Pour : M. Denis MALLET (Maître de stage)
M. Patrick MAGNAUD (Enseignant-tuteur)

Mathieu SOUM

Stage en entreprise chez
MEMOBOX/2G TECHNOLOGIES
Du 10 avril 2012 au 22 juin 2012

Rapport de Stage

Développement d'un outils multi-protocoles d'acquisition de données télécom

MEMOBOX/2G TECHNOLOGIES
12, Bvd de l'Europe
31850 – MONTRABÉ

Pour : M. Denis MALLET (Maître de stage)
M. Patrick MAGNAUD (Enseignant-tuteur)

Mathieu SOUM

Stage en entreprise chez
MEMOBOX/2G TECHNOLOGIES
Du 10 avril 2012 au 22 juin 2012

Rapport de Stage

Développement d'un outils multi-protocoles d'acquisition de données télécom

MEMOBOX/2G TECHNOLOGIES
12, Bvd de l'Europe
31850 – MONTRABÉ

Remerciements

Tout d'abord, je souhaite remercier Mme. Olga Bensadoun, directrice des études du département Informatique de l'IUT pour m'avoir proposé ce stage de fin d'étude.

Je souhaite ensuite remercier l'entreprise Memobox/2G Technologies, en particulier M. Denis Mallet, mon maître de stage, pour m'avoir accepté et pour m'avoir accompagné tout au long de ces 10 semaines de stage.

Je tiens également à remercier M. Patrick Magnaud, enseignant-tuteur, pour nous avoir rendu visite au cours de ce stage.

J'ai également une pensée pour mes collègues de Memobox/2G Technologies, qui ont su m'orienter et m'aider à bien m'intégrer au sein de l'entreprise tout en gardant une atmosphère amicale.

Plus particulièrement Antoine de Roquemaurel, étudiant à l'IUT, effectuant son stage dans la même entreprise, pour avoir su apporter ses conseils et une dose d'humour à nos pauses café.

Table des matières

Chapitre 1

Introduction

Pour mon stage de fin de diplôme, je souhaitais me lancer dans un projet concret dont je pouvais saisir les proportions et qui m'apporterais une expérience dans le language de programmation utilisé. Sur les trois offres de stages qui m'ont été proposées, j'ai choisi celle de l'entreprise Memobox/2G Technologies pour plusieurs raisons :

- Le language utilisé
- Le sujet traité
- Le lieu

En effet, j'affectionne particulièrement la rigueur et la précision que demande le développement en C/C++. Contrairement aux languages des autres sujet (essentiellement du Java), j'ai une plus grande expérience dans le C++. De plus, je trouve que le ce language offre de plus grandes possibilités dans la mesure où on accède à des outils de bas niveau et donc il est possible de manipuler des données binaire sans trop de difficultés, relativement à d'autres languages. Ce qui pour ce sujet était indispensable.

Le sujet, c'est le développement d'un outils d'acquisition de données télécoms multi-protocoles en C++. C'est-à-dire que cet outil devra récupérer des données transmises via une interface de télécommunication pour les stocker sur le disque. Ces données sont des 'tickets' qui représentent des données sur une communication, comme un coup de fil par exemple. Or, il en existe de plusieurs formats en fonction des fabricants et des opérateurs. Il nous faut donc récupérer ces tickets, normalement envoyés sous format texte, comme s'ils étaient des fichiers binaires afin de ne pas être perturbé par des conventions de caractères de fin de ligne, différentes selon les systèmes, ou des caractères de contrôles de communication parasites qui pourraient engendrer la perte de données. Ces caractères de communications varient en fonction du protocole utilisé et peuvent parfois nous gêner dans la récupération correcte des données. C'est pourquoi, il faut que l'outil puisse s'adapter aux différents protocoles de communication que l'on peut rencontrer chez les clients.

Le lieu du stage étant sur Toulouse, je n'ai pas eu besoin de chercher un nouvel appartement pour me loger près de mon lieu de stage. En effet, les locaux se situent à une vingtaine de minutes de chez moi.

Pour ce qui est de la structure de ce rapport, dans un premier temps, nous parlerons de l'entreprise Memobox/2G Technologies. Ensuite, de la mission effectuée pendant ce stage et plus en détail du projet étudié, de la solution adoptée pour répondre à toutes les problématiques de performances et de fonctionnalités, puis des résultats obtenus en fonction des objectifs principaux.

Chapitre 2

Contexte professionnel

2.1 L'entreprise Memobox

La société Memobox est présente depuis 1994 sur le marché des télécommunications en entreprise. Créée à l'origine pour apporter aux professionnels des solutions de communication sécurisée entre les PABX [1] et les outils micro-informatiques de gestion, elle a pris un virage important en 1997, en s'orientant vers le service, grâce à des solutions hébergées de taxation et d'analyse de trafic téléphonique. L'évolution du marché et des solutions techniques lui a permis de développer une gamme complète de services de Gestion Financière des Télécommunications (GFT), couvrant aujourd'hui l'ensemble du domaine financier des télécoms en entreprise : téléphonie fixe, téléphonie mobile, VoIP, ToIP, données, équipements, contrats associés, abonnements, consommations, ... En 2007, l'entreprise réalise sa première opération de croissance externe en prenant le contrôle de 2G Technologies, société spécialisée dans l'édition et la vente de logiciels de gestion financière des télécoms (logiciel `GeoTaxe ES`/`GeniT@x`).
Elle réalise également une levée de fonds d'un million d'euros, ce qui lui permet d'accroître encore son avantage concurrentiel en accentuant ses investissements en R&D, en renforçant ses équipes et en déployant une stratégie commerciale adaptée aux attentes de ses clients en France comme à l'international.
Memobox se positionne, aujourd'hui, comme le leader des fournisseurs de solutions de Gestion Financière des Télécommunications sur le territoire français. L'entreprise compte de très nombreux clients, notamment Air France-KLM, Generali, AXA, MACIF, La Poste, les Ministères de l'Agriculture, de l'Intérieur et des Finances, Veolia.

2.1.1 La mission

La mission de Memobox est de proposer aux entreprises des logiciels et des services pour les aider à observer les usages, à réduire leurs coûts télécoms et à optimiser leurs installations télécoms.
Forte de son expérience sur le marché, Memobox propose à ses clients une gamme complète d'appliances, de logiciels et de services performants et adaptés aux évolutions permanentes du marché des télécoms.
Memobox est à l'écoute des entreprises pour les conseiller et leur proposer les solutions les mieux adaptées à leurs besoins. Dans cette optique, elle propose de nombreuses alternatives à ses clients en termes de gammes (logiciels pour TPE/PME, grands comptes, hôtels, hôpitaux) mais aussi en termes de modes d'installation (appliance, logiciels ou services externalisés).
Pour compléter l'offre, des prestations d'administration et d'audit des outils sont proposées aux

1. **P**rivate **A**utomatic **B**ranch **E**xchange. (Commutateur d'échanges privés)

clients pour leur permettre de se concentrer sur leur coeur de métier et bénéficier de conseils d'experts avisés.

2.1.2 L'historique

1994 Création de MEMOBOX.
A l'origine de MEMOBOX, deux spécialistes, l'un des services de télécommunications : Christophe FORNÈS, l'autre des services réseaux : Denis MALLET. Ensemble ils décident en 1994 de développer et commercialiser leur propre ligne d'interfaces de communication pour PBX [2] : `EdelBox`. Ces interfaces sont commercialisées auprès d'éditeurs de logiciels de taxation et d'analyse de trafic téléphonique.

