Alcatel · Lucent

International

Développement Android (JAVA / C++) pour automatisation de tests 2G/3G/4G

Anthony STEPHAN

sept.2012 - sept.2014

UPMC - Master 2 I3S

Sommaire

Remerciements

Avant toute chose, je tiens à adresser tous mes remerciements aux personnes qui m'ont permis de réaliser cette première année de Master par l'apprentissage. Ils reviennent dans un premier temps à monsieur MICHEL COMBES, Directeur Général d'Alcatel-Lucent, à monsieur JEAN CHAMBAZ, Président de l'Université Pierre et Marie Curie, ainsi qu'à monsieur YVES FOUCHET, Président de la Chambre de Commerce et de l'Industrie de Versailles-Yvelines à laquelle est rattachée le CFA de l'UPMC.

J'exprime particulièrement ma gratitude à M. CHRISTOPHE ANOUILH, mon maître d'apprentissage au sein d'Alcatel-Lucent, de m'avoir offert l'opportunité de travailler dans son équipe et de m'avoir proposé un sujet d'apprentissage en accord avec mes centres d'intérêt. La grande confiance qu'il m'a accordée pendant ces deux années m'a permis de mener à bien les projets qu'il m'a confiés. Je lui suis reconnaissant de m'avoir permis d'avancer au rythme de mon choix sans m'imposer de pression, ainsi que pour l'intérêt qu'il a voué à mon travail, si bien dans son équipe qu'à l'université. Ce fut un réel plaisir de travailler à ses côtés.

Je tiens à remercier également chacun des membres de l'équipe *AIDA* dans laquelle j'ai été intégré ; leurs qualités humaines d'écoute et de partage ont fait de ces deux années d'expérience un réel plaisir :

- Monsieur HERVE TOULAN pour l'accueil chaleureux qu'il m'a réservé lors de ma première journée ainsi que pour m'avoir présenté les locaux et les autres membres. Il m'a apporté une aide précieuse tout au long de cette première année en ce qui concerne l'utilisation des différents outils logiciels ;

- Monsieur ALEXENDRE DEHENNIN, puits de connaissances en programmation, pour avoir su m'indiquer maintes fois les meilleures démarches à suivre concernant ma manière de programmer ;

- Messieurs ARNAUD CUVILLIER, VINCENT DUPUY et SEBASTIEN BELLEAU, avec qui je suis amené à travailler lors de l'intégration de mon travail dans le projet *AIDA*, pour leur patience et pour toutes les réponses qu'ils apportent à mes questions ;

- Monsieur LY CHAU, pour son aide précieuse à propos du fonctionnement des systèmes à base de noyau Linux et des différentes couches protocolaires ;

- Messieurs FRANCK ORTSCHEIT et ABDERRHAMANE BAHOUS, pour me permettre d'utiliser de réaliser tous les tests dont j'ai besoin sur leurs plateformes ;

- Messieurs ANDRE MICHEL, VINCENT DESLOGES, CHOKRI CHAARI et STEPHANE FAILLER pour s'être rendus disponibles chaque fois que je sollicitais leur aide, ainsi que pour leur bonne humeur et pour l'agréable ambiance de travail qu'ils instaurent chaque jour.

Merci également à tous les membres des autres équipes d'Alcatel-Lucent avec qui j'ai eu l'occasion de travailler. Ils m'ont apporté une aide précieuse, le plus souvent spontanément.

J'adresse ma gratitude à messieurs BRUNO GAS et SYLVAIN ARGENTIERI, respectivement responsable de la spécialité I3S du master Science de l'Ingénieur de l'UPMC et responsable du Master 1 II/IMI orientation I3S en apprentissage, ainsi qu'à toute l'équipe enseignante qui nous ont dispensé cours, travaux dirigés et travaux pratiques. Leurs enseignements ont été de qualité et d'un réel intérêt pour notre insertion en entreprise. Leur bonne humeur et leurs qualités humaines d'écoute et de compréhension tout au long de l'année m'ont permis d'acquérir de nombreuses connaissances dans une ambiance de travail conviviale et motivante.

Je remercie monsieur JEAN-LUC ZARADER, mon référent à l'université, pour le suivi de mon travail. Je lui suis reconnaissant d'avoir fait le déplacement la première année jusqu'Alcatel-Lucent Vélizy dans l'optique de s'assurer de ma bonne intégration dans l'entreprise et du bon déroulement de mon apprentissage. J'ai été sensible à son intérêt qu'il a manifesté à l'égard de mon ressenti au sein de l'entreprise et de l'équipe.

Mon travail de recherche d'entreprise m'a été particulièrement facilité par madame PARFAITE PANTOU, alors chargée de relations entreprises au CFA de l'UPMC. Je lui suis extrêmement gré pour le travail de mise en relation avec les entreprises qu'elle a fourni. De la même manière, toutes les procédures administratives ont été facilitées par madame SYLVIE MONNIER, secrétaire du master I3SR. Je la remercie pour sa gentillesse, sa patience, et surtout pour sa réactivité pour répondre à nos mails et requêtes. Egalement, actuel chargé de relations entreprises, monsieur PHILIPPE BRUGEILLES, que je remercie pour sa visite sur le site de Villarceaux lors de la deuxième année.

Je remercie mes collègues de Master (dont les noms sont trop nombreux pour être cités ici) avec qui l'ambiance de travail est très agréable. Le climat d'entraide et les échanges que nous avons concernant nos missions respectives en entreprise sont très riches. De fortes amitiés se sont créées pendant ces années d'études, ce qui est selon moi tout aussi important que l'apprentissage de connaissances dans une formation.

Ces remerciements seraient incomplets si je n'en adressais pas à l'ensemble des membres de ma famille et tout particulièrement mes parents, qui ont su me soutenir dans mes études et dans tous les choix que j'ai réalisés. Leur confiance et leurs encouragements n'ont pu me porter que vers le haut, et c'est principalement grâce à eux que j'en suis arrivé là. Il en va de même pour l'ensemble des amis qui m'entourent chaque jour, qui savent me soutenir mais également m'apporter de précieux conseils. Leur aide m'a notamment été chère lors de la rédaction de ce rapport, et je tiens à remercier tout particulièrement :

- PAULINE AJOUX, pour avoir effectué une première relecture et pour m'avoir aidé à déterminer la meilleure syntaxe possible pour certaines de mes phrases ;
- KEVIN GIUSTO, pour avoir effectué une seconde relecture et pour avoir corrigé la syntaxe de certaines de mes phrases.

Enfin, j'ai une pensée profonde pour mes grands-parents, et notamment pour MICHEL STEPHAN, qui nous a malheureusement quittés fin 2013, mais qui aurait aujourd'hui été fier de ce que je suis devenu au terme de mes études. Ce mémoire, fruit de mes études, lui est dédié.

Résumé de la mission

Les télécommunications font aujourd'hui pleinement partie de nos vies. En tant que domaine en perpétuelle évolution, elles voient émerger de nouvelles normes et de nouvelles technologies, fournissant à leurs utilisateurs des services toujours plus efficaces et puissants.

Alcatel-Lucent est l'un des plus gros groupes mondiaux en charge du développement, des tests et validation ainsi que du déploiement de toutes ces nouvelles technologies de télécommunications, notamment en ce qui concerne les 2G/3G et la 4G LTE.

L'entreprise développe sa propre solution logicielle « AIDA » en interne, permettant d'automatiser les tests matériels et logiciels réalisés sur ses plateformes, sur quelques téléphones mobiles jusqu'à plusieurs centaines d'appareils. Malheureusement, à cause de limitations logicielles, les smartphones sous Android ne sont pas compatibles avec cette solution.

Intégré dans l'équipe de développement « AIDA » au sein d'Alcatel-Lucent Villarceaux (91), le travail que je réalise au cours de mon apprentissage consiste en la programmation d'une application Android permettant d'établir une communication entre les mobiles Android et « AIDA » ainsi que d'exécuter tous les tests demandés (passer des appels vocaux, envoyer des SMS, exécuter du trafic FTP, HTTP, UDP, piloter des tests eMBMS …).

Dans un second temps, j'ai été amené à développer une deuxième application Android en code natif (C/C++) permettant la collecte de traces réseaux, leur analyse et le décodage de messages spécifiques, le tout en temps réel.

Ce présent rapport décrit le travail que j'ai réalisé pendant mes deux années d'apprentissage.

Synopsis of the mission

The telecommunications are now a great part of our life. As a purview in a constant development, they see the emergence of new standards and technology, providing powerful and more effective services to their users.

Alcatel-Lucent is one of the most important companies in charge of the development, tests and validation, and deployment of all these new technologies of telecommunications, especially concerning 2G/3G and 4G LTE.

The company develop his own internally automation software solution called “AIDA”, allowing hardware and software tests, carried out on platforms, on some devices to more than thousand mobile phones. Unfortunately, dues to software limitations, the Android phones can’t be used with this solution.

Integrated in the « AIDA » development team within Alcatel-Lucent Villarceaux (France), the work I’ve done for my apprenticeship consists in programming an Android application, allowing establishing a the communication between Android devices and “AIDA”, together with all the requested tests (setup voice call, send SMS, realize FTP, HTTP, UDP traffic, monitoring eMBMS tests ...)

In a second step, I was in charge of programming a second Android application in native code (C/C++) to handle and analyse specific network traces, and decode the wanted messages, in real time.

This present report describes the two-years apprenticeship work I’ve done.

1

Présentation de l'entreprise

I. – PRESENTATION DE L'ENTREPRISE

1 – Le groupe Alcatel-Lucent International

Le groupe Alcatel-Lucent est né en 2006 de la fusion entre la société française Alcatel et le groupe américain Lucent-Technologies. Effectif au 1er décembre 2006, le rachat de Lucent par Alcatel montre aujourd'hui une répartition actionnariale de 60% pour Alcatel et de 40% par Lucent-Technologies.[ref.1] Il compte, fin 2013, un effectif de 68 000 employés contre 72 000 fin 2012, et à sa tête Michel Combes depuis avril 2013.

Alcatel-Lucent est le partenaire privilégié de la transformation des fournisseurs de services, des entreprises, des secteurs stratégiques tels que la défense, l'énergie, la santé et les transports, mais aussi des administrations du monde entier. Il leur fourni toutes les solutions permettant des services « voix », « données » et « vidéo » destinées à leurs utilisateurs et clients.

Deuxième plus grand équipementier réseau et télécommunications au niveau mondial, Alcatel-Lucent se plaçait derrière l'américain *Cisco-Systems* jusque 2013, mais devant le suédois *Ericsson* et le germano-finlandais *Nokia Siemens Networks*. En 2014, le groupe n'est plus que le quatrième au niveau mondial derrière Cisco, Huawei et Ericsson. Toutefois depuis l'été 2013, il a su se repositionner en tant que leader mondial dans les réseaux haut débit fixes, mobiles et convergés et dans les technologies IP et optiques. Le groupe s'appuie sur l'expertise technique et scientifique des *Bell Labs*, l'une des plus grandes organisations mondiales de recherche dans le secteur des communications.[ref.2]

Présent dans plus de 130 pays (cf. image [1] ci-dessous), Alcatel-Lucent est un partenaire local mais avec une dimension internationale. Société de droit français, son siège social est implanté à Boulogne-Billancourt depuis mai 2014.

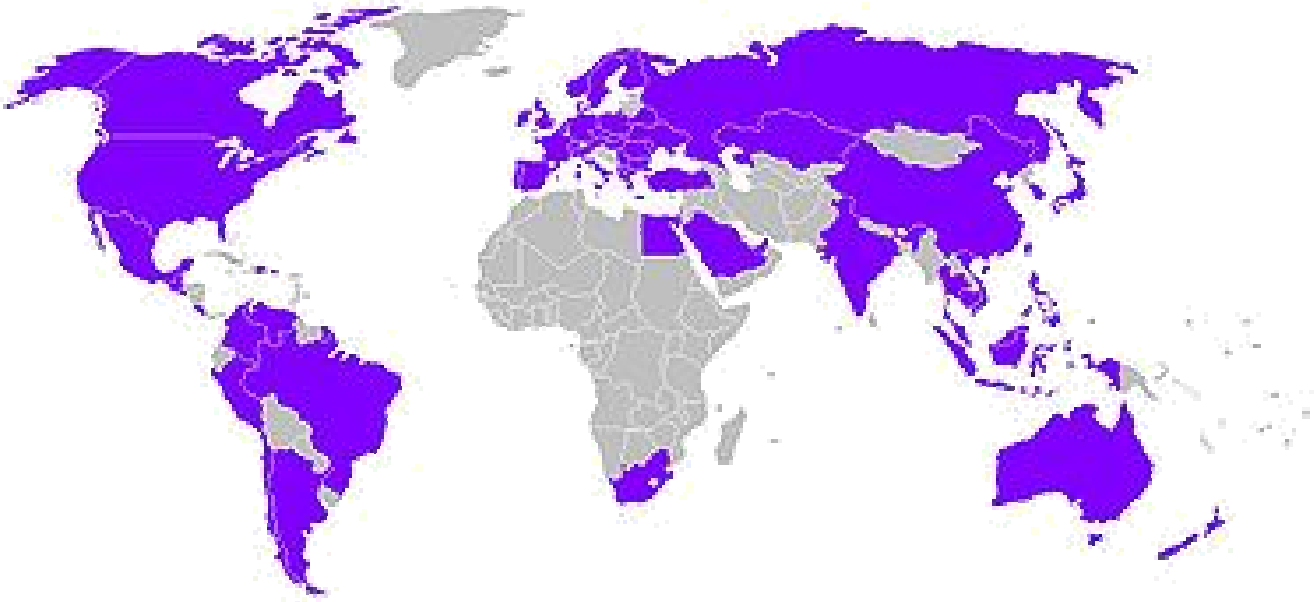


Image [1] : Carte de l'implantation d'Alcatel-Lucent dans le monde
(Les pays en violet sont ceux où Alcatel-Lucent exerce au moins une activité)

Le groupe s'articule autour de quatre principaux segments :

- **Application**, qui a pour mission de développer et de maintenir des applications et logiciels innovants pour la clientèle internationale du groupe ;

- **Réseaux**, qui a pour mission de gérer le catalogue de produits réseaux du groupe, et de fournir les produits les plus performants du secteur pour répondre aux exigences de ses clients ;

- **Services**, qui a pour but de fournir des offres complètes pour l'ensemble du cycle de vie des réseaux (allant du conseil à l'intégration, du déploiement à la migration des réseaux, de la transformation à la maintenance) ;

- **Opérationnel**, qui englobe les activités relatives aux composants pour l'industrie et aux technologies du vide.

Fier de ses nombreuses innovations à travers les années, Alcatel-Lucent notifiait en 2011 sept prix Nobels remportés et près de 30 000 brevets déposés à son actif, dont plus de 2600 déposés cette année là. Employant en 2011 plus de 76 000 personnes à travers le monde et collaborant avec plus de 250 universités, elle a généré cette année là 15.3 milliards d'euros de revenus (cf. image [2] ci-dessous).

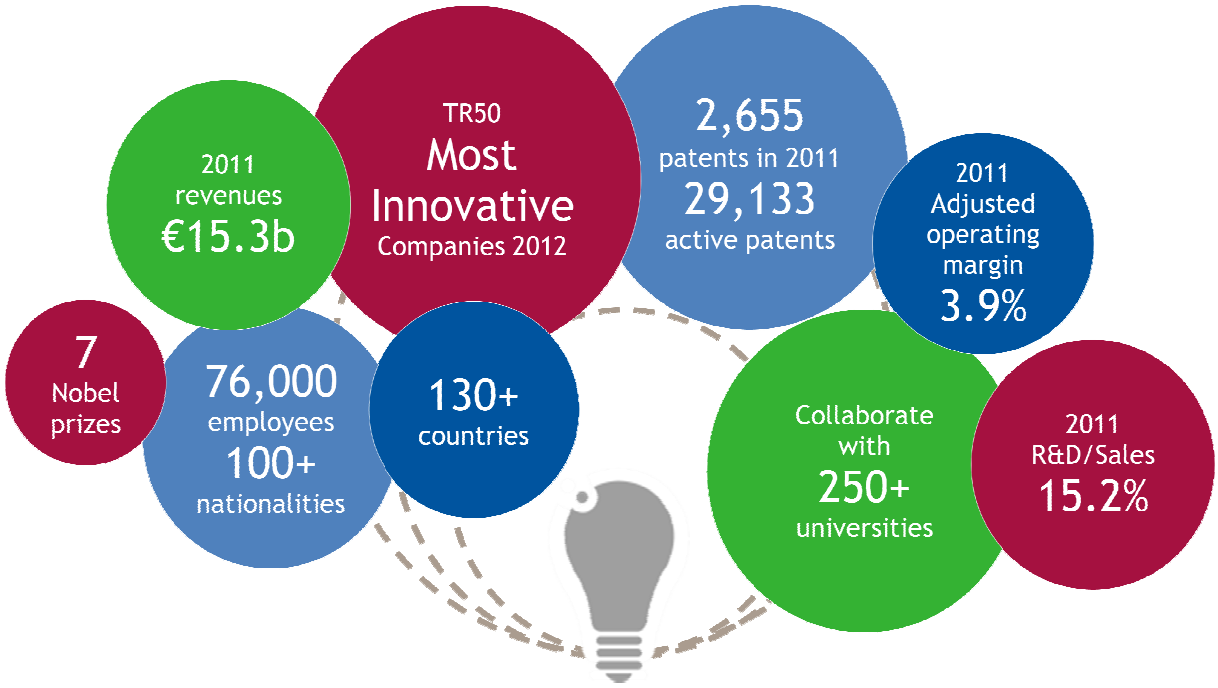


Image [2] : schéma de présentation d'Alcatel-Lucent « At A Glence »
(d'un coup d'œil) utilisé lors des présentations du groupe en 2011 et 2012.

Fondés en 1925 dans le New Jersey, les *Bell Labs* font partie du centre de recherche et de développement d'Alcatel-Lucent depuis 2009. Les recherches qui y sont menées sont d'une importance capitale dans le domaine de l'informatique et des télécommunications : on leur doit notamment la communication par fibre optique, le laser, la cellule photoélectrique, mais également le développement des réseaux téléphones, de la transmission télévisuelle et des communications satellite. Ils déposent chaque jour 3 nouveaux brevets. [ref.3]

2 – Alcatel-Lucent France

Sur le territoire Français, Alcatel-Lucent a la position de leader en transmission, en réseaux IP (Internet Protocol), en services de solutions innovantes pour l'industrie et le secteur public, mais également en télécommunications d'entreprises (centres de contact, communications unifiées, téléphonie IP …)

Le groupe est numéro 1 sur le marché français en temps que fournisseur en télécommunications et partenaire stratégique, en infrastructures sans fil, en Accès Large Bande, en IMS (*IP Multimedia Subsystem*, architecture standardisée pour les opérateurs de téléphonie, qui permet de fournir des services multimédias fixes et mobiles) ainsi qu'en réseau de télécommunication.

Présent sur 12 sites en France dont les principaux en région Parisienne (cf. image [3] ci-à droite), Alcatel-Lucent a pour principaux clients les opérateurs historiques (France-Télécom/Orange, SFR, Bouygues Telecom), les entreprises des marchés stratégiques (RTE, SCNF, RATP…) ainsi que les fournisseurs de solutions d'entreprises (Orange, Business Services, Nextiraone…)[ref.4]

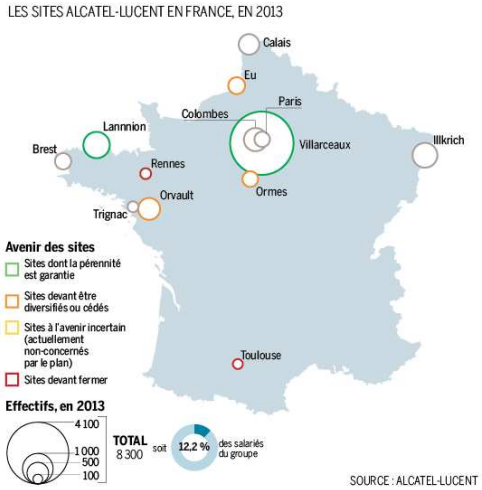


Image [3] : Carte d'implantation d'Alcatel-Lucent en France (2014)

Plusieurs filiales du groupe sont présentes sur le territoire français :

- **Alcatel-Lucent** : siège Monde, Alcatel-Lucent International et fonctions centrales Région EMEA à Paris ;
- **Alcatel-Lucent France** : Unité Commerciale France et Compte Global FT-Orange, Services, Applications, Groupe Réseaux (notamment à Vélizy) ;
- **Alcatel-Lucent Villarceaux** : 1er grand centre de recherche en Europe, ce centre héberge les équipes des *Bell Labs* France ;
- **Alcatel *Submarine Networks*** (notamment à Villarceaux) ;
- **Alcatel-Lucent Entreprise** ;
- ***Genesys*** (centres de contacts) ;
- **RFS** (systèmes d'antennes).

3 – Le site de Villarceaux : Cité de l'Innovation

Alcatel-Lucent Nozay est implémenté sur le site dit de « *Villarceaux* ». Dans un espace de vie et de travail de plus de 200 000m² accueillant les Bell Labs France, ce sont plus de 5000 personnes qui y travaillent chaque jour.

Depuis début 2014, les salariés de Villarceaux comptent désormais dans leurs rangs ceux de Vélizy (dont les postes n'ont pas été impactés par les plans de licenciement lors de la délocalisation du site). Cette fusion, accompagnée de nombreux aménagement spécifiques (construction de nouveaux bâtiments de bureaux et de plateformes de tests) est en partie expliquée par l'ambition du groupe de vouloir créer la Cité de l'Innovation en France : un centre de R&D au niveau mondial sur lequel sont regroupés des activités de dernière technologie, dans un cadre innovant.

Image [4] : La Tour centrale sur le site de Villarceaux à Nozay.

Situé à 30km de Paris, entre la nationale N118 et l'autoroute A6, il jouit d'une position privilégiée de par sa proximité avec la capitale et son siège social situé à Boulogne Billancourt, ainsi que par sa capacité à accueillir plusieurs milliers d'employés.

Depuis le déménagement, plusieurs domaines d'activités y sont exercés :

- R&D (recherche et développement) ;
- Tests et Validations ;
- Opérations et activités de support ;
- Direction, ressources humaines et activités commerciales ;
- Plateformes et logistique ;
- 3S Photonics (ex-Alcatel Optronics)

Plusieurs bâtiments de grandes superficies sont consacrés aux plateformes de tests et validations, où sont concentrés des milliers de téléphones mobiles à des fins de tests 3G, 4G et toutes autres technologies relatives au matériel mobile et réseaux.

4 – Le plan de restructuration

Très médiatisé depuis son annonce, Alcatel-Lucent vit actuellement un important plan de restructuration. Annoncé en juillet 2012, les principaux objectifs sont de recentraliser les activités du groupe et d'économiser un milliard d'euros annuellement d'ici à 2015. Ce plan fait suite à la confrontation du groupe à une baisse des investissements de ses clients opérateurs en télécommunication, ainsi qu'à la concurrence de plus en plus féroces des équipementiers chinois, qui proposent des produits à prix très attractifs.

En 2012, Alcatel-Lucent avait annoncé la suppression de plus de 1200 postes en France. Fin 2013, ce chiffre a été révisé à près de 900 postes dont notamment :

- 509 (sur 3277) à Nozay (Essonne) ;
- 128 (sur 483) à Orvault (Loire-Atlantique) ;
- 62 à Rennes (Ille-et-Vilaine) ;
- 61 à Ormes (Loiret) ;
- 56 à Lannion ;
- 28 à Toulouse ;
- En Ile-de-France, 37 postes seraient aussi supprimés au siège à Paris, selon le calcul de la CFDT.

A ces 900 postes, il faut ajouter plus de 900 autres suppressions postes, par des redéploiements internes et externes (cessions de sites et mobilités internes). [ref. 5] Au niveau mondial, ce sont plus de 10 000 postes qui sont impactés. De nombreux postes d'ingénieurs, notamment sur les activités 4G, sont impactés par le plan Shift au profit d'Altran (société de prestations).

