

Capgemini Technology Services
Parc Technologique La Pardieu
9 allée Alan Turing 63170 Aubière

Rapport de stage d'élève ingénieur, seconde année
Filière : F2 - Génie Logiciel et Systèmes Informatiques

Maintenance et ajout de fonctionnalités à un logiciel de suivi de qualité et de gestion des stocks

Présenté par : Gaëtan RUBIN

Responsable ISIMA : M. Christian LAFOREST
Responsable de l'entreprise : M. Fabien RIEU

Date de la soutenance : 24 août 2017
Durée du stage : 5 mois



Institut Supérieur d'Informatique, de Modélisation et de leurs Applications

www.isima.fr

Campus universitaire des Cézeaux • 1 rue de la Chebarde
TSA 60125 • CS 60026 • 63178 Aubière CEDEX
Tel (+33/0) 473 405 000 • Fax (+33/0) 473 405 001



UCA
UNIVERSITÉ
Clermont
Auvergne

Remerciements

Je tiens à remercier M. Fabien RIEU pour m'avoir accompagné pendant ce stage et m'avoir aidé sur la partie relation humaine quand j'en avais besoin.

Je tiens également à remercier M. Sébastien GOUBAND pour m'avoir encadré et aidé sur la partie technique du stage ainsi que toute l'équipe du plateau mutualisé pour leur accueil.

Pour finir, je remercie M. Christian LAFOREST pour m'avoir encadré et suivi tout au long de ce stage pour la partie ISIMA.

Table des figures

1.1	Logo de Capgemini depuis 2004	2
1.2	Répartition des collaborateurs de Capgemini à travers le monde	2
1.3	Historique du chiffre d'affaires et du résultat d'exploitation de Capgemini entre 2005 et 2015	3
1.4	Les sept valeurs de Capgemini	3
1.5	Logo de Plastic Omnium	5
1.6	Logo de SQP	5
1.7	Architecture client / serveur de SQP	5
1.8	Produit représenté dans SAP et dans SQP (tbAssociation)	6
1.9	Table de vérité de l'envoi des transactions à SAP	6
1.10	Postes clients dans une usine de Plastic Omnium	7
1.11	SQP Client : Interface de sélection des utilisateurs	8
1.12	SQP Client : Interface de sélection de la version, de la couleur et du type d'une nouvelle pièce	8
1.13	SQP Client : Interface de sélection de l'état et du défaut de la nouvelle carrosserie	9
1.14	Architecture multi-processus de SQP	10
2.1	Logo de Visual Studio	12
2.2	Logo de Microsoft SQL Server	12
2.3	Logo de Team Foundation Server	13
2.4	Logo de TeamForge	13
2.5	Charge de travail des évolutions du lot 1 (en j)	15
2.6	SQP Manager : Interface de correction des erreurs retournées par SAP	17
2.7	SQP Manager : Menu d'administration en mode historique	18
2.8	SQP Manager : Interface de corrections des erreurs SAP en mode historique	18
2.9	SQP Manager : Exemple d'association	19
2.10	SQP Manager : Interface de modification des quatre trios code produit, couleur et essai d'une association	20
2.11	SQP Client : Interface de sélection de la qualité d'une pièce avec le bouton Essai activé	20
2.12	SQP Client : Interface de modification d'un poste client	22
2.13	SQP Manager : Interface de modification d'une association avec Ref_for_Ctrl	22
2.14	SQP OEE : Format du fichier transmis par la chaîne peinture	24
2.15	SQP OEE : Relation entre la table tbOEE et tbCodeArret	25
2.16	SQP OEE : Interface de modification des causes d'arrêt	25
2.17	SQP OEE : Architecture de l'évolution Overall Equipment Effectiveness	26
2.18	Charge de travail des évolutions du lot 2 (en j)	27
2.19	SQP Manager : Interface de modification d'une association après l'ajout de 8 nouveaux champs	28
2.20	SQP Manager : Interface de modification des associations pour la saisie multiple lors de la lecture automatique	29

2.21	SQP Client : Interface de saisie de l'état des pièces permettant de modifier leur quantité pour le sous-mode automatique	30
2.22	SQP Manager : Interface affichant la liste de toutes les associations	30
2.23	SQP Crypto : Interface permettant de générer une licence	31
2.24	SQP Manager : Interface de modification de la configuration d'un SQP Manager	32
2.25	Charge de travail des évolutions du lot 3 (en j)	33
2.26	SQP Manager : Interface listant tous les types avec l'ajout du filtre	33
2.27	Patron de conception <i>Bridge</i> pour la refonte du mode offline de SQP Client	34
2.28	Exemple de fichier XML : tbTransaction.xml	35
2.29	Classes Transaction et TransactionRow	35
2.30	SQP Manager : Interface de modification des transactions	37
2.31	Interface de modification du déclencheur d'une tâche dans le planificateur de Windows	39
2.32	Exemple de déclencheur pour le déploiement automatique de PPCM	40
2.33	Architecture du déploiement automatique de PPCM	40
2.34	Exemple de fichier de paramètres pour le déploiement automatique de PPCM	41
2.35	Exemple de fichier déclencheur pour un serveur SQP	42
2.36	Exemple de fichier de paramètres des scripts pour SQP	42
2.37	Architecture du déploiement automatique de PPCM et SQP	43
2.38	Diagramme de Gantt prévisionnel	44
2.39	Diagramme de Gantt réel	44
3.1	Architecture finale pour SQP Client et Supervision	46

Résumé

Le but de ce stage était de faire de la tierce maintenance applicative (**TMA**) pour l'entreprise Plastic Omnium, leader mondial des pièces et modules de carrosserie. Cette **TMA** portait sur une suite de logiciels nommée SQP fonctionnant en mode client / serveur sur des machines **Windows**. Ces logiciels sont installés dans des usines et permettent d'assurer le suivi de la qualité de la peinture et la gestion des stocks des nouvelles carrosseries sorties des chaînes de production.

SQP était développé en **Visual Basic.NET** mais ma première tâche a été de le convertir en **C#**. De plus, il utilise plusieurs bases de données **Microsoft SQL Server**. Pour ces logiciels, une nouvelle version a été demandée avec plus de vingt nouvelles évolutions ainsi que des corrections de bugs pour l'ancienne. Une problématique de déploiement automatique de cette suite a également dû être prise en compte.

Le développement a été réalisé avec l'environnement de développement **Visual Studio** sur une machine virtuelle **Windows Server 2012**.

Mots-clés : TMA, C#, Visual Basic.NET, Windows, Microsoft SQL Server, Visual Studio, Windows Server 2012

Abstract

The goal of the internship was to make third party maintenance (**TMA**) for Plastic Omnium, the world leader in body parts and modules. This **TMA** involved a software suite called SQP running in client/server mode on **Windows** machines. These softwares are installed in factories and allow the quality and stock tracking of the bodies coming out of the production lines.

SQP was developed in **Visual Basic.NET** but my first task was to convert it to **C #**. In addition, it uses several **Microsoft SQL Server** databases. For these softwares, a new version has been requested with more than twenty new evolutions as well as bug fixes for the old one. A problem of automatic deployment of this suite also had to be taken into account.

The development was carried out under an integrated developpement environment named **Visual Studio** on a **Windows Server 2012** virtual machine.

Keywords : TMA, C#, Visual Basic.NET, Windows, Microsoft SQL Server, Visual Studio, Windows Server 2012

Table des matières

Remerciements	i
Table des figures	iii
Résumé	iv
Abstract	iv
Table des matières	vi
Introduction	1
1 Présentation et contexte du sujet	2
1.1 Présentation de Capgemini	2
1.1.1 Présentation générale	2
1.1.2 Capgemini en France et le site de Clermont-Ferrand	4
1.2 Présentation du Plastic Omnium et de SQP	5
1.2.1 SQP Client	7
1.2.2 SQP Manager	10
1.2.3 Service SQP SAP	10
1.3 Présentation du travail à réaliser	11
2 Réalisation et conception	12
2.1 Outils utilisés	12
2.2 Process de Capgemini	13
2.2.1 Différentes demandes des clients	13
2.3 Évolutions	15
2.3.1 Évolutions du lot 1	15
2.3.2 Évolutions du lot 2	27
2.3.3 Évolutions du lot 3	33
2.4 Correction de bugs	37
2.4.1 Bugs liés à l'instabilité du réseau	37
2.4.2 Problème fréquence d'import des données depuis SAP	38
2.5 Déploiement automatique	39
2.5.1 Déploiement automatique de PPCM	39
2.5.2 Déploiement automatique de SQP	42
2.6 Planification du stage	44
3 Résultats et discussions	45
3.1 SQP version 5.0	45
3.1.1 Lot 1	45
3.1.2 Lot 2	47

3.1.3	Lot 3	47
3.2	Modifications pour la version 4.5	48
3.2.1	Corrections de bugs	48
3.2.2	Changement des dlls de connexion à SAP	48
3.2.3	Résultats	48
3.3	Discussions et perspectives	49
Conclusion		50
Glossaire		v
Webographie		vii

Introduction

Dans le cadre de mon stage de deuxième année à l'ISIMA, j'ai réalisé un stage de 5 mois, du 3 avril au premier septembre 2017 au sein du plateau mutualisé du site d'Aubière de Capgemini Technology Services.

J'ai été affecté au plateau mutualisé qui est un plateau pluritechnologique. Il traite de nombreux sujets de maintenance applicative pour différents clients.

J'ai travaillé sur de la maintenance applicative d'une suite de logiciels nommée SQP pour l'entreprise Plastic Omnium. Cette suite permet le suivi de la qualité de la peinture de leurs produits ainsi que leurs stocks.

Dans une première partie, je présenterai, plus en détail, le contexte de mon stage en présentant Capgemini et brièvement Plastic Omnium et enfin en expliquant le sujet sur lequel j'ai travaillé. Ensuite, dans une seconde partie, j'exposerai le travail effectué tout en expliquant les différents process de Capgemini pour le suivi des demandes des clients jusqu'à la livraison du produit final. Et enfin, je présenterai les résultats de mon travail, les problèmes rencontrés, leur solution ainsi que les améliorations possibles.

Chapitre 1

Présentation et contexte du sujet

1.1 Présentation de Capgemini

1.1.1 Présentation générale



FIGURE 1.1 – Logo de Capgemini depuis 2004

Capgemini est la première entreprise de services du numérique (ESN) en France et la sixième du secteur dans le monde en 2016. Elle a été créée par Serge Kampf le premier octobre 1967 à Grenoble sous le nom de Sogeti. Cette année, il s'agit donc du cinquantième anniversaire de l'entreprise. Au fil des ans, elle s'est développée grâce à de multiples acquisitions dans tous les secteurs d'activités liés à l'informatique. Ses principales activités sont le conseil en management, l'intégration de systèmes et l'infogérance.

Capgemini est présent dans plus de 40 pays et emploie plus de 190 000 collaborateurs à travers le monde. Ils sont majoritairement répartis entre l'Europe et l'Asie.

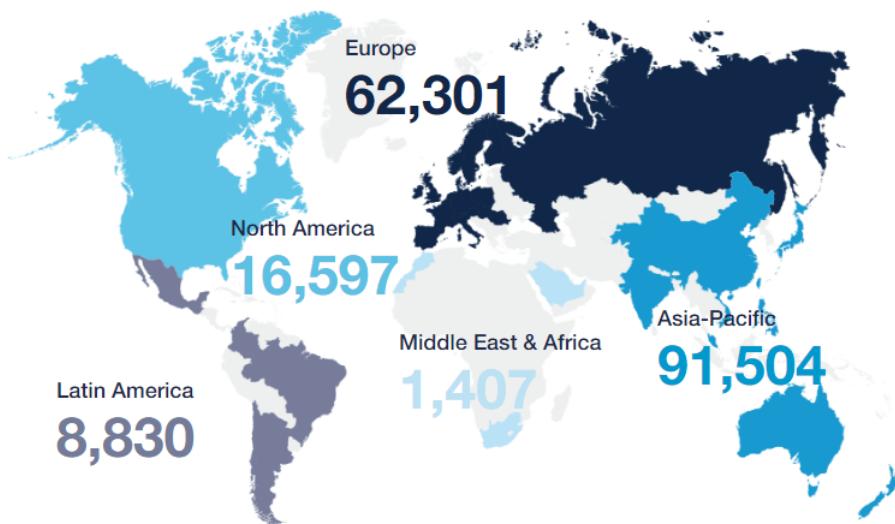


FIGURE 1.2 – Répartition des collaborateurs de Capgemini à travers le monde

En 2016, l'entreprise a réalisé un chiffre d'affaires de plus de 12,5 milliards d'euros et un résultat net de plus de 920 millions d'euros.

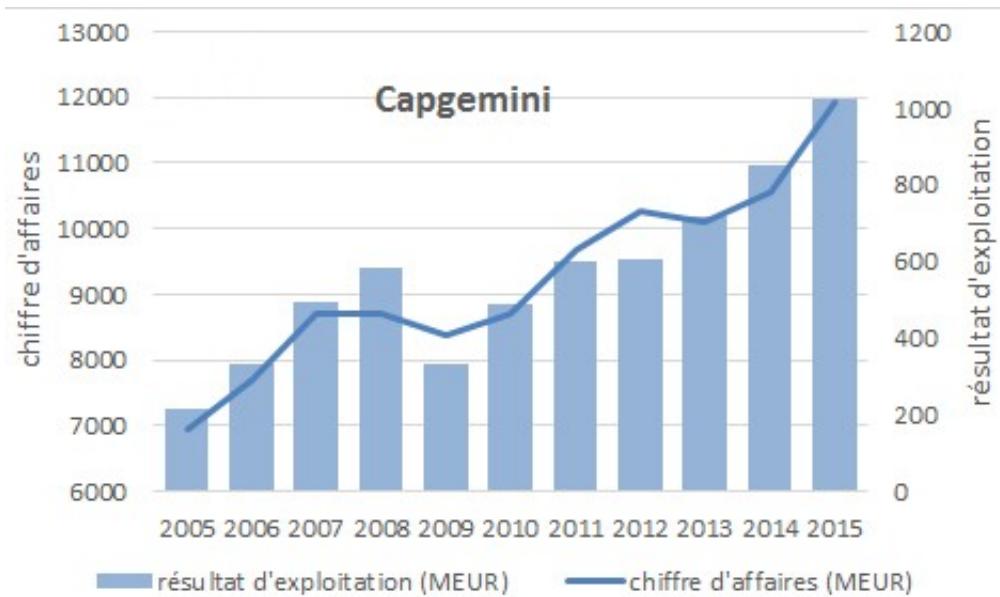


FIGURE 1.3 – Historique du chiffre d'affaires et du résultat d'exploitation de Capgemini entre 2005 et 2015

Capgemini a pour slogan "People matter, results count" (L'homme est vital, le résultat capital) et repose, depuis sa création, sur sept valeurs :

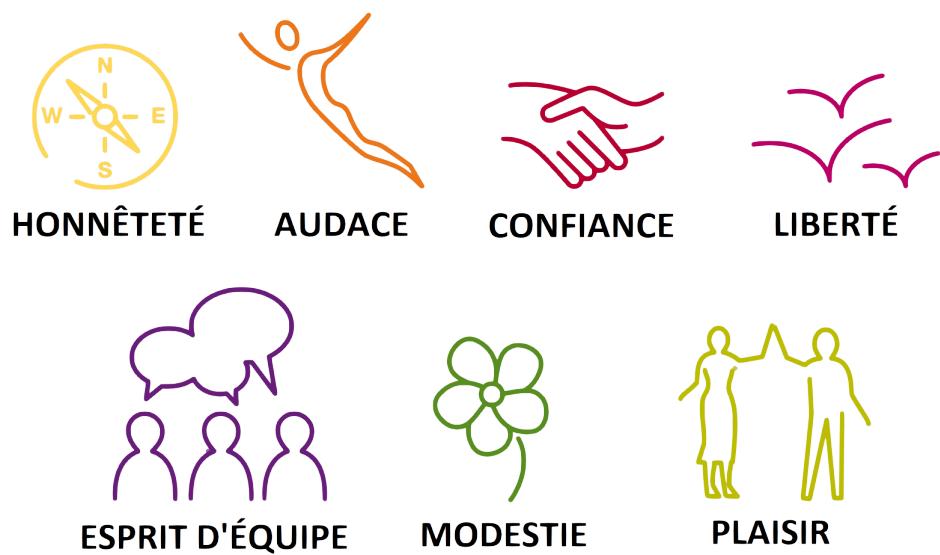


FIGURE 1.4 – Les sept valeurs de Capgemini

1.1.2 Capgemini en France et le site de Clermont-Ferrand

En France, Capgemini est présent dans une vingtaine de villes comme Paris, Bordeaux, Grenoble et Clermont-Ferrand.

Le site Capgemini de Clermont-Ferrand est composé de 5 plateaux différents :

- Le plateau RSI (Régime Social des Indépendants)
- Le plateau Michelin
- Le plateau Hermès
- Le plateau Schneider Electric
- Le plateau mutualisé

Les quatre premiers plateaux travaillent sur plusieurs projets qui sont liés à un client spécifique.

Contrairement aux autres, le plateau mutualisé travaille avec 7 clients différents :

- Michelin AIM (Afrique, Inde, Moyen-Orient) : Développement et support sur des applications de suivi des commandes de pneus jusqu'au paiement dans les pays d'Afrique, d'Inde et du Moyen-Orient.
- PSA Peugeot-Citroën : Maintenance d'un site internet de vente de véhicules d'occasion.
- Union des industries et métiers de la métallurgie (UIMM) : Maintenance d'applications web de gestion de leurs certifications.
- Vallourec : Maintenance d'une application web permettant à leurs clients de suivre leurs commandes ainsi que leurs factures.
- Plastic Omnium Auto Exterior (POAE) : Maintenance de logiciels de gestion et de suivi qualité de leurs produits.
- GammaWeb : Maintenance et ajout de fonctionnalités à un logiciel de gestion de paie pour Capgemini.
- Altitude : Maintenance d'une application Android pour l'enregistrement de la collecte de lait, la géolocalisation des producteurs et l'enregistrement des inventaires et d'une application Java pour la gestion des domaines liés au paiement du lait.

Le plateau est spécialisé dans la tierce maintenance applicative (TMA) qui est la maintenance appliquée à un ou plusieurs logiciels et assurée par un prestataire externe.

Un engagement manager, M. Fabien RIEU, est responsable du plateau, il gère toute la partie management ainsi que les échanges contractuels avec les clients.

Pour chacun de ces clients, un team leader est chargé de mener à bien tous leurs projets. Ils font la liaison entre Capgemini et les entreprises clientes pour les parties plus techniques comme, par exemple, les études sur les évolutions demandées.

Une quinzaine de collaborateurs travaillent sur le plateau et tous peuvent être amenés à travailler sur différents projets de différents clients. Tous les projets n'utilisent pas la même technologie donc les collaborateurs ont besoin d'avoir des connaissances sur les technologies principales. Ces technologies sont le framework .NET, SharePoint et SQL Server de Microsoft ainsi que le Java J2EE.

1.2 Présentation du Plastic Omnium et de SQP

Mon stage se déroule au sein du plateau mutualisé sur un projet lié à Plastic Omnium et plus précisément leur division Auto Exterior (POAE).



FIGURE 1.5 – Logo de Plastic Omnium

Plastic Omnium est un groupe industriel français créé en 1946. Il s'agit du leader mondial des pièces et modules de carrosserie et des systèmes à carburant pour l'automobile. Son chiffre d'affaires s'élevait à environ 7,5 milliards d'euros en 2016 et le groupe emploie plus de 33 000 personnes dans 128 usines réparties dans 31 pays dans le monde.

Mon objectif principal était de faire de la tierce maintenance applicative (TMA) sur une suite de logiciels nommée SQP (Système Qualité Peinture). Il s'agit d'une suite qui permet le suivi de la qualité de la peinture des carrosseries en sortie de la chaîne de production. Cette qualité est définie suivant quatre états : bon, polishé, recirculé et rebuté.



FIGURE 1.6 – Logo de SQP

SQP fonctionne en mode client / serveur et est installée dans les différents sites, usines de Plastic Omnium. Chacun de ces sites dispose d'un serveur qui lui est lié donc il y a autant de serveurs que de sites.

