Projet de langage object avancé : Gestionnaire de contacts

Romain Beaumont et Thomas Lourseyre

du 12 Mars 2013 **au** 2 Avril 2013

Table des matières

1 Introduction **2**
1.1 Spécifications 2
1.2 Fichiers et dossiers du projet 2

2 Hiérarchie des classes **3**
2.1 Contrôleur 4
2.2 Modèle 5
2.3 Vue 7

3 Fonctionnalités implémentés **8**
3.1 Import/Export 8
3.1.1 vCard 8
3.1.2 XML 8

4 Interface graphique **10**

5 Conclusion **11**
5.1 Résultats 11
5.2 Avancement personnel 11

1 Introduction

L'objectif de ce projet est de réaliser un gestionnaire de contact en utilisant Qt et la stl.

1.1 Spécifications

Voici les différentes spécifications qui ont été données :
- Concevoir des classes permettant d'implémenter une liste de contacts et les champs qui permettront de les caractériser
- Concevoir une interface graphique en utilisant Qt
- Implémenter l'import/export des données de l'application sous deux formats : le format vCard et un format personnalisé
- Concevoir la documentation du projet en utilisant Doxygen.

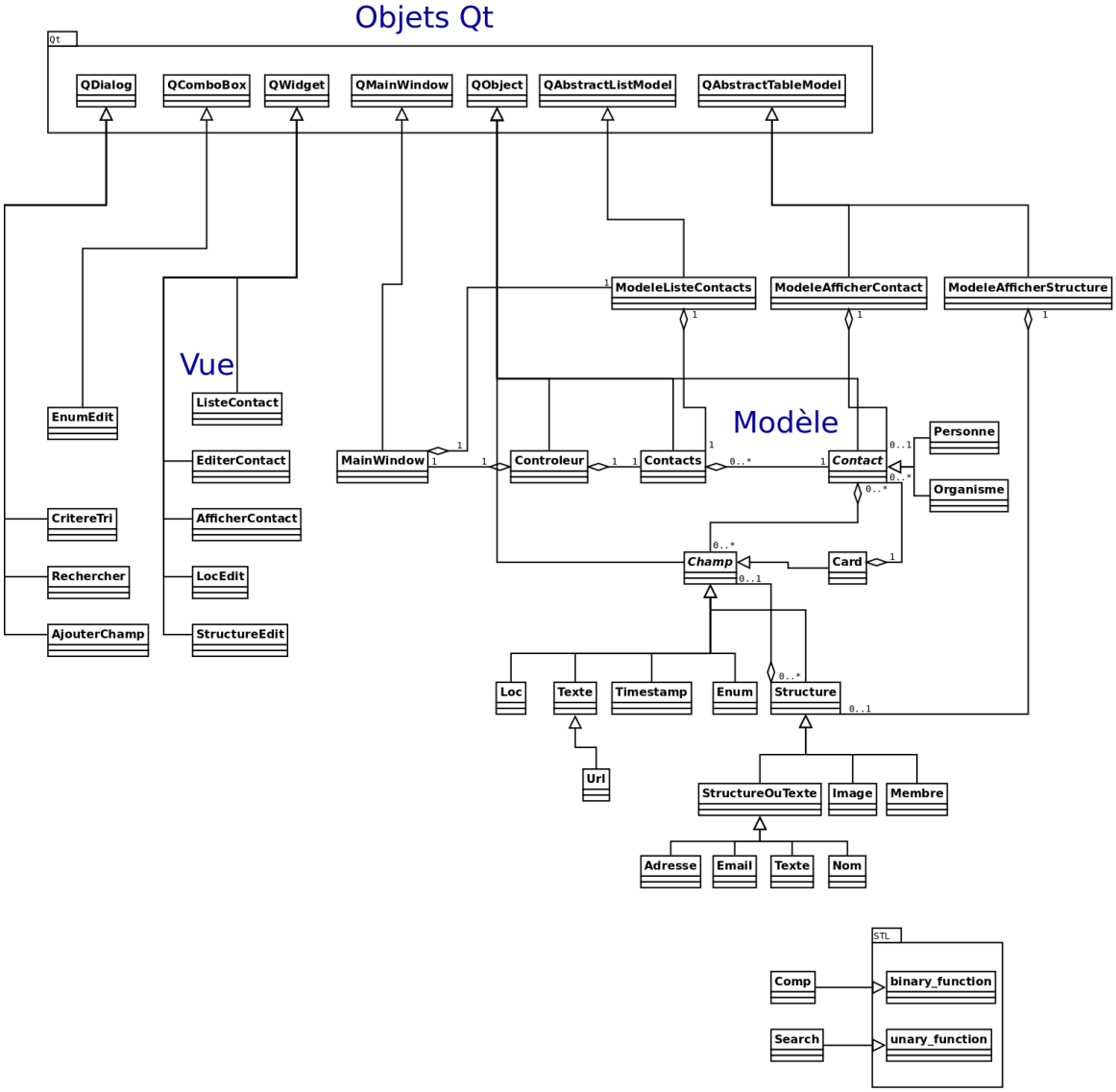
Le projet se divise en trois parties :
- La conception du modèle : trouver une hiérarchie pratique et efficace
- Implémentation des fonctionnalités : rendre le projet fonctionnel
- La conception de l'interface graphique : donner à l'utilisateur accès à toutes les fonctionnalités