1995 Développement commercial.
Poussée par ses premiers clients, MEMOBOX étend sa gamme en créant un système de taxation téléphonique autonome pour les TPE/PME : Taxabox, décliné également en version hôtelière (Hotelbox). Pour se faire, elle embauche son premier salarié pour étendre ses capacités de développement.

1997 Le pari du service avec AUDITELcom®.
A cette date, MEMOBOX compte 2 salariés. Après le succès rencontré auprès des éditeurs de logiciels intégrateurs en télécom, MEMOBOX pressent le besoin des entreprises en matières de gestion des coûts télécoms et pour s'adapter à son marché, décide de lancer AUDITELcom®, le premier service de GFT [3] en mode hébergé. Une augmentation de capital accompagne ce nouvel axe de développement.

2003 MEMOBOX renforce sa position.
Le déploiement des 800 premiers sites nationaux du ministère de l'Intérieur est bouclé en 4 mois et le service satisfait pleinement le client. Pour réaliser cette prouessen MEMOBOX a consolidé ses équipes de développement et d'exploitation. L'effectif est porté à 13 personnes.

2004 Les deux fondateurs de MEMOBOX rachètent la totalité de l'entreprise.
La bonne santé de MEMOBOX en 2004 permet à Christophe FORNÈS et Denis MALLET de racheter à Global Concept la totalité des parts que ce dernier détenait dans MEMOBOX. La société est désormais filiale à 100% du groupe TEL&COM FM (Facilities Management) dont Christophe FORNÈS et Denis MALLET sont les deux seuls actionnaires.

2007 MEMOBOX renforce sa stratégie
En octobre 2007, elle rachète 2G TECHNOLOGIES à sa maison mère, QUESCOM.
En décembre 2007, elle lève 1 million d'euros aurpès d'ALTO Invest.

2.1.3 Les produits

MEMOBOX/2G TECHNOLOGIES a développé les gammes de logiciel :

`AUDITELcom` Plateforme complète de TEM [4] en mode SaaS [5] ou logiciel.

`GEODITEL` Logiciel de contrôle du trafic téléphonique entrant ou sortant pour les TPE/PME.

`GEOTAXE ES` Progiciel de Call Acounting pour les grandes entreprises.

`GEOVOX` Serveur d'application de messagerie vocale.

2. **P**ublic **B**ranch **E**xchange
3. **G**estion **F**inancière des **T**élécoms.
4. **T**ransformation **E**ngine **M**odule
5. **S**oftware **a**s **a** **S**ervice

2.2 Le site de Toulouse

2.2.1 Le matériel

Pour ma part, j'utilisais deux machines, une fixe et une portable. Elles tournaient toutes les deux sous environnement Windows XP SP3. La machine fixe était un ordinateur Dell avec un processeur Intel® Core™ 2 CPU 6300 avec deux cœurs cadencés à 1.86GHz, et 2.49Go de RAM.

Les environnements de développement intégrés Eclipse et VisualStudio 2008 et 2010 étaient installés sur la machine fixe, principalement utilisée pour le développement. La machine portable fonctionnait sous Windows XP SP3 et comprenait des programmes de tests générant des connexions et du transfert de contenu vers la machine fixe de développement pour les différents tests.

Pour le travail collaboratif, nous utilisions le logiciel de versionnement SVN [6] afin de garder un historique des changements en cas de problème afin de pouvoir revenir rapidement à une version stable en enlevant le code qui pose problème. De plus, le système se charge automatiquement de synchroniser les derniers changements effectués et validés par les autres collaborateurs à chaque lancement de mon environnement de développement.

2.2.2 Le personnel

Nous étions 6 sur le site de Toulouse :

Romain Auriac Ingénieur développement Web

Philippe Dureux Ingénieur Télécoms

Jean-Marie Lagarde Responsable R&D

Denis Mallet Vice Président, Directeur Technique, Maître de stage

Antoine de Roquemaurel Étudiant en stage, Développement Web

Mathieu Soum Étudiant en stage, Développement client lourd C++

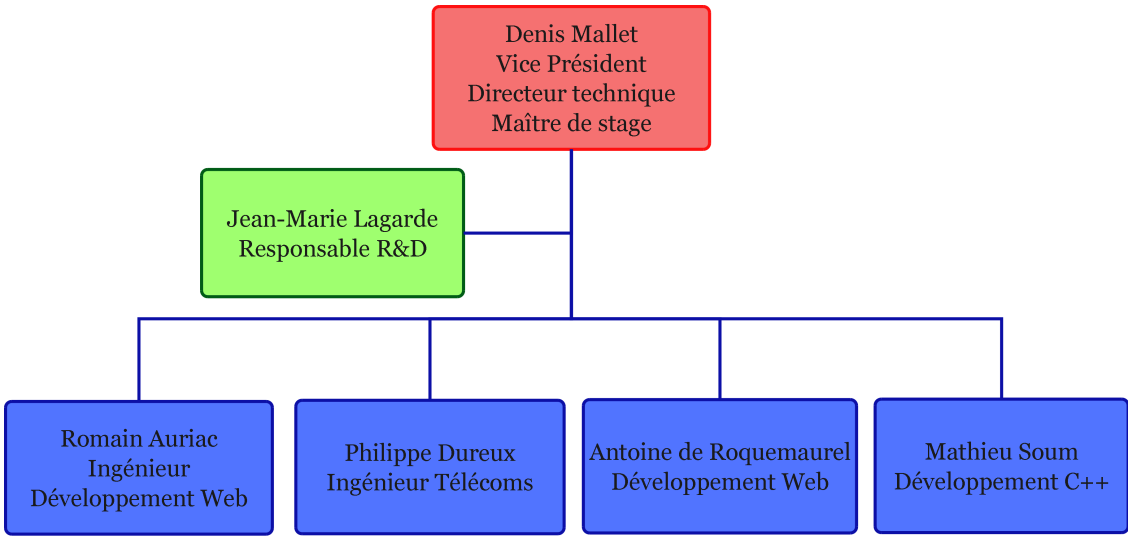


Figure 2.1 – Organigramme

6. Abréviation de `Subversion`.

2.2.3 Horaires

Les horaires de travail étaient assez flexibles et il n'était pas exigé de présence particulière à des horaires précis. Cependant, il est nécessaire qu'un employé soit présent de 9h30 à 12h et de 14h à 17h30 afin d'assurer un service d'assistance téléphonique auprès des client. Nous commencions donc aux alentours de 9h30 le matin. Nous nous arrêtions vers les 12h30 pour la pause déjeuner. Nous reprenions entre 13h30 et 14h pour finir la journée à 18h30. Nous travaillions du Lundi au Vendredi et ces horaires ne variaient pas (ou très peu) d'un jour à l'autre.

Chapitre 3

Mission effectuée

3.1 Le projet étudié

Durant ce stage, j'ai participé à la création d'un nouvel outil `MBX_Stream`, basé sur un système existant, `GeoditelGateway`, inclut dans `GEODITEL` en tant que module. La majeure partie de la conception était déjà établie à mon arrivée, cependant l'ancienne version ne répondait pas aux nouvelles attentes et à l'incorporation de nouveaux protocoles de communication.

3.1.1 Description du concept

`GeoditelGateway` est un module qui communique avec les autres modules d'administration et de contrôle de `GEODITEL`. `GEODITEL` est destiné aux TPE/PME, leur permettant de :

- contrôler les consommations de téléphonie fixe,
- vérifier la qualité de leur accueil téléphonique,
- être alerté sur des événements exceptionnels,
- facturer aux clients le temps passé avec leurs collaborateurs,
- disposer de rapports automatiques reçus par email.