De plus, SQP est fortement liée à SAP qui est un progiciel de gestion intégré (ERP). Un ERP peut être défini comme un système d'information reliant les différentes fonctions d'une entreprise (comptabilité, production, ressources humaines, etc). La tierce maintenance applicative de la partie SAP de SQP est assurée par une équipe Capgemini en Inde.

SQP permet de déclarer de nouvelles transactions sur la qualité et la quantité des nouvelles carrosseries sortant de la production et toutes ces données sont envoyées à SAP afin que Plastic Omnium puisse faire le suivi de leurs stocks.

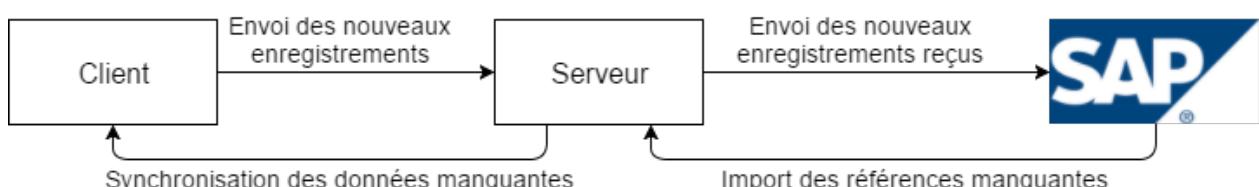


FIGURE 1.7 – Architecture client / serveur de SQP

Dans leur SAP, tous les produits, avec leurs spécificités, sont identifiés par une référence unique ainsi qu'une paire de deux éléments : le code du produit et son code couleur. C'est-à-dire que si un même modèle de carrosserie possède plusieurs couleurs, plusieurs types ou encore plusieurs versions, il existera une référence, un couple code produit et code couleur unique pour chaque trio couleur, type et version.

Dans SQP, les produits sont identifiés directement par ce trio version, type et couleur. Donc afin de lier la référence SAP à ce trio dans SQP, une classe spéciale a été mise en place : la classe *Association*.

SAP	tbAssociation
<u>Référence</u> : String	<u>Type</u> : Type
<u>Code produit</u> : String	<u>Version</u> : Version
<u>Code couleur</u> : String	<u>Couleur</u> : Color
	<u>Code produit</u> : String
	<u>Code couleur</u> : String
	<u>Envoi rebut</u> : Bool
	<u>Envoi bon</u> : Bool

FIGURE 1.8 – Produit représenté dans SAP et dans SQP (tbAssociation)

Elle contient ces cinq informations ainsi que toutes autres informations complémentaires sur cette pièce. Parmi celles-ci, deux autres sont importantes : *Envoi bon* et *Envoi rebut*. Elles permettent de savoir s'il faut envoyer les nouvelles transactions à SAP en fonction de leur qualité et sont donc sous la forme de booléen (variable pouvant prendre les valeurs *Vrai* ou *Faux*). Par exemple, si *Envoi bon* est vrai alors toutes les transactions liées à l'association qui sont dans un état bien seront envoyées. Alors que si, par exemple, *Envoi rebut* est faux alors toutes les transactions liées à l'association qui sont dans un état rebut ne seront pas envoyées à SAP.

	Etat bon	Etat rebut
Envoi bon vrai	X	
Envoi bon faux		
Envoi rebut vrai		X
Envoi rebut faux		

FIGURE 1.9 – Table de vérité de l'envoi des transactions à SAP

Il faut également savoir que SQP ne fonctionne que sur Windows et est découpée en trois composants principaux. Chacun de ces composants possèdent un ou plusieurs fichiers de log afin de permettre à Plastic Omnium et à Capgemini d'identifier les différents problèmes et de les tracer afin de faciliter leur correction. Ils ont tous été développés en Visual Basic.NET (VB.NET) et utilisent des bases de données Microsoft SQL Server. Une licence est également nécessaire afin de pour utiliser SQP. Cette licence contient également les informations de connexion à la base de données serveur.

1.2.1 SQP Client

Le premier des composants se nomme SQP Client et il s'agit un logiciel installé sur les postes clients directement dans les usines. Il permet à un utilisateur de répertorier l'état d'une carrosserie et de l'envoyer au serveur afin que cette nouvelle pièce soit transmise à SAP (en fonction des deux booléens présentés ci-dessus) pour le suivi de la qualité et de la quantité des différentes pièces. Comme évoqué précédemment, ces informations envoyées au serveur s'appellent des transactions. Le client peut également imprimer des étiquettes avec ces informations afin de les coller sur les pièces pour les identifier.

Tous ces postes clients possèdent une base de données locale en cas de perte de connexion avec le serveur afin de stocker toutes les nouvelles informations entrées via le logiciel puis de les renvoyer lorsque la connexion est retrouvée. Le client SQP possède également un fichier de configuration contenant les informations de connexion à cette base de données locale.

Comme Plastic Omnium est une entreprise multinationale présente dans de nombreux pays, SQP est traduit dans 8 langues : le français, l'anglais, le chinois, l'espagnol, le portugais, l'allemand, le polonais et le slovène.



FIGURE 1.10 – Postes clients dans une usine de Plastic Omnium

SQP Client dispose de deux modes de fonctionnement : un manuel qui est le mode standard et un automatique nommé supervision.

1.2.1.1 Mode standard

Le mode standard est également divisé en deux sous-modes :

- Le mode scan d'un code barre pour récupérer directement le trio version, type et couleur
- Le mode de sélection par l'utilisateur de ce trio

Pour commencer pour ces deux sous-modes, il faut sélectionner le ou les utilisateurs qui vont utiliser le logiciel.

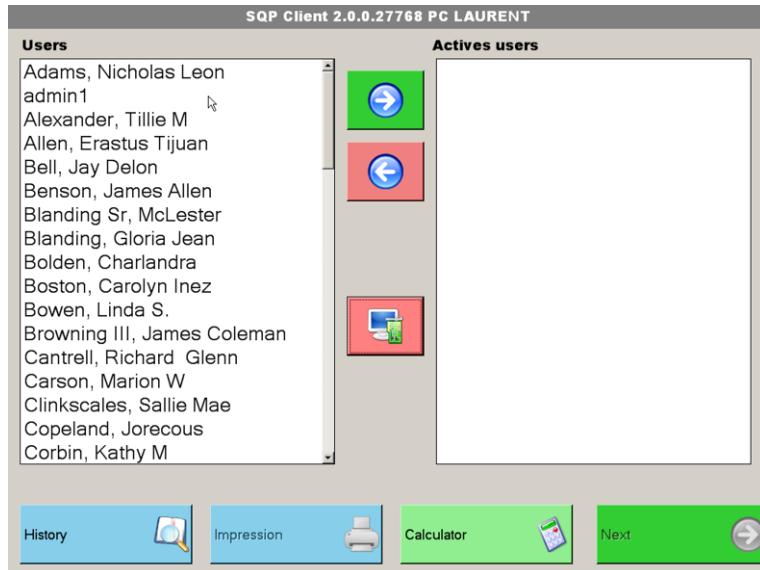


FIGURE 1.11 – SQP Client : Interface de sélection des utilisateurs

Ensuite, soit l'utilisateur scanne un code barre présent sur la pièce. Ce scan permet de récupérer toutes les informations possibles (notamment le type, la version et la couleur) liées à la référence de ce code barre.

Soit l'utilisateur sélectionne manuellement le type, la version et la couleur. Il est également possible de sélectionner, uniquement dans ce sous-mode et si l'option est active, le nombre de nouvelles carrosseries à enregistrer.

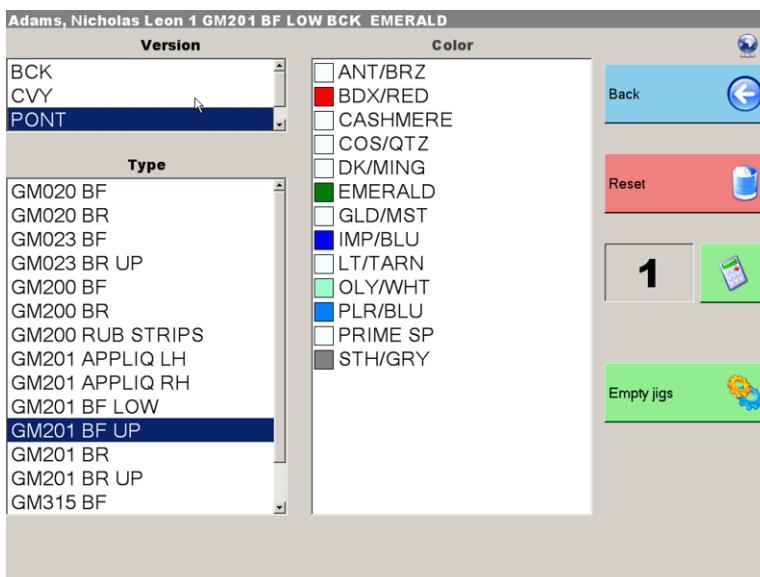


FIGURE 1.12 – SQP Client : Interface de sélection de la version, de la couleur et du type d'une nouvelle pièce

Enfin la dernière étape, commune aux deux sous-modes, est de sélectionner la qualité de la pièce, son défaut (s'il y en a un) ainsi que d'autres informations qui ne sont pas nécessaires à la compréhension du fonctionnement et d'envoyer cette nouvelle déclaration de production au serveur.

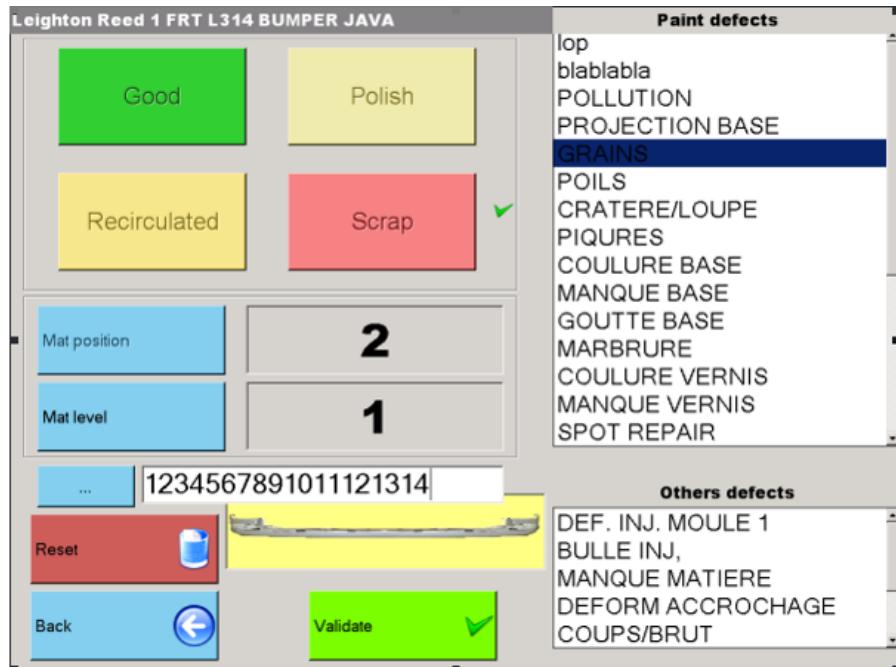


FIGURE 1.13 – SQP Client : Interface de sélection de l'état et du défaut de la nouvelle carrosserie

1.2.1.2 Mode supervision

Le deuxième mode du client est donc automatique, il est également appelé le mode supervision et il est plus complexe que le premier. Contrairement au mode standard, ce mode ne déclare que des nouvelles pièces dans une qualité dite bonne. Il permet de lier le mode supervision de la chaîne peinture directement à SQP sans avoir besoin d'un utilisateur pour renseigner les différentes informations nécessaires. Il est nécessaire de préciser que, contrairement à SQP, la chaîne de peinture identifie un modèle de carrosserie avec toutes ses spécificités, en utilisant les mêmes identifiants que SAP : le code produit et le code couleur.

Pour commencer, SQP Client récupère dans un répertoire des fichiers textes déposés par le mode supervision de la chaîne peinture et crée les transactions correspondant aux valeurs contenues dans ces fichiers. Ces valeurs sont le code produit, le code couleur de la pièce ainsi que la quantité. Ensuite, le mode supervision retrouve, grâce au code produit et couleur, toutes les informations complémentaires comme la version, le type et la couleur de cette pièce puis crée une nouvelle transaction avec ces informations et enfin l'envoie au serveur. Une fois qu'elle est envoyée le fichier est supprimé.

1.2.2 SQP Manager

Le second composant est un logiciel installé sur le serveur et s'appelle SQP Manager. Il permet d'ajouter ou de supprimer des postes clients ainsi que de modifier leur configuration. Il peut également gérer la liste des utilisateurs ainsi que toutes les données stockées dans la base de données comme par exemple la liste de tous les carrosseries ou encore de toutes leurs couleurs. Une autre de ses fonctions est de pouvoir créer, modifier ou supprimer les associations présentées dans la partie précédente. SQP Manager permet également de gérer les liens et de corriger les transactions retournées en erreur lors de l'envoi des données à SAP.

1.2.3 Service SQP SAP

Enfin, le dernier composant principal de la suite SQP est un service Windows nommé SQP SAP Service. Il établit la connexion entre SQP et SAP, avec les informations renseignées dans SQP Manager et envoie les données reçues par les clients à SAP. Il permet également de mettre à jour la base de données serveur avec les données stockées dans SAP comme par exemple la liste de toutes les paires codes produits et codes couleurs SAP. L'envoi des nouvelles données de SQP à SAP se fait toutes les minutes alors que la mise à jour des informations depuis SAP ne se fait qu'une seule fois par jour.

Ci-dessous une image résumant l'architecture multi-processus de la suite SQP :

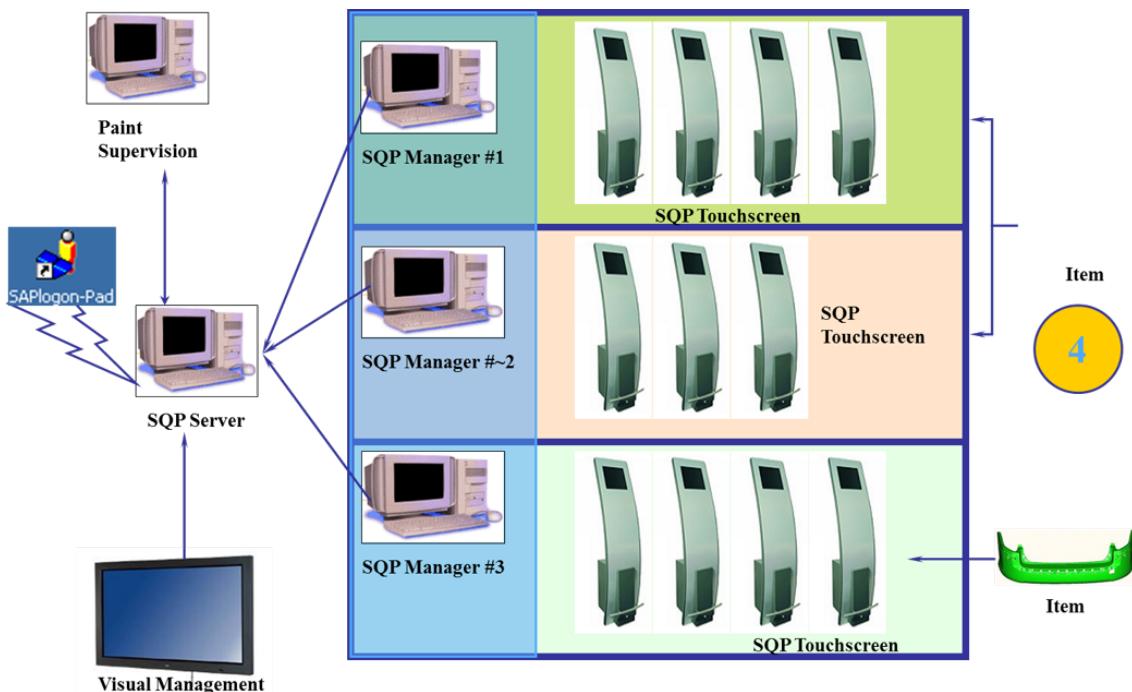


FIGURE 1.14 – Architecture multi-processus de SQP

1.3 Présentation du travail à réaliser

Capgemini a reçu une demande de Plastic Omnium afin d'ajouter de nouvelles fonctionnalités à SQP, passant de la version 4.5 à la version 5.0. Ces fonctionnalités étaient découpées en trois lots et mon travail a été de développer et d'intégrer les évolutions comprises dans ces trois lots. Évolutions que je détaillerai tout au long de ce rapport.

Avant de commencer le développement de ces évolutions, ma première tâche a été de convertir le code de SQP du VB.NET au C#.NET parce qu'il s'agit d'une technologie maîtrisée par un plus grand nombre de collaborateurs de Capgemini ce qui facilitera la maintenance applicative de cette suite dans le futur.

Au cours du stage, des demandes de corrections de bugs pour la version 4.5 sont venues se rajouter à ces évolutions ainsi qu'une problématique de déploiement automatique de cette suite. En effet pour installer SQP sur les différents serveurs et postes clients, les responsables informatiques de Plastic Omnium devaient se rendre d'usine en usine et de poste en poste afin d'exécuter son installateur ce qui engendrait des frais importants pour l'entreprise. Cette problématique avait déjà été posée pour un autre logiciel de Plastic Omnium, PPCM, qui est également en TMA à Capgemini. Donc pour créer la solution répondant à ce problème, j'ai dû adapter la solution existante au mode d'installation de SQP.

Chapitre 2

Réalisation et conception

Avant d'expliquer plus en détails les différentes évolutions et le travail effectué, je vais commencer par présenter les outils que j'ai utilisés tout au long de mon stage, ensuite les différents process de Capgemini auxquels j'ai dû me familiariser. Et pour finir, je parlerai de la planification de mon stage et des contraintes de temps auxquelles j'ai dû m'adapter.

2.1 Outils utilisés

Afin de développer et intégrer les évolutions demandées et comme SQP ne fonctionne que sur Windows, j'ai travaillé sur des machines virtuelles internes à Capgemini avec pour système d'exploitation Windows Server 2012. Ce système d'exploitation correspond à Windows 8.1 pour les ordinateurs classiques.



FIGURE 2.1 – Logo de Visual Studio

Les versions précédentes de SQP ont été développées avec l'environnement de développement (IDE) Visual Studio 2012 donc j'ai continué avec ce même IDE. Il permet notamment de créer ou modifier facilement des services Windows (comme le service SQP SAP).



FIGURE 2.2 – Logo de Microsoft SQL Server

SQP utilise une base de données Microsoft SQL Server que ce soit pour celle du serveur ou celle locale aux postes clients. Afin d'effectuer toutes les modifications nécessaires pour la nouvelle version 5.0, j'ai donc utilisé Microsoft SQL Server 2012 Management.



FIGURE 2.3 – Logo de Team Foundation Server

Afin de gérer les différentes versions et l'avancement des évolutions, Plastic Omnium a mis à disposition de Capgemini un outil nommé *Team Foundation Server* (TFS). Il s'agit d'une forge logicielle éditée par Microsoft qui permet notamment la gestion des sources avec un historique des modifications de ces sources. Il permet également de fusionner facilement et rapidement les différentes modifications apportées par différents développeurs sur le même projet. TFS est intégré par défaut dans Visual Studio et permet donc très facilement de passer d'une version à une autre d'un projet ou encore de récupérer les modifications effectuées par une autre personne.

2.2 Process de Capgemini

Capgemini étant une entreprise de services, tous ses process sont fortement liés à ses clients et notamment à leurs demandes. Ces process vont du contrat avec l'entreprise cliente jusqu'à la livraison des solutions répondant à leurs différentes sollicitations.

2.2.1 Différentes demandes des clients

Les demandes des clients sont principalement des demandes de nouvelles évolutions ou des demandes de corrections de bugs.

Toutes ces requêtes sont créées sous la forme de tickets dans un outil nommé TeamForge.



FIGURE 2.4 – Logo de TeamForge

Il s'agit d'un outil de communication entre Capgemini et ses clients qui permet la création et le suivi des tickets suite à ces requêtes. TeamForge permet également de stocker et de versionner tous les différents documents liés à ces tickets (études, mails) ou aux projets (contrats).

