je n'avais jamais codé en Bash alors j'ai dû apprendre, cela a été à la fois angoissant de devoir apprendre quelque chose qui m'était nouveau et dans un cadre sérieux et professionnel, mais cela m'a beaucoup plu de pouvoir suivre mes progrès à travers la correction des différentes erreurs de code puis la résolution du problème que j'avais rencontré.

Test de la première version du programme et détermination des modifications et corrections à apporter à partir des résultats obtenus

Après la réalisation de cette première mission, nous sommes allés avec ma maîtresse de stage directement en plateforme afin de tester la configuration produite. C'était l'occasion pour elle d'introduire la deuxième étape de cette première mission, à savoir l'optimisation de l'utilisation de mon programme et la modification du fichier XML pour répondre aux normes établies dans l'entreprise.

Elle m'a alors indiqué les modifications, améliorations et corrections à apporter à mon programme. Tout d'abord, pour pouvoir réellement utiliser mon programme et plus précisément le fichier XML qui créait, il fallait pouvoir adapter l'affichage de la racine du fichier aux informations saisies par l'utilisateur comme la description de la configuration ainsi générée, son

nom, sa version, son Trigramme (ensemble de 3 lettres permettant d'identifier le site concerné), mais aussi permettre, lors de la création du fichier, d'intégrer directement à la racine la date du jour afin de pouvoir mieux référencer la création de la configuration. Il s'agissait également de travailler sur la gestion d'erreurs, afin qu'elles soient plus explicites. En effet, la gestion d'erreur est un aspect que j'ai pu être amené à travailler, mais que je n'ai pas intégré comme une reflexe lors de la réalisation de mes programmes. Ainsi, le premier test effectué par Claudine a été de lancer le programme depuis le terminal sans aucun argument, ce qui a renvoyé une erreur « List Index out of range », ce qui, pour un utilisateur du programme ne codant pas en python, ne veut pas dire grand-chose. Il fallait alors revoir mon programme pour intégrer des affichages adaptés en fonctions des possibles erreurs levées. On privilégiera alors un affichage « Il faut fournir 3 arguments : le nom du fichier d'abord, puis le nom voulu pour la nouvelle configuration de travail, et enfin le Trigramme. ».

Corrections et modifications du programme

J'ai d'abord commencé par cette partie, avant de m'attaquer à la modification de la racine, afin d'essayer d'intégrer la gestion d'erreur et les tests à des réflexes de travail, sans avoir à repasser sur mon code dans un second temps pour les effectuer, ce qui peut par ailleurs le déstructurer.

Après cela, je suis passé à la modification de la racine qui était auparavant définie en dur, qu'il fallait maintenant modifier afin qu'elle devienne dynamique. Pour réaliser cette étape, je me suis replongé dans la documentation du module que j'avais précédemment utilisé pour le parcours, à savoir xml.Etree.ElementTree, mais cette fois, il me fallait trouver une solution de création ou de modification de fichier XML afin de pouvoir agir directement sur la racine du fichier XML que je créais. J'ai alors utilisé les méthodes xml.Etree.ElementTree.Element(attribut=variable) ainsi que xml.Etree.ElementTree.SubElement(Element, attribut=variable). La première crée un élément, un nœud parent en quelque sorte, et la deuxième crée une feuille, ou un enfant. Avec ces deux méthodes, je pouvais créer n'importe quel attribut et n'importe quel enfant, j'ai alors pu créer une racine principale avec les informations d'identification de la configuration de travail, puis avec la méthode SubElement, spécifier dans les enfants de cette racine d'autres informations comme la date, la description, le Trigramme, etc.

Grâce à ces deux étapes, le programme était bien plus adapté à une utilisation dans de nombreux cas, par des personnes de n'importe quelle équipe. Toutefois, une dernière étape était nécessaire, sur des points observés après utilisation du programme lors des différents tests effectués. Ainsi, il allait falloir permettre de choisir les classes à garder dans le nouveau fichier XML créé directement dans le terminal, au moment de lancer le fichier. Auparavant en effet, cela était défini en dur au début du programme python, ce qui permettait de modifier les classes à garder assez facilement, mais impliquait tout de même d'ouvrir le programme et de savoir un minimum s'y repérer. Il m'a donc été demandé d'ajouter un nouvel argument : un fichier .txt contenant toutes les classes que l'utilisateur souhaitait garder, lui permettant de seulement créer un fichier texte qu'il pourra directement utiliser avec le programme. Pour inclure ce fameux fichier texte en argument, j'ai de nouveau utilisé le module « sys », mais en ajoutant une ligne récupérant le nom du fichier à ouvrir encore une fois en fonction de son index (index 5, 1 étant le nom du programme, 2 le nom du fichier XML de référence, 3 le nom du fichier XML à créer, 4 le Trigramme, et 5 le fichier .txt à ouvrir). Cependant, il me fallait également inclure un traitement de fichier afin de pouvoir ouvrir puis parcourir et lire ce le fichier .txt. Pour cela, j'ai utilisé deux fonctions natives à python, open() et split(). La première ouvre simplement le fichier texte, et la deuxième va le parcourir et récupérer tous les éléments selon un séparateur à préciser. Dans mon cas, j'ai choisi le séparateur « , » , mais j'aurais tout aussi pu choisir « » ou

« ; ».

Test de la deuxième version puis finalisation du programme

Une fois que toutes ces étapes ont été réalisées, tests inclus, j'ai présenté le résultat final, et j'ai commencé à travailler sur la finalité de mon programme : sa documentation. Toutefois, Claudine m'a fait remarquer que pour certains fichiers texte, une classe, et spécifiquement la dernière, n'était pas traitée, c'est-à-dire qu'elle se trouvait dans le fichier XML de référence mais que les objets ayant les class-name correspondant à cette dernière classe ne se trouvaient pas dans le nouveau fichier crée. C'était assez étrange car ce cas de figure arrivait seulement avec certains fichiers .txt, d'autres fonctionnant parfaitement. Par ailleurs, les fichiers texte fonctionnant et ceux ne fonctionnant pas n'avaient aucune différence particulière. J'ai remarqué d'où venait le problème après plusieurs tests, mais celui qui m'a permis de l'identifier clairement a été d'afficher les différentes étapes de récupération des classes lors du split(). J'ai alors remarqué que la dernière classe se terminait par \n, trahissant un retour à la ligne. Ainsi, lors du parcours du fichier XML, aucune classe n'était de la forme « class-name\n », et donc elle n'était pas récupérée. J'ai fait le choix de simplement modifier les fichiers .txt de test, et de spécifier directement dans la documentation qu'il fallait faire attention à ce qu'il n'y ai pas de retour à la ligne à la fin du fichier .txt en argument.

Après ce problème résolu, je suis alors passé pour de bon à la dernière étape : la rédaction de la documentation. C'est une étape qui n'était pas particulièrement compliquée, mais qui demandait beaucoup de rigueur. C'était la première fois que je rédigeais une documentation alors je n'étais pas familier avec la syntaxe et l'organisation à adopter, c'est donc cet apprentissage qui m'a posé le plus de problèmes. Il fallait également être attentif quant au fait d'être très précis dans les instructions pour que le code fonctionne : version de python, format des fichiers, spécificités... En effet, il faut que ma documentation permette, si elle est suivie correctement, de faire fonctionner le programme dans 100% des cas, où dans une situation où la personne l'utilisant rencontre une erreur, il faut qu'elle puisse la résoudre en allant voir la documentation et en lisant les instructions et conditions à respecter. Après cela, mon programme ainsi que la documentation ont pu tous les deux être mis en Conf, ce qui signifie utiliser git pour les push sur un dépôt distant commun aux personnes travaillant sur le projet SRSA.

Cette première mission m'a beaucoup plu. Elle m'a permis de mieux comprendre la structure d'un fichier XML et leur utilisation dans CMoIP. J'ai également découvert les problèmes potentiels liés à leur utilisation, que ce soit en tant que configurations de travail ou dans des cas plus généraux, comme la récupération d'informations, par exemple.