`GeoditelGateway` est chargé de récupérer les différents tickets venant d'un PABX pour les envoyer au module d'analyse afin de générer des rapports et/ou des statistiques. Le mot 'ticket' est

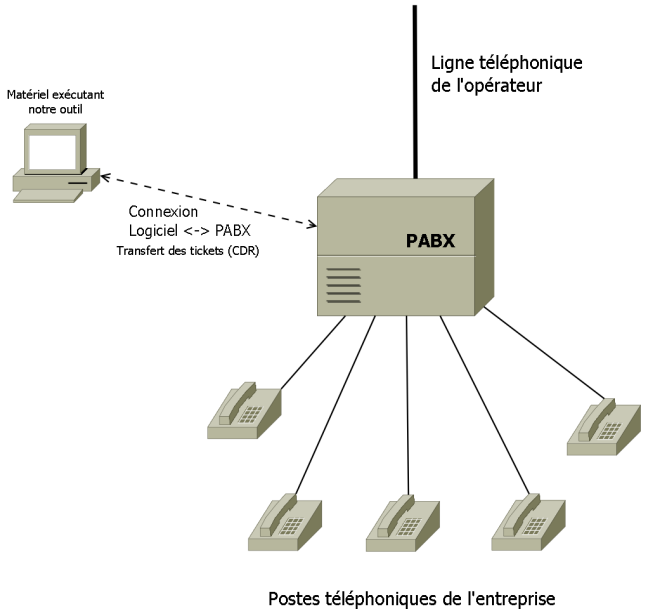


FIGURE 3.1 – Schema de connexion avec un PABX

une adaptation française du nom original CDR [1]. Ces CDR sont générés par un PABX, que l'on pourrait définir comme un serveur faisant office de routeur pour acheminer les communications téléphoniques vers les différents postes d'une entreprise, comme le ferait un serveur informatique avec des connexions réseau. Ce sont les données de ces appels, comme par exemple le temps de communication, le nombre de sonneries avant décrochage, le numéro appelant, le numéro appelé, etc., que nous allons récupérer sous forme de tickets formatés selon le modèle de la machine. (cf. figure 3.1)

3.1.2 Etat initial du logiciel

Du point de vue technique, l'application actuelle est codée en C/C++ mais n'exploite que très peu le concept de programmation objet qu'offre ce language. De nombreux algorithmes sont basés sur des technologies C alors que des algorithmes plus optimisés existent déjà dans la STL [2] du C++.

Pour les communications réseaux, les connexions de type TCP [3] (serveur et client) et UDP [4] sont opérationnelles. Ces fonctionalités ont été mises en place à l'aide d'un framework de développement baptisé `Poco`. Il fournit des classes et des interfaces pour implémenter facilement des clients et serveurs TCP ou mettre en place des sockets de communication UDP. L'avantage principal de ce framework est de faire abstraction du système sur lequel est exécuté l'application en utilisant des outils "cross-platform", c'est-à-dire fonctionnant autant dans le monde Windows que dans le monde UNIX sans avoir de duplication de code et donc permettre un développement plus facile et une maintenance plus efficace.

Les communications via le port série (RS232) fonctionnent sous FreeBSD (UNIX) via l'utilisation d'une bibliothèque d'émulation de terminal, `termios`. Cependant, cette fonctionnalité ne fonctionne pas de la même manière sous Windows. On voit ici un des problèmes lors du développement d'application étant déployées sur plusieurs systèmes.

L'application utilise la gestion des `threads` systèmes afin d'effectuer plusieurs actions simultanément et permettre de commander l'application grâce au module d'administration sans pour autant mettre en pause les connexions de données arrivant du PABX. La gestion des threads système est elle aussi légèrement différente entre Windows et UNIX. Là encore, une multiplication de code servant à effectuer la même chose entraîne une difficulté de maintenance et un ralentissement du développement.

La figure 3.2 montre le diagramme de classes initial. De nombreuses méthodes n'y apparaissent pas car elles ne sont pas encapsulées dans des classes.

3.1.3 Objectifs à terme

Les objectifs au terme du stage concernant `MBX_Stream` peuvent être séparés en 3 catégories :

- les fonctionnalités
- les performances
- la stabilité

Les fonctionnalités attendues en plus de celles déjà présentes sont la prise en charge du protocole FTP [5] côté client pour les communications vers le(s) PABX, la possibilité de gérer plusieurs connexions vers plusieurs PABX utilisant des protocoles de connexions différents, la possibilité de tranférer des fichiers binaires, comme des exécutables ou du contenu multimédia sans perte de données et la rotation des fichiers générés contenant les tickets reçus depuis le(s) PABX. Un système

1. **C**all **D**ata **R**ecord (Enregistrement de données d'appel)
2. **S**tandard **T**emplate **Library**
3. **T**ransfert **C**ontrol **P**rotocol
4. **U**ser **D**atagram **P**rotocol
5. **F**ile **T**ranfert **P**rotocol

