UNIVERSITÉ SIDI MOHAMMED BEN ABDELLAH
FACULTÉ DES SCIENCES ET TECHNIQUES FES
DÉPARTEMENT D'INFORMATIQUE

Projet de Fin d'Études

Licence Sciences et Techniques Génie Informatique

Adaptation de SugarCRM et développements de composants personnalisés

---

Lieu de stage : Nevo-Technologies, Rabat, Maroc

Réalisé par :
Moncef BAAZET
< mob.ajm@gmail.com >

Encadré par :
Pr. A. ZARGHILI
Pr. A. MAJDA

Soutenu le 15/06/2012 devant le jury composé de :

Pr. L. LAMRINI
Pr. I. CHAKER
Pr. A. ZARGHILI

Année Universitaire 2011-2012

Table des matières

Liste des abréviations

CE	Community Edition
COM	Component Object Model
CRM	Customer Relationship Management
CSV	Comma-Separated Values
DCOM	Distributed Component Object Model
GRC	Gestion de la relation client - Traduction controversée de l'anglais Customer Relashionship Management
HTML	HyperText Markup Language
IDN	Intuit Developer Network
IIS	Internet Informations Services
MS	Microsoft
MVC	Model-View-Controller
OSR	On-Screen Reference
PHP	Hypertext PreProcessor
QBFC	QuickBooks Foundation Classes
QBWC	QuickBooks Web Connector
QBXML	QuickBooks eXtensible Markup Language
QWC	QuickBooks Web Connector
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
url	Uniform Resource Locator
XML	eXtensible Markup Language
XP	eXtreme Programming

1. Remerciements

Je dédie ce rapport et ce travail

À mes très chers parents, mes chères soeurs, ma chère grand-mère et à la mémoire de mes défunts grand-parents ; l'amour, le soutien et la confiance que vous n'avez cessé de me prodiguer sont ce qui me fait et me nourrit aujourd'hui, aucun mot ne saurait exprimer ma gratitude.

À l'ensemble de ma famille, pour qui j'ai un très grand respect.

À tous mes amis. Je vous souhaite une vie pleine de bonheur et de réussite. Pour tous les instants inoubliables que j'ai passé avec vous, je vous remercie beaucoup.

À tous mes professeurs. Pour tout le travail fournit et pour avoir toujours essayé de nous pousser vers l'avant.

À tous ceux qui me sont chers.

2. Introduction générale

Ce rapport de stage présente le travail que j'ai accomplis dans le cadre de mon stage de fin de licence au sein de la société Nevo-Technologies.

Le travail effectué a consisté en le développement de composants personnalisés pour le logiciel de gestion de relation client SugarCRM, ainsi que l'adaptation et la modification des composants déjà existants pour mieux convenir au flot de travail quotidien de l'équipe de Nevo-Technologies

3. Organisme d'accueil

Nevo-Technologies est un distributeur à valeur ajouté de solutions technologiques spécialisées. L'entreprise est basé aux États-unis, à Princeton, New Jersey, ainsi qu'au Maroc, à Rabat.

3.1. Mission et Philosophie

La mission de l'entreprise est de fournir une précisieuse passerelle entre les sociétés innovatrices sur le plan technologique et le marché nord-africain. Pour cela, Nevo-Technologies travaille en étroite collaboration avec les revendeurs et les développeurs de nouveaux produits susceptibles de générer de nouveaux marchés.

La philosophie de Nevo-Technologies est qu'une distribution de qualité ne se résume pas simplement à un rôle d'intermédiaire passif qui fournit des produits. Nevo-Technologies s'implique en tant que partenaire stratégique dans la croissance des entreprises des revendeurs, en offrant un service clientèle irréprochable, des livraisons rapides et des prix compétitifs, en plus des produits et technologies de pointe.

3.2. Services

Nevo-Technologies fournit également entre autres services :

– Les services *marketing* et *lead generation*, qui permettent de générer des opportunités pour le réseau de revendeurs en initiant les clients finaux aux produits de Nevo-Technologies. Les formation.

– Les formations et certifications pour les partenaires ainsi que les utilisateurs finaux. Toutes les formations sont délivrées par des experts certifiés et sont personnalisables pour répondre à des besoins spécifiques.

– L'organisation d'évènements pour rencontrer les anciens clients et en attirer de nouveaux. Nevo-Technologies peut fournir le matériel aux conférenciers et formateurs.

3.3. Solutions

Nevo-Technologies se spécialise dans la distribution de solutions de :

Connectivité Gestion de bande passante, *Load Balancing*, Point d'accès sans fil, *Monitoring* de réseau, etc.

Sécurité *Unified Threat Management*, Anti-Phishing, Anti-Virus, SSL-VPN, *Email Content Filtering*, etc.

Stockage et Sauvegarde *Network Attached Storage*, *Direct Attached Storage*, *Storage Area Network*, Technologie *Beyong-Raid*, etc.

Productivité *Link Balancing*, Contrôler qui fait quoi, Contrôle et archivage de message instantée, Identification du traffic par utilisateur, etc.

3.4. Distinctions

The American Chamber of Commerce in Morocco a accordé à Nevo-Technologies le prix du meilleur importateur de produits innovants au Maroc. En un an d'existence, Nevo-Technologies a introduit pas moins de 8 nouvelles marques américaines sur le marché marocain, parmi lesquelles : *Drobo*, *Cyberoam*, *Barracuda* ou encore *Patton*.

4. Sujet de stage et problèmatique

Pour proposer les meilleurs solutions et fournir le meilleur service à ses clients, Nevo-Technologies enregistre et gère méticuleusement les informations les concernant. La quantité d'informations gérée devenant assez grande a poussé l'équipe de Nevo-Technologies à chercher une solution facilitant ce travail, pour éviter les pertes et les données non à jour, et les désagréments que cela peut causer.

L'équipe a donc choisit d'utiliser une solution logicielle de gestion cliente, et a adopté SugarCRM. SugarCRM est un CRM [1] généraliste qui facilite énormément de tâches et permet de centraliser la gestion des données concernant les clients, les opportunités, les campagnes marketing, etc.

Mais les besoins de Nevo-Technologies, pour le travail quotidien de l'équipe, sont plus poussés et nécessitent d'adapter le CRM, en ajoutant des fonctionnalités.

Dans le cadre des études à la FST, tout étudiant doit effectuer un stage d'une période de deux mois à la fin de son année de licence, qui permet de découvrir le monde de l'entreprise et du travail, de mettre en pratique ses études et d'être prêt à affronter la vie active.

J'ai donc décidé, avec l'équipe de Nevo-Technologies, de consacrer mon stage à l'adaptation de SugarCRM aux besoins de l'entreprise, pour permettre à l'équipe d'effectuer son travail dans les meilleurs conditions et d'offrir un service encore meilleur.

1. Customer Relationship Management

Première partie .

Expression des besoins et méthodologie

Cette partie présente le cahier des charges détaillé ainsi que la méthodologie de travail adoptée et le planning de développement établit.

5. Le cahier des charges

Le travail d'adaptation et de personnalisation de SugarCRM pour correspondre aux besoins de Nevo-Technologies peut être catégorisé en trois types :

1. La création de nouveaux modules, représentants de nouvelles entités, et permettant d'étendre les types d'informations gérés par le CRM et de mieux centraliser les données. Les nouvelles entités à introduire ici sont : le client final, le devis et le produit en stock.
2. La modification des modules existants, pour y ajouter certaines caractéristiques et propriétés permettants de mieux cerner les entités en jeu et d'avoir un meilleur modèle relationnel entre les données.
3. L'installation de logiciels externes pour récupérer les données existantes ou bien synchroniser SugarCRM avec les différents logiciels utilisés.

Après quelques réunions avec l'équipe de Nevo-Technologies pour recueillir leurs remarques et besoins, nous présentons ici le cahier des charges qui permet de résumer, préciser et servir de guide aux développements à effectuer pour adapter SugarCRM au flot de travail de l'équipe Nevo-Technologies et ainsi fluidifier le travail et tirer le meilleur de cette numérisation de la gestion de la clientèle.

5.1. Gestion des clients finaux

5.1.1. Objectifs

Le rôle de Nevo-Technologies dans la chaîne de distribution (fig. 4.1) d'un produit est celui de **distributeur**. Ce rôle consiste à se placer entre le constructeur et le revendeur, pour faciliter la distribution de solutions au premier et l'acquisition au second.

Nevo-Technologies n'interagit donc pas directement avec l'utilisateur final d'une solution technologique. Néanmoins, dans le cadre de la démarche qualité et d'assistance aux clients de Nevo-Technologies, le contact avec le client est gardé pour l'informer de mises à jour, de solutions plus performantes ou l'assister en cas de problème.

Il est aussi intéressant de séparer les revendeurs des clients finaux pour pouvoir faire des campagnes d'informations ou de marketing mieux ciblées, et de mieux contrôler l'information envoyée.

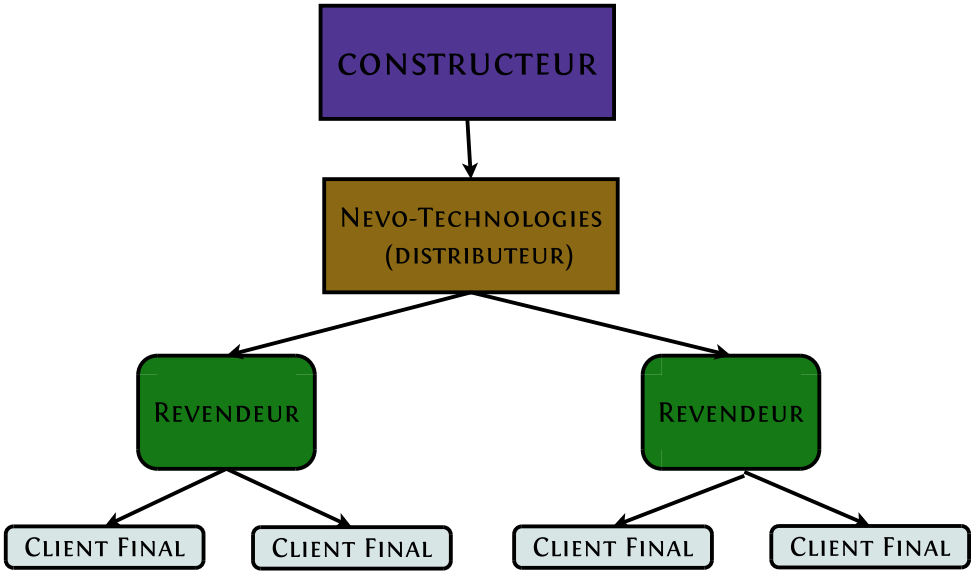


FIGURE 5.1.: La chaîne de distribution des solutions proposées par Nevo-Technologies

SugarCRM, étant un CRM généraliste, ne fait pas cette distinction entre **client** (revendeur) et **client final**. Le concept de base utilisé par SugarCRM pour gérer les clients est celui de **compte**. À Nevo-Technologies, celui-ci est utilisé pour représenter les revendeurs.

Ces raisons ont donc poussé à la création d'un nouveau module permettant d'enregistrer et de toujours avoir à disposition les informations de contact d'un client final.

5.1.2. Règles de gestion

- Un **client final** est représenté par un nom, un téléphone, une ville, un pays.
- Un **client final** peut être ajouté ou consulté par un utilisateur du CRM.
- Un **client final** peut être modifié ou supprimé par : l'administrateur uniquement, l'administrateur et le créateur du compte final, ou par tout le monde. L'administrateur s'occupe de règler les permissions nécessaires.
- Un *contact* (voir 8.2) peut être associé à un **client final**.
- Un **client final** peut être associé à une *opportunité* (voir 8.2) ou un *lead* (voir 8.2).

5.2. Consultation des devis

5.2.1. Objectif

Un CRM, bien qu'indispensable au bon fonctionnement de la majorité des entreprises aujourd'hui, n'est pas suffisant, puisqu'il ne permet de gérer que les informations clientes. Il ne fournit pas par exemple d'interface de création et de gestion de

devis ou de bons de commandes. L'un des principaux composants manquants est celui de gestion de la comptabilité de l'entreprise.

À Nevo-Technologies, la solution utilisée — bien avant l'adoption d'un CRM — est QuickBooks. (voir 7) C'est un logiciel très puissant, permettant de gérer tous les aspects relevants de la comptabilité.

Il est néanmoins très frustrant de devoir jongler entre SugarCRM et QuickBooks constamment pour consulter les devis d'un client. C'est ce qui a poussé au développement d'un module représentant les devis et permettant de les consulter.

5.2.2. Règles de gestion

- Un **devis** est représenté par un numéro de référence, un montant total et une liste de *produits*.
- Chaque *produit* de la liste est représenté par une quantité, une désignation, une référence, un prix unitaire hors taxe, un prix total hors taxe et un prix total TTC.
- Un **devis** peut être associé à un *compte*. (voir 8.2)
- L'utilisateur peut récupérer les **devis** à partir de QuickBooks.
- Les **devis** doivent être synchronisés avec QuickBooks périodiquement et automatiquement.

5.3. Gestion des produits en stock

5.3.1. Objectif

Pour un distributeur de solutions technologiques, connaitre à tout moment la disponibilité des produits en stock est indispensable au bon fonctionnement de l'entreprise. Les contre-performances et pertes que peut engendrer une mauvaise gestion du stock doivent être évitées à tout prix.

Il est également très important de tracker le produit. Savoir s'il a été vendu ou prêté pour une démonstration, à qui, quand et par qui, sont autant d'informations cruciales dont il est indispensable de garder trace.

C'est dans cette optique qu'à Nevo-Technologies, on a décidé de créer un module qui permettrait de gérer le stock de produits disponible.

5.3.2. Règles de gestion

- Un **produit** est caractérisé par un nom, un numéro de constructeur, une référence, un numéro de série, les différentes quantités dans les différents stocks de la société, un statut, une date de vente ou de près, une date d'expiration de licence et une date de fin de près.
- Un **produit** peut être consulté, ajouté, supprimé ou modifié par un utilisateur.
- Un nombre arbitraire d'exemplaires d'un **produit** peuvent être vendus ou prêtés pour démonstration par un utilisateur à partir d'un des stocks.
- Si la date d'expiration de licence est renseigné, SugarCRM doit informer l'utilisateur et envoyer un email au *client final*. (voir 4.1)
- Si la date de fin de près est renseigné, SugarCRM doit informer l'utilisateur.
- Un *contact* peut être associé à un **produit**. Il représente le contact de Nevo-Technologies chez le constructeur.
- Les **produits** peuvent être filtrés par : la quantité disponible, la référence ou le constructeur.
- Les **produits** peuvent être récupérés de QuickBooks, d'une façon automatique et périodique, ou bien manuelle par l'utilisateur qui déclenche la procédure.

5.4. Synchronisation de comptes

5.4.1. Objectif

Nevo-Technologies a accumulé au fil de son existence une clientèle riche et bien fournie. Avant l'adoption de SugarCRM, le système utilisé pour gérer les informations relatives à cette clientèle était QuickBooks.

Pour assurer le passage à SugarCRM sans pertes de données, on a créé la possibilité de synchroniser les comptes clients entre QuickBooks et SugarCRM. Ainsi les incohérences de données et les pertes ne sont plus à craindre, et l'utilisation du CRM peut se faire en toute sérénité.

5.4.2. Règles de gestion

- L'utilisateur peut récuperer les **comptes** à partir de QuickBooks.
- Les **comptes** sont synchronisés automatiquement et périodiquement.

5.5. Catégorisation des contacts

5.5.1. Objectif

Les contacts (voir 8.2) de l'équipe Nevo-Technologies, gérés par SugarCRM, doivent pouvoir être catégorisés, c'est à dire leur assigner une catégorie, qui représente leur poste dans l'entreprise cliente. Quelques exemples sont : Directeur, Comptable, Responsable avant-vente, etc.

Cette catégorisation permet à chaque membre de l'équipe Nevo-Technologies de communiquer directement avec la personne qui s'occupe du dossier ou sujet à discuter dans l'entreprise cible. Les problèmes de comptabilité ne seront pas discuté avec le service commercial par exemple.

5.5.2. Règles de gestion

– Chaque **contact** peut se voir assigné une catégorie parmi une liste décidée à l'avance.

5.6. Catégorisation des leads

5.6.1. Objectif

Les *leads* (voir 8.2) dont on prend connaissance n'ont pas la même valeur. Parfois un *lead* est clairement intéressant et a beaucoup de potentiel. Pour d'autres il est clair que ça ne vaut pas grand chose, et que sauf exceptionnellement, le *lead* ne mènera nulle part.

Dans le monde du business ces différences ont des noms : *hot lead*, *warm lead* et *cold lead*. Cette catégorisation a été adoptée par Nevo-Technologies et on a voulu l'implémenter dans le module de gestion des *leads* déjà présent.

5.6.2. Règles de gestion

– Un **lead** peut être *hot*, *warm* ou *cold*. L'utilisateur choisit la catégorie qui convient.

5.7. Liaisons inter-modules

5.7.1. Objectif

SugarCRM offre la possibilité de lier les différentes entités manipulées entre elles. Un contact (voir 8.2) peut par exemple être associé à un compte (voir 8.2) ou à une

opportunité (voir 8.2).

Il devient ainsi possible de voir et consulter toutes les informations associées à une entité sur une seule page, et de déclencher des évènements spéciaux reliés à ces associations. Il est par exemple possible d'envoyer un email à un contact associé à un rendez-vous lorsque la date de ce dernier approche.

Certains des modules existants n'ont pas toutes les associations nécessaires au bon déroulement du travail de l'équipe Nevo-Technologies. Une opportunité par exemple devrait pouvoir être reliée à un client final (voir 4.1) pour convenir au flot de travail à Nevo-Technologies. Ce manque d'information ne peut être toléré, et c'est pour cela que nous avons procédé à l'ajout de ces associations.