C. *Troisième mission : Comparaison de deux fichiers XML et affichage adaptatif*

Introduction à la mission

Contexte et objectif de la mission

Ma nouvelle mission m'a conduit à intégrer l'équipe du projet CLA.

Au sein de ce nouveau projet, ma nouvelle mission était de faire de la comparaison de deux fichiers XML. Le problème était en effet le suivant : lors d'une montée en version, de nombreux changements sont fait dans les fichiers XML, que ce soit de la création d'objet, suppression d'objet, changement de valeur, ajout d'attributs, etc... Toutefois, le fichier est si dense qu'il est impossible de savoir précisément ce qui a été changé. Il faut alors que je crée un programme qui prends en argument un fichier récent, et un fichier ancien, qu'il les compare intelligemment, ligne par ligne, en utilisant le fait que les deux fichiers sont triés par ordre alphabétique selon cet ordre : class-name → nom → valeurs.

Confrontation à de nombreux défis liés aux difficultés de la mission

Difficulté d'implémentation d'un parcours intelligent et adaptatif

Très vite, je me suis rendu compte des nombreux défis auxquels j'allais être confronté : tout d'abord, le principe de « parcours intelligent » ligne par ligne posait un problème. En effet, je me suis vite rendu compte qu'une simple comparaison n'allait pas suffire, car au moindre ajout ou retrait, le reste des éléments allaient se retrouver décalés d'un fichier à l'autre, et donc tout le reste du fichier allaient présenter des différences alors que ce n'est pas forcément le cas. Il fallait donc observer les éléments n+1 des deux côtés et dans les deux sens afin d'une part de mieux adapter l'affichage, au lieu de seulement afficher une différence, on pourra détecter un retrait, ajout, ou simplement un changement, et d'autre part, cela laissait la possibilité de recalibrer le parcours à la détection d'un ajout ou retrait pour ne pas condamner le reste du parcours. J'ai alors choisi de créer une valeur d'écart, ainsi, si l'élément à l'indice n du nouveau fichier correspondait à l'élément à l'indice n+1 de l'ancien, on retire 1 à la variable correspondant à l'écart du nouveau fichier, afin qu'il n'avance pas et rattrape l'ancien fichier. Le reste du programme affichera alors que les deux éléments sont identiques. En suivant cette logique et en l'appliquant à chaque cas, on se retrouve avec un parcours plus adapté, qui peut mieux détecter les « vraies » différences, avant de se recalibrer pour les prochains éléments. J'ai toutefois été mis en difficulté par le grand nombre de cas à prendre en compte, afin de détecter ajout, retrait, modifications, de nombreux tests étaient nécessaires, et parfois certaines différences n'étaient pas détectées à cause d'un cas qui n'avait pas été pris en compte dans le test.

Problèmes liés à la gestion d'index

La deuxième difficulté à laquelle j'ai été confronté a été les problèmes d'index. Il fallait en effet pouvoir gérer le fait qu'un fichier soit plus court que l'autre (sauf si les deux fichiers sont tout à fait identiques). J'ai donc choisi de créer un nouveau test afin de voir si la longueur des éléments que je récupérais à la comparaison était inférieure ou égale à la valeur n du parcours + celle de l'écart (et ce dans les deux sens) auquel cas j'instanciais l'index de comparaison du fichier arrivé à sa fin (selon la boucle activée dans la comparaison) à -1, pour toujours comparer le dernier élément, et donc éviter toute erreur d'index.

Problème de coût en temps et en mémoire induit par un parcours complexe

La troisième difficulté concernait le parcours en lui-même. En effet, il m'a été demandé de comparer selon 3 attributs. Tout d'abord, comparer les différents class-name du premier niveau, ainsi que les noms puis valeurs du deuxième niveau. Afin d'optimiser mon code en coût en

mémoire et en coût en temps, j'ai choisi de ne comparer les valeurs et noms des sous objets que si les class-names étaient identiques. En effet, si le class-name n'est pas le même, inutile de comparer le reste puisque si le parent est différent, l'enfant le sera forcément (puisqu'une différence ne peut venir que d'un ajout ou d'un retrait, et que dans les deux cas il ne sera pas utile de comparer l'enfant).

Mise en place de solutions

Structuration du programme et découpage du code

Ainsi, j'ai finalement adopté un découpage de mon code en 3 parties, avec 3 fonctions, une par comparaison. J'avais donc une fonction de comparaison par class-name, une fonction de comparaisons en fonction des noms, et une fonction de comparaison en fonction des valeurs. Cela permettait notamment une récupération plus efficace des données nécessaire. On parcourait les fichiers en début de fonction, et on récupérait respectivement les class-name, noms, et valeurs, avec la méthode .get(). Chaque élément est alors ajouté à une liste, qui sera ensuite parcourue afin de les comparer. J'ai choisi de faire un parcours par indice dépendant de la valeur maximale de la longueur des deux listes ainsi crées (une liste des class-name du fichier récent, et une liste des class-name du fichier ancien), sinon il était impossible de faire une comparaison en simultané des deux fichiers simplement avec une boucle for. Si les deux éléments sont différents, on l'affiche, sinon, s'ils sont identiques, on l'affiche aussi, puis on lance la fonction de comparaison par nom, avec en argument, la class-name correspondante (qui a pu être récupéré puisqu'on effectue directement la ligne de code dans la boucle concernée). Le programme de comparaison par nom va donc avoir accès au class-name afin de récupérer tous les noms des sous-objets de l'objet ayant le class-name ainsi spécifié. La même opération est ensuite répétée avec la comparaison par valeur, qui encore une fois, n'est effectuée que si les noms sont identiques, dans le cas contraire il n'y aura pas d'affichage.

Ce découpage du code en trois parties permet également d'éviter un problème de racine. Pour parcourir le fichier XML et récupérer certains attributs avec la méthode .get(), il est nécessaire de spécifier la racine. Cette racine correspond à un certain niveau dans la hiérarchie du fichier. Par exemple, pour les class-name, on se trouve au premier niveau, tandis que pour les noms et valeurs, on se trouve au deuxième niveau, ce qui implique une racine différente. Dans cette version, chaque racine est spécifiée au début du code, sans avoir besoin de détecter l'élément à comparer et d'adapter ainsi la racine utilisée. Une première version de mon code, qui ne séparait pas les différentes fonctions du programme en plusieurs fonctions Python, nécessitait la création de plusieurs racines et donc plusieurs parcours. Cela a grandement impacté le cinquième et dernier problème que j'ai rencontré, le plus problématique : le coût en temps.

Correction des problèmes liés au temps d'exécution du programme trop important

En effet, après avoir réglé les problèmes précédemment énoncés, il en restait un, et de taille, le temps d'exécution du code. Lors d'un premier retour sur mon programme, c'est l'un des deux commentaires principaux qui m'ont été fait, le temps d'exécution était bien trop long pour que le programme soit optionnel (l'autre commentaire concernait l'affichage, j'en parlerais ensuite). Il fallait le réduire considérablement. Chaque exécution sur des fichiers d'environ 50.000 lignes prenait plus de 6 minutes à exécuter (temps d'exécution récupéré grâce au module time). Afin de réduire ce temps, j'ai dû employer plusieurs méthodes. Premièrement, modifier l'ordre de comparaisons afin de pouvoir en éviter certaines. Ensuite, supprimer certains affichages

inutiles, qui avaient été laissés pour pouvoir mieux détecter d'où venait telle ou telle erreur. Cependant, mon erreur initiale, et ce qui m'a finalement permis de gagner le plus de temps, a été de revoir la gestion des variables liées au traitement des fichiers XML, à la récupération des racines, et au parcours de ces racines, etc. Initialement, chacune de ces opérations — ouverture des fichiers, récupération des racines, parcours pour récupérer les attributs concernés — était effectuée au début de chaque fonction. Cela impliquait que, à chaque appel de fonction, ces opérations devaient être répétées. Dans un fichier de 50 000 lignes, le nombre de comparaisons devient très élevé, entraînant la répétition de ces opérations un très grand nombre de fois. Pour éviter cela, il suffisait d'ouvrir les fichiers XML, de récupérer les racines et de les parcourir au début du programme. Ensuite, il suffisait d'ajouter chacune des racines ainsi créées en argument de chaque fonction. Au début de chaque fonction, il ne restait plus qu'à parcourir les racines pour récupérer les attributs correspondants. Après toute ces étapes effectuées, le temps d'exécution du programme passait de 6 minutes, à 30 secondes.