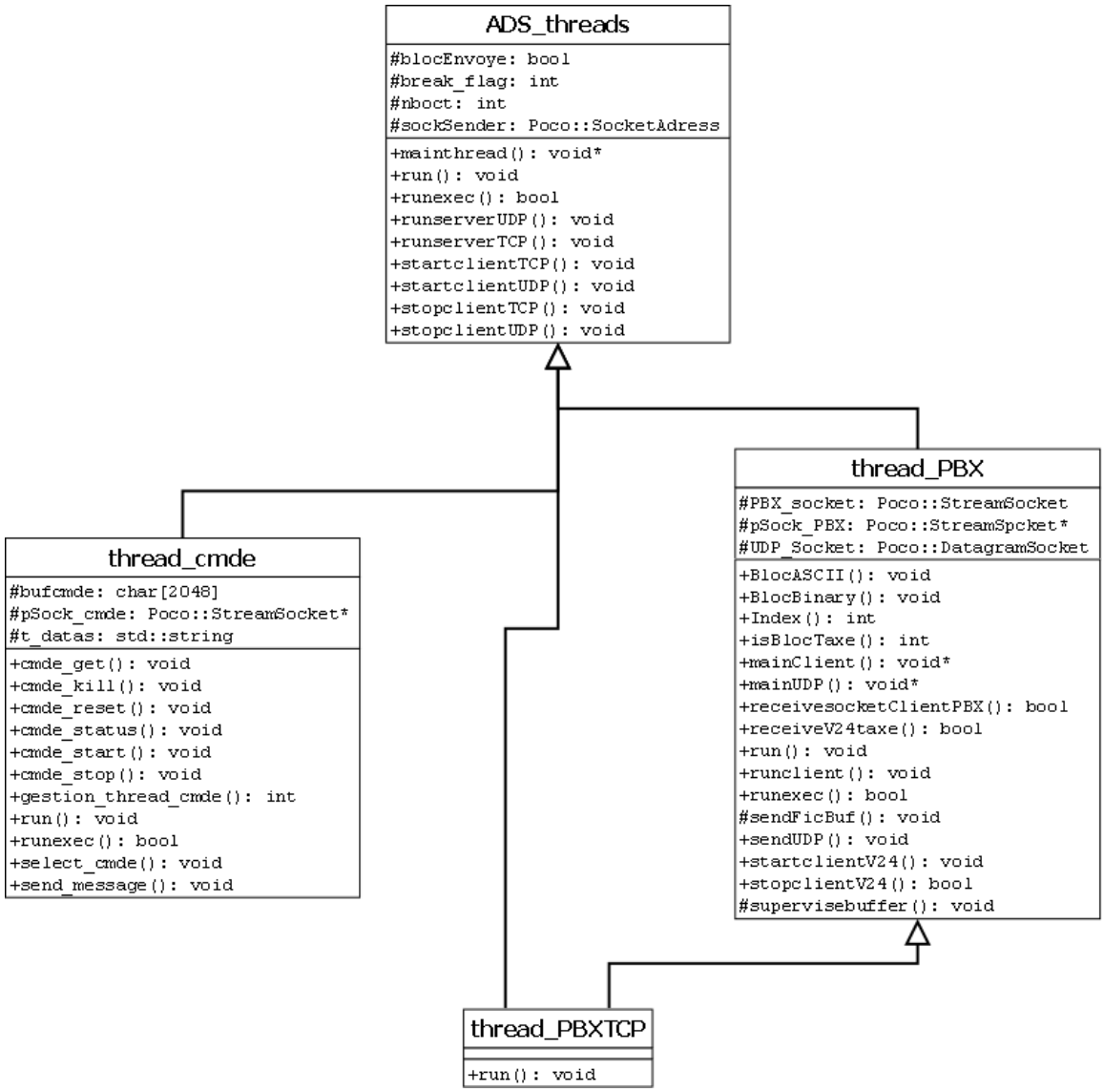


FIGURE 3.2 – Diagramme de classes UML initial

de rotation (archivage) des fichiers contenant les tickets reçus sera également mise en place afin de fournir des fichiers réguliers à la partie analyse. Les critères de rotations seront le temps depuis la dernière rotation et/ou la taille du fichier.

Les performances quant à elles doivent être améliorées afin d'éviter une charge inutile notamment lorsque le programme attend des données des PABX. Un piège serait d'éffectuer ces contrôles de présence de flux de données à des intervalles de temps assez cours mais suffisamment long pour que le programme se mette en sommeil entre deux contrôle. En effet, une telle conception ferait perdre un temps précieux lors des communications de gestion qui peuvent demander l'arrêt ou la mise pause du programme et des connexions ouvertes. Celles-ci seraient décalées, même de quelques dixièmes de secondes, ce qui pour un ordinateur est énorme, et engendreraient une potentielle perte de données.

La stabilité se décline en trois cas distincts. Pour de multiples raisons, comme la modification de la configuration par exemple, le logiciel peut être ammené à redémarrer un certains nombre de fois dans un délai très court. A l'heure actuelle, l'outil ne fonctionne que sur un mode synchrone, c'est-à-dire qu'il ne peut répondre ou effectuer une action qu'à des intervalles de temps réguliers, très courts certes mais pas immédiats. Un des objectifs de stabilité est donc de rendre le fonction-

nement de cet outil asynchrone afin que les temps de réponse, d'acceptation de connexion, et de traitement des différentes actions demandées soit effectués immédiatement après la demandes de ces actions.
De ce fait, il faux que les différentes connexions soient fermées proprement, les ports de communication de nouveau disponibles pour d'autres applications et la mémoire allouée dynamiquement pour l'exécution du programme libérée et rendue au système. De plus, le logiciel devra tenir les charges de connexion et ne pas saturer lorsque de nombreuses connexions sont activées simultanément. Enfin, il devra également supporter un durée d'exécution importante, de l'ordre de plusieurs semaines voir même de plusieurs mois, sans avoir besoin de réinitialisation et sans provoquer de fuites de mémoire.

3.2 La solution adoptée

Dans l'optique d'atteindre les objectifs cités ci-dessus, la conception a été revue pour factoriser le maximum de code et ainsi éviter duplication et difficultés de maintenance et/ou d'évolution du code. La figure 3.3 page 15 montre la nouvelle conception choisie grâce à un diagramme de classe.

On peut voir que le nombre de classes a augmenté, ce qui signifie que le code sera remonté le plus haut possible dans l'arbre des classes et que les classes feuilles seront spécialisées dans un traitement précis. Ainsi, on peut identifier plus facilement le code défectueux car il est ordonné de manière plus rigoureuse.

Pour nous aider dans le développement de notre outil, nous allons utiliser un framework C++ fournissant un grand nombre d'interfaces pour implémenter des applications multi-plateforme, `Poco`.
`Poco` est développé par AppliedInformatics et se défini comme :

> "Modern, powerful open source C++ class libraries and frameworks for building network- and internet-based applications that run on desktop, server and embedded systems."

des frameworks et des bibliothèques de classes modernes, puissantes et open source pour créer des applications réseau et internet pour des ordinateurs, des serveurs ou des systèmes embarqués.
Mais `Poco` c'est aussi tout un utilitaire de gestion des systèmes de fichiers, d'analyse de fichiers ou de flux XML, de création d'archives compressées, de gestion des threads, et de bien d'autres aspects que nous n'utiliserons pas dans notre outil comme la gestion des bases de données en interfaçant des systèmes comme ODBC, MySQL ou SQLite. L'avantage d'avoir tous ces outils dans un seul grand ensemble applicatif est qu'ils suivront tous la même logique et pourront s'utiliser les uns avec les autres. Par exemple, pour la gestion des threads, `Poco` nous fournis une classe abstraite que l'on devra réimplémenter pour préciser le traitement qui sera exécuté par le fil d'exécution. Et bien la plupart des classes et interfaces réseaux héritent déjà de cette classe abstraite. Ainsi, une fois le traitement spécifié, nous pourrons gérer l'exécution de ce traitement par nous-même, ou dire à `Poco` que nous voulons l'exécuter dans un thread distinct.

La plupart des interfaces des bibliothèque de `Poco` sont accompagnées d'une implémentation rudimentaire qui peut suffire dans certains cas. Cependant, il nous sera nécessaire de réimplémenter certaines des méthodes de ces classes afin qu'elles effectuent les traitements qui correspondent à nos besoins.

`Thread_PBX` et les classes déscendantes sont chargées d'effectuer des traitements précis en fonction du type de connexion. Elles héritent de `Poco::Runnable` qui fourni la méthode abstraite `run()`, point d'entrée d'un thread. (cf. section 3.2.3)
`Thread_PBX` est une classe abstraite qui généralise le stockage et l'archivage sur le disque des fichiers contenant les tickets reçus.

