Guillaume Daniel
L3

Rapport professionnel d'activités

(Octobre 2011 - Septembre 2012)

E.P.S.I - PARIS
70 rue Marius Aufan
92300 Levallois-Perret

Digiplug SAS
12 rue Godot de Mauroy
75009 Paris

Je tiens tout d'abord à remercier mes parents pour leur soutien et leur aide.

Je tiens aussi à remercier tout particulièrement Charles Viry, mon tuteur au sein de Digiplug, pour sa disponibilité et les précieux conseils qu'il m'a donné pendant cette année.

Je remercie également Lucille Desbois, assistante aux ressources humaines chez Digiplug, qui m'a beaucoup aidé dans les démarches administratives.

Je remercie également toute l'équipe pédagogique de l'EPSI pour leur engagement et qui ont su répondre à toutes mes questions.

Introduction

Dans le cadre de ma troisième année à l'EPSI, j'ai eu l'opportunité d'effectuer ma formation en alternance au sein de la société Digiplug. Spécialisée dans la gestion et la distribution de contenus numériques, elle développe principalement une application permettant aux producteurs (les labels ou les éditeurs) de distribuer le contenu de leur catalogue vers les revendeurs finaux (sites de téléchargement ou de streaming).
Cette plate-forme est utilisée par une multitude d'acteurs répartis sur l'ensemble de la planète. Ils n'utilisent pas les mêmes protocoles de communication et n'ont pas forcément les mêmes exigences de confidentialité. De plus, les ayants droits peuvent avoir des besoins plus spécifiques qu'un simple transfert de fichier, notamment des contraintes liées aux droits de diffusion ou à une politique commerciale. C'est donc un produit très sujet aux évolutions et aux changements, mais qui se doit de rester cohérent dans son fonctionnement interne.

J'ai intégré l'équipe de développement sous la tutelle de Charles Viry, *release manager* d'Anthologic en octobre 2011. Au cours de cette année il m'a été confié trois missions, toutes liées à la plate-forme. La première consistait à réaliser une application permettant d'effectuer des recherches dans les très nombreux fichiers journaux générés. Ce projet m'a été confié personnellement et les technologies à utiliser étaient les mêmes que celles de la plate-forme. C'est un bon point d'entrée pour se familiariser avec les outils et les processus utilisés par Digiplug. La seconde mission était le développement d'un outil pour exploiter certains fichiers de méta-données dont la taille et la structure empêchait leur analyse avec un éditeur de texte. Comme cette application n'était pas uniquement destinée à un usage interne, des étapes supplémentaires ont été nécessaires : conception et validation, développement d'un prototype, tests, puis intégration de la version finale au sein de la plate-forme. J'ai donc du collaborer avec différentes équipes (AD, test) afin de mener à bien ce projet. La dernière mission m'a permis de rentrer dans le vif du sujet : Anthologic. Il m'a été demandé d'implémenter une tâche automatique dans l'application afin de supprimer certains contenus expirés. J'ai du me familiariser avec le projet, étudier les sources, et demander aux autres développeurs les détails dont j'avais besoin pour réaliser l'implémentation.

Dans ce rapport je présenterai en premier lieu Digiplug et les différents services qui composent la société, puis je présenterai de manière approfondie les différentes missions qui m'ont été confiées. J'exposerai ensuite les différentes problématiques que j'ai pu rencontrer, les solutions apportées et les choix opérés. Enfin j'essaierai d'expliquer les résultats obtenus et de mettre en avant de possibles améliorations.

Présentation de l'entreprise et des services

I - Digiplug :

Fondée en 1998, Digiplug est un éditeur de logiciel et un fournisseur de service d'applications spécialisé dans le multimédia et les divertissements. La société développe plusieurs solutions pour offrir aux différents acteurs de la distribution digitale, qu'ils soient titulaires des droits, détaillants ou consommateurs les outils nécessaires. Le cœur de Digiplug est une plate-forme (« Anthologic ») qui permet la gestion et la distribution automatique du catalogue d'un ayant droit vers une multitude de *BPs (Business Partners)*. Capable d'ingérer différents types de contenus : audio, vidéo, livre et images (de manière générale, on parlera de *binaires*) , la plate-forme a besoin de certaines informations, comme les droits commerciaux et légaux (*Terms of Use, Price* Codes, planification) et les méta données (titre, artistes, etc) pour pouvoir assurer l'intégralité de la distribution.

Digiplug expose un catalogue unique à chaque BP selon les droits et les campagnes fixés par les producteurs. Ils sont mis à jour dynamiquement lorsqu'un nouveau *produit* est ingéré, et la plate-forme va se charger d'effectuer les opérations de transcodage afin que les binaires soient au format approprié. La livraison au détaillant se fera une fois toutes les étapes de contrôle effectuées, et en respectant la gestion des droits (*watermarking, DRM)*. De nombreux outils de suivi (*reporting*) sont mis à disposition aux ayants droits par la plate-forme afin de suivre en temps réel les changements d'états de leurs produits, et l'avancement de la distribution.