5.7.2. Règles de gestion

- Un **client final** peut être associé à une ou plusieurs **opportunités**.
- Un **contact** peut être associé à un ou plusieurs **client finaux**.
- Un **client final** peut être associé à un ou plusieurs **contacts**.
- Un **client final** peut être associé à un ou plusieurs **leads**.
- Un **devis** peut être associé à un **compte**.

6. La conduite du projet

6.1. Choix du processus de développement

Le choix du processus de développement est déterminant dans la réussite d'un projet. Il cadre ses différentes phases et caractérise les principaux traits de sa conduite. Le choix d'une méthode de développement, qui soit adéquate et adaptée aux besoins et particularités du projet, doit donc être fait au préalable afin d'obtenir un produit de qualité et répondant aux exigences des utilisateurs dans **des temps et coûts prévisibles**.

L'objectif ultime est de **livrer une application fonctionnelle**, dans des **délais réduits** avec une (très) petite équipe de développement.

En analysant les besoins, les contraintes et les disponibilités de l'équipe de Nevo-Technologies, il s'avère que le choix d'une méthode de **développement agile** est le plus pertinent. Notre choix s'est porté sur la méthode XP [1], l'une des méthodes agiles les plus reconnues et utilisées à travers le monde.

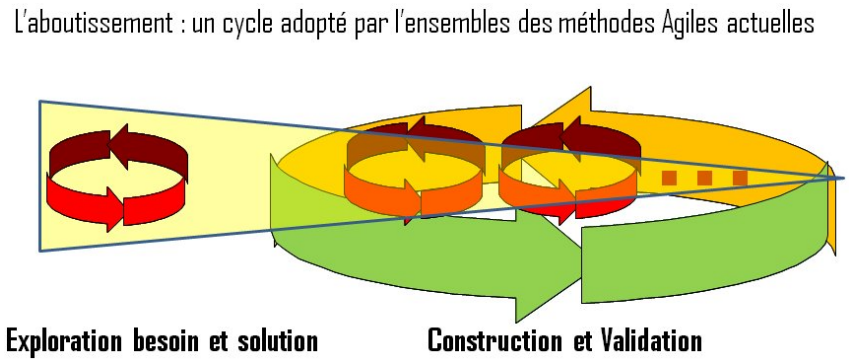


FIGURE 6.1.: Cycle des méthodes agiles

6.2. eXtreme Programming

L'eXtreme Programming est une méthode de développement agile qui met l'accent sur l'aspect réalisation de l'application, sans pour autant négliger l'aspect gestion de

1. eXtreme Programming

projet. Elle est adaptée aux équipes réduites, avec des besoins changeants continuellement. C'est une méthode **itérative**, **incrémentale** et **adaptative** :

Itérative Cette méthode consiste en plusieurs petits cycles complets de vie, que l'on appelle des itérations. Chaque itération reprend les phases d'analyse, conception, implémentation et tests pour faire avancer le projet et produire un résultat livrable.

Incrémentale À chaque fin d'itération, le produit livrable est appelé un incrément. Il vient se greffer sur les réalisations passées pour produire un logiciel encore plus complet, jusqu'à atteindre la version finale désirée par le client.

Adaptative XP permet de s'adapter aux changements dans les spécifications ou de prendre en considération les nouveaux besoins du client à tout moment du processus du développement.

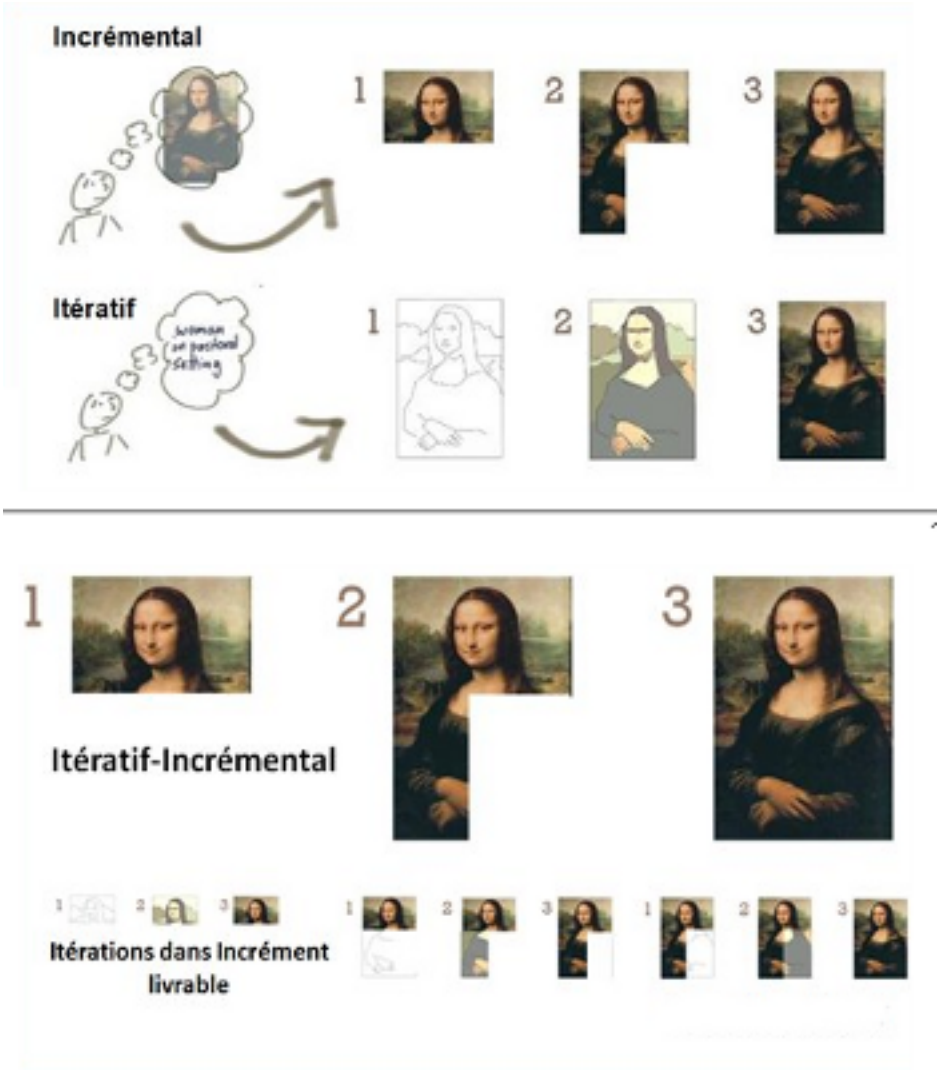


FIGURE 6.2.: Processus itératif, incrémental, et itératif-incrémental

Cette méthode vise principalement à réconcilier développement et productivité, en rendant le projet plus flexible et ouvert au changement. Le principal moyen employé pour atteindre ce but est de s'appuyer sur des principes et des pratiques qui existent dans l'industrie du développement logiciel depuis des décennies, mais qui sont poussés à l'extrême par l'XP. Ces principes sont **les tests**, **la revue du code**, **la simplicité**, **la compréhension**, **la communication**, **l'intégration continuelle** et **l'évolutivité**.

Enfin, et c'est peut-être le point le plus important, XP remet **le client au centre du processus** de développement, et met en avant le travail d'équipe. Le client, les utilisateurs et l'équipe de développement font tous partie d'un même **effort collaboratif** dont le but ultime est un **logiciel fonctionnel** et **répondant aux besoins exigés**. L'équipe de développement est constamment à l'écoute du *feedback* fournit par le client et les requêtes présentées par celui-ci pour affiner, améliorer et faire avancer le projet.

Les principaux avantages de cette organisation du développement sont :

- Le client est constamment **informé sur l'état d'avancement** du projet.
- La mise en production intervient très tôt dans la durée de vie du projet, ce qui permet au client de tirer **rapidement des bénéfices**.
- Le logiciel, après mise en production, sert de support pour la définition et le choix des fonctionnalités à implémenter au cours des prochaines itérations. Il devient possible de **détecter et corriger** très rapidement les erreurs de conception ou d'interprétation et **d'orienter le développement** en fonction du retour d'expérience des utilisateurs.
- La **priorité d'implémentation** des fonctionnalités n'est pas guidée par les contraintes techniques mais par leur **valeur métier**. Les efforts de l'équipe de développement sont focalisés sur les fonctionnalités métiers les plus importantes dès le début du projet.

6.3. Méthodologie de développement

6.3.1. Phases du projet

Les différentes phases de développement du projet sont :

Étude préalable

Cette phase est focalisée sur **l'étude du contexte** et **l'analyse du métier** de l'entreprise, la **collecte d'informations** relatives au projet et la **rédaction du plan qualité** du projet afin de synthétiser les différentes composantes de celui-ci.

Cette phase permet également de faire un état des lieux sur les différentes technologies utilisées, les frameworks, **l'infrastructure technique** et **les systèmes d'information** en place. Elle débouche sur la présentation du **cahier des charges**, le choix de la **méthodologie de développement** et **le planning de travail** qui sera suivi tout au long du projet.

Itérations de développement

Cette phase est celle où le développement à proprement parler (le codage) se déroule. Elle consiste en plusieurs **itérations** de développement qui débouchent chacune sur un **incrément** du logiciel conformément au processus de développement XP (voir 5.2). Chaque itération consiste en un petit **cycle de vie** (figure 5.3) complet, qui inclut :

Recueil des besoins Durant cette étape, la partie du cahier des charges relative à la fonctionnalité à produire lors de cette itération est revue et précisée avec l'équipe. Une **description précise** des besoins est rédigée et les **cas d'utilisation** sont étudiés. La synthèse de cette étape sert de fil directeur au reste de l'itération.

Analyse et conception Les **objets métiers** sont dégagées de la spécification des besoins rédigée à l'étape précédente. Leurs **relations et propriétés** sont précisées et le meilleur moyen de les **intégrer à l'architecture** en place est décidé.

Implémentation et tests Cette étape consiste à implémenter (coder) les composants conçus lors de l'étape précédente. Elle permet de **concrétiser** tout ce qui a été analysé et discuté depuis le début de ce cycle de vie. Ce travail est guidé par les **tests unitaires** et validé et confirmé par les **tests d'intégration** et **d'acceptance**. Ces tests se déroulent sur le serveur de développement, pour éviter de compromettre l'environnement de production.

Déploiement et tests C'est la dernière étape d'un cycle de vie. On procède au déploiement de **l'incrément** résultant de l'ensemble du travail précédent sur le serveur de production. L'équipe procède à des **tests fonctionnels** et une petite réunion est tenu pour faire l'état des lieux et préparer l'étape suivante.

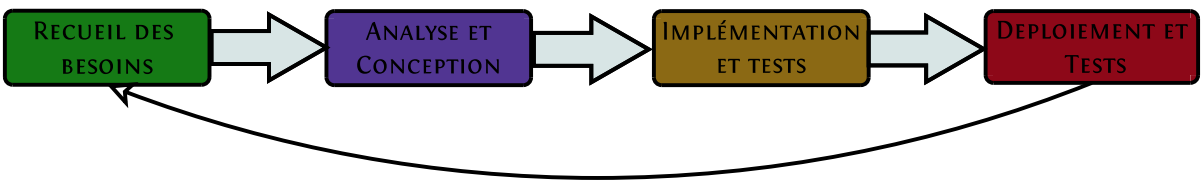


FIGURE 6.3.: Une itération du cycle de développement

Maintenance

Cette phase interviendra après la complétion du développement du projet. L'équipe reste en contact pour pouvoir faire évoluer l'application de le temps. La maintenance recherchée est principalement **corrective** et **évolutive**.

Les problèmes qui seraient éventuellement rencontrés lors de l'utilisation quotidienne de l'application doivent pouvoir être corrigés et l'application doit pouvoir évoluer dans le temps pour s'adapter au petits changements de flots de travail, de matériel ou d'infrastructure technique.

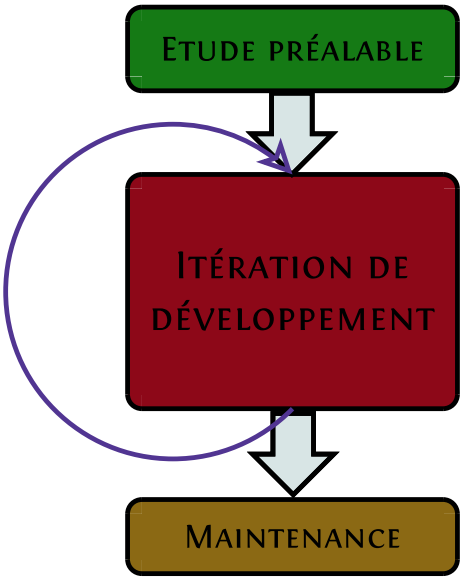


FIGURE 6.4.: Les phases de développement du projet

6.4. Planning

Ci-dessous est présenté le planning (figure 5.5) établit en concertation avec l'équipe Nevo-Technologies, qui prend en compte les priorités de développement et permet de suivre un chemin bien tracé lors du développement.

Il permet également de ne pas se perdre en cours de route et de gérer efficacement le temps. Ce planning a été revu et amélioré tout a long du processus de réalisation, pour prendre en compte les changements de priorité ou les difficultés rencontrées lors du développement.

Nom	Durée	Date de début	Date de fin
Etude préalable	12	16/4/12	1/5/12
Familiarisation avec le CRM	3	16/4/12	18/4/12
Elaboration du cahier des charges	3	19/4/12	23/4/12
Étude du fonctionnement interne du CRM	7	23/4/12	1/5/12
Itérations	46	30/4/12	2/7/12
Itération 1 - Synchroniser les comptes avec QB	9	30/4/12	10/5/12
Recueil des besoins	1	30/4/12	30/4/12
Analyse et conception	3	30/4/12	2/5/12
Implémentation et tests	4	4/5/12	9/5/12
Déploiement et tests	2	9/5/12	10/5/12
Itération 2 - Gestion des clients finaux	5	11/5/12	17/5/12
Recueil des besoins	2	11/5/12	14/5/12
Analyse et conception	1	14/5/12	14/5/12
Implémentation et tests	2	15/5/12	16/5/12
Déploiement et tests	1	17/5/12	17/5/12
Itération 3 - Associations inter-modules	3	18/5/12	22/5/12
Itération 4 - Module Gestion des devis	9	23/5/12	4/6/12
Recueil des besoins	2	23/5/12	24/5/12
Analyse et conception	3	25/5/12	29/5/12
Implémentation et tests	4	29/5/12	1/6/12
Déploiement et tests	2	1/6/12	4/6/12
Itération 5 - Récupération des devis de QB	4	4/6/12	7/6/12
Itération 6 - Gestion des produits	18	7/6/12	2/7/12
Recueil des besoins	3	7/6/12	11/6/12
Analyse et conception	5	11/6/12	15/6/12
Implémentation et tests	7	20/6/12	28/6/12
Déploiement et tests	3	28/6/12	2/7/12

FIGURE 6.5.: Planning de travail détaillant l'aspect gestion du temps du projet

Sur la figure 5.6 on peut voir le diagramme de Gantt [2] correspondant au planning.

6.4.1. Analyse des écarts

Le planning montré ci-dessus est une version finale. Nous avons jugé inutile de montrer les versions intermédiaires et prévisionnelles, mais il convient tout de même d'analyser les causes de ces révisions. Les principales sources de retard sur les temps prévus ont été :

La participation aux tâches quotidiennes de Nevo-Technologies En effet lors de mon travail j'ai été intrigué par certains aspect du fonctionnement de la société et j'ai passé du temps à les comprendre. Ces aspects concernent notamment le fonctionnement global de l'entreprise, les principes directeurs et le fonctionnement de l'économie de marché, l'infrastructure réseau de Nevo-Technologies, etc.

2. http://fr.wikipedia.org/wiki/Diagramme_de_Gantt

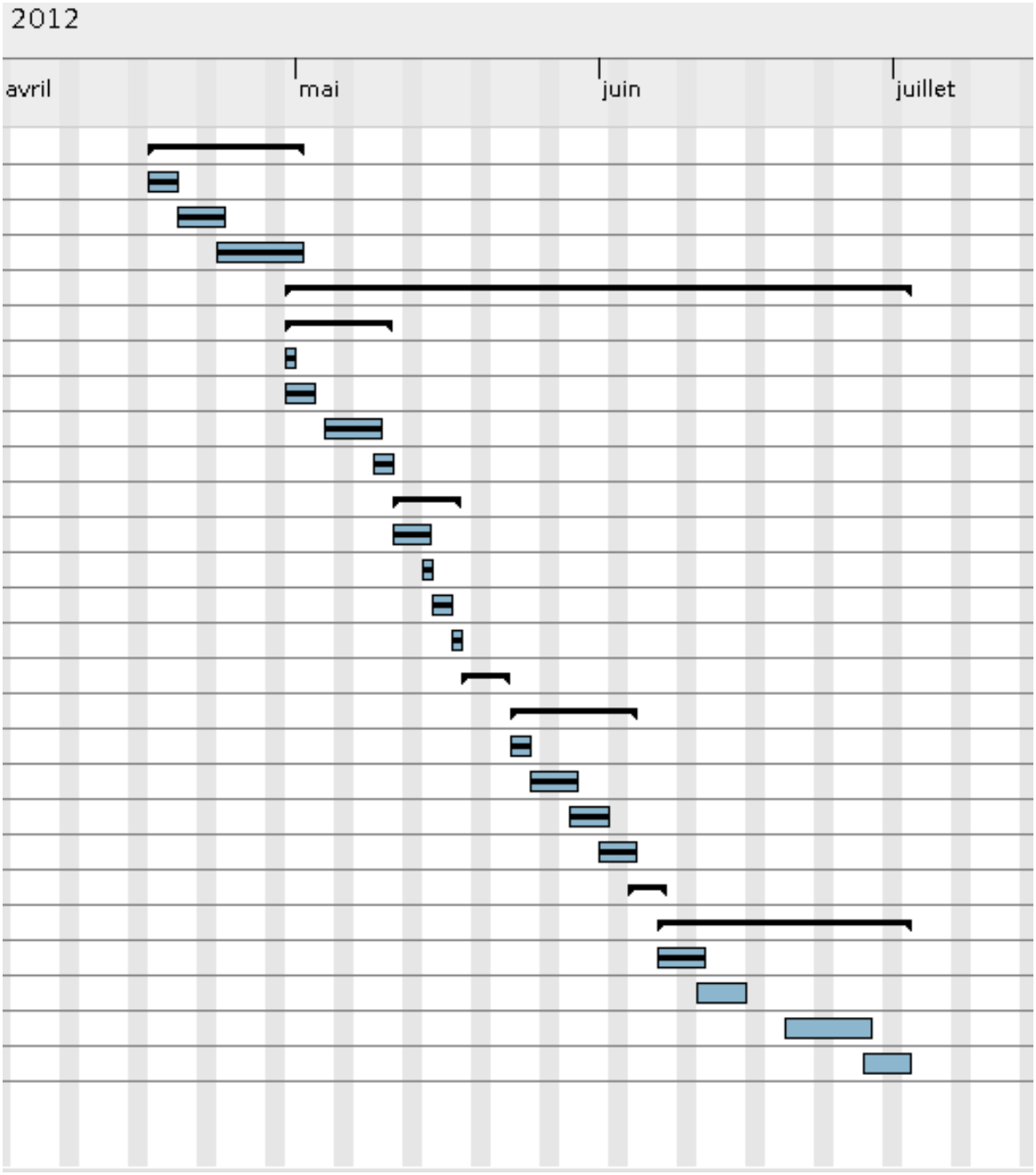


FIGURE 6.6.: Diagramme de Gantt représentant visuellement les étapes du projet

J'ai eu la chance d'avoir des gens responsables et à l'écoute autour de moi, et j'en ai profité pour apprendre de nouvelles choses.