Corrections liées à l'affichage

Ce problème étant réglé, un deuxième point m'avait été fait remarquer, et il concernait l'affichage. Lors des différentes étapes de la réalisation de mon code, j'ai adapté l'affichage pour obtenir un maximum d'informations en cas d'erreurs. Cela me permettait de localiser précisément l'origine des erreurs et de vérifier que certaines parties de mon programme fonctionnaient correctement. Par exemple, lorsque la fin d'un fichier est atteinte, le programme passe dans une autre boucle pour éviter les erreurs d'index. J'ai donc inclus un message d'affichage pour confirmer que le programme entre bien dans cette boucle au moment approprié. Si le programme se terminait sans afficher ce message, je savais qu'il y avait un problème dans son exécution, et je pouvais alors le corriger en conséquence. Cependant, cela gênait l'affichage final lors de l'utilisation concrète de mon programme. Il n'était en effet pas pertinent de connaître le type d'erreur levée (chaque type d'erreur étant identifié par une appellation spécifique) ou de savoir si un fichier arrivait à sa fin. J'ai donc supprimé ces messages de "test" et adapté les autres afin de n'afficher que les erreurs, en fournissant les informations nécessaires, notamment les attributs qui différaient. Enfin, il m'a été suggéré de mieux faire apparaître la logique du code et le déroulement du programme : d'abord la comparaison d'un objet, puis, si les deux objets sont identiques, la comparaison de leurs enfants, et enfin, l'affichage des différences si les attributs "nom" ou "val" de ces enfants diffèrent.

Apprentissages et impacts sur ma manière de travailler

Toute ces étapes m'ont fait comprendre que l'organisation du code est primordiale. Il me fallait effectivement revoir l'ordre des opérations à effectuer. Il fallait par exemple afficher les messages avant de mettre à jour les différentes variables (notamment les variables d'écart), sinon les erreurs levées seraient incorrectes. La séparation de mon code en différentes fonction notamment m'a beaucoup aidé à adapter l'affichage, et plus particulièrement lors du travail sur l'affichage permettant un meilleur suivi du déroulement de mon programme. En effet, avant chaque appel de telle ou telle fonction – si c'était le cas –, je pouvais récupérer certaines informations propres à l'objet à l'origine du parcours qui allait être effectué, afin de permettre un meilleur affichage d'éventuelles erreurs.

Conclusion de la mission

Après la correction de l'affichage, mon programme a été validé par l'équipe CLA et il a pu être intégré au projet. Cette mission était très difficile à cause des nombreux défis que j'ai dû relever, mais il m'a également permis de développer de meilleurs réflexes en tant que développeur, notamment en intégrant une meilleure réflexion sur l'organisation de mon code, mais aussi en me permettant d'adopter certains réflexes et outils lors de l'apprentissage d'une notion nouvelle. En particulier, j'ai appris l'importance de structurer mon code de manière modulaire et de bien documenter chaque étape du processus de développement. Cela m'a aidé à identifier rapidement les sources d'erreurs et à corriger les bugs de manière plus systématique. De plus, cette expérience m'a sensibilisé à l'importance de la communication et de la collaboration au sein d'une équipe de développement, renforçant ainsi mes compétences en travail d'équipe.

En somme, cette mission a été une opportunité précieuse pour renforcer mes compétences techniques et méthodologiques, et m'a permis de me sentir mieux préparé pour affronter des projets futurs avec une approche plus structurée et réfléchie.

D. Quatrième mission : Traitement, mise à jour et affichage de fichiers YAML

Introduction à la mission

Contexte et objectif de la mission

Toujours au sein du projet CLA, une autre mission m'a été confiée, cette fois-ci au sein d'un projet nécessitant un travail non pas avec des fichiers XML, mais avec des fichiers YAML, une notion de programmation qui m'était alors inconnue.

Dans ce projet, ma nouvelle mission concernait donc du traitement de fichiers YAML et de dictionnaires. Un fichier YAML est composé d'un ensemble de couples de clés / valeurs, tout comme dans un dictionnaire. Cependant, dans les fichiers qui me sont fournis, certaines clés fonctionnent différemment des autres clés. Ce sont les clés « __Inclure__ », « __inclure__ », « __Include __ », ou « __include__ », dont les valeurs sont des dictionnaires eux même composés de deux éléments. Le premier est un couple clé : « fichier » et valeur le nom de ce fichier, et l'autre élément est un couple clé : « cle », et valeur une chaîne de caractères, par exemple, « C3650-98TS ».

Enjeux du code

Il est donc nécessaire que, pour chaque Include, le programme parcoure le fichier spécifié (indiqué par « fichier » : nom du fichier) et effectue un merge des deux fichiers YAML (le fichier spécifié doit être un fichier YAML). Les éléments présents sous la clé correspondant à la chaîne de caractères du dictionnaire doivent être intégrés au fichier YAML d'origine. Cependant, certaines règles s'appliquent.

Tout d'abord, chaque clé Include doit être traitée, de manière à ce que chaque niveau soit parcouru. Si la valeur d'une clé dans le fichier d'origine est un dictionnaire, il faut parcourir ce dictionnaire pour vérifier s'il contient des éléments Include ou non. Il est donc nécessaire de parcourir récursivement le fichier YAML à la recherche de chaque Include, et d'exécuter une fonction gestionInclude à chaque itération de Include.

Cette approche garantit que tous les niveaux du fichier sont correctement analysés et que chaque occurrence de Include est traitée de manière appropriée. En parcourant récursivement le fichier YAML, on s'assure que tous les Include sont détectés et fusionnés selon les règles définies. Ensuite, étant donné que les fichiers YAML sont traduits en dictionnaires par le programme, chaque clé doit être unique. Si, dans le dictionnaire à inclure, obtenu à partir de l'élément correspondant à la chaîne de caractères du fichier à inclure, une clé est déjà présente dans le fichier d'origine, c'est ce dernier qui a la priorité. Par conséquent, les valeurs du dictionnaire de l'autre fichier seront écrasées. En revanche, pour chaque clé "nouvelle" du fichier à inclure, le couple clé-valeur doit être ajouté au fichier d'origine.

Ainsi, le processus garantit que le fichier d'origine conserve ses valeurs prioritaires tout en intégrant les nouvelles informations des fichiers inclus.

Différences avec les projets précédents et conséquences sur le déroulé de la mission

Toutefois, cette fois ci, ma mission est un petit peu différent. En effet, dans les deux cas précédents, je devais créer un nouveau programme, ce qui me laissait une certaine liberté dans ma manière de coder, de structurer mon code, dans la logique de mon programme, et dans ma logique pour résoudre le problème qui m'est posé. Mais cette fois ci, un travail a déjà été réalisé. En effet, on me fournit un code que je dois modifier afin d'intégrer les modifications demandées, pour qu'au rendu, le programme comporte les deux fonctionnalités expliquées auparavant. Le code qu'on me fournit s'occupe déjà du traitement du fichier YAML ainsi que de la gestion des __Include__. Cependant, le code comporte plusieurs points que l'équipe de CLA souhaite améliorer pour réellement obtenir un programme fonctionnel et utilisable dans de nombreuses situations. Tout d'abord, pour effectuer un merge des deux fichiers YAML (et donc des deux dictionnaires) lors d'une instance de __Include__, le programme utilise la fonction .update(). Cependant, cette fonction donne une priorité absolue d'un dictionnaire sur l'autre. Imaginons que dico1 soit égal à {1 : 2, 3 : 4} et que dico2 soit égal à {1 : 3, 3 : 9, 6 : 10}, on voudrait obtenir le résultat suivant après le .update() : {1 : 2, 3 : 4, 6 : 10}, mais le update renverra {1 : 2, 3 : 4}. Une de mes missions consistera donc à faire en sorte de créer un update « intelligent » qui saura garder la priorité de dico1 tout en incluant les éléments nouveaux de dico2. Pour le deuxième problème, le programme intègre bel et bien une logique récursive pour parcourir chaque dictionnaire et détecter les __Include__. Cependant, avec une telle logique, si deux Include se trouvent au même niveau (exemple : __Include__ : {fichier : exemple.yaml, cle : C89}, xpos : 4.87, ypos : 0.78, __inclure__ : {fichier : exemple.yaml, cle : N90}), alors le deuxième __Include__ ne sera pas détecté, et donc pas traité.