La plate-forme étant connectée à différents acteurs qui n'utilisent pas les mêmes techniques pour communiquer, elle doit embarquer beaucoup de configuration unique pour chacun des BPs ou des producteurs. Cette configuration doit pouvoir changer facilement sans avoir à relancer toute l'application. Plusieurs outils ont donc été mis au point en plus des fonctionnalités métiers, afin de permettre la maintenance, la personnalisation et le suivi: un SDK qui permet aux développeurs d'utiliser une API standardisée pour manipuler certaines données et d'effectuer des opérations de masse, des interfaces internes à destination des équipes opérationnelles afin d'effectuer des modifications sur les droits commerciaux, des outils de *reporting* pour le management et d'autres destinés à une utilisation interne.

C'est donc un « super projet » qui regroupe différents modules et qui est interfacé avec d'autres applications hébergées chez les partenaires. Compte tenu du volume d'informations susceptible d'être traité, une grande cohésion des différentes étapes et un cloisonnement entre les *produits* est nécessaire. Le suivi, les tests et le contrôle de qualité sont donc indispensables, d'autant plus dans un secteur où l'image est très importante. Digiplug se doit d'être très réactive aux évolutions des besoins du marché où l'évolution technologique occupe

un rôle importante (nouveaux codecs, nouveaux supports) et où les contraintes de diffusion (protection des droits) sont omniprésentes.

Les locaux sont basés au cœur de Paris dans le 8e arrondissement. Les différentes équipes sont regroupées au sein de deux open space, des bureaux individuels étant également

occupés par certains membres de la direction. Malgré cette proximité entre tous les employés, l'ambiance est très calme et studieuse. Beaucoup d'échanges se font en anglais. En effet, un certain nombre de collaborateurs sont anglophones et la communication n'est donc possible que dans cette langue. On peut distinguer plusieurs entités : les équipes opérationnelles, les équipes techniques et les équipes de design et de test. Nous allons voir comment elle sont organisées entre elles, et quelle est la hiérarchie décisionnelle.

II - Les différents services :

Un projet aussi conséquent qu'Anthologic ne peut pas être maintenu ni amélioré par quelques individus qui feraient tout le travail. Plusieurs équipes, elles mêmes composées de plusieurs membres se répartissent le travail de conception, de réalisation, de support et de maintenance. Les interactions entre elles sont très présentes, et il est rare qu'une journée se déroule sans avoir été consulté ou aller chercher de l'information auprès d'une autre équipe. Nous allons d'abord commencer par présenter les outils de communication utilisés en interne, et qui sont nécessaires au suivi des différentes étapes. Nous introduirons dans le même les équipes qui composent la société.

Digiplug utilise l'outil *Jira*, qui est un système de gestion de projet, de suivi de bugs et de gestion des incidents. Tout problème, modification, nouvelle fonctionnalité ou autre, possède un "ticket *Jira*" qui lui est associé. Cela permet facilement de regrouper les problèmes liés entre eux, de les affecter aux équipes adéquates, et de centraliser l'information sur les tâches en cours. Néanmoins cet outil ne peut pas servir de référence : un ticket n'est pas de la documentation, ni une spécification. Pour cela, Digiplug utilise l'application *Confluence*, qui est un logiciel de wiki. Ainsi, tout document technique est archivé, et peut facilement être consulté et modifié. Les pages du wiki contiennent la spécification telle qu'elle a été exprimée par les équipes de design (*Application Design, AD)*. C'est elle qui traduit les demandes ou les besoins des clients ou des BPs en un document plus technique à l'attention des développeurs. La page est ensuite complétée par le développeur en charge de la fonctionnalité, avec le *design technique*. Il a pour but d'exposer quelles seront les parties de l'application à modifier, quelles données vont être utilisées et surtout comment elles vont interagir entre elles. On peut voir cette étape comme un prototypage.

Viens ensuite la phase d'implémentation, réalisée par l'équipe de développement. Les tâches sont attribuées via un ticket *Jira*, en fonction des disponibilités et des compétences de chacun. Des revues de code ("*peer review*") sont régulièrement mises en place afin d'assurer la qualité du code et la formation continue entre développeurs. Une première série de tests est effectuée à ce moment là (test unitaires) afin de vérifier que la fonctionnalité a bien le comportement défini par la spécification sur la page correspondante de *Confluence*. Une fois les tests validés, le code est prêt à être intégré au reste de l'application à l'aide de *SVN*, un outil de gestion de versions.

Digiplug travaille par "*sprints*", qui est une période de trois semaines au bout desquelles aboutira une version livrable du produit. C'est un des aspects de la méthodologie *Scrum*, utilisée par Digiplug. Une fois le sprint actuel terminé, toutes les nouvelles fonctionnalités et corrections sont alors embarquées dans une *release*, qui sera la nouvelle

version du produit. C'est une équipe dédiée qui s'occupe de la configuration et de la construction (*Configuration and Build Management*, *CBM team*). Elle est en charge de transformer le code source en une "archive" prête à être installée sur les serveurs de production. Néanmoins, des tests un peu plus poussés sont nécessaires avant la *release* effective de l'application. Les équipes d'administration système (*IT*) ont donc mis en place d'autres serveurs sur lesquels sont déployés des versions intermédiaires, on parle alors d'*environnement*. Ils sont en tout point semblables à la production, mais sont à usage des développeurs afin de réaliser des tests d'intégration ou des tests de performance sur des données quasi réelles afin de simuler la production. Ces environnements ne disposent bien évidemment pas de toutes les informations confidentielles telles que les noms de comptes, les mots de passe ou les adresses *IP* des serveurs.