La sous-estimation de la difficulté de certaines tâches Certaines tâches apparaîssent comme le reflet même de la simplicité, mais s'avèrent en fait très coriaces à réaliser ou à conceptualiser correctement, ce qui débouche forcément sur de petits retards.

La familiarisation avec de nouveaux environnements Se familiariser à l'API de QuickBooks (voir 7), comprendre son fonctionnement dans les détails et apprendre

à l'utiliser correctement a pris plus de temps que prévu, notamment à cause de la nature fermée du code source et de la documentation éparpillée sur le web. La familiarisation avec Windows Server 2008 quant à elle a été rapide et n'a pas engendré de retard.

Le manque de documentation Aux premiers abords la documentation de SugarCRM semble riche et bien fournie. Malheuresement, si l'on a besoin de choses très détaillée et d'une explication précise du fonctionnement interne de ce dernier, il n'existe pas grand chose à se mettre sous la dent.

SugarCRM est un logiciel OpenSource, ce qui veut dire qu'on peut se plonger dans le code et comprendre petit à petit le fonctionnement. C'est l'approche qui a été effectivement adoptée, mais celle-ci est loin d'être aussi productive que la présence d'une documentation et d'un guide du développeur offrants les informations nécessaires.

6.5. Outils et frameworks

Ci-dessous une brève description des outils utilisés pour le développement, les serveurs, etc.

Système d'exploitation (production) Windows Serveur 2008.

Système d'exploitation (développement) GNU/Linux Archlinux.

Serveur HTTP Apache HTTPD. Ce serveur propulse des millions de sites webs. Il a été l'un des acteurs majeurs du développement du web et par extension de la popularisation d'Internet.

Base de donnée MySQL. Une valeur sûr de la gestion de base de données, facile à déployer, des performances correctes et pléthore de ressources.

Gestion de version Git. Git est un logiciel de gestion de versions de management du code décentralisé, qui met l'accent sur la vitesse d'exécution.

Gestion de projet BitBucket.org. Un dépôt privé sur le site de collaboration bitbucket.org permet de recenser les problèmes et bugs trouvés par l'équipe, de partager la documentation et de synchroniser les dépôt git du serveur de production et celui de développement.

Développement GNU Emacs - Firefox - Firebug.

Modélisation UML. UML est un langage standardisé de modélisation. Au fil du temps, il est devenu le standard industriel pour modéliser les systèmes qui font une utilisation intensive des logiciels.

7. Conclusion

Dans cette partie nous avons présenté **les besoins reccueillis** au près de l'équipe Nevo-Technologies et nous les avons **organisé** sous forme de **cahier des charges** présentant les **objectifs** et les **règles de gestion** de chaque développement souhaité.

Nous avons également présenté le **processus général de développement** suivi, ses avantages et les raisons de son adéquation, ainsi que **la méthodologie** de sa mise en place pour ce projet particulier.

Finalement nous avons présenté **le planning du projet** tel qu'il est utilisé pour **guider et cadrer** le développement, et nous avons analysé certaines **causes du retard** pris sur quelques étapes du développement.

Deuxième partie .

Architecture et composants existants

Dans cette partie nous présentons les composants déjà en place et l'architecture technique avec laquelle il a fallu se familiariser avant de commencer le développement.

Nous expliquons également le fonctionnement interne du logiciel à adapter, SugarCRM.

8. QuickBooks

8.1. Présentation

QuickBooks est un logiciel développé par Intuit, Inc.. Cette société a été fondée en 1983 par Scott Cook et Tom Proulx à Moutain View, en Californie. Leur premier produit, Quicken, s'attaquait à la gestion financière personnelle. Quicken rencontra un grand succès, et Intuit décida de développer une solution similaire pour les petites entreprises.

Au fil des années, QuickBooks n'a cessé de s'améliorer et d'engranger toujours plus de parts de marché. Ainsi en Septembre 2005, QuickBooks dominait 74% des parts de marché. Aujourd'hui QuickBooks est l'un des logiciels les plus utilisés par les entreprises pour mener à bien leur projet et faciliter leur travail de gestion quotidien.

QuickBooks intègre des fonctionnalités de comptabilité, de gestion commerciale et de paye. Il permet entre autres :

- La comptabilisation des factures d'achats, de ventes et des relevés bancaires et cartes.
- La création à la demande de journal ou balance comptable.
- La création des documents des comptes annuels : Bilan, compte de résultat, etc.
- Création de facture, de bon de commande ou de devis.
- Payement directement à partir du logiciel.

QuickBooks permet donc de gérer l'ensemble des opérations de l'entreprise. Toute l'exploitation de celle-ci peut être facilement analysée. On peut voir sur l'interface principale (7.1) tout ce qui est géré par QuickBooks, de la création de devis et bons de commandes à l'écriture et l'envoi de chèque de paiement.

8.2. Intégration avec QuickBooks

QuickBooks n'est pas un logiciel *opensource*[1]. Cela veut principalement dire que son fonctionnement interne et le format dans lequel les données sont enregistrées ne peuvent être examinés pour apprendre comment récupérer ces données.

1. http://fr.wikipedia.org/wiki/Open_source

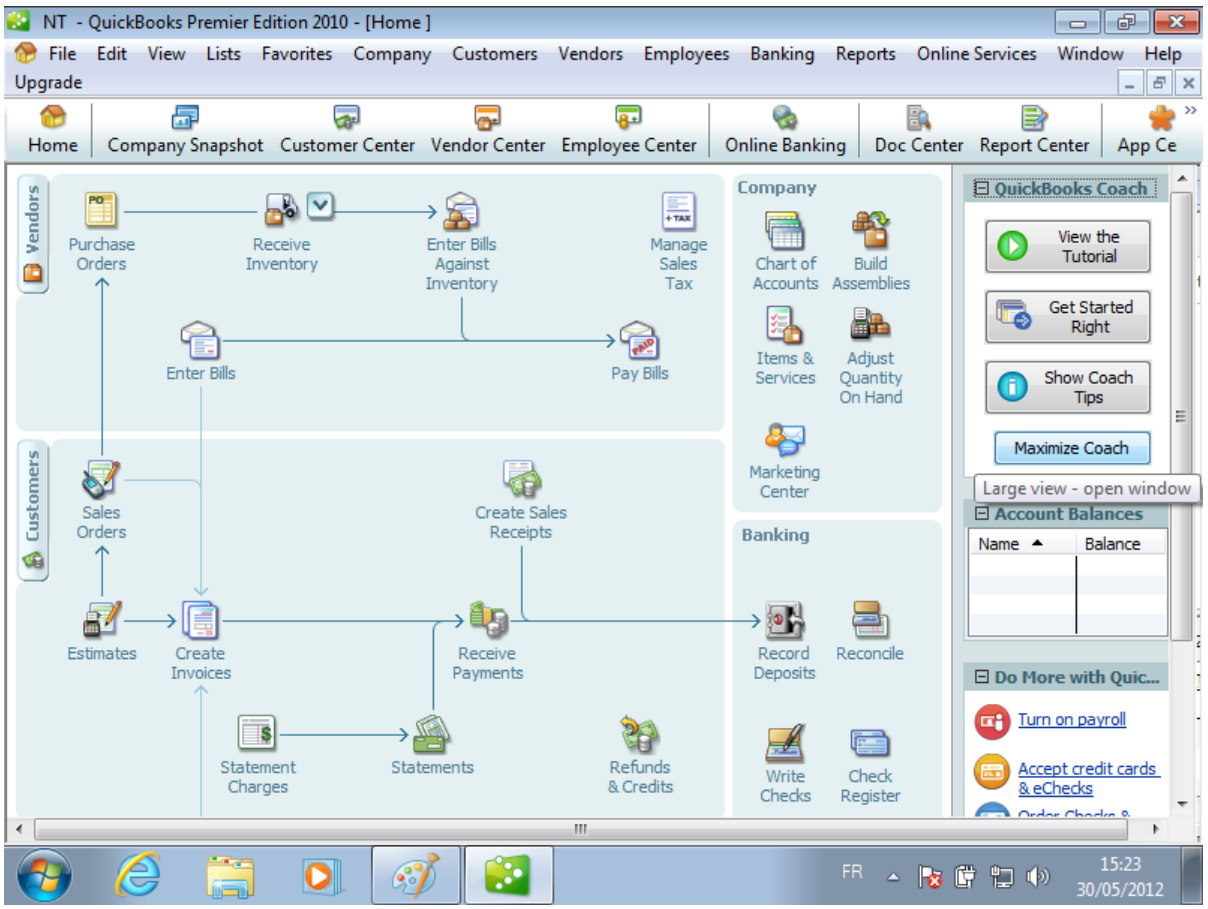


FIGURE 8.1.: Interface principale de QuickBooks

Après quelques recherches, il s'est avéré qu'il existe néanmoins certains moyens pour interagir avec QuickBooks, récupérer des données et même en créer si cela s'avérait nécessaire. Parmi ces moyens figurent :

CSV [2]/Excel Import/Export Cette première méthode est la plus simple. Elle consiste à importer ou exporter les données de QuickBooks dans l'un des formats CSV ou Excel. Cette méthode totalement statique n'est pas adéquate puisqu'elle demande un effort constant de la part de l'utilisateur de l'application pour effectuer les opérations d'import/export.

COM [3]/ DCOM [4]/ QBFC [5] Les classes fournies par ces frameworks permettent aux développeurs d'applications de les connecter avec QuickBooks et d'inter-changer les données. L'application développée doit cependant se trouver sur le même hôte que l'instance de QuickBooks utilisée.

2. Comma-Separated Values
3. Component Object Model
4. Distributed Component Object Model
5. QuickBooks Foundation Classes
6. QuickBooks Web Connector

Nous n'avons pas choisis cette solution pour différentes raisons. Premièrement les classes fournies sont beaucoup plus adaptées au développement d'application de bureau avec l'un des langages développés par la société Microsoft, notamment les langages .NET (C#, VB.NET, etc.). Deuxièmement l'application développée ne se trouve pas sur le même hôte que QuickBooks. Ce point sera expliqué plus en détail ci-dessous.

QBWC[6] QuickBooks Web Connector est une application qui vient se placer entre QuickBooks et une application web, et qui gère l'échange de données entre ces deux entités. C'est la méthode qui a été adoptée pour notre projet. Son fonctionnement est expliqué en détail dans la section suivante.

8.3. QuickBooks Web Connector

QBWC, comme dit précédemment, se place entre l'application web désirant échanger des données avec QuickBooks et ce dernier. Avant d'expliquer la nature de cet échange et les éléments mis en jeu, il serait éclairant d'expliquer pourquoi une connection directe entre l'application web et QuickBooks n'est pas possible.

8.3.1. QuickBooks et COM

COM est une interface binaire créée par Microsoft. Elle permet *la communication inter-processus* et la création dynamique d'objets. La communication inter-processus est l'ensemble des techniques et méthodes permettants à différents *threads*[7] — appartenants à un ou plusieurs processus — de communiquer. Les processus peuvent tourner sur la même machine ou sur des machines différentes connectées par un réseau.

Pour qu'une application puisse accéder aux données de QuickBooks à travers son SDK[8], elle doit instancier le *processeur de requêtes* de ce dernier à travers COM. Le processeur de requêtes du SDK de QuickBooks est l'élément qui s'occupe de recevoir les requêtes de logiciels extérieures, les traiter et éventuellement les passer à QuickBooks.

Le problème rencontré est que COM nécessite que le COM Object Server (le serveur qui envoit les données) et son client (l'application web développée) se trouvent sur la même machine, ou sur le même réseau local. Pour contourner cette limitation, la solution trouvée par les développeurs par le passé a été de développer des applications qui se placent entre QuickBooks et l'application web.

7. Thread ou fil d'éxécution. Forme de processus très léger à ressources partagées. Généralement contenu dans un processus

8. Software Development Kit

Cette application intermédiaire règle parfaitement le problème posé plus haut. Elle gère la communication avec l'application web, et communique avec QuickBooks via COM sans problèmes puisqu'elle se trouve sur la même machine hôte. Néanmoins, la création de ce type d'applications demande un travail long et considérable, qui ajoute beaucoup de travail au processus de développement.

C'est pour alléger ce travail et le faciliter que QBWC a été créé. C'est une solution gratuite et standard au problème. Sa principale fonction est d'agir en tant que tunnel ou conduit par lequel passent toutes les requêtes et réponses échangées entre l'application web et QuickBooks.

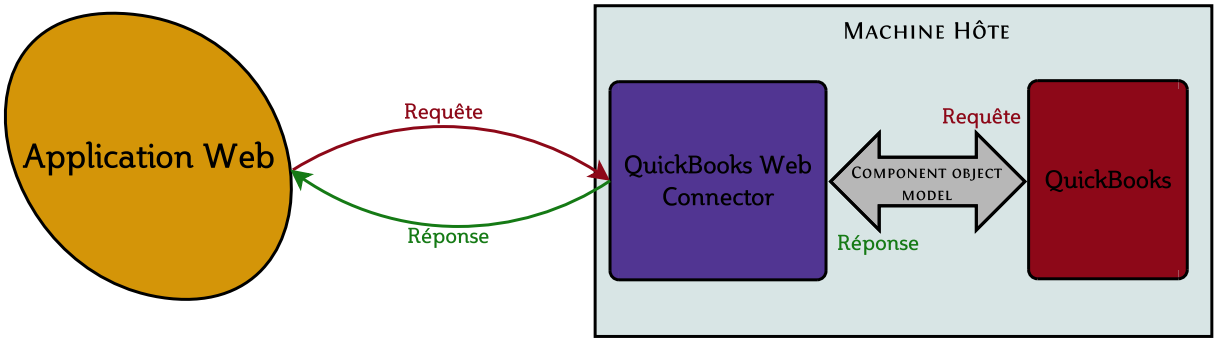


FIGURE 8.2.: Rôle de QBWC dans la communication avec QuickBooks

Établissement de connexion avec QuickBooks Web Connector

Pour communiquer avec QBWC, l'application web doit pouvoir se connecter à la machine hôte hébergeant ce dernier. Or pour pouvoir se connecter à cette machine, l'administrateur doit ouvrir son pare-feu, ce qui affaiblit la sécurité de la machine. On peut comprendre que cela ne soit pas acceptable, surtout lorsque des données sensibles sont stockées dessus.

Le diagramme plus haut (figure 7.2) est donc légèrement incorrect, puisque le pare-feu de la machine hôte ne laisse pas la communication passer vers QBWC.

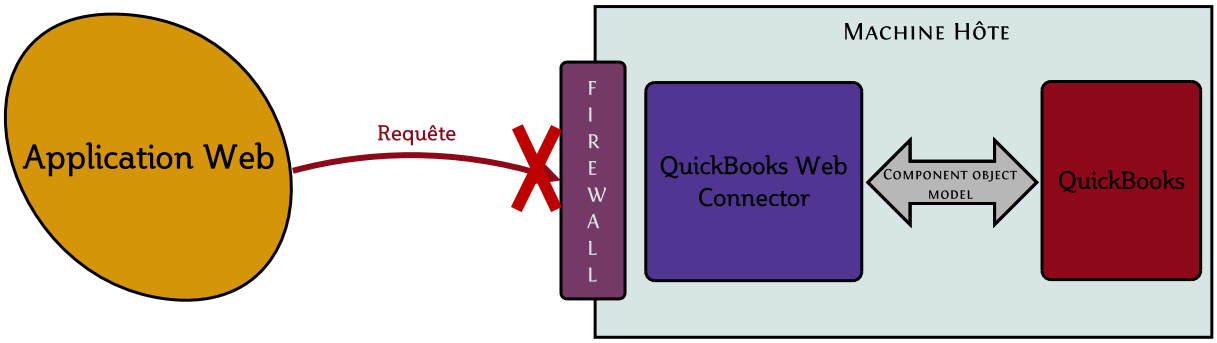


FIGURE 8.3.: Le pare-feu bloque l'établissement de la communication

Pour comprendre comment QBWC traite ce problème, il faut s'attaquer à son mode de connection et de communication avec l'application web.