Conséquence directe du « Plan Performance », les équipes présentes sur le site de Vélizy ont été contraintes d'être relocalisées sur le site de Villarceaux quelques kilomètres plus au sud, soit près de 2000 personnes. Au total, plus de 6 sites[ref.6] sont sous le coup d'une fermeture, au niveau des filiales ALF (Alcatel-Lucent France), ALUI (Alcatel-Lucent International) et ALBLF (Alcatel-Lucent Bell Labs France) : le groupe souhaitant focaliser la majorité de ses effectifs Français sur ses sites phares de Villarceaux et de Lannion.[ref. 7]

Plusieurs activités vont être pleinement délocalisées : c'est notamment le cas des tests 2G qui à terme seront réalisés en Inde ; tandis que d'autres activités seront cédées à des entreprises tierces.

Lors de l'Assemblée générale du groupe qui s'est tenue le 28 mai 2014, Michel Combes est revenu sur les premières avancées du plan Shift avec notamment : 363 millions d'euros d'économie en 2013, une amélioration de 3,9 point de la marge brute et enfin un effort renforcé au niveau des investissements de R&D sur les technologies d'avenir[ref 8]

Malgré ces plans de restructuration et de licenciements, Alcatel-Lucent recrutera tout de même près de 200 jeunes ingénieurs à l'horizon 2015, et propose également quelques opportunités en interne à ses employés (principalement des postes vacants qui n'étaient à la base pas impactés par le plan de licenciement).

Missions confiées et réalisées

2

II. – MISSIONS CONFIEES ET REALISEES

1 – Contexte industriel et enjeux

C'est entre la fin des années 80 et le début des années 90 que la téléphonie cellulaire a commencée à se répandre dans de nombreux foyers des pays développés : la technologie ayant bénéficié de l'évolution des composants électroniques, principalement de leur miniaturisation. Chacun pouvait ainsi passer des appels de n'importe quel endroit où le réseau le lui permettait. Nous assistions également à l'implantation de nombreuses antennes et antennes relais, couvrant peu à peu la quasi-totalité des territoires habités.

Image [5] : Evolution des téléphones cellulaires sur une période de 15 ans.

Les constructeurs participent aujourd'hui encore activement à cette course effrénée pour l'évolution des téléphones mobiles. Bien que la miniaturisation soit encore en ligne de mire, notamment pour embarquer davantage de périphériques et des processeurs gravés de plus en plus fins, l'accent est principalement mit sur le perfectionnement et la conception de nouvelles normes de transmission.

Les smartphones d'aujourd'hui (entendez « téléphones intelligents ») offrent un panel de fonctionnalités bien garnies. Il est possible de consulter ses mails, d'envoyer des messages multimédias, d'accéder à internet, de regarder la télévision en streaming … Et d'autres s'ajouteront très certainement à la liste dans le futur. On comprend alors la nécessité d'améliorer toute la technologie réseau qui se cache derrière ces écrans de poche, en sachant également que le nombre de téléphone actif s'accroit jour après jours.

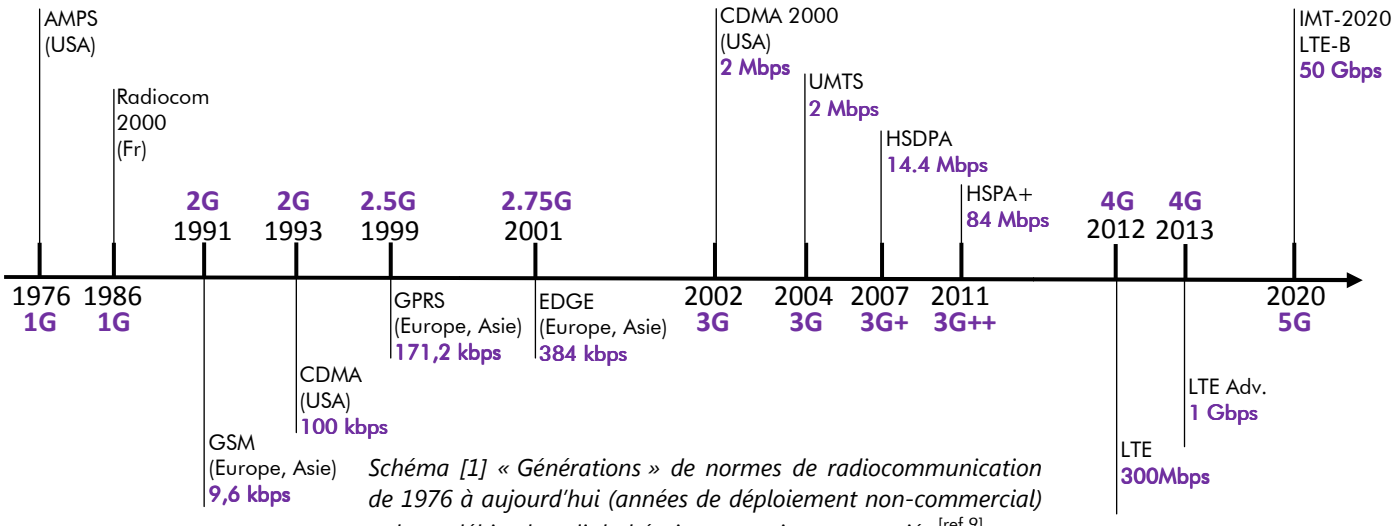
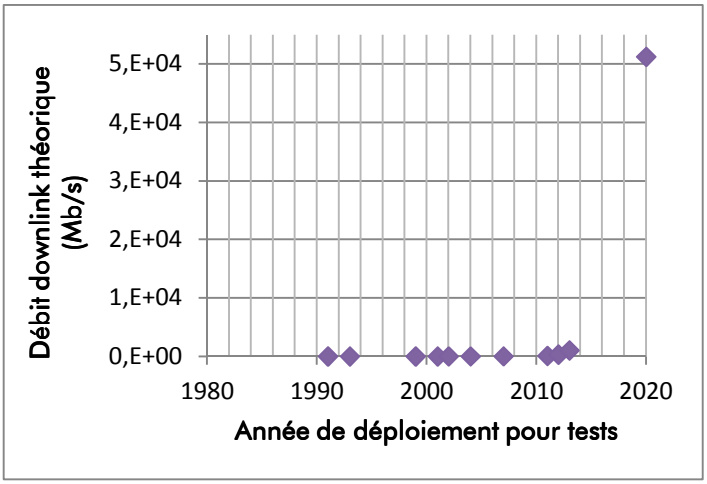


Schéma [1] « Générations » de normes de radiocommunication de 1976 à aujourd'hui (années de déploiement non-commercial) et leurs débits downlink théoriques maximaux associés [ref.9]

Pour apprécier cette saisissante évolution, on peut s'intéresser au déploiement des normes de radiocommunication successivement apparues (cf. schéma [1] page précédente) et notamment en termes de débits downlink qu'elles proposent pour le transfert de données sur le réseau. (Notons que les normes GPRS [1], EDGE [1] et LTE [1] ne proposent que la possibilité de transférer des données et non pas d'appels vocaux). On prévoit le déploiement de la 5G pour l'horizon 2020.



Graph [2] : débits downlink théoriques en Mbits/s en fonction de l'année de déploiement de la génération associée.

Le constat est davantage frappant lorsque l'on trace ces débits en fonction de l'année de déploiement à usage non-commercial de la technologie.

On remarque clairement que l'évolution est exponentielle en termes de débits downlink, permettant ainsi aux utilisateurs de télécharger toujours plus vite des fichiers de plus en plus volumineux. (cf. graph [2] ci-contre)

Toutes ces avancées technologiques sont le fruit d'années de recherches et de développement. Mais une des phases les plus importantes avant le déploiement grand public est la phase de tests. En effet, il faut nécessairement valider le bon fonctionnement de la technologie, de vérifier si celle-ci sera bien supportée par le matériel réseau, si elle sera capable de faire ses preuves face à une quantité importante de trafic, etc. Ces expérimentations restent importantes en post-déploiement afin de poursuivre l'optimisation de logiciels et de matériel physique.

Alcatel-Lucent, faisant partie des leaders dans le secteur des télécommunications, est naturellement en charge de réaliser ses propres innovations, mais également de tester les technologies émergentes (telles que la 4G LTE) qui font partie des normes définies par l'Union Internationale des Télécommunications.

Sur le site de Villarceaux notamment, Alcatel-Lucent dispose de plusieurs grandes plateformes de tests organisées suivant les générations de technologie. Plusieurs milliers de téléphones physiques et datacards[2] sont utilisés pour les tests 3G, disposés en baies découvertes ou dans d'imposantes cages de Faraday. Il en va de même pour la technologie 4G, qui tend à s'accroitre au vu du déploiement de cette nouvelle génération en France. (La technologie 2G est toujours testée, mais les plateformes sont désormais localisées en Inde). En plus de ça, les testeurs disposent de machines permettant de simuler des téléphones par émulation[3] logicielle. Tous ces dispositifs permettent de réaliser deux types de tests :

- **Des tests fonctionnels**, sur une faible quantité de mobiles, permettant de vérifier et de valider le fonctionnement de certaines fonctions précises ;

- **Des tests d'endurance/capacité**, sur une grande quantité de mobiles, permettant de tester les capacités matérielles et logicielles dans des conditions extrêmes.

[1] GPRS : General Radio Packet Service – EDGE : Enhanced Data Rates for GSM Evolution – LTE : Long Term Evolution
[2] datacard : périphérique USB permettant un accès internet par le réseau mobile sur la machine sur laquelle il est branché
[3] emulation : simulation d'un environnement système au sein d'un autre environnement

Depuis 2007, on assiste à l'émergence d'une nouvelle gamme de téléphones dit « intelligents ». Ces téléphones évolués sont pour la plupart équipés d'un assistant numérique personnel, d'un appareil photo numérique, d'un écran tactile et d'un ordinateur embarqué. La quasi-totalité des constructeurs ont su convaincre le secteur de la téléphonie mobile : à ce jour, chacun propose une gamme composée presque intégralement de ces bijoux technologiques. Mais certains acteurs ont su tirer partie autrement de ce marché en développant un système d'exploitation mobile. C'est notamment le cas d'Apple et de son système iOS, de Windows et de son Windows Phone, ainsi que de Google avec Android, pour ne citer que les principaux.

Plusieurs problèmes se posent avec ces téléphones :

- L'architecture matérielle est désormais bien plus complexe que pour les « anciens téléphones mobiles », et est généralement différente pour chaque constructeur ;

- Ils sont équipés de différents systèmes d'exploitation, et au sein d'un même système d'exploitation, on rencontre plusieurs dizaines de versions différentes (c'est le cas pour Android, iOS et Windows Phone) ;

- Chaque constructeur peut imposer ses propres restrictions et protections sur le matériel et le logiciel qu'il fourni.

Couplés, ces différents problèmes sont une véritable plaie pour les testeurs, car ils rendent incompatibles les téléphones embarquant les dernières technologies (i.e. la 4G) avec les logiciels utilisés.

Le problème est d'autant plus important qu'en 2011, le point d'inflexion a été franchi : le volume des ventes de smartphones a dépassé celui des téléphones classiques. Au premier trimestre 2014, ce sont près de 281.5 millions de smartphones qui ont été écoulés dans le monde, tous constructeurs confondus, pour quelques 120 millions de téléphones portables « basiques » :

Constructeur	Ventes au 1Q 13	Ventes au 1Q 14	Progression sur l'année
Samsung	69,7	85	+ 22%
Apple	37,4	43,7	+ 17%
LG	10,3	12,3	+ 19%
Huawei	9,9	13,7	+ 38%
Lenovo	7,9	12,9	+ 63%
Others	78,8	113,9	+ 45%
Total	216,2	281,5	+ 30%

Tableau [1] : comparatif du volume des ventes de smartphones par constructeurs en million d'unités entre le 1Q 2013 et le 1Q 2014 (source : ZDnet chiffres clés, 25 juin 2014) [ref.10]

Restons dans les chiffres. Officiellement, le volume de smartphones sous Android enregistrés sur le réseau a largement dépassé celui d'iPhones sous iOS. En juin 2014, ils représentent plus de 80% des smartphones actifs dans le monde :

Système	Q1 2013	Q1 2014
Android	75%	81.1%
iOS	17.3%	15.2%
RIM	2.9%	0,5%
Windows	3.2%	2.7%
Symbian	0,50%	0,10%
Other	1.1%	0,40%

Tableau [2] : comparatif du volume des smartphones actifs par système d'exploitation (source : Kantar Worldpanel, 12 Juin 2014) [ref.11]

Android est donc le système d'exploitation le plus largement répandu. L'avantage est qu'Android est un système open-source, qui est porté sur une grande diversité de smartphones par de nombreux constructeurs. Nous pouvons ainsi tirer les deux conclusions suivantes :

- Pouvoir effectuer des tests à base de mobiles Android est désormais primordial car, au vu des précédents chiffres, ils reflètent en majorité le parc de téléphonie mobile dans le monde, en plus d'embarquer les dernières technologies du marché ;

- La diversité des constructeurs et des modèles permettrait de piloter des mobiles récents aux caractéristiques différentes.

Depuis 2009, Alcatel-Lucent a choisi de développer en interne sa propre solution logicielle « AIDA » (qui sera détaillée ci-après) pour automatiser ses tests mobiles et réseaux, pour plusieurs raisons :

– **Réduire les coûts** : il existe sur le marché de nombreux logiciels permettant d'effectuer des tests sur les mobiles, mais dont les licences coûtent parfois plusieurs centaines de milliers d'euros annuellement ;

– **Proposer une solution tout-en-un** : tous les différents tests nécessaires aux unités de validation peuvent être réalisés au sein d'un seul et même outil centralisé ;

– **Apporter souplesse et garantie** : les clients (internes à Alcatel-Lucent) peuvent demander de nouvelles fonctionnalités, et ont un accès direct au support du logiciel pour la résolution de leurs problèmes.

Pour résumer, l'intégration des mobiles Android dans la solution logicielle « AIDA » a donc un enjeu important pour Alcatel-Lucent : le groupe s'affranchi de l'achat de nouvelles licences auprès d'autres sociétés ; la solution « AIDA » garde sa notion de tout-en-un ; et la souplesse est conservée, les clients internes pouvant décider des features[1] à implémenter si besoin.

Le travail que j'ai réalisé au cours de mes deux années d'apprentissage a ainsi consisté à apporter les solutions nécessaires pour pouvoir répondre à cet enjeu majeur.

[1] feature : fonctionnalité

2 – La solution logicielle « AIDA »

Située au cœur des équipes « System Test », l'équipe de développement AIDA, composée d'une dizaine de personnes, est en charge de développer, d'intégrer et déployer la solution logicielle AIDA (*Air Interface Device Assistant*) pour le pilotage de mobiles, datacards et smartphones. Cette solution est utilisée en interne à Alcatel-Lucent et notamment sur le site même de Villarceaux sur les plateformes Tests et Validations Systèmes et Fonctionnelles, mais également dans le monde, par exemple aux Etats-Unis et en Inde.

Ecrite en Java 1.6 et compatible Windows XP/7 et Debian, AIDA est le fruit de la migration de trois précédents outils en un seul en 2010, ce qui en fait un projet récent mais néanmoins déjà relativement robuste. Utilisé, mais toujours en cours de développement, il supporte les technologies réseau GSM, UMTS et LTE (et ainsi les générations 2G/3G/4G).

Du point de vue utilisateur, AIDA permet de contrôler et de communiquer avec des mobiles branchés sur l'ordinateur dont on a ouvert pour chacun d'eux (voir schéma [1]), principalement :

- **un port modem**, permettant d'établir une communication avec eux par commandes Hayes AT[1] ;
- une « **network interface** » (interface réseau) permettant d'exécuter du trafic entre le PC (client) et un media server (serveur multimédia) en passant par le réseau du mobile sur lequel il est attaché ;
- un **port de diagnostic** sur certains mobiles, permettant de remonter un flux de traces réseaux en provenance du mobile.

On identifie chaque mobile par son IMSI (une suite de chiffre inhérente à la carte SIM insérée dans le mobile) qu'il est possible de récupérer en interrogeant chacun des ports modem, network interface et port diag avec des commandes spécifiques. Ainsi, AIDA est capable, sur plusieurs mobiles en simultané, de :

- Emmètre des appels vocaux, envoyer des SMS... (par commandes AT) ;
- Effectuer du trafic FTP, HTTP, UDP, ICMP... (upload, download, par network interface) ;
- Récupérer les traces réseau côté mobile (par port diagnostic).

De cette manière, AIDA permet de générer un trafic défini sur la radio de la même manière qu'un réseau déployé, et récupère la série de statistiques et de métriques associée, comprenant :

- Une notion de statuts, d'erreurs, de succès ;
- Une notion de débits moyens, instantanés, de quantité de paquets transmis/émis ;
- Une notion de progression de transfert, etc.

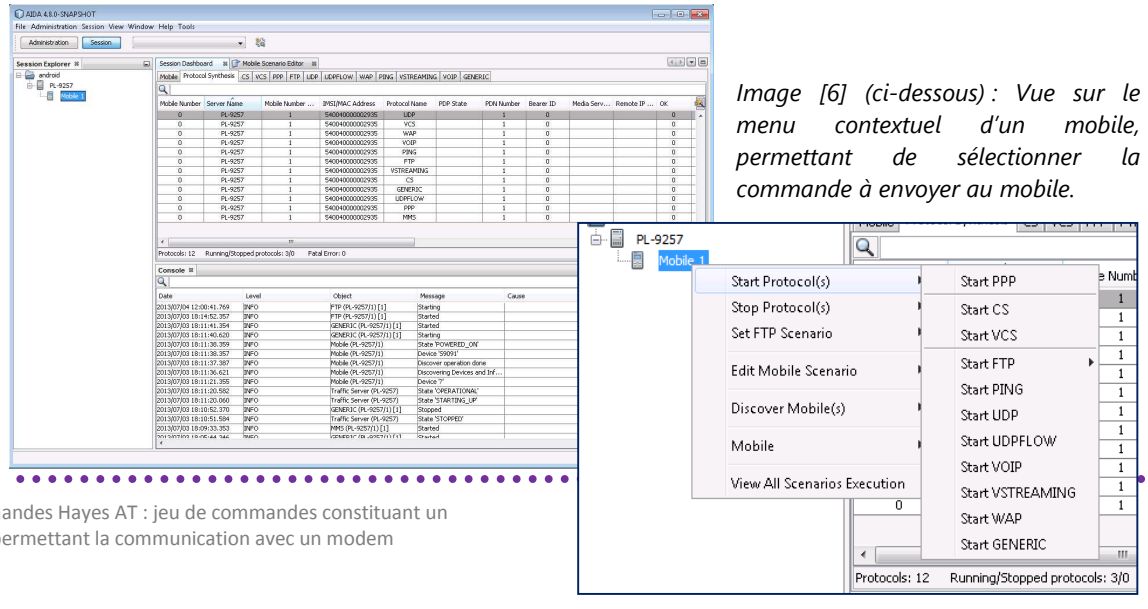


Image [6] (ci-dessous) : Vue sur le menu contextuel d'un mobile, permettant de sélectionner la commande à envoyer au mobile.

[1] commandes Hayes AT : jeu de commandes constituant un langage permettant la communication avec un modem

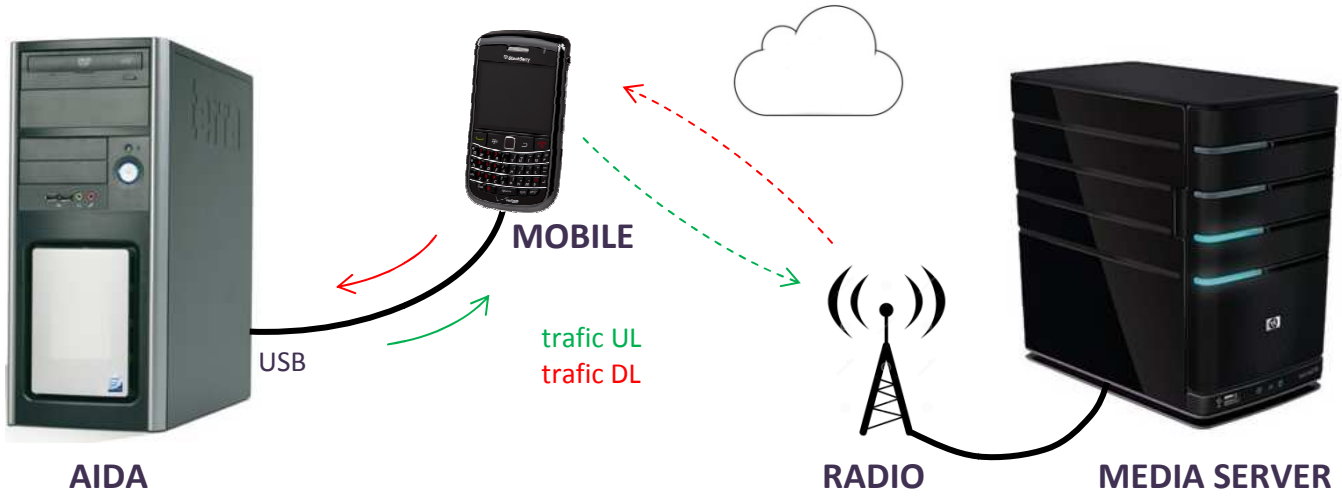


Schéma [1] (ci-dessus) : schéma de fonctionnement global d'AIDA (en standalone (un seul PC client), avec un seul mobile) Selon les périphériques du PC il est possible d'y connecter jusqu'une dizaine de mobile. En configuration distribuée, un PC maître pilotera par le réseau plusieurs PC clients, et donc autant de fois plus de mobiles.

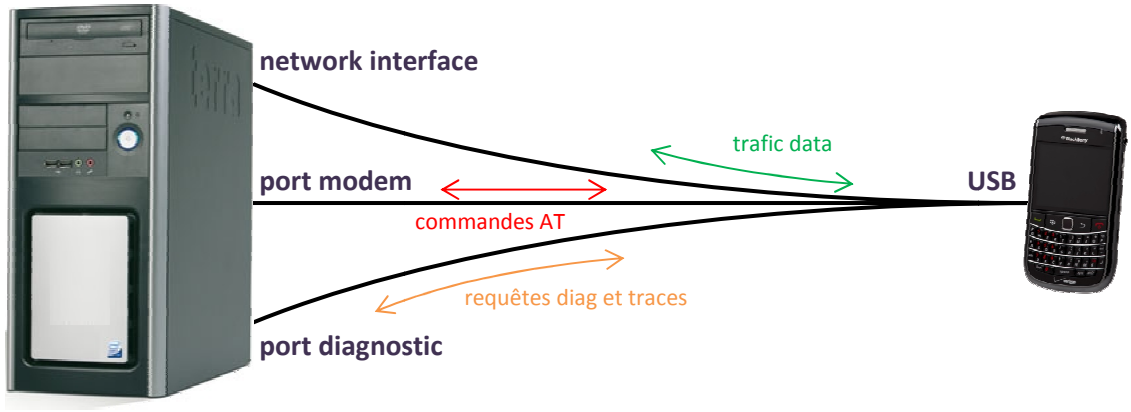


Schéma [2] (ci-dessus) : schéma des interfaces entre AIDA et un mobile donné. Ces ports sont tous ouverts sur le PC par le biais d'un seul et même câble USB reliant le mobile à l'ordinateur, tant que les pilotes sont installés correctement.

L'équipe de développement AIDA, dans laquelle j'ai donc été intégré pendant mon apprentissage, assure non seulement le développement de l'outil et la gestion des retours/requêtes des utilisateurs, mais a également la charge d'effectuer les tests fonctionnels et tests de validation nécessaires avant de fournir une nouvelle version aux clients. Pour ce faire, l'équipe dispose de trois plateformes de tests :

- **Une première** permettant de lancer des tests de capacité/endurance, basée sur environ 70 à 80 mobiles physiques WCDMA[1] ;
- **Une seconde** permettant de lancer des runs de capacité intense et de vérifier les comportements des logiciels dans des conditions extrêmes ;
- **Une troisième** permettant de lancer des tests de non régression sur chaque version du logiciel, basée sur un réseau « live ».