Du code exécutable sans données à manipuler ne fait pas grand chose.. Une grande partie des fonctionnalités d'Anthologic est due à l'utilisation des bases de données relationnelles. Leur importance est telle que deux équipes sont dédiées à leur bon fonctionnement : l'*IT-DB*, chargée de l'administration et de la configuration des serveurs, ainsi que la *Data Team*, responsable de l'utilisation qui en est faite. La quasi totalité des opérations effectuées par l'application implique un traitement en base de donnée, il en va de même pour les différents objets objets métiers qui sont systématiquement persisté. Les accès à la base ne se font pas directement dans l'application, des *mappings* objet-relationnel sont mis en place afin d'assurer la cohérence et la sécurité des accès à la base.

Toutes ces équipes travaillent ensemble afin de continuer à améliorer Anthologic, mais nous avons oublié d'en mentionner certaines. Commençons par l'Architecture : c'est elle qui veille a ce que les nouvelles fonctionnalités soient cohérentes avec l'existant, que les briques s'intègrent correctement avec le reste du mur. Ce sont eux qui effectuent les tests de performance, et qui valident en amont les spécifications de l'*AD* avant le développement. Ils représentent un peu la clé de voûte de l'application. De l'autre côté, l'équipe de test valide les fonctionnalités, et assure qu'il n'y aura pas de régression par rapport à la *release* précédente. C'est un travail très important, l'automatisation des tests étant indispensable afin de couvrir complètement l'application.

Voilà pour le côté technique, mais il existe également des équipes responsables du fonctionnement métier de la plate-forme : l'*Operations Team (Ops)* et le "Studio". Les *Ops* doivent assurer que la plate-forme fournit bien le service demandé par les utilisateurs, ils ont donc accès à presque toute les écrans de *reporting* qui leur permet de connaître l'état d'une livraison ou d'un produit à n'importe quel moment. Ce sont vers eux que les acteurs se tournent lorsqu'il y a un problème. Ils sont sous la responsabilité de la direction commerciale.

En effet, ce sont eux qui vont vérifier les droits et le bon acheminement des produits commandés, et sont donc responsables d'un éventuel problème lors de la distribution.

Le "Studio" occupe également un rôle important. C'est lui qui est en charge de la qualité des *binaires* qui vont être livrés par *Anthologic*. Un important travail de post production est réalisé par Digiplug : réécoute des pistes audio, vérification des clips, etc. Ses membres sont également de très bons référents techniques, ils connaissent les techniques de mixage, de transcodage, de watermarking et d'autres. Le "Studio" est indispensable lorsque une nouvelle fonctionnalité ou un bug touche une partie multimédia.

Pendant cette année, j’ai été amené à collaborer directement avec toutes ces équipes. J’ai beaucoup appris de ces nombreux échanges : j’ai vu de nouvelles façons de résoudre certaines problématiques, j’ai été formé sur certaines technologies, je me suis familiarisé avec de nombreux outils et les processus utilisés par Digiplug.

Missions

I - LogFinder :

Comme nous avons pu le voir précédemment, Anthologic est un projet très conséquent et très complexe. Pour pouvoir mesurer l’état de la plate-forme au fur et a mesure de son exécution,, mais surtout d’être capable d’obtenir des traces afin de remonter a l’origine d’un problème, les erreurs ainsi que les opérations courantes sont enregistrées dans des fichiers journaux (*logs*). De part sa nature, Anthologic génère un grand nombre de ces fichiers. On parle ici de plusieurs centaines de gigaoctets. Les logs sont regroupes sur un serveur, et sont accessibles en consultation via une simple interface web.

Néanmoins, lorsque l'on souhaite faire une recherche sur plusieurs fichiers, que ce soit une simple chaîne de caractères ou une expression rationnelle (*regexp*) compliquée, la tâche s'avère tout de suite très fastidieuse. En effet, il n'existe aucun moyen de masquer le grand nombre de fichiers, voir même de réduire le champ de recherche afin de simplifier et d'accélérer les recherches. Lorsque cette opération doit être effectuée plusieurs fois par jour, cela entraîne une perte de temps, mais c'est surtout très répétitif et sujet aux erreurs.

Il m'a été demande de concevoir et de réaliser une application web permettant de rechercher des *motifs* efficacement dans tous les fichiers de logs. Cet outil est a destination des équipes techniques, et doit être le plus simple possible. De plus l'application doit être capable d'effectuer des statistiques sur les *stack traces*, qui sont la suite d'appels de fonctions lorsqu'une exception survient pendant l'exécution. L'enjeu ici, est le traitement d'un volume de données considérable dans un laps de temps acceptable (une ou deux minutes).