Ces tickets sont découpés en trois types :

- Les évolutions : toutes les demandes d'ajout de fonctionnalités pour un logiciel existant. Ces évolutions sont facturées au client.
- Les retours : toutes les demandes de correction des bugs lors de la recette du client suite à une livraison. Ces corrections ne sont pas facturées au client.
- Les incidents : toutes les demandes de correction de bugs qui n'ont pas été détectés lors de la recette client. Ces corrections sont facturées au client.

De la création à la résolution du ticket, les tickets passent par plusieurs phases :

- La phase d'analyse du problème par Capgemini. Si le ticket est conséquent et comporte plusieurs évolutions ou corrections d'incidents, une étude d'impact macroscopique est nécessaire. Cette étude comporte la liste de toutes les requêtes avec pour chacune :
 - L'analyse de l'existant
 - La description détaillée du besoin
 - Ses impacts fonctionnels dans l'application cible
 - Sa complexité
 - Sa charge de travail en jour ou unité d'oeuvre
 - Sa priorité s'il y a plusieurs lots
- La phase de résolution du problème c'est-à-dire le développement et les tests de la solution.
- La phase de recette du client après livraison de la solution. Celui-ci teste sur un serveur de pré-production si la correction ou l'évolution apportée est conforme à ses attentes et s'il n'y a pas de régression. S'il trouve un problème, il doit créer un nouveau ticket de gestion des retours.
- La phase de mise en production une fois que la solution apportée répond au besoin.

Lorsque que Capgemini a reçu de Plastic Omnium, la demande d'ajout de nouvelles fonctionnalités pour SQP, un seul ticket et une seule étude d'impact macroscopique ont été créés regroupant ces 25 nouvelles évolutions. L'étude a été faite avant mon arrivée donc je n'ai eu qu'à me focaliser sur leur développement et leur implémentation.

2.3 Évolutions

Avant de commencer à comprendre en détail le fonctionnement de SQP afin de développer les évolutions demandées, j'ai d'abord dû me familiariser avec les process Capgemini expliqués précédemment. Ensuite, j'ai étudié la documentation fournie par Plastic Omnium afin de comprendre le fonctionnement global de la suite SQP ainsi que celui détaillé de chaque logiciel composant cette suite. Cette documentation n'étant pas à jour sur certains points, j'ai dû tester et regarder directement dans le code et dans la base de données afin de prendre connaissance des différentes modifications apportées.

Comme expliqué précédemment les évolutions pour la 5.0 sont découpées en trois lots. Avant de développer, j'ai d'abord examiné plus en détail chaque évolution du premier lot afin de repérer, grâce à l'étude d'impact macroscopique, celles qui étaient à implémenter avant d'autres afin de ne pas modifier plusieurs fois la même chose. Parmi ces évolutions à faire en priorité dans le lot 1, j'ai également sélectionné celles qui semblaient les plus faciles à développer afin d'augmenter plus facilement ma compréhension de SQP. Mais j'ai quand même commencé par la tâche la plus longue de toute c'est-à-dire la migration du code du VB.NET au C# parce que je suis plus à l'aise pour développer dans ce dernier.

2.3.1 Évolutions du lot 1

Sur les 25 évolutions comprises dans la nouvelle version 5.0 de SQP, 11 sont intégrées dans le lot 1. Parmi ces 11 évolutions d'après l'étude, 4 avaient une charge de travail inférieure ou égale à 1 jour, 4 avaient une charge de travail comprise entre 2 et 5 jours et 3 avaient une charge de travail supérieure à 5 jours.

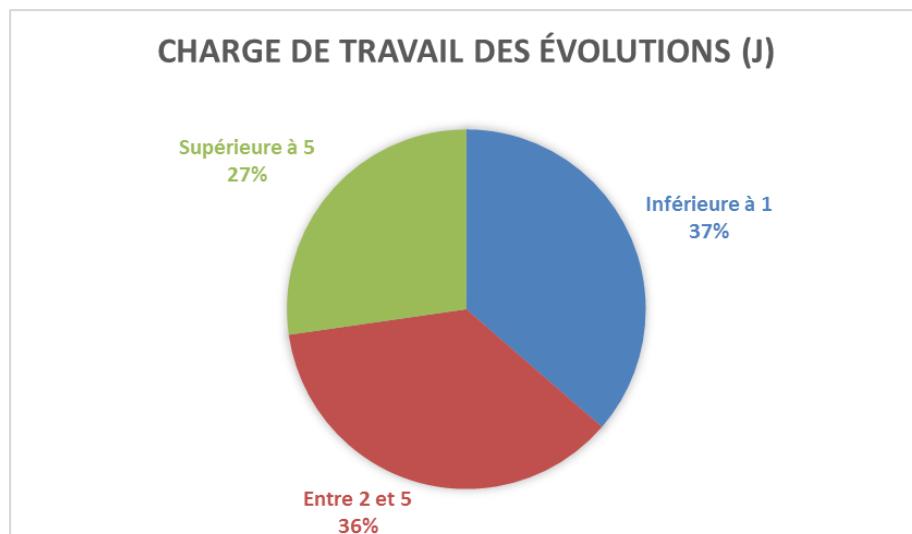


FIGURE 2.5 – Charge de travail des évolutions du lot 1 (en j)

Pour ce lot, j'ai été seul à faire le développement donc j'expliquerai brièvement toutes les évolutions et j'irai plus dans les détails pour les plus importantes et intéressantes.

2.3.1.1 Migration VB.NET / C#.NET

Comme présenté brièvement dans l'introduction de cette partie, la première évolution sur laquelle j'ai travaillé a été la migration du code de SQP du langage de programmation Visual

Basic.NET vers le C#.NET. Sa charge de travail a été de 8 jours. J'ai commencé par celle-ci afin de pouvoir développer et intégrer les autres évolutions plus facilement et rapidement parce que je maîtrise mieux le C# que le Visual Basic. Une autre raison est le fait que si j'avais développé les nouvelles évolutions dans le langage d'origine, il aurait fallu les convertir également et il y aurait eu potentiellement plus de sources de problèmes lors de la conversion.

Ces deux langages de programmation faisant partie du même framework (.NET) développé par Microsoft, la conversion de l'un à l'autre peut être faite automatiquement par des logiciels spécialisés. Un framework est ensemble cohérent de composants logiciels qui servent à créer les fondations ainsi que les grandes lignes d'une partie de l'architecture d'un logiciel. Par exemple pour VB.NET et le C#.NET, le fonctionnement de la très grande majorité de leurs méthodes natives sont communes. Le changement le plus important entre ces deux langages est la notation ou le nom de ces méthodes. Par exemple, en Visual Basic.NET pour convertir une variable du format chaîne de caractères à celui de booléen, il faut appeler la méthode *CBool()* alors qu'en C# cette même méthode se nomme *Convert.ToBoolean()*.

Le logiciel de conversion préconisé par Microsoft s'appelle *VBConversions* et permet de convertir n'importe quel projet développé en Visual Basic.NET en projet C#. Par contre, il ne fait pas la conversion du C# vers le Visual Basic.

En utilisant ce logiciel, la migration du code en elle-même a été très rapide car en effet le logiciel l'a converti quasiment instantanément. La tâche la plus longue a été de tester la suite SQP après cette migration afin de détecter et corriger les éventuelles régressions.

La seule régression trouvée a été au niveau de la méthode de conversion des entiers ou chaîne de caractères en booléen. En effet, dans la base de données la majorité des variables booléennes sont stockées sous forme de chaîne de caractère (et non pas en entier ou booléen directement) avec pour valeur '0' et '1' pour respectivement *Faux* et *Vrai*. En VB.NET, la méthode de conversion *CBool()* permet de faire cette liaison entre le caractère '0' ou '1' et sa valeur booléenne. Mais en C#, la méthode correspondante *Convert.ToBoolean()* ne le permet pas.

Il a donc fallu corriger ce problème à chaque fois qu'il apparaissait dans le code. Pour cela, j'ai dû tester directement si le caractère retourné par la base de données était égal au caractère '1'. Par exemple, si le caractère retourné a pour valeur '0', il est donc différent de '1' et le test retournera bien la valeur *Faux*.

2.3.1.2 Évolutions ne concernant que la base de données

La plupart des évolutions ayant une charge de travail inférieure à un jour (3 sur 4) sont liées à des modifications dans la base de données. Ces modifications sont :

- La correction de certaines traductions.
- L'ajout de nouvelles tables et procédures stockées fournies par le client permettant de faire des statistiques sur l'état des nouvelles pièces (par exemple, le pourcentage de nouvelles carrosseries dans un état rebut).
- La suppression en cascade de certaines données.

Toutes ces modifications ont été rajoutées dans un seul script SQL qui est exécuté lors de la mise à jour de la version 4.5 vers la version 5.0.

2.3.1.3 Expiration automatique du mode administration des erreurs SAP

La dernière de ces petites évolutions est liée à l'interface de correction des transactions retournées en erreur par SAP dans SQP Manager.

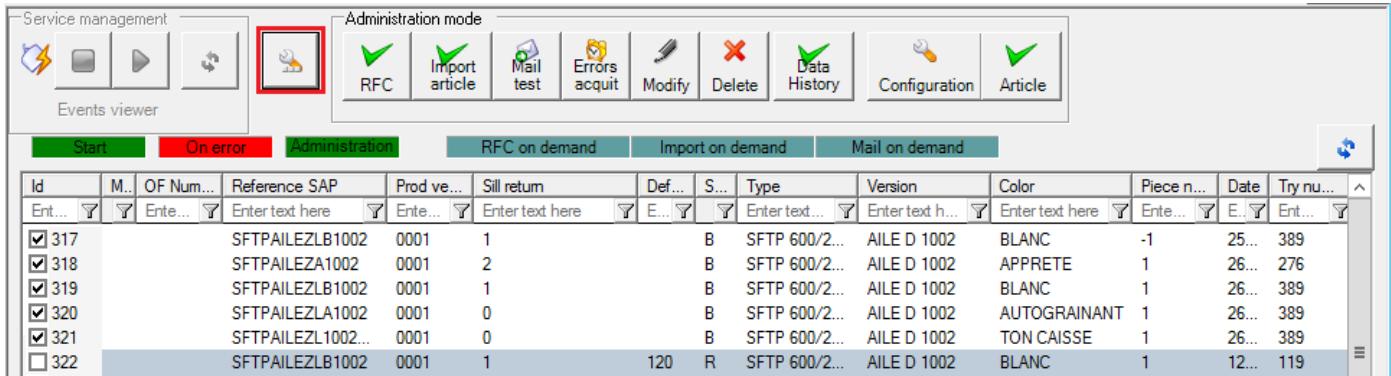


FIGURE 2.6 – SQP Manager : Interface de correction des erreurs retournées par SAP

Afin de modifier ou supprimer ces transactions, il faut activer un mode administration en appuyant sur le bouton encadré en rouge dans l'image ci-dessus. Pour le désactiver, il suffit de rappuyer sur ce même bouton. Lorsque ce mode est activé, cela bloque l'envoi des données de SQP vers SAP. Donc si un utilisateur laisse l'application lancée, avec ce mode activé, pour une longue durée cela pose problème pour l'intégrité des informations dans SAP. La demande du client est donc de quitter automatiquement ce mode administration au bout de 30 minutes d'inactivité afin de conserver cette intégrité.

Afin de répondre à ce besoin, j'ai d'abord dû récupérer le temps d'inactivité de l'utilisateur. Pour cela, j'ai utilisé la méthode native de Windows qui est présente dans une bibliothèque logicielle (DLL : Dynamic Link Library) présente par défaut dans ce système d'exploitation. Ensuite, j'ai rajouté un minuteur (timer) qui toutes minutes regarde si le mode administration est activé et si le temps d'inactivité est supérieure ou égale à 30 minutes. Si c'est le cas, je simule l'appui sur le bouton d'activation / désactivation de ce mode afin de le désactiver et de supprimer le blocage de l'envoi des données à SAP.

2.3.1.4 Historisation des actions liées aux erreurs retournées par SAP

Cette évolution concerne la même interface que celle expliquée précédemment mais il n'y a pas d'interférence au niveau du code c'est-à-dire qu'elles peuvent être implémentées dans n'importe quel ordre. Sa charge de travail était de trois jours.

Comme expliqué dans le précédent paragraphe, il est possible pour l'utilisateur de modifier ou de supprimer une de ces transactions via le menu d'administration. Ces actions n'étaient pas historisées et la demande du client a été de rajouter cette fonctionnalité afin de garder une trace de toutes ces actions en cas de problème potentiel suite à celles-ci. Lorsqu'une action est effectuée par l'utilisateur, il faut historiser la transaction avant et après cette action. Le client voulait également pouvoir n'afficher que les actions s'étant déroulées dans un intervalle de temps choisi par l'utilisateur. Cet intervalle doit également être par défaut la semaine en cours.

Pour cela, j'ai commencé par incorporer un bouton *Data History* dans le menu d'administration. Lorsque l'utilisateur clique sur ce bouton, l'interface passe en mode historique. Un

deuxième appui sur ce même bouton repasse l'interface en mode normal. J'ai également ajouté deux sélecteurs de date afin que l'utilisateur puisse sélectionner n'importe quel intervalle. Ils ne sont affichés que lorsque le mode historique est activé. De plus, lorsque ce mode est activé, les boutons *Modify* et *Delete* sont désactivés afin de ne pas modifier ou supprimer un élément de l'historique.



FIGURE 2.7 – SQP Manager : Menu d'administration en mode historique

Cet historique reprend de façon chronologique la déclaration originale suivie de la modification ou de la suppression. Afin de stocker ces modifications et suppressions, j'ai créé une nouvelle table, *tbHistoTrames*, dans la base de données quasiment identique à la table contenant toutes les transactions retournées en erreur, *tbTrames*. Cette nouvelle table possède toutes les colonnes de *tbTrames* mais avec une nouvelle colonne, *Type_Histo*. Cette colonne peut avoir 3 valeurs différentes afin d'identifier le type d'enregistrement :

- Original : s'il s'agit de la transaction avant modification ou suppression
- Update : s'il s'agit de la transaction après modification
- Delete : s'il s'agit de la transaction après suppression

Afin d'afficher cet historique et comme les transactions historisées sont quasiment identiques aux originales (plus cette nouvelle colonne *Type_Histo*) j'ai utilisé le composant / tableau déjà existant. Les données affichées par ce composant changent lorsque l'on passe d'un mode à l'autre (historique vers normal ou normal vers historique).

Id	Type Histo	M...	OF Num...	Reference SAP	Prod ve...	Sill return	Def...	S...	Type	Version	Color	Piece n...	Date		
315	ORIGINAL	3...	333	00000000055001...	0001	0			B	SFTP 600/2...	AILE D 1002	TON CAISSE	24	13...	
315	UPDATE	3...	333	00000000055001...	0001	0			B	SFTP 600/2...	AILE D 1002	TON CAISSE	24	13...	
315	ORIGINAL	3...	333	00000000055001...	0001	0			B	SFTP 600/2...	AILE D 1002	TON CAISSE	24	13...	
315	DELETE	3...	333	00000000055001...	0001	0			B	SFTP 600/2...	AILE D 1002	TON CAISSE	24	13...	
316	ORIGINAL			SFTPAILEZLB1002	0001	1			110	R	SFTP 600/2...	AILE D 1002	BLANC	1	25...
316	UPDATE	t...		SFTPAILEZLB1002	0001	1			110	R	SFTP 600/2...	AILE D 1002	BLANC	1	25...
316	ORIGINAL	t...		SFTPAILEZLB1002	0001	1			110	R	SFTP 600/2...	AILE D 1002	BLANC	1	25...
316	DELETE	t...		SFTPAILEZLB1002	0001	1			110	R	SFTP 600/2...	AILE D 1002	BLANC	1	25...

FIGURE 2.8 – SQP Manager : Interface de corrections des erreurs SAP en mode historique

2.3.1.5 Déclaration du type Essai

La demande du client pour cette évolution était de rajouter un nouveau type de déclaration : le type *Essai*. La charge de travail de cette demande était de quatre jours. Cette évolution concerne SQP Client en mode manuel ou supervision, SQP Manager et le service SQP SAP. Ce nouveau type permettra, grâce à d'autres informations que je détaillerai dans cette partie, de pouvoir transmettre ou non les nouvelles transactions à SAP avec de nouvelles règles sans tenir compte des deux booléens existants : *Envoi bon* et *Envoi rebut*. Ce changement permet à Plastic Omnium de pouvoir effectuer des tests facilement sans avoir à se soucier de l'intégrité des données dans SAP. Le premier changement demandé est de rajouter un champ booléen *Essai* aux transactions afin de différencier facilement les transactions normales et celles de type essai.

Avant de parler plus en détail du fonctionnement demandé, il est nécessaire d'approfondir le fonctionnement existant du mode supervision de SQP Client. Comme expliqué dans la présentation de SQP, ce mode reçoit des fichiers au format texte dans un répertoire. Ces fichiers contiennent les deux critères qui permettent d'identifier un produit dans SAP afin de pouvoir créer la transaction correspondante qui sera envoyée au serveur puis à SAP. Pour rappel, ces deux critères sont le code du produit et son code couleur. Ces fichiers contiennent également le nombre de déclarations de productions à effectuer et toutes ces informations sont séparées par des points-virgules.

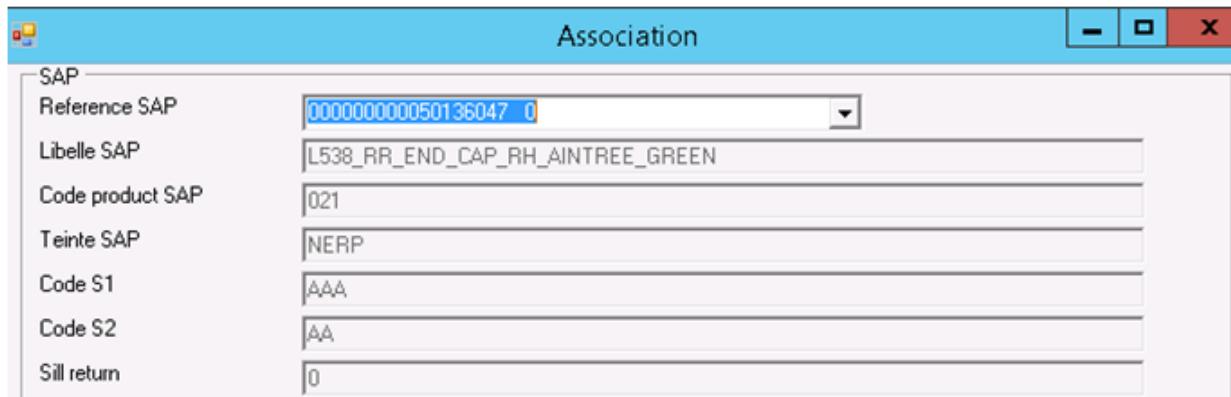


FIGURE 2.9 – SQP Manager : Exemple d’association

Le contenu d'un de ces fichiers peut-être par exemple : 021;NERP;1. Le client SQP associera donc 021/NERP à la référence SAP 50136047 et fera une déclaration de production pour une seule pièce.

Le premier changement demandé intervient au niveau des associations : une même association peut maintenant avoir quatre paires code produit, code couleur avec pour chacune de ces paires, un nouvel élément permettant de savoir s'il s'agit d'un essai ou non : un champ *Essai*. Chacun des couple code produit et code couleur doit rester unique. Afin de pouvoir ajouter, modifier ou supprimer ces quatre trios, dans SQP Manager j'ai rajouté un nouveau bouton dans l'écran de modification d'une association qui ouvre une nouvelle interface permettant de gérer ces quatre trios.