En plus de ces plateformes, l'équipe peut avoir ponctuellement accès à d'autres plateformes de tests, notamment pour des fonctionnalités propres au LTE.

Deux membres de l'équipe sont en permanence en plateforme pour réaliser les tests, tandis que tous les autres sont en bureau afin d'assurer le développement, le support, et la communication avec les clients. Toutefois, chaque membre peut aller en plateforme pour tester une fonctionnalité, ou pour consulter les fichiers de log [2] par exemple.

[1] WCDMA : Wideband Code Division Multiple Access ; « Multiplexage par code à large bande »
[2] log : fichier enregistré sur disque, contenant les détails de l'exécution d'un logiciel par exemple

3 – Réalisations et résultats

Au cours de mes deux ans d'apprentissage au sein de l'équipe AIDA, il m'a été confié plusieurs projets de développement relatifs aux mobiles Android. Ce présent chapitre décrit les principales applications que j'ai été amené à réaliser :

- L'application **Android « OTA »** d'automatisation des tests AIDA sur les mobiles Android ;
- L'application **Android native « On-device DIAG »** d'analyse de traces réseaux en temps réel à même le mobile ;
- L'application **JAVA « OTA Manager »** permettant de monitorer « OTA » en s'affranchissant d'AIDA ;
- La rédaction de **documents** d'architecture, d'interfaces et d'utilisation.

3.1 – L'application Android « OTA »

Il est aisé de comprendre l'enjeu d'intégrer les smartphones sous Android à la solution AIDA, tel que je l'ai expliqué précédemment. D'un point de vue technique, il y a de nombreuses contraintes faisant que cela nécessite une implémentation particulière :

- De moins en moins de modèles de smartphones sous Android permettent la remontée de ports modem ou d'interface réseau sur le PC. Souvent parce que le constructeur a fait le choix de ne pas développer de pilote officiel, ou bien que la fonctionnalité n'est tout simplement matériellement pas implémentée ;
- Lorsqu'il est quand même possible de remonter ces ports, les pilotes ne sont pas forcément cross-plateform[1] (un pilote par système d'exploitation) ; or AIDA est développé en JAVA pour être justement portable sur n'importe quelle plateforme équipée de l'environnement JAVA ;
- Il existe une multitude de constructeurs différents, qui produisent annuellement en moyenne une dizaine de modèles chaque année, eux même déclinés en plusieurs variantes (selon les normes réseau du pays par exemple) – en parallèle Google met à disposition une nouvelle version majeure d'Android tous les ans ; il est donc difficile vis-à-vis des utilisateurs de trouver un smartphone Android directement compatible avec AIDA.

Pour ce faire, la seule solution viable qui a été trouvée, permettant de s'affranchir au possible du constructeur, du modèle de téléphone, du matériel embarqué et de la version logicielle installée sur le mobile, est de développer une application Android embarquée sur le téléphone, communicant avec AIDA par USB, mais exécutant elle-même le trafic sur le mobile. C'est-à-dire, en quelques sortes, déporter la partie « génération du trafic » d'AIDA sur le mobile, comme le décrit le schéma [3] (cf. page suivante)

[1] Cross-plateform : caractéristique d'un logiciel ou d'un code à être compilé et exécuté sur toutes les plateformes possibles (Unix, Linux, MacOs, Windows …)

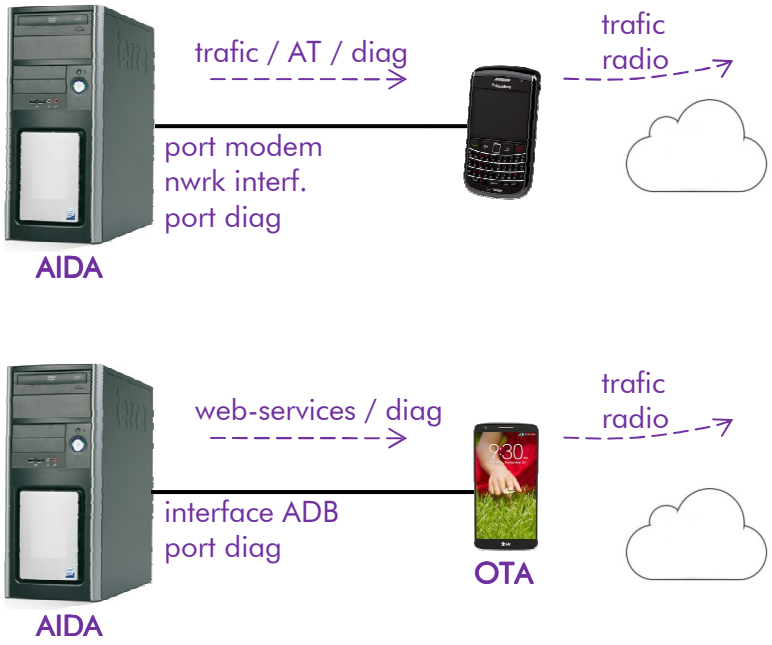


Schéma [3] : divergences de fonctionnement entre le pilotage classique et le pilotage de mobiles Android

A mon arrivée au sein de l'équipe AIDA, il n'y avait pas réellement de cahier des charges défini ; toutefois cette solution semblait être la plus simple et la plus en correspondance avec les intégrations déjà existences dans AIDA ; la charge fonctionnelle la plus importante étant de faire en sorte que l'utilisation des mobiles Android avec la solution logicielle soit la plus transparente possible pour les utilisateurs. De même, aucune dead-line[1] n'était imposée ; dans le sens où l'intégration des téléphones Android était perçue comme une grosse plus-value par les utilisateurs, mais sans que cela soit imposé à l'équipe de développement.

Deux prestataires d'Alten sont intervenus en début de projet pour tenter d'apporter une première solution interne à Alcatel-Lucent (indépendante d'AIDA) calquée sur le mode de communication décrit précédemment. Etait ainsi disponible : un service Android (application en tâche de fond en permanence) avec un serveur de web-services, communicant avec le PC par le réseau WIFI. Le nom de l'application, « OTA : Over The Air[2] » est donc purement historique et ne reflète plus de son comportement actuel ; toutefois nous avons conservé ce nom originel.

Mon travail a ainsi consisté, dans un premier temps, à adapter cette première bride d'application au fonctionnement d'AIDA, en s'affranchissant du WIFI (le tout-USB permet d'assurer une stabilité de communication exemplaire par rapport au WIFI). La suite de ce rapport décrit ainsi toutes les tâches que j'ai du effectuer dans le cadre du projet « OTA », et notamment les charges principales suivantes :

- Permettre à AIDA de communiquer avec l'application, en utilisant la connexion USB ;
- S'affranchir totalement de l'utilisation du Wifi, qui n'a aucun intérêt pour AIDA
- Aligner au maximum le fonctionnement de l'application avec celui d'AIDA ;
- Intégrer tous les protocoles pris en charge par AIDA permettant de générer du trafic ;
- Intégrer un mécanisme de capture de statuts en temps réel pour chaque protocole ;
- Intégrer un web-service permettant à AIDA de remonter en « temps réel » ces statuts.

[1] dead-line : date butoir pour la livraison d'un projet
[2] Over the air : à travers l'air ; c'est-à-dire par les technologies de communication sans fils

3.1.1 – La communication entre OTA et AIDA

L'interface de communication entre un téléphone Android et le PC est fournie par l'outil ADB (Android Debug Bridge) du SDK[1] (kit de développement Android de Google). En effet, chaque mobile Android branché à un PC est apte à communiquer avec ADB sous réserver que l'option adéquat soit activée dans les paramètres du téléphone. Une commande « adb devices » permet de remonter un identifiant unique par mobile connecté ; on peut de ce fait détecter sa présence simplement.

Afin que le mobile prenne connaissance du trafic qui doit être exécuté et à quel instant, il a été décidé de lui envoyer des commandes par le biais de web-services : une partie cliente est développée en JAVA dans AIDA, et une partie serveur en JAVA également sur le mobile. Ces web-services permettent également de remonter toutes les informations nécessaires, inhérentes au mobile mais également quant à son état dynamique. Pour que la communication s'établisse, l'outil ADB fourni une commande « adb forward[2] » permettant de mettre en place un pipe de communication entre un port local au PC et un port local au mobile (8080, qui est le port normalisé pour les web-services).
La détection d'un mobile Android diffère donc de la détection d'un mobile classique :

Etapes	Détection des mobiles classiques	Détection des mobiles Android
1	scan des ports disponibles	commande « adb devices »
2	---	commande « adb forward »
3	commande AT+CIMI (IMSI)	web-service « getpresence » (i.e. IMSI)
4	commandes AT (caractérist. du mobile)	---
5	commandes AT (état du mobile)	---

Tableau [3] : différences de techniques de détection de mobiles classiques et Android

Par la suite, l'envoi de commandes (requêtes de trafics notamment) se fait par web-services « PUT » : on envoie un scénario de trafic au mobile, qui l'exécute en interne.

Pour qu'AIDA prenne connaissance de l'état du mobile, de l'avancement de l'exécution et des statistiques recueillies, j'ai pris soin d'intégrer de nombreux web-services « GET » : ceux-ci demandent au mobile de renvoyer telle ou telle donnée au format JSON (un format de données textuel proche du même type que le XML). Concernant notamment l'exécution du trafic, j'ai intégré des mécanismes permettant de générer des objets StatusData en temps réel en cours de trafic ; et des objets AckData en fin de chaque trafic.

- Les StatusData : un objet existe pour chaque protocole (FTPStatusData, HTTPStatusData, CallStatusData …) et contient les métriques spécifiques à celui-ci (par exemple l'état d'un appel (appelant, entrant, décroché…), les débits FTP instantanés, moyens, en DL et en UL, etc.) ;

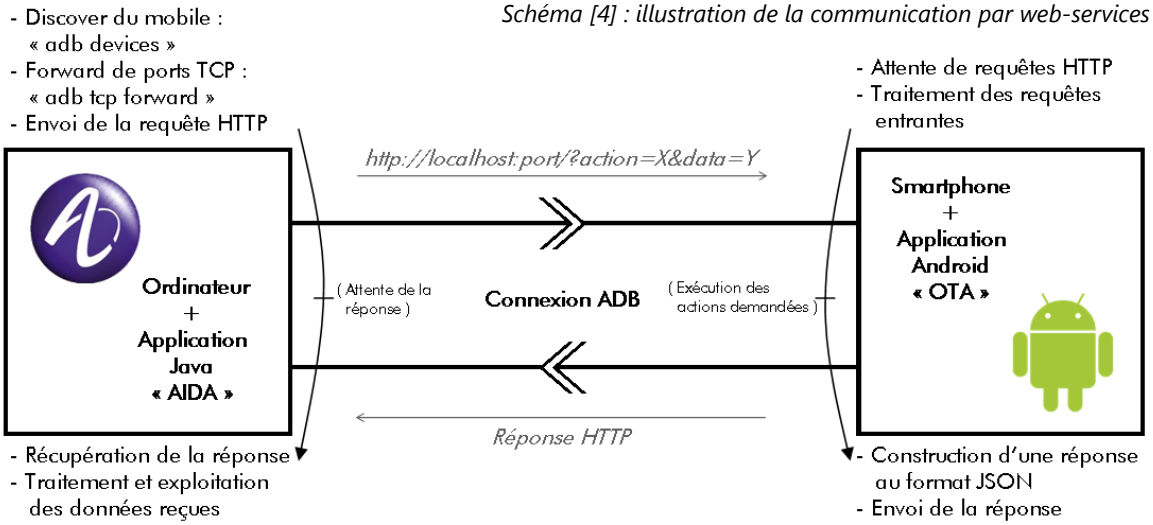
- Les AckData : c'est un objet générique qui indique si une commande spécifique s'est déroulée avec succès, ou avec une erreur particulière (un appel raccroché inopinément, un transfert incomplet, des paquets de données perdus…)

J'ai inclus ces objets dans une librairie JAVA (au format JAR), qui est partagée entre AIDA et l'application OTA : elle sert ainsi d'interface entre les deux parties. En effet, les objets StatusData et AckData sont sérialisés (représentés textuellement) au format JSON à l'aide d'une librairie dédiée. La chaîne JSON est transmise à AIDA en réponse aux web-services associés, cette dernière librairie permet de la dé-serialiser (reconstruire l'objet avec ses valeurs).

[1] SDK : Software Development Kit, contenant les interfaces et outils nécessaires au développement
[2] forward : transférer

Pour récupérer ces objets StatusData et AckData, un mécanisme de polling a été mis en place côté AIDA pour interroger OTA par web-services à intervalle régulier (la période de polling est aujourd'hui de l'ordre de 2 secondes, mais il nous est arrivé de la réduire à 400ms).

Schéma [4] : illustration de la communication par web-services



3.1.2 – L'exécution de commandes en provenance d'AIDA

Le fonctionnement d'OTA est classique en programmation : les requêtes sont récupérées par un serveur qui va en extraire les commandes (dans un protocole de communication propriétaire à l'équipe de développement (dans notre cas), mais il est également possible d'utiliser un protocole de communication standardisé).

Un processus maître (ControlService) prend les décisions associées aux commandes reçues avec l'aide d'une interface permettant d'analyser les commandes (CommandParser) ; puis un process Scheduler va ordonnancer et planifier l'exécution de celles-ci, en appelant notamment les managers de protocoles associés. La partie client permet de renvoyer une réponse directe au web-service : dans le cas des web-service « PUT » un message de succès ou d'erreur est simplement renvoyé, dans le cas des commandes « GET » une structure sérialisée JSON est renvoyée, sans qu'aucun process ne soit déclenché derrière.

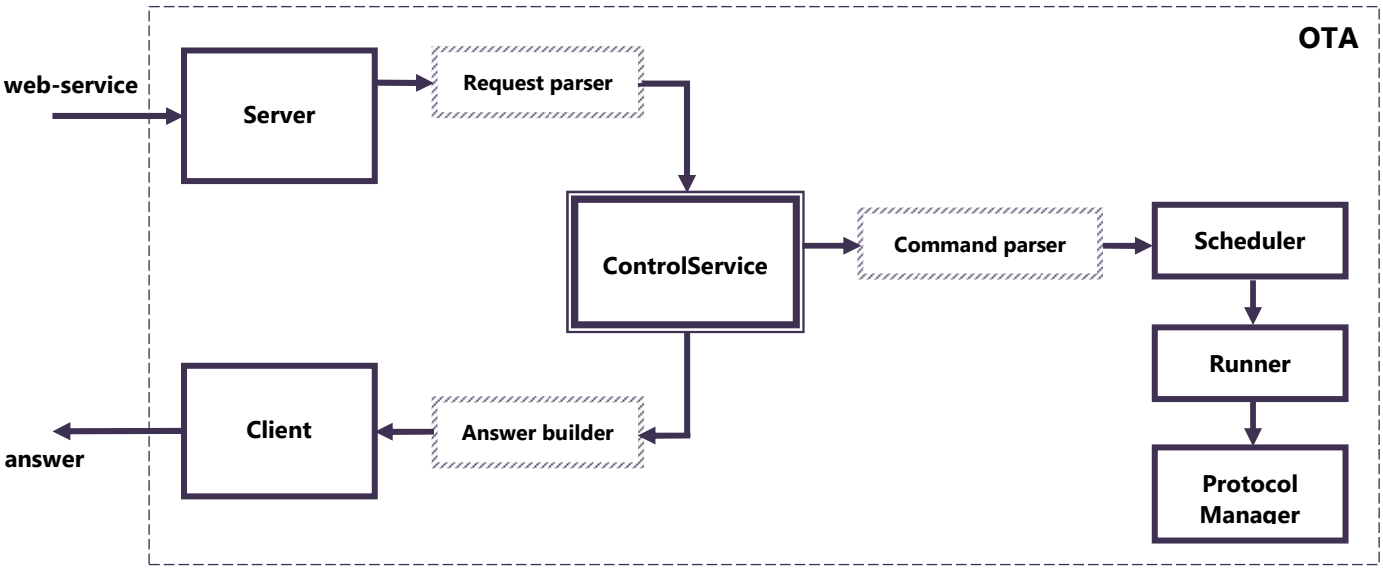


Schéma [5] : Fonctionnement global de l'application OTA.

Les commandes d'exécution de protocoles sont envoyés notamment avec l'utilisation du web-service « playscript ». Le contenu du script est bufférisé[1] et accompagne la requête :

```
localhost : port / ? action = playscript & scriptname = myscript.txt
```

Les commandes d'exécution de protocole d'AIDA ont le schéma suivant (exemples) :

PING -> « adresse_ip » : « nb_packets » : « taille_packets »
FTP_PUT -> « adresse_ip_serveur » : « login » : « mdp » : « fichier »
CALL -> « numero_téléphone » : « durée »

J'ai donc fait en sorte de calquer au maximum et au possible les commandes comprises par OTA avec les commandes AIDA de manière à utiliser les mêmes normes de communication déjà implémentés dans la solution logicielle.

Communication entre AIDA / OTA et mécanisme d'exécution des commandes de protocoles combinés correspondent ainsi au squelette de base du projet. Dans le cas, par exemple, d'une commande d'exécution de trafic FTP, voici un diagramme de séquences qui illustre ce fonctionnement de base :

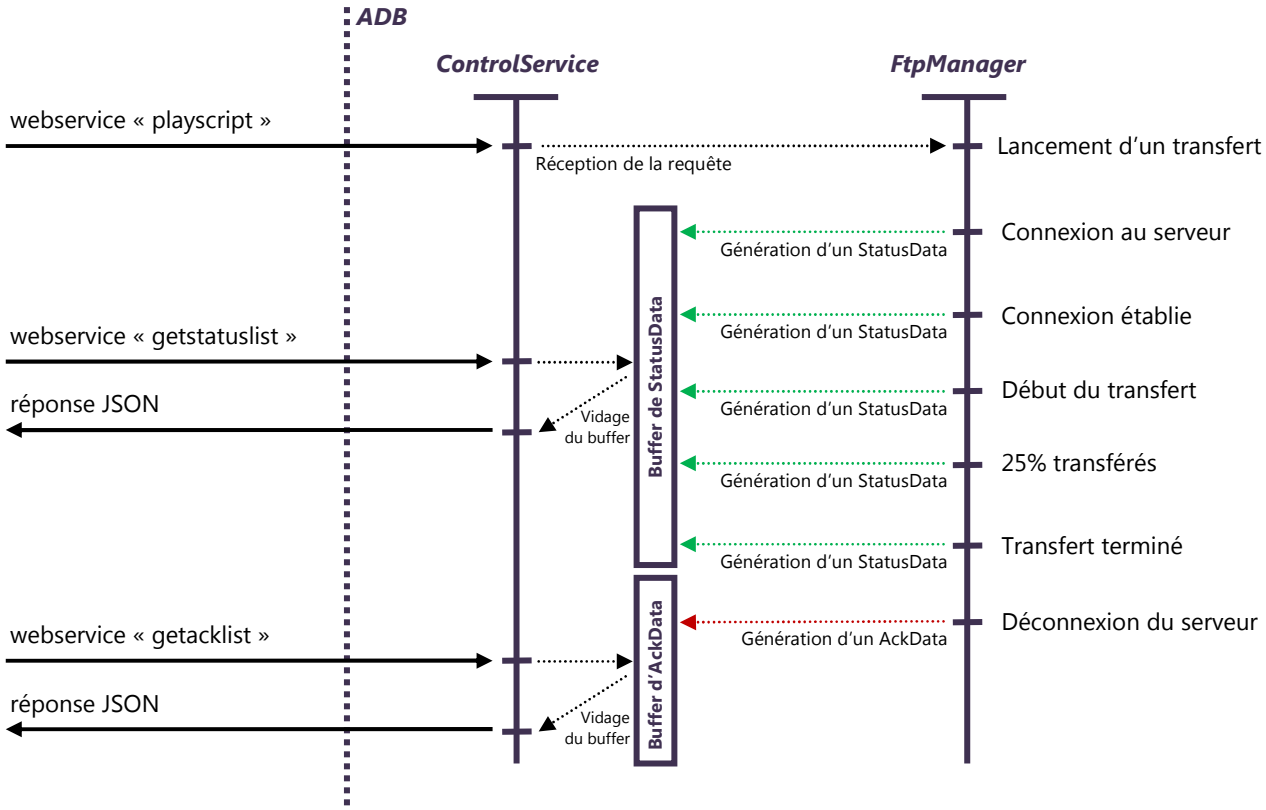


Schéma [6] : Exemple de remontée de StatusData et d'AckData dans le cas d'une commande de type « transfert FTP ».

La seconde partie du travail consiste alors en la programmation des objets Managers permettant d'exécuter le type de trafic demandé : cette étape a représenté 80% du temps de travail car chaque protocole a nécessité des intégrations spécifiques, parfois calqués sur ce qui est déjà intégré dans AIDA (c'est le cas du FTP et de l'UDP par exemple), parfois en adaptant totalement le processus en utilisant les API d'Android (c'est le cas des appels vocaux et des SMS qui ne peuvent pas être exécutés par commandes AT).

[1] buffer : structure de type « tableau » permettant de stocker une série de données du même type, le plus souvent en lien les unes à la suite des autres

3.1.3 – L'intégration des protocoles pris en charge par AIDA

AIDA propose aux utilisateurs de tester le réseau dans des conditions réelles : pour se faire, chaque protocole et technologie doit pouvoir être testé. Nous avons vu comment ces protocoles étaient exécutés sur les mobiles classiques (network interface et commandes Hayes AT). Mais OTA est en charge d'exécuter le trafic de manière autonome : d'où la nécessité de réaliser des intégrations spécifiques, en exploitant au maximum les interfaces de programmation (API[1]) proposées par le kit de développement d'Android.

Voici la liste exhaustive des protocoles pour lesquels j'ai donc du réaliser des intégrations dans OTA :

Protocole	Commande	Description
Appels vocaux	CALL	Etablir un appel vocal vers un numéro (durée spécifiée)
	SEND CALL	Etablir un appel vocal vers un numéro
	WAIT CALL	Attendre l'arrivée d'un appel vocal
	ANSWER CALL	Décrocher un appel vocal
	RELEASE CALL	Relâcher un appel vocal
	DTMF	Etablir un appel vocal suivi d'une séquence DTMF
Messages	SMS	Envoyer un SMS vers un numéro
	SMSREAD	Lire les SMS présents sur le téléphone
	SMSDEL	Effacer les SMS présents sur le téléphone
	MMS	Envoyer un MMS vers un numéro
	MMSREAD / MMS	Lire les MMS présents sur le téléphone
	MMSDEL	Effacer les MMS présents sur le téléphone
HTTP	BROWSE	Récupérer une page web (HTML + contenu multimédia)
	GET	Récupérer une page web (HTML)
	SEC_BROWSE	Récupérer une page web (HTML + c.m.) en mode sécurisé
	SEC_GET	Récupérer une page web (HTML) en mode sécurisé
FTP	GET	Télécharger un fichier depuis un serveur FTP
	PUT	Envoyer un fichier vers un serveur FTP
UDP	UDP (RAN)	Envoyer et recevoir un flux de paquets UDP (avec authentification par RAN)
	UDP (ETH)	Envoyer et recevoir un flux de paquets UDP (avec authentification par Ethernet)
PING	PING	Envoyer des paquets ICMP et recevoir leur écho
PDP	PDPACT	Activation du contexte PDP (packet data protocol)
	PDPDES	Désactivation du contexte PDP (packet data protocol)
	PDPACT_APN	Activation du contexte PDP sur un APN spécifié
NETWORK	REGNET	Enregistrement du téléphone sur le réseau mobile
	UNREGNET	Désenregistrement du téléphone sur le réseau mobile
VOIP	VOIP	Etablir un appel vocal sur IP
VSTREAMING	STREAM	Récupérer un flux RTSP (vidéo-streaming)
EMBMS	EMBMS	Piloter un client eMBMS (e-Multimediat-Broadcast-Multicast-Service)

Tableau [4] : court descriptif des protocoles implémentés dans l'application OTA

[1] API : Application Programing Interface

L'intégration de chacun de ces protocoles nécessite :

- L'implémentation de la remontée de StatusData (statistiques en cours d'exécution du protocole) et de la remontée de AckData (statut en fin d'exécution) ;
- La fiabilité des métriques (précision des débits, durées, quantités…) ;
- La robustesse à l'exécution pendant de longues durées ;
- La meilleure gestion possible des ressources ;
- La possibilité d'exécuter plusieurs instances de ce protocole en parallèle.