Fonctionnement de QBWC

Pour régler le problème cité plus haut, QBWC inverse le sens de la communication avec l'application web. Au lieu d'attendre une connexion provenante de cette dernière et de la traiter, voici comment se déroule la communication :

- QBWC se connecte à intervalles réguliers à l'application web. C'est lui qui initie la communication, non pas l'application, et donc élimine le problème de pare-feu bloquant les connexions entrantes.
- L'application web confirme la demande de connexion. Un canal de transmission est établit.
- QBWC demande à l'application s'il y a des requêtes en attente.
- L'application répond en envoyant les requêtes qui ont été mises en attente par l'utilisateur.
- QBWC traite les requêtes et envoie sa réponse contenant les données requises.

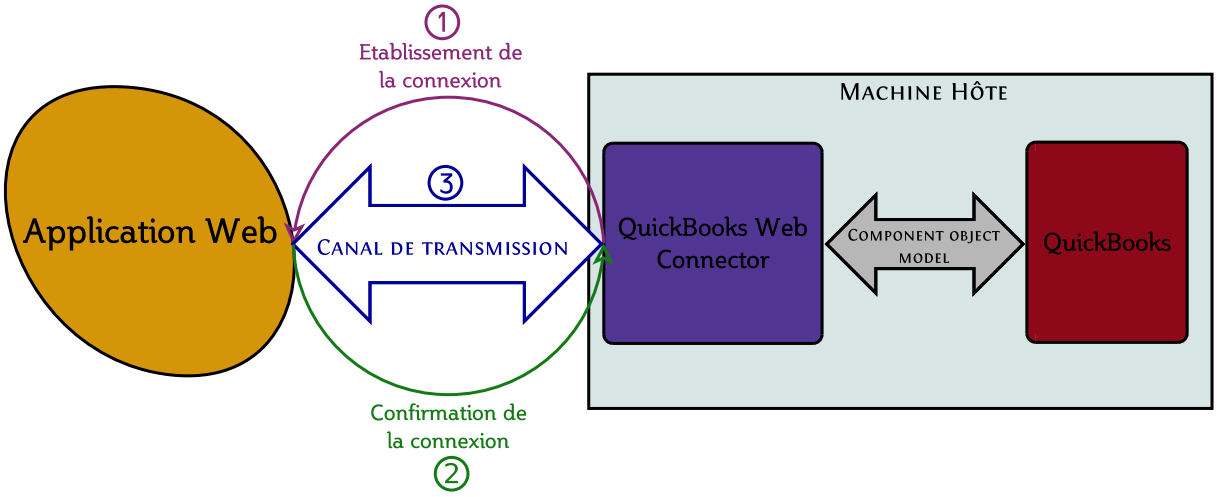


FIGURE 8.4.: Déroulement de la communication entre l'application et QBWC

8.3.2. Communication avec QBWC

Maintenant que nous avons expliqué le fonctionnement de QBWC ainsi que les raisons qui ont poussé à son utilisation, attaquons-nous au déroulement des communications. Deux aspects sont importants à comprendre : la configuration initiale de QBWC, et le déroulement des communications à proprement parler.

Configuration initiale

La configuration initiale de QBWC qui indique l'existence de l'application web se fait à travers un fichier QWC[9]. C'est un fichier XML[10] qui donne certaines informations pour établir la connection qui permettra de communiquer avec votre application. Voici un exemple de fichier QWC :

Listing 8.1: Exemple de fichier QWC

```
<?xml version="1.0"?>
<QBWCXML>
  <AppName>SCRM QB Integrator</AppName>
  <AppID></AppID>
  <AppURL>http://localhost/sugar/qb_integ/qbwc_soap_server.php</AppURL>
  <AppDescription></AppDescription>
  <AppSupport>http://localhost/sugar/qb_integ/qbwc_support.php</AppSupport>
  <UserName>sugarcrm</UserName>
  <OwnerID>{90A44FB7—33D9—4815—AC85—AC86A7E7D1EB}</OwnerID>
  <FileID>{57F3B9B6—86F1—4fcc—B1FF—967DE1813D20}</FileID>
  <QBType>QBFS</QBType>
  <Scheduler>
    <RunEveryNMinutes>10</RunEveryNMinutes>
  </Scheduler>
  <IsReadOnly>false</IsReadOnly>
</QBWCXML>
```

L'interface de QBWC (figure 7.1) permet d'ajouter votre application en renseignant le fichier QWC. Après quelques étapes votre application est prête et la communication peut être enclenchée. L'interface de QBWC permet également de régler l'intervalle de temps séparant les communications successives.

Communications

Après cette configuration initiale, les communications peuvent enfin commencer. La première étape pour échanger des données est de mettre une requête en attente. C'est le rôle de l'application web, et nous verrons comment c'est fait un peu plus loin (sec. 7.4). Après cette étape, c'est au tour de QBWC de se connecter à l'application, de traiter la requête et de retourner la réponse, comme expliqué dans la section Fonctionnement de QBWC (7.3.1).

9. QuickBooks Web Connector
10. eXtensible Markup Language

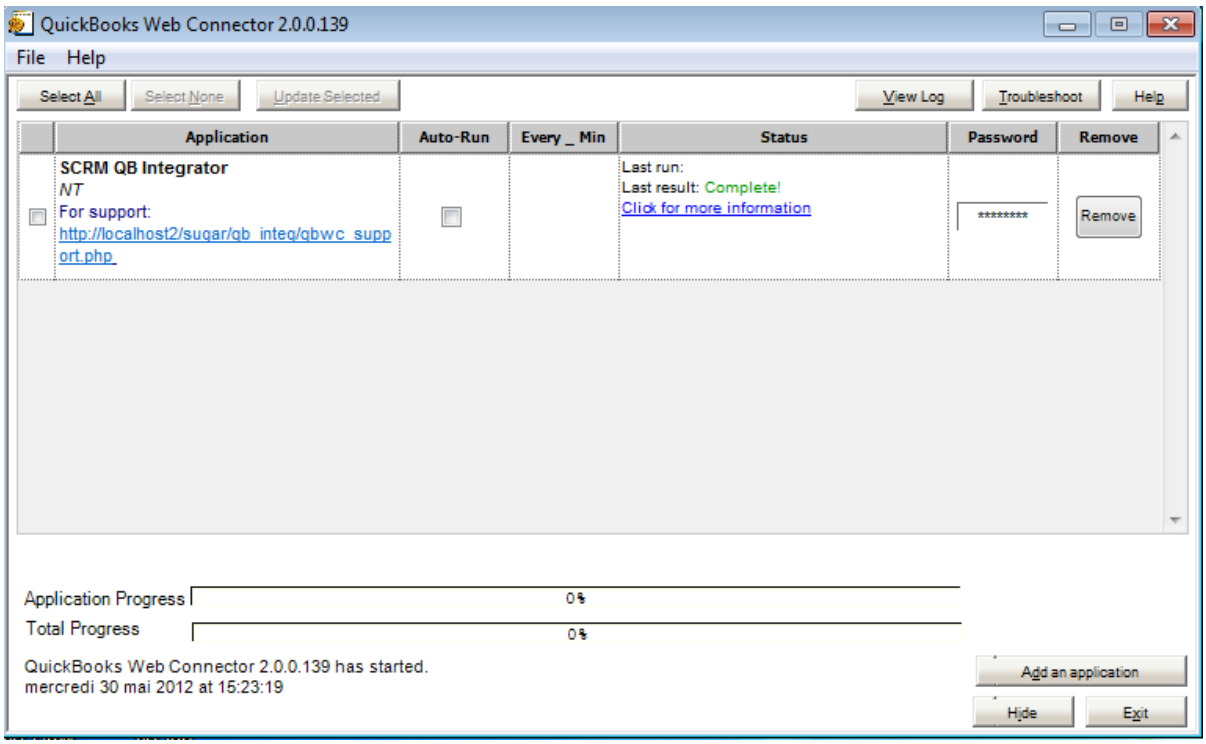


FIGURE 8.5.: Interface de QuickBooks Web Connector

Le langage utilisé pour communiquer est QBXML[11]. Il est basé sur XML et permet de récupérer et d'envoyer toutes sortes d'informations de ou vers QuickBooks. Voici un exemple qui permet de récupérer les 40 premiers comptes clients de QuickBooks :

Listing 8.2: Exemple de requête QBXML

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <?qbxml version="9.0"?>
3 <QBXML>
4   <QBXMLMsgsRq onError="stopOnError">
5     <CustomerQueryRq iterator="Start">
6     <MaxReturned>40</MaxReturned>
7   </CustomerQueryRq>
8 </QBXMLMsgsRq>
9 </QBXML>
```

IDN[12] *Unified OSR*[13] est une référence complète qui décrit les formats de toutes les requêtes et réponses possibles en QBXML. La référence est consultable à l'url[14] http://developer.intuit.com/qbsdk-current/common/newosr/index.html. Voici une partie

11. QuickBooks eXtensible Markup Language
12. Intuit Developer Network
13. On-Screen Reference
14. Uniform Resource Locator

du format de la réponse en QBXML envoyé par QuickBooks suite à une requête pour récupérer les devis :

Listing 8.3: Exemple de format de réponse QBXML