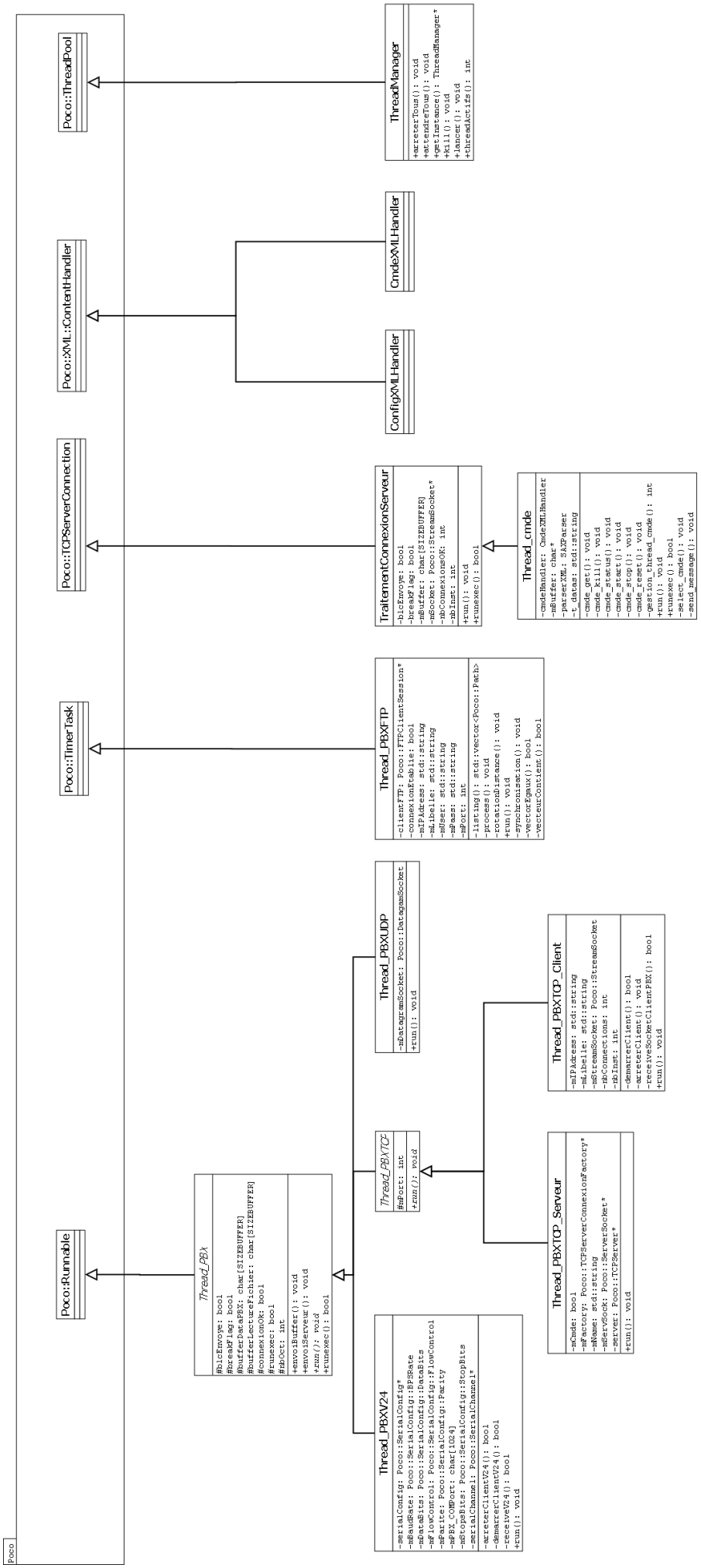


FIGURE 3.3 – Nouveau diagramme de classes

La classe `Thread_PBXFTP` est à part car elle est un peu particulière. Certes elle représente le traitement d'une connexion mais son fonctionnement n'est pas le même, au niveau du protocole de communication, au niveau des données récupérées, où au niveau de sa fréquence d'exécution. (cf. section 3.2.5)

`TraitementConnexionServeur` et `Thread_cmde` n'héritent pas directement de `Poco::Runnable` mais de `Poco::TCPServerConnexion`. Cette classe de `Poco` permet, une fois passée en paramètre à un `Poco::TCPServer` de lancer des traitements dans un thread différent à chaque nouvelle connexion à ce sereur. La différence entre nos deux classes est que la première sert à récupérer des tickets, alors que l'autre sert à récupérer et traiter les commandes.

`ConfigXMLHandler` et `CmdeXMLHandler` héritent de `Poco::XML::ContentHandler` et réimplémentent les méthodes fournies pour s'adapter à notre cas. La première permet à l'analyseur XML de manipuler les données du fichier de configuration et de valoriser les bonnes variables de notre programme. Quant à la seconde, elle sert à manipuler les données contenues dans la chaîne XML transmise lors d'une commande.

`ThreadManager` hérite de `Poco::ThreadPool` et nous fourni des outils pour gérer nos threads (lancement, arrêt, nombre d'actifs, nombre restant disponibles, etc.). (cf. section 3.2.3)

3.2.1 Les communications réseau

Les communications réseau obéissent à un modèle que l'on appelle modèle OSI [6] proposé par l'ISO [7]. Ce modèle s'étend sur 7 couches. Le message que l'on souhaite envoyé part d'un certaine

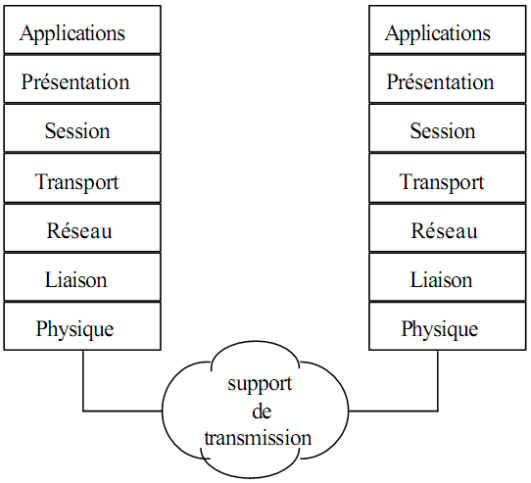


FIGURE 3.4 – Les 7 couches du modèle OSI

couche (Application dans la plupart des cas) et est encapsulé dans chaque couche en descendant vers la couche Physique pour constituer une trame qui transitera via le support de transmission. Une fois chez le destinataire, cette trame est décapsulé dans le sens inverse, toujours en suivant le même modèle, pour être traitée par les couches successives et enfin lu par la couche correspondant au message initial. Pour notre part, nous travaillerons essentiellement sur la couche Transport en travaillant avec les protocoles TCP/UDP ou V24. Seul le protocole FTP appartient à la couche Application et nécessite par conséquent une gestion plus complexe.

6. **O**pen **S**ystems **I**nterconnection
7. **I**nternationnal **O**rganization for **S**tandardisation

Pour l'ensemble des communications, nous allons utiliser le framework `Poco`. Il existe une partie additionnelle à ce framework qui y sera intégrée dans une version ultérieure mais qui est d'ores et déjà fonctionnelle et disponible séparément. Cette surcouche nous permet de gérer les connexions séries sans avoir à passer manuellement par des bibliothèques systèmes ou des outils propres à un compilateur donné. Ceci nous permet de nous libérer du problème déjà évoqué plus haut de la gestion des liaisons séries (RS232) qui diffère selon le système sur lequel est lancé l'application. Grâce à `Poco`, ce n'est plus un problème car on lui délègue cette gestion et c'est lui qui appliquera la bonne stratégie suivant l'environnement de compilation et d'exécution.

`Poco` a également d'autres atouts qui vont nous être utiles. Outre les communications réseau, nous allons nous tourner vers l'analyse de fichiers XML [8] et l'interface de gestion des threads.

3.2.2 Le pilotage et la configuration

Nous utilisons des données XML lors des échanges avec le module de gestion qui fait passer ses requêtes sous la forme :

```
<geoport><cmde><fonction>xxx</fonction><action>xxx</action></cmde></geoport>
```

avec à l'intérieur des balises `fonction` et `action` respectivement la nature du message (exemple `supervision`) et l'action demandée (exemple `status`). Les réponses sont aussi en XML de la forme :

```
<geoport><cmde><action>xxx</action><version>1.0</version>
<status>xxx</status><etat>xxx</etat><porttaxe>xxx</porttaxe>
<servtaxe>xxx</servtaxe></cmde></geoport>
```

avec à l'intérieur de la balise `action` l'action qui vient d'être effectuée et dans les autres balises, différentes selon la requête demandée, des précisions sur l'état du programme ou de l'action effectuée.
Le fichier de configuration est lui aussi en XML. Il comprend les différents paramètres de l'utilisateur comme l'emplacement des fichiers dans lesquels seront stockés les tickets reçus avant et après rotation et les paramètres de communication avec le(s) PABX (type de connexion, adresses IP distantes, ports d'émission ou de réception, paramètres de la liaison V24, identifiants de connexion au serveur FTP distant).