Intégration des protocoles « appels vocaux »

Les appels vocaux, aussi appelés « CS calls » (circuit-switching calls) peuvent être établit de manière très simple par n'importe quelle application Android, à partir du moment où elle demande au système, dans son fichier manifest, les permissions associés (obligatoires pour avoir l'autorisation d'utiliser les APIs, notamment lorsque celle-ci est destinée à être distribuée à des tiers).

L'établissement d'un appel vocal passe par la construction d'un objet Intent en JAVA. C'est un objet spécifique à la programmation Android, qui permet entre autre de demander au système le déclenchement d'une action spécifique. Cet Intent est construit en lui donnant le numéro de téléphone à appeler par le biais de l'URI suivante « tel : 0611223344 ».

Autour de ce principe, j'ai intégré tout un mécanisme permettant d'être informé en temps réel de l'état de l'appel vocal : inactif (IDLE), sonnant (RINGING) ou établit (OFFHOOK) par l'utilisation du fournisseur d'état « PhoneStateListener », ainsi que le process permettant de temporiser l'appel si besoin.

L'établissement d'un appel vocal suivit d'une séquence DTMF exploite le même système d'Intent, il suffit de préciser lors de sa construction l'URI[1] normalisée suivante :
« tel : 0611223344,9,,5 » (le numéro appelé sera ainsi le 0611223344, une seconde s'écoulera, puis la séquence « 9 – pause – pause – 5 » sera exécutée). Ce protocole est notamment utilisé par les testeurs lorsqu'il s'agit de contacter des boîtes vocales, messageries, etc.

Les commandes permettant de répondre à un appel entrant ou pour relâcher un appel établi sont plus compliquées à implémenter : en effet Google ne fourni aucune interface de programmation pour ce faire ; il convient alors de faire appel à des services plus bas-niveau en appelant des librairies natives (answer call) ou à des commandes systèmes (release call).

Intégration des protocoles « messages »

De la même manière que pour les appels vocaux, l'envoi d'un SMS se fait par l'utilisation d'un Intent auquel on passe le numéro de téléphone cible par l'URI « sms : 0611223344 » ainsi que le corps du message à envoyer. Il est possible de monitorer l'envoi effectif du message, ainsi que l'accusé de réception par l'utilisation des fournisseurs d'états SENT et RECEIVED. A nouveau, une permission spécifique doit être ajoutée au fichier manifest pour autoriser OTA à envoyer des SMS.

[1] URI : Uniform Resource Identifier – c'est-à-dire une courte chaîne de caractères permettant d'identifier une ressources par son type et par son emplacement dans le système

L'envoi des MMS (Multimedia Message Service) a été quant à lui beaucoup plus compliqué : en effet Google ne fourni aucune interface. Il a donc fallu que je développe un client MMS capable de communiquer par le protocole HTTP avec un « MMS Center », dont il est possible de définir l'adresse IP et le port dans un APN (Access Point Name) spécifique à l'envoi et la réception de MMS au niveau du mobile.

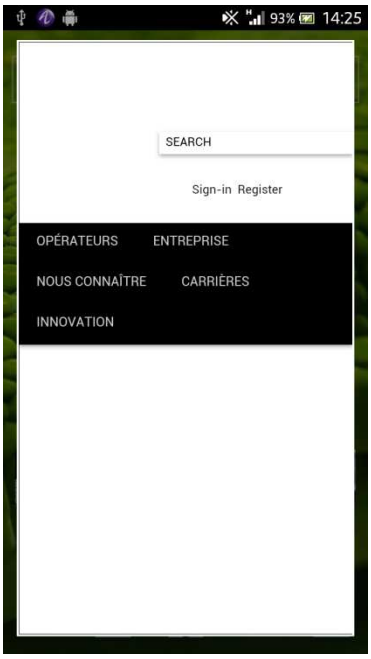
Toutefois, cette implémentation se limite à l'utilisation d'un « MMS Center » personnel, car il n'est pas possible de développer aisément une application permettant l'envoi de MMS sur le réseau réel des opérateurs par soucis de sécurité (il faudrait que notre application ait l'autorisation de la part des opérateurs d'émettre des MMS sur le réseau pour que cela soit fonctionnel). Néanmoins cela n'est pas bloquant, car il est facile pour les testeurs de déployer leur propre « MMS Center ».

Intégration des protocoles « HTTP »

A l'époque de la technologie 2G, le protocole HTTP était redéfini par le protocole WAP (Wireless Application Protocol) et permettait aux appareils mobiles d'accéder à Internet par le biais d'une passerelle, les données des pages étant transmises en WML (Wireless Markup Language, un langage à balise spécifiquement développé pour afficher les pages web sur les écrans des mobiles).

Aujourd'hui, tous les téléphones Android sont directement connectés à Internet, et utilisent des navigateurs pour accéder aux sites web, transmis cette fois-ci en HTML (Hypertext Markup Language) par le protocole HTTP (Hypertext Transfer Protocol).

Google fourni une API permettant non seulement de réaliser des transferts HTTP, mais également d'afficher à l'écran du mobile la page web téléchargée (lorsqu'il s'agit d'une page HTML et non pas d'un fichier, auquel cas celui-ci est automatiquement téléchargé dans la mémoire du téléphone). Encore une fois, la requête s'effectue par la construction d'un Intent auquel on passe l'URI « http://mon-site-web.fr ». Cette API offre également la possibilité de ne récupérer que le code HTML de la page (squelette) ou bien le HTML + le contenu multimédia (images, icônes…)



J'ai utilisé à nouveau une interface permettant de monitorer le taux de chargement de la page. Il est ainsi possible de monitorer également le débit de transmission en temporisant ce taux de chargement.

Pour permettre d'accéder à Internet par le biais du réseau, il est primordial d'ajouter la permission dans le fichier manifest.

Image [7] : Exemple d'un http BROWSE exécuté en spécifiant l'adresse du site web d'Alcatel-Lucent. Les images et le code html sont téléchargés à 100%.

Image [8] : Exemple d'un http GET exécuté en spécifiant l'adresse du site web d'Alcatel-Lucent. Le code html est téléchargé à 100% mais le contenu multimédia n'est pas téléchargé.

Intégration du protocole « FTP »

Cette fois-ci, Google ne fourni pas d'interface native permettant de développer un client FTP. Toutefois, l'Apache Foundation propose une série de librairies communes en JAVA, totalement Open Source, contenant notamment tous les outils nécessaires pour développer son propre client FTP.

Ainsi, j'ai donc développé ce client, dont la création se fait suivant deux contextes distincts :

- PUT : envoi d'un fichier local au mobile, présent sur sa mémoire externe (SDCARD) vers le serveur FTP, dans un dossier spécifié sur son disque ;

- GET : téléchargement d'un fichier distant présent sur le serveur FTP vers la mémoire externe du mobile (SDCARD).

Cette librairie étant Open Source, Alcatel-Lucent en a profité pour améliorer son code source en implémentant notamment un mécanisme d'écoute des paquets transmis et reçus : cette nouvelle interface permettant ainsi de monitorer la progression du transfert, me permettant donc de calculer des métriques telles que des débits moyens, instantanés, la quantité d'octets transférés/reçus, le pourcentage du transfert effectué…

La principale difficulté rencontrée a été de régler les tailles des buffers du client en émission et en réception : en effet, le débit effectif dépend directement de ces buffers, qui ne doivent être ni trop faibles (débit réduit et pertes de paquets) ni trop importants (débit réduit et pertes de connexion). Bien sûr, en fonction de la technologie testée, les débits attendus varient fortement ; c'est pourquoi il a fallut que je rende configurable les tailles des buffers par les testeurs en implémentant une nouvelle commande de configuration du client FTP.

Intégration du protocole « UDP »

Le protocole UDP (*User Datagram Protocol*) est un des principaux protocoles de communication utilisé sur Internet. Il fait partie de la couche transport de la pile protocole TCP/IP. Le rôle de celui-ci est de permettre la transmission de données de manière très simple entre deux entités, chacune étant définie par une adresse IP et un numéro de port. Contrairement au protocole TCP, qui nécessite une demande de connexion par le client et une acceptation de la connexion par le serveur avant de pouvoir échanger des données, UDP ne nécessite pas cette phase de négociation, il fonctionne en mode non-connecté.

UDP est notamment utilisé lorsque l'on souhaite envoyer rapidement une petite quantité de donnée ou bien dans les cas où la perte d'une donnée est moins importante que sa retransmission. Toutefois, UDP s'affranchi des couches de contrôle qu'embarque TCP, ce qui fait que c'est au programmeur de veiller à ce que tous les paquets soient bien transmis par le client et reçus dans l'ordre par le serveur.

Au sein de l'application OTA, UDP a nécessité une implémentation particulière afin de pouvoir échanger avec un serveur dédié au sein d'Alcatel-Lucent. AIDA propose deux modes d'utilisation du protocole UDP FLOW :

- Le mode « négociation par le RAN[1] ». Ce mode permet au client UDP de signaler une requête de transfert et de négocier lui-même à travers le RAN avec le serveur les ports

[1] RAN : Radio Access Network, partie radio d'un système de télécommunications mobiles

à utiliser pour les transferts (ports uplink local et distant et ports downlink local et distant).

- Le mode « négociation par Ethernet[1] ». AIDA dispose d'une liaison Ethernet avec le serveur UDP, dans ce mode la négociation des ports avec celui-ci se fait donc via cette liaison, avant la création du client UDP.

Sur les mobiles Android, le client UDP est embarqué au sein d'OTA. On ajoute alors un intermédiaire supplémentaire entre le client UDP et le serveur UDP, notamment dans le cas de la négociation par Ethernet (cf. schéma [7] ci-dessous).

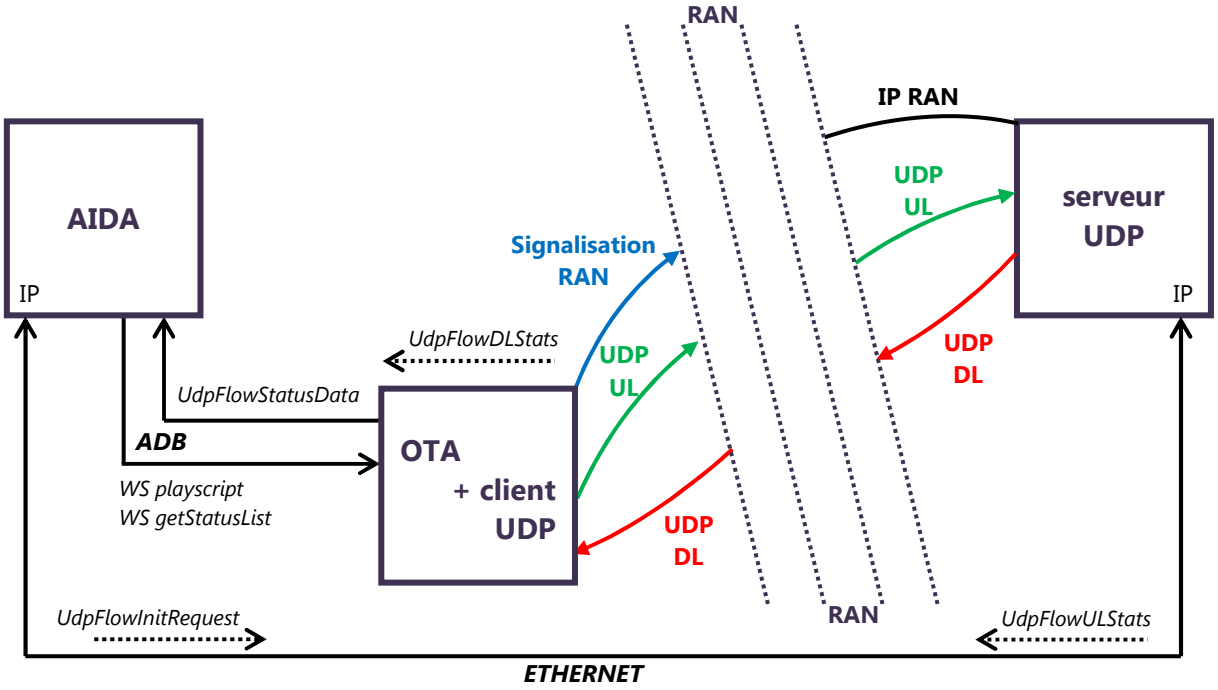


Schéma [7] : Schéma de fonctionnement de l'UDP FLOW intégré à OTA piloté par AIDA en mode ETHERNET et en mode RAN.

En mode Ethernet, AIDA doit configurer le client UDP avec une série de paramètres spécifiés par l'utilisateur (adresses IP, ports de contrôle, tailles de buffers, identificateur de client…). Il a donc fallut introduire une commande permettant d'initialiser la configuration du client UDP au sein d'OTA avant de lancer le trafic.

Une fois configuré convenablement (que ce soit en mode RAN ou Ethernet), le client peut démarrer le trafic UDP avec le serveur. Des statistiques sont remontées par le client (statistiques *downlink*[2]) et par le serveur (statistiques *uplink*[3]). On envoi à AIDA les statistiques *downlink* à l'intérieur de UdpFlowStatusData. Les statistiques uplink sont quant à elles remontées par le serveur vers AIDA en mode Ethernet, et vers OTA en mode RAN.

Bien que solution fonctionnelle, le principal souci de celle-ci est que la synchronisation des statistiques *uplink* et *downlink* en mode Ethernet est légèrement atteinte. En effet le serveur remonte les statistiques *uplink* vers AIDA en temps réel alors que les statistiques *downlink* capturées par OTA sont remontées à chaque requête web-service getstatuslist (soit toute les secondes dans le cas d'AIDA aujourd'hui). C'est pourquoi il a fallu que j'organise la resynchronisation de ces données de manière à pouvoir les confronter convenablement dans AIDA.

[1] Ethernet : protocole de communication réseau implémentant la couche physique (notamment par cordon RJ45)
[2] downlink : réception de paquets distants en provenance d'un serveur
[3] uplink : envoi de paquets locaux à destination d'un serveur

Intégration du protocole « PDP + APN »

Le « Packet Data Protocol » (PDP) est l'élément fondamental du transport des informations dans tous les réseaux d'ordinateurs et dans les réseaux de téléphonie mobile. Pour la transmission des packets contenant les données, il est indispensable d'ouvrir une passerelle appelée « Packet Data Network Gateway » sur le mobile : lors d'une utilisation classique, cette passerelle peut être activée et désactivée, de manière volontaire (pour couper tout échange de packets) ou involontaire (erreur sur le réseau ou sur le mobile, déterminée ou non).

Il est indispensable de pouvoir provoquer et monitorer l'activation et la désactivation de cette passerelle dans AIDA, et ainsi sur les mobiles Android. C'est pourquoi j'ai du implémenter les mécanismes nécessaires pour activer (PDPACT) et désactiver (PDPDES) ce contexte. Google est peu friand d'applications proposant ce genre de service (ce pour raison de sécurité) aucune API officielle n'est mise publiquement à disposition ; toutefois il est possible d'utiliser des API masquées en réalisant des projections JAVA (c'est-à-dire en construisant l'appel aux méthodes cachées par le biais de constantes plutôt qu'en réalisant des appels classiques). Toutefois cette manière de programmer impose une forte contrainte de maintenance car les API masquées sont susceptibles d'être modifiées sans préavis de la part de Google d'une version d'Android à une autre. Au cours de mes 2 années d'apprentissage, j'ai ainsi été contraint d'ajouter une nouvelle méthode à 3 reprises pour conserver l'aspect fonctionnel sur tous les mobiles, même les plus récents.

Au niveau des « Access Point Names » (APN), qui définissent les paramètres d'adresse, de port, et d'identification auprès du réseau pour obtenir l'accès internet sur le mobile, il a fallu que j'implémente la possibilité de passer d'un APN à un autre lors de l'activation du PDP. Ceci permettant à l'utilisateur d'avoir un accès internet dont la qualité de service (QoS) dépend directement de celui-ci. Toutefois, Google interdit la modification de ces APNs par le biais d'applications : pour réaliser une implémentation fonctionnelle, j'ai aussi du programmer une seconde application Android à installer dans la partie « system » d'Android, communicant avec OTA et réalisant les opérations nécessaires : étant déclarée comme application système, celle-ci obtient toutes les autorisations nécessaires. Toutefois, il faut absolument que le mobile soit « rooté » (c'est-à-dire « débridé », les couches protections ayant été retirées) pour que cette solution soit fonctionnelle.

Intégration du protocole « EMBMS »

Grosse partie du projet OTA, l'implémentation du pilotage des tests eMBMS sur les mobiles Android a été entamée au début de ma deuxième année d'apprentissage, et constitue une grosse plus-value pour AIDA : en effet, l'automatisation des tests eMBMS permet de faire gagner un temps considérables aux testeurs, et permet également de les réaliser sans supervision humaine sur des durées considérables tout en apportant un suivi des tests.

l'eMBMS (e-Multimedia-Broadcast-Multicast-Service) est un protocole spécifique aux réseaux 4G LTE : un flux multimédia (audio, vidéo, ou bien des fichiers) sont émis de manière continue sur une (ou plusieurs) cellules d'une antenne. C'est à l'utilisateur, par le biais d'un client associé, d'aller récupérer ces flux s'il le souhaite. Ainsi, eMBMS n'est pas un protocole connecté, et est basé sur UDP.

Alcatel-Lucent possède les licences d'utilisation d'une application Android cliente pour eMBMS, dont le code source est fermé car ce protocole est propriétaire. C'est pourquoi l'implémentation du protocole eMBMS n'est pas réalisable dans OTA ; la seule manière de monitorer ce type de trafic étant de piloter cette application cliente via OTA.

Ainsi, pendant de nombreuses semaines, j'ai été chargé de récolter et d'étudier la moindre information sur ce fameux client ; de manière à établir le meilleur mécanisme possible capable de réaliser des actions sur ce dernier. A force d'analyses, j'ai réussi à implémenter l'automatisation des fonctionnalités suivantes :

- démarrage du client et donc de la récupération du flux eMBMS ;
- récupération du flux vidéo (streaming) suivant ou précédent ;
- activation/désactivation de la récupération d'un fichier particulier ;
- mise en écoute des packets UDP reçus et calcul de débits en réception ;
- désactivation du client et donc de la récupération du flux eMBMS.

3.1.4 – Développement de l'interface graphique

OTA fonctionne en mode « service », c'est-à-dire en tâche de fond persistante sur le mobile. Ainsi, l'interface graphique est dispensable. Toutefois, j'ai tout de même développé une interface légère permettant aux utilisateurs d'avoir une visualisation directe de l'état de l'application et du trafic en cours d'exécution.

Les couleurs sombres, très proches du noir, ne sont pas choisies au hasard : elles permettent d'économiser au maximum la batterie du téléphone, car même en charge, celle-ci risque de se vider plus rapidement qu'elle ne charge lors de l'exécution de longues requêtes de trafic.

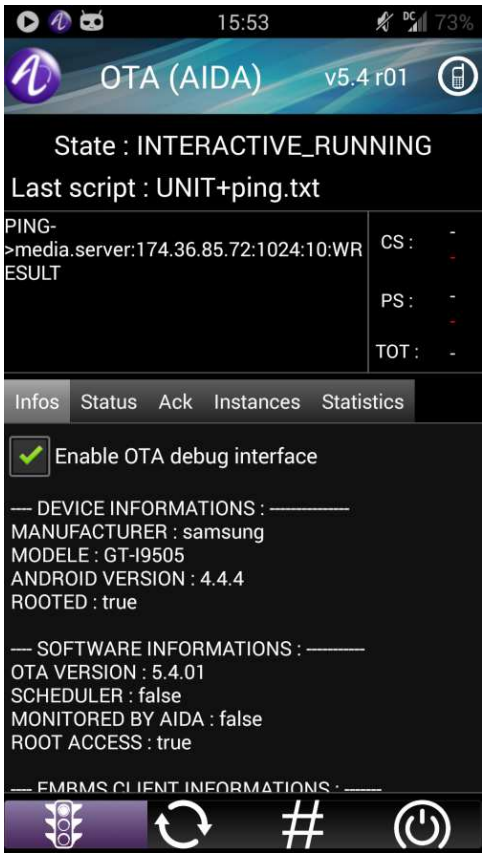


Image [9] : Interface graphique principale d'OTA, avec toutes les informations inhérentes au mobile et au réseau auquel il est connecté, ainsi que des informations quant au trafic en cours.

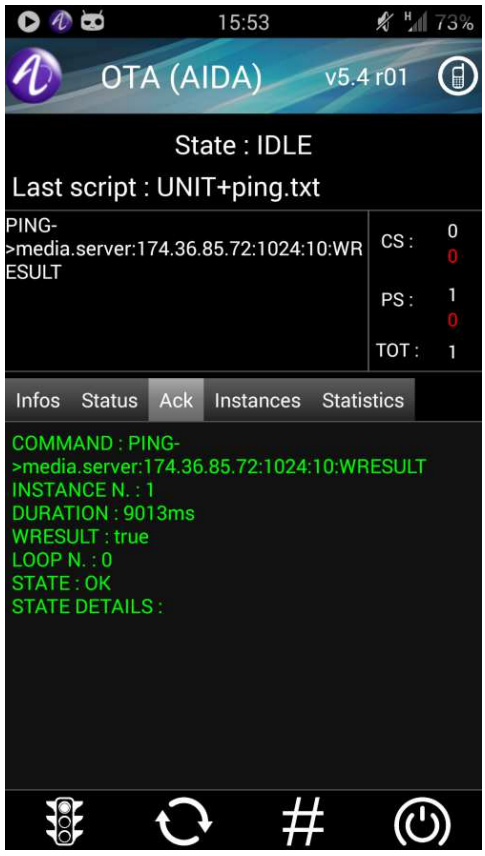


Image [10] : Interface graphique complémentaire d'OTA, avec les informations du dernier AckData généré. Le texte est en vert lorsque le trafic s'exécute avec succès, sinon en rouge.

3.2 – L'application Android native « On-device DIAG »

L'application « OTA » permet de d'apporter une solution à l'absence des ports de communication avec le mobile sur le PC (modem et network interface), remplacées par une communication via ADB. Il reste le port DIAG, commun au pilotage classique et au pilotage via « OTA » pour tous les mobiles qui supportent cette fonctionnalité, et qui permet la récupération de traces réseaux depuis le PC, extrêmement intéressantes pour les équipes de tests car elles contiennent toutes les actions précises qui ont été exécutées entre le mobile et le réseau, les erreurs éventuellement survenues, l'état des composants, connexions, communications, les débits réels, etc.

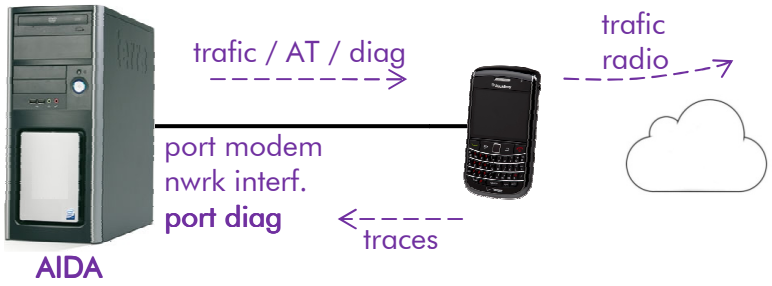
A l'heure de la 4G LTE, le débit des traces générées est de l'ordre de 30 à 50 Mo par minute. Ainsi, ce sont plus de 13Go de traces soit plusieurs centaines de millions de messages, qui sont enregistrées par mobile chaque nuit. Le travail de dépouillage de celles-ci par les testeurs est donc considérable, d'autant plus qu'il faut multiplier ces valeurs par plusieurs dizaines de mobiles ; tout en prenant en considération qu'il faut pouvoir les confronter aux traces récupérées côté cœur réseau, tout aussi nombreuses, si on souhaite diagnostiquer des erreurs ciblées.