Pour développer cette application, il m'a été demandé d'utiliser les mêmes technologies et outils que ceux employés pour le projet Anthologic, à savoir le langage de programmation Java et le *framework* Spring. De plus, le développement se fait au sein de l'environnement Eclipse, la compilation et le déploiement étant assurées par Maven. Ce sont des outils qui sont très utilisés dans de nombreuses entreprises, et qui offrent de nombreuses fonctionnalités très appréciables en plus de la simple édition de texte. On peut notamment évoquer les capacités d'Eclipse à naviguer très facilement dans un code source très large et à retrouver des symboles pour accéder aux définitions de méthodes. Maven permet lui de gérer efficacement les multiples dépendances d'un projet, notamment lorsqu'il s'agit de librairies externes, qui ne sont pas maintenues par les mêmes équipes, et d'automatiser les cycles de développement (compilation, tests unitaires, tests d'intégration, …).

Je n'étais pas familier avec toute cette suite a part le langage Java dans lequel j'avais déjà écrit quelques programmes et l'utilisation d'autres outils de gestion de production (GNU make, Ant). J'ai du me former seul sur ces technologies, en consultant plusieurs ressources trouvées sur Internet, mais aussi en échangeant avec différents collègues. J'ai commencé par lire la documentation fournie avec les outils et quelques tutoriels. Ici j'ai appris qu'une grande partie du code source pouvait être generée ppréalablement par Eclipse et Maven, notamment des *templates* et les squelettes de classes. J'ai également appris comment je pouvais utiliser Spring afin de réaliser l'application.

Je devais réaliser la conception et le développement seul, et j'avais trois mois pour mener à bien ce projet. Je jouissais donc d'une très grand autonomie, même si j'ai souvent été

amené à aller consulter des collègues afin d'obtenir de l'aide ou un renseignement. Ici la difficulté était d'arriver à assimiler rapidement de nouvelles connaissances et de les mettre tout de suite en pratique.

II - DDEX XML Viewer :

La plate-forme Anthologic ingère de très nombreux contenus différents. Il y a les binaires qui sont les fichiers multimédia (image, musique, vidéo, e-book), mais surtout les fichiers de métadonnées qui sont bien plus volumineux. Un problème de standard se pose : chaque plate-forme de téléchargement de musique applique sa propre politique pour

représenter les différents droits et campagnes. DDEX (Digital Data Exchange) est une norme qui vise à regrouper tous les différents acteurs de la distribution digitale autour d'un standard commun.

Cette commission vise à élaborer un format unique d'étiquette numérique, un *tag*, à appliquer sur un binaire vendu sur Internet. Cette étiquette contiendra des informations clefs telles que le titre de l'œuvre, son auteur/compositeur, son interprète, mais aussi les pays autorisés ou un *price code*. Ce tag sera standardisé et lisible par tous les intervenants de la chaîne de valeur de la musique, tout les acteurs devraient tirer profit de cette standardisation.

Les producteurs pourront récupérer des statistiques précises sur les ventes en ligne afin d'optimiser leur marketing et fluidifier les échanges à destinations des plates-formes de téléchargement. Ces dernières tireront profit de ce meilleur marquage dans le cadre de leurs moteurs de recommandation et ce, afin de formuler des propositions commerciales au plus près des comportements d'achat et des goûts réels des consommateurs.

Pour Digiplug, cela signifie d'être capable d'ingérer ce nouveau format de méta données, mais aussi de proposer une solution pour consulter ces fichiers dans le cadre d'une vérification ou d'une erreur. En effet, DDEX utilise le format de balisage XML, et il est courant qu'un fichier fasse plus de cent mille lignes. On comprend qu'un parcours et une recherche manuelle soient quasiment impossible, même en utilisant un visionneuse de fichier XML. De plus les acteurs qui vont utiliser ce format ont uniquement besoin des informations *décrites* par le standard (les droits) qui sont reparties un peu partout dans le fichier.

Il m'a été demandé de réaliser cette application, qui transformerait un fichier XML de DDEX en un format plus lisible, et mettant en avant les informations métier. L'application devait d'abord répondre à un besoin interne, mais elle a ensuite été proposée à notre principal client Universal Music, qui s'est montré très intéressé par une telle solution. En plus de devoir couvrir les usages propres à Digiplug, il faut maintenant tenir compte des demandes du client. C'est donc des phases de développement plus compliquées par rapport à la mission précédente. En effet, des réunions ont été organisées avec un représentant d'UMGI, et le projet pouvant être monnayé par la suite, les contraintes de qualité sont vraiment présentes. J'ai donc du travailler très étroitement avec un membre de l'équipe d'AD et un membre de l'équipe de test, afin d'assurer le suivi du projet.

Une présentation des résultats sous forme de page web était souhaitée car cela facilite la distribution du projet par la suite, et surtout cela peut facilement être intégré à Anthologic. J'ai utiliser la technologie XSLT, qui permet de transformer du XML en d'autres types de documents texte, ici du HTML. Néanmoins une simple mise en forme ne suffit pas pour masquer la complexité des fichiers DDEX. Des options de filtrage et de regroupement sont indispensables, ainsi qu'un résumé fonctionnel des méta-données (artiste, nombre de pistes dans le cas d'un album audio, …).

Ce projet était donc beaucoup plus encadré que le LogFinder, et des points quotidiens étaient organisés afin de mesurer l'avancement du développement. Un prototype a ensuite été présenté au client, et a été mis a disposition interne afin de remonter les impressions et les avis des utilisateurs.