3.2.3 Les threads

Chaque connexion sera executée par un thread différent. La gestion des threads avec `Poco` facilite grandement cette structure. En effet, comme on peut le voir dans le diagramme 3.3, la classe `Thread_PBX`, ancêtre de toutes les classes des modes de communication, hérite de `Poco::Runnable`, une classe conçue pour l'exécution des threads. Cette classe, abstraite, contient une méthode `run()` à réimplémenter dans les classes déscendantes. Cette méthode représente le point d'entrée d'un thread. Ainsi, il suffit de donner à `Poco` un objet héritant de `Poco::Runnable` et réimplémentant la méthode `run()` pour qu'il execute le traitement de cette méthode dans un nouveau thread tout en gardant l'encapsulation à l'intérieur de l'objet donné. Pour passer un objet à `Poco` en tant que traitement de thread, il faut utiliser la classe `Poco::ThreadPool` qui s'occupe de lancer et arrêter les threads de manière autonome tout en laissant la possibilité de demander l'arrêt des threads ou au contraire d'attendre l'exécution d'autres traitement avant l'arrêt des threads. Elle fournit également la possibilité de contrôler l'état des threads actifs ou inactifs comme le nombre de threads actuellement en fonction ou le nombre de threads restant disponibles.

8. e**X**tensible **M**arkup **L**anguage

3.2.4 Les réponses rapides

Reste à réglé le problème de l'attente de communication avant de se tourner vers la partie applicative FTP. Pour cela, nous allons encore une fois nous utiliser les interfaces de `Poco`. Le framework nous donne la possibilité d'utiliser une gestion d'événements qui pourront être lancés ou attendus dans tous les threads de l'application. Ainsi, la demande d'altération de l'état du programme (stop ou reset) déclenchera un événement arrêtant les threads et l'action en elle même ne sera appliquée qu'à la fin des communications gérées par les threads, représentée par un autre événement déclenché lors de la fin de tous les threads. Ainsi, les structure d'attente sous forme d'une boucle avec un sommeil d'une seconde entre chaque tests pour ne pas surcharger le processeur – contrairement à une boucle sans sleep qui effectue un test toutes les quelques nanosecondes et surcharge le processeur par des instructions inutiles – se transforme en une instruction d'attente d'événement plus propre et moins gourmande en ressource que la précédente (cf. figure 3.5).

Avant :

```
bool condition;
// ...
while (condition) {
  sleep(1); // Attente d'une seconde
}
```

Après :

```
Poco::Event evenement;
// ...
evenement.wait();
```

FIGURE 3.5 – Mise en place d'un événement

Ce modèle fait de notre outil un outil asynchrone qui répondra dès qu'il recevra une commande. De plus, cette gestion d'événement nous permet de rester dans un état cohérent en arrêtant les threads de communications avant d'en lancer de nouveaux.

3.2.5 La liaison FTP

La gestion du FTP s'est révélée plus complexe à concevoir que prévu. En effet, le FTP est un protocole de communication de niveau applicatif et donc son utilisation abstraction des couches réseaux en masquant le protocole TCP brut dont il est une surcouche. Ce dernier ne se charge que d'envoyer des trames de données à travers un réseau alors que le FTP s'appui sur le système de fichier pour fournir des méthodes plus haut niveau de récupération et de manipulation de fichiers. Vu sous cet angle, ce protocole pourrait sembler plus amical pour le développeur car il lui enlève la gestion réseau, déjà pilotée en arrière plan. Cependant l'existence de ces méthodes dans le protocole même, nous empêche de faire les choses exactement comme on le voudrait. Il nous faut utiliser uniquement les fonctionnalités fournies. Cela nous réduit le développement de la partie connexion réseau, mais alourdi considérablement la partie de gestion des fichiers reçus, des potentielles erreurs de connexion ou de transfert ou de s'assurer que les données n'ont pas été altérées durant le transfert. La partie FTP du projet nécessite donc une plus grande conception algorithmique que les autres qui n'ont pour tâche que de récupérer un flux réseau et de l'écrire dans un fichier. Nous avons donc établi plusieurs scénarios à effectuer en fonction de la configuration faite par le client. Ainsi, nous avons établi 3 scénarios :

Les fichiers distant doivent êtres laissés en l'état. Dans ce cas là, nous devons garder une liste des fichiers déjà récupérés afin de ne récupérer que les derniers fichiers ajoutés sur le serveur. Ensuite, il nous faut effectuer une synchronisation inverse, c'est à dire que les fichiers que nous avons récupérés une précédente fois et qui ne sont plus sur le serveur doivent être supprimés de notre côté afin de ne pas effectuer des traitements inutiles sur des fichiers qui ne sont plus nécessaires. Ce scénario est le plus répandu chez les clients actuels.

Les fichiers distants doivent êtres déplacés ou renommés. Après la récupération, nous devons déplacer les derniers fichiers récupérés vers un repertoire de rotation [9], éventuellement sous un nom différent. Il est également possible d'uniquement les renommés en les laissant dans leur répertoire d'origine.

Les fichiers distants doivent êtres supprimés. Après la récupération, nous devons supprimer les fichiers sur le serveur après s'être bien assuré de l'intégrité des fichiers reçus car nous en aurons l'unique exemplaire. Encore une fois, `Poco` nous viendra en aide en nous fournissant les principales commandes FTP ainsi que des outils de manipulation des systèmes de fichiers pour conserver un outil multi-plateforme.

3.3 Les résultats obtenus

3.3.1 Les communications réseau

Les communications des couches basses (TCP/UDP/RS232) sont opérationnelles et fonctionnent grâce aux outils de `Poco`. La rotation est elle aussi correctement en place et déplace les fichiers contenant les données des tickets reçus selon des règles de taille ou de temps. Les fichiers une fois archivés sont accéssibles dans le dossier spécifié en configuration sous la forme `yyyymmdd_hhmmss_nomFichierOriginal` où, dans l'ordre :

`yyyy` correspond à l'année sur quatre caractères *(ex. 2012)*
`mm` correspond au numéro du mois sur deux caractères *(ex. 06)*
`dd` correspond au jour du mois sur deux caractères *(ex. 18)*
`hh` correspond à l'heure sur deux caractères *(ex. 13)*
`mm` correspond au minutes sur deux caractères *(ex. 37)*
`ss` correspond au secondes sur deux caractères *(ex. 42)*

D'après l'exemple si le fichier `monFichier.txt` avait été archivé le 18 juin 2012 à 13 : 37 : 42, il aurait le nom suivant : `20120618_133742_monFichier.txt`.

Ce format n'est pas facilement lisible pour un humain mais un ordinateur pourra facilement classer ces fichiers par ordre chronologique, simplement en les triant par ordre alphabétique. C'est le principal avantage de ce format de date.

Exemple :

Les fichiers
`20120101_001000_fichier`
`20110101_100000_fichier`
`20120102_020000_fichier`
`20120318_230000_fichier`
sont triés par ordre alphabétique par l'ordinateur très facilement et par ordre chronologique grâce au formatage de la date.