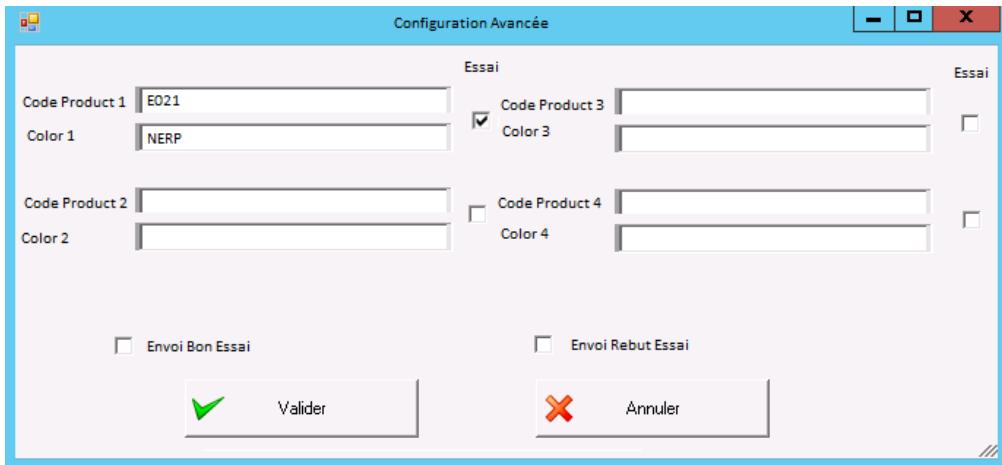


FIGURE 2.10 – SQP Manager : Interface de modification des quatre trios code produit, couleur et essai d'une association

Le premier de ces trios est le même que celui de la page de modification d'une association et ne peut donc pas être supprimé contrairement aux trois autres. Un autre changement qui a été demandé et qui a été intégré dans cette nouvelle interface, est l'ajout de deux nouveaux booléens *Envoi bon essai* et *Envoi rebut essai* qui ont le même rôle que ceux déjà présents : *Envoi bon* et *Envoi rebut*. J'expliquerai dans un second temps la différence entre ces deux paires de booléens.

Le deuxième changement intervient au niveau de SQP Client que ce soit en mode standard ou supervision. Pour le mode supervision, il récupère toujours les informations de la carrosserie grâce au couple code produit et code couleur mais il récupère également le champ *Essai*. Si ce champ est vrai alors la transaction créée et envoyée au serveur sera une transaction de type essai avec son propre champ *Essai* à vrai et utilisera les nouvelles règles, sinon la transaction sera une transaction standard avec son champ *Essai* à faux et suivra les anciennes règles expliquées dans la partie précédente.

Pour le mode standard, un bouton *Essai* est rajouté dans l'interface de sélection de la qualité de la pièce. Si ce bouton est activé alors la transaction créée sera de type essai sinon elle sera standard.

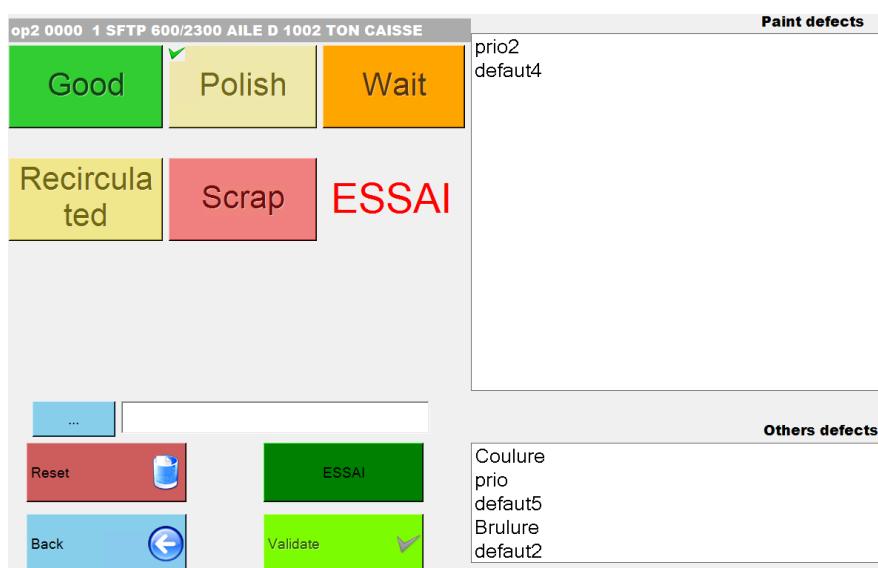


FIGURE 2.11 – SQP Client : Interface de sélection de la qualité d'une pièce avec le bouton Essai activé

Le dernier changement intervient donc dans le service SQP SAP et change les règles d'envoi ou non de données à SAP pour les transactions de type essai. Le fonctionnement est le même que pour celles standards mais à la place de regarder la valeur des booléens *Envoi bon* et *Envoi rebut* il faut regarder la valeur des deux nouveaux : *Envoi bon essay* et *Envoi rebut essay*.

2.3.1.6 Changement des dlls de connexion à SAP

Une des évolutions de ce lot 1 a été de changer les dlls qui permettaient au service SQP SAP de se connecter à SAP et elle avait une charge de travail de quatre jours.

Une dll est une bibliothèque logicielle contenant différentes méthodes. Par exemple, les dlls permettant la connexion à SAP contiennent toutes les méthodes permettant d'établir cette connexion et d'envoyer et de recevoir des données depuis SAP.

Celles utilisées par la version 4.5 de SQP ne sont pas les dlls officielles mais celles d'une autre application permettant de se connecter à SAP en mode graphique : SAP Logon. Parfois avec ces bibliothèques, le serveur SQP se déconnectait de SAP et n'arrivait pas à se reconnecter par lui-même. Le seul moyen trouvé par les équipes de Plastic Omnium était de redémarrer plusieurs fois le serveur jusqu'à ce que la connexion se soit finalement rétablie.

Donc l'entreprise a demandé à Capgemini de changer et d'utiliser les dlls officielles. Pour intégrer ces nouvelles dlls à la place des anciennes, il n'y a pas eu de modification importante dans le code du service mais juste une légère mise à jour. En effet, les méthodes sont quasiment identiques : le principe reste le même mais la manière de l'implémenter change légèrement.

De plus, Plastic Omnium a demandé à ce que cette évolution soit livrée avant le reste du lot 1 parce que les problèmes de déconnexion devenaient trop nombreux. J'ai donc dû faire cette intégration en Visual Basic.NET pour la version 4.5 et une fois que la correction livrée au client a été validé par celui-ci, j'ai dû la reporter sur la version 5.0 en C#.

2.3.1.7 Option code barre de contrôle avant déclaration

La dernière évolution ayant une charge de travail comprise entre 2 et 5 jours était l'ajout d'une option code barre de contrôle lors de la déclaration d'une nouvelle transaction dans le mode standard de SQP Client. Ce scan de contrôle est rajouté juste après le premier scan permettant de récupérer les informations de la carrosserie scannée et permet de contrôler la validité de cette pièce.

La demande de ce scan de contrôle est configurable mais pour cela il a fallu rajouter une option dans la configuration des postes clients afin de l'activer ou non : *USE_BARCODE_FOR_CTRL*.

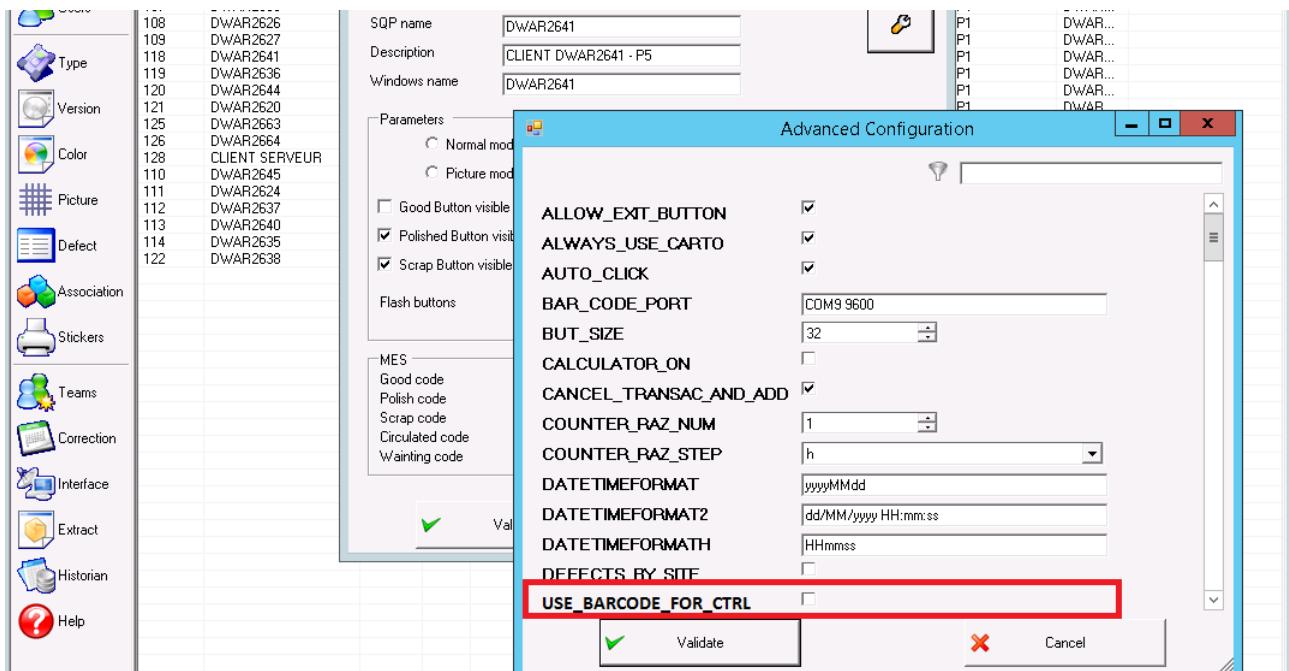


FIGURE 2.12 – SQP Client : Interface de modification d'un poste client

Il a également fallu rajouter un nouveau champ, *Ref_for_Ctrl* dans l'association et dans son interface de modification afin de comparer le scan du code barre de contrôle à cette référence.

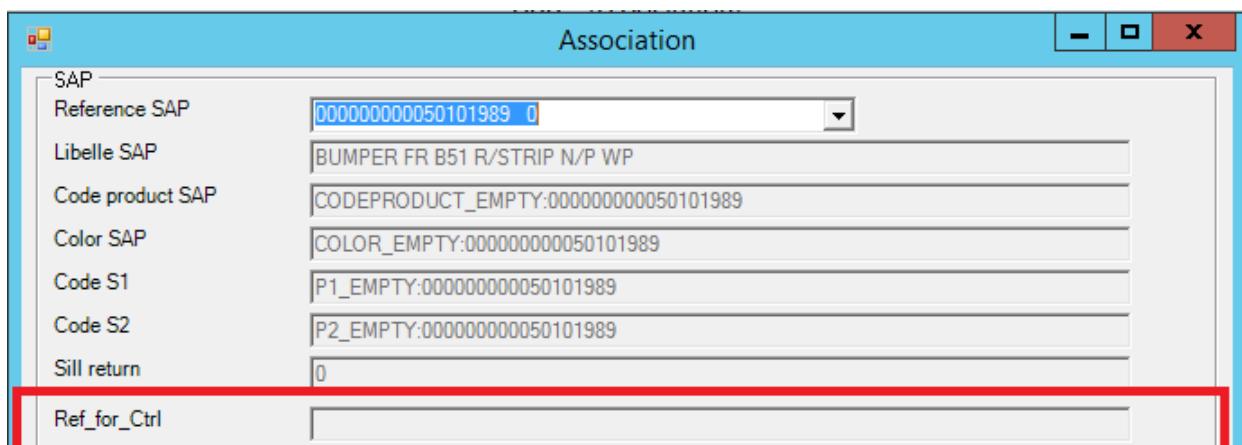


FIGURE 2.13 – SQP Manager : Interface de modification d'une association avec Ref_for_Ctrl

2.3.1.8 Mode supervision sous forme de Service Windows

Comme expliqué, un client SQP peut fonctionner en mode supervision, dans ce cas il surveille un répertoire et dès qu'un fichier est déposé il est traité afin de générer une déclaration automatique de production et d'imprimer une ou plusieurs étiquettes correspondantes. Ce type de client est utilisé en sortie des chaînes peintures et les nouvelles installations dans les usines de Plastic Omnium permettent maintenant d'avoir jusqu'à six sorties. Dans ce cas de figure six stations en mode supervision sont nécessaires lorsque l'on veut imprimer des étiquettes pour une même référence sur six imprimantes différentes, chaque imprimante étant affectée à un poste distinct.

La demande du client est de garder ce mode de fonctionnement à des fins de compatibilité avec les installations existantes et il souhaite que cette fonctionnalité puisse être utilisée comme un service Windows afin ne plus systématiquement utiliser un client lourd (SQP Client). D'autre part, il faut pouvoir utiliser plusieurs instances de ce service sur une même machine. Chaque service sera installé dans un répertoire distinct et aura un fichier de configuration dans lequel l'utilisateur indiquera le répertoire dans lequel les fichiers textes seront déposés, l'imprimante à utiliser (réseau ou local) et aura un fichier de log quotidien répertoriant l'ensemble des traitements effectués.

La première chose à faire a donc été de créer un nouveau service Windows et d'implémenter le même processus que celui présent dans le mode supervision de SQP Client. Pour commencer, j'ai juste dupliqué le code du client et je l'ai modifié pour qu'il soit compatible à un service Windows autonome. J'ai donc dû enlever toutes les interfaces présentes et les remplacer par de nouvelles classes.

Le changement principal a été au niveau de la configuration. SQP Client récupère sa configuration directement depuis le serveur alors que pour ce service elle est contenue dans un fichier de configuration à la racine de l'exécutable. Ce fichier contient, comme pour le client, les informations de connexion à la base de données locale mais aussi le chemin du répertoire à surveiller pour récupérer les fichiers texte, le chemin du fichier de log, le nom de l'imprimante qui lui sera liée ainsi que le nom de ce service. Ce nom est nécessaire parce que comme il doit pouvoir y avoir six fois le même service Windows, chacun de ces services doit avoir un nom différent afin d'être installé et de pouvoir fonctionner en même temps.

2.3.1.9 Overall Equipment Effectiveness (OEE)

La dernière évolution du lot 1 est une évolution indépendante des logiciels existants de la suite SQP. Elle rentre dans le cadre du reporting de Plastic Omnium et permet de calculer l'Overall Equipment Effectiveness (OEE) ou taux de rendement global. C'est-à-dire que le client doit pouvoir connaître les temps d'arrêt de la chaîne peinture ainsi que la cause de ces arrêts. Pour cela, les chaînes peintures transmettront un fichier par heure qui permettra de connaître le temps perdu durant la dernière heure de production. Ces fichiers seront intégrés dans SQP via un nouveau client lourd ou via SQP Manager afin qu'un utilisateur puisse renseigner la raison des arrêts.

Le format de ce fichier est le suivant :

SMR REPORTING DEFINITION		
PKLeine/Llevesque/Jporro		
Nov. 9th		
The file is generated every hour in CSV format.		
No Alarm Log needed by the moment		
Production active time	min	managed throw supervision
Target skids	skids num	
Real skids throw unloading area	skids num	
Empty skids	skids num	
Trials	skids num	
Downtime	skids	
Downtime	time	skids lost during last hour* takt time of paint line

FIGURE 2.14 – SQP OEE : Format du fichier transmis par la chaîne peinture

Comme pour le mode supervision, la demande du client est d'utiliser un service Windows qui traitera le fichier reçu en mettant à jour une nouvelle table dans la base de données, *tbOEE*. Ce service ne sera installé que sur le serveur et sera paramétrable via un fichier de configuration afin d'indiquer le répertoire de dépose des fichiers de reporting et tout autre paramètre utile au bon fonctionnement du service que je présenterai dans cette partie.

La table *tbOEE* reprend l'ensemble des champs du fichier transmis par la chaîne peinture plus un champ pour le code arrêt qui est vide lors de l'ajout d'un enregistrement. Elle contient également un champ pour la date et l'heure de création du fichier, un champ pour l'identifiant du site auquel est lié le serveur ainsi qu'un identifiant unique.

Le champ code arrêt est lié à une autre table *tbCodeArret* dans laquelle on retrouve l'ensemble de ces codes. Cette liste est prédéfinie et est fournie par Plastic Omnium. Les différentes causes d'arrêt sont les suivantes :

- Problème électrique
- Problème mécanique
- Problème d'automatisation
- Problème de processus
- Problème d'équipement
- Problème externe

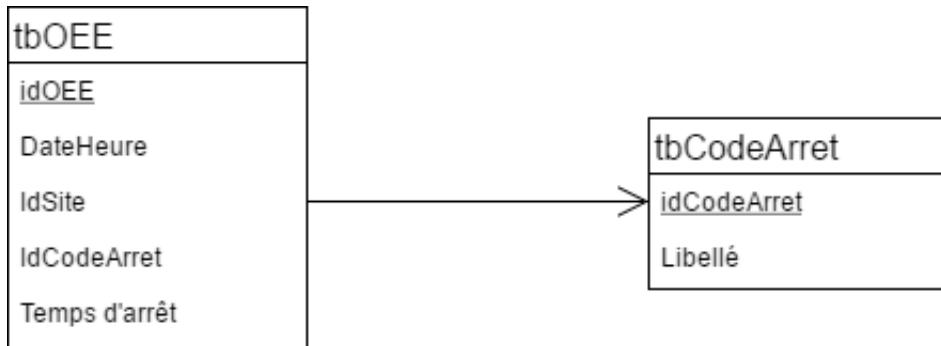


FIGURE 2.15 – SQP OEE : Relation entre la table tbOEE et tbCodeArret

Le script SQL de création de ces deux tables ainsi que de celui permettant d'insérer les codes d'arrêts dans la table tbCodeArret ont été rajoutés dans le même script SQL utilisé pour les évolutions précédentes afin de tout centraliser.

Contrairement au mode supervision, le fichier traité est déplacé dans un répertoire de sauvegarde et non supprimé directement. Ce répertoire est défini dans le fichier de configuration. Les fichiers sauvegardés doivent être supprimés au bout de X jours. Cette valeur X est de 30 jours par défaut et a également été rajoutée dans la configuration afin qu'elle soit modifiable par les utilisateurs. Afin d'effectuer cette suppression, j'ai rajouté un minuteur qui, tous les jours, parcourt la liste des fichiers de reporting et supprime ceux dont la date de création remonte à plus de X jours.

Une autre demande du client quant à cette évolution est de créer un nouveau client lourd, nommé SQP OEE et dédié à la gestion des arrêts de la chaîne de peinture. Il permet de renseigner les causes d'arrêts. On y retrouve l'ensemble des arrêts sur la journée en cours, triés par date de création, avec la possibilité via une liste déroulante d'affecter la cause d'arrêt pour chaque ligne. Cette liste déroulante propose l'ensemble des arrêts de la table tbCodeArret.

The screenshot shows a user interface titled "SAISIE DES CAUSES D'ARRETS". At the top, there is a date selector labeled "Date de début" with the value "02/10/2016 15:03:17". Below the date selector is a table with three columns: "Heure de référence", "Temps d'Arrêt (min)", and "Cause d'arrêt". The table contains three rows of data:

Heure de référence	Temps d'Arrêt (min)	Cause d'arrêt
08/12/2016 08:00	03:30	(dropdown menu)
08/12/2016 09:00	06:00	(dropdown menu)
08/12/2016 10:00	30:25	(dropdown menu)

FIGURE 2.16 – SQP OEE : Interface de modification des causes d'arrêt

Plastic Omnium a également demandé que cette fonctionnalité soit rajoutée dans SQP Manager via l'ajout d'un menu supplémentaire.

Cette évolution étant indépendante du reste de SQP, le seul point commun est l'utilisation de la même base de données. J'ai donc pu la développer avec une meilleure architecture que les autres composants.

Afin de ne pas dupliquer encore une fois le code, j'ai utilisé une couche d'accès aux données ou *Data Acces Layer* (DAL) en Anglais. Elle regroupe toutes les méthodes accédant et récupérant des informations depuis la base de données. Cela permet de simplifier l'accès et l'ajout de nouvelles méthodes et d'améliorer la maintenabilité du code. J'ai créé cette couche dans une bibliothèque logicielle (dll) nommée *LibOEE.dll* afin qu'elle soit accessible par toutes les applications ayant référencées cette dll.

Généralement avec cette couche, deux autres sont utilisées :

- La couche de présentation, il s'agit de la couche contenant toutes les interfaces du projet
- La couche métier, il s'agit de la couche contenant toutes les classes permettant de stocker et de traiter les données reçues par celle d'accès aux données.