Alcatel-Lucent voit donc un enjeu considérable dans l'automatisation de l'analyse de ces traces. Notamment, dans le cas des mobiles tournant sous Android, une solution d'application capable de récupérer ces traces réseaux à même le mobile, tout en les analysants en temps réel de manière à n'enregistrer que certaines fenêtres de traces lors de la détection d'événements particuliers, serait un produit dont la portée serait phénoménale, et je pèse mes mots. Effectivement, une telle application permettrait :

- De s'affranchir de l'utilisation d'un PC pour récupérer les traces (le mobile serait autonome) ;
- De ne récupérer qu'une faible fraction des traces, uniquement les parties intéressantes ;
-
- D'écrire des fichiers de logs permettant d'indiquer aux testeurs où chercher l'événement particulier dans les traces enregistrées ;
- De réaliser toute sorte d'actions sur le mobile à la détection d'un événement ;
- De calculer et d'afficher en temps réel toutes les métriques relatives au réseau sur l'écran même du mobile ;
- D'envoyer des commandes et des requêtes spécifiques directement au modem …

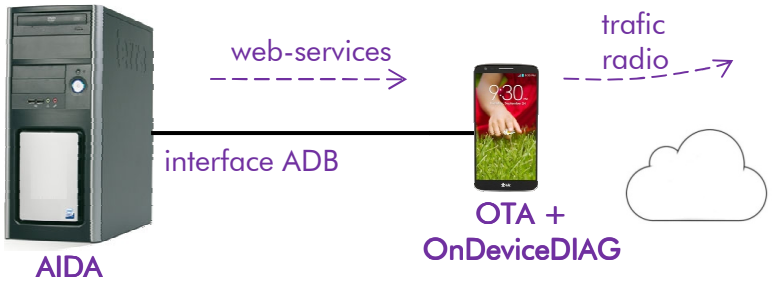
Force est de constater, donc, que les possibilités sont nombreuses. Bien entendu, certains constructeurs fournissent déjà ce genre d'outils sous forme d'applications Android, mais encore une fois, les licences ont un coup très élevé, et ces solutions apportent peu de souplesse par rapport aux besoins propres aux équipes de test d'Alcatel-Lucent.

Après de nombreuses recherches pendant ces deux dernières années, l'équipe AIDA a réussi très récemment à trouver une solution pour développer sa propre application capable de réaliser toutes les fonctionnalités que je viens d'énoncer. Alcatel-Lucent possédant toutes les licences nécessaires, il m'a été confié la conception et le développement complet de cette application.



PILOTAGE AIDA CLASSIQUE

- plusieurs interfaces par mobile
- traces réseaux récupérées sur le PC
- statistiques calculées par le PC
- plus de 13Go de traces/mobile/nuit



PILOTAGE AIDA ANDROID

- une seule interface par mobile (ADB)
- traces réseau récupérées à même le mobile
- statistiques calculées les applications Android
- quantité réduite de traces/mobile/nuit

Schéma [8] : divergences de fonctionnement entre le pilotage classique et le pilotage avec OTA+OnDevice DIAG

Le schéma 8 ci-dessus est l'évolution du schéma [3] de la page 20 : il permet de constater que, pour les mobiles Android intégrant les solutions « OTA » et « OnDeviceDIAG », il ne persiste plus que l'interface de communication « ADB » entre AIDA et le mobile. Le trafic est exécuté par OTA, tandis que les traces réseaux sont récupérées et analysées par OnDeviceDIAG.

Historiquement, j'ai conçu et développé deux versions distinctes de l'application « On-Device Diag » : une première version basée sur un programme natif fourni dans le système Android et une deuxième, plus évoluée, basée sur une librairie native plus bas niveau.

Dans l'optique de rendre publique mon mémoire d'apprentissage, tout en protégeant le secret professionnel dans lequel ce projet a évolué, je ferais volontairement abstraction de nombreux détails d'implémentations et de nommages par la suite.

3.2.1 – Conception et développement de la première version

Un module natif[1] « M », livré dans le système d'exploitation Android aussi bien sur les mobiles de test que sur les mobiles du commerce, permet de demander l'enregistrement des traces réseaux dans des fichiers à même la mémoire interne et/ou externe du mobile.

Ce module fait partie de l' « user-space » d'Android ; c'est-à-dire la partie applicative du système pour laquelle l'utilisateur a les autorisations nécessaires pour l'exécution des programmes qu'elle contient. Ainsi, il s'appui lui-même sur des interfaces propriétaires « L » bas-niveau au niveau du noyau d'Android (kernel[2]) permettant d'établir la communication avec le modem (envoi de commandes spécifiques, requêtes d'états, récupérations des traces…) tel qu'illustré sur le schéma [9] page suivante.

C'est donc par l'automatisation de l'exécution de ce module que se base la première version de l'application Android « On-Device Diag ».

[1] natif : le langage natif est le langage bas niveau compris par la machine. Dans le cas d'Android, il s'agit des langages C / C++.
[2] kernel : noyau du système d'exploitation. C'est la couche logicielle faisant le lien entre le matériel et la couche applicative utilisateur.

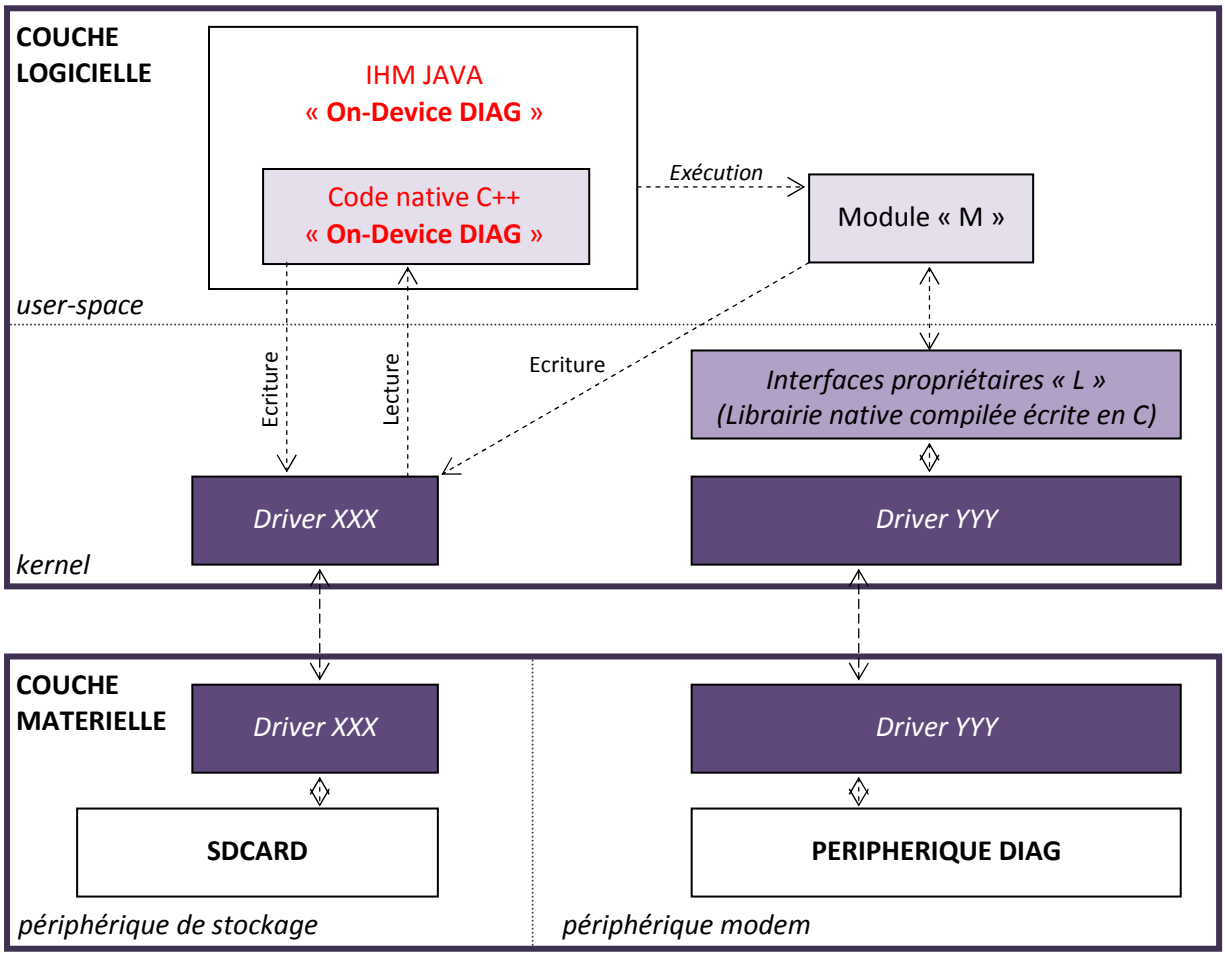


Schéma [9] : illustration des différentes couches logicielles et matérielles utilisées pour la conception de première version de l'application Android « On-Device Diag »

Exécuter une instance du module « M »

Afin de lancer la récupération des traces réseaux, il m'a fallut automatiser le lancement du module natif « M ». Une fois celui-ci démarré, il est possible de connaître son identifiant de processus (soit son PID, « process identifier ») par l'appel système « `ps` » et en analysant sa réponse. Ce PID est nécessaire pour pouvoir arrêter le processus lorsque l'utilisateur souhaite stopper la récupération des traces, par l'appel système « `kill -9 <pid>` ». J'ai du contourner les couches de sécurité d'Android pour pouvoir automatiser cette exécution ; celle-ci étant réalisé en JAVA (sujette donc aux permissions d'Android).

Récupérer les traces réseau

Le module « M » a un comportement par défaut sur les mobiles du commerce : les traces sont enregistrées dans un fichier sur la mémoire de stockage du téléphone (interne ou externe) par fichiers de 100M (voir schéma [9]); ainsi à chaque saturation du fichier diag_log_(x), le module va créer un nouveau fichier diag_log_(x+1) et va renommer le fichier diag_log_(x) avec le timestamp[1] (date et heure).

Lors de l'exécution du module « M », il est possible de visualiser dans quel fichier les traces sont écrites et à partir de quel instant, dans la sortie standard. Toutefois, j'ai été confronté au fait qu'il n'est pas possible de récupérer la sortie du module lorsqu'il est exécuté par l'application Android On-Device Diag, ceci étant du à la manière dont le module a été

[1] timestamp : dans les systèmes Linux notamment, le timestamp est une valeur entière dont la valeur correspond à la quantité de millisecondes écoulées depuis le 1 janvier 1970 à 00h00, heure officiellement reconnue comme étant le « début de l'informatique moderne »

codé. Ainsi, j'ai du implémenter en JAVA un thread[1] scrutant à intervalle régulier le dossier de destination prévu des fichiers contenant les traces et prévenant le process cœur de la disponibilité d'un nouveau fichier.

Préserver la mémoire du téléphone

Comme expliqué précédemment, le débit des traces écrites dans les fichiers est de l'ordre de 30 à 50Mo par minute. Les mémoires internes des téléphones du commerce ainsi que les cartes μSD d'extension de mémoire sont des composants extrêmement denses pour une taille très réduite : ainsi, un tel débit de lectures/écritures sur des durées de plusieurs heures n'est pas supporté par ceux-ci. Il pourrait en résulter une surchauffe, qui induirait une mort rapide du composant.

C'est pourquoi nous avons eu l'idée d'implémenter la possibilité d'écrire les traces réseaux issues du module « M » directement dans la mémoire vive du téléphone, bien plus résistante car conçue pour recevoir une quantité importante d'accès en lecture/écriture. De cette manière, j'ai développé un système en JAVA réalisant tous les appels système nécessaires pour monter le répertoire de destination des fichiers de traces en tant que « RAM DISK[2] », notamment à l'aide de l'appel « `tmpfs` » (soit « temporary file system »). Une longue étape de tests ont permit de valider le fonctionnel de ces intégrations (il a fallu absolument vérifier que les données sont réellement écrites en RAM et non pas dans la mémoire interne, dans toutes les conditions). L'appel système « `df` » permet de vérifier le point de montage en RAM, ainsi que l'espace alloué utilisé et restant.

Analyser les traces réseau en temps réel

Pour analyser les traces en temps réel, j'ai du développer un module de lecture du fichier de traces actuellement écrit par le module « M ». J'ai fait le choix de réaliser ce module en langage natif (c'est-à-dire en C++) dans un souci de performances, pour respecter au mieux le temps réel. D'où la nécessité d'utiliser le NDK d'Android (Native Development Kit) permettant l'intégration de code natif (C/C++) dans une application Android. L'interface JNI permet d'appeler ce code natif depuis le code JAVA.

La grande difficulté a consisté en la lecture en temps réel du fichier alors que celui-ci était lui-même en cours d'écriture, sans savoir à quel moment le fichier arrivera à saturation (c'est-à-dire ~100Mo, limite de taille de fichier intrinsèque au module « M »). De cette manière, j'ai conçu une architecture capable de lire le flux du fichier, de récupérer la position du curseur après lecture des données disponibles, de fermer le flux une fois arrivé à la fin des données disponibles, puis de le ré-ouvrir pour l'actualiser de manière à vérifier si, après la position du curseur enregistré, de nouvelles données sont disponibles pour l'analyse. Ce process ne s'arrêtera que sur événement : soit sur décision de l'utilisateur, ou bien lors de la disponibilité d'un nouveau fichier de trace par le module « M ».

Décrypter le contenu des traces

En suivant les documentations que nous avons pu nous procurer par le biais de licences en possession par Alcatel-Lucent, j'ai pu décrypter la structure du contenu des traces enregistrées par le module « M », toujours en temps réel. Ces traces sont écrites au format binaire, toutes les opérations sur celles-ci sont donc constitués d'appels système très bas niveau, tels que des décalages, des masques, additions, soustractions…

[1] thread : un thread est un processus s'exécutant en parallèle au processus principal qui l'a instancié
[2] RAM Disk : un RAM disk est un espace de la mémoire flash pour lequel toutes les données écrites en son sein seront en réalité écrite dans la mémoire vive de la machine. Au redémarrage de cette dernière, toutes les données sont perdues.

Avant de pouvoir analyser chaque message un à un, il faut retirer dans un premier temps l'encapsulation HDLC (High-Level Data Link Control) qui est un protocole de la couche de liaison définissant un mécanisme permettant de délimiter des trames de différents types en ajoutant un code d'erreur. Une fois les en-têtes retirés, un code CRC (de contrôle de redondance cyclique) est extrait du message brut, permettant de calculer l'intégrité de celui-ci par un algorithme que j'ai du porter en langage C++.

Une fois le message extrait, son premier octet défini un « log code » : celui-ci est important car il permet de connaître le type de message auquel nous avons à faire. Un process différent est à appliquer suivant le type du message : à l'heure actuelle seuls 5 types de messages différents sont à prendre en charge. Ainsi, j'ai implémenté les 5 méthodes de décryptage associées (log[1], debug, debug text, optimized debug text, event).

Détecter des événements spécifiques par l'analyse des messages

Prenons l'exemple du premier événement qui m'a été demandé de détecter : le « call drop ». Cet événement constitue la bête noire des testeurs : il correspond à un appel qui a été rejeté par le réseau, pour une raison X ou Y. Hors, dans les spécifications matérielles fournies dans les catalogues de produits d'Alcatel-Lucent, on retrouve un ratio « call drop per calls », c'est-à-dire la quantité moyenne d'appels rejetés par appel effectué. Plus ce ratio est faible, plus le matériel intéressera potentiellement les clients car jugé comme fiable.

C'est pourquoi les testeurs sont très intéressés par la détection automatique de ces calls drop dans les traces réseaux, car la raison est généralement présente dans certains messages particuliers. Une équipe de test d'Alcatel-Lucent a remarqué l'apparition quasi-systématique de deux logs précis à chaque call drop. C'est sur la détection et l'analyse de ces logs que je défini au sein de l'application Android On-Device Diag qu'un call drop s'est produit.

Pour les détecter, il m'a fallu introduire un mécanisme de décodage de tous les messages de type logs en temps réel. Encore une fois, le premier octet de chaque log correspond à son log ID, un identifiant qui permet de connaître à quel type de log nous avons à faire. De ce fait, je suis capable de trouver les deux logs qui nous intéressent pour la détection des calls drop, puis de les décoder plus en détail afin de récupérer la valeur d'un octet spécifique (ce, en me servant de la documentation fournie). Sur une valeur particulière de ce dernier, on annonce la détection d'un call drop.

J'ai également du réaliser une opération de trigger sur ces deux logs, une fois détectés. En effet, lors d'un call drop, il arrive des cas où ces deux logs apparaissent dans les traces, tout comme il arrive des cas où soit l'un, soit l'autre, apparait. Ce trigger, temporisé, est primordial pour éviter d'effectuer une « double détection » du call drop.

J'ai réalisé mon architecture de manière modulaire : en effet, à chaque besoin d'implémentation d'un nouvel événement, il suffira d'implémenter seulement la partie décodage des logs associés.

Déclencher des actions sur détection d'événement

Lors de la détection d'événements, nous souhaitons déclencher une action spécifique. Encore une fois, j'ai développé une architecture modulaire pour permettre de ne développer que la partie « action » pour un nouvel événement détecté.

[1] log : ici, log fait référence à un message particulier remonté par le port diag

Pour reprendre l'exemple de la détection d'un call drop, j'ai du implémenter un mécanisme permettant de copier vers le stockage externe du téléphone une fenêtre de traces réseau de l'ordre de 4 à 5 minutes avant la détection du call drop, et de quelques secondes après la détection. En effet, la raison du call drop est contenue dans les traces précédant les deux fameux logs que nous avons décodés.

Lors d'un run d'une durée de 10h (soit une nuit), on considère en moyenne que l'on a une dizaine de call drop pour un mobile. A raison de 30Mo de traces en moyenne par minutes par exemple, ce mécanisme de détection permet de ne récupérer que ~1.5Go de trace par mobile, contenant principalement les raisons des calls drop rencontrés ; au lieu des 18Go de traces brutes qui ont été initialement générées. Soit un gain de temps de travail relatif de l'ordre de près de 1000%.

Tests fonctionnels de la première version

J'ai réalisé des tests fonctionnels sur cette première version de l'application Android On-Device Diag, afin de valider toutes les intégrations que j'ai réalisées, et notamment la détection de call drop.

Pour ce faire, nous avons utilisé une série de Samsung Galaxy S4, enregistrés sur un réseau réel, sur un run d'une nuit. Cela m'a permit de confirmer la bonne récupération des fenêtres de traces contenant les raisons des calls drop le matin suivant le run.

Egalement, des tests unitaires ont été réalisés de manière à valider les modules que j'ai développés un à un ; notamment en provocant des calls drop nous même pendant que l'application analysait les traces.

3.2.2 – Conception et développement de la version évoluée

Très récemment (vers fin juillet 2014) nous avons, suite à de nombreux échanges, détecté le fichier de la libraire « L » compilée, contenant les interfaces propriétaires entre la couche logicielle et le périphérique diag du modem ; mais surtout, nous avons réussi à nous procurer tous les fichiers « header[1] » nous permettant de développer notre propre version du module « M » au sein même de l'application On-Device Diag.

L'enjeu est extrêmement important car il nous permet de nous affranchir des limitations du module « M » que nous exploitions jusqu'à maintenant :

- Nous pouvons récupérer les traces réseau brutes directement en sortie du périphérique diag ;
- Nous nous affranchissons du passage des traces dans un fichier, et donc des problématiques associées à leur lecture ;
- Nous nous affranchissons dans un même temps des problématiques de temps de vie des mémoires, et donc plus besoin de monter le répertoire en RAM DISK ;
- Nous pouvons également prévoir des applications communiquant directement avec le modem (ce que ne fournissait pas le module « M »)…

[1] fichier header : en langage C et C++, les fichiers headers ont l'extension .h et ne contiennent que la déclaration du prototype des fonctions (c'est-à-dire le nom de la fonction, son type de valeur de retour et le type de ses paramètres).

Le schéma [10] ci-dessous illustre l'évolution de l'application On-Device Diag par rapport à la première version, elle-même illustrée dans le schéma [9] précédent :

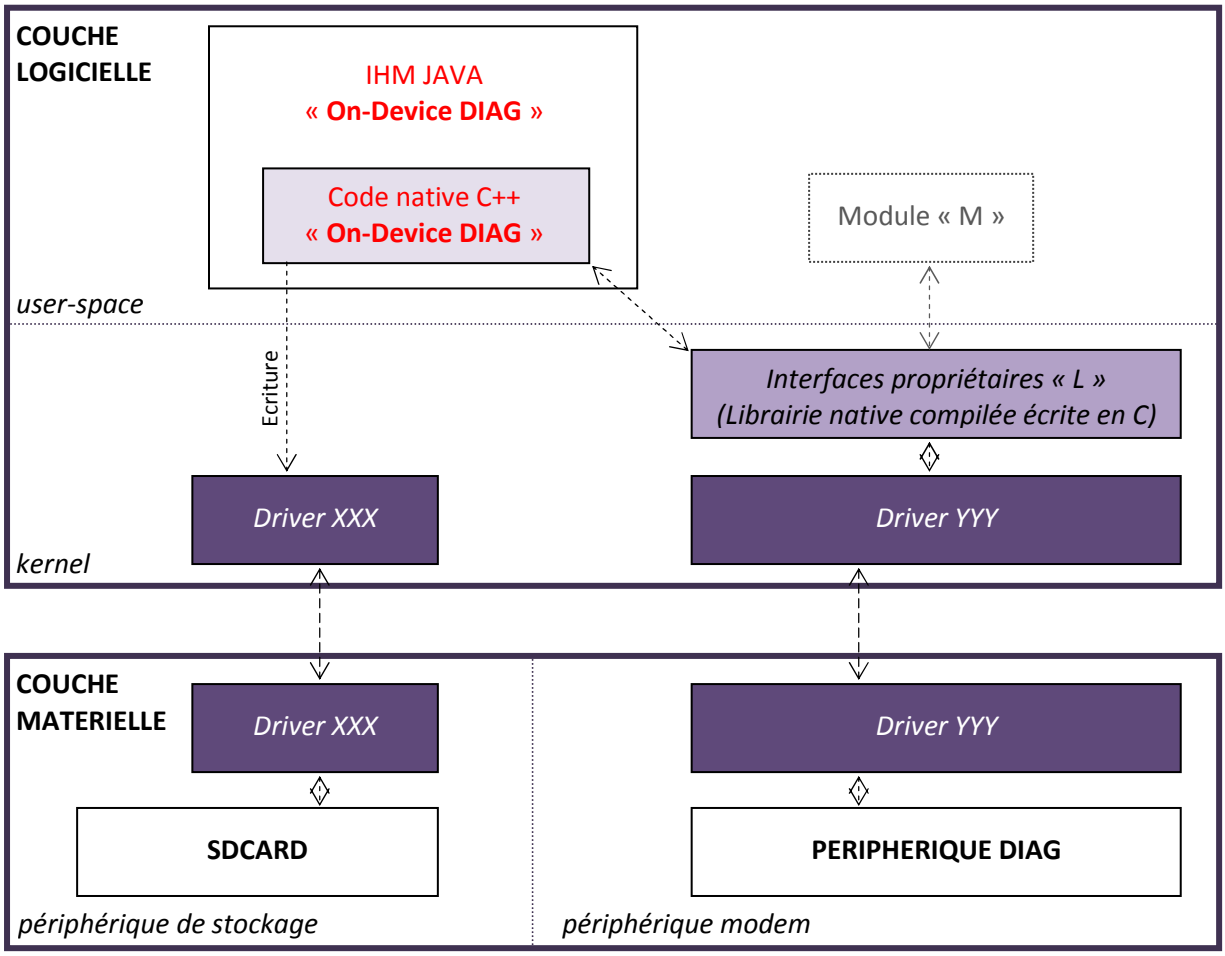


Schéma [10] : illustration des différentes couches logicielles et matérielles utilisées pour la conception de seconde version de l'application Android « On-Device Diag »

Récupérer les sources nécessaires

La première étape pour concevoir cette version améliorée de l'application Android « On-Device Diag » a consisté en la récupération de tous les fichiers sources nécessaires depuis le code source des versions Android Open Source Project, donc Alcatel-Lucent possède toutes les licences nécessaires pour la recompilation des librairies propriétaires, comme c'est le cas pour la librairie « L ».