1.2 Fichiers et dossiers du projet

- `ProjetLOA.pro` est le fichier de description du projet (our `qtcreator`)
- `main.cpp` contient la fonction `main` du projet
- `controleur/`, `modele/` et `vue/` sont les dossiers où sont mises les sources
- `html/` et `latex/` sont les dossier où sont enregistré la documentation
- `rapport/` est le dossier où se trouve ce rapport
- `icone/` et `dia/` sont les dossiers regroupant icones et diagramme
- `ressources.qrc` est le fichier ressource du projet
- `loadoc` est le fichier de configuration de `doxygen`

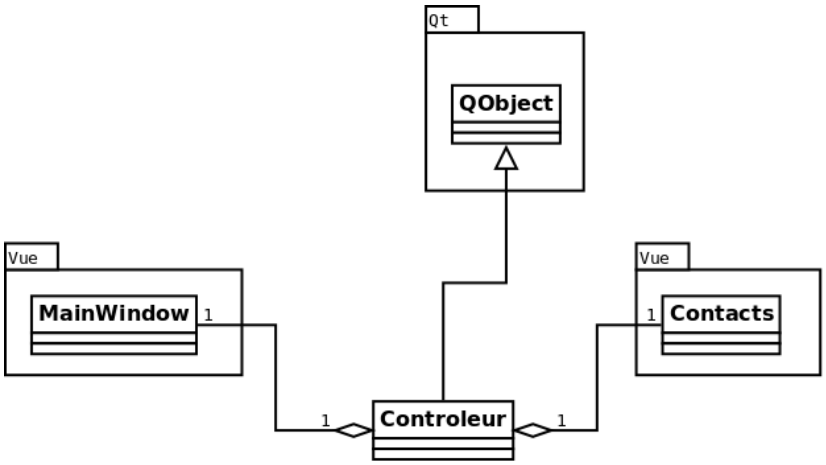
2 Hiérarchie des classes

Voici un diagramme de classe résumant les héritage et aggrégats :



La documentation du projet est disponible au format HTML (`./html/index.html`) et au format pdf (`./latex/refman.pdf`). Elle a été générée en utilisant `doxygen`.