Les trois modes d'archivage sont paramétrables dans le fichier XML de configuration. Une archivage *par la taille* donnera des fichiers de taille identique mais à des intervalles de temps différents. Un archivage *par le temps* donnera des fichiers de taille très variable mais à des intervalles de temps réguliers, avec une précision à la seconde. Un archivage *couplé* donnera soit un fichier de la taille souhaitée, soit un archivage dans le temps si la taille requise n'a pas été atteinte, au bout de la période définie depuis le dernier archivage. (cf. section 3.3.2 pour des détail sur la configuration)

Ces fichiers une fois archivés ne seront plus modifiés et seront donc disponible pour un traitement par une autre application.

9. Répertoire d'archive ne contenant que des fichiers déjà traités.

3.3.2 Le pilotage et la configuration

Le pilotage

L'analyse des commandes reçues par le serveur de commande sous format XML sont maintenant traitées par `Poco::SAXParser`[10] et non plus par des manipulations de chaînes qui demandaient beaucoup de code et qui n'étaient pas très facile à relire pour corriger d'éventuels problèmes. Ainsi, l'ajout de nouvelles commandes ou de nouveaux paramètres à ces commandes en sera grandement facilité par une analyse structurée des données XML.

Les commandes implémentées sont :

`get` Renvoi des 1024 derniers caractères reçus pour vérifier que la connexion avec le PABX est opérationnelle.

`kill` Arrêt complet du système. Nécessite un redémarrage manuel de l'application.

`reset` Fermeture de toutes les connexions ouverte et relancement du processus en réexaminant le fichier XML de configuration.

`status` Renvoi des données relatives à l'état du système.

`start` Démarrage du processus après un `stop`.

`stop` Mise en 'pause' du processus en fermant les connexion et attente d'une commande de redémarrage (`start` ou `reset`).

La configuration

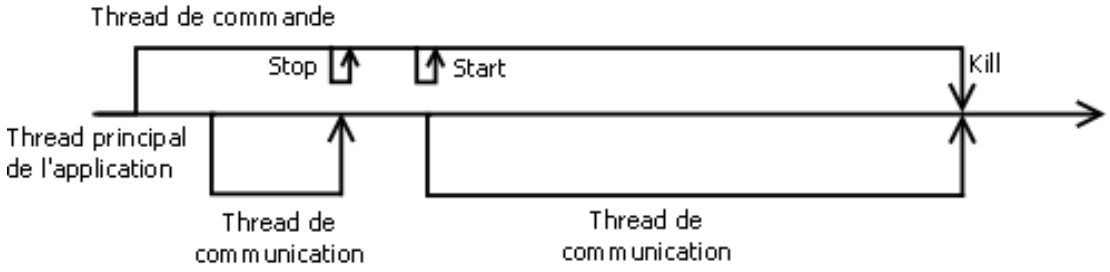
Le fichier XML de configuration a été grandement épuré pour supprimer d'anciennes données, aujourd'hui inutiles. Il contient maintenant 3 sections :

- `parametres` regroupant les paramètres fixés lors de l'installation.
- `general` regroupant les paramètres de trace d'exécution et de rotation du fichier contenant les tickets reçus.
- `buffer` regroupant les paramètres de communications, les données distantes (adresses IP, port, etc.), et tous les paramètres nécessaire à l'établissement du scénario FTP.

Ce fichier sera rempli à l'installation et pourra être modifié par le client via une future interface web qui interrogera notre serveur de commande pour mettre à jour les données du XML.

3.3.3 Les threads

Les threads sont administrés à l'aide de `Poco`. `Poco::ThreadPool` offre un lot de threads 'disponibles' qu'il peut nous fournir pour lancer des traitements précis. Cependant, les outils fournis par cette classes ont été surchargés dans une de nos classes personnelles, `ThreadManager`. La surcharge de ces méthodes via des méthodes propres à notre outil nous permettrons, si un meilleur framework arrive sur le marché ou si `Poco` change sa gestion des threads, de ne changer que le corps de ces méthodes pour qu'elles fassent la même chose sans toucher au code qui les utilise. Un



thread est lancé au premier démarrage de l'application dont le traitement sera celui du serveur de

10. **S**imple **A**PI for **X**ML est originellement une API d'analyse de fichier XML en Java. `Poco` l'a réadaptée en C++.

commande. Celui-ci ne s'arrête qu'à l'arrêt complet de l'application. Ainsi, un `reset` ou un `stop` ne l'affecteront pas et les commandes seront donc encore reçues et traitées correctement.
Ensuite, après une analyse de la configuration, un thread est lancé pour effectué le traitement associé à la communication désignée dans le XML de configuration. Celui-ci sera affecté par les fonction de `reset` ou de `stop` et relancé après un `start`.

3.3.4 Les réponses rapides

La suppression des boucles évoquées dans la figure 3.5 permettent maintenant de ne plus avoir de boucles qui chargent le processeur indéfiniment. Les événements de `Poco` gèrent maintenant les interruptions causées par les commandes d'arrêt ainsi que l'état de terminaison des threads de communication. Ainsi, une interruption suivra le schema suivant :
→ Réception de la commande
→ Déclenchement d'un événement pour signaler l'interruption
→ Demande d'arrêt des threads
→ Déclenchement d'un événement lors de la terminaison du(des) thread(s) de connexion avec le(s) PABX
→ Arrêt ou redémarrage selon l'interruption traitée

De plus, par la suppression des `sleep();`, les temps de réponse et de traitement de ces interruptions est immédiat. Avec l'ancien système d'attente, il arrivait d'attendre presque une seconde entre l'interruption et son traitement. Ce qui pouvait engendrer un transfert de près de 800kio non désirés lors d'une connexion client/serveur TCP dans notre configuration de test en réseau local. Sachant que la configuration de rotation des fichiers de tickets reçus était d'archiver tous les 1024kio, la différence n'était pas négligeable.

3.3.5 La liaison FTP

La difficulté du client FTP résidait dans le fait que tous les scénario devaient être gérés dans le même algorithme. Une fois l'algorithme établi, une autre problématique d'optimisation est apparue.

Tout d'abord, pour connaître le scénario souhaité par l'utilisateur, nous avons ajouté au fichier de configuration plusieurs champs de données qui n'apparaissent pas pour les autres types de connexion. Ces balises sont :

`remote_working_dir` Répertoire de travail sur le serveur distant.
`remote_rotation_dir` Répertoire de rotation des fichiers sur le serveur.
`local_retrieved` Répertoire local de mémorisation des fichiers déjà récupérés.
`del_remote_files` Suppression des fichiers distants une fois récupérés sur notre machine.
`sched` Mode d'ordonnancement de l'exécution du scénario. Prend deux valeurs `each_day` ou `frequency`.
`each_day` L'heure à laquelle le scenario est effectué chaque jour, une fois par jour.
`frequency` Fréquence personnalisée, en seconde, d'exécution du scenario. La première exécution sera au lancement de l'application.

Le client n'est pas obligé de saisir tous les paramètres – certains étant tout de même obligatoires. Les paramètres non saisis prendront des valeur par défaut définies 'en dur' dans le code du programme. C'est d'ailleurs en fonction des paramètres saisis et non-saisis que l'ont va établir le scénario souhaité par l'utilisateur selon l'algorithme suivant :

```
// fonction principale du traitement
listing de tous les fichiers distants;
pour chaque fichier de la liste faire
  appliquer scénario(élément courant de la liste);
```

```
fin pour chaque;
si on ne doit pas supprimer les fichiers sur le serveur après récupération
  synchronisation liste locale --> liste distante:
fin si;
//

// fonction d'application du scénario
si le répertoire local de mémorisation des fichiers déjà récupérés a été saisi alors
    si le fichier a déjà été récupéré alors
        on ne traite pas ce fichier;
    fin si;
fin si;

récupération du fichier;

si le répertoire local de mémorisation des fichiers déjà récupérés a été saisi alors
    mémorisé le fichier fraîchement récupéré.
fin si;
si le répertoire local de mémorisation des fichiers déjà récupérés a été saisi alors
    rotation distante du fichier;
fin si;

si la suppression des fichiers sur le serveur est activée alors
    suppression du fichier distante;
fin si;
//
```

Cet algorithme n'est pas affiné et les détails des fonctions membres ne sont pas donnés ici.