Ma mission se concentre donc premièrement sur ces deux points : l'intégration d'un update « intelligent » s'adaptant au contenu des dictionnaires au lieu d'utiliser une logique de priorité absolue, et la création d'une nouvelle boucle pour s'assurer du traitement de chaque __Include__, même au même niveau.

Première étape : compréhension du programme d'origine

La première étape a été la compréhension du programme. Je me suis alors rendu compte qu'il était très difficile de comprendre le code de quelqu'un d'autre, même lorsque celui-ci est commenté et bien structuré. C'était en effet un programme assez complexe avec beaucoup de fonctions, il y avait en effet une grande utilisation du principe « Diviser pour régner » avec 1 fonction = 1 tâche. Il y avait également une gestion d'erreurs très développée avec de nombreux

affichages très précis sur la cause et la source de l'erreur. J'ai eu la chance de pouvoir discuter avec les développeurs du programme et leur poser des questions jusqu'à ce que ma compréhension de ce dernier soit suffisante pour m'attaquer au fichier YAML. Il fallait en effet que je comprenne comment cette ce langage de programmation fonctionne. Ces deux étapes peuvent souvent être négligées, il est souvent plus naturel de tout de suite se mettre à essayer de coder, mais mes deux missions précédentes notamment m'ont appris à adopter une méthode de travail organisée et rigoureuse. Une fois que j'avais développé une bonne connaissance du programme et du fichier YAML, j'ai pu commencer à coder. En effet, comprendre le programme m'a permis de saisir son fonctionnement et d'identifier des pistes pour ajouter des instructions visant à améliorer le parcours et à modifier la mise à jour. De plus, la compréhension des fichiers YAML m'a aidé à travailler efficacement avec ce langage et à déterminer quelles méthodes utiliser pour leur traitement.

Deuxième étape : apport de modifications au programme

Modification de la boucle pour assurer un traitement de tous les éléments

Une fois cette étape réalisée, j'ai décidé de modifier la boucle afin de créer un meilleur parcours, étant capable de détecter, puis de traiter, chacun des instances de __Inclure__. Lors de ma lecture du programme afin d'essayer de mieux le comprendre, j'avais identifié un passage où data (le dictionnaire crée à partir du fichier YAML) est parcouru afin de chercher une instance de __Inclure__. Si une instance est effectivement présente, alors on définit la clé prends comme valeur cette instance, à savoir « __Include__ » par exemple. Si aucune instance n'est trouvée, la clé prends comme valeur None. Pour la suite du programme, il suffit donc d'inclure une condition « if cle is not None » afin de s'assurer que chaque « update » se fasse quand il y a une clé. Ces instructions se trouvent dans la fonction gestionInclude(), qui est une fonction récursive. Elle prend en effet en argument le dictionnaire (issu du fichier YAML) sur lequel effectuer le parcours, et à chaque élément de type dict, donc à chaque dictionnaire, elle sera appelée à nouveau sur le dictionnaire, afin de tester s'il présente ou non des instances de __Inclure__, et ainsi de suite récursivement. J'ai d'abord pensé que le problème venait de l'ordre de détection d'instances de __Include__. En effet, chaque cas était testé l'un après l'autre (if __Include__ elif __include__ elif __Inclure__ elif __inclure__) et donc j'ai pensé que l'entrée dans un cas empêchait les autres d'être détectés. J'ai donc créé une boucle unique de détection intelligente de chaque instance de __Include__ qui prenait en compte les variantes (langue, majuscules/minuscules...). J'ai ensuite rapidement créé mon propre fichier YAML de test, bien moins volumineux, avec seulement deux instances, et où je choisissais moi-même les clés et valeurs ainsi que le contenu des __Include__ à inclure. Cela facilitait le test des différentes fonctions. Ce fichier réduit me permettait de savoir précisément quel résultat je souhaitais obtenir, et le volume moins important de données me permettait d'identifier et de corriger plus facilement les erreurs. J'ai alors pu tester ma première modification, mais le résultat était le même que celui obtenu après l'exécution du programme d'origine (là encore avec mon fichier de test). Mon changement n'avait apporté aucune modification ou correction, la logique n'était pas la bonne. Je me suis alors replongé dans le code d'origine, et notamment le passage de test de présence d'instances de __Include__, puisque je savais que c'était à cet endroit que j'allais devoir apporter une modification. Puis, après réflexion, j'ai compris d'où venait le problème. En réalité, il n'y avait pas de boucle qui intervenait sur le niveau parcouru. En effet, à la fin d'une détection d'instance __Include__ et de son update, la fonction était à nouveau appelée lors de la présence d'un dictionnaire, donc à un niveau inférieur, mais elle n'était pas répétée sur le même niveau. La solution était donc une boucle while, qui bouclerait tant que clé (while cle) donc tant qu'une instance de __Include__ est détectée. Cela permet au programme de ne pas s'arrêter à la

première instance et de continuer tant qu'une clé est détectée. J'ai alors testé une première fois le programme tel quel, mais aucun Include n'était traité, alors que dans le programme d'origine, seul le premier Include était traité. J'avais donc introduit un problème supplémentaire. Cependant, après relecture de mon code, je me suis rendu compte qu'en ajoutant cette condition, à chaque appel de la fonction, la variable clé n'ayant pas de valeur par défaut permettant un premier parcours, aucun parcours n'avait lieu, et donc aucune gestion d'__Include__ ne s'effectuait. Il a donc fallu ajouter la variable clé en argument de la fonction et la définir sur True par défaut pour qu'à chaque appel de celle-ci, le test de présence d'__Include__ soit effectué. Après avoir testé cette version, le problème du parcours avait bel et bien été géré, et chaque Include était traité. Cependant, ils n'étaient pas traité correctement, et ce à cause du .update() mettant en œuvre la logique de « priorité absolue ».

Modification de l'update : gestion des priorités et conservation des éléments

Il fallait donc que je fasse en sorte que l'update se fasse correctement, c'est-à-dire en gardant une priorité du fichier d'origine pour les clés identiques, mais en incluant les clés nouvelles du dictionnaire de l'autre fichier. Pour cela, j'avais 3 possibilités. Je pouvais garder l'utilisation de l'update, et réaliser des opérations avant ou après cette update, ou je pouvais abandonner l'idée de l'update et « créer mon propre update » qui permettait de réaliser précisément ce que je souhaitais. J'ai choisi d'effectuer une opération avant l'update, directement sur les dictionnaires, afin que l'update, qui donne déjà la priorité au premier dictionnaire, permette également de conserver les clés uniques du dictionnaire de l'autre fichier. Tout d'abord, afin de réaliser de gains en temps, j'ai créé une condition de lancement de ma fonction. Cette condition est que les deux dictionnaires, le premier issu du fichier d'origine, et l'autre issu du fichier annexe (clé __Inclure__) soient différents. Ensuite, j'ai choisi de parcourir le dictionnaire issu du fichier lié à la clé __Inclure__. En effet, puisque l'update donne une priorité absolue au premier dictionnaire, il est certain que tous ses couples de clés/valeurs seront conservés. Les opérations à effectuer sont donc liées au dictionnaire de la clé __Inclure__. J'ai donc choisi de parcourir les clés du dictionnaire __Inclure__. Ensuite, pour chaque valeur que prenait la variable associée au parcours, je vérifiais si cette valeur se trouvait dans les clés du premier dictionnaire, à l'aide d'un autre parcours. Cela me permettait de déterminer les clés uniques du dictionnaire __Inclure__. Pour chaque clé unique, j'itérais dans dico.items(), afin de récupérer le couple clé / valeur associé à la clé unique, et je l'ajoutais à une liste vide créé en début de fonction (append). Ensuite, il suffisait d'ajouter ce couple clé valeur au premier dictionnaire, de manière que lors de l'update, les valeurs soient conservées.