Comme il n'y a qu'une seule interface pour cette évolution et seulement deux classes différentes, ces deux autres couches ne sont pas utiles.

Toujours dans un souci de ne pas dupliquer le code, j'ai ajouté l'interface demandée dans cette même dll. Grâce à ceci, l'interface utilisée par SQP Manager et par SQP OEE sera la même.

Au final, comme le rôle de SQP OEE est d'afficher l'interface de modification des causes d'arrêts, ce nouveau client lourd ne fait que de référencer cette nouvelle dll et d'afficher, grâce à celle-ci, cette interface.

Quant à SQP Manager, j'ai d'abord dû référencer la dll puis rajouter un nouveau bouton dans le menu principal et enfin afficher, toujours grâce à cette dll, le nouvel écran lors de l'appui sur ce bouton.

Au final, l'architecture de cette évolution peut être résumée par ce diagramme :

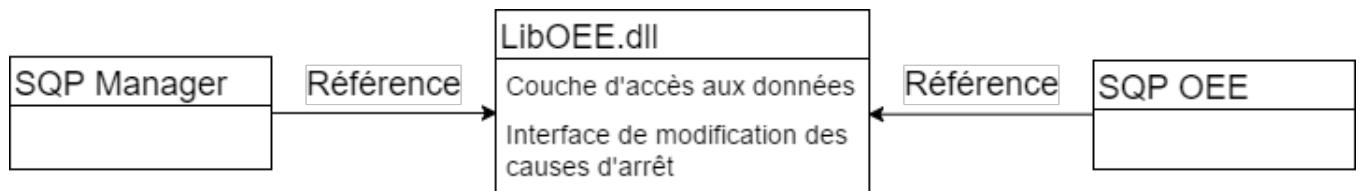


FIGURE 2.17 – SQP OEE : Architecture de l'évolution Overall Equipment Effectiveness

2.3.2 Évolutions du lot 2

Une fois le lot 1 livré et les différents bugs corrigés, j'ai pu commencer le développement des évolutions comprises dans le deuxième lot. Ce lot contient également 11 évolutions mais la majorité de celles-ci ont une charge de travail faible. En effet, 7 évolutions sur les 11 possèdent une charge de travail inférieure ou égale à un jour. Les quatre dernières évolutions ont une charge de travail allant de 1.5 à 2.5 jours.

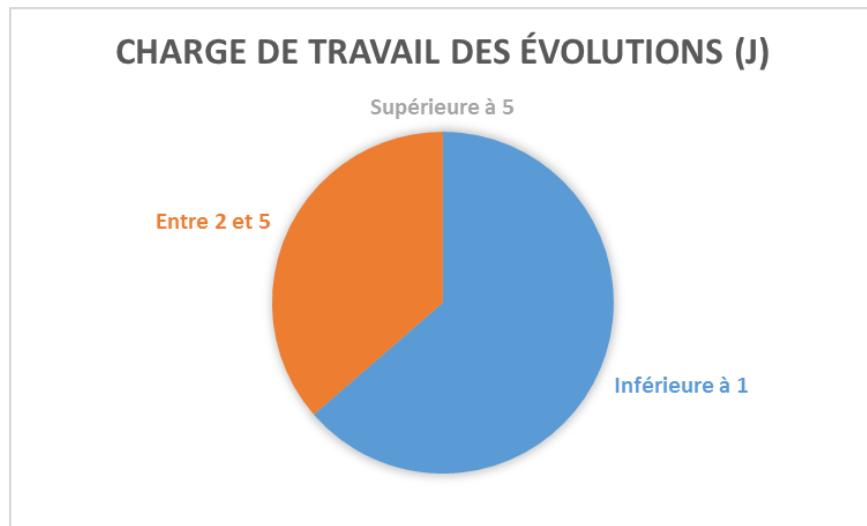


FIGURE 2.18 – Charge de travail des évolutions du lot 2 (en j)

Contrairement au premier lot, nous avons été deux à développer les différentes évolutions donc j'expliquerai brièvement toutes les évolutions et plus en détail celles sur lesquelles j'ai travaillé. Cette deuxième personne est un stagiaire arrivé au mois de Juillet et auquel, en parallèle de mes développements, j'ai dû transmettre mes connaissances sur SQP et lui apporter mon aide s'il en avait besoin.

Deux de ces évolutions étaient très simples et rapides à mettre en place (charge de travail de 0,25 jour) et ont donc été effectuées en premier par le nouveau stagiaire.

La première était de rajouter un bandeau dans SQP Client, en haut de la fenêtre, affichant les statistiques ajoutées dans le lot précédent.

La deuxième était de changer le nombre de caractères des numéros des étiquettes et de le passer de sept à six caractères.

2.3.2.1 Logs SQP / SAP dans le même dossier

Comme expliqué précédemment, chaque logiciel de SQP possède un fichier de log mais ce fichier ne trouve pas forcément dans le même dossier. En effet, ce dossier est le dossier temporaire de l'utilisateur ayant lancé le logiciel alors que pour les services Windows, l'utilisateur est le système lui-même donc le fichier de log se trouvera dans le dossier temporaire de Windows.

Plastic Omnium a donc demandé à ce que tous ces fichiers de log soit centralisés dans un seul et même dossier afin qu'ils soient plus facile d'accès.

La solution proposée a été de les regrouper dans un dossier *Logs* à la racine du dossier d'installation de SQP (par défaut *C :|Program Files (x86)|SQPNET|Logs*).

Cette évolution a été réalisée par le deuxième stagiaire et avait un demi-jour de charge de travail.

2.3.2.2 Ajouter 4 champs texte libres dans l'association pour l'impression d'étiquettes

Une autre demande de Plastic Omnium a été de rajouter quatre nouveaux champs libres *PREFx*, en plus des deux existants, dans les associations pour l'impression des étiquettes après chaque nouvelle saisie de pièces. Chacun de ces nouveaux champs a une valeur par défaut qui est renseignée dans la configuration de chaque poste client.

Cette évolution avait une charge de travail d'un jour et demi et a été réalisée par le deuxième stagiaire.

2.3.2.3 Ajouter 4 nouveaux champs dans l'association venant de SAP

Cette autre évolution est quasiment similaire à celle présentée précédemment : la demande a été de rajouter quatre nouveaux champs *Code Sx*, aux deux existants, dans les associations. Mais contrairement à cette autre évolution, la valeur de ces champs est importée directement depuis SAP grâce au service SQP SAP.

Comme pour l'évolution précédente, elle a été développée par le deuxième stagiaire et avait deux jours comme charge de travail.

Ci-dessous, l'interface de modification d'une association après avoir rajouté ces huit nouveaux champs :

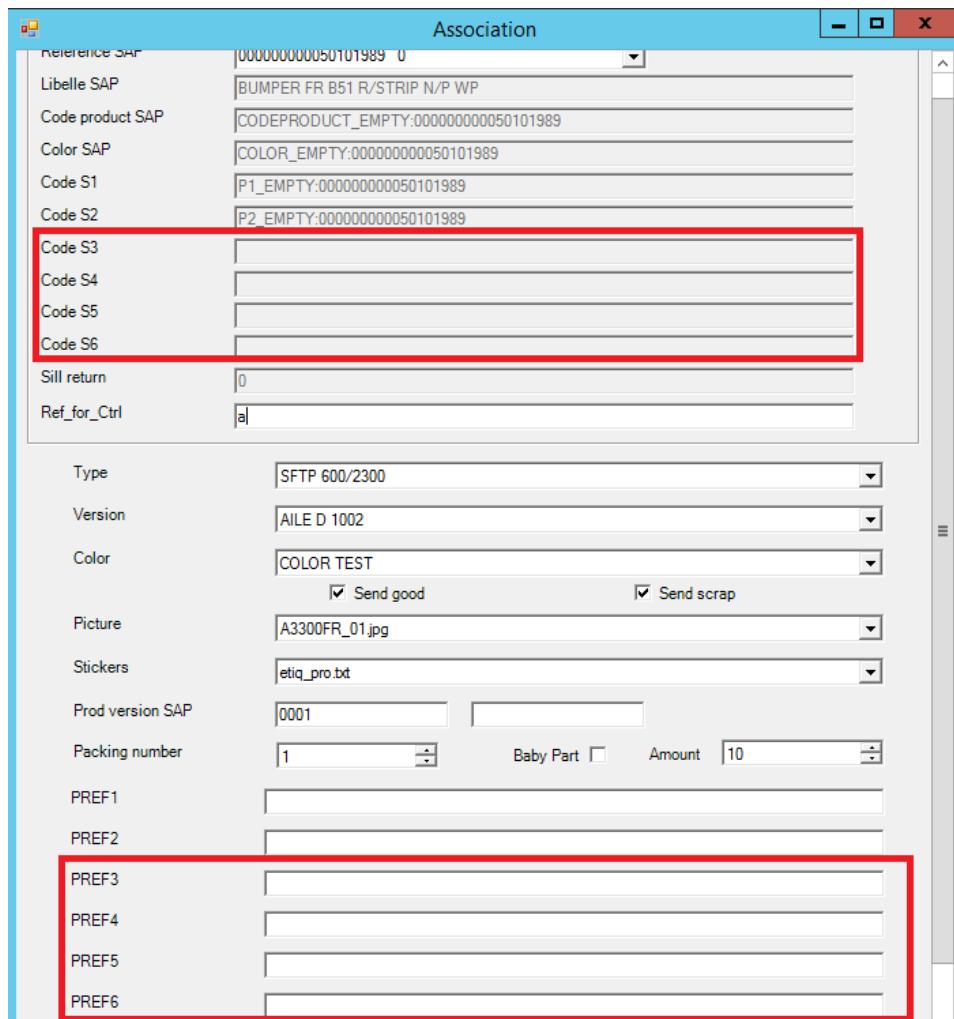


FIGURE 2.19 – SQP Manager : Interface de modification d'une association après l'ajout de 8 nouveaux champs

2.3.2.4 Saisie d'une quantité lors de la lecture automatique

Comme expliqué dans la présentation du mode standard de SQP Client, ce mode possède deux sous-modes :

- Scan d'un code barre
- Sélection manuelle du type, de la version et de la couleur

Et seul le sous-mode de sélection manuelle permet de déclarer une ou plusieurs nouvelles pièces. La demande de Plastic Omnium a donc été d'ajouter cette fonctionnalité au second sous-mode (scan d'un code barre) et avait une charge de travail de deux jours et demi.

Pour cela, j'ai dû incorporer deux nouvelles lignes dans la table *tbAssociation* et dans l'écran de modifications de ces associations dans SQP Manager :

- Une option nommée *Baby Part* permettant d'activer ou non cette sélection multiple.
- Une ligne *Quantité* permettant de rentrer le nombre par défaut de pièces à déclarer.

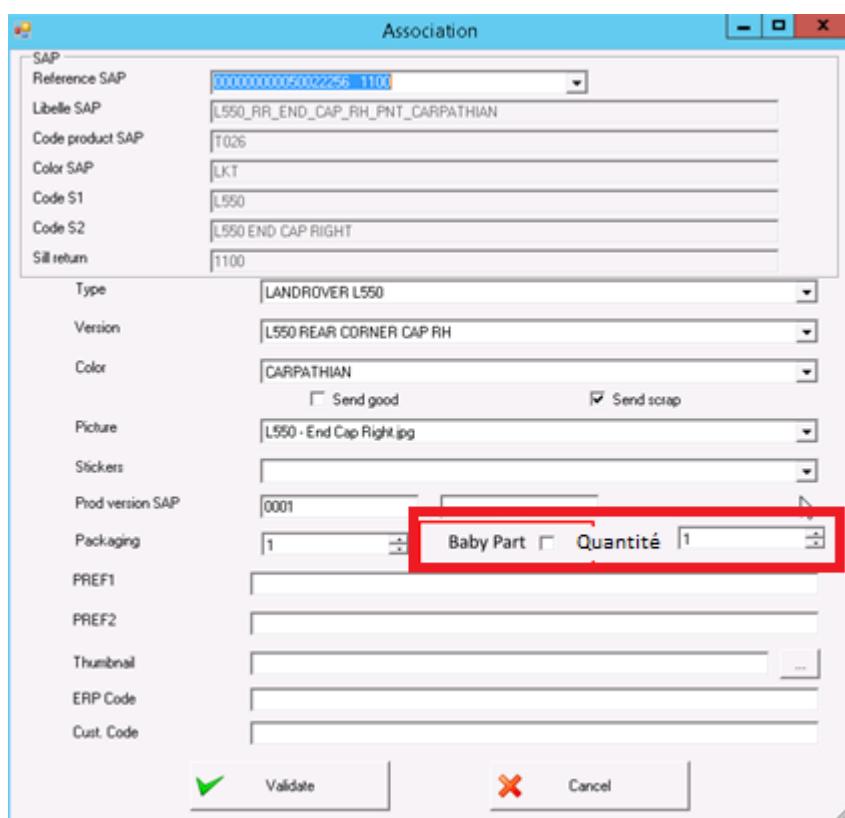


FIGURE 2.20 – SQP Manager : Interface de modification des associations pour la saisie multiple lors de la lecture automatique

Ensuite, il a fallu insérer dans SQP Client deux nouveaux contrôles permettant la modification de cette quantité lorsque l'option *Baby Part* est active. Plastic Omnium a précisé que ce rajout devait se situer sur l'écran de sélection de l'état des nouvelles carrosseries.



FIGURE 2.21 – SQP Client : Interface de saisie de l'état des pièces permettant de modifier leur quantité pour le sous-mode automatique

L'appui sur le bouton vert *Calculatrice* ouvre un nouvel écran permettant de modifier la quantité. Ce nouvel écran existait déjà mais a dû être légèrement modifié afin d'être compatible avec cette nouvelle évolution.

2.3.2.5 Statut Actif / Inactif dans la liste des associations

Avant de parler de l'évolution, il faut tout d'abord préciser certains éléments. Grâce à SQP Manager, il est possible de désactiver un type, une version ou une couleur. Cette désactivation de l'un de ces critères a pour conséquence de rendre toutes les associations où se trouvent ces critères inactives donc non sélectionnables sur le client SQP.

Par contre dans SQP Manager, il n'y a aucun signe distinctif dans la liste des associations indiquant leur activité. La demande du client a donc été de permettre cette distinction. La charge de travail était d'une demi-journée.

Pour cela, dans la liste des associations, j'ai inséré une case à cocher pour chaque association permettant de connaître leur activité. Si cette case est cochée alors l'association est active sinon elle est inactive. J'ai également ajouté une autre case à cocher permettant de sélectionner toutes les associations ou seulement les actives.

	Add	Modify	Delete	<input checked="" type="checkbox"/> Inactive associations										
I...	Reference SAP	Libelle SAP	Prod ve...	Type	Version	Color	Pi...	Stic...	P1	P2	Send g...	Send sc...	Code pr...	Co
<input type="text"/>	Enter text here	<input type="text"/>	Enter text here	<input type="text"/>	Enter tex...	<input type="text"/>	E...	E...	E...	E...	Ente...	Ente...	E...	En
<input checked="" type="checkbox"/>	SFTP AILEZU1002	SFTP 600/2300 AIL...	PP15	TYPE2	VERSION2	BLEU	A3...	Typ...			1	1	p	
<input type="checkbox"/>	000000978076153431	GRILLE 050W M...	PP001	TYPE3	VERSION1	BLEU CLAIR	G...	etiq...			1	1	EC45	43
<input type="checkbox"/>	00000000005501637a	BMW Z3 RR HP B...	100	TYPE3	VERSION1	JAUNE	JB...	TO...			1	0	CODEP...	CC
<input type="checkbox"/>	SFTPAILEZU3456	GRILLE INNER 07...	0001	NEWTYPE	VERSION2	BLEU1	G...	etiq...	DEFAUT	1	0		CODEP...	CC
<input checked="" type="checkbox"/>	SFTPAILEZA1002	SFTP 600/2300 AIL...	0001	SFTP 600/2...	AILE D 1002	TON CAISSE	vi...	etiq...	SF...	1	1	p	Z	
<input checked="" type="checkbox"/>	SFTPAILEZU1002	SFTP 600/2300 AIL...	0001	SFTP 600/2...	AILE D 1002	AUTOGRAIN...	A3...	etiq...	SF...	DEFAUT	1	1	p	
<input checked="" type="checkbox"/>	SFTPAILEZU1002	SFTP 600/2300 AIL...	0001	SFTP 600/2...	AILE D 1002	BLANC	A3...	etiq...	SF...		1	1	p	
<input checked="" type="checkbox"/>	SFTPAILEZU1002	SFTP 600/2300 AIL...	0001	SFTP 600/2...	AILE D 1002	APPRETE	A3...	etiq...	SF...	xxxPRE...	1	1	p	a
<input checked="" type="checkbox"/>	000000000050101989	BUMPER FR B51 ...	0001	SFTP 600/2...	AILE D 1002	COLOR TEST	A3...	etiq...			1	1	CODEP...	CC

FIGURE 2.22 – SQP Manager : Interface affichant la liste de toutes les associations

2.3.2.6 Amélioration du processus de création de licence serveur

SQP a besoin d'une licence afin de pouvoir être utilisé et cette licence a pour but principal de contenir les informations de connexion à la base de données serveur. Par conséquent, tous les logiciels ayant besoin de se connecter à cette base de données ont besoin d'une licence.

Il s'agit d'un simple fichier qui peut être unique sur un même poste ou qui peut être à la racine de chaque exécutable nécessitant une licence. Si elle est unique, son chemin d'accès est renseigné, manuellement, soit dans une variable d'environnement, soit dans une clé dans la base de registres de Windows.

De plus, afin de générer une licence, il faut utiliser un logiciel nommé *SQP Crypto*. Dans ce logiciel, il faut renseigner le serveur sur lequel se trouve la base de données, le nom de cette base, ses identifiants de connexion ainsi que la date de validité de la licence générée. En appuyant sur le bouton *Generate*, l'utilisateur doit sélectionner le chemin où la licence sera créée.

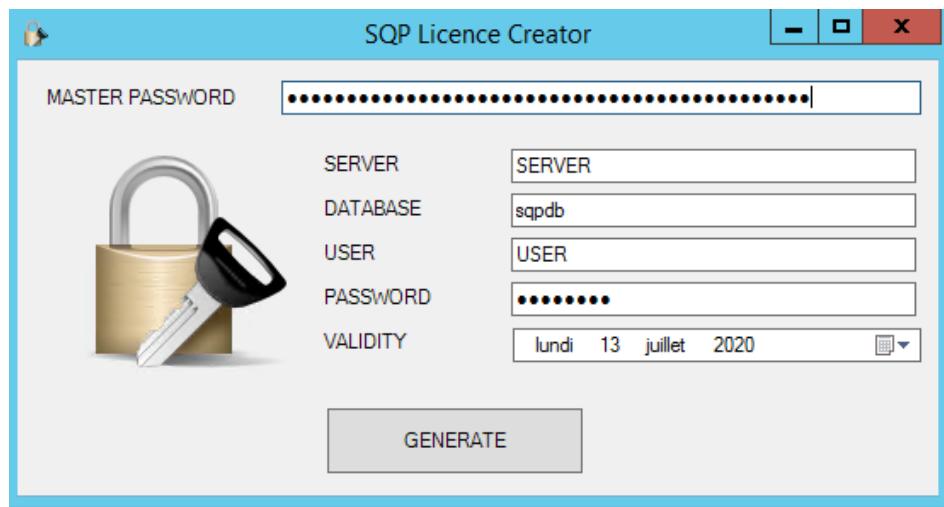


FIGURE 2.23 – SQP Crypto : Interface permettant de générer une licence

La demande du client a été d'automatiser la création de la clé dans la base de registre depuis SQP Crypto plutôt que de devoir la créer manuellement. Cette requête avait une charge de travail d'un jour.

Pour cela, après que l'utilisateur ait sélectionné le chemin de création de la licence, il suffit d'ajouter, grâce à une méthode déjà présente dans le framework .NET, la clé de registre avec pour valeur ce chemin de création.

2.3.2.7 Mode de numérotation des réimpressions d'étiquette

Depuis SQP Manager, l'utilisateur peut réimprimer des étiquettes. Dans ce cas, l'étiquette est réimprimée avec son numéro d'origine. Plastic Omnium souhaitait mettre en place une numérotation dédiée à ces réimpressions tout en conservant le mode actuel.