```
<?xml version="1.0" encoding="utf-8"?>
<?qbxml version="9.0"?>
<QBXML>
  <QBXMLMsgsRq onError="stopOnError">
    <EstimateQueryRs statusCode="INTTYPE" statusSeverity="STRTYPE"
       statusMessage="STRTYPE" retCount="INTTYPE" iteratorRemainingCount="
       INTTYPE" iteratorID="UUIDTYPE">
      <EstimateRet> <!-- optional, may repeat -->
        <TxnID >IDTYPE</TxnID> <!-- required -->
        <TimeCreated >DATETIMETYPE</TimeCreated> <!-- required -->
        <CustomerRef> <!-- required -->
          <ListID >IDTYPE</ListID> <!-- optional -->
          <FullName >STRTYPE</FullName> <!-- optional -->
        </CustomerRef>
        ...
        ...
        <EstimateLineRet> <!-- optional -->
          <!-- Informations sur les produits du devis -->
        </EstimateLineRet>
        ...
        ...
      </EstimateRet>
    </EstimateQueryRs>
  </QBXMLMsgsRq>
</QBXML>
```

8.4. Gestion de la communication par l'application web

Le fonctionnement de la communication du côté de QBWC étant maintenant clair, nous allons nous intéresser à ce qui se passe du côté de l'application web.

Il est nécessaire de développer un serveur SOAP [15] qui répond à certaines méthodes pour pouvoir communiquer avec QBWC. En effet ce dernier se connecte au serveur SOAP pour récupérer les requêtes en attente. Le déroulement de la communication est

15. Simple Object Access Protocol

le suivant :

QBWC → SOAP Serveur Web Connector se connecte au serveur SOAP en utilisant le nom d'utilisateur indiqué dans le fichier de configuration et le mot de passe configuré sur l'interface de QBWC.

QBWC → SOAP Serveur Web Connector envoie un message au serveur SOAP demandant s'il existe des requêtes en attente.

SOAP Serveur → QBWC Le serveur doit générer une requête en QBXML et l'envoyer à Web Connector.

QBWC → QuickBooks Web Connector passe la requête à QuickBooks et reçoit la réponse.

QBWC → SOAP Serveur Web Connector envoie la réponse au serveur SOAP. Celui-ci la traite et effectue les opérations nécessaires.

SOAP Serveur → QBWC Le serveur SOAP envoie un pourcentage représentant la quantité de travail accomplie. Si ce pourcentage est est inférieur à 100%, ces étapes se répètent en boucle jusqu'à l'accomplissement du travail à effectuer.

Pour chacune de ces étapes, le serveur SOAP doit implémenter une fonction qui s'occupera d'effectuer le travail nécessaire : gérer l'authentification, traiter les réponses ou envoyer les requêtes. Les fonctions à implémenter sont les suivantes :

authenticate QBWC se connecte au serveur SOAP en appelant cette fonction. La vérification du nom d'utilisateur et du mot de passe s'effectue ici. Cette fonction doit également générer un *ticket* qui identifiera cette communication. Il sera inclut dans toutes les requêtes et réponses suivantes.
Cette fonction indique à QBWC s'il y a du travail en attente dans la file de requêtes ou pas.

sendRequestXML Si la connexion est réussie et la file de requêtes n'est pas vide, QBWC utilise cette fonction pour demander au serveur d'envoyer la prochaine requête à traiter.

receiveResponseXML QBWC envoie la réponse et les données nécessaires au serveur SOAP en appelant cette fonction. Le traitement des données de la réponse se fait ici. Cette fonction doit retourner un nombre indiquant l'avancement des opérations.

connectionError, closeConnection Ces fonctions sont appelées respectivement si une erreur s'est produite et pour fermer la connexion en cours.

9. SugarCRM

Dans ce chapitre nous allons présenter SugarCRM, détailler son fonctionnement interne et expliquer la structure de ses composants.

9.1. Présentation

SugarCRM une solution logicielle opensource de GRC[1]. La société SugarCRM a été fondée en 2004 par John Roberts, Clint Oram et Jacob Taylor.

Le projet s'est rapidement imposé comme l'une des meilleurs solutions opensource de GRC, atteignant la barre des 25 000 téléchargements le mois de septembre de la même année de sa création. Il a été nommé « projet du mois » sur SourceForge[2], un site de travail collaboratif abritant des milliers de projets opensource.

Forte du succès de sa solutoin, la société a pu lever 46 millions de dollars de fond capital-risque[3], ce qui lui a permit de se développer énormément. Aujourd'hui SugarCRM est un des leaders du marché des solutions de GRC opensource. La société emploie plus de 180 personnes et distribue 5 versions différentes de son logiciel : Sugar Community Edition, Sugar Professional, Sugar Corporate, Sugar Enterprise et Sugar Ultimate. Toutes les versions sont basées sur le même code source, et diffèrent simplement par le nombre de fonctionnalités inclues. Sugar CE[4] par exemple, contient a peu près 85% des fonctionnalités possibles.

SugarCRM a été initialement développé sur une infrastructure Linux, Apache, PHP et MySQL, mais peut fonctionner avec des systèmes alternatifs. Il est possible par exemple de l'installer sur Windows, avec le serveur web MS IIS[5], et d'utiliser MS SQL ou Oracle comme serveur de base de données. SugarCRM peut être également déployé sur le cloud[6] au lieu d'être hébergé localement. Le système donc est très flexible et peut s'adapter à n'importe quel environnement.

1. Gestion de la relation client - Traduction controversée de l'anglais Customer Relashionship Management
2. http://sourceforge.net/potm/potm-2004-10.php
3. http://venturebeat.com/2008/02/07/sugarcrm-raises-20m-more-for-open-source-crm/
4. Community Edition
5. Internet Informations Services
6. http://en.wikipedia.org/wiki/Cloud_computing

La version utilisée à Nevo-Technologies est Sugar CE, les fonctionnalités contenues dans les autres versions n'étants pas indispensables au travail quotidien de l'entreprise.

9.2. Modules Principaux

Nous présentons ici succintement certains modules de SugarCRM (voir qu'est-ce qu'un module sec. 8.3 pour une explication technique).

Accounts Le module *Accounts* sert à collecter et organiser les données concernant les clients directs de l'entreprise. Il permet de centraliser les données qui concernent les clients, de les partager et de les garder un jour. Un compte client peut être relié à des contacts, des rendez-vous, des appels, etc. ou à toute autre type d'information enregistrée dans SugarCRM.

Contacts Ce module permet de gérer les contacts de l'utilisateur du CRM. Il permet de garder trace de leurs informations : email, numéro de téléphone, etc. Il sert d'annuaire et peut être partagé avec les autres utilisateurs pour leur laisser la possibilité de profiter de ces informations.

Opportunités La gestion des opportunités est l'un des aspects les plus importants d'un *business*. Ce module, entièrement dédié à cet aspect, permet d'enregistrer toutes les informations nécessaires comme la probabilité de réussite, le montant impliqué, le client concerné, la date d'expiration, le statut courant, etc. Il va sans dire qu'il facilité énormément la tâche et permet de capitaliser les informations reccueillies par le passé pour aider au mieux l'entreprise à faire des profits.

Leads[7] La possibilité d'enregistrer rapidement et précisément les informations concernant une potentielle opportunité est essentielle. Essayer de garder ces informations en tête pendant une discussion ou bien les écrire quelque part d'une manière destructurée et hasardeuse pourrait mettre en péril la justesse de l'information reccueillie, et par là même faire rater une opportunité à l'entreprise.

C'est précisément à cette tâche que se consacre le module *Leads*, permettant de collecter toutes sorte d'informations sur une opportunité potentielle, la relier à d'autres informations et la convertir en une réelle opportunité le temps venu.

Calendrier Le nom de ce module est assez clair pour ne pas mériter d'amples explications. Néanmoins c'est l'un des modules les plus importants du CRM pour

7. La traduction la plus satisfaisante à laquelle je peux penser est tuyau, mais étant trop familière et pas exactement fidèle au sens d'origine, j'utilise Lead

optimiser le travail quotidien d'une entreprise. Avoir un calendrier partagé veut dire connaitre les disponibilités de toutes l'équipe ainsi que leurs besoins, et permet d'organiser son temps efficacement et d'en faire profiter toute l'entreprise.

Emails SugarCRM permet de gérer les emails sans utiliser aucun autre logiciel. Ce module permet de recevoir et d'envoyer des emails, en utilisant n'importe quel boite de l'utilisateur. On peut ainsi rassembler plusieurs boites emails professionnelles et les gérer à l'aide d'une seule interface. L'avantage est également d'avoir accès aux informations concernant les emails pour les relier et enrichir toute la base de connaissance gérée par le CRM.

Beaucoup d'autres modules *Appels*, *Projets*, *Tâches*, *Bugs*, *Rendez-vous*, etc.

On peut voir les liens vers les différents modules sur l'interface principale de SugarCRM. (8.1)

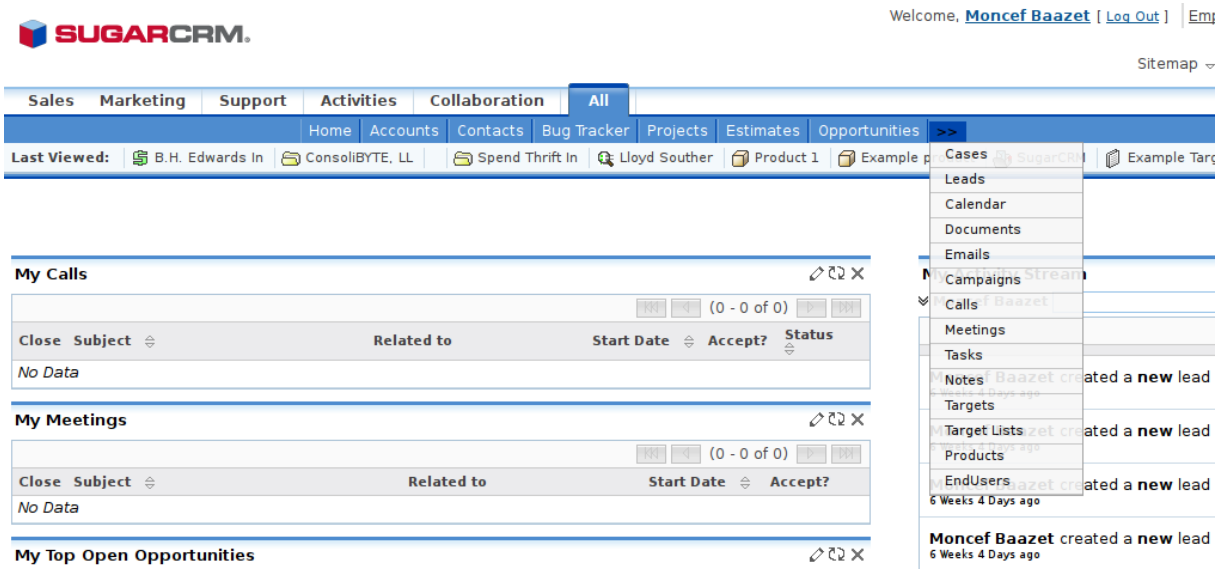


FIGURE 9.1.: Interface principale de SugarCRM

9.3. Concepts structurants

Maintenant que les présentations sont faites, nous allons nous attaquer au fonctionnement interne SugarCRM. Toute la structure de SugarCRM est basée sur certains concepts qui, avec quelques variations, permettent de schématiser l'application.

Record

Un **record** est l'entité de base manipulée par SugarCRM. Il possède un **identifiant** et peut avoir un nombre arbitraire d'attributs le décrivant. Il peut être assimilé à un

élément de type produit dans un langage de programmation quelconque ou un élément d'un produit cartésien d'ensembles.

D'une manière un peu moins abstraite, il peut être représenté par un enregistrement dans une table d'une base de données, et c'est effectivement la représentation qui lui est conférée dans SugarCRM. Deux exemples de record sont présentés dans les figures 8.2 et 8.3.

Un contact est représenté par :
- Un numéro d'identification.
- Un nom.
- Un numéro de téléphone.
- La date de la création du record.
- La date de dernière modification.
- Une adresse.
- Un lieu de travail.

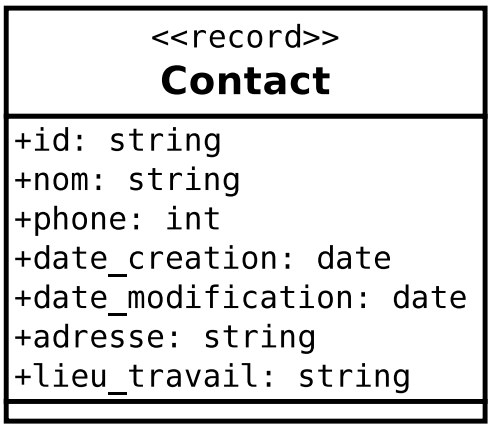


FIGURE 9.2.: Exemple d'un record représentant un contact

Un des points forts de SugarCRM et qui est très important dans son fonctionnement est qu'il est possible de relier un record à n'importe quel autre. Ce lien est enregistré au même titre que les autres informations du record.

Une société est représentée par :
- Un numéro d'identification.
- Une raison sociale.
- Une adresse.
- Un revenu annuel.
- Un site web.

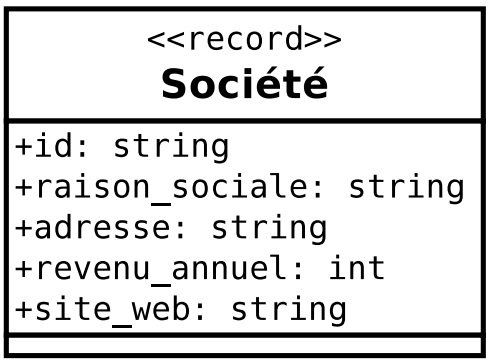


FIGURE 9.3.: Exemple d'un record représentant une société

Module

Le module est la deuxième unité structurante de SugarCRM, se trouvant à un niveau un peu plus elevé d'abstraction. Un module est essentiellement un groupement de record de même nature. Un module peut par exemple s'occuper de regrouper les records

des voitures, tandis qu'un autre peut s'occuper de regrouper les records de types de nourritures servis par un restaurant. D'un point de vue technique, les modules et records sont liés par une relation d'aggrégation.

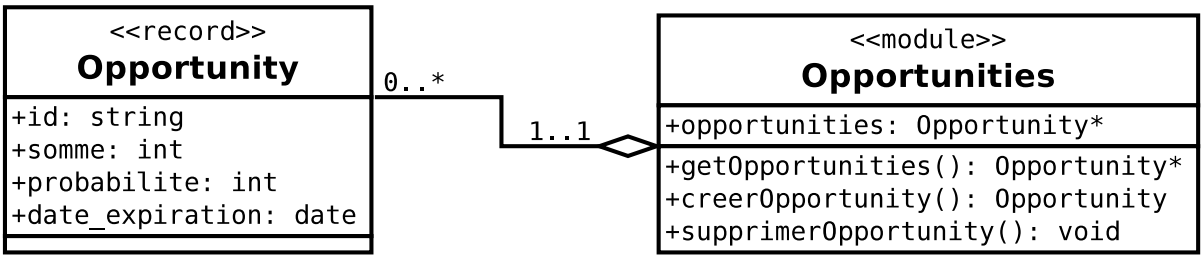


FIGURE 9.4.: Exemple de module réunissant des opportunités.

Les modules s'occupent également d'offrir le minimum de fonctionnalités communes et nécessaires, à savoir la création de nouveaux records, la modification de records existants, la suppression et la consultation des records existants et toutes leurs informations associées.

Ainsi par exemple le module *Rendez-vous* permet de créer de nouveaux rendez-vous, de consulter ceux existants, de les supprimer ou les modifier lorsque cela s'avère nécessaire. Il est également possible de savoir à quel client (*Account*) est relié ce rendez-vous, à quel *contact* s'adresser et quelle est *l'opportunité* ou le *lead* concerné.

Les actions principales possibles à effectuer sur tous les modules par défaut sont la consultation, la modification, la création, la suppression et la recherche de record. D'autres actions, moins utilisées, sont la fusion et l'exportation. (voir figure 8.5)

Les extensions et modification apportée à SugarCRM sont généralement soit l'ajout d'un nouveau module, soit l'ajout d'une action possible à effectuer sur les records d'un ou tous les modules.

Pour l'ajout d'un nouveau module, SugarCRM propose une structure facilitant cette opération et permet de profiter des actions de bases assez rapidement. L'ajout d'actions à un module sort un peu du cadre fixé par SugarCRM, mais est néanmoins rendu possible par l'arrangement des composants.