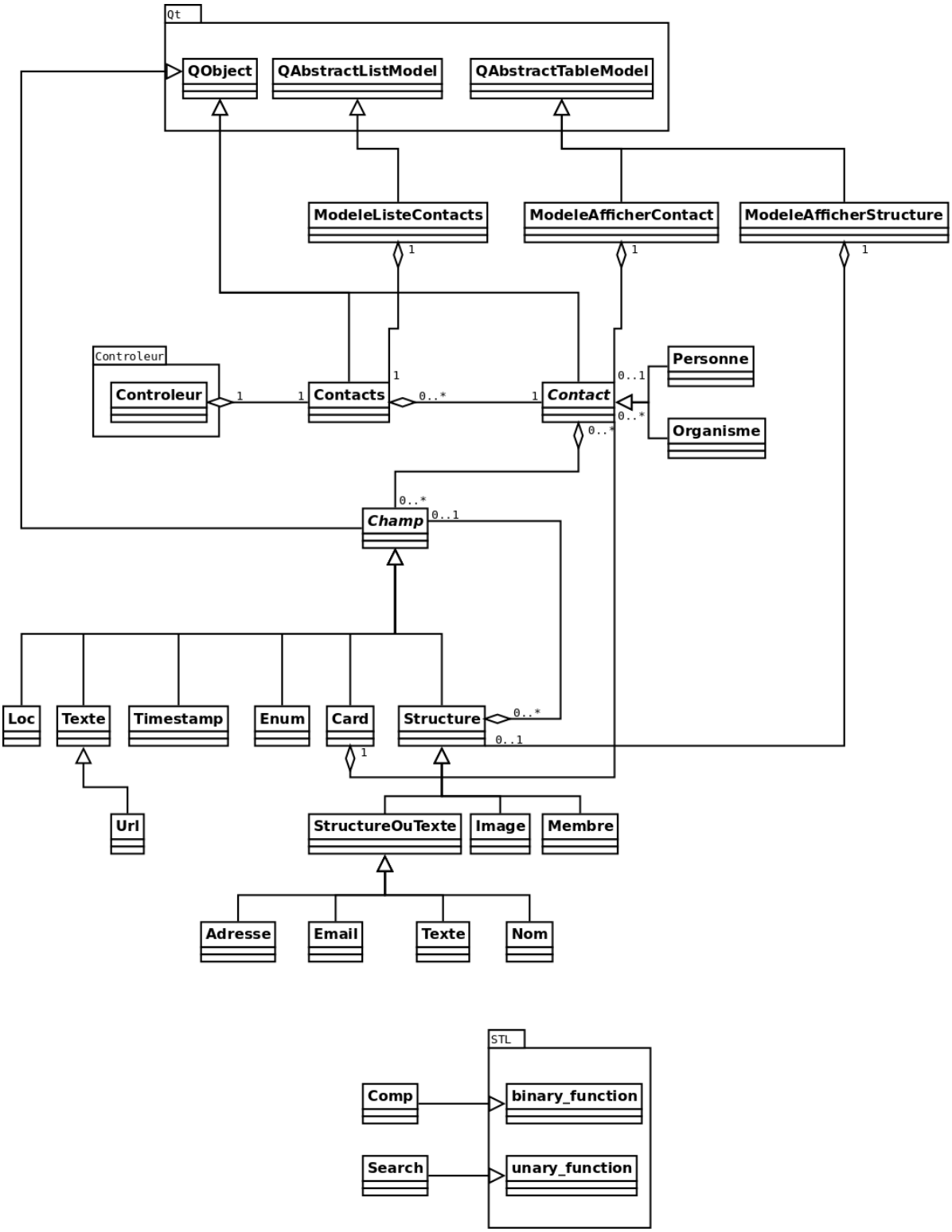
2.1 Contrôleur



La partie controleur de l'application s'occupe uniquement de lier les signaux et les slots des classes de la vue et du modèle entre elles.

L'unique classe du contrôleur instancie un objet `MainWindow` (de la partie vue) et un objet `Contacts` (de la partie modèle), et les connecte entre eux.

2.2 Modèle



Le modèle consiste en une classe `Contacts` tout en heut de la hiérarchie, qui implémente une liste de `Contact`. Cette dernière est abstraite car d'elle hérite deux templates de `Contact`, `Personne` et `Organisme`, qui ne font que définir les valeurs