La nouvelle numérotation utilise toujours le numéro d'origine de l'étiquette mais y ajoute un code unique à chaque SQP Manager, nommé *UNIQ_LABEL_NUMBER* et qui permet d'identifier le manager l'ayant réimprimé.

De plus, afin de faire cohabiter ces deux modes différents, il a fallu rajouter une nouvelle propriété, nommée *USE_DEDICATE_CHRONO_ON_REIMP*, dans la configuration des différents SQP Manager afin de sélectionner l'un ou l'autre de ces modes.

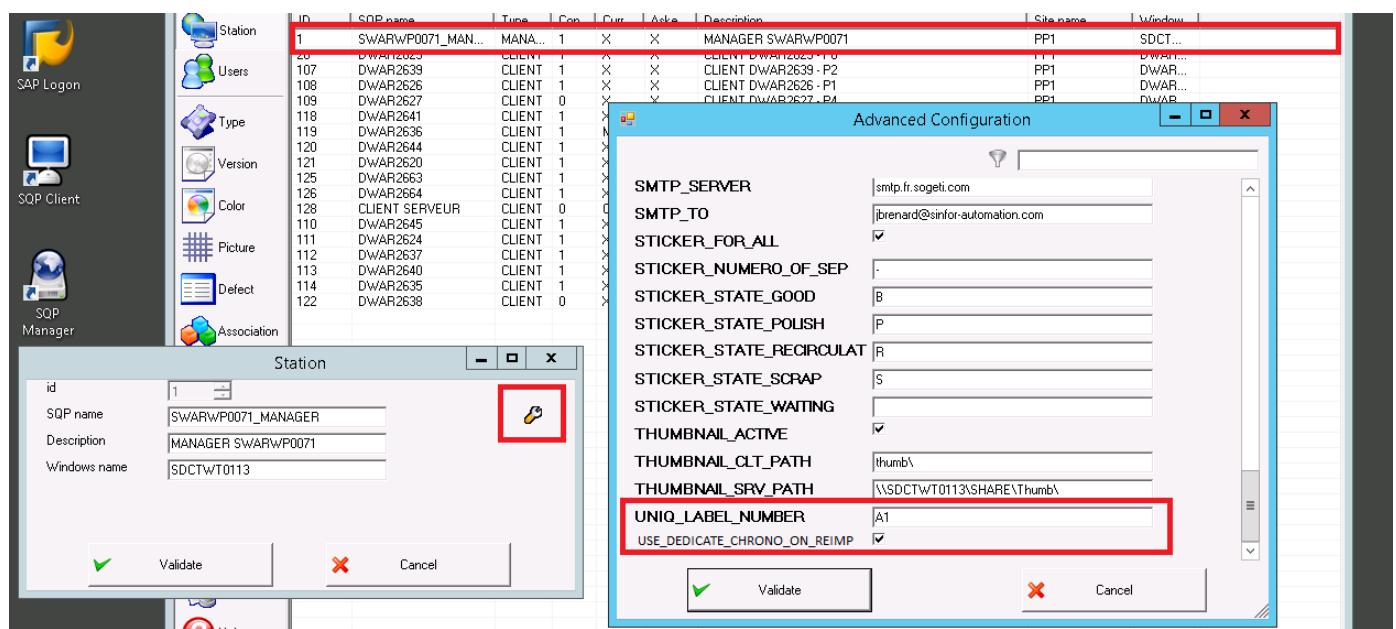


FIGURE 2.24 – SQP Manager : Interface de modification de la configuration d'un SQP Manager

2.3.3 Évolutions du lot 3

Contrairement aux deux premiers lots, le troisième lot ne contient que trois évolutions mais l'une d'entre elle est plus longue que le lot 2 entier. Cette évolution a une charge de travail de 20 jours et les deux autres de respectivement 3 et 2.5 jours.

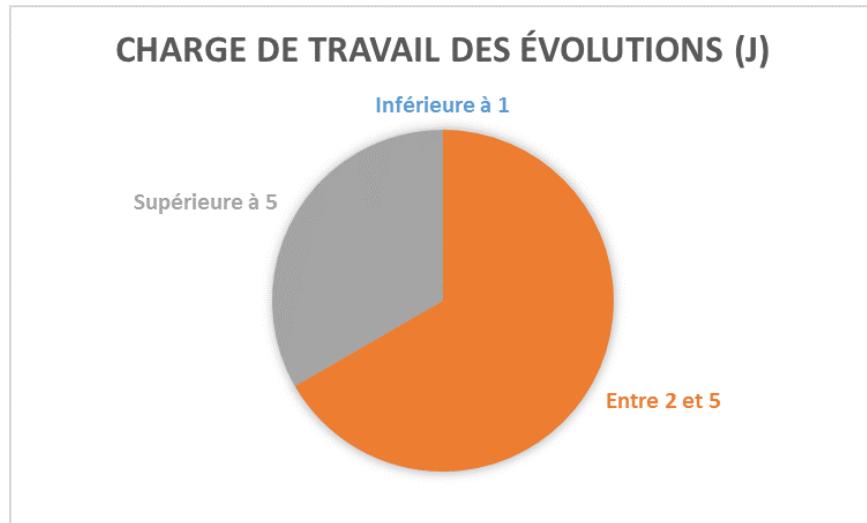


FIGURE 2.25 – Charge de travail des évolutions du lot 3 (en j)

Comme pour le lot précédent, nous étions deux pour effectuer les développements donc je présenterai brièvement ces trois évolutions et j'irai plus dans les détails pour celles auxquelles j'ai participé.

2.3.3.1 Filtre de recherche pour les formulaires Type, Version et Teinte

SQP Manager permet de lister tous les types, toutes les versions et toutes les teintes disponibles mais aucun filtre n'est implémenté pour faciliter la recherche de ces éléments. Plastic Omnium a demandé d'en intégrer un pour chacun de ces éléments.

Je n'ai pas développé cette évolution mais il a suffi d'ajouter un champ pour ce filtre et d'utiliser le même principe que le filtre déjà présent dans l'interface de modifications de la configuration des postes.

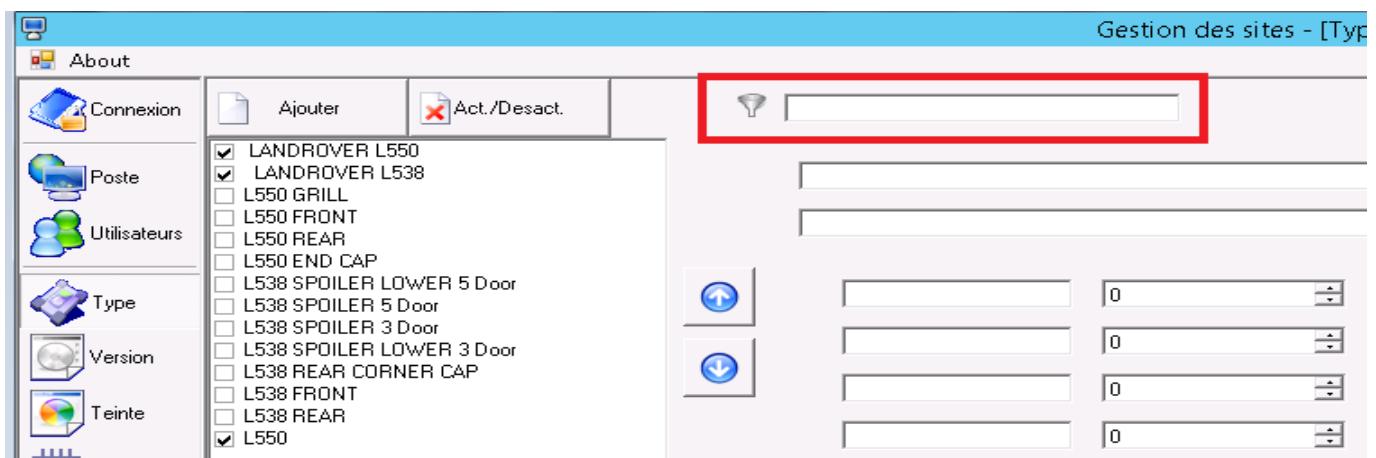


FIGURE 2.26 – SQP Manager : Interface listant tous les types avec l'ajout du filtre

2.3.3.2 Refonte du système d'envoi de mail

Lorsque le service SQP SAP détecte une erreur lors de l'envoi ou l'import de données, le service envoie un mail à l'administrateur afin de la lui notifier. Pour envoyer ces mails, SQP se base sur un programme nommé *BLAT*. Le problème est que Plastic Omnium ne reçoit aucun message d'erreur lorsque les mails ne sont pas envoyés. Notre travail a été d'améliorer cette fonctionnalité.

La solution qui a été préconisée par Capgemini lors de l'étude a été d'utiliser la bibliothèque d'envoi de mails intégrée au framework .NET et d'envoyer les mails depuis un expéditeur avec une boite mail réelle afin de pouvoir consulter les erreurs potentielles.

Elle a été facilement mise en place par le second stagiaire de fait que toutes les informations nécessaires étaient déjà présentes (expéditeur, destinataire, sujet et corps du mail).

2.3.3.3 SQP Client : Refonte du mode offline (SQL vers XML)

La dernière évolution était la plus importante. Son objectif était de remplacer la base de données locale des postes clients par des fichiers XML afin d'alléger l'installation de SQP sur ces postes.

Avant de générer les fichiers XML depuis la base de données existante, j'ai d'abord décidé de créer une couche d'accès aux données pour SQP Client et Supervision afin de regrouper tous les appels à la base de données.

Dans cette couche, j'ai rajouté le patron de conception *Bridge* suivant :

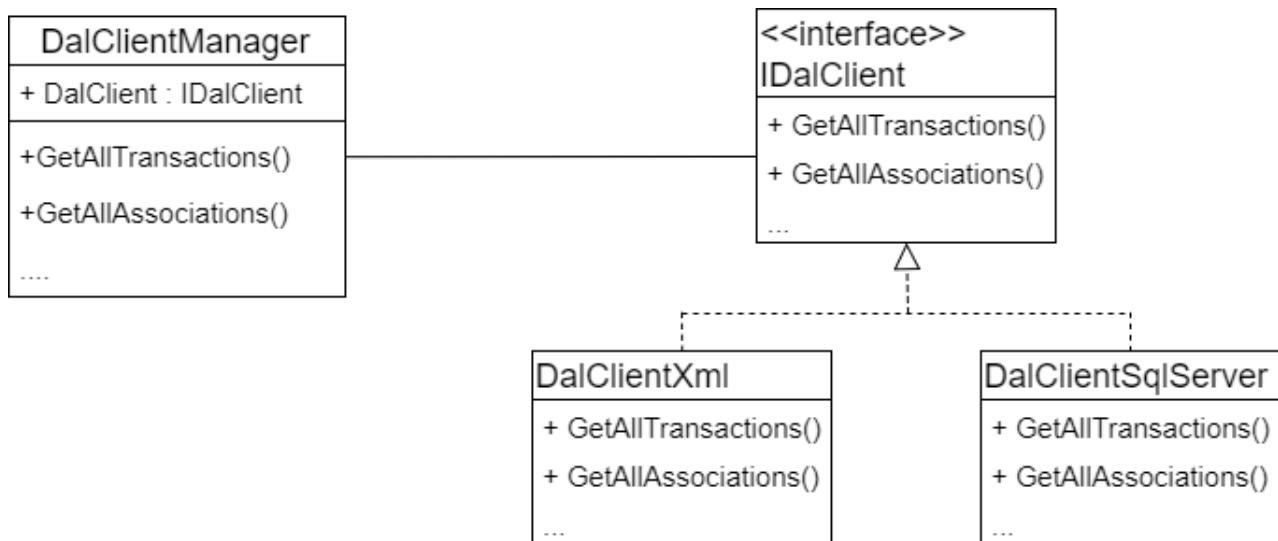


FIGURE 2.27 – Patron de conception *Bridge* pour la refonte du mode offline de SQP Client

Les deux classes *DalClientXml* et *DalClientSqlServer* héritent de la même interface nommée *IDalClient* et possèdent donc les mêmes méthodes (adaptées respectivement aux fichiers XML et à la base de données locale SQL Server).

La classe *DalClientManager* contient un attribut *DalClient* pouvant être du type *DalClientXml* ou *DalClientSqlServer*. Elle possède également les mêmes méthodes mais celles-ci servent uniquement à appeler les méthodes de l'attribut *DalClient*. Par exemple, la méthode

GetAllTransactions() de cette classe appelle la méthode *GetAllTransactions()* de l'attribut *DalClient*.

Cette classe fait l'interface avec le reste du code de SQP Client et Supervision c'est-à-dire que si ces deux programmes veulent effectuer n'importe quelle action sur la base de données locale, ils appellent les méthodes de la classe qui appelleront ensuite celles de l'attribut *DalClient*.

L'implémentation de ce patron permet de changer très facilement de type de base de données utilisée (SQL Server ou XML). En effet grâce à la classe d'abstraction *DalClientManager*, il suffit de changer une seule ligne dans son constructeur afin de créer l'attribut *DalClient* soit de type *DalClientSqlServer*, soit de type *DalClientXml* et d'appeler ses méthodes correspondantes.

Ensuite, après avoir créé cette couche, j'ai récupéré, depuis *stackoverflow.com*, un script SQL permettant de générer la base de données locale SQL Server en fichiers XML. Ce script génère un fichier XML par table contenue dans SQL Server.

```
<?xml version="1.0" encoding="utf-8"?>
<tbTransaction xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<row>
  <ID_TRANSACTION>1</ID_TRANSACTION>
  <dateHeure>2017-08-02T11:48:59.0182458+02:00</dateHeure>
  <ID_op>5</ID_op>
  <ID_poste>0</ID_poste>
  <ID_Type>17</ID_Type>
  <ID_teinte>20</ID_teinte>
  <Etat>B</Etat>
  <numTransac>1</numTransac>
  <ID_Version>19</ID_Version>
  <FLAG>0</FLAG>
  <ID_defaut>9999</ID_defaut>
  <dtTransmit>2017-08-02T11:48:59.0182458+02:00</dtTransmit>
  <IsTrial>0</IsTrial>
</row>
</tbTransaction>
```

FIGURE 2.28 – Exemple de fichier XML : tbTransaction.xml

Par souci de lecture, des champs ont été supprimés.

Pour pouvoir récupérer les données stockées dans ces fichiers, j'ai dû concevoir une classe par fichier avec les mêmes propriétés. Pour cela, un outil présent par défaut dans Visual Studio permet de créer les classes correspondantes à un fichier XML. Par exemple, pour le fichier *TbTransaction.xml* ci-dessus, les classes suivantes ont été générées :

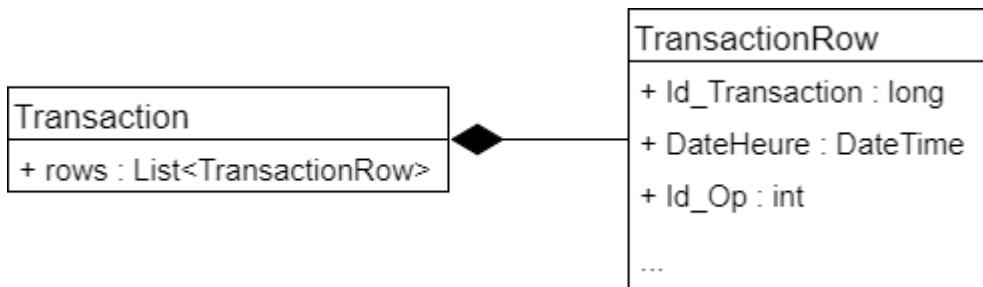


FIGURE 2.29 – Classes Transaction et TransactionRow

De plus, j'ai créé une nouvelle couche dite métier pour contenir toutes ces classes.

Afin de remplir les classes avec les informations des fichiers correspondants, une classe est présente par défaut dans le framework .NET : *XmlSerializer*.

Pour l'exemple ci-dessus, elle permet de remplir la liste *rows* de la classe *Transaction* avec tous les enregistrements du fichier *tbTransacion.xml*.

Afin de modifier n'importe quel fichier XML, il suffit de modifier la liste récupérée précédemment et d'utiliser cette même classe *XmlSerializer* pour réécrire les données dans le fichier.

Enfin, j'ai commencé à modifier toutes les méthodes contenant des requêtes SQL de la classe *DalClientSqlServer* en méthodes compatibles avec les classes créées précédemment.

A la place d'utiliser des requêtes SQL, j'ai utilisé des requêtes LINQ (Language-Integrated Query) qui peuvent être utilisées pour n'importe quelle liste en .NET. Elles sont également présentes par défaut dans le framework .NET et intègrent les mêmes clauses que le SQL : Select, From, Where, Order By, Group By, Join, etc.

Comme ces deux types de requêtes possèdent les mêmes clauses, la migration de l'une à l'autre n'a pas été compliquée à réaliser mais surtout longue car il y avait plus d'une quarantaine de méthodes à modifier.

2.4 Correction de bugs

Comme je l'ai rapidement expliqué dans les parties précédentes, au cours du développement du lot 1, trois corrections de bugs pour la version 4.5 ont été demandées par Plastic Omnium. Comme pour le changement des dlls, j'ai dû développer toutes les corrections en VB.NET et les migrer en C# pour la version 5.0 une fois qu'elles étaient validées par Plastic Omnium.

2.4.1 Bugs liés à l'instabilité du réseau

Deux de ces trois problèmes sont liés entre eux et sont causés par l'instabilité du réseau dans les usines de POAE.

Le premier de ces deux bugs était le fait que l'envoi d'une nouvelle transaction du client au serveur expirait. Par conséquent cette transaction n'était jamais transmise au serveur. Ce problème d'expiration était problématique pour l'intégrité des données dans la base de données du serveur et dans SAP. Il ne se posait problème que pour l'insertion d'un nouvel enregistrement ou la modification d'un existant dans une seule table de la base de données : *tbTransaction*. Le problème était une ancienne requête d'insertion dans cette même table qui n'avait pas aboutie suite à problème réseau et qui bloquait toute nouvelle insertion.

La solution a été d'exécuter une requête SQL qui supprime toutes les requêtes bloquantes dès lors qu'une transaction n'est pas transmise pour cause de délai d'envoi expiré.

Le deuxième bug concernait l'envoi de données du serveur SQP à SAP. Le souci était l'envoi de la même transaction plusieurs fois à SAP. Ceci posait également un gros problème pour l'intégrité des données et le suivi du stock. Pour comprendre ce problème, il faut savoir que chaque transaction possède un champ *Flag* (nommé *RFC* dans l'interface ci-dessous) qui permet de connaître l'état dans lequel elle se trouve. Cet état est défini par quatre valeurs :

- 0 : La transaction n'a pas encore été envoyée à SAP
- 1 : La transaction est en cours d'envoi
- 2 : La transaction a été envoyée et peut être corrigée s'il y a une erreur
- 3 : La transaction a été corrigée et renvoyée et ne peut pas être modifiée à nouveau

ID	Date	Station	Current ...	Users	Referen...	Type	Color	Version	RFC
Entert...	Ente...	Ente...	Ente...	Ente...	Ente...	Ente...	Ente...	Ente...	Ente...
1071	03/07/...	BORNE1	C	Capgem... 000000...	SFTP 6...	COLOR...	AILE D ...	0	
1070	03/07/...	BORNE1	C	Capgem... 000000...	SFTP 6...	COLOR...	AILE D ...	0	
1069	03/07/...	BORNE1	C	.SUPE... 000000...	SFTP 6...	COLOR...	AILE D ...	0	
1068	30/06/...	BORNE1	C	.SUPE... 000000...	SFTP 6...	COLOR...	AILE D ...	0	
1067	30/06/...	BORNE1	C	.SUPE... 000000...	SFTP 6...	COLOR...	AILE D ...	0	
1066	29/06/...	BORNE1	C	.SUPE... 000000...	SFTP 6...	COLOR...	AILE D ...	2	
1065	22/06/...	BORNE1	C	Capgem... SFTPAI...	SFTP 6...	APPRE...	AILE D ...	2	
1064	22/06/...	BORNE1	C	Capgem... SFTPAI...	SFTP 6...	APPRE...	AILE D ...	2	
1063	22/06/...	BORNE1	C	op3 SFTPAI...	SFTP 6...	TON C...	AILE D ...	2	
1062	22/06/...	BORNE1	C	op3 SFTPAI...	SFTP 6...	TON C...	AILE D ...	2	

FIGURE 2.30 – SQP Manager : Interface de modification des transactions

Le problème venait du fait que la mise à jour de ce flag n'arrivait pas à s'effectuer. Les transactions n'étaient alors pas marquées comme déjà envoyées à SAP ce qui avait pour conséquence le ré-envoi de ces transactions par le service SQP SAP. Ce problème était causé, comme pour le bug précédent, par une requête qui bloquait l'insertion ou la modification d'une ligne dans la table *tbTransaction*. Et comme pour le problème précédent, la solution a été d'exécuter la même requête SQL permettant la suppression de toutes les requêtes bloquantes.