Avant de s'attaquer à la manière dont sont construits les modules et actions ajoutés, nous allons expliquer comment fonctionnent ceux déjà présents dans SugarCRM, l'architecture technique de ce dernier ainsi que les détails du framework proposé.

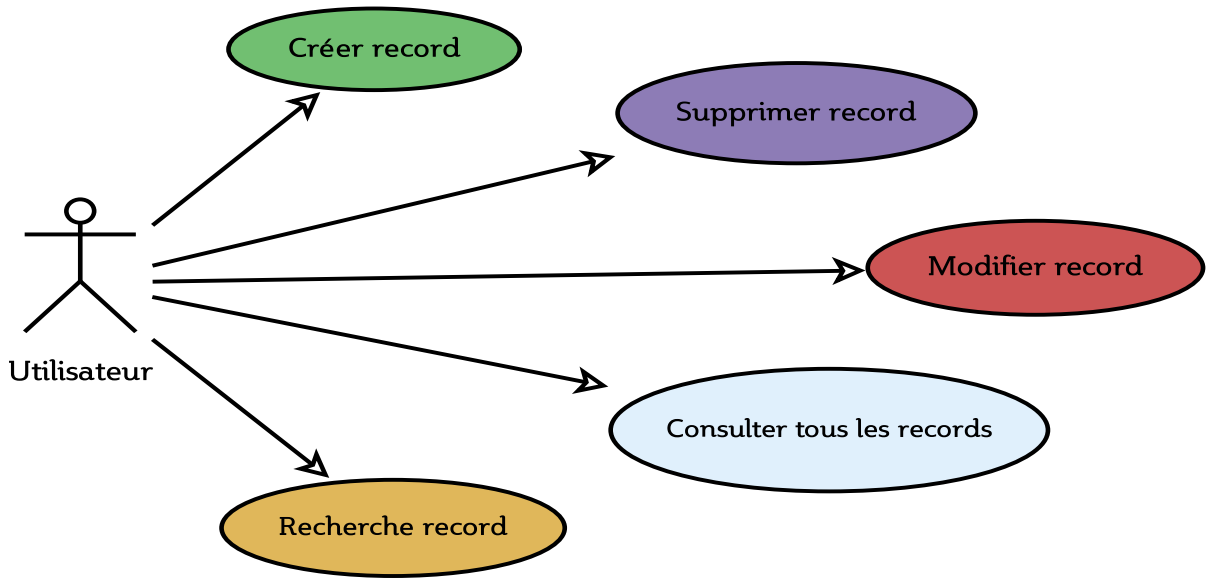


FIGURE 9.5.: Cas d'utilisations généraux. Tous les modules possèdent le même fonctionnement.

9.4. Architecture technique

Le fonctionnement de tous les modules de SugarCRM est conceptuellement similaire. Les différences qui peuvent exister se trouvent à des niveaux plus bas d'abstractions, notamment au niveau des actions proposées par chaque module, et seront donc expliquées dans les sections appropriées.

9.4.1. MVC

MVC [8] est un *design pattern* architectural très répandu permettant de **séparer le code de traitement et de présentation** d'une application du **code métier** représentant le coeur du domaine traité. Ce *pattern* a été introduit initialement dans Smalltalk [9] — un langage de programmation orienté objet —, notamment par Trygve Reenskaug [10], qui explique :

> *"MVC was conceived as a general solution to the problem of users controlling a large and complex data set."*

Grossièrement, la manière la plus simple de voir MVC est que le modèle représente les données, la vue représente la fenêtre ou l'interface de l'application sur l'écran, et le controlleur est la « colle » qui relie les deux.

8. Model-View-Controller
9. Un article présentant le *pattern* et son utilisation originelle dans Smalltalk-80 : http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html
10. http://c2.com/cgi/wiki?TrygveReenskaug

SugarCRM utilise le *pattern* MVC. Lorsqu'un utilisateur arrive sur l'interface, un composant spécial appelé le *dispatcher*, *router* ou encore *controlleur frontal* s'occupe de récupérer l'adresse demandée ainsi que toutes les données fournies par l'utilisateur, les traiter et décider quel module utiliser. Le controlleur frontal est implémenté par la classe `SugarApplication`.

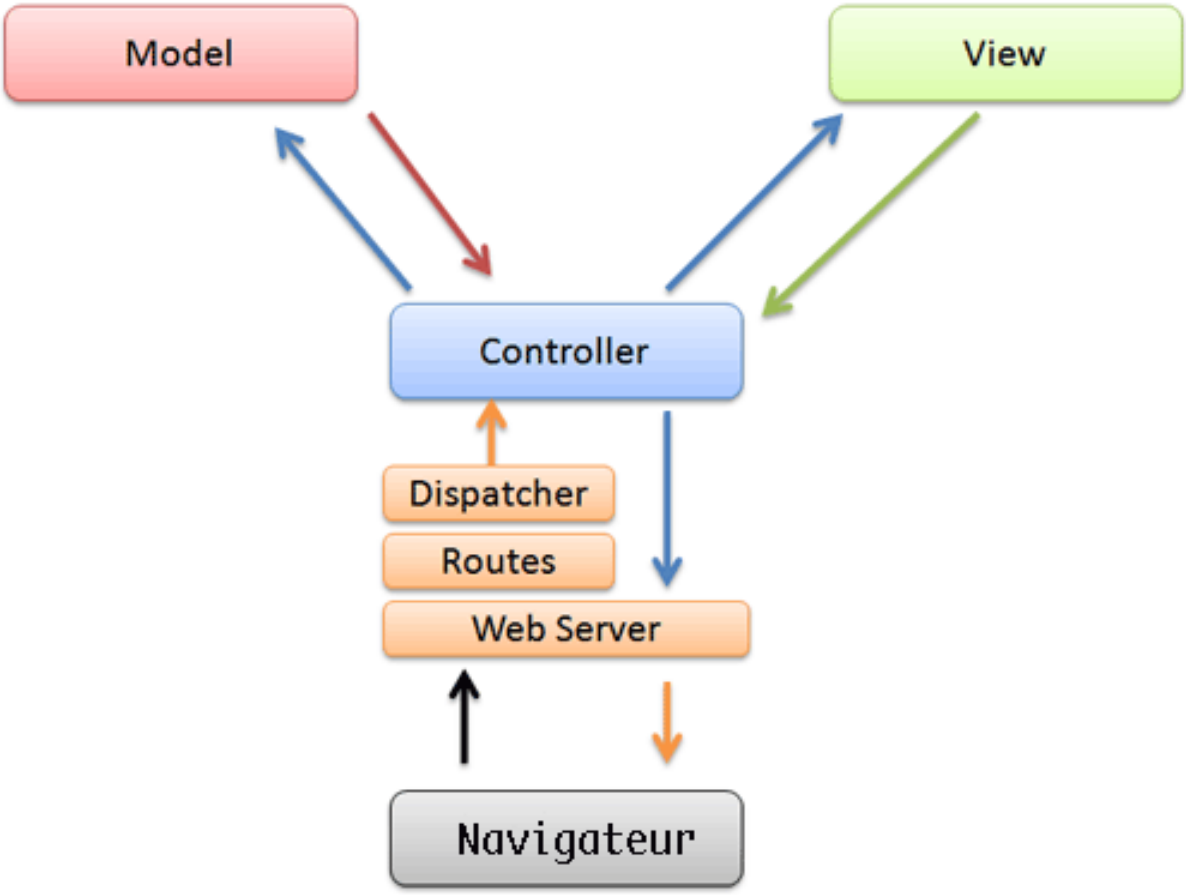


FIGURE 9.6.: Le *design pattern* Model-View-Controller

Lorsque le module à utiliser est connu, son controlleur prend le déroulement des opérations suivantes en main. SugarCRM fournit un controlleur par défaut, qui est utilisé si un module ne fournit pas son propre controlleur. Dans le cas contraire, le controlleur du module « hérite » des fonctionnalités du controlleur par défaut, qui permettent souvent d'éviter de refaire beaucoup de choses déjà faites.

Les fonctionnalités de bases telle l'enregistrement de record ou leur récupération sont ainsi déjà présentes, bien qu'elles doivent être modifiées lorsque les records traités ont des particularités propres. Le controlleur par défaut de SugarCRM est implémenté dans la classe `SugarController`. (voir figure 8.7)

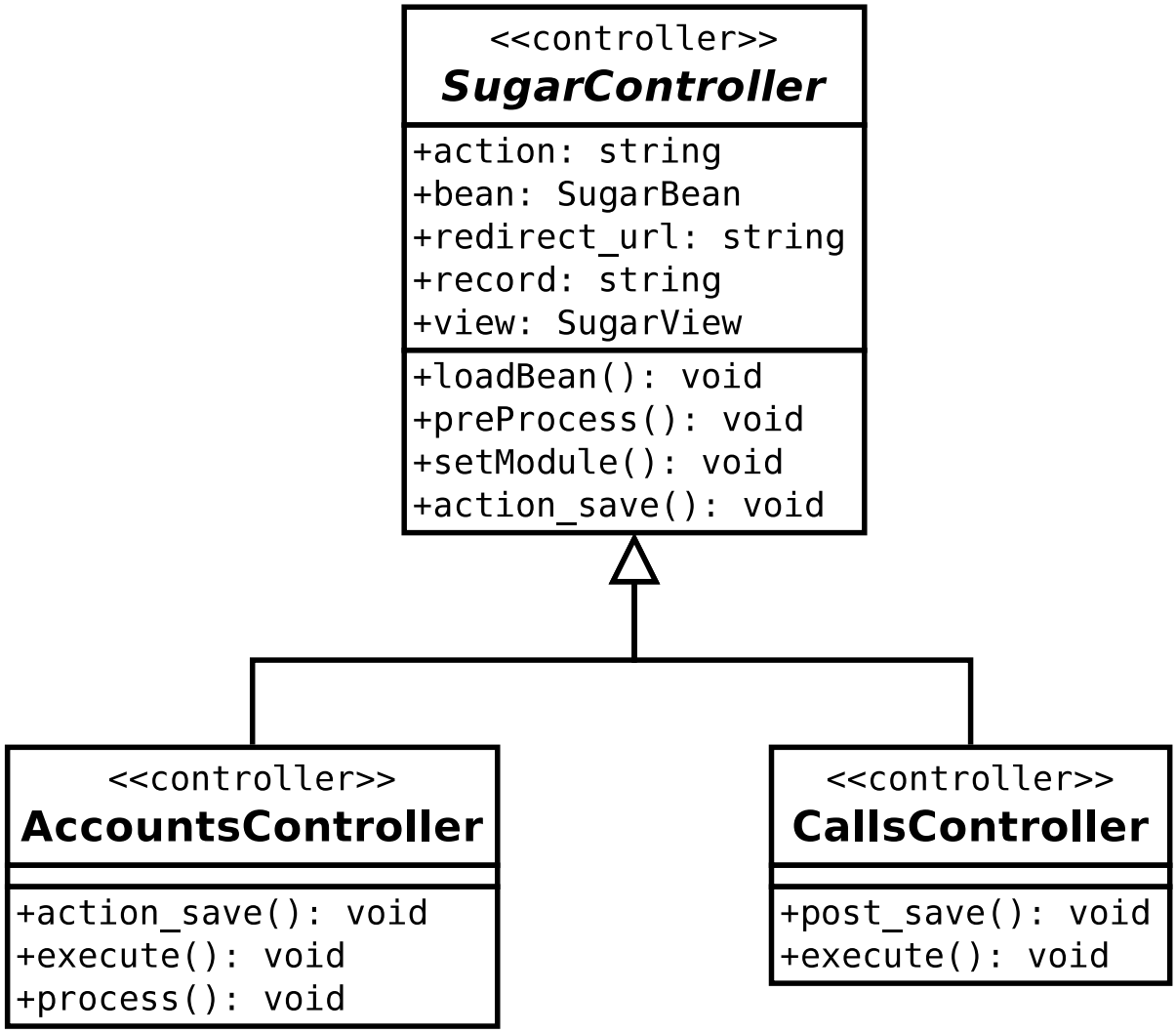


FIGURE 9.7.: Les controlleurs dans SugarCRM héritent tous du controlleur par défaut

Pour faire son travail correctement, le controlleur doit récuperer les informations concernant le ou les records à traiter. Le concept même de MVC l'empêche de récuperer ces informations directement. Il doit donc passer par le modèle.

Les modèles de SugarCRM sont des classes héritants du modèle par défaut fournit, SugarBean. Les attributs du record sont représentés par les variables membres de la classe, et les méthodes publiques fournies sont utilisées par le controlleur pour récupérer les données nécessaires au bon déroulement des opérations. (voir figure 8.8)

Template Objects

Lors de la création de modèles pour représenter les entités en jeu, on remarque que certains attributs se répètent très souvent. Pour les modèles représentants un type de personne par exemple, on retrouve souvent le nom complet, l'email, l'adresse, le téléphone, etc.. Pour un modèle représentant une société, on retrouve l'adresse, le propriétaire, la raison sociale, etc.. Plus généralement, pour n'importe quel modèle

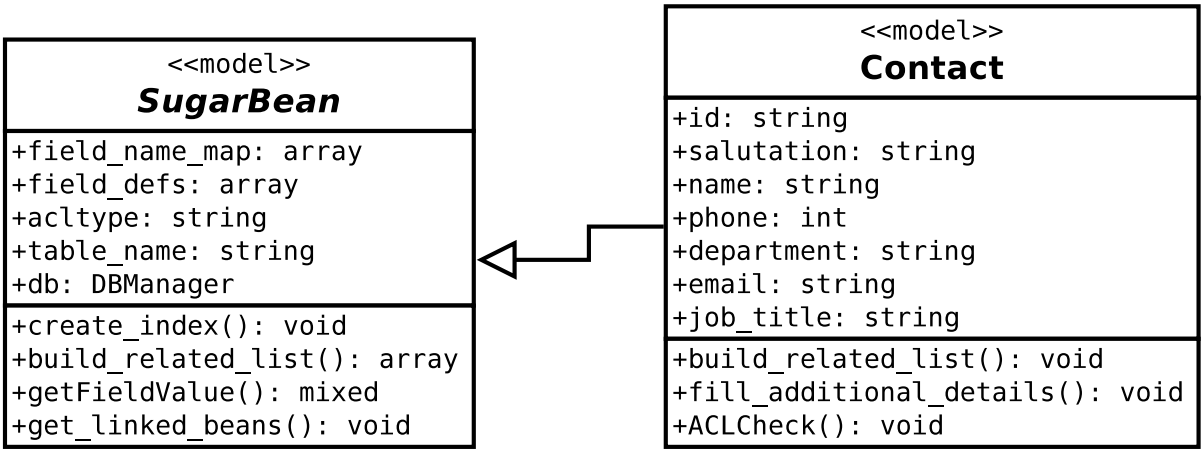


FIGURE 9.8.: Classe représentant le modèle dans SugarCRM

représentant un record, on retrouve un numéro d'identification, une date de création, une date de dernière modification, une description, etc..

Pour éviter de refaire tout ce travail à chaque fois, SugarCRM propose des modèles un peu plus développé que le modèle de base. Ce sont des modèles prêts, héritants des fonctionnalités de SugarBean et dont il est possible d'hériter à leur tour. L'avatange est qu'il y a moins de travail à refaire. Ces modèles sont appellés *template objects* dans SugarCRM. (voir figure 8.9)

Pour le développement du module EndUser par exemple, on a fait hériter le modèle du *template object* Company, qui représente une société. Bien qu'il ait fallu apporter des modifications importantes, une grosse partie du travail fût éliminée en évitant l'utilisation directe du modèle SugarBean très basique. Company offrait par exemple de base les attributs d'adresse, de telephone, email, etc.

Revenons à notre flot de déroulement des opérations. Maintenant que nous avons récupéré les informations nécessaires, il est temps de les afficher. Le controlleur passe les informations à la vue adéquate. La vue affichée est determinée par l'action demandée par l'utilisateur. Il existe par exemple une vue pour la consultation d'un sommaire des records, une deuxième pour la création de record, une autre pour la consultation des détails d'un record particulier, etc.

Ceci nous donne une idée sur l'ajout d'actions particulières à un module. Pour ajouter une action, il faut ajouter l'action adéquate au controlleur et créer la vue correspondante du module.

Les vues de SugarCRM sont représentées par des classes qui héritent toutes de SugarView, la vue par défaut offerte par SugarCRM. Ainsi la vue permettant de consulter

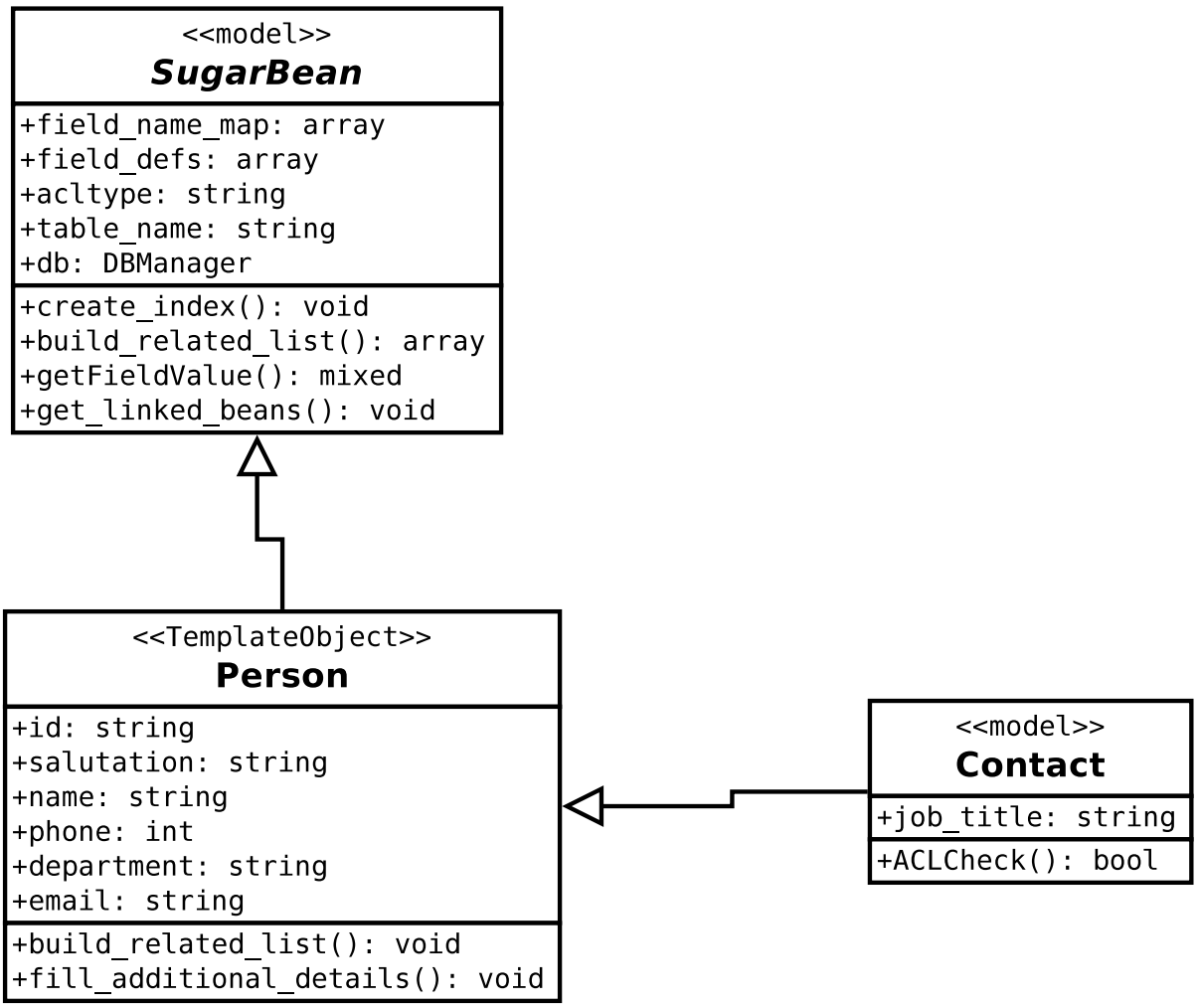


FIGURE 9.9.: Les *template objects* simplifient les modèles

le sommaire des records d'un module, nommée *ListView*, est représentée par la classe `ListView`, qui hérite de `SugarView`. De même pour la vue détaillée, `DetailView`, etc.. (voir figure 8.10)

Les vues principales (création, modification et consultation) peuvent être créées à l'aide d'un *framework* particulier proposé par SugarCRM, le *metadata framework* (voir 8.6). Pour créer une vue sortant de ce cadre, ou l'une des vues précédentes avec certaines particularités, il est nécessaire d'utiliser une méthode un peu plus traditionnelle et d'écrire le code des fichiers des vues et de présentation.

9.5. Base de données

Nous passons maitenant à la structure de la base de données utilisée par SugarCRM. Celle ci reflète très clairement la structuration conceptuelle modules/records du code.

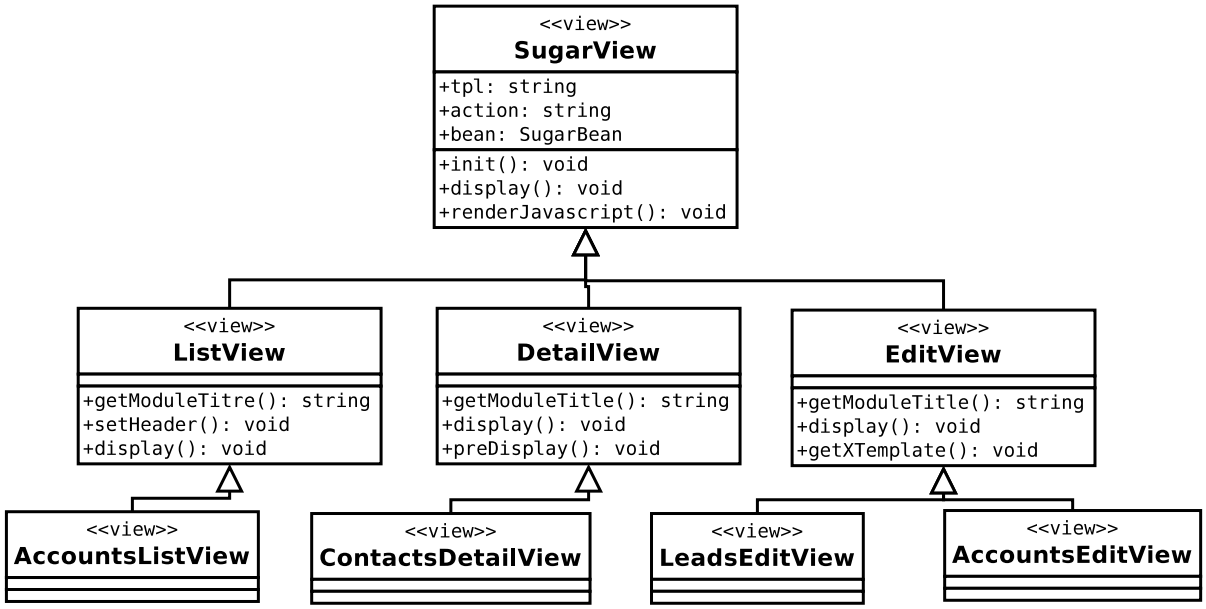


FIGURE 9.10.: Les vues standards de SugarCRM factorisent une grosse partie du travail

9.5.1. Tables principales

Chaque module de SugarCRM est représenté par une table dans la base de données, portant le même nom en minuscules, qui enregistre les records appartenant à ce module. Les records du module *Accounts* par exemple sont enregistrés dans la table *accounts*. Tous les attributs renseignés représentent des colonnes dans la table. (voir exemple figure 8.11)

9.5.2. Liaisons inter-modules

Comme dit précédemment (8.3), il est possible de relier les records de différents modules entre eux. Le problème qui se pose alors est comment enregistrer cette liaison.

Types de liaison

Tout d'abord, il existe deux types de liasons. **one-to-many** et **many-to-many**.

one-to-many (figure 8.12) veut dire qu'une même entité de type A peut être reliée à plusieurs entités de type B à la fois, mais une entité de type B ne peut être reliée qu'un une seule entité de type A à la fois. Un *account* par exemple peut être relié à plusieurs *contacts* (on peut avoir plusieurs contacts au sein d'une société particulière), mais un *contact* ne peut pas être relié à plusieurs *accounts* à la fois (un contact ne peut travailler que dans une seule société à la fois).

many-to-many (figure 8.13) veut dire que toute entité de type A peut être reliée à plusieurs entités de type B, et vice-versa. Plusieurs *contacts* par exemple peuvent être

Server: localhost ▸ Database: sugarcrm ▸ Table: accounts

Browse | Structure | SQL | Search | Insert | Export | Import | Operations

	Field	Type	Collation	Attributes	Null	Default	Extra
☐	id	char(36)	utf8_general_ci		No	None	
☐	name	varchar(150)	utf8_general_ci		Yes	NULL	
☐	date_entered	datetime			Yes	NULL	
☐	date_modified	datetime			Yes	NULL	
☐	modified_user_id	char(36)	utf8_general_ci		Yes	NULL	
☐	created_by	char(36)	utf8_general_ci		Yes	NULL	
☐	description	text	utf8_general_ci		Yes	NULL	
☐	deleted	tinyint(1)			Yes	0	
☐	assigned_user_id	char(36)	utf8_general_ci		Yes	NULL	
☐	account_type	varchar(50)	utf8_general_ci		Yes	NULL	
☐	industry	varchar(50)	utf8_general_ci		Yes	NULL	
☐	annual_revenue	varchar(100)	utf8_general_ci		Yes	NULL	
☐	phone_fax	varchar(100)	utf8_general_ci		Yes	NULL	
☐	billing_address_street	varchar(150)	utf8_general_ci		Yes	NULL	
☐	billing_address_city	varchar(100)	utf8_general_ci		Yes	NULL	
☐	billing_address_state	varchar(100)	utf8_general_ci		Yes	NULL	
☐	billing_address_postalcode	varchar(20)	utf8_general_ci		Yes	NULL	
☐	billing_address_country	varchar(255)	utf8_general_ci		Yes	NULL	
☐	rating	varchar(100)	utf8_general_ci		Yes	NULL	
☐	phone_office	varchar(100)	utf8_general_ci		Yes	NULL	

FIGURE 9.11.: Exemple de table principale

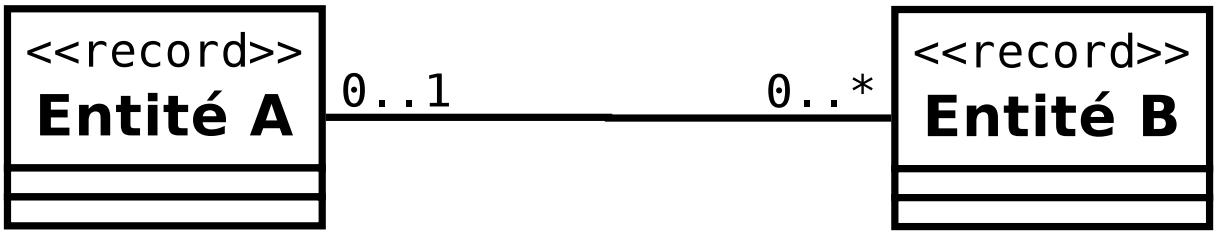


FIGURE 9.12.: Relation one-to-many

impliqués dans une *opportunité*, et une *opportunité* peut impliquer plusieurs *contacts* à la fois.

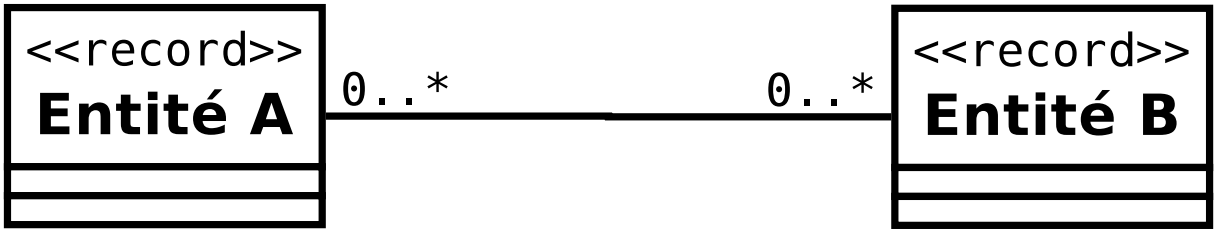


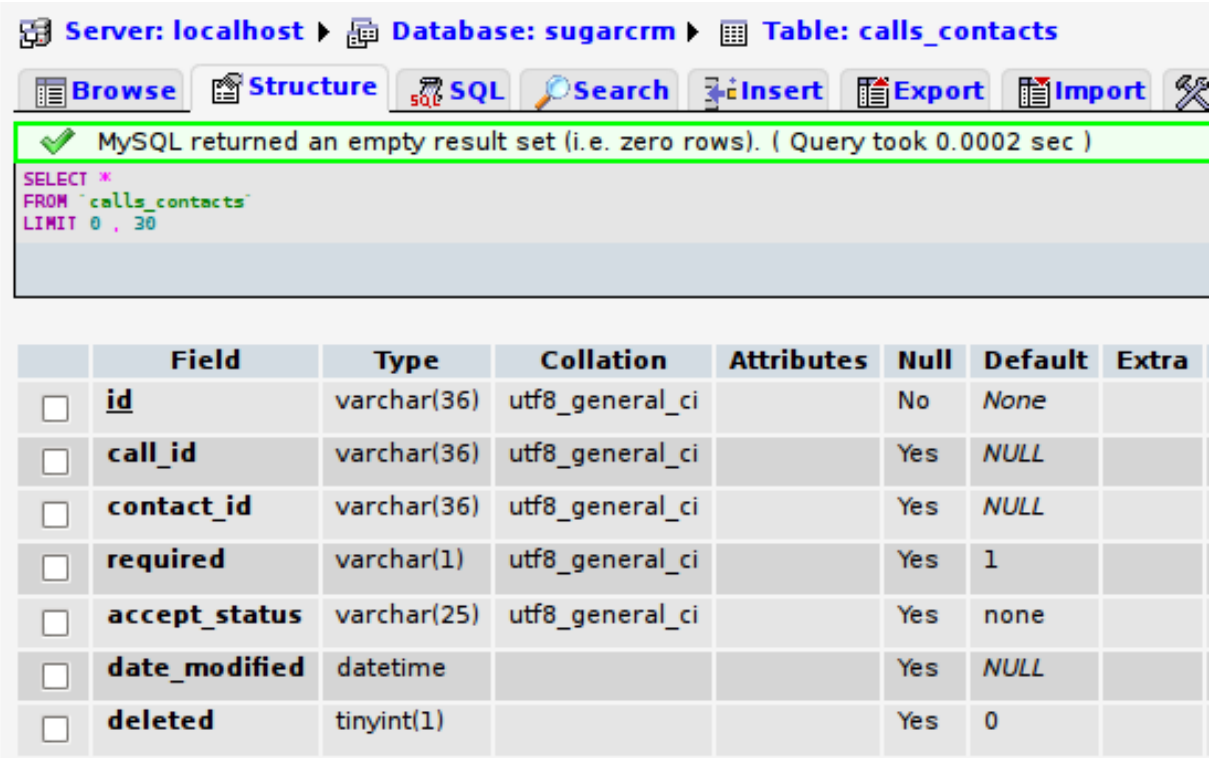
FIGURE 9.13.: Relation many-to-many

Pour les relations one-to-many, la relation est assymétrique. Nous appellerons l'entité pouvant être reliée à plusieurs autres (*account* dans l'exemple plus haut) l'entité principale, et l'entité liée sera appelée l'entité secondaire (*contacts* dans l'exemple).

La solution pour enregistrer cette liaison est d'ajouter une ou plusieurs colonnes à la table de l'entité secondaire qui permettront d'enregistrer les données de l'entité principale que nous voulons garder. La table *contacts* aura par exemple une colonne *account_id* qui permettra de la relier à un *account*.

Pour les relations many-to-many, la solution consiste à ce que, pour chaque pair de modules liés, il existe une table qui enregistre les liaisons entre les différents records des deux modules. Le nom de la table, ainsi que les attributs clefs des records, sont configurables.

Pour prendre un exemple plus concret, supposons qu'on veuille relier un appel (module *Calls*) à un contact (module *Contacts*). Un même appel peut être relié à plusieurs contacts, et un même contact peut être le sujet de plusieurs appels. La relation est donc de type many-to-many, et il est nécessaire de créer une table pour gérer les records reliés. On peut voir la structure d'une telle table sur la figure 8.14.



	Field	Type	Collation	Attributes	Null	Default	Extra
☐	id	varchar(36)	utf8_general_ci		No	None	
☐	call_id	varchar(36)	utf8_general_ci		Yes	NULL	
☐	contact_id	varchar(36)	utf8_general_ci		Yes	NULL	
☐	required	varchar(1)	utf8_general_ci		Yes	1	
☐	accept_status	varchar(25)	utf8_general_ci		Yes	none	
☐	date_modified	datetime			Yes	NULL	
☐	deleted	tinyint(1)			Yes	0	

FIGURE 9.14.: Exemple de table qui enregistre les relations entres différents records de différents modules

9.6. Metadata Framework

Le *metadata framework* est un ensemble de classes et de composants qui facilitent la création des vues pour SugarCRM. Le principe est le suivant : les vues par défaut (`ListView`, `DetailView`, ... 8.10) ont en règle générale la même structure, quelque soit le module. L'écriture du code HTML[11] pour chacune de ces vues est répétitif et contre-productif. C'est précisément pour éviter cela que le *metadata framework* a été conçu.

SugarCRM permet de créer automatiquement les vues par défaut à partir de fichiers de configurations créés par le développeur. Pour créer une vue de création de *record* pour un module par exemple, il n'est pas nécessaire d'écrire tout le code HTML, mais de simplement créer un fichier de configuration qui indique les propriétés des records gérés par ce module qui devront être affichés. Voici un exemple de fichier de configuration pour une vue de création et de modification (`EditView`) : (8.1).

Listing 9.1: Exemple d'un fichier de configuration d'une vue