III - Purge UCS :

Les missions précédentes sont des excroissances d'Anthologic. Elle ne font pas pas partie intégrante du projet. La dernière mission qui m'a été confiée au cours de cette année m'a permis de rentrer vraiment dans le vif du sujet. Pour effectuer la livraison finale vers le BP, Anthologic a besoin d'un ou plusieurs dépôts qui contiennent les binaires à transférer. Un de ces espaces est le dépôt UCS, utilisé pour Deezer par exemple. Leur taille augmente avec le temps, car de nouveaux contenus sont sans cesse acheminé, mais il n'existe aucun moyen de purger les *orders* qui auraient expirés ou qui n'ont plus besoin d'être conservés.

La plate-forme embarque des *jobs*, qui sont des tâches ordonnancées par l'application, et qui permettent d'automatiser certains traitements. La teneur de la fonctionnalité à développer fait qu'il est naturel de l'implémenter de cette façon. En effet, on peut décrire son principe la sorte : à intervalles régulières, trouver les *orders* à purger et les marquer. C'est donc un traitement qui n'a pas besoin de données particulière, à part bien sûr les informations quant à l'*order*.

On peut remarquer le le *job* en lui-même ne va pas effectuer la suppression en elle-même, mais va uniquement trouver les *orders* qui rentrent dans certains critères. Le travail de suppression sera effectué par un autre mécanisme.

Pour cette mission, j'ai été confronté à tous les processus utilisés par Digiplug. La conception n'a pas été réalisée par moi, le seul document de référence que j'avais à ma disposition était la page de wiki Confluence réalisé par l'équipe d'AD, qui ne donnait que quelques indications et spécifications fonctionnelles. J'ai du réaliser l'étude technique, puis effectuer le développement. L'enjeu ici était de laisser le technique de côté pour bien comprendre le fonctionnement attendu, et être capable d'utiliser l'existant pour arriver à une solution. En effet, l'application intègre une API a destination des développeurs, qui permet d'effectuer de nombreuses tâches courantes.

Le cahier des charges n'était pas du tout précis dans sa version initiale, et j'ai souvent du aller chercher la bonne information auprès de mes collègues afin de réellement comprendre ce qui était attendu. De plus, la technique a utiliser ne faisait pas l'unanimité : traitement uniquement en base de données ou alors code applicatif ? De nombreux jours ont été consacrés à la compréhension et l'analyse.

Réalisations et résultats

I) LogFinder :

L'outil de recherche est une application web client-serveur simple. L'utilisateur se verra présenter un formulaire lui permettant de choisir les paramètres de sa recherche, puis le serveur se chargera de traiter la demander et renverra les résultats.

Pour ce projet, j'ai du utiliser le framework Spring base sur Java qui facilite le développement et les tests d'applications. Il intègre de puissantes fonctionnalités cote serveur, telles que les *jobs* dont nous avons déjà brièvement parle. J'ai surtout utilise l'aspect MVC de Spring pour développer le LogFinder. L'outil Maven permet d'automatiser les différents cycles de développement, mais surtout est capable de gérer les dépendances et de générer un squelette d'application via les archétypes.

Après l'installation des outils sur mon poste de travail, j'ai commencé par lire un tutoriel sur Spring, me permettant de comprendre la structure d'une *webapp*, et comment sont organisés les composants entre eux. J'ai ensuite généré un prototype à l'aide de Maven, et j'ai commencé à expérimenter autour.

Une fois bien familiarisé avec les différents composants, je me suis lancé dans la réalisation de l'application. On peut distinguer différentes étapes lors du traitement :

- l'utilisateur saisi les paramètres de sa recherche via un formulaire
- la requête est ensuite traitée par l'application (recherche sur les fichiers de logs)
- les résultats sont renvoyés au client

Dans le patron de conception MVC, cela se traduira par deux contrôleurs. Un qui gérera le formulaire, et un autre chargé de la recherche. Des vues seront crées pour le formulaire et les deux types de recherches (recherche standard, et *stack traces*). Quant aux modèles ils contiendront les résultats. J'ai créer le squelette de ces différents composants, puis j'ai ensuite écrit les différentes pages JSP, qui permet de créer du contenu HTML statique et d'y intégrer du code Java. Il y avait la page de formulaire, la page pour afficher les résultats de la recherche classique et celle pour la recherche de *stack traces*.

Le formulaire demande à l'utilisateur de choisir un projet, et des fichiers parmi ceux prédéfinis. Le contenu de la liste déroulante est chargé dans la mémoire lorsque la page est générée, et la liste des fichiers change lorsqu'on sélectionne un autre projet à l'aide d'un code JavaScript.

Cette configuration se fait côté serveur à l'aide d'un simple fichier texte, qui contient les fichiers à utiliser ainsi que les chemins d'accès. Cette ligne se découpe en trois parties séparées par des deux-points: le nom du projet, le chemin vers les fichiers de log, et enfin le nom des fichiers. Dans l'exemple précédent, on voit que le projet ATLC utilise les fichiers « intégration », « monitoring », « r2 », « sdk » et « server ». Comme il peut y en avoir plusieurs, l'application considère tout les fichiers dont le nom commence par un de ceux spécifiés. Le chemin contient des caractères spéciaux « %% » qui permettent également de désigner des répertoires commençant par la même chaîne .