Premier test et tirage de conclusions

Toutefois, après test, le résultat n'était pas celui attendu. En effet, les valeurs du deuxième dictionnaire devenaient prioritaires, et seul le __Inclure__ conservait les valeurs d'origine du premier dictionnaire. En relisant mon code, je me suis rendu compte que l'erreur provenait de la multiplicité des parcours et des variables utilisées qui s'annulaient entre elles. À chaque parcours de type « for n in dico », j'utilisais parfois la même variable, tout en ayant besoin de comparer les valeurs associées à ces variables. De plus, l'ordre de mes parcours n'était pas correct et certaines indentations étaient incorrectes, ce qui faussait les parcours, les comparaisons, et l'ajout des couples clés/valeurs à la liste de transition. Je me suis rendu compte qu'il était assez facile de se perdre avec le nombre important de boucles, de comparaisons, de récupérations de valeurs nécessaires, mais aussi avec les indentations et les index à rallonge.

Mise en place de nouvelles méthodes pour répondre à ce nouveau défi

Utilisation du papier crayon

Cela m'a poussé à me tourner d'autant plus vers le papier et le crayon. Auparavant, lors de la réalisation de mes anciens programmes, j'utilisais le papier et le crayon pour mettre mes idées sous forme de tirets. Sans réellement avoir besoin de me référer à ce que j'écrivais, c'était surtout une manière d'organiser mes pensées, de mettre des mots sur les problèmes que je rencontrais, afin de mieux comprendre ma situation et d'identifier les solutions à ma disposition. C'était un moyen pour moi de mettre de l'ordre dans mes idées et, après quelques réflexions faites au papier et au crayon, je pouvais me remettre à coder.

Une pratique qui m'avait été recommandée, mais que je n'aimais pas particulièrement, était de faire un premier « schéma » du code au papier et au crayon avant de réellement commencer à coder. J'étais partisan de l'idée qu'il était bien plus facile de se rendre compte de ce qu'on pouvait et voulait faire en codant directement sur l'ordinateur, mais le problème auquel je me suis heurté cette fois m'a prouvé le contraire. Cela m'a aidé pour chacun des problèmes que j'avais rencontrés. Premièrement, cela me permettait d'avoir une bien meilleure vue d'ensemble de mon programme. Deuxièmement, la création des boucles, leur début, leur fin, ainsi que les indentations qu'elles engendraient et où les terminer a été grandement facilité. Troisièmement, Il était bien plus facile de visualiser quelle variable prenait quelle valeur dans quelle boucle, et donc d'attribuer les bons index aux bons endroits. Enfin, quatrièmement, cela permettait de facilement simuler le fonctionnement du programme, avec le passage dans chaque boucle, le changement de valeur des variables à chaque itération de chaque boucle ainsi que leur valeur lors de l'accès via un index. Tous ces éléments m'ont permis de pouvoir avancer petit à petit, en m'aidant de mon travail sur papier, et pas à pas, à résoudre chacun des problèmes expliqués précédemment.

Modification du programme en accord avec les pistes trouvées

J'ai ainsi pu reprendre mon programme, en intégrant la logique suivante. Si les deux dictionnaires sont différents, alors on parcourt le dictionnaire __Include__. Si une des valeurs est de type dictionnaire, alors on va entrer dans ce dictionnaire afin de parcourir ses clés. En parallèle, on lance un parcours des clés du premier dictionnaire, et pour chaque clé issue de la valeur de type dictionnaire du dictionnaire __Include__ qui ne se trouve pas dans le premier dictionnaire, on ajoute le couple clé / valeur à une liste vide. Fin de la première boucle. J'ai créé ensuite une deuxième boucle, qui ne se lance que si le nombre d'éléments dans la liste contenant les couples de clés / valeurs uniques est supérieur ou égal à 2. En effet, les ajouts d'éléments à la liste se font 2 par 2, une clé et une valeur, il n'y jamais d'ajout d'une clé seule ou d'une valeur seule. Si la liste a une taille inférieure à deux, alors le programme précédent n'est jamais entré dans la boucle d'ajout, la liste étant vide, cela signifie qu'il n'y a aucun élément à ajouter dynamiquement au premier dictionnaire. Dans le cas contraire, on va chercher la clé à l'indice 0 + une variable écart, qu'on incrémentera à chaque fin de parcours et on va tester si cette clé se trouve ou non dans le premier dictionnaire. Si elle ne s'y trouve pas, on va parcourir les clés du dictionnaire se trouvant à l'indice 1 + écart de la liste, c'est à dire la valeur associée à l'élément à l'indice 0 + écart. Pour rappel, ce couple clé valeur représente les dictionnaires (valeur) dont la clé, issue du dictionnaire __Include__, est unique. Ainsi, l'élément parcouru est un dictionnaire associé à une clé d'un dictionnaire, c'est donc un dictionnaire dans un dictionnaire. Dans ce dictionnaire issu lui-même d'un dictionnaire, on va parcourir ses clés.

Cela va permettre de tester si ces clés se trouvent, ou non, dans les clés de l'élément issu du premier dictionnaire, donc du dictionnaire issu du fichier YAML donné en argument, plus spécifiquement en parcourant les clés du dictionnaire appelé par la clé issue de l'élément à l'indice écart + 0 de la liste. Si la clé s'y trouve, alors on garde la même valeur d'origine, puisque le dictionnaire d'origine a la priorité sur le dictionnaire __Include__, cependant, si la clé ne se trouve pas dans le dictionnaire d'origine, on crée l'élément directement dans le dictionnaire d'origine, en récupérant la valeur du dictionnaire __Include__, en appelant directement l'élément par la clé issue de la valeur du parcours des clés du dictionnaire dans le dictionnaire. A la fin de tout cela, on ajoute 2 à la valeur écart, pour passer, lors de la prochaine instance, au prochain couple clé / valeur de la liste.

Exemple

Voici un exemple pour illustrer ce que je viens d'expliquer qui est assez complexe en raison du nombre important d'utilisations de boucles d'index, et des valeurs auxquelles je fais référence. Un fichier YAML est donné en paramètre. A partir de ce fichier YAML, un dictionnaire est créé. On l'appellera dico YAML. Ce dico YAML va être parcouru pour chercher les __Include__. A chaque __Include__ trouvé (cf. premier programme que j'ai créé, explication du premier défi de ma mission sur les fichiers YAML), on va parcourir le fichier YAML appelé par le dictionnaire se trouvant à la valeur de la clé __Include__. On appellera le dictionnaire ainsi obtenu (issu de __Include__) dico __Include__. Ce dictionnaire ainsi obtenu va être parcouru. On va parcourir ses valeurs, et pour chaque valeur étant un dictionnaire, on va tester si la clé issue de la valeur se trouve également dans le dictionnaire d'origine. Si elle ne s'y trouve pas, on ajoute la clé et la valeur (dictionnaire également) à une liste vide qu'on appellera liste_dico. Dans une deuxième partie du code, on va récupérer le premier élément de liste_dico, qui est donc une clé. On va tester si cette clé ne se trouve pas dans le dico YAML (dictionnaire d'origine). Si elle ne s'y trouve effectivement pas, on va parcourir la valeur associée à cette clé (qui pour rappel est un dictionnaire), on parcourt donc le dictionnaire se trouvant au premier indice de la liste dico. Pour chacune des clés de ce dictionnaire ainsi parcouru, on va tester si la clé se trouve également dans l'élément issu du fichier d'origine, appelé par la clé ainsi testée. Si cette clé existe, on garde la même valeur, car la clé issue du dico YAML est prioritaire. Sinon, on remplace la valeur par l'élément du dico __Include__ issu de l'appel de la valeur se trouvant à la clé ainsi testée.