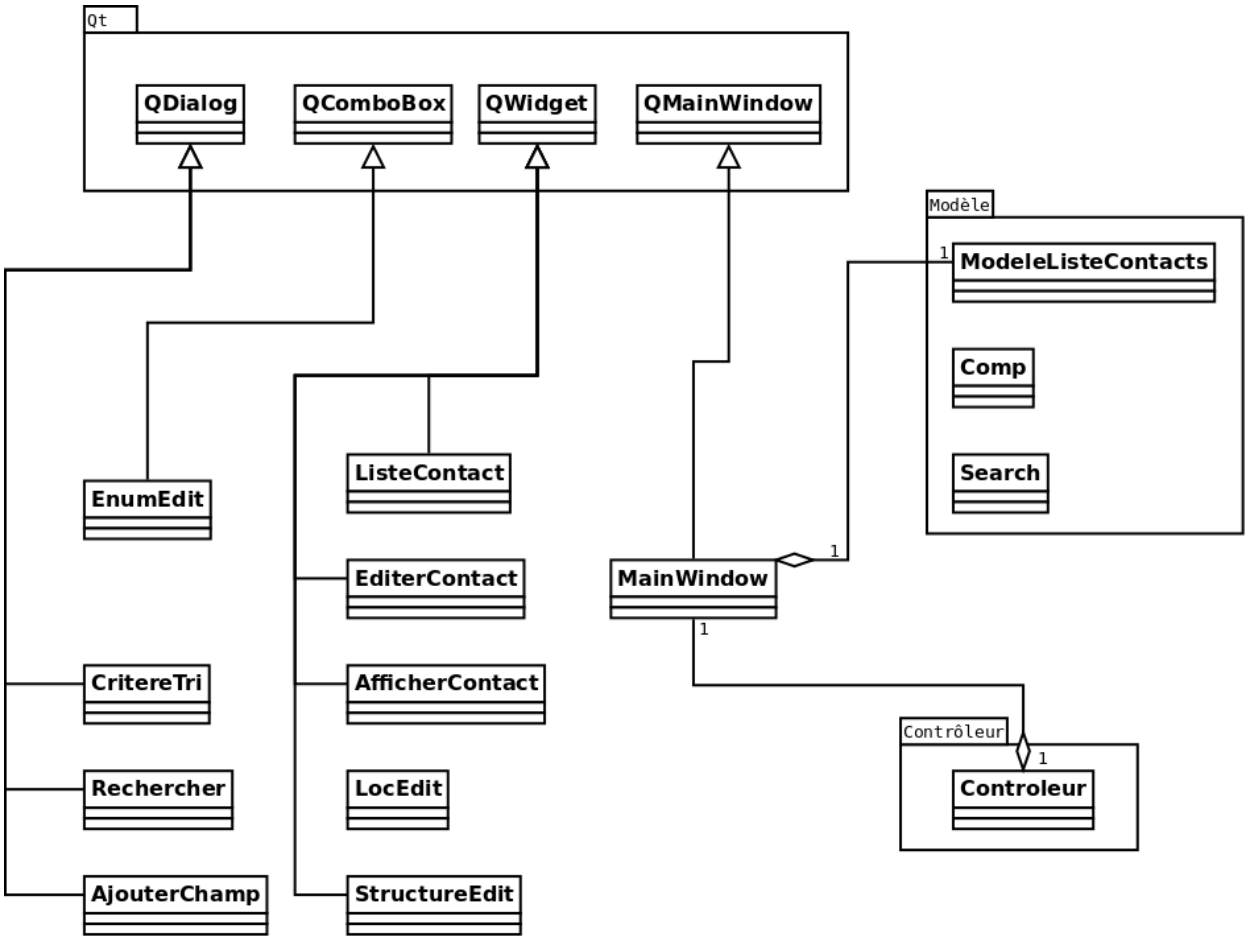
de champ possibles et par défaut.

`Contact` contient une liste de `Champ`, qui est une classe abstraite car d'elle hérite toutes les implémentations de champ géré par l'application. La plupart des filles de `Champ` utilise leur propre implémentations, hormis `Card` qui instancie un Contact (représentant soit une personne, soit un organisme) et `Structure`, qui implémente un champ avec plusieurs entrées, instanciant donc une liste de `Champ`. Elle possède deux filles correspondant à des structures particulières : `Image` et `StructureOuTexte` qui elle-même possède quatre filles : `Adresse`, `Email`, `Text` et `Nom`.

Les classes `ModeleListeContact`, `ModeleAfficherContact` et `ModeleAfficherStructure` sont des modèles Qt qui permettent à la vue de gérer correctement l'affichage (qui permet de modifier dynamiquement l'affichage).

Il y a également deux foncteurs : `Comp` et `Search`. Elles sont aggréguées dans plusieurs classes qui n'ont pas été mentionnées sur le diagramme par soucis de lisibilité. `Comp` est l'opérateur binaire de comparaison entre les `Contact`. Elle prend en entrée deux **pointeurs** de `Contact`, compare les contacts pointés selon le critère passé en argument dans le constructeur et renvoie un booléen. `Search` est un opérateur unaire qui cherche le contact dont le nom est donné dans le constructeur dans l'objet `Contact*` passé à l'opérateur.