Je suis rapidement arrivé au constat suivant : il n'est pas possible de récupérer les sources C mêmes de la librairie, du fait de la quantité de dépendance trop importante avec d'autres sources du code d'Android. Par ailleurs, nous souhaitons pouvoir porter notre application à la fois sur les mobiles de test, mais également sur les mobiles du commerce compatibles, ce qui ne pourrait être le cas en recompilant la librairie, dont les sources sont fournies pour une version d'Android en particulier.

Ainsi, j'ai récupéré uniquement les fichiers « header » (.h) contenant simplement les prototypes des fonctions C disponibles dans la librairie compilée. De cette manière, il convient de récupérer cette dernière depuis la partie système du mobile pour lequel on souhaite faire un portage de l'application (elle est suffixée par l'extension .so, typique des librairies partagées du système Linux).

Intégrer celles-ci au projet et compiler

La compilation du code natif de l'application Android On-Device Diag a été alors très spécifique vu que la librairie « L » était précompilée et que nous n'avions que les fichiers « .h ». Cette étape m'a demandée une attention toute particulière, d'autant plus que cette librairie a été codée purement en C alors que notre code natif a été codé en C++. J'ai ainsi été confronté à de nombreux soucis dans les étapes de compilation, mais surtout d'édition des liens, avant de réussir cette tâche.

A ce moment, nous étions en « concurrence directe » avec des équipes d'Alcatel-Lucent présentes aux Etats-Unis et en Pologne notamment : ceux-ci ont été mit au courant de l'existence de la librairie « L » en même temps que nous, et chaque équipe a décidé « d'emboîter le pas » sur les autres de manière à gagner l'exclusivité du développement du logiciel, tant l'enjeu est important. Nous avons été les premiers à réussir la compilation de l'exemple d'utilisation de la librairie « L » au sein d'une application Android en utilisant le compilateur du NDK, ce qui nous a valu d'avoir une longueur d'avance.

Valider le concept sur une première série de mobiles

Avant de me lancer dans le développement de notre propre client interfacé sur le périphérique diag du mobile via la libraire « L », j'ai réalisé des tests fonctionnels sur une série de mobiles de tests et du commerce potentiellement compatibles (en fonction notamment du modèle du couple processeur/modem installé sur ceux-ci). J'ai donc produit une version d'Android « On-Device Diag » par mobile sélectionné, pour lesquelles j'ai intégré la librairie « L » du mobile lors de la compilation.

Ceci m'a notamment permit de noter les divergences d'interfaces en fonction des mobiles et de leur date de mise sur le marché, particulièrement lors de la phase d'édition des liens, où le compilateur tombait en échec pour des raisons d'absence de définition de certaines fonctions dans la librairie compilée « L ».

Exploiter les fonctions d'interfaces

Une fois la mise en place de cet environnement de développement, j'ai pu me lancer dans le développement de notre propre client natif, s'interfaçant avec le périphérique diag du mobile via les interfaces de la librairie « L ».

J'ai du porter une attention particulière à la gestion des ressources, notamment en initialisant convenablement le client, mais également en le dé-initialisant dès que cela était nécessaire, de manière à libérer les ressources pour toute réutilisation future (d'après les documentations que nous avons à notre disposition, il n'est pas possible d'instancier plus de 4 clients simultanément, d'où l'importance de la libération en temps et en heure des ressources qui ne sont plus utilisées).

Intégrer le code de la première version avec les traces issues de ce client

Une fois l'étape de développement du client, il m'a simplement suffit de réécrire quelque peu mes interfaces déjà développées dans la première version de l'application pour les rendre compatibles avec celui-ci : l'application n'est plus chargée de lire des fichiers dans un emplacement spécifique, on récupère désormais directement le flux de traces brutes en continu grâce à une instruction du client. Il convient simplement d'analyser les traces de ce flux en temps réel de la même manière que précédemment (retrait des encapsulations[1], vérification du CRC, récupération du log code…)

[1] encapsulations : dans ce cas, ce terme fait référence aux préfix et sufix ajoutés dans les messages, correspondant aux spécifications du protocole de transmission HDLC

Le principal point à prendre en compte a été le respect du temps réel pour avoir le moins de pertes de traces possible. En effet, autant la première version de l'application « pouvait se permettre » de prendre du retard par rapport à l'écriture des traces (car les fichiers étaient persistants sur le téléphone), autant cette version évoluée de l'application risquait de perdre des traces si elle venait à prendre du retard. J'ai ainsi optimisé mon code au maximum, et j'ai introduit un mécanisme de buffer, me permettant de conserver une fenêtre de traces confortable (de l'ordre de 4 à 5 minutes, soit ~200Mo de traces conservées en RAM). Ce buffer me permettant également de conserver en mémoire la fenêtre de traces intéressantes lors de la détection d'un call drop ; alors écrites dans un fichier lors du déclenchement d'un tel événement.

Tests fonctionnels de la version évoluée

Comme lors de l'étape de validation fonctionnelle du concept de la version évoluée de l'application, j'ai effectué de nouveaux tests fonctionnels complets en situation réelle, sur un banc de plusieurs mobiles différents (mobiles de tests, mais surtout différents mobiles du commerce pour lesquelles cette solution voit les enjeux les plus importants).

J'ai ainsi pu soulever de nombreux points visant à être corrigés, tandis que d'autres aspects méritaient d'être corrigés, notamment vis-à-vis de la portabilité de la solution sur des mobiles différents (bien que tous équipés du même modèle de couple processeur/modem).

A l'heure où j'écris ces lignes, je n'ai pas encore eu l'occasion de réaliser toutes les corrections nécessaires, ni de réaliser de nouveaux tests, notamment car je me suis chargé de la rédaction de documentations d'architecture et d'utilisation dans l'optique de transmettre mes connaissances et compétences sur le projet au nouvel apprenti en passe de me remplacer au terme de mon contrat.

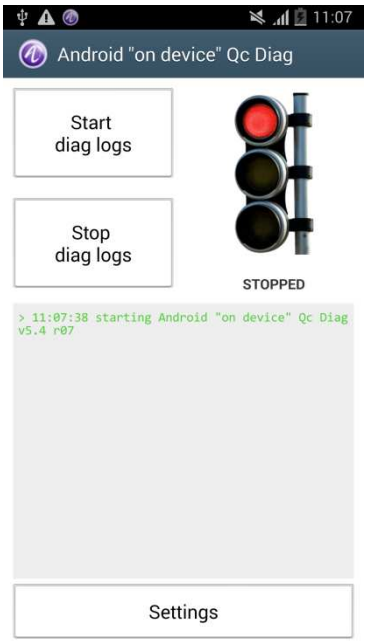


Image [11] : Interface de On-Device Diag. L'application est au repos, en attente d'action de la part de l'utilisateur (depuis l'IHM ou par ligne de commande)

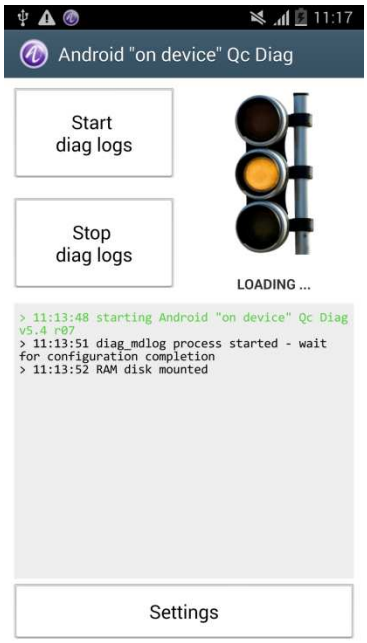


Image [12] : Interface de On-Device Diag. Ecran de chargement, la configuration est en cours d'envoi au couple processeur/modem via les commandes spécifiques.

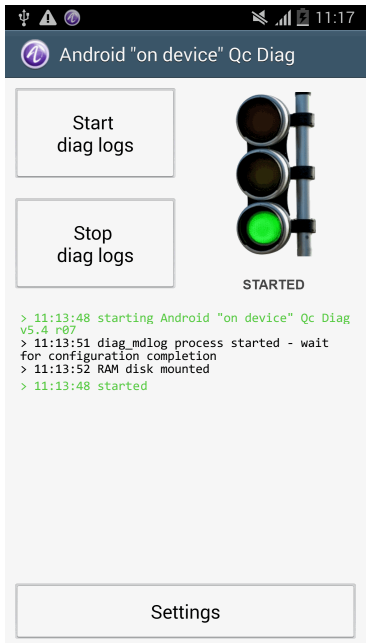


Image [13] : Interface de On-Device Diag. Les traces sont en cours d'analyse en temps réel. A tout moment, un événement peu être détecté, auquel cas une ligne s'affichera dans la console.

3.3 – L'application desktop « OTA Manager »

Dans l'optique de faciliter et d'automatiser une grande majorité des tests de validation fonctionnels de l'application Android « OTA » sur, par exemple, une baie d'une dizaine de mobiles, j'ai développé une application outil en JAVA SE 1.6 destiné aussi bien aux développeurs apportant des modifications à OTA même, qu'aux utilisateurs de mobiles Android sur plateformes de tests en général.

Embarquant le module « ADB » pour la communication entre l'ordinateur et les mobiles Android, dans ses versions Windows 32 bits, 64 bits et Linux, elle propose aux utilisateurs plusieurs dizaines de fonctions classées suivant deux catégories : les commandes ADB pures, et les commandes web-services.

Chaque mobile connecté à l'ordinateur est identifié par son identifiant ADB unique grâce à la commande « adb devices ». Ainsi, OTA Manager est capable de proposer une liste des mobiles reconnus, l'utilisateur pouvant sélectionner ceux avec lesquels il souhaite travailler (OTA Manager pouvant supporter jusqu'à 128 mobiles simultanément (limitation imposée par ADB) aussi bien réels que simulés (via l'émulateur de mobiles Android fourni dans le SDK de Google).

Features ADB proposées dans OTA Manager, pour tous mobiles sélectionnés

- Déployer/mettre à jour l'application OTA ;
- Désinstaller l'application OTA ;
- Lancer l'application OTA ;
- Redémarrer l'application OTA ;
- Redémarrer les mobiles ;
- Placer un fichier sur la SDCARD (pour les transferts FTP PUT) ;
- Placer les fichiers de configuration EMBMS sur la SDCARD.

Web-services proposées dans OTA Manager, pour tous mobiles sélectionnés

- Lancer un scénario de trafic (plus de 50 scénarios pré-chargés mis à disposition) ;
- Récupérer les informations du mobile (getpresence) ;
- Installer les modules complémentaires embarqués (OTA APN, eMBMS simulator) ;
- Activer le mode debug dans OTA, le mode avion, le monitoring par AIDA ;
- Ping du mobile (vérifier la communication par web-services) ;
- Synchroniser l'heure du mobile avec celle du PC ;
- Récupérer l'état des interfaces de connexions ;
- Récupérer l'état des process internes à OTA (scheduler, machine à état …) ;
- Récupérer la liste des StatusData générés lors des trafics ;
- Récupérer la liste des AckData en fin de trafics ;
- Récupérer les statistiques des issues, des web-services, des AckData générés…

Ainsi, en plus de permettre de réaliser tous les tests de validation fonctionnelle sur OTA, OTA Manager permet également de réaliser des actions de debug plus poussées, et de contrôler les mobiles à distance sans avoir à y accéder physiquement, ce qui simplifie grandement le développement d'OTA notamment.

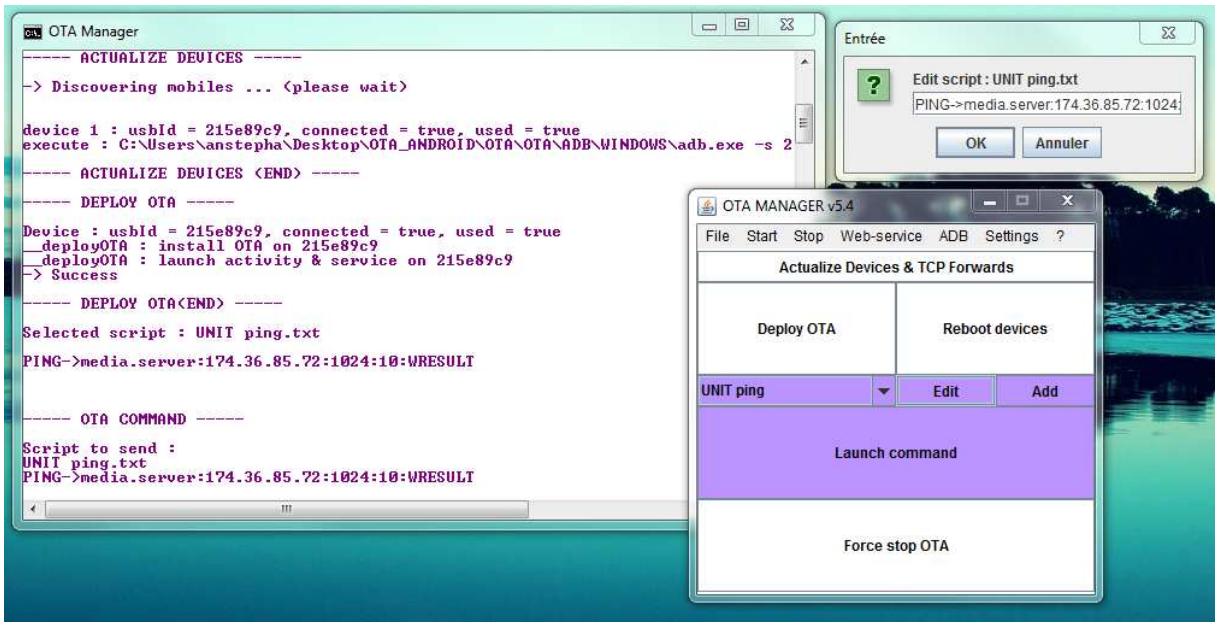


Image [14] : Interface de l'outil OTA Manager, avec sa console (à gauche) contenant tous les messages à destination de l'utilisateur, le contrôleur (en bas à droite) permettant de piloter les actions, et une boîte de dialogue de personnalisation.

3.4 – Documents d'architecture et d'utilisation

Pour chacun des projets sur lesquels j'ai été amené à travailler, j'ai pris l'initiative de réaliser tous les documents associés permettant d'assurer le meilleur suivi possible de chacun, sa compréhension, ainsi que de décrire tous les processus importants (installations, paramétrages, limitations…).

Documents d'architecture

J'ai réalisé un document d'architecture par application, principalement basé sur les diagrammes des classes complets, ainsi que les diagrammes de séquences des principaux process. Associé à de nombreux commentaires dans le code, ces documents permettent de décrire structurellement les implémentations que j'ai réalisées, de manière à ce qu'une personne puisse reprendre le développement confortablement.

Documents d'interfaces

L'application Android « OTA » et AIDA communiquent entre elles par le biais d'interfaces. De ce fait, certaines classes JAVA sont mises en commun entre les deux parties par le biais d'une librairie. Il a été primordial que je documente ces interfaces, notamment vis-à-vis des classes et structures mises à disposition, des variables de retour, des énumérations : description, conditions d'utilisation…

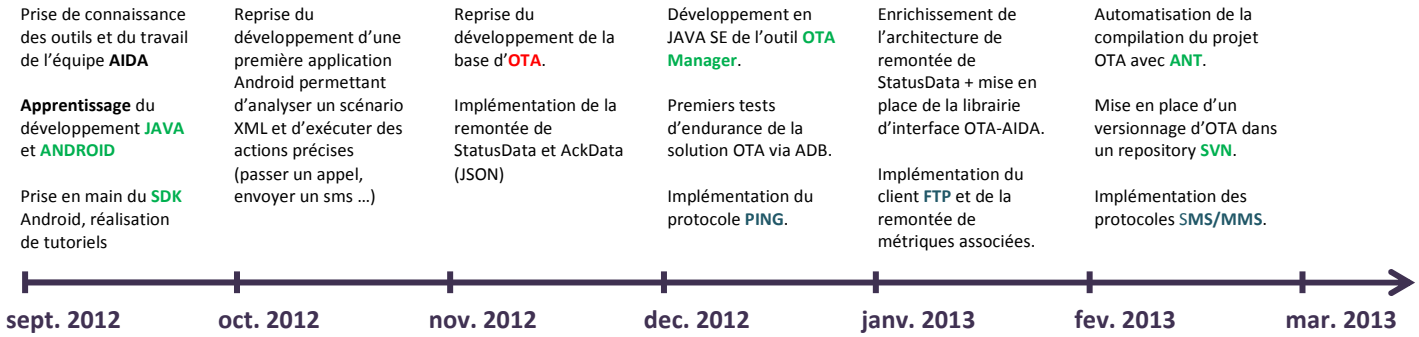
Documents d'utilisation

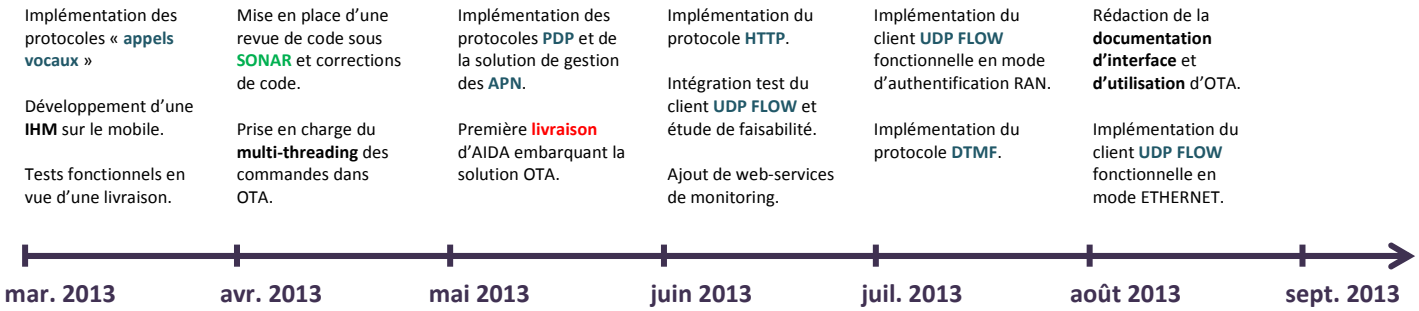
Importants également, ces documents s'adressent aux utilisateurs finaux des applications. Elles décrivent les procédures d'installation, de configuration, les conditions à réunir, mais également les notes de versions (changelog[1]), les divergences possibles suivant les mobiles, les limitations…

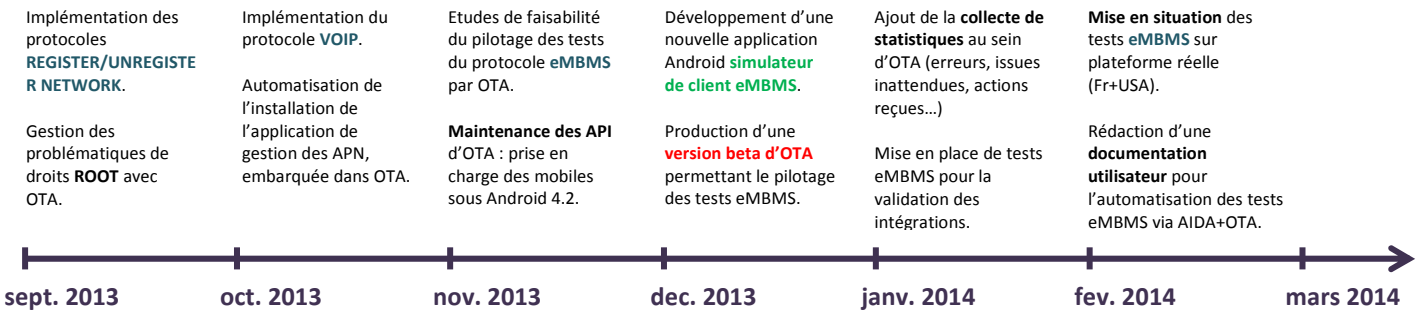
[1] changlog : document décrivant les changements, ajouts et suppressions apportées à un logiciel pour chacune des révisions qu'il a subit.

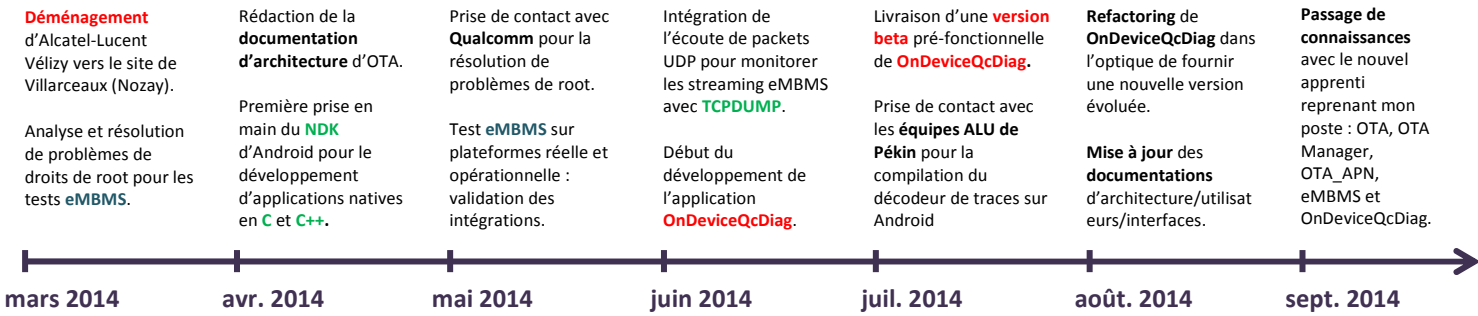
3.5 – Représentation temporelle du travail réalisé

Voici une représentation temporelle non-exhaustive mais toutefois significative de mon travail réalisé lors de mes deux années d'apprentissage au sein d'Alcatel-Lucent :









4 – Difficultés rencontrées

Bien que n'ayant jamais programmé d'application Android ni même utilisé le langage Java avant d'arriver au sein d'Alcatel-Lucent, l'apprentissage de la programmation en elle-même ne m'a pas posé de problème. En effet, j'ai pu apprendre cette programmation « sur le tas », par le biais de tutoriels sur internet et en étudiant le code source de quelques applications. Par la suite, j'ai pu acquérir de nouvelles techniques de programmation et à utiliser une grande partie des API Java et Android en rencontrant des difficultés liées aux implémentations que je souhaitais réaliser, ainsi qu'en consultant les collègues qui m'entourent.

Effectuer des intégrations transparentes

Un des plus importants enjeux de l'application Android « OTA » est de rentre l'utilisation des mobiles Android au sein d'AIDA la plus transparente possible pour les utilisateurs. C'est-à-dire permettre aux testeurs une utilisation similaire aux mobiles classiques dans AIDA.

Ainsi, il m'a fallu prendre en main le logiciel AIDA au fur et à mesure de mes intégrations, afin d'analyser les besoins précis des implémentations à réaliser dans OTA. C'est pourquoi mon architecture logicielle devait être réalisée de manière à être la plus évolutive et modulaire possible afin que chaque amélioration portée à AIDA puisse être répercutée aisément dans OTA.