Il faut maintenant implémenter la logique de recherche. Comme l'utilisateur peut chercher dans plusieurs fichiers, il faut d'abord identifier les cibles de la recherche. On peut ensuite effectuer la recherche sur chacun de ces fichiers. Les résultats sont ensuite insérés dans le modèle qui sera alors affiché par la vue correspondante une fois le traitement terminé. L'utilisateur peut également demander d'obtenir du contexte supplémentaire (lignes avant et après la correspondance) , il y a donc beaucoup de traitement et d'écriture en mémoire pour faire cette étape.

En ce qui concerne la recherche de *stack traces*, c'est quasiment le même fonctionnement, sauf qu'il faut effectuer certains groupements. En effet, on souhaite regrouper les erreurs entre elles, pour pouvoir faire des statistiques. La logique est la même, sauf dans la recherche du motif.

Plusieurs questions se posent alors. En effet, que se passe-t-il si deux utilisateurs effectuent une recherche sur le même projet et mêmes fichiers ? Il y a de grandes chances que les fichiers soient identiques si le laps de temps entre les deux requêtes est suffisamment court. De plus, il peut y avoir un très grand nombre de résultats, et donc une empreinte mémoire très importante. Ceci peut entraîner un ralentissement de l'application, voir un dysfonctionnement si le serveur mets trop de temps à répondre.

Pour résoudre le premier point, j'ai mis en place un petit système de cache , qui est une copie des fichiers de logs. Lorsqu'une nouvelle recherche est effectuée, on vérifie si un tel super-fichier existe, et si il est suffisamment récent. Si c'est le cas, on utilisera celui-ci plutôt que ceux normalement utilisés. Néanmoins cette solution entraîne une redondance sur le système de fichier, mais accélère le traitement.

Le deuxième point est plus délicat. En effet, lors du développement, je n'avais pas accès à la totalité des fichiers de logs, j'ai donc du travailler avec des fichiers que j'avais sur mon poste, et non les logs eux-mêmes. Je n'ai pas rencontré de problème particulier pendant mes tests, et même les gros fichiers étaient traités rapidement. Néanmoins, lorsque le logFinder à été installé sur un des serveurs de log de Digiplug, l'application était extrêmement lente. En effet il y a un très grand nombre de fichiers (des dizaines de milliers), et tous sont très volumineux (plusieurs mégaoctets). Le logfinder tel que je l'ai réalisé est inutilisable dans l'environnement de production.

Une solution aurait été d'envoyer les résultats au client de manière asynchrone au fur et à mesure qu'ils étaient trouvés. Je n'avais pas pensé à orienter l'application de cette manière au début, et il aurait été très coûteux de la réécrire maintenant. De plus, mon manque de connaissance du framework Spring, m'a incité à suivre une voie qui était bien documentée, mais non adaptée au cas présent. Un meilleur travail de conception aurait du être réalisé par mes soins avant la réalisation.

II) DDEX XML Viewer :

Lorsque Digiplug à commencé à implémenter le format DDEX au sein du projet Anthologic, un besoin de pouvoir consulter efficacement les fichiers XML des métadonnées a été ressenti. En effet, ces fichiers font souvent plusieurs centaines de millier de lignes, et il est donc très difficile de les parcourir manuellement. Mme en utilisant une visionneuse adaptée, le format DDEX reste lourd et peu lisible. Il m'a donc été demande de développer une petite application permettant aux équipes opérationnelles de parcourir rapidement les fichiers XML DDEX.

Les utilisateurs ne doivent pas avoir de configuration ni d'installation à effectuer. Les fichiers XML etant statiques, il est facile de les transformer en une page web, ou l'on pourra présenter convenablement les informations. De plus l'ajout de fonctionnalités dynamiques avec un langage tel que JavaScript permet de faire gagner encore plus de temps et de confort aux utilisateurs.

Mon tuteur m'a recommandé d'utiliser la technologie XSLT, qui permet d'interroger et de transformer un fichier XML. J'ai à nouveau lu de la documentation sur internet, puis j'ai

realisé un prototype qui affichait uniquement le contenu des balises du XML. On utilise ici le processeur integré aux navigateurs web, ce qui permet une utilisation facile par la suite. XSLT est un langage declaratif, et s'articule autour du concept de *template.* Un template defini comment transformer un nœud correspondant à une expression XPATH. J'ai donc realisé un premier prototype qui affichait toutes le informations contenues dans le XML, sans mise en forme.

Le format DDEX distingue trois sections : les *resources*, les *releases*, et les *deals.* Une ressource identifie un binaire (musique, video, etc) et contient toutes les informations telles que la taille ou le format. Une *release* identifie des groupes de *ressources*. Un *deal* consiste en des informations sur les pays autorisés ou le prix de vente. J'ai donc créer trois colonnes visant a contenir chacune de ces sections, ainsi qu'un petit *header* donnant tout de suite certaines informations (titre, date, ISRC, …). C'est cette version qui a commencee à être utilisée par les equipes de Digiplug. J'ai aussi été amené à travailler frequemment avec Patrick Ayemeli, membre de l'équipe de test. Tout changement ou bug était documenté par un ticket Jira, et des points très réguliers étaient organisés.