2.3 Vue



La vue est composée d'une `MainWindow` et de six widgets `ListeContact`, `EditerContact`, `AfficherContact`, `LocEdit`, `StructureEdit` et `EnumEdit` qui servent à l'affichage et l'édition des champs et des contact dans la fenêtre.

On y trouve également les boîtes de dialogue `CritereTri`, `Rechercher` et `AjouterChamp` qui permettent à l'utilisateur d'entrée des informations lorsqu'il veut effectuer un tri, une recherche dans la liste des contacts ou ajouter un champ.

3 Fonctionnalités implémentés

Voici la liste des fonctions implémentées :
- Documentation du code dans son intégralité et utilisation de doxygen pour générer la doc
- Conception des classes permettant de représenter les contacts et les champs les décrivant.
- L'affichage de la liste des contacts
- L'ajout/suppression de contacts dans la liste
- L'affichage des détails d'un contact lors d'un clic sur celui-ci dans la liste
- La modification des champs d'un contact
- L'implémentation de tous les types de champ
- L'ajout/suppression de champs pour un contact
- La génération de champs par défaut lors de la création d'un contact
- La différentiation Personne/Organisme
- Le tri de la liste des contacts selon un critère
- La recherche dans la liste des contacts selon un critère
- L'import/export de la liste des contact en utilisant le format vCard
- L'import/export de la liste des contact en utilisant un format XML
- Deux langages sont gérés : Français et Anglais (automatiquement ajusté en fonction de la langue de la machine)

3.1 Import/Export

3.1.1 vCard

Pour ce qui est du format vCard, toutes les classes du modèle possède une méthode `toVCard()` qui renvoie une `QString` correspondant à la conversion de ce champ en chaîne de caractère respectant le format vCard.

3.1.2 XML

Quant au format XML, il exporte les champs et les contacts en utilisant `toString()`, du coup la méthode `toXML()` n'a besoin d'être implémentée que dans `Contact` et `Champ`.

Syntaxiquement, le format XML se résume à une structure semblable à celle du modèle : une suite de `<Contact> ... </Contact>` avec des balise dont le nom

est type du champ et qui possède un attribut `nomChamp` qui contient le nom de ce champ, et dont le contenu est la valeur du champ.

4 Interface graphique

L'interface graphique contient une liste de contact à gauche et l'affichage ou bien l'édition des contacts à droite. On peut aussi sélectionner plusieurs boutons dans le menu afin d'afficher une fenêtre de recherche, de tri, d'ouverture de fichier, d'enregistrement de fichier.

Chaque contact est précédé d'une image qui le représente (un logo ou une photo) si ce champ est renseigné.

La vue des contacts est composé d'une liste de champ accompagné de leur valeur et pour certains champs d'icones renseignants sur les champs (par exemple le type de téléphone : fixe,mobile, ou bien le type d'adresse : home,work)

En mode édition chaque champ est modifiable par un widget particulier : par exemple l'édition des dates peut se faire via l'affichage d'un calendrier, l'édition des adresses peut se faire ou bien via une ligne d'édition de texte ou bien via un tableau afin d'éditer chaque sous-champ séparé. Les autres type de champs sont eux aussi être édités par des widgets dédiés.

Les photos peuvent être chargées à partir d'internet ou d'un fichier local.

L'interface graphique se compose d'une barre de menu, d'une barre de raccourcis et d'un layout central.

Toutes les fonctionnalités sont accessible via la barre de menu, et quasiment toutes le sont via la barre de raccourcis.

Le layout central est divisé en trois layouts : la liste des contacts, l'affichage du contact et l'édition du contact. Ces deux derniers ne sont jamais visibles simultanément.

Options des menus : Nouvelle liste (pour réinitialiser la liste de contact), Nouvelle personne, Nouvel organisme, Ouvrir, Enregistrer, Trier, Rechercher, Arrêter la recherche, Supprimer contact, Quitter.

5 Conclusion

5.1 Résultats

Toutes les fonctionnalités demandées sont réalisés.

Le modèle implémenté permet facilement d'intégrer d'autres types de champ en créant d'autres classes héritant de `Champ` ou d'une de ses filles/petites-filles.

5.2 Avancement personnel

Ce projet a permis d'apprendre mieux la partie modèle/vue de Qt.

Il nous a permis de mieux connaître la librairie Qt et qtcreator.