Contourner les couches d'optimisation d'Android

Android est un système d'exploitation mobile qui est tout particulièrement optimisé pour être utilisé par un utilisateur classique, interagissant avec le téléphone depuis l'écran et d'éventuels boutons physiques. Ce système n'a aucunement été développé en ayant la vocation d'être utilisé à distance, d'être piloté par une application tierce, etc.

C'est pourquoi il m'a été très difficile de faire en sorte qu'OTA s'exécute avec un contrôle total sur le mobile sur des périodes allant jusqu'à plusieurs jours. En effet, le système Android embarque une panoplie de services permettant de nettoyer la mémoire lorsque le téléphone en manque, de fermer les applications qui ne semblent plus interagir avec un utilisateur depuis un certain temps, ou bien de prévenir le système qu'un service non administrateur s'approprie « trop de ressources » par rapport à ce qu'il devrait, etc.

Le plus difficile a été notamment de repérer à quel moment et sous quelle condition intervenaient ces problèmes, de trouver quelle est la couche d'optimisation d'Android qui causait cette issue, et surtout de trouver la solution pour la contourner. Heureusement, Android est forte d'une grande communauté de développeurs établie sur internet à travers de nombreux forums dédiés, et malgré la quantité d'information plus ou moins pertinente que l'on peut trouver sur certains moteurs de recherche, j'ai toujours eu moyen d'apporter un fix[1] à l'application OTA.

[1] fix : correction logicielle

Contourner les couches de sécurité d'Android

Encore une fois, Android est un système optimisé pour être utilisé par des utilisateurs du marché, c'est-à-dire pour le grand public. Android ne cible pas les développeurs en particulier. C'est pourquoi, lorsque l'on souhaite développer une application pour ce système, il faut expliciter dans un fichier de *manifest* [1] toutes les permissions qu'Android devra lui accorder pour sa bonne exécution.

Ces permissions sont de l'ordre de l'accès à internet, de la modification de fichiers présents sur le téléphone (ne serait-ce que pour pouvoir simplement les lire), d'accéder aux valeurs des paramètres du téléphone, d'utiliser des protocoles en particulier (appels, envoi de sms …) etc. Ces permissions seront explicitement indiquées à tout utilisateur souhaitant installer l'application sur son téléphone, afin de l'avertir des risques qu'il peut encourir.

Sans celles-ci, il est très facile de développer une application Android dans un but malveillant : la preuve en est qu'OTA est une application permettant de faire l'interface entre un PC et le téléphone sans intervention de l'utilisateur. On pourrait aller plus loin en contrôlant OTA par le biais d'un serveur distant et en communicant avec par le réseau et non plus par USB. De plus, OTA peut fonctionner sans aucune interface utilisateur, de manière quasiment invisible à tout utilisateur non averti. Sans couche de sécurité appropriée, on pourrait alors développer un malware sur Android, exploitant notamment des failles qui existent dans ce système de sécurité.

Néanmoins, au niveau d'OTA, ces permissions peuvent devenir très contraignantes. L'exemple parfait est celui de la gestion des APN, une permission spécifique pouvait être accordée à l'application jusqu'aux versions d'Android 4.0, mais qui a été retirée du système dans les versions ultérieures. Ce cas de figure peut être rencontré pour d'autres types de requêtes. Heureusement, Android propose aux développeurs et amateurs de modifications de s'accorder les permissions de *root-user*[2].

Les droits de *root-user* permettent à l'utilisateur d'effectuer toutes les opérations qui n'étaient pas permises aux utilisateurs « classiques ». Elle offre notamment la possibilité d'accéder aux fichiers du système et de les modifier (c'est pourquoi les utilisateurs n'y ont pas accès d'office, afin d'éviter toute manipulation malencontreuse). Pour en profiter, il faut *rooter* son téléphone. Plusieurs moyens existent, certains étant fournis par les constructeurs eux-mêmes, d'autres étant développés par la communauté amateur en exploitant des failles du système.

Bien que semblant être le Saint-Graal des développeurs d'Android, le principal souci est qu'aucune API d'Android n'existe pour exploiter les droits de *root-user*, de même qu'il n'existe aucune documentation quant aux commandes qu'il est possible d'exécuter en *root*. C'est pourquoi il faut effectuer de nombreuses recherches sur internet avant de trouver un exemple de la manipulation que l'on souhaite effectuer hors-API, si bien sûr il est possible de l'effectuer même en *root-user*.

Utiliser des API non disponibles nativement dans le SDK

Bien que le SDK d'Android soit très bien fourni en API et en documentation, il arrive par fois des cas où ce que l'on souhaite réaliser n'est tout simplement pas proposé par le SDK. C'est par exemple le cas de l'enregistrement des communications réalisées sur le mobile : on peut enregistrer la source audio en provenance du micro, mais on ne peut pas enregistrer la source audio en provenance du réseau, ceci faisant partie de la politique de vie

[1] manifest : fichier permettant d'indiquer au système les composants, permissions et services dont aura besoin l'application. Il spécifie également le numéro de version et de révision, ainsi que certains paramètres à utiliser (thèmes, noms d'espace de nommage…)
[2] root : utilisateur « racine », celui qui a tous les droits sur la machine (au dessus de l'administrateur même)

privée et de la sécurité d'Android. Cette notion, qui fait l'objet de requêtes de la part de certains clients, ne pourra donc pas être intégrée dans OTA à l'heure d'aujourd'hui.

Egalement, il n'est pas possible de déclencher le décrochage automatique des appels entrant dans le téléphone. Ceci était handicapant pour OTA, notamment lors de l'intégration de la commande ANSWER-CALL (répondre à un appel). Fort heureusement, des solutions annexes existent : les API cachées. Ce sont des API du SDK d'Android qui ne sont pas intégrées à celui-ci, ou dont on n'a pas ou plus l'accès, la plupart du temps parce qu'elles sont en cours de développement ou bien trop dangereuses dans leur utilisation.

La difficulté est qu'elles ne sont recensées officiellement à aucun endroit, et que par conséquent il n'existe aucune documentation ni aucun modèle d'utilisation. Encore une fois, pour pouvoir les trouver et les exploiter, il faut faire preuve de patience dans ses recherches.

Enfin, de nombreuses API n'ont pas encore été intégrées au SDK et ne sont officiellement pas en cours de développement chez Google. C'est le cas de l'envoi de MMS, où aucune API n'existe. Il faut dans ce cas écrire ses propres API permettant d'encoder le message et celles permettant son envoi sur le réseau en utilisant le protocole HTTP, et donc en exploitant des API bas-niveau existantes. Il en est de même pour le DTMF, Android ne permettant de définir l'exécution de séquences à effectuer qu'au moment où l'on souhaite passer l'appel et non en temps réel.

Anticiper les divergences de fonctionnement suivant les mobiles

Plusieurs centaines de modèles de mobiles apparaissent sur le marché tous les ans, accompagnées d'une nouvelle version d'Android. Sans anticiper les nouvelles fonctionnalités proposées par les constructeurs, et les nouvelles restrictions posées par Google sur ses interfaces de programmation, les divergences de fonctionnement entre les mobiles risquent de devenir très handicapant pour une utilisation convenable de la solution à long terme.

La réponse à cette problématique réside ainsi, à nouveau, en un développement architectural très précis d'OTA, de manière à produire un code dont l'enrichissement, l'apport de corrections et la maintenance sont simples et rapides. Il constitue ainsi de s'informer continuellement des modifications apportées par Google et/ou les constructeurs, et de répéter fréquemment de nombreux tests de validation fonctionnelle sur de nouveaux mobiles.

Respecter la confidentialité et les droits d'exploitation de licences

Lors du développement de l'application Android « On-Device Diag », j'ai eu l'occasion de travailler en étroite collaboration avec une grosse société experte dans le matériel de télécommunication, partenaire d'Alcatel-Lucent sur de nombreux projets dont AIDA. Egalement, j'ai reçu tous les accès nécessaires à des codes sources propriétaires sous licence, ainsi qu'à des documentations protégées par le secret industriel.

Travailler sur ce projet a impliqué une rigueur incomparable. Effectivement, je me devais de respecter les conditions imposées par les licences en possession au sein d'Alcatel-Lucent, et donc n'utiliser que le code source pour lequel j'avais les autorisations d'exploitation nécessaires. Egalement, je ne devais faire fuiter aucun document, code source, ni quelconque information quand à ce projet de manière à conserver l'exclusivité au sein de l'équipe.

Portage d'une application desktop sur Android

Dans le cadre du projet « On-Device Diag », nous avons tenté d'apporté une grosse plus value à cette application Android en portant le décodeur des messages de type « logs » (ceux-ci étant encodés au format hexadécimal, les informations importantes étant incluses dans certains octets particuliers).

Ce décodeur, initialement développé en C++ en France par Alcatel-Lucent, est aujourd'hui développé par Alcatel-Lucent Shanghai Bell (la filiale chinoise du groupe) à destination d'utilisateurs Windows. J'ai été pendant quelques jours en relation avec un ingénieur chinois de l'équipe en charge du projet décodeur afin de réaliser les étapes suivantes :

- Obtenir un accès au code source du décodeur ;
- Réaliser une première étude des modules du décodeur dont nous avions besoin pour un décodage en temps réel dans la solution On-Device Diag ;
- Tenter une compilation des third-parties[1] ;
- Tenter une compilation du décodeur sur Android.

Malheureusement, au terme de l'étude préliminaire de faisabilité que j'ai réalisée, j'ai estimé que la difficulté serait trop importante s'il suffisait d'essayer de compiler tel quel le décodeur sur Android. Ainsi, j'ai proposé de réaliser le travail en suivant les étapes suivantes :

- Réaliser une version allégée du décodeur (c'est-à-dire retirer tous les modules non utiles pour nos besoins de décodage, ainsi que toutes les dépendances associées dans les sources) ;
- S'affranchir de toutes les third-parties dans la mesure du possible (certaines librairies utilisées n'étant pas portables sur Android) ;
- Analyser et réaliser la portabilité de cette version allégée sur les systèmes Linux (le code étant principalement orienté Windows, bien qu'étant développé en C++ et donc possiblement cross-plateformes) ;
- Tenter une compilation sur Android via le compilateur du NDK.

Cette dernière étape reste un point sensible ; en effet j'ai pu réaliser, lors de l'utilisation des interfaces proposées par le NDK d'Android que certaines fonctions de la STL[2] n'étaient tout simplement pas fournies pour la compilation sur Android, ceci étant du à la volonté de Google de ne fournir aux développeurs Android natifs que les interfaces dont le bon fonctionnement sur ce système d'exploitation est validé par les équipes de testeurs.

Déménagement des locaux

Vers le milieu de mon apprentissage, j'ai assisté à la fermeture du site de Vélizy sur lequel j'ai été affecté, tous les effectifs (dont moi-même) étant affecté au site de Villarceaux. J'ai ainsi pu constater la difficulté de maintenir le développement et le support d'un projet comme AIDA dans des conditions difficiles. Notamment, le déménagement de la radio de Vélizy à Villarceaux a prit beaucoup de retard, ce qui a largement compliqué nos tests de validation car nous n'avions plus la possibilité d'exploiter certaines fonctionnalités hors-ligne.

[1] third-party : librairies indépendantes disponibles gratuitement sur internet, permettant d'enrichir le programme
[2] STL : « Standard Template Library », ensemble de librairies standards du langage C++

3

Héritage technique et méthodologique

III. – HERITAGE TECHNIQUE ET METHODOLOGIQUE

1 – Bilan des acquis techniques

L'apprentissage permet de vivre une véritable expérience en entreprise, sur des projets industriels concrets. Ces deux années au sein d'Alcatel-Lucent m'ont permit d'y appliquer mes connaissances acquises et enseignées en parallèle lors des cours dispensés par le Master I3S ; mais également de m'enrichir de nombreuses compétences techniques.

1.1 – Langages de programmation

Le langage orienté objet « JAVA Standard Edition »

JAVA est un langage de programmation orienté objet développé depuis 1995 par Sun Microsystems, puis racheté en 2009 par Oracle. Sa spécificité par rapport aux autres langages de même niveau réside dans la grande portabilité des logiciels d'une plateforme à une autre : Windows, UNIX, Mac OS, GNU Linux… En effet, les applications ne sont pas compilées pour un processeur ou un système en particulier, car elles sont prévues pour s'exécuter sur une machine virtuelle[1] unique fournie par JAVA (Java Runtime Environment).

C'est en partie cet aspect de portabilité simplifiée qui en fait un des langages les plus utilisés dans le monde de l'entreprise, notamment pour le développement des applications desktop et des clients/serveurs. AIDA est elle-même développée avec la version 1.6 de JAVA Standard Edition.

Lors de mon arrivée au sein d'Alcatel-Lucent, je n'avais jamais développé en JAVA ni en orienté objet. C'est pourquoi il m'a fallu acquérir toutes les bases du langage avant de pouvoir me lancer dans la programmation d'applications. Au terme de ces deux années, je ne peux bien entendu pas prétendre d'être devenu un expert en JAVA, je suis conscient qu'il me reste de nombreux aspects à apprendre, toutefois je pense être devenu un développeur confirmé. Je suis en effet maintenant capable de concevoir rapidement une architecture propre, fiable et modulaire en programmation objet (notamment en utilisant les *design patterns* standardisés) ainsi que d'assurer le développement en respectant des délais restrictifs.

La programmation d'applications Android : SDK

Le développement des applications Android se fait en JAVA 1.6. En effet, Android embarque une machine virtuelle dédiée, nommée *Dalvik*, permettant l'exécution de programmes JAVA classiques, mais dont les classes exploitant des fonctionnalités caractéristiques du SDK (Software Development Kit) d'Android sont compilées dans un format spécifique.

La réelle différence entre le développement d'application desktop en JAVA et le développement d'applications Android en JAVA réside ainsi dans les API (interfaces de

[1] machine virtuelle : une machine virtuelle est un système informatique qui permet de simuler un système d'exploitation et/ou les couches matérielles.

programmations) spécifiques, dédiées à l'utilisation des composants internes du système et du matériel. De nombreuses interfaces permettent par exemple de réaliser des interfaces graphiques, de communiquer avec les fonctionnalités du téléphone et avec d'autres applications, de gérer la mémoire et les différents processus…

Au cours de mes deux années d'apprentissage, j'ai développé pour l'équipe AIDA plus de 5 applications Android différentes, dont 2 projets conséquents. Sur mon temps libre j'ai également développé près d'une dizaine d'applications, dont deux pour des projets de Master. Le développement de toutes ces applications m'ont permit de parcourir la quasi-totalité de toutes les interfaces fournies par le SDK d'Android ; ayant été confronté à de nombreuses problématiques que j'ai su résoudre à chaque fois.

C'est pourquoi aujourd'hui, je considère avoir acquis toutes les compétences nécessaires en développement Android pour apporter mon expertise au sein de projets de grande ampleur.

Les langages C (impératif) et C++ (orienté objet)

Le langage JAVA est calqué en majeure partie sur le langage C++, en apportant un niveau d'abstraction permettant d'épurer ses concepts les plus subtiles et difficiles à prendre en main, tels que les pointeurs, références, allocations de mémoire… Apparu en 1983 (soit 12 ans avant le JAVA) le C++ est donc un langage bas niveau (proche du langage machine). Par rapport au langage C, qui est un langage impératif, le C++ apporte la dimension « objet » très typique des gros projets industriels d'aujourd'hui.

Au sein d'Alcatel-Lucent, j'ai notamment été amené à développer du code en C et C++ dans le cadre du projet Android « On-Device Diag ». En effet, ce langage apporte au logiciel une proximité importante avec le matériel, et donc des performances améliorées. Néanmoins, la compilation est spécifique, et cible une architecture processeur ainsi qu'un système d'exploitation particulier ; c'est un élément à ne pas négliger dans un projet.

Bien que j'aie majoritairement développé en JAVA pendant ma période d'apprentissage, j'ai quand même eu l'occasion de pratiquer suffisamment le C et le C++ pour être aujourd'hui apte à travailler pleinement avec ces langages en entreprise. J'ai également pris l'habitude de passer de ces langages au JAVA, et inversement, notamment par l'utilisation de l'interface JNI (Java Native Interface) permettant l'intégration de code natif dans une application JAVA, comme c'est le cas pour beaucoup de projets.

La programmation d'applications natives Android : NDK

Le NDK d'Android (Native Development Kit) fournie toutes les bibliothèques et outils nécessaires pour la compilation d'applications natives en C et/ou C++ pour les mobiles Android. Ces applications peuvent être purement natives (programmes ou librairies) ou bien embarquées dans des applications Android développées en JAVA avec le SDK grâce à l'interface JNI.

Bien que fourni par Google de la même manière que le SDK, ce kit de développement est relativement difficile à prendre en main, tant au niveau de la mise en place de l'environnement de développement que pour la programmation et la compilation. En effet, je me suis rendu compte que celui-ci n'était malheureusement pas aussi bien documenté que le SDK, c'est pourquoi j'ai rencontré de nombreuses difficultés, tant les subtilités sont nombreuses. Egalement, je me suis rendu compte au fur et à mesure que certaines interfaces

standard en C++ n'étaient pas mise à disposition dans le NDK, ce qui a davantage compliqué la situation.

Ainsi, après près de 5 mois d'utilisation, je suis également capable de mettre à profit ces compétences sur de nouveaux projets, malgré le fait que j'ai encore un long chemin à parcourir avant de connaître tous les aspects du NDK.

Les langages de balisage XML et JSON

XML est un langage de balisage que j'ai principalement appris à utiliser dans le cadre de la conception d'interfaces graphiques d'applications Android. En effet, XML permet d'agencer des objets par inclusions dans d'autres objets, tout en leurs affectant des propriétés spécifiques. XML est également un langage que j'ai couramment utilisé pour le stockage de faibles quantités de données (propriétés, configurations, états …) car facilement accessible par des librairies JAVA ou C++.

JSON est également un langage de balisage, dont la syntaxe est légèrement différente de celle du XML. Il permet une description davantage verbeuse par rapport à ce dernier. JSON est couramment utilisé en réponse aux web-services, comme c'est le cas pour l'application « OTA », mais j'ai également eu l'occasion de l'utiliser pour la communication d'objets entre deux processus différents par exemple.

1.2 – Systèmes d'exploitation mobiles

Le système d'exploitation Android

A mon arrivée au sein d'Alcatel-Lucent, la majeure partie des mobiles du commerce tournaient sous Android dans ses versions 2.2 (estampillée *Frozen-Yogurt*, soit en français « Yahourt Glacé ») ; 2.3 (*Gingerbread*, « pain d'épice ») ; 3.0 (*Honeycomb*, « nid d'abeilles ») destinée aux tablettes ; et surtout sous 4.0 (*Ice-Cream Sandwish*, « sandwish à la crème glacée »). S'en sont suivi les versions 4.1 ; 4.2 et 4.3 (Jelly Bean, « bonbons gélifiés ») ; ainsi que 4.4 (KitKat, du biscuit chocolaté du même nom).

En deux années, le système Android a ainsi beaucoup évolué, de même que la programmation d'applications Android. J'ai ainsi appris à comprendre le fonctionnement d'Android au même rythme que ses évolutions. Mais le plus instructif a été de me retrouver confronté à des problématiques directement liées à ce système d'exploitation.

Cumulés, cela fait donc plus de 5 ans que je m'intéresse en détail au fonctionnement d'Android, mais ce sont ces deux années qui m'ont été les plus instructives, à tel point que je suis désormais capable de répondre à beaucoup de questions relatives à Android, quelque soit son niveau.

Les systèmes d'exploitation Linux

Le noyau du système d'exploitation Android est basé sur une version allégée du noyau Linux. C'est pourquoi il est possible d'y exécuter les mêmes appels systèmes et commandes shell[1] que sous un système Linux pour PC.

De ce fait, je suis capable aujourd'hui d'utiliser un grand nombre de commandes Linux, des plus classiques au moins communes : `ls`, `pwd`, `cd`, `grep`, `chmod`, `chown`, `mkdir`, `find`, `kill`, `ps`, `df`, `top`, `cp`, `ln`, `alias` …

[1] shell : couche logicielle qui fournit l'interface utilisateur d'un système d'exploitation.

1.3 – Technologies

Les technologies de réseaux mobiles

Ayant contribué au projet AIDA au sein d'Alcatel-Lucent, j'ai bien entendu acquis de nombreuses connaissances dans les technologies des réseaux mobiles, aussi bien matérielles que logicielles. Ces connaissances parcourent notamment les différentes infrastructures des radiocommunications, les dispositifs et processus mis en place pour le déploiement d'antennes et de réseaux mobiles, les méthodes de communication entre le *core network* et les mobiles, les processus d'attachement et de détachement des mobiles sur le réseau, etc.

En parallèle j'ai également appris les spécificités et différences technologiques et historiques entre les générations de technologies : 2G, 3G et 4G. J'ai notamment participé à des présentations techniques sur des journées complètes, par exemple pour apprendre les bases de la technologie 4G LTE, qui m'ont permis de comprendre comment celle-ci proposait des débits bien plus importants que ses prédécesseurs.

Les protocoles de communication

Le développement d'OTA m'a proposé une formation très complète au niveau de la compréhension des différents protocoles utilisés aussi bien dans les réseaux mobiles que terrestres. En particulier les protocoles FTP, HTTP, TCP, UDP, ICMP, SIP, RTSP, EMBMS, PPP, HDLC… définis dans différentes couches protocolaires du modèle OSI (Open System Interconnection).

Je suis ainsi capable de définir et de reconnaître les protocoles, de savoir quels sont leurs enjeux et leurs domaines d'intervention, et particulièrement de développer des clients et serveurs dédiés spécifiques.

1.4 – Outils et logiciels

Les outils de gestion de code

En entreprise notamment, il est primordial de mettre rapidement en place un système de gestion de code lorsque l'on débute un projet. Au sein de l'équipe AIDA, j'ai été amené à utiliser la solution de gestion de versions *SVN*, puis *Mercurial*, qui sont deux solutions qui restent quelque peu similaires au niveau des fonctionnalités proposées, mais dont le fonctionnement est différent.

En ce qui concerne la compilation, j'ai été amené à utiliser le langage *ANT*, qui permet de réaliser l'automatisation et l'harmonisation de la compilation d'un projet. Cet outil permet de rendre un projet compilable sur n'importe quelle machine sans avoir à réinstaller tout l'environnement de développement.

Enfin, j'ai aussi utilisé l'outil *PROGARD*, qui permet « d'assombrir » le code compilé d'un projet JAVA : en effet, JAVA étant un langage interprété (c'est-à-dire que le code est compilé « à la volée » lors de l'exécution et non pas avant la livraison logicielle) il est aisé de réaliser une décompilation du logiciel et de revenir au code source. *PROGARD* permet de remplacer les noms de méthodes et de variables par de simples lettres de l'alphabet : ainsi la méthode « void parse(int file, boolean closeAtEnd) » devient « void a(int b, boolean c) ». Cela permet de rendre le code décompilé illisible et donc de rendre difficile la copie de code, protégeant donc le projet.

Les environnements de développement

J'ai utilisé plusieurs environnements de développement au sein de l'équipe AIDA :

- Eclipse, pour le développement d'applications Android ;
- Netbeans, pour le développement d'AIDA ;
- Visual Studio, pour le développement C/C++.

Toutefois c'est principalement le logiciel Eclipse que je maîtrise le plus car c'est celui que j'ai été amené à utiliser le plus souvent, étant principalement en charge de développement Android.

Les logiciels de contrôle par le réseau

Lors de la réalisation de tests, ou bien du support apporté aux utilisateurs d'AIDA, j'ai utilisé plusieurs solutions de contrôle de machines à distance, par le réseau. Notamment, lors qu'il s'agissait de contrôler un ordinateur Windows depuis un ordinateur Windows, j'utilisais l'outil *Remote Desktop* fourni dans le système par Microsoft.