Conséquences des modification apportées

Ainsi, la règle de priorité du premier dictionnaire est respectée, et les valeurs uniques du dictionnaire Include sont ajoutées, permettant de conserver leur valeur. J'ai ensuite effectué de nombreux tests pour m'assurer du bon fonctionnement du programme. Après avoir présenté le résultat à l'équipe CLA, on m'a demandé de transformer ce programme en module. Ce module doit permettre d'utiliser la fonction principale pour afficher le fichier YAML après sa mise à jour en fonction de ses Include. De plus, il m'a été demandé, si possible, de rendre toutes les autres fonctions du programme inaccessibles à l'extérieur du programme lui-même.

Pour cela, j'ai dû faire des recherches sur le fait de rendre « private » des fonctions python. Après m'être renseigné, j'ai découvert qu'il était impossible à proprement parler de rendre une fonction private en dehors d'une classe. Toutefois, il était possible de rendre son accès plus difficile. En effet, en ajoutant « __ » au début de certaines fonctions, celles-ci n'apparaissaient plus lors d'un « dir » ou d'un « help ». Il était toujours possible de « brute force » leur utilisation en les appelant par leur nom directement, mais sans accès au code, impossible de deviner leur

nom. Pour ce qui est du changement du programme en module, j'ai simplement modifié le « main » afin d'en faire une fonction. Lors de l'import du programme, on importera directement cette fonction. J'ai ajouté un argument pour préciser le nom du fichier à traiter, qui était défini en dur auparavant. Après un dernier test, j'ai pu rendre le rendu final à l'équipe CLA, qui a validé mon projet.

Présentation puis prise en main et utilisation de la plateforme GAIA

Après cela, on m'a présenté GAIA, un outil WEB basée sur le logiciel open source Tuleap et utilsé par CS Group. GAIA permet de créer des tableaux de suivi, par exemple pour gérer les FT dont on a déjà parlé, ainsi des dépôts Git distants et de déterminer les droits d'accès de chaque membre de l'équipe selon les souhaits du créateur du dépôt. Son fonctionnement est similaire à GitHub, mais au sein du réseau CS, et donc inaccessible de l'extérieur. On m'a donc créé un dépôt distant pour que je puisse y déposer tous mes projets, au lieu de passer par un dossier "Echange" visible par toutes les personnes de l'entreprise, ce qui peut rendre la récupération de fichiers longue et pénible en raison du nombre important de dossiers existants. L'intégration de mes programmes des différentes missions à ce dépôt distant permettrait de les récupérer directement depuis le terminal, simplement avec un "git pull". Pour pouvoir utiliser GAIA et coupler le dépôt distant avec un "git clone", il fallait que je crée une clé SSH et que je l'intègre à GAIA. Sans cela, chaque opération sur mon dépôt distant nécessiterait de renseigner mes identifiants CS fournis par l'entreprise, ce qui causerait une grande perte de temps. Il m'a également été conseillé de créer cette clé SSH pour me familiariser avec cette notion et son utilité. J'ai donc suivi plusieurs guides sur internet pour créer la clé SSH, l'ajouter à ssh-agent, puis la lier à mon compte GAIA. J'ai ensuite pu créer un dépôt local, cloner le dépôt distant « git clone », ajouter tous les projets sur lesquels j'avais travaillé jusque-là « git add », puis commiter tous ces changements « git commit » et enfin effectuer un push « git push ». Mes fichiers étaient maintenant visibles et disponibles sur le dépôt distant, et un simple « git pull » permettait de les récupérer.

Conséquences de la mission

Cette mission a été très difficile. En effet, elle a tout d'abord introduit un nouveau format de données et langage de programmation : le langage YAML. Dans le cadre du traitement de ces fichiers, j'ai découvert la difficulté de comprendre le programme de quelqu'un d'autre, et par la même occasion, la nécessité de garder à l'esprit de rendre notre programme le plus compréhensible possible pour autrui, notamment dans sa structure, sa logique de fonctionnement, et en utilisant efficacement les commentaires. Cette mission m'a également permis d'adopter le réflexe du papier crayon, afin d'avoir une meilleure vue d'ensemble sur le projet, ce qui peut permettre de nombreux problèmes d'organisation ou de logique du code. Enfin, cela m'a également introduit à l'utilisation de Git au sein d'une entreprise, ce qui m'a permis d'avoir une meilleure compréhension de la logique et de l'utilité de l'utilisation d'un dépôt local et d'un dépôt distant.

E. Cinquième mission : Parcours et affichage personnalisé pour modèles graphiques

Introduction à la mission

Contexte et objectif de la mission

Cette dernière mission était plus générale, car elle visait à être utile à toute personne utilisant des fichiers XML, en particulier celles qui manipulent des fichiers XML provenant de modèles graphiques. Elle allait donc pouvoir concerner plusieurs équipes de différents projets, selon les missions sur lesquelles ces équipes travaillent.

Dans ce projet, ma nouvelle mission allait à nouveau concerner du traitement de fichiers XML, mais cette fois ci ce allaient être des fichiers XML de modèles graphiques, ce qui impliquait certaines différences par rapport aux fichiers XML de configurations de travail sur lesquels j'ai travaillé jusqu'à maintenant. Ces différentes concernaient l'ajout de certains scripts au sein même des fichiers XML. L'inclusion de ces scripts sont délimités par des symbole « % », et permettent de boucler autant de fois que nécessaire, afin de créer plusieurs objets, et de permettre de changer leurs attributs de manière dynamique, avec une incrémentation des valeurs de ces attributs à chaque itération de la boucle. De manière pratique, cela n'empêche pas l'utilisation des méthodes et modules que j'ai pu utiliser jusqu'à maintenant, mais cela implique de compléter le code afin de prendre en compte ces scripts, pour d'éviter des erreurs lors du parcours, puisque le programme ne cherche que des objets XML. Pour ce qui est de la mission en elle-même, il allait falloir que je crée un module constitué de fonctions d'affichages personnalisé. Je devais ainsi proposer moi-même un mode de fonctionnement pour les différentes fonctions, ainsi que différents parcours possibles. Le but étant d'envisager les différents besoins que pourraient avoir les personnes qui doivent utiliser des fichiers XML issus de modèles graphiques.

Différences avec les missions précédentes

Ainsi, cette mission était assez différente des précédentes : la difficulté principale n'était pas dans le codage en soi, mais plutôt dans le fait de se mettre à la place d'un utilisateur de ce genre de fichiers XML, afin d'essayer d'anticiper ce dont il pourrait avoir besoin, et de penser à des méthodes à implémenter qui pourraient être utiles. Pour ce qui est des défis concernant le code lui-même, la plupart des méthodes de parcours et d'affichages utilisés pour les précédentes missions peuvent être réimplémenté, le seul défi concerne donc le traitement des scripts dans les fichiers XML, même si un affichage seul laisse la possibilité de seulement les ignorer sans les traiter, puisque l'affichage final tiendra compte des objets crée à l'issue de toutes les itérations de toute les boucles de tous les scripts.