2.4.2 Problème fréquence d'import des données depuis SAP

Comme je l'ai expliqué brièvement, le service SQP SAP rapatrie une seule fois par jour les nouvelles données de SAP vers SQP. Un nouveau problème a été remonté par Plastic Omnium lorsque j'ai livré la version 4.5 avec les nouvelles dlls. L'import des données depuis SAP s'effectuait toutes les minutes au lieu d'une seule fois par jour.

Au final, ce bug ne se produisait que lorsque la date du dernier import remontait à plus de deux jours et existait avant le changement des dlls. Il n'avait pas été détecté auparavant parce que sur les différents serveurs en production, cette date n'excède jamais un jour. Il a été décelé lorsque le client a fait sa recette afin de tester le fonctionnement des nouvelles dll, sur un serveur de pré-production sur lequel la date était supérieure à deux jours.

Le problème se trouvait dans la méthode permettant de récupérer la date du nouvel import à effectuer. Avant sa correction, le service rapatriait, toutes les minutes, X fois les données depuis SAP avant de passer à son fonctionnement normal avec X le nombre de jours entre la date actuelle et la date du dernier import. La solution a donc été de modifier et corriger cette méthode de récupération de la date du nouvel import.

2.5 Déploiement automatique

SQP étant installé dans plusieurs usines de plusieurs pays et ses mises-à-jour étant manuelles, le déploiement des nouvelles versions devait être fait par un ou plusieurs salariés de Plastic Omnium. Ils devaient donc se déplacer d'usine en usine ce qui représentait un coût non négligeable pour l'entreprise. Plastic Omnium voulait donc mettre en place une solution de déploiement automatique afin de réduire ses coûts. Cette solution avait déjà été demandée et utilisée par Plastic Omnium dans le cadre d'un autre logiciel en tierce maintenance application au plateau mutualisé nommé PPCM et voulait l'étendre à SQP.

Ce déploiement ne concerne que les mises à jour de SQP et non pas les nouvelles installations.

Mon travail a donc été dans un premier temps de me familiariser avec la solution mise en place pour PPCM et ensuite de l'adapter à SQP. Je vais donc présenter cette première solution et ensuite expliquer les modifications que j'ai dû apporter.

2.5.1 Déploiement automatique de PPCM

Le déploiement automatique permet à Plastic Omnium de choisir sur quel serveur doit être installé la mise à jour de PPCM et il utilise des scripts Powershell qui s'exécute tous les jours grâce au planificateur de tâches de Windows.

Le Powershell est un langage de script fonctionnant uniquement sur Windows et permettant d'interagir avec ce système d'exploitation. Les actions les plus utilisées sont, par exemple, de lister les fichiers dans un répertoire ou encore de créer, renommer, supprimer un fichier ou un dossier. Il s'agit de l'équivalent Windows des langages Shell / Bash sous Unix.

Le planificateur de tâches quant à lui permet d'exécuter n'importe quel script ou programme à une heure précise lorsqu'un certain évènement survient grâce à des déclencheurs. Ces déclencheurs peuvent être de plusieurs types comme par exemple effectuer une tâche à l'ouverture d'une session, au démarrage de la machine ou encore à un horaire précis. La solution de déploiement automatique de PPCM utilise ce dernier type de déclencheur.

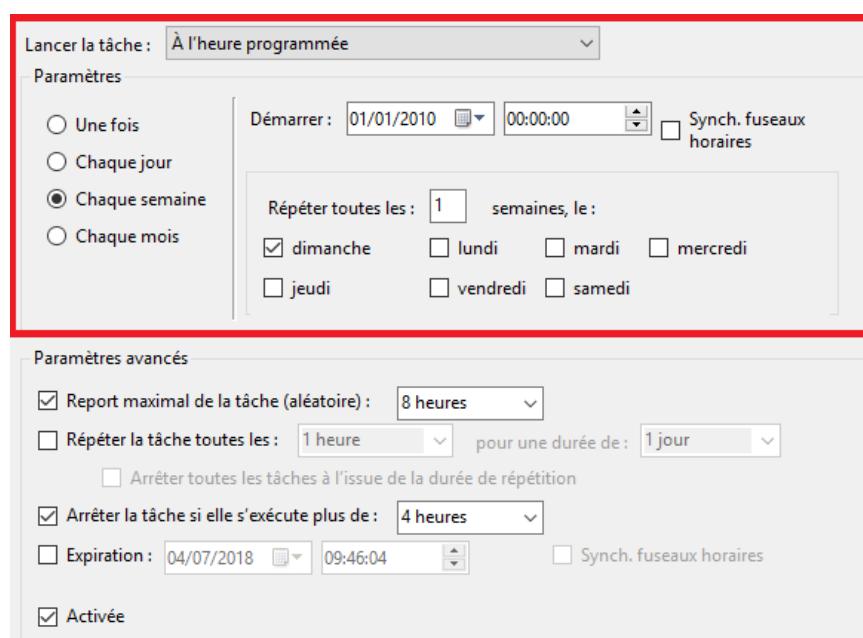


FIGURE 2.31 – Interface de modification du déclencheur d'une tâche dans le planificateur de Windows

Sur l'exemple ci-dessus, la tâche s'exécutera chaque semaine uniquement le dimanche à minuit.

En plus d'utiliser ces scripts, la solution utilise un serveur FTP (File Transfer Protocol) qui permet de transférer des fichiers par Internet ou par le biais d'un réseau local. Toutes les personnes ayant accès à ce serveur, généralement via l'utilisation d'un identifiant et d'un mot de passe, peuvent y envoyer et télécharger des fichiers.

Sur ce serveur, Capgemini doit déposer le paquet (fichier zip) contenant la mise à jour de PPCM et des fichiers XML appelés *déclencheurs* qui permettent d'exécuter la mise à jour sur des serveurs spécifiques. Le nom de ces déclencheurs est le même que celui du serveur sur lequel la mise à jour doit être effectuée et ils ne contiennent que le numéro de version du nouveau logiciel.

```
LFR408988.xml
<Declencheur>
  <NewVersion>6.53.0.0</NewVersion>
</Declencheur>
```

FIGURE 2.32 – Exemple de déclencheur pour le déploiement automatique de PPCM

L'architecture du contenu des déclencheurs est toujours le même : la balise parente *Declencheur* et la balise fille *NewVersion* qui contient la nouvelle version de PPCM à installer.

Pour résumé, l'architecture globale de ce déploiement automatique de PPCM est la suivante :

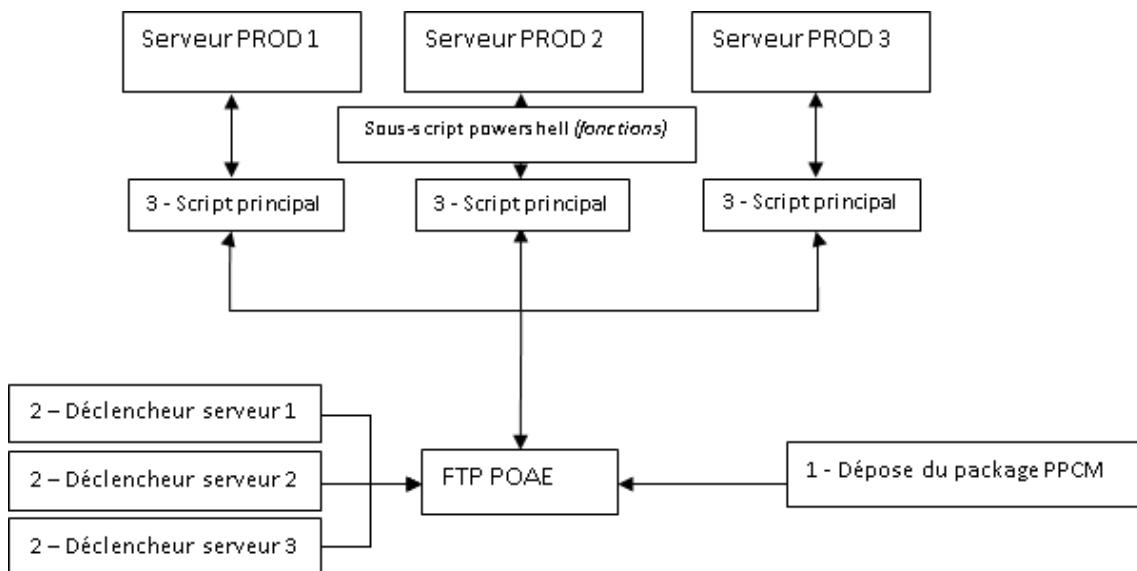


FIGURE 2.33 – Architecture du déploiement automatique de PPCM

Les scripts Powershell doivent être installés, manuellement, sur tous les serveurs où PPCM doit être mis à jour. Le fonctionnement des scripts est divisé en plusieurs étapes :

- Se connecter au serveur FTP de Plastic Omnium
- Vérifier que le déclencheur est installé sur le serveur
- S'il c'est le cas, faire une sauvegarde de l'existant puis télécharger et installer le paquet de mise à jour et enfin supprimer le déclencheur
- Sinon ne rien faire

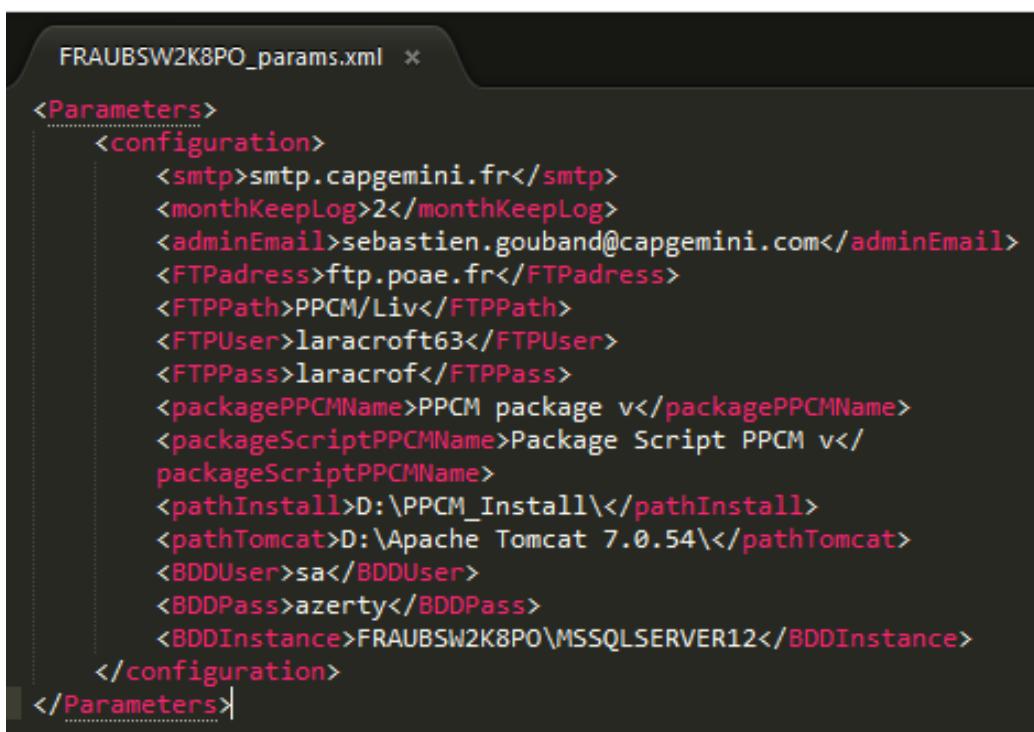
- Envoyer un mail précisant si le déploiement a été effectué avec ou sans erreur

La mise à jour de PPCM est une simple copie des nouveaux fichiers, contenus dans le paquet, à la place des anciens.

De plus, comme nous pouvons le voir sur l'image ci-dessus, il existe un script principal et plusieurs sous-scripts. Ceux-ci contiennent des fonctions spécifiques à une seule partie spécifique du déploiement. Par exemple, il y a un script qui contient les méthodes de connexion et de récupération des fichiers sur le serveur FTP, un autre qui permet d'exécuter les scripts SQL pour modifier la base de données lors de la mise à jour, etc.

Le script principal suit les étapes présentées précédemment et appelle les fonctions correspondantes depuis les autres scripts.

Il est important de préciser que les scripts possèdent un fichier de paramètres contenant toutes les informations nécessaires à leur bon fonctionnement. Ces paramètres sont remplis et modifiés directement par le client en fonction de chaque serveur.



```

FRAUBSW2K8PO_params.xml *

<Parameters>
    <configuration>
        <smtp>smtp.capgemini.fr</smtp>
        <monthKeepLog>2</monthKeepLog>
        <adminEmail>sebastien.goubard@capgemini.com</adminEmail>
        <FTPadress>ftp.poae.fr</FTPadress>
        <FTPPPath>PPCM/Liv</FTPPPath>
        <FTPUser>laracroft63</FTPUser>
        <FTPPass>laracroft</FTPPass>
        <packagePPCMName>PPCM package v</packagePPCMName>
        <packageScriptPPCMName>Package Script PPCM v</
            packageScriptPPCMName>
        <pathInstall>D:\PPCM_Install\</pathInstall>
        <pathTomcat>D:\Apache Tomcat 7.0.54\</pathTomcat>
        <BDUser>sa</BDUser>
        <BDPass>azerty</BDPass>
        <BDInstance>FRAUBSW2K8PO\MSSQLSERVER12</BDInstance>
    </configuration>
</Parameters>

```

FIGURE 2.34 – Exemple de fichier de paramètres pour le déploiement automatique de PPCM

Ce fichier possède toujours la même arborescence et contient notamment les informations de connexion au serveur FTP et à la base de données ainsi que le chemin et le nom du paquet à télécharger sur ce serveur.

Il est également nécessaire de savoir que le même principe a été mis en place pour pouvoir mettre à jour automatiquement les scripts d'installation de PPCM.

2.5.2 Déploiement automatique de SQP

Après m'être familiarisé avec le fonctionnement pour PPCM, j'ai apporté des modifications et intégré la mise à jour de SQP. Les étapes du fonctionnement des scripts restent les mêmes mais avec quelques spécificités notamment au niveau du mode d'installation. Pour PPCM, cette installation est juste une copie de fichiers alors que celle de SQP est l'exécution d'un installateur Windows au format *.msi*.

L'autre grande différence est le fait que SQP fonctionne en mode client / serveur. Cela implique de devoir faire la mise à jour du serveur et si celle-ci s'est effectuée sans erreur, devoir faire celle des clients qui lui sont rattachés. Une solution a été apportée afin d'empêcher la mise à jour des clients si celle du serveur correspondant avait renvoyée une erreur. Pour ce faire, j'ai rajouté dans le déclencheur de la mise à jour des serveurs SQP, la liste de tous les clients qui lui sont rattachés afin de supprimer leur déclencheur si la mise à jour échoue.

```
<Declencheur>
    <NewVersion>5.0</NewVersion>
    <Clients>
        <Client>LFR000001</Client>
        <Client>LFR000002</Client>
        <Client>LFR000003</Client>
        <Client>LFR000004</Client>
        <Client>LFR000005</Client>
    </Clients>
</Declencheur>
```

FIGURE 2.35 – Exemple de fichier déclencheur pour un serveur SQP

De plus, l'installateur Windows (*msi*) doit fonctionner en mode silencieux c'est-à-dire qu'il ne doit demander aucune information à l'utilisateur et effectuer l'installation automatiquement. Pour cela, il a fallu insérer des nouvelles données dans le fichier de paramètres.

```
<Parameters>
    <configuration>
        <smtp>smtp.capgemini.fr</smtp>
        <monthKeepLog>2</monthKeepLog>
        <DaysKeepBackups>15</DaysKeepBackups>
        <adminEmail>gaetan.rubin@capgemini.com</adminEmail>
        <FTPPadress>localhost</FTPPadress>
        <FTPUser>Lingfar</FTPUser>
        <FTPPass>azertyuiop</FTPPass>
        <FTPPath>SQP/Liv</FTPPath>
        <PackageName>SQP package v</PackageName>
        <PathInstall>C:\Users\pruhin\Documents\Install_SQP\</PathInstall>
        <InstallLocation>C:\Program Files (x86)\SQPNET2\</InstallLocation>
        <!-- Logiciels à installer grâce au msi (Manager et/ou Client) -->
        <Features>GeneralFeature,ClientFeature,ManagerFeature,SAPFeature,BDDFeature</Features>
        <BDDUser>sa</BDDUser>
        <BDDPass>azerty</BDDPass>
        <BDDInstance>FRAUBSW2K8P0\MSSQLSERVER12</BDDInstance>
    </configuration>
</Parameters>
```

FIGURE 2.36 – Exemple de fichier de paramètres des scripts pour SQP

Ces paramètres sont :

- Le chemin où la version précédente de SQP est installée
- Les logiciels à mettre à jour (généralement SQP Client ou SQP Manager)

Suite à l'ajout de SQP, l'architecture finale du déploiement automatique de PPCM et SQP est la suivante :

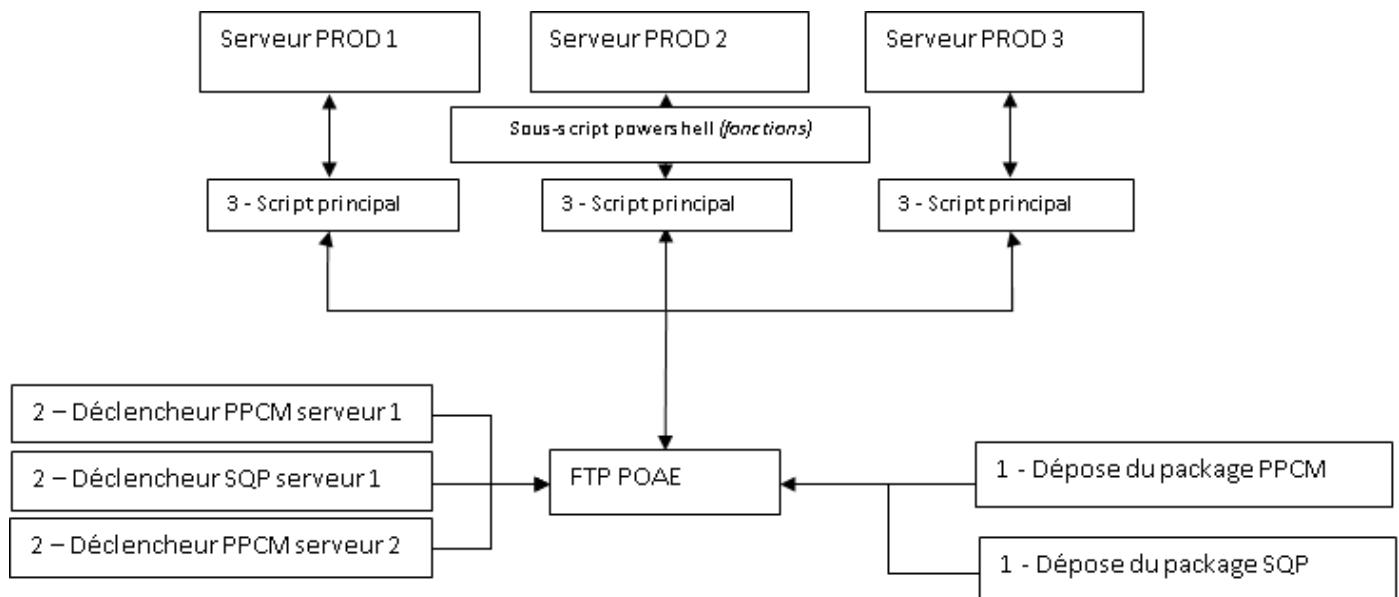


FIGURE 2.37 – Architecture du déploiement automatique de PPCM et SQP

L'architecture est donc quasiment identique, les deux seuls changements sont l'ajout d'un autre paquet d'installation et des déclencheurs pour chaque serveur et client de SQP.