Pour la prise de contrôle d'ordinateurs sous Linux, j'ai du utiliser des logiciels dont le principal a été *VNC*, en mode graphique. Pour des besoins spécifiques, j'ai également appris à utiliser le logiciel PUTTY, permettant entre autre d'exécuter des instructions distantes sur un PC en ligne de commandes.

1.5 – Dispositifs informatiques

Les web-services

Les web-services permettent la communication et l'échange de données entre des applications et des systèmes hétérogènes dans des environnements distribués. C'est un ensemble de fonctionnalités, très utilisées à la fois sur internet et sur intranet. OTA exploite notamment plusieurs web-services pour la communication avec AIDA.

Les bases de données

Les bases de données sont des conteneurs permettant de stocker et de retrouver l'intégralité des informations qui y ont été stockées, même après une extinction de la machine qui l'héberge. Elles sont très utilisées dans Android, notamment en ce qui concerne le stockage des SMS, MMS, APN, et de certains paramètres de configuration.

L'accès à ces données s'effectue par le biais de requêtes SQL (*Structured Query Language*) dont la syntaxe est conventionnelle sous Android. Le SDK propose notamment la bibliothèque *SQLITE 3* pour réaliser les accès en base.

J'ai été amené à faire de la gestion de bases de données dans le cas de la lecture de SMS/MMS et de leur suppression, et dans le cas de la lecture et de l'insertion d'APN. De même, pour l'application Android On-Device Diag, j'ai créé et rempli une nouvelle base de données contenant de manière indexée les quelques 500 000 lignes d'un fichier mettant en relation un *hash code* et sa valeur textuelle associée pour le décodage de messages particuliers dans les traces réseaux. Cette base de données me permet ainsi de faire de la récupération ciblé d'une donnée en particulier de manière bien plus efficace qu'en réalisant une recherche classique dans le fichier.

Les analyseurs de paquets

Lors de l'intégration du pilotage des tests eMBMS dans OTA, j'ai utilisé la librairie *TCPDUMP*, que j'ai portée et compilée pour les mobiles Android. Il s'agit d'une libraire open source permettant de faire de l'analyse de paquets réseaux en ligne de commande, avec la possibilité de réaliser des filtrages tels que la récupération de paquets :

- par type de protocole (TCP, UDP, ICMP…) ;
- vers ou depuis un port particulier ;
- vers ou depuis une adresse IP particulière ;
- relatifs à une certaine taille de paquet …

TCPDUMP est une libraire développée depuis 1987 pour les systèmes Linux, aujourd'hui portée sur Windows sous la dénomination de *WINDUMP*.

Egalement, j'ai exploité la commande Linux `netstat`, permettant de récupérer des valeurs moyennes quant aux paquets échangés sur tous les ports ouverts.

2 – Modèles et méthodes appliqués

En plus des aspects techniques dont j'ai hérité des connaissances, j'ai également reçu un apprentissage complet quant aux modèles et méthodes appliqués en entreprise sur des projets concrets.

2.1 – Environnement

Le travail en autonomie

Plus de 50% de mon temps a été consacré au travail en autonomie, du fait de la séparation du développement d'AIDA et des applications Android. Il a ainsi fallu que je m'auto-forme sur de nombreux aspects de manière à faire preuve de la plus grande indépendance possible, tout en communicant régulièrement sur l'avancé de mes travaux.

Le travail en binôme

Lors de ma première année d'apprentissage, j'ai travaillé en binôme sur certains aspects du développement d'Android (notamment la recherche de la remontée des traces réseaux à même le mobile qui, 10 mois après le départ de ce dernier, est devenu Android On-Device Diag) ainsi que sur la solution de modification des APN du téléphone depuis l'application OTA. Le travail en binôme sur le code d'un même projet est complexe, notamment parce le fait qu'il faut prendre garde à ne jamais travailler sur les mêmes sources simultanément.

Le travail en équipe

Intégré à l'équipe AIDA, et développant une solution interfacée avec le logiciel développé par celle-ci, j'ai bien entendu du travailler en équipe une grande partie de mon temps, notamment dans le but de m'accorder avec les développeurs d'AIDA, mais également pour veiller à ne jamais introduire de rupture d'interface entre les deux parties, étant intégralement responsable de celles-ci.

2.2 – Méthodes

La conception en Agile

AIDA est développé en Agile. L'agilité est un groupement de pratique en conception logiciel, régissant le cycle de développement de celui-ci. A la différence de méthodes de conception du type « cycle en V [1]», l'agile permet de développer le logiciel par itérations, en s'appuyant sur 4 fondamentaux : l'équipe, l'application, la collaboration, et l'acceptation du changement.

Ainsi, il n'y a pas de cahier des charges défini et arrêté ; le développement de la solution s'organise suivant les besoins directs du logiciel et des clients. AIDA est développé suivant la méthode agile dite de *Scrum* [2]:

- Un « ***planning meeting*** » permet de définir les fonctionnalités à implémenter pour la future itération, dont la livraison se fera le mois suivant (ou parfois, 2 mois après). Chaque membre de l'équipe est au même moment affecté d'une ou plusieurs tâches à réaliser, sous l'approbation du *product owner* [3];
- Un « ***daily meeting*** » est organisé à horaire régulier sur une durée de 10 minutes, afin de faire un bilan journalier du travail réalisé et à réaliser pour chacun des développeurs. Ce meeting est organisé par le *Scrum Master* [4], en charge de la répartition de la parole et de la collecte des informations ;
- Les 15 derniers jours de l'itération sont alloués aux **tests** de validation fonctionnelle et aux tests de non régression, afin d'apporter toutes les corrections nécessaires avant la livraison ;
- Puis un nouveau « ***planning meeting*** » est organisé…

J'ai ainsi été confronté aussi bien aux avantages, qu'aux inconvénients du développement en méthode agile, par rapport au développement en cycle en V notamment.

L'analyse de l'existant

L'analyse de l'existant est une méthode de conception consistant à étudier des propositions et des besoins spécifiques dans le but de vérifier si une solution équivalente n'existe pas déjà. C'est une étape primordiale avant la conception même d'une architecture, car cela permet la plupart du temps de la simplifier, de gagner du temps et d'alléger le code. J'ai fréquemment été confronté à ce type d'analyse, ce qui m'a permit de repérer l'existence de nombreuses librairies open-source très utiles pour mon projet.

L'analyse de faisabilité

L'analyse de faisabilité permet d'évaluer la capacité d'un besoin à être réalisé avec tous les moyens que l'on a à sa disposition, permettant de la même manière que l'analyse de l'existant de gagner un temps crucial avant de se lancer dans une conception ou dans du développement. Notamment, dans le cas de notre tentative de portage d'une application décodeur de traces sur Android, j'ai pu constater la non-faisabilité du projet tel quel, produisant alors une série de recommandations préalables plutôt que d'investir un temps trop important dedans, se soldant au final par un échec.

[1] cycle en V : méthode de développement logiciel dont la description schématique fait penser à un V
[2] scrum : melée
[3] product owner : chef de projet, dont la direction du projet appartient
[4] scrum master : maître de mélée

2.3 – Gestion de projet

Le versionnage de code

Le versionnage du code est primordial pour la bonne tenue de n'importe quel projet. Cela permet de connaître quel version utilise tel ou tel utilisateur, et ainsi remonter plus facilement au cœur des problèmes. Cela passe par des codes de version structurés : par exemple « v2.1.05 », qui pourrait ainsi être analysé comme étant « la cinquième révision mineure de la première révision majeure de la deuxième version de l'application ». Ces versions/révisions étant décidées par le type d'intégrations et / ou de corrections apportées d'une livraison à l'autre. Chaque nouvelle révision doit être accompagnée de son *changelog*.

Les gestionnaires de code du type *SVN* ou *MERCURIAL* permettent également d'apporter un versionnage par fichier source ; cela permet de remonter unitairement sur les modifications apportées à un fichier en particulier, et notamment de réaliser un *roll back* (un retour en arrière).

La traçabilité des intégrations

Dans le cadre d'un projet multi-développeur, il est primordial de réaliser tous les documents associés aux intégrations. Cela passe bien entendu, dans un premier temps, par la rédaction de commentaires dans le code (javadoc, en-têtes, commentaires unitaires…). Dans un second temps, cela consiste en la rédaction de document d'architecture (permettant à n'importe quel développeur de prendre connaissance rapidement de la manière dont tel ou tel module a été conçu), de documents d'interfaces (permettant à un utilisateur souhaitant s'interfacer avec la solution de connaître les valeurs des constantes, commandes, structures communes…) ainsi que de documents d'utilisation (afin de savoir comment le logiciel fonctionne, en faisant abstraction de la manière dont il a été programmé).

Le passage de connaissances

Le passage de connaissances est une méthode supplémentaire au versionnage du code et à la traçabilité des intégrations : il permet de s'assurer qu'au moins une autre personne de l'équipe est mise au courant de certains aspects ou subtilités particuliers rencontrés. En outre, avant mon départ au terme de mon contrat, j'ai du réaliser un important passage de connaissance avec le nouvel apprenti me remplaçant, dans l'optique de lui apporter tout le savoir nécessaire pour lui permettre de reprendre le projet.

La résolution de problèmes sur tickets

L'équipe AIDA a à sa disposition une solution accessible par l'intranet, sur laquelle notamment les utilisateurs peuvent ouvrir des tickets à destination du support logiciel. Ces tickets peuvent être consultés et modifiés par les développeurs, afin de demander des informations supplémentaires au client, de s'approprier la résolution du problème, de le marquer comme étant en cours de résolution ou résolu, d'indiquer dans quelle livraison la correction sera apportée, etc.

Cette méthode de travail permet notamment de conserver une trace de tous les problèmes soulevés et/ou résolus, de manière à pouvoir proposer une solution rapide à tout client ouvrant un ticket similaire.

Conclusion

4

IV. – CONCLUSION

1 – Bilan des deux années d'apprentissage

J'ai choisi de faire un master par apprentissage pour de nombreuses raisons, et principalement car j'avais la conviction que cela me permettrait de d'être maître de ma formation, en choisissant le domaine, l'entreprise et le projet de mon choix. Tremplin vers un poste d'ingénieur à temps plein, l'apprentissage permet d'étudier tout en ayant les avantages d'un contrat de travail, offrant deux années d'expérience significative sur son CV.

Au terme de ces deux années d'expérience passées au sein d'Alcatel-Lucent, mon bilan est plus que positif. J'ai été intégré dans une équipe dont tous les membres ont été très pédagogues avec moi. En parallèle, mon maître d'apprentissage a su m'accorder une grande confiance quant à la réalisation des missions qu'il m'avait confiées, tout en ayant une grande souplesse vis-à-vis du rythme auquel mon travail devait avancer (par exemple lorsque j'ai rencontré les difficultés que j'ai énumérées précédemment). Cette confiance s'est bien entendue instaurée grâce à investissement permanant de sa part et de la mienne tout au long de l'apprentissage.

Très formateur sur tous les plans, l'apprentissage m'a ainsi permit d'acquérir de nombreuses compétences techniques et méthodologiques, mais également au niveau de la connaissance de l'entreprise, ainsi que sur l'aspect humain. J'ai notamment vécu les difficultés rencontrées par Alcatel-Lucent ces dernières années : pour mon expérience personnelle, cela m'a permit de constater les différentes erreurs que peuvent réaliser les entreprise, l'organisation de plans sociaux, la restructuration, etc. Bien que peu joyeuses pour les employés, tout cela a grandement participé à ma formation : on apprend toujours des erreurs, qu'elles soient siennes ou non.

J'ai également pu constater que, qu'importe le poste et le projet, il y aura toujours de nouvelles connaissances et compétences à acquérir pour se perfectionner, ou se spécialiser. La technologie évolue aujourd'hui à un rythme très soutenu, d'où la nécessité de rester constamment ouvert à l'apprentissage sans se reposer sur ses acquis.

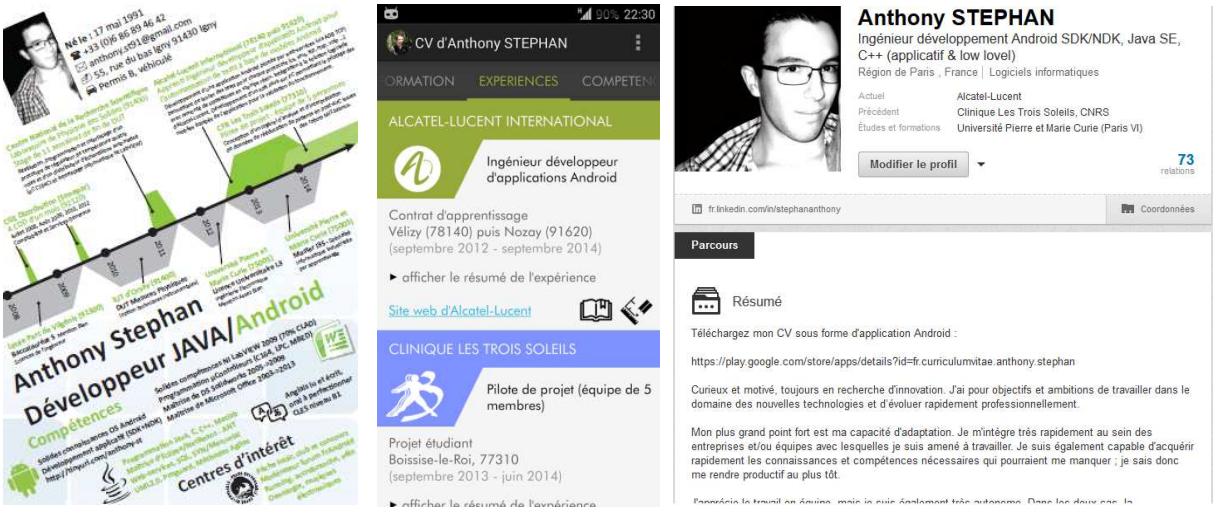
Je n'ai donc aucune raison de regretter d'avoir choisi la voie de l'apprentissage : ce que je regrette par contre est que ce type d'étude ne soit que très peu reconnu en France. C'est pourquoi je n'hésiterais pas à recommander à mon entourage de ne pas hésiter à se lancer sur un diplôme par alternance, qu'importe le domaine ou le niveau. Bien entendu, la principale (voir même la seule) difficulté réside dans la gestion de son temps, il est parfois compliqué de passer de l'université à l'entreprise et inversement. Mais bien entendu, c'est un rythme et une habitude à prendre.

2 – Perspectives d'avenir à court et à long termes

Pour être honnête, je me suis senti parfaitement intégré et à ma place sur mon poste au sein de l'équipe AIDA. Si ça n'avait tenu qu'à moi, je serais resté avec grand plaisir dans l'équipe, sur un CDI. Par ailleurs, mon maître d'apprentissage a tenté d'ouvrir un poste pour m'offrir cette opportunité, qui malheureusement s'est soldée sur un échec.

C'est à cause du contexte difficile dans lequel évolue actuellement Alcatel-Lucent que je me suis permis de chercher à signer un contrat dans une autre entreprise. J'ai bien entendu pris soin d'avoir une discussion avec mon maître d'apprentissage pour lui expliquer ce choix, ce qui était selon la moindre des politesses à avoir à son égard ; et il a été très compréhensif.

Ainsi, au début du mois de mars 2014, j'ai commencé mes recherches en soignant tout les supports possibles permettant de présenter mon Curriculum Vitae. J'ai dans un premier temps réalisé un CV très soigné et complet. Dans un second temps, j'ai soigné ma présence sur les réseaux sociaux professionnels : profils LinkedIn et Viadeo complétés au maximum, CV en ligne sur le site DoYouBuzz, etc. Et pour me différencier des autres développeurs Android, j'ai fait le choix de développer une application Android présentant mes diplômes, expériences et compétences de manière intuitive, distribuée et téléchargeable sur tous les mobiles Android sur le site Google Play.



Images [11] [12] [13] : Exemple des différents supports de mon Curriculum Vitae, de gauche à droite : mon CV papier, mon CV-application Android et ma page LinkedIn.

Une fois tous mes supports en place, j'ai orienté mes recherches suivant plusieurs axes afin de m'assurer d'obtenir un maximum de résultats, à partir de mi-mars 2014 :

- **La recherche classique** d'entreprises dans la région de mon choix (l'Île-de-France en l'occurrence) et la prise de contact spontanée par courrier postal, ou bien sur leurs sites dédiés au recrutement ;
- **La réponse aux annonces** postées sur les plus grands sites de recherche d'emploi ;
- **Le contact actif d'entreprises** et de recruteurs par le biais des réseaux sociaux professionnels ;
- **La prise de contact passive** de recruteurs (principalement des « chasseurs de têtes ») ayant visualisé mon CV ou mes profils en ligne ;
- **Le partage** par tous les moyens mis à ma disposition de mon CV papier et application Android.

A la suite de quoi, j'ai été contacté par de nombreux recruteurs (responsables de ressources humaines ou de recrutement, ingénieurs d'affaires…) à raison, en moyenne, de 2 appels ou mail reçus par jours pendant près de 2 mois.

En outre, j'ai eu l'occasion de passer beaucoup d'entretiens téléphoniques dans un premier temps, puis des entretiens physiques dans les entreprises ayant le plus éveillé ma curiosité. La plupart étant des boîtes de prestation de services :

Alten - Astek - Altran - Extia - AGAP2 - Valtech
ConceptIT - Oberthur - Milibris - OnePoint

Mais également entreprises à grand compte :

ID App - Parrot - Thalès - Safran - BNP - H42

Lors de ces entretiens, j'ai bien entendu eu l'occasion de développer mes motivations et mes ambitions. Mais j'ai également réalisé de nombreux tests permettant d'attester de mes compétences. La plupart des postes ou missions qui m'étaient proposées concernaient du développement JAVA SE et/ou Android, mais également du développement de bas-niveau, type microcontrôleurs.

Je me suis enfin vu proposer de nombreuses promesses d'embauches (dont 6 dans des sociétés de prestation de service, et 2 dans des entreprises). Mon choix s'est finalement porté sur la société Parrot, qui m'a proposé un CDI à partir du 1er octobre 2014, que j'ai signée le 16 juin dernier. Le projet et les avantages de travailler dans cette société en pleine expansion on fait de celle-ci mon premier choix, ce depuis le début de mes recherches.

Ainsi, je serais amené à travailler en tant qu'Ingénieur en développement sur la gamme des ordinateurs de bords de véhicules haut de gamme, sous Android. Je mettrais alors en application mes compétences en développement d'applications Android, en JAVA et en C/C++, mais également en développement de modules et interfaces au sein même du système Android dans l'optique de communiquer avec les composants physiques du véhicule (commandes au volant, boutons, voyants, périphériques Bluetooth pour l'automobile…) Ce dernier point implique que je serais amené à travailler également sur des microcontrôleurs et sur l'aspect électronique du projet.

A court terme, je souhaite continuer à travailler sur l'aspect technique. Ce poste chez Parrot va me permettre d'apprendre davantage de concepts en développement d'applications Android et tout ce qui s'en suit, car c'est ce qui me passionne le plus à ce jour.

Néanmoins, à long terme, j'envisage une évolution vers des branches orientées management et gestion de projet. En effet, je sais faire preuve de toute la rigueur nécessaire pour gérer un projet au sein d'une entreprise comme Parrot : ma grande chance est que cette entreprise semble pouvoir me permettre de prendre très rapidement de prendre un tel virage.

5

Annexes

V. – ANNEXES

1 – Crédits images

Cette page réfère les crédits des différentes images utilisées au cours de ce rapport dont les droits ne me reviennent pas.

[Page de garde] (en haut) *Logo Alcatel Lucent*, 11 juillet 2013, http://www.alcatel-lucent.fr/

[Page de garde] (en bas) *Digital Planisphere*, 07 juin 2014, http://commercemondial.com/resources/banner_img3.jpg

[Page Chapitre 1] (Présentation de l'entreprise) *Alcatel-Lucent : petites-cellules, de nouveaux horizons pour la couverture réseau*, REUTERS/Albert Gea, 11 juillet 2014, http://www.alcatel-lucent.fr/solutions/petites-cellules

[Page Chapitre 2] (Missions confiées et réalisées) *Alcatel-Lucent, At the speed of ideas*, 15 juin 2013, http://www.alcatel-lucent.fr/

[Page Chapitre 3] (Héritage technique) *Alcatel-Lucent, home presentation, NJ,* David Lucantoni, 2011, http://www.dltconsulting.com

[Page Chapitre 4] (Conclusion) *Open-Cloud, le choix de l'innovation*, 13 avril 2014, http://www.alcatel-lucent.fr/

[Page Chapitre 5] (Annexes) *At the speed of ideas*, 10 août 2010, document interne à Alcatel-Lucent

[1] *Implantation mondiale d'Alcatel-Lucent*, Brejnev, 7 février 2009, http://commons.wikimedia.org/wiki/File:Alcatel-Lucent_global_locations.JPG

[2] *Alcatel-Lucent at a glance*, 10 août 2010, document interne à Alcatel-Lucent

[3] *Implantation française d'Alcatel-Lucent*, 11 juin 2014, document interne à Alcatel-Lucent

[4] *Alcatel-Lucent Villarceaux*, 13 mai 2014, Le Républicain, http://www.le-republicain.fr/images/ACTUALITES/Locale/NORD/Nozay/nozay%20alcatel.jpg

[5] *Mobile phone evolution, respectively, from left to right: Sagem 8900X-2, Nokia 3310 orange 5.1, Nokia 3210, LG cookie, Samsung Galaxy S5*, 3 juillet 2014, http://commons.wikimedia.org/wiki/File:Mobile_phone_evolution.jpg

2 – Références bibliographiques

Cette page réfère les différentes sources utilisées lors de la rédaction de ce rapport.

[ref.1] *Alcatel se paye Lucent pour 13 milliards de dollars*, Julien, 3 avril 2006, http://www.clubic.com/actualite-33387-alcatel-se-paye-lucent-pour-13-milliards-de.html

[ref.2] *Alcatel-Lucent*, Wikipedia, dernière mise à jour le 25 juin 2013, http://fr.wikipedia.org/wiki/Alcatel-Lucent

[ref.3] *Laboratoires Bells*, Wikipedia, dernière mise à jour le 6 juillet 2013, http://fr.wikipedia.org/wiki/Laboratoires_Bell

[ref.4] *Document interne Alcatel-Lucent*, 13 septembre 2012

[ref.5] *Alcatel-Lucent : le site de Nozay dans l'œil du cyclone*, La Tribune, 1 novembre 2013, http://www.latribune.fr/technos-medias/electronique/20131101trib000793590/alcatel-lucent-le-site-de-nozay-dans-l-oeil-du-cyclone.html

[ref.6] *Alcatel-Lucent : les conséquences de la restructuration sur les 6 sites français*, Reynald Fléchaux, 8 octobre 2013, http://www.silicon.fr/alcatel-lucent-consequences-restructuration-6-sites-france-89910.html

[ref.7] *Alcatel-Lucent pourrait fermer rapidement 2 sites en France*, Maryse Gros, 8 octobre 2013, http://www.lemondeinformatique.fr/actualites/lire-alcatel-lucent-pourrait-fermer-rapidement-2-sites-en-france-55280.html

[ref.8] *Résultats financiers du 2ème trimestre 2014*, Alcatel-Lucent, 31 juillet 2014, http://www.alcatel-lucent.fr/investisseurs/resultats-financiers/q2-2014

[ref.9] *Téléphonie mobile*, Wikipédia, dernière mise à jour le 6 juillet 2013, https://fr.wikipedia.org/wiki/T%C3%A9l%C3%A9phonie_mobile

[ref.10] *Chiffres clés : les ventes de mobiles et de smartphones*, ZDnet, 2 juillet 2014, http://www.zdnet.fr/actualites/chiffres-cles-les-ventes-de-mobiles-et-de-smartphones-39789928.htm

[ref.11] Chiffres clés : les OS pour smartphones, ZDnet, 2 juillet 2014, http://www.zdnet.fr/actualites/chiffres-cles-les-os-pour-smartphones-39790245.htm