On remarque que les sections sont liées entre elles : un *deal* contient une ou plusieurs *releases*, qui elle même contient une ou plusieurs *ressource.* Il faut donc un moyen efficace de representer ces liens. J'ai choisi de faire apparaître un cadre rouge autour des elements liés entre eux , ce qui donne tout de suite un repère. De plus, j'ai ajouté des filtres, qui permettent de sélectionner un ou plusieurs pays, afin d'être encore plus précis lors d'une recherche. Chaque *resource*, *release* ou *deal* affiché est également contenu dans une boîte, qui peut être cachée en laissant seulement son identifiant de visible. L'activation se fait en cliquant sur ce dernier, et le titre de chaque colonne a le même fonctionement. Un bouton permet également d'étendre toutes les boîtes, et vice-versa.

Une fois ces changements effectués et des tests supplémentaires, une démonstration a été réalisée devant un représentant d'Universal Music. Cette présentation a permis de mieux identifier les attentes et a abouti à une nouvelle version de l'outil. Jusqu'à présent, l'utilisateur devait éditer le fichier XML afin d'ajouter le lien vers la XSLT. Désormais il peut choisir le fichier à analyser via un simple glisser/déposer. De plus une section « résume » est ajoutée, et contient une liste de tous les ISRC décrits par le fichier XML. Lorsque l'on clique sur un des liens, le mécanisme de cadre rouge mets en avant les informations contenues dans chacune des trois sections. Avec ces changements, l'outil est beaucoup plus fonctionnel, l'utilisateur peut directement aller chercher l'information qu'il désire et les éléments clés sont tout de suite repérés.

Fort de mon expérience, nous avons procédé à des tests sur des fichiers XML assez volumineux. Le temps de traitement était long mais acceptable, et j'ai ajouté une vérification sur la taille du fichier afin de prévenir l'utilisateur. Néanmoins j'ai quand même chercher à optimiser le processus. Après un profilage, j'ai remarqué qu'une grande partie du temps de traitement était passé dans une seule fonction JavaScript. J'ai donc réécris cette fonction, et factorisé une autre partie de mon code en utilisant une structure de donnée plus générique, permettant d'éviter de nombreux appels de fonctions. Le gain de performance est visible tout de suite.

Néanmoins, c'est la transformation XSLT qui prend le plus de temps. Lors de la génération du HTML, l'arborescence du XML est parcourue à de nombreuses reprises. En effet, le *header*, le résumé ainsi que les trois colonnes possèdent tous des informations identiques, mais qui vont être récupérées 4 fois. Une solution pourrait être de stocker une seule fois ces données dans un objet JavaScript, et les insérer au bon moment dans le HTML à partir de l'objet.

III) Purge UCS

La dernière mission qui m'a été confiée touchait réellement au projet Anthologic. Elle consistait en l'implémentation d'un *job* qui est une routine exécutée à intervalle de temps réguliers. La tâche ici était de trouver les *orders*, qui sont la représentation d'une commande effectuée par un BP, qui correspondaient à certains critères, puis de les marquer pour une utilisation ultérieure. Cette spécification fonctionnelle était décrite dans une page du wiki Confluence, et il m'a été demandé de réaliser la spécification technique.

J'ai donc écrit les différents traitements à effectuer, qui sont principalement des opérations sur des objets persistés en base de données à l'aide du langage SQL. Puis j'ai fait un script qui ferait tout le traitement. Néanmoins, ce n'était pas la méthode attendue. En effet, le traitement doit se faire en Java, et en utilisant l'API développée par l'équipe d'architecture. Ce SDK fourni des routines pour effectuer des opérations de masse, des *batches.* De plus l'utilisation des DAO (Data Access Object) permet d'effectuer des vérifications supplémentaires au lieu de l'écriture brute en base de données. L'algorithme reste néanmoins sensiblement similaire.

Pour pouvoir se connecter à la base, il faut ouvrir une session. Une fois établie, on récupère la liste des *orders* via une requête SQL, puis l'API *batch* se chargera de découper le traitement. Cette logique est implémentée dans un *service.* On doit spécifier le nombre maximum d'éléments à traiter en une passe, ainsi que d'autres paramètres tel que la possibilité d'effectuer les opérations en parallèle ou la durée maximale de traitement. L'API a également besoin d'une fonction à appliquer sur chaque élément. C'est dans cette routine que se fera le marquage. Comme un *order* peut contenir des *assets*, il faut également marquer ces derniers. On peut y accéder facilement via l'*order*. Ce traitement se fera dans un *repository.*

J'ai également du ajouter une vérification de l'état de l'*order* lorsqu'on souhaite y accéder. Si il est marqué comme étant à purger, une erreur doit être levée. J'ai donc du ajouter un nouveau message dans l'énumération de l'exception correspondante. Une fois le code terminé, il faut l'intégrer à la plate-forme. Ceci implique d'injecter le repository dans le service et le service dans le job, ainsi que de déclarer les classes à utiliser. Cette configuration se fait dans un fichier XML. C'est l'inversion de contrôle (IoC). En effet, on pourrait utiliser une autre classe simplement en en injectant une autre à la place. Ceci ne nécessite pas de recompilation.