```
1  <?php
2  $module_name = 'EndUsers';
3  $viewdefs[$module_name]['EditView'] = array(
4      'templateMeta' => array(
5          'maxColumns' => '2',
6          'widths' => array(
7              array('label' => '5', 'field' => '20'),
8              array('label' => '5', 'field' => '20'),
9          ),
10     ),
11     'panels' => array(
12         'default' => array(
13             array(
14                 array('name' => 'name', 'label' => 'LBL_ENDUSER_NAME'),
15                 array('name' => 'phone'),
16             ),
17             array(
18                 array('name' => 'enduser_address_street',
19                       'type' => 'address',
20                       'hideLabel' => true,
21                       'displayParams' => array(
22                           'cols' => '30',
23                           'rows' => 2,
```

11. HyperText Markup Language

```
24                              'key' => 'enduser',
25                          ),
26                  ),
27                  array('name' => 'contact_name'),
28              ),
29          ),
30      ),
31  );
32  ?>
```

Ce framework permet de gagner en vitesse de développement et d'éviter de réécrire le même code plusieurs fois. Par contre s'il est nécessaire de créer des vues qui ne correspondent pas à celles proposées par défaut, le *metadata framework* n'est plus d'aucune aide.

10. Conclusion

Dans ce chapitre nous avons présenté en détail les différents composants logiciels constituants le système sur lequel le travail de développement va se porter. Le fonctionnement interne de SugarCRM, QuickBooks et QBWC a été éclairci, pour pouvoir construire et travailler sur des bases seines.

Il est important de noter que ce travail n'est jamais totalement terminé et que c'est cette partie est celle qui prend le plus de temps.

Troisième partie .

Composants développés

Cette partie présente le travail de développement effectivement effectué lors du stage.

Chaque chapitre présente le développement d'une partie du cahier des charges. On commence par présenter une partie du travail d'analyse et de conception lorsque le composant n'est pas trop simple pour cela. Puis on présente le travail d'implémentation, c'est à dire le codage et les tables créées.

10.1. Cas d'utilisation

Le concept de *record* uniformise beaucoup d'aspects de SugarCRM. Les cas d'utilisation par exemple sont à peu près les mêmes pour chaque module, et peuvent être exprimés par la figure 8.5.

Pour cela nous ne présenterons pas les mêmes cas d'utilisation pour chaque module, mais seulement lorsqu'il existe un cas d'utilisation non exprimé par ceux par défaut. Dans ce cas nous présenterons seulement le nouveau cas d'utilisation, sans décrire la création, modification, suppression et consultation des records, puisque ceux ci existent par défaut pour tous les modules.

11. Gestion des clients finaux

11.1. Analyse et conception

Un **client final** est représenté par un nom, un téléphone, une ville, un pays.

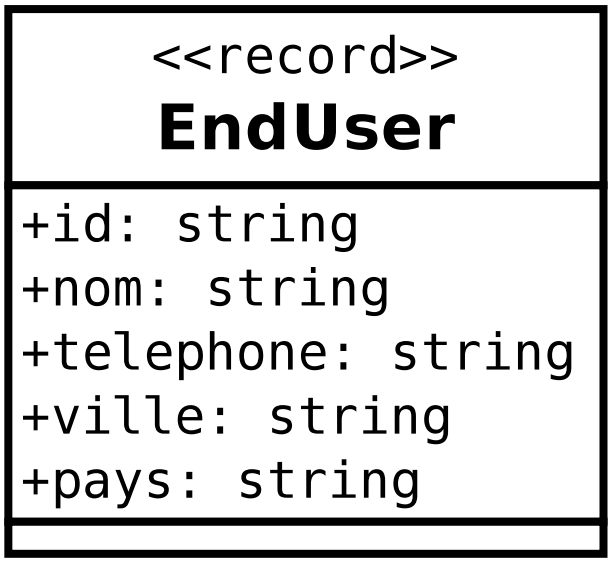


FIGURE 11.1.: Diagramme de classe résumé d'un *EndUser*

- Un *contact* peut être associé à un **client final**.
- Un **client final** peut être associé à une *opportunité* ou un *lead*.

11.2. Implémentation

Le module de gestion des clients finaux ne requiert aucune logique particulière sortant du cadre proposé par SugarCRM. L'implémentation est donc direct et s'appuie sur les mécanismes offerts par SugarCRM et décrits dans le chapitre SugarCRM.

Le controleur par défaut (`SugarController`) est utilisé. Les vues du module héritent directement des vues existantes (`ListView`, `EditView`, etc.). Le modèle quant à lui hérite du template object `Company` puisque celui-ci représente assez fidèlement ce qu'un *EndUser* est sensé décrire.

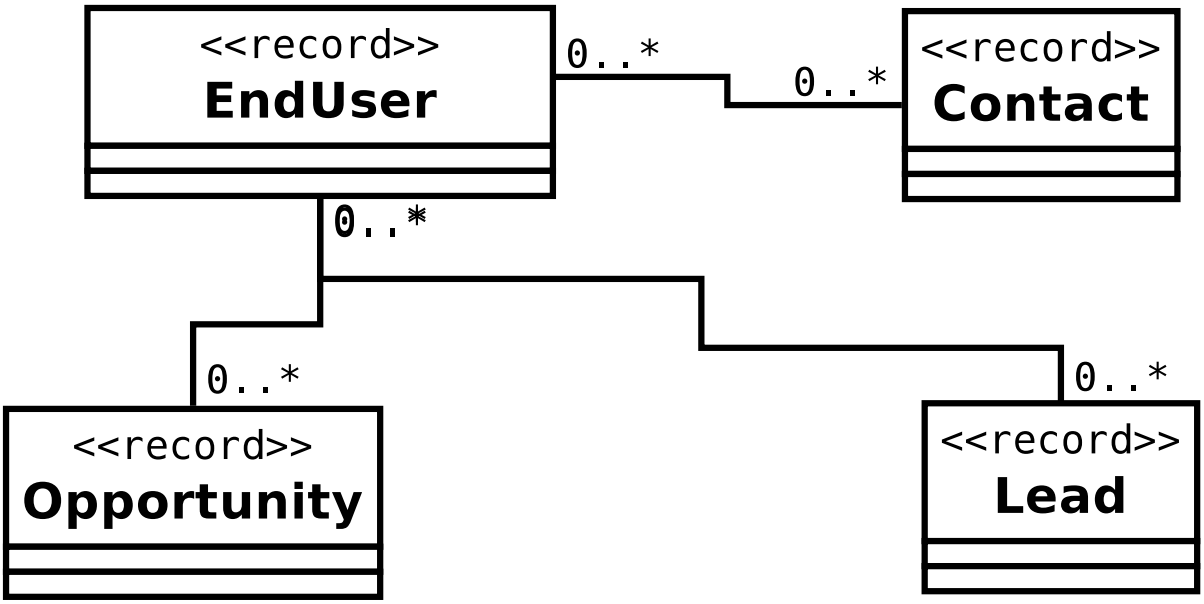


FIGURE 11.2.: Diagramme représentant les associations d'un *EndUser*

Les attributs du modèle reflètent les propriétés d'un *EndUser* et certaines méthodes qui permettent de récupérer les données et les mettre à disposition des vues sont surchargées. En voici un exemple :

Listing 11.1: EndUser.php

```
1  <?php
2
3  function save_relationship_changes($is_update) {
4    if (!empty($this->contact_id)
5        and !empty($this->rel_fields_before_value['contact_id'])
6        and trim($this->contact_id) != trim($this->rel_fields_before_value['
             contact_id']))
7    {
8      $this->load_relationship('contacts');
9      $this->contacts->delete($this->id,$this->rel_fields_before_value['
           contact_id']);
10   }
11
12   parent::save_relationship_changes($is_update);
13 }
14
15 public function get_list_view_data() {
16   $temp_array = $this->get_list_view_array();
17
18   $temp_array['CONTACT_NAME'] = $this->contact_disp_name;
```

```
19    $temp_array['CONTACT_ID'] = $this->contact_disp_id;
20
21    return $temp_array;
22  }
23
24  ?>
```

Les tables utilisées sont une table principale et trois tables de liaisons. La table principale enregistre les attributs d'un *EndUser* ainsi que certaines informations supplémentaires comme la date de création du record, la date de dernière modification, etc.. Les tables de liaisons quant à elles enregistrent les données qui concernent la relation d'un *EndUser* avec un *Contact*, une *Opportunité* ou un *Lead*.

id	char(36)	utf8_general_ci		No	*None*	
name	varchar(255)	utf8_general_ci		Yes	*NULL*	
date_entered	datetime			Yes	*NULL*	
date_modified	datetime			Yes	*NULL*	
modified_user_id	char(36)	utf8_general_ci		Yes	*NULL*	
created_by	char(36)	utf8_general_ci		Yes	*NULL*	
description	text	utf8_general_ci		Yes	*NULL*	
deleted	tinyint(1)			Yes	0	
phone	varchar(255)	utf8_general_ci		Yes	*NULL*	
enduser_address_street	varchar(150)	utf8_general_ci		Yes	*NULL*	
enduser_address_city	varchar(100)	utf8_general_ci		Yes	*NULL*	
enduser_address_postalcode	varchar(100)	utf8_general_ci		Yes	*NULL*	
enduser_address_state	varchar(100)	utf8_general_ci		Yes	*NULL*	
enduser_address_country	varchar(100)	utf8_general_ci		Yes	*NULL*	

FIGURE 11.3.: Table principale représentant un *EndUser*

id	varchar(36)	utf8_general_ci		No	*None*	
enduser_id	varchar(36)	utf8_general_ci		Yes	*NULL*	
contact_id	varchar(36)	utf8_general_ci		Yes	*NULL*	
date_modified	datetime			Yes	*NULL*	
deleted	tinyint(1)			Yes	0	

FIGURE 11.4.: Table qui relie les *EndUsers* aux *Contacts*

Field	Type	Collation	Attributes	Null	Default	Extra
id	varchar(36)	utf8_general_ci		No	*None*	
enduser_id	varchar(36)	utf8_general_ci		Yes	*NULL*	
opportunity_id	varchar(36)	utf8_general_ci		Yes	*NULL*	
date_modified	datetime			Yes	*NULL*	
deleted	tinyint(1)			Yes	0	

FIGURE 11.5.: Table qui relie les *EndUsers* aux *Opportunités*

id	varchar(36)	utf8_general_ci		No	*None*	
enduser_id	varchar(36)	utf8_general_ci		Yes	*NULL*	
lead_id	varchar(36)	utf8_general_ci		Yes	*NULL*	
date_modified	datetime			Yes	*NULL*	
deleted	tinyint(1)			Yes	0	

FIGURE 11.6.: Table qui relie les *EndUsers* aux *Leads*

12. Synchronisation de comptes

12.1. Cas d'utilisation

L'utilisateur peut récuperer les **comptes** à partir de QuickBooks.

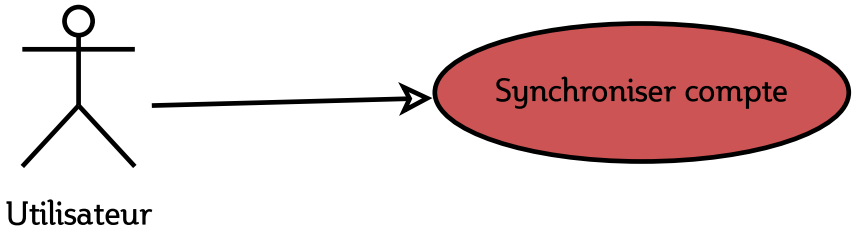


FIGURE 12.1.: Cas d'utilisation ajouté au module *Accounts*

12.2. Analyse et Conception

Il n'y a pas de nouvelle entité à créer pour cette modification, le travail du côté du module est donc assez réduit et consiste à créer une nouvelle action du controleur qui gère la synchronisation.

La grosse partie du travail se trouve du côté du serveur SOAP. Lorsque l'action du controleur est invoquée, une nouvelle requête pour récupérer tous les clients (*Accounts* sur SugarCRM, *Customers* sur QuickBooks) est ajoutée à la file de requête du serveur. Celui-ci la communiquera par la suite à QBWC comme expliqué dans la section Gestion de la communication par l'application web.

12.3. Implémentation

La première étape est l'ajout d'une action de synchronisation à un nouveau controleur, qui hérite de celui par défaut.

Listing 12.1: Accounts Controller