Le problème d'optimisation rencontré était lorsque sur le serveur distant il y avait beaucoup de fichiers (environ 2000 fichiers dans plusieurs sous-répertoires). En effet, l'algorithme manipulait des listes de noms de fichiers et de dossiers distants et locaux, or pour effectuer la synchronisation, ces liste étaient parcourues plusieurs fois.
De plus, l'algorithme de parcours récursif lors du listing des fichiers du serveur récupérait certains noms de fichiers en double ou en triple. Après analyse du contenu de ces liste, il est ressorti que l'une des liste était triée et à valeur unique, alors que l'autre n'était pas triée et comportait beaucoup de doublons. Un tri et une suppression des doublons dans la liste des fichiers distant a permis de gagner environ 10 secondes sur la synchronisation des 2000 fichiers. Pour effectuer le tri et la suppression des doublons, nous avons utilisé des algorithmes (`std::sort` et `std::unique`) de la STL, connus pour être très rapides, très fiables et très propres.

Ajouté à cela, il a fallu mettre en place une gestion d'erreurs robuste pour que notre outil ne se retrouve pas dans un état incohérent et ne sache pas en sortir. Ainsi, s'il y a un problème quelconque durant la connexion, la tâche courante est immédiatement stoppée, tous les fichiers temporaires non complets sont supprimés et on relance une connexion vers le serveur jusqu'à ce que celui-ci réponde. L'erreur en question est enregistrée dans le fichier de sauvegarde des erreurs (définit dans le fichier XML de configuration).
A la nouvelle connexion, on execute à nouveau le traitement en entier pour récupérer les éventuels fichiers qui se seraient ajoutés au répertoire(s) distant(s) pendant que nous n'étions pas connectés et bien sûr pour terminer notre récupération initiale.
Cette gestion d'erreurs assure donc toujours un état cohérent et permet à notre logiciel de ne

jamais être coincé, par exemple dans une récupération de fichier qu'il ne terminera jamais. Un mal nécessaire vu que notre principale préoccupation est l'intégrité des données reçues.

3.3.6 Le futur du projet

Une fois opérationnel et sûr, ce logiciel sera intégré dans un boîtier, une sonde d'acquisition que sera reliée au PABX et sera donc chargée de récupérer les tickets générés par ce-dernier. Il pourra également fonctionné sur un ordinateur en tant que service Windows ou en tâche de fond sous UNIX.
Il sera piloté par une interface Web comme c'était le cas de l'ancien outil mais avec une nouvelle gestion de la configuration et des commandes supplémentaires.

M. Denis Mallet m'ayant proposer de rester pour le mois de Juillet, nous allons perfectionner l'outil en y ajoutant une fonction supplémentaire afin d'exploiter au mieux l'aspect multi-tâches des threads. L'objectif de cette modification est la possibilité pour un seul outil de récupérer les données envoyés depuis plusieurs PABX simultanément. Ce qui ajoute donc des contraintes au niveau de la configuration des différentes connexions avec les PABX qui peuvent être différentes ainsi qu'au niveau de l'archivage des données reçus en fonction du PABX d'émission. Tout ceci en gardant un fonctionnement asynchrone au niveau des commandes et de l'exécution des tâches.

Chapitre 4

Conclusion

Pour faire un bilan des ces 10 semaines de stages passées dans l'entreprise Memobox/2G Technologies, j'aborderais 2 points spécifiques.

Le premier point sera mon expérience professionnelle. En effet, ce stage a été ma 1ère expérience professionnelle dans le domaine de l'informatique. Cela m'as permis de mieux cerner la problématique d'analyste programmeur informatique et les réelles actions qu'il mène au sein d'un service de recherche et développement. De plus, j'ai été confronté à des problèmes différents que ceux rencontrés durant ma formation. J'ai dû faire face à de nouvelles problématiques, me familiariser avec de nouveaux outils et apprendre à travailler autrement tout en fournissant un résultat de qualité, qui pourra être mis en production et ne restera pas un exercice corrigé. J'apprécie l'idée que le travail que j'ai réalisé puisse être proposé au client ou amélioré encore par d'autres développeurs, cela donne une autre dimension à la mission effectuée durant ce stage et aide à sortir du contexte des études.

Cela m'a également permis de me faire une idée sur le rythme de travail des développeurs et de ce que constitue réellement une journée de travail en tant qu'analyste programmeur. Je craignais en arrivant en stage de rester devant un ordinateur toute la journée à écrire du code et d'être rapidement lassé de ne pas me penché sur des problèmes plus théoriques. Or, la réflexion est toujours présente dans le travail de l'analyste développeur car on rencontre toujours des problèmes plus ou moins faciles à résoudre. Certains demande même plusieurs heures de réflexions avant de trouver un algorithme efficace et qui résout correctement le problème rencontré. Ainsi, chaque solution apporte de la satisfaction et donne envie d'aller au bout du projet. On sent que petit à petit le projet avance et on voit les nouvelles fonctionnalités se mettre en place et apporter des solutions à des problèmes plus complexes que l'on avait subdiviser pour parvenir à la solution. C'est gratifiant de sentir que l'on apporte notre pierre à l'édifice et que l'on a réussi à faire quelque chose de reconnu comme fonctionnel.

Ce qui m'amène au deuxième point, les acquis technique. Durant ce stage, j'ai appris énormément de choses sur les communications réseaux que l'on ne voit pas en détail à l'IUT, voire même pas du tout. Ainsi, j'ai appris les subtilités de la connexion en liaison série qui tend à disparaître mais qui est encore utilisée dans le domaine de l'informatique industrielle et de l'électronique mais aussi dans le monde des télécommunications. Également, j'ai approfondi mes connaissances en C++. En arrivant, je me suis plongé dans un code que je n'avais pas l'habitude de voir et qui utilisait des aspects de ce langage que je ne maîtrisait pas ou peu. J'ai apporté les connaissance en conception objet que j'ai apprise à l'IUT pour rendre le projet plus structuré, plus robuste et plus facilement maintenable. Cependant, afin de modifier la structure tout en gardant un outil fonctionnel, il a fallu comprendre son fonctionnement détaillé et sa structure initiale. De nombreuses portions de codes manipulait directement des octets en faisant des décalages logiques ou en les

comparant avec des codes ASCII en hexadécimal. Il n'a pas été évident de saisir le fonctionnement de ces parties là du code mais avec une bonne documentation et les précisions de mes collègues et de mon maître de stage sur les protocoles de communication bas niveau, comme la liaison série, j'ai réussi à comprendre le fonctionnement et l'utilité de ces méthodes.

Cette expérience m'a conforté dans mon choix de poursuite d'étude et je souhaite en apprendre d'avantage sur la conception et la réalisation de logiciels complets qui demande certaines contraintes de sécurité et de stabilité. J'espère acquérir au cours de ma formation les capacités et les astuces pour résoudre plus facilement les problèmes auquel j'ai été confronté durant ce stage et, bien évidemment, ceux auquel je ferais face durant ma carrière professionnelle.