Déroulement de la mission

Début de la réflexion avec papier et crayon

J'ai ainsi débuté cette mission sur papier, en essayant de noter sous forme de tirets toutes les possibilités que j'avais en termes de parcours, toutes les solutions à implémenter qui pourraient se révéler utiles, voire certaines qui pourraient être considérées comme nécessaire. J'ai ainsi pu déterminer certains types de parcours à implémenter, ce qui m'a permis de mieux pouvoir envisager comment j'allais implémenter mes fonctions. En effet, la liberté étant totale, plusieurs possibilités s'offraient à moi, surtout pour ce qui est des arguments de la fonction, c'est-à-dire des éléments que l'utilisateur de la fonction va devoir saisir afin d'adapter le parcours à ce qu'il souhaite, l'objectif étant de laisser suffisamment d'arguments afin de permettre à l'utilisateur de

pouvoir adapter grandement le parcours à la situation face à laquelle il est, tout en évitant de créer des arguments à rallonge, qui rendraient l'utilisation du programme plus difficile et moins pratiques, le but de la création de ce programme étant initialement de faciliter le travail sur les fichiers XML des modèles graphiques.

Décision de l'approche à adopter

Après réflexion, j'ai décidé de travailler avec une approche qui comprends 3 arguments principaux (en plus du nom du fichier ainsi que du nom du programme) que l'utilisateur pourra saisir afin de modifier un maximum le parcours selon ses besoins. Le premier argument est l'attribut, qui correspond à l'attribut souhaité. Un exemple de valeur pour cet argument serait donc « class-name », ou encore « nom ». Le deuxième argument correspond à la valeur de cet attribut souhaité. Un exemple de valeur serait « switch&routeurs » par exemple. Enfin, le 3ème et dernier argument correspond au paramètre. Le paramètre correspond à une lettre que l'utilisateur va saisir, afin de spécifier un type de parcours. La suite de ma réflexion a donc ensuite porté sur les différents paramètres que j'allais pouvoir implémenter, afin de traiter des parcours utiles ou nécessaires. Ainsi, j'ai pu dans un premier temps déterminer les paramètres suivants : le paramètre -e, qui va permettre de récupérer tous les objets dont la valeur de l'attribut recherché est parfaitement égale à la valeur de l'attribut de l'objet lors du parcours de ce dernier. De ce fait, si on recherche la class-name « cmoIP.mg2S », la class-name « cmoIP » ne fonctionnera pas. Il y a également le paramètre -c, qui permet de récupérer tous les objets dont la valeur de l'attribut est contenue dans la valeur de l'attribut chercher. Ainsi, si on choisit comme valeur d'attribut class-name « switch+routeurs », et que la valeur de l'attribut class-name de l'objet parcouru est « switch », celui-ci va être compris dans l'affichage. J'ai également pensé au paramètre -1, qui permettrait d'afficher tous les descendants des objets affichés, selon le parcours effectué en fonction d'un autre paramètre. Concrètement, le mot clé -1 peut être ajouté après un autre type de parcours déjà existant, comme -e ou -c, ce qui permet d'afficher non pas les objets qui correspondent aux conditions ainsi énoncées, mais leurs descendants. Enfin, pour les paramètres de l'attribut, et de la valeur de l'attribut, j'ai pensé à inclure la possibilité de renseigner « any », ce qui permettrait de sélectionner tous les éléments. Ainsi, un « any » pour l'argument de l'attribut sélectionnerait n'importe quel attribut dont la valeur est égale (-e), ou contenue (-c) dans l'argument de la valeur de l'attribut, tandis qu'un « any » pour l'argument de la valeur de l'attribut afficherait tous les éléments ayant un attribut spécifié dans l'argument attribut.

Réflexion puis décision de la structure du code à adopter et donc de l'organisation des différentes fonctions à implémenter

Je suis ensuite passé à l'organisation et la structure de mon code. En effet, j'ai pensé que pour ce projet tout particulièrement, il était important de travailler sur l'évolutivité de mon projet, pour une raison principale : le fait que la réalisation de ce programme émanait d'un souhait des personnes travaillant sur des fichiers XML issus de modèles graphiques d'obtenir un outil de facilitation de leur travail grâce à des fonction de parcours personnalisées, et donc que ces fonctions souhaitées sont amenées à changer au fil du temps et au fil des projet, puisque de nouveaux besoins apparaitront en fonction des projets. J'ai donc pensé qu'il était primordial de coder mon module de manière à permettre une implémentation facilitée de nouvelles fonctions de parcours, selon les besoins et les envies futurs. Afin de permettre cela : deux choses étaient nécessaires. Tout d'abord, il fallait être tout particulièrement rigoureux dans la documentation du programme. Ensuite, il fallait faire une séparation claire du programme en plusieurs fonctions, afin de permettre facilement la création de nouveaux parcours. Pour ce dernier point,

je vois deux solutions : tout d'abord, la création d'une fonction pour chaque parcours (chaque paramètre, ou la création d'une fonction de récupération des paramètres, puis une création de boucle if qui test les différents paramètres, puis effectue le parcours. Afin de faciliter la lisibilité de mon code, j'ai choisi de mettre en place le deuxième cas. En effet, j'ai choisi de créer une séparation claire au sein de la fonction, qui permet une meilleure compréhension du fonctionnement du programme. Tout d'abord, il y a la récupération des arguments saisis par l'utilisateur, et l'attribution à des variables. Puis, on peut directement créer la boucle conditionnelle « if » qui va tester tous les paramètres et effectuer le parcours souhaité. Ainsi, l'ajout d'un nouveau parcours ne demande que l'ajout d'un « elif » à la suite du code, avec en condition le paramètre créé, sans aucune restriction, puis la création du parcours à effectuer.

Programmation du parcours

Pour ce qui est du parcours en lui-même, j'ai pu réutiliser les mêmes méthodes que lors de mes programmes de traitement XML précédent, avec l'ajout d'une simple condition permettant d'ignorer les éléments commençants et se terminant par « % », ce qui permettait de ne faire un parcours que sur les objets, et pas sur les scripts. J'ai ensuite rédigé la documentation, en incluant une partie décrivant comment créer et inclure une nouvelle fonction de parcours, ainsi qu'une documentation à part pour les différents paramètres, ce qui permet, lors de la création d'un nouveau parcours, d'enrichir facilement la documentation des paramètres avec celui qui vient d'être créé. Il ne restait plus qu'à tester, et à rendre le programme, qui a été validé. Mes efforts sur l'évolutivité de ce dernier ont particulièrement été appréciés.

Conclusion de cette mission et avantages tirés

Cette mission m'a donc permis de découvrir une notion très importante du développement d'un programme, à savoir l'évolutivité. En effet, dans une entreprise, on travaille rarement seul et pour soi, ce qu'on produit va donc très probablement être utilisé par d'autres, et va donc très probablement être amené à changer ou être modifié. Il est important de travailler sur l'évolutivité de son programme, tout d'abord afin de garantir une utilisation facilitée par d'autres personnes de l'entreprise, mais aussi et surtout afin d'assurer que la modification du programme, ou l'ajout de nouvelle fonctionnalité ne nécessite pas une refonte totale du programme, et que ces ajouts ou modifications peuvent se faire relativement facilement.

F. Conclusion sur l'ensemble des missions

Ainsi, mon stage s'est articulé autour de 5 missions principales que j'ai eu à réaliser au sein de différents projets et avec différentes équipes. La première mission était la découverte de CMoIP, puis sa manipulation avec de la supervision et de la correction. La deuxième mission était la création d'un parcours de traitement de fichiers XML, en créant une nouvelle configuration de travail (nouveau fichier XML) ne contenant que certains objets, choisis en fonction des attributs saisis par l'utilisateur. La troisième mission concernait la création d'un double parcours de fichier XML, et dans le même temps une comparaison intelligente et affichage personnalisé et adapté des différences de deux fichiers XML. La quatrième mission était la découverte du langage de programmation et de formatage de fichiers YAML, puis la complétion d'un programme de gestion de ces fichiers, et notamment des différentes instances __Include__, puis un affichage à l'issue du traitement du fichier. Enfin, la cinquième et dernière mission était la création d'un module évolutif d'affichage personnalisé de fichiers XML issus de modèles graphiques.