```
1 <?php
```

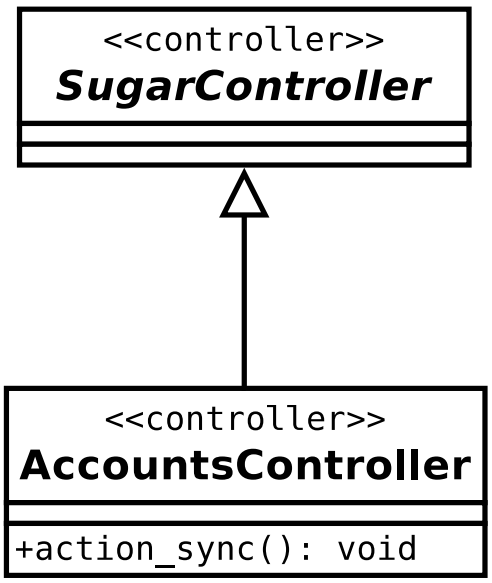


FIGURE 12.2.: Controleur du module *Accounts* augmenté pour gérer la synchronisation

```
2
3  public function action_sync() {
4    global $sugar_config;
5
6    $dsn = $sugar_config['dbconfig']['db_type'] . '://' .
7           $sugar_config['dbconfig']['db_user_name'] . ':' . $sugar_config['
                dbconfig']['db_password'];
8    $dsn .= '@' . $sugar_config['dbconfig']['db_host_name'] . '/' .
9            $sugar_config['dbconfig']['db_name'];
10   $queue = new QuickBooks_Queue($dsn);
11   $queue->enqueue(QUICKBOOKS_QUERY_CUSTOMER);
12
13   $this->view = 'sync';
14 }
15 ?>
```

Le code de la vue est trivial puisqu'il ne fait qu'afficher un message de confirmation de l'enregistrement de la demande et réenvoie l'utilisateur sur la page de consultation (`ListView`) des comptes.

Le serveur SOAP gère l'envoie de la requête et le traitement de la réponse de QuickBooks. Voici un extrait du code de cette gestion :

Listing 12.2: Extrait du traitement de la réponse

```
1  <?php
2
```

```
function _qb_customer_query_res($requestID, $user, $action, $ID,
                                $extra, &$err, $last_action_time,
                                $last_actionident_time, $xml, $idents)
{
  global $dsn, $sugar_config, $user;
  ...
  require_once(SUGAR_PATH.'/include/nusoap/nusoap.php');
  $sug_soap_client = new nusoapclient($sugar_config['site_url'].'/soap.php')
      ;
  $sess = $sug_soap_client->call('login', array(
      'user_auth' => array(
          'user_name' => $user->user_name,
          'password' => $user->user_hash),
      'application_name' => 'QuickBooks Sync'));

  $parser = new QuickBooks_XML_Parser($xml);
  $errnum = 0;
  $errmsg = '';
  if ($doc = $parser->parse($errnum, $eprrmsg)) {
    $ctms = $doc->getRoot()->getChildAt('QBXML/QBXMLMsgsRs/CustomerQueryRs')
        ;
    foreach ($ctms->children() as $c) {
      ...
      if ($account->retrieve_by_string_fields(array('name' => $acc_name)) ==
          null) {
        $new_account = array(
            'session' => $sess,
            'module_name' => 'Accounts',
            'name_value_lists' => array(
                array(
                    'name' => 'imported_id',
                    'value' => $c->getChildDataAt($ret . ' ListID')
                ),
                ...
            ),
        );
        $sug_soap_client->call('set_entry', $new_account);
        ...

  return true;
}
```

```
41
42 ?>
```

Pour chaque compte reçu, le serveur vérifie qu'il n'existe pas déjà dans la base de donnée, et l'ajoute en invoquant le serveur SOAP de SugarCRM. En effet SugarCRM met également à disposition un serveur SOAP avec lequel il est possible de communiquer pour effectuer diverses opérations sur les données du CRM.

12.3.1. Synchronisation automatique

Les **comptes** sont synchronisés automatiquement et périodiquement.

SugarCRM possède un système de *jobs*. Un *job* est une tâche effectuée périodiquement et automatiquement, sous peine d'avoir configuré le serveur web ainsi que le *task scheduler* du système hôte correctement.

Nous avons donc créé un *job* qui ajoute une requête des clients de QuickBooks automatiquement à intervalles déterminés.

13. Consultation des devis

13.1. Analyse et conception

- Un **devis** est représenté par un numéro de référence, un montant total et une liste de *produits*. - Chaque *produit* de la liste est représenté par une quantité, une désignation, une référence, un prix unitaire hors taxe, un prix total hors taxe et un prix total TTC.

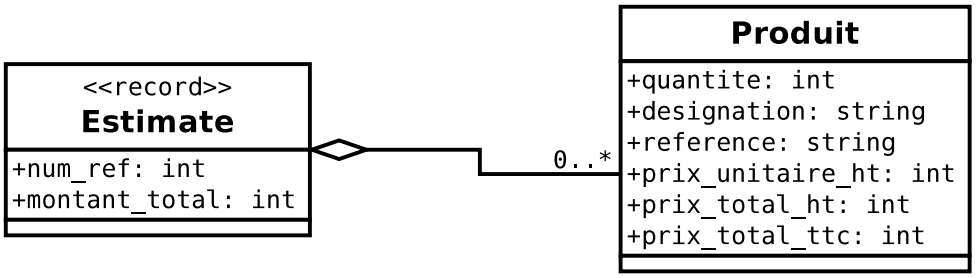


FIGURE 13.1.: Représentation d'un *Estimate* (devis)

- Un **devis** peut être associé à un *compte*.

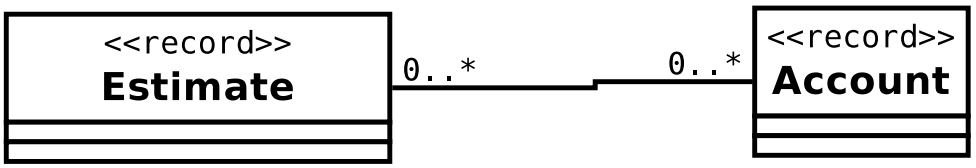


FIGURE 13.2.: Liaison d'un *Estimate* et un *Account*

13.2. Implémentation

Le module de gestion des devis n'est pas exactement un module comme les autres. Il est par exemple impossible de créer des devis à partir de SugarCRM, et la consultation des détails d'un devis se transforme en la consultation de ses produits.

Pour implémenter ces changements, il a été nécessaire de bloquer la vue de création et de modification en redirigeant l'utilisateur vers les devis présents. Nous avons également crée une nouvelle vue héritant de la vue détaillée (`DetailView`) qui permet de voir la liste des produits d'un devis.

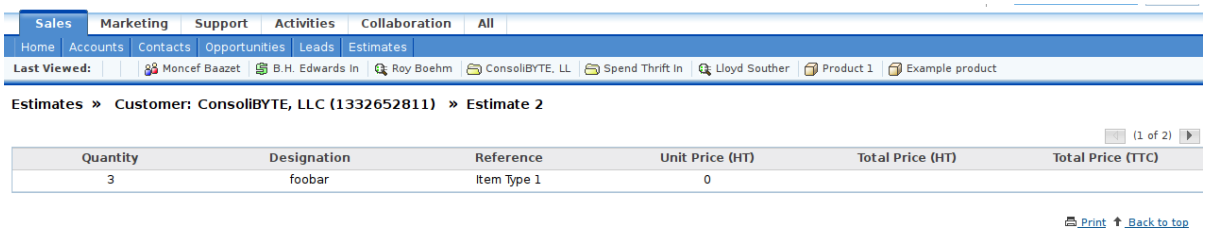


FIGURE 13.3.: Vue détaillée d'un *Estimate*

Listing 13.1: Fonction qui récupère les données des produits à afficher

```php
<?php

public function preDisplay()
{
  global $db;

  $query = 'SELECT id, est_id, designation, reference, quantity,
                   unit_price, total_price_ht, total_price_ttc
            FROM estimate_items
            WHERE est_id = ' . intval($this->bean->est_id);
  $result = $db->query($query, true, "Error fetching estimate items");
  $items = array();
  while ($item = $db->fetchByAssoc($result))
    $items[] = $item;

  $metadataFile = $this->getMetaDataFile();
  $this->dv = new DetailView2();
  $this->dv->ss =&  $this->ss;
  $this->dv->setup($this->module, $this->bean, $metadataFile, 'modules/
      Estimates/tpls/DetailView.tpl');
  $this->dv->ss->assign('ITEMS', $items);
  $this->dv->ss->assign('rowColor', array('oddListRow', 'evenListRow'));
  $this->dv->ss->assign('headerTpl', 'modules/Estimates/tpls/header.tpl');
}

?>
```

Les produits d'un devis n'ont pas de module de gestion, et ont donc dû être enregistrés dans une table à part.

Field	Type	Collation	Attributes	Null	Default	Extra
id	char(36)	utf8_general_ci		No	None	
name	varchar(255)	utf8_general_ci		Yes	NULL	
date_entered	datetime			Yes	NULL	
date_modified	datetime			Yes	NULL	
modified_user_id	char(36)	utf8_general_ci		Yes	NULL	
created_by	char(36)	utf8_general_ci		Yes	NULL	
description	text	utf8_general_ci		Yes	NULL	
deleted	tinyint(1)			Yes	0	
quantity	int(11)			Yes	NULL	
designation	varchar(255)	utf8_general_ci		Yes	NULL	
reference	varchar(255)	utf8_general_ci		Yes	NULL	
unit_price	int(11)			Yes	NULL	
total_price_ht	int(11)			Yes	NULL	
total_price_ttc	int(11)			Yes	NULL	
est_id	varchar(25)	utf8_general_ci		No	None	

FIGURE 13.4.: Table qui enregistre les produits d'un devis

13.2.1. Synchronisation

La synchronisation avec QuickBooks se fait presque de la même manière que pour les comptes. La particularité des devis est qu'il faut également récupérer les produits liés et les insérer directement dans la table concernée. Il faut également gérer la liaison avec le compte client relié à ce devis sur QuickBooks.

Listing 13.2: Gestion des produits et des comptes clients lors de la récupération d'un devis

```
1  <?php
2  function _qb_estimate_query_res(...)
3  {
4    ...
5    if ($doc = $parser->parse($errnum, $errmsg)) {
6      $ctms = $doc->getRoot()->getChildAt('QBXML/QBXMLMsgsRs/EstimateQueryRs')
           ;
7      foreach ($ctms->children() as $c) {
8        ...
9        /* Customer relationship */
10       $query = 'INSERT INTO accounts_estimates (id, account_id, estimate_id)
11                 VALUES ("' . create_guid() . '",
```

```
12                          "' . $c->getChildDataAt($ret . " CustomerRef ListID"
                                ) . '",
13                          ' . $est_num . ')';
14        $db->query($query, true, "Error inserting account relationship");
15
16        /* Items */
17        $query_values = '';
18        foreach ($c->children() as $line) {
19          if (strpos("EstimateLineRet", $line->name()) !== FALSE) {
20            $it = $line->name();
21
22            $query_values .= '("' . create_guid() . '", ' . $est_num;
23            $query_values .= ', "' . $line->getChildDataAt($it . " ItemRef
                  FullName") . '",
24                                  "' . $line->getChildDataAt($it . " Desc") . '
                                      ",
25                                  ' . $line->getChildDataAt($it . " Quantity")
                                      . ',
26                                  ' . intval($line->getChildDataAt($fit . "
                                      Amount"));
27            $query_values .= ') ';
28          }
29        }
30
31        if (!empty($query_values)) {
32          $query = "INSERT INTO estimate_items (id, est_id, reference,
33                                                designation, quantity,
                                                    unit_price)
34                    VALUES";
35          $query .= $query_values;
36          $db->query($query, true, "Error inserting estimates items");
37        }
38      }
39    }
40    return true;
41  }
42  ?>
```

14. Catégorisation des leads et contacts

14.1. Analyse et Conception

Cette catégorisation consiste en l'ajout d'une propriété aux records *lead* et *contact*. Pour le record *record* cette propriété désignera la position du contact dans l'entreprise cliente. Pour le record *lead* elle désignera le type du *lead* (hot, cold, warm).

14.2. Implémentation

L'ajout de ces propriétés se traduit par l'ajout d'une variable membre aux classes modèle de ces deux record (`Contact` et `Lead`), ainsi que l'ajout d'attributs aux deux tables principales de ces modules.

Il est nécessaire également de modifier les différentes vues pour afficher ces nouvelles propriétés, ainsi que le controleur du module pour les gérer, lors d'une sauvegarde par exemple.

La modification des vues par défaut (`ListView`, etc.) est facilitée par le *metadata framework*. Elle consiste en la modification des fichiers de configuration des différentes vues. (voir 8.6)

15. Liaisons inter-modules

Les différentes liaisons à ajouter entre modules sont toutes des liaisons many-to-many (voir 8.5.2). Le travail à faire consiste donc en la création des différentes tables qui enregistreront les informations concernant les liaisons.

Il est également nécessaire de modifier les controleurs des modules impliqués dans une liaison pour récupérer les données des modules liés, ainsi que les vues pour afficher certaines informations du record des modules liés.

Listing 15.1: Affichage des données d'un contact sur la ListView d'un EndUser

```
<?php
class EndUser extends Basic {
  ...
  public function get_list_view_data() {
    $temp_array = $this->get_list_view_array();

    $temp_array['CONTACT_NAME'] = $this->contact_disp_name;
    $temp_array['CONTACT_ID'] = $this->contact_disp_id;

    return $temp_array;
  }
  ...
}
?>
```

Listing 15.2: Enregistrement des données d'un contact relié à un EndUser

```
<?php
class EndUser extends Basic {
  ...
  function save_relationship_changes($is_update) {
    if (!empty($this->contact_id)
        and !empty($this->rel_fields_before_value['contact_id'])
        and (trim($this->contact_id) !=
             trim($this->rel_fields_before_value['contact_id']))) {
      $this->load_relationship('contacts');
```

```
10        $this->contacts->delete($this->id,
11                                $this->rel_fields_before_value['contact_id']);
12      }
13
14      parent::save_relationship_changes($is_update);
15    }
16    ...
17  }
18  ?>
```

Listing 15.3: Récupération des données d'un contact sur le module EndUser

```
1   <?php
2   class EndUser extends Basic {
3     ...
4     function fill_in_additional_detail_fields() {
5       parent::fill_in_additional_detail_fields();
6       if (empty($this->id)) return;
7
8       $query = "SELECT e_c.contact_id, c.first_name, c.last_name
9                 FROM endusers endu
10                LEFT JOIN endusers_contacts e_c ON
11                    e_c.enduser_id = '".$this->id."' and e_c.deleted = 0
12                LEFT JOIN contacts c ON
13                    c.id = e_c.contact_id and c.deleted = 0
14                WHERE endu.id = '".$this->id."'";
15      $result = $this->db->query($query, true, "Error fetching contact
            information: ");
16
17      $row = $this->db->fetchByAssoc($result);
18      if ($row != NULL) {
19        $this->contact_name = $row['first_name'] . ' ' . $row['last_name'];
20        $this->contact_id = $row['contact_id'];
21      } else {
22        $this->contact_name = '';
23        $this->contact_id = '';
24      }
25    }
26    ...
27  }
28  ?>
```

16. Conclusion

Dans ce chapitre nous avons présenté le travail effectivement effectué au sein de l'entreprise, les nouveaux composants développés et les adaptations et fonctionnalités ajoutées à SugarCRM.

Nous avons montré quelques éléments d'analyse et de conception, qui expliquent le raisonnement et la structure derrière chaque composant, ainsi que quelques parties de l'implémentation, qui montrent globalement le travail de codage réalisé.

17. Conclusion générale

Ce rapport a présenté le travail de développement effectué au sein de la société Nevo-Technologies, dans le cadre de mon stage de fin de licence à la Faculté des Sciences et Techniques.

Nous avons commencé par présenter la société accueillante, Nevo-Technologies, ses solutions et ses services. Puis nous avons décrit le sujet du stage et la problèmatique de celui-ci.

Nous avons alors procédé à la description détaillée du cahier des charges, puis de la méthodologie de développement. Nous avons enchaîné en décrivant l'infrastructure et le système déjà en place, qui a servi de berceau aux développement effectué.

Finalement nous avons présenté les modules effectivement développés et codés, avec quelques élements d'analyse et quelques parties du code.

Ce stage m'a permis de m'immerser dans le monde du travail et de l'entreprise, et de beaucoup apprendre que ce soit sur le plan personnel, technique, ou encore sur le fonctionnement interne d'une entreprise et les particularités du marché technique d'aujourd'hui.

Nevo-Technologies est une entreprise très promettante, qui en un an d'existence a réalisé un travail collosale et formidable. Il reste beaucoup de travail à effectuer sur SugarCRM pour l'adapter encore plus au travail quotidien de l'équipe et permettre à l'entreprise d'atteindre encore plus facilement ses buts.

Après discussion avec l'équipe de Nevo-Technologies, il a été prévu que nous continuerons le travail à faire sur SugarCRM. Nous avons également évoqué la possibilité de travailler sur d'autres projets à venir en rapport avec la société et ses plans futurs.

Cette expérience n'a donc été qu'enrichissante, et je remercie la Faculté des Sciences et Techniques de nous en avoir donné l'opportunité, ainsi que Nevo-Technologies pour m'avoir donné la chance d'y travailler et de m'avoir procuré les meilleurs conditions de travail.