Pour résumé, pour le déploiement automatique de SQP, la solution mise en place utilise le même principe que celle déployée pour un autre logiciel de Plastic Omnium nommé PPCM. Cette solution utilise des scripts Powershell installés sur chaque serveur ou client devant être mise à jour et permettant de télécharger le paquet d'installation depuis un serveur de stockage et de l'exécuter. Ces scripts possèdent également un fichier de paramétrage distinct pour PPCM et pour SQP.

2.6 Planification du stage

Lors de mon arrivée au sein du plateau mutualisé, la première version de l'étude de la nouvelle version de SQP venait d'être terminée. Dans cette étude, les dates de livraison des différents lots étaient précisées.

Mon diagramme de Gantt prévisionnel était donc calqué sur ces différentes dates de livraison :

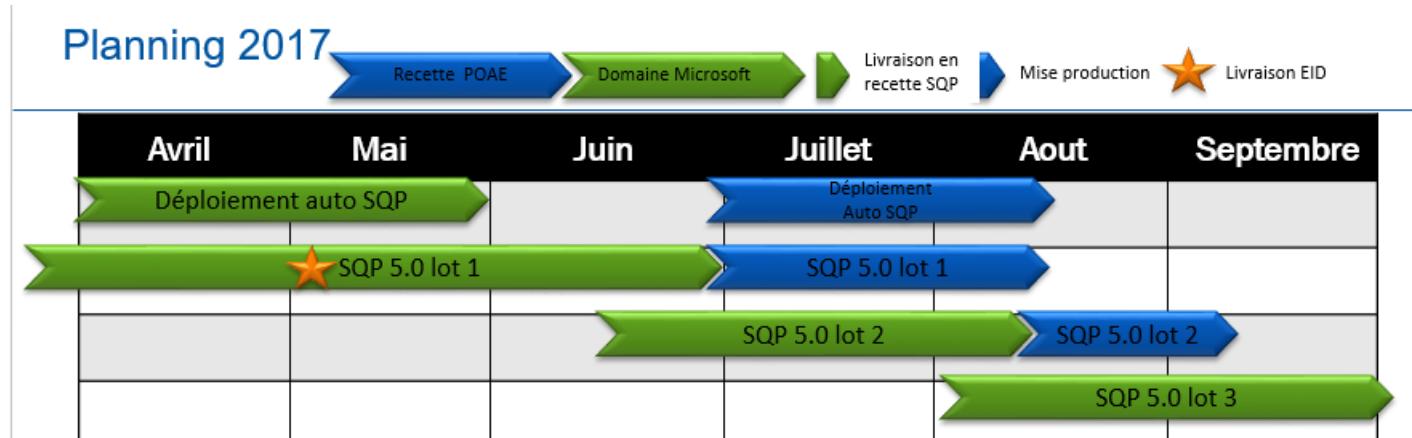


FIGURE 2.38 – Diagramme de Gantt prévisionnel

Toutes les tâches de recette POAE (en bleu) ont été effectuées par le client et sont affichées à titre d'information.

Le planning des livraisons a été retardé de deux semaines suite aux différentes demandes de corrections de bugs de l'ancienne version qui étaient prioritaires aux demandes d'évolutions. Mon diagramme de Gantt réel est donc quasiment identique au prévisionnel, le seul changement est ce décalage de deux semaines :

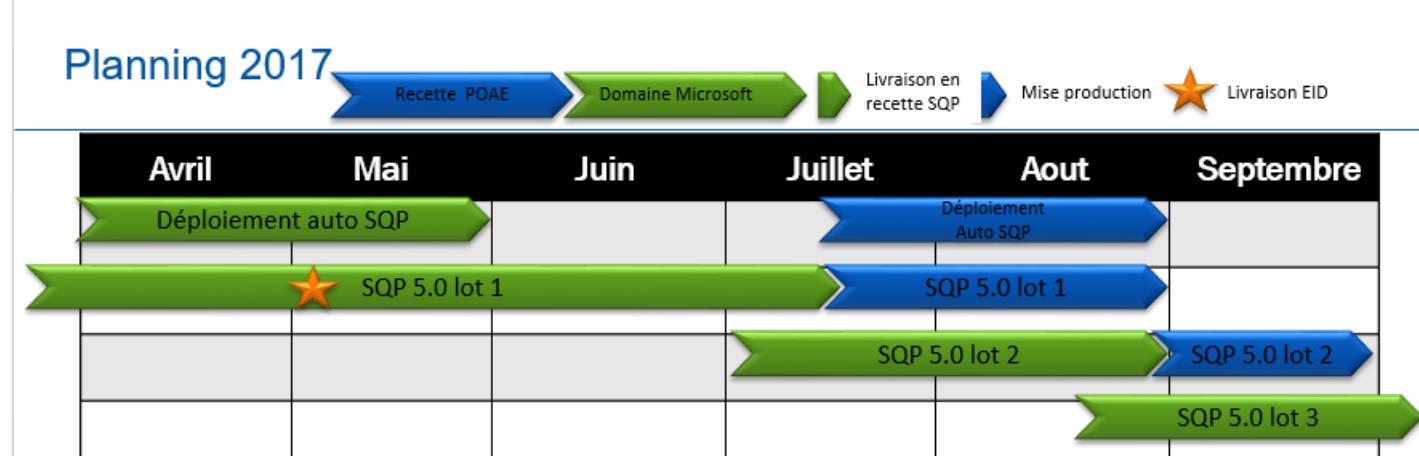


FIGURE 2.39 – Diagramme de Gantt réel

Chapitre 3

Résultats et discussions

Dans cette partie, je présenterai plus en détail les résultats de mon travail ainsi que les problèmes rencontrés et leur solution apportée afin d'arriver à ces résultats. J'évoquerai également les améliorations possibles.

3.1 SQP version 5.0

Lorsque je suis arrivé au sein du plateau mutualisé pour développer la version 5.0 de SQP, personne dans l'équipe ne connaissait en détail l'application. J'ai donc dû me familiariser avec celle-ci grâce à la documentation fournie par Plastic Omnium, à la description du fonctionnement existant dans l'étude préalable et en consultant le code.

Malheureusement, le fait qu'une partie de la documentation ne soit plus à jour et que le code de SQP ne soit pas facile à comprendre rapidement, cette étape de prise de connaissance a pris plus de temps que prévu. J'ai vraiment découvert son fonctionnement lorsque j'ai commencé à développer les nouvelles évolutions.

Malgré cette phase d'apprentissage plus longue que prévu, j'ai malgré tout réussi à livrer le lot 1 dans les délais.

3.1.1 Lot 1

Pour le lot 1, la plus grande difficulté que j'ai rencontrée était la prise en main des différentes applications contenues dans la suite SQP, notamment le fonctionnement du service SQP SAP. En effet, avant d'utiliser les nouvelles dlls, la connexion à SAP depuis les environnements de développement ne fonctionnait pas donc je ne pouvais pas tester ce service.

Un autre problème rencontré avant de commencer les évolutions était de tester le scan du code barre dans SQP Client parce que nous n'avions pas de lecteur fonctionnel.

Une problématique commune à la majorité des évolutions demandées était le manque de précisions définies dans l'étude. Pour la résoudre, j'ai échangé avec Plastic Omnium afin d'avoir les réponses à mes interrogations.

Hormis ce manque d'explications et cette difficulté à prendre en main SQP, je n'ai eu que deux problèmes assez importants lors du développement de ce lot 1.

3.1.1.1 Fonction supervision sous forme de service Windows

Comme je l'ai expliqué dans la partie précédente, j'ai dupliqué le code de la partie supervision de SQP Client pour l'intégrer dans un nouveau service Windows.

Cette duplication m'a posé problème pour intégrer de nouvelles évolutions. En effet, il m'est arrivé d'avoir oublié de reporter une modification dans le client lourd ou dans le service.

Afin de ne plus rencontrer ce problème de duplication et comme j'étais en avance sur la date de livraison, j'ai décidé de créer une nouvelle bibliothèque nommée *LibClient.dll* afin de regrouper tout le code identique à ces deux applications.

L'architecture finale du client lourd SQP Client et du service Windows SQP Supervision est donc la suivante :

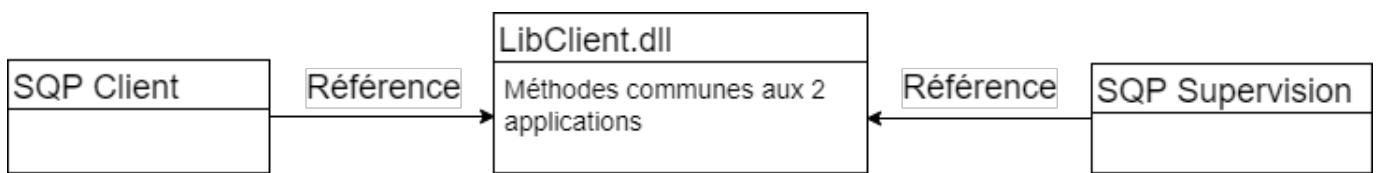


FIGURE 3.1 – Architecture finale pour SQP Client et Supervision

3.1.1.2 Déclaration de type Essai

Le problème que j'ai rencontré pour implémenter cette évolution a notamment été de comprendre exactement ce que Plastic Omnium demandait. En effet, cette évolution touche les trois logiciels les plus importants de SQP : le client, le manager et le service SQP SAP.

Afin de comprendre le fonctionnement exact, nous avons organisé une réunion avec le client pour qu'il nous apporte des précisions.

Pour cette évolution, j'ai également rencontré également une autre difficulté lors de l'envoi ou non des transactions à SAP en fonction de leur type (essai ou non). En effet, les données que nous avions dans notre environnement de test ne correspondaient à celle dans le SAP de test ce qui avait pour cause l'échec de tous les envois de transactions.

3.1.1.3 Résultats

Après avoir fait une recette interne à Capgemini, les quelques retours ont été pris en compte mais le client n'ayant pas encore fait sa propre recette, je ne sais pas si des nouveaux bugs vont être trouvés ou si l'implémentation des évolutions est en adéquation avec ses demandes.

3.1.2 Lot 2

Une des difficultés rencontrées pour ce lot 2, a été l'arrivée du nouveau stagiaire. En effet, comme j'étais la personne la plus expérimentée du plateau mutualisé à propos de SQP, j'ai dû lui transmettre mes connaissances et l'aider en même temps que je développais. Il en a résulté que je devais déterminer quelles tâches intéressantes et faisables je pouvais lui déléguer afin qu'il puisse se familiariser rapidement au fonctionnement difficile de SQP.

Comme pour le premier lot, le manque de précisions pour la majorité des évolutions demandées a nécessité beaucoup d'échanges avec le client.

Mais contrairement au lot précédent, il n'y a pas eu de problème important rencontré pour ces évolutions.

3.1.2.1 Résultats

Comme pour le lot 1, la recette interne a été effectuée et les quelques retours ont été modifiés mais le client ne nous a pas encore fourni les conclusions de sa phase de test.

3.1.3 Lot 3

Comme je l'ai présenté précédemment, je n'ai développé qu'une seule évolution sur trois pour ce lot.

Le seul gros problème que j'ai rencontré lors de la refonte de la base de données locale de SQP Client et Supervision a été de décomposer ces 2 applications en plusieurs couches : la couche d'accès aux données et la couche métier.

Cette décomposition aurait pu être plus compliquée à réaliser si je n'avais pas déjà créé la bibliothèque *LibClient.dll* regroupant toutes les méthodes communes à ces deux applications.

Je n'ai pas vraiment eu de gros problèmes à convertir les requêtes SQL en LINQ parce que je connaissais déjà bien ce type de requêtes notamment grâce à mon projet de deuxième année à l'ISIMA.

3.1.3.1 Résultats

Comme pour les deux premiers lots, nous sommes en attente des retours de la recette du client.

3.2 Modifications pour la version 4.5

Pour toutes les modifications que j'ai dû apportées à la version 4.5 que ce soit les corrections de bugs ou le changement des dlls, la difficulté principale a été de développer en Visual Basic.NET. Ce langage est proche du C# mais les notations sont suffisamment différentes pour se perdre facilement lorsque l'on ne le maîtrise pas totalement.

3.2.1 Corrections de bugs

Toutes les demandes de corrections de bugs venant de l'instabilité du réseau ont été très difficiles à simuler dans notre environnement de test. Pour cela, j'ai dû créer, volontairement, une requête bloquante. Mais elles étaient faciles à corriger une fois cette simulation réussie. En effet, la requête pour supprimer les requêtes bloquantes est facilement trouvable sur Internet.

La demande de rectification de la fréquence d'import des données depuis SAP a été facile à simuler parce que la situation entraînant l'erreur remontée par Plastic Omnium était également présente dans notre environnement de test. De plus, la correction de ce problème était déjà présente dans le code mais elle était simplement commentée.

3.2.2 Changement des dlls de connexion à SAP

Le changement des dlls a été assez facile à réaliser car l'appel des différentes méthodes d'envoi ou d'import des données depuis SAP étaient quasiment identiques. L'étape la plus compliquée a été de réussir à se connecter à un SAP de test depuis la machine virtuelle sur laquelle je réalisais mes développements. Pour cela, j'ai dû créer un ticket au support de Capgemini afin qu'il ouvre un certain port de la machine.

3.2.3 Résultats

Les corrections de tous les bugs ainsi que le changement des dlls ont été validés par le client.

3.3 Discussions et perspectives

Le fonctionnement de la suite SQP est difficile à comprendre parce qu'elle est composée de beaucoup de logiciels différents et parce que la documentation est incomplète et non mise à jour.

De plus, le code de tous les logiciels n'est pas du tout structuré ou organisé donc il est difficile de s'y retrouver et de tout comprendre lorsque l'on ne connaît pas leur fonctionnement. Il existe également de nombreuses duplications de codes entre ces différents logiciels qui réduisent leur maintenabilité.

L'amélioration que nous préconisons est de repartir de zéro et redévelopper la suite SQP suivant une architecture dite 4 tiers.

Cette architecture est composée de 4 couches différentes :

- La couche de présentation : elle contient toutes les interfaces de l'application. Il faudrait une couche de présentation par logiciel de la suite SQP.
- La couche métier : elle contient toutes les classes communes à tous les logiciels. Il faut donc étendre celle créée lors de la refonte de la base de données locale de SQP Client à tous les autres logiciels.
- La couche d'accès aux données : elle contient toutes les méthodes permettant de récupérer les données depuis la base de données. Comme pour la couche métier, il faut étendre celle déjà créée à tous les autres logiciels.
- La couche business : elle permet de récupérer les données depuis la couche d'accès aux données, de les manipuler et ensuite de les envoyer à la couche de présentation pour les afficher.

Cela prendrait énormément de temps à effectuer mais une fois que ce serait terminé, la maintenabilité des différents logiciels seraient largement supérieures à l'actuelle et cela permettrait donc de gagner du temps sur les futures évolutions.

Conclusion

L'objectif de ce stage était de faire la tierce maintenance applicative d'une suite de logiciels nommée SQP fonctionnant en mode client / serveur et notamment d'implémenter 25 nouvelles évolutions demandées par le client, Plastic Omnium. Ces évolutions allaient de la correction des traductions dans la base de données à la migration de la base de données des clients en fichiers XML.

Suite à la phase de recette interne à Capgemini, les objectifs ont été remplis et les résultats correspondent aux attentes mais nous n'avons pas encore reçu les conclusions de la recette de Plastic Omnium.

Grâce à ce stage et surtout dans le cadre de la TMA, je me suis rendu compte que la communication avec le client est très importante et qu'elle est souvent difficile par mail notamment pour des problèmes techniques. Pour résoudre ces problèmes de communication, j'ai appris qu'il ne faut pas hésiter à faire des points téléphoniques directement avec le client parce qu'il est plus facile d'échanger de vive voix sur des sujets compliqués.

Les différents objectifs de ce stage ont été atteints, sous réserve des retours du client. Un point d'amélioration pour la suite SQP est de la redévelopper entièrement avec une architecture facilitant sa maintenabilité comme, par exemple, l'architecture 4-tiers.

Glossaire

Architecture logicielle : L'architecture logicielle décrit d'une manière symbolique et schématique les différents éléments d'un ou de plusieurs systèmes informatiques, leurs interrelations et leurs interactions. Contrairement aux spécifications produites par l'analyse fonctionnelle, le modèle d'architecture, produit lors de la phase de conception, ne décrit pas ce que doit réaliser un système informatique mais plutôt comment il doit être conçu de manière à répondre aux spécifications. L'analyse décrit le « quoi faire » alors que l'architecture décrit le « comment le faire ».

Bibliothèque logicielle (dll) : En informatique, une bibliothèque logicielle est une collection de fonctions, qui peuvent être déjà compilées et prêtées à être utilisées par des programmes.

Classe : En programmation orientée objet, une classe déclare des propriétés communes à un ensemble d'objets. La classe déclare, d'une part, des attributs représentant l'état des objets et, d'autre part, des méthodes représentant leur comportement. Une classe représente donc une catégorie d'objets. Elle apparaît aussi comme un moule ou une usine à partir de laquelle il est possible de créer des objets ; c'est en quelque sorte une « boîte à outils » qui permet de fabriquer un objet.

ERP : Un ERP (Enterprise Resource Planning) ou également appelé PGI (Progiciel de Gestion Intégré) est un système d'information qui permet de gérer et suivre au quotidien, l'ensemble des informations et des services opérationnels d'une entreprise. En cas d'impact d'un module, l'information est mise à jour en temps réel dans l'ensemble des autres modules associés. C'est un système qui garantie la piste d'audit : il est facile de retrouver et d'analyser l'origine de chaque information. Il peut couvrir l'ensemble du Système d'Information (SI) de l'entreprise (sauf si l'entreprise ne choisit dans un premier temps d'implémenter que certains modules de l'ERP). Il garantit l'unicité des informations qu'il contient, puisqu'il n'a qu'une seule base de données au sens logique.

Framework : Un framework est un ensemble d'outils et de composants logiciels organisés conformément à un plan d'architecture et des patrons de conception, l'ensemble formant ou promouvant un « squelette » de programme. Il est souvent fourni sous la forme d'une bibliothèque logicielle. Un framework est conçu en vue d'aider les programmeurs dans leur travail. L'organisation du framework vise la productivité maximale du programmeur qui va l'utiliser — gage de baisse des coûts de construction et maintenance du programme.

Forge (logicielle) : En informatique, une forge est un système de gestion de développement collaboratif de logiciel. Son objectif est de permettre à plusieurs développeurs de participer ensemble au développement d'un ou plusieurs logiciels.

Patron de conception : En informatique, et plus particulièrement en développement logiciel, un patron de conception (souvent appelé design pattern) est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels.

PowerShell : PowerShell est un langage de script développé par Microsoft qui est fondé sur la programmation orientée objet. Ce langage permet d'effectuer des actions au niveau du système d'exploitation comme par exemple créer, modifier ou supprimer un fichier.

Tierce maintenance applicative (TMA) : La tierce maintenance applicative est la maintenance appliquée à un logiciel (« applicative ») et assurée par un prestataire externe dans le domaine des technologies de l'information et de la communication.

XML : L'XML (Extensible Markup Language) permet de représenter de l'information structurée. Un document XML a pour fonction de représenter de l'information accompagnée de balises permettant d'en interpréter les divers éléments, sans aucune restriction sur le nombre de celles-ci.

Webographie

[Stack Overflow] <https://stackoverflow.com>, (Date de consultation tout le temps)

[MSDN] <https://msdn.microsoft.com>, (Date de consultation tout le temps)

[Wikipedia] <https://fr.wikipedia.org>, (Date de consultation juillet / août 2017)

[Capgemini] <https://www.fr.capgemini.com>, (Date de consultation juillet 2017)

[SAP Support] <https://support.sap.com>, (Date de consultation mai 2017)