4. Conclusion

A. Responsabilité et collaboration dans le développement

En conclusion, en repensant et en revenant sur les différentes missions que j'ai eu à réaliser, ainsi que sur mon stage d'une manière générale, je me rends compte qu'il m'a beaucoup apporté en tant que développeur mais aussi en tant que futur employé dans le monde du travail. J'ai en effet pu rencontrer de nombreuses difficultés tout le long de la réalisation de ces missions, et chacun d'entre elle m'a permis d'adopter de nouvelles méthodes de travail qui m'ont permis d'améliorer mon code ainsi que mon efficacité d'une manière générale. En effet, à l'issue de ce stage et de toute ces missions, j'ai pu me rendre compte de l'importance du travail sur papier, afin d'avoir une meilleure vue d'ensemble du programme ou du projet. Ce travail sur papier permet notamment d'organiser de structurer son code en amont, autre point majeur dont j'ai réalisé l'importance au cours d'une des missions. En effet, l'organisation et la structure du code permet une meilleure compréhension, non seulement si on souhaite revenir sur ce même code plus tard, mais aussi et surtout pour permettre une meilleure compréhension aux autres personnes travaillant sur le projet et devant utiliser le code ou même y apporter des modifications. J'ai en effet appris qu'il fallait garder différentes choses à l'esprit en codant, et le fait de faire attention à rendre son code le plus clair possible aux autres en fait partie. Un autre élément très important et que j'ai pu apprendre durant ce stage, est le fait de toujours réfléchir en codant, de manière à essayer d'anticiper de possibles évolutions à proposer. C'est l'avantage de travailler à plusieurs sur un projet, chacun aura une solution à proposer et chacun aura une vision différente du projet, ce qui peut faire qu'une personne se rendra compte d'un cas à traiter que quelqu'un d'autre n'avait pas envisagé. C'est pourquoi à de nombreuse reprise on m'a encouragé à ne pas m'imposer de limites et à proposer des solutions ou améliorations au programme si je les jugeais utiles voire nécessaires. Un autre élément important est l'intégration au code, le plus tôt possible mais surtout au fur et à mesure, d'une gestion d'erreur rigoureuse, afin de guider l'utilisateur sur l'erreur déclenchée, afin qu'il puisse savoir comment la corriger. Cela passe notamment par un contrôle des éléments passés en argument par l'utilisateur, afin d'afficher les éléments attendus si la saisie de l'utilisateur n'est pas correcte (pour n'importe quel type d'erreur), mais aussi par de nombreux tests tout le long du code afin de déclencher des affichages adaptés. Cela peut notamment se faire avec l'ajout de fonction de débug qu'on effectue à chaque étape cruciale du code. Enfin, un dernier point important à garder à l'esprit lorsque l'on code, et à intégrer dans ses méthodes de travail, est l'évolutivité. L'évolutivité d'un programme est la capacité à ce dernier de changer, d'être modifier. Or, tout au long d'un projet, il y a de fortes chance qu'un programme soit modifié, que ça soit pour apporter des corrections, des modifications, ou des ajouts. Ces changements ne sont par ailleurs pas forcément effectués par le créateur du programme. Si un programme est peu ou pas évolutif, alors les modifications vont poser des problèmes, que ça soit dans la structure du programme, la gestion d'erreur et celle des variables. Il est alors primordial, de rendre le plus facile possible les améliorations futures. Pour ce faire, il faut faire notamment attention aux variables, aux boucles, et à la structuration du code, l'idéal étant d'intégrer le principe de « diviser pour régner », c'est-à-dire d'isoler chaque fonction en une fonction. Une fonction python aura un et un seul rôle, et utilisera si besoin d'autres fonctions. Il est aussi important de bien gérer les variables, avec des noms les plus explicites possibles, et en faisant attention à la gestion des variables locales et globales pour faciliter leur utilisation dans de nouvelles fonction, ou encore la création de variables qui ne lèveront pas d'erreur dans leurs interactions avec les autres variables du programme.

Ayant intégré tous ces éléments au fur et à mesure de la réalisation de mes missions j'ai réalisé

que selon moi, chaque développeur a une sorte de responsabilité par rapport à son code. Cette responsabilité revient à ne pas coder de manière « égoïste », et de toujours garder à l'esprit de faire son maximum pour faciliter l'utilisation de son code par d'autres personnes, mais aussi de permettre le plus possible la modification de ce dernier, ou l'ajout de fonctionnalités. Cela passe également par l'intégration d'une gestion d'erreur suffisamment développée, afin de guider les utilisateurs vers des solutions aux erreurs déclenchées par leur utilisation du programme. J'ai par ailleurs pu observer cette notion de responsabilité au sein des équipes dans lesquelles j'ai eu la chance de travailler, avec une grande importance de la communication, où personne ne craint de poser des questions à chacun afin de précisément comprendre un aspect de son code ou de son travail. J'ai alors pu observer l'importance de travailler dans un projet en réfléchissant en tant qu'équipe, et non pas de manière individualiste, ce qui va finalement être bénéfique non seulement au plus grand nombre mais aussi à soi-même.

B. Impacts sociétaux de l'activité de l'entreprise

L'activité de l'entreprise dans laquelle j'ai effectué mon stage a des impacts sociétaux importants, notamment en termes de communication et de collaboration. L'accent mis sur la responsabilité individuelle et collective des développeurs dans la production de code de qualité et facilement modifiable favorise une culture de transparence et de partage des connaissances. Cette approche renforce les liens entre les membres de l'équipe et encourage un environnement de travail inclusif et solidaire. De plus, en facilitant la compréhension et l'utilisation du code par d'autres développeurs, l'entreprise contribue à la formation continue et au développement professionnel de ses employés, ce qui a des répercussions positives sur le bien-être et la satisfaction au travail.

C. Impacts écologiques de l'activité de l'entreprise

En ce qui concerne les impacts écologiques, l'entreprise est consciente de l'importance de réduire sa consommation de ressources et sa génération de gaz à effet de serre. Elle adopte des pratiques de développement durable, telles que l'optimisation de l'utilisation des serveurs et des infrastructures informatiques, afin de minimiser l'empreinte carbone de ses activités. Durant mon stage, j'ai été impliqué dans des projets visant à améliorer l'efficacité énergétique des systèmes informatiques, par exemple en optimisant le code pour qu'il consomme moins de ressources et en promouvant l'utilisation de technologies plus écologiques.

D. Résultats et perspectives pour et par rapport à l'entreprise

Les résultats obtenus par rapport aux objectifs fixés ont été satisfaisants. J'ai pu développer des outils de traitement de fichiers XML, notamment des outils de manipulation, de comparaison, et de création de ces fichiers. Ces outils représentaient l'objectif initial de mon stage, visant à faciliter le travail sur les fichiers XML, en particulier dans le cadre du contrôle aérien. L'objectif principal était de simplifier la création de configurations de travail pour les différentes équipes et projets.

En outre, ces objectifs ont même été dépassés, puisque j'ai également travaillé sur la création d'outils pour des modèles graphiques et des fichiers YAML. Cela a permis de proposer une plus grande diversité de solutions pour des projets tels que CLA et SRSA.

En ce qui concerne les perspectives au sein de l'entreprise, j'ai trouvé ce stage très intéressant et enrichissant. Mes relations avec les équipes avec lesquelles j'ai travaillé ont été excellentes, ce qui me fait envisager cette entreprise pour une alternance en troisième année. J'ai déjà commencé à me renseigner à ce sujet.

Pour finir, je dirais donc que ce stage a été une expérience très positive. Non seulement j'ai acquis de nouvelles compétences techniques et approfondi mes connaissances en traitement de fichiers XML, mais j'ai également pu développer des outils supplémentaires pour des modèles graphiques et des fichiers YAML. Ces contributions ont été bien accueillies par les équipes, renforçant ainsi mon intérêt pour une future collaboration avec l'entreprise.

Sources :

https://www.soprasteria.com/fr

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

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

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