La grosse difficulté que j'ai rencontré ici était la compréhension de la spécification. Il n'était pas clairement demandé de faire un traitement via Java. Et j'ai perdu du temps à

recommencer le travail. De même, je ne connaissais pas les méthodes du SDK à utiliser, ni même comment me connecter à la base. J'ai du demander de l'aide aux autre développeurs.

Conclusion

Au cours de cette année, et à travers les missions qui m'ont été confiées, j'ai évolué mon rapport avec un projet informatique. Je ne m'attendait pas à ce que les équipes soient si distinguées les unes des autres, malgré des interactions quasi permanentes. Cela permet une grande granularité dans la gestion du projet. Je me suis familiarisé avec des méthodologies de travail : gestions de tickets, gestion des estimés, documentation.

J'ai également beaucoup appris techniquement, notamment sur les phases de conception et de tests. Je me suis sensibilisé à des problématiques de fort volume de données, mais aussi à l'expérience de l'utilisateur et à la relation avec le client.

J'ai beaucoup aimé l'organisation en open space, qui facilite les échanges entre les collaborateurs. Cela permet d'apprendre des autres, et assure une certaine cohésion entre les équipes. J'ai également beaucoup apprécié de pouvoir réaliser un projet entièrement, c'est quelque chose de très créatif et que j'aimerai pouvoir continuer à faire plus tard. Je suis maintenant très soucieux des performances, et je me suis beaucoup intéressé à des techniques de programmation concurrentes.

Approche managériale

I) Développement offshore :

Tous les développements ne se fonts pas a Paris. En effet, l'équipe qui s'occupe d'Anthologic est composée de 6 personnes. C'est un petit groupe pour un projet aussi gros. C'est pourquoi Digiplug emploie également de nombreuses personnes en Inde. On appelle cette pratique l'*offshoring* et le "nom de code" utilisé par la société est IDC (Indian Development Center).

Une partie des membres de l'équipe de Paris occupe également le rôle de chef de projet et assure la communication et le suivi avec l'Inde située à des milliers de kilomètres du siège. Cela implique de nombreuses contraintes d'organisation qu'il ne faut pas négliger.

1. Différences culturelles et linguistiques. En effet, la vie d'un Indien est très ancrée dans les traditions et la famille, et leur emploi du temps se base dessus. De plus ce qui peut sembler évident pour un européen ne le sera pas forcément là-bas.
2. Coûts financiers de communication et efforts de communication accrus, engendrant un temps d'exécution plus long. Malgré l'utilisation des supports numériques pour communiquer, les échanges ne peuvent pas se faire directement. Il peut parfois y avoir des incompréhensions, ce qui entraîne une perte de temps des deux côtés.

3. Défis ciblés dans la gestion des spécifications, la gestion de configuration et l'intégration du code applicatif. Une grande attention doit être apportée lors de la rédaction des documents à destination des équipes offshore afin de faciliter leur compréhension, c'est la double gestion de projet.
4. Confiance limitée dans l'équipe offshore de la part des équipes sur site. Des revues de codes sont régulièrement organisées afin de vérifier la qualité du travail réalisé et sa bonne intégration avec le reste du projet.
5. Manque d'informations contextuelles du côté offshore. Les équipes Indienne n'ont pas forcément une vision sur la stratégie de l'entreprise, ou même les liens entre les différentes tâches à faire. Cela peut entraîner des manquements, voire des dysfonctionnements dans le code applicatif produit.

   Néanmoins cette pratique possède également des avantages, le plus important étant bien évidemment le coût du jour/homme qui est au moins trois fois inférieur à celui d'un développeur français. On peut donc avoir une équipe plus fournie pour le même coût de revient et cela facilite le *scaling-up*. En effet, si le besoin se fait sentir, il ne sera pas trop coûteux d'employer des développeurs supplémentaire pour collaborer sur un projet.

II) Organisation locale :

On peut maintenant parler de l'organisation du développement dans les locaux de paris. Digiplug utilise la méthode agile Scrum. Cette méthode s'appuie sur le découpage précis des tâches à réaliser, et sur l'autogestion des développeurs. Ces découpages sont nommes sprints, et durent trois semaines chez Digiplug. Les objectifs à atteindre sont précisément définis et un sprint aboutit sur une version livrable du projet. Les réunions qui permettent de définir ces objectifs sont appelées "scoping sessions", et font directement intervenir le client et le chef de projet. S'ensuit alors une discussion avec les différentes équipes afin d'évaluer les temps estimes. Ce sont les développeurs qui réalisent cette estimation. Lorsque la tache avancera, on parlera alors de reste a faire. On voit que beaucoup de responsabilités sont confiées aux développeurs. En plus de l'estimation,c'est lui qui doit également réaliser le design technique mais aussi la documentation. C'est ici que l'utilisation d'outils de communication efficace joue un rôle important. Tout le monde peut accéder aux ressources facilement, et être mis au courant en temps réel d'un changement dans la spécification. De plus l'organisation spatiale en open space facilite la communication entre les équipes. Tout le monde est au courant de l'évolution des différentes tâches, et des revues de code sont souvent organisées. Il y a donc beaucoup de transfert de compétences, tout le monde est "expert", ce qui permet une grande flexibilité dans la gestion du projet.