Université de Nantes
UFR des Sciences et Techniques
Master ALMA 1

TP Génie Logiciel - MyCalendar

Guillaume Charon
Jerôme Pages

Nantes, le 5 décembre 2012

Sommaire

Introduction

Ce document a pour objectif de présenter la conception de l'application MyCalendar. Cette application, écrite en C++ avec le framework Qt4, présente une interface graphique correspondant à un emploi du temps semaine par semaine modifiable localement. L'utilisateur peut s'il le souhaite créer, modifier, ou supprimer des évènements.

Autour de ces fonctionnalités basiques s'attachent deux autres fonctionnalités permettant à l'utilisateur d'intéragir avec certains services en ligne pour importer ou exporter des évènements depuis Internet vers l'application locale et vice-versa.

Petit rappel des spécifications des exigences logicielles à partir desquelles la présente conception est basée :

- Architecture modulaire permettant une maintenance facile de l'application ;
- Interface graphique (GUI [1]) *user-friendly* par semaine ;
- Respect de la charte graphique GNU/Linux ;
- Édition locale de l'emploi du temps (ajout, modification, suppression) par clics intuitifs sur la GUI ;
- Gestion des conflits (que faire lorsque l'utilisateur essaie d'ajouter un évènement sur un créneau non libre ?) ;
- Sauvegarde locale des évènements ;
- Exporter l'emploi du temps vers un service en ligne ;
- Récupérer l'emploi du temps depuis ce même service en ligne ;
- Protection de l'emploi du temps distant avec des identifiants de connexion ;
- Utilisation du protocole REST pour la communication avec ce service en ligne ;
- Communication non bloquante et temps de réponse de l'ordre de la seconde ;

Une autre exigence est concernée par la conception de l'application puisque nous avons eu le temps de l'implémenter : la récupération d'un emploi du temps universitaire depuis le site de gestion d'emploi du temps de l'Université de Nantes. Cependant, aucun autre format d'emploi du temps n'a été supporté, cette exigence ayant été elle aussi reportée.

1. Graphical User Interface

1 Conception générale

Afin de répondre à la première exigence, c'est-à-dire concevoir l'application à partir d'une architecture modulaire, nous avons utilisé le pattern architectural MVC (pour Model-View-Controller). Cette architecture nous permet de diviser l'application en plusieurs sous-systèmes :

1. L'interface graphique utilisateur ;
2. Le modèle, contenant tous les évènements et proposant des méthodes pour agir dessus ;
3. Les interfaces de communication, pour interagir avec différents services d'emploi du temps en ligne ;

Le contrôleur joue quant à lui le rôle d'arbitre entre tous ces systèmes.
Exemples :

La vue demande la sauvegarde des évènements dans un fichier local : le contrôleur reçoit le signal correspondant et traite la demande.

La vue demande l'export des évènements vers un service en ligne : le contrôleur reçoit ce signal, vérifie la connexion à ce service (est-ce que l'utilisateur est authentifié ?) et appelle la bonne interface de communication.

Un conflit est détecté lors de l'ajout d'un nouvel évènement : le contrôleur traite ce conflit, soit en demandant à l'utilisateur que faire, soit en appliquant un comportement par défaut par exemple.

Ci-dessus, un diagramme de classe généraliste présentant ce système dans sa globalité. La conception de chaque sous-système est présentée dans les prochains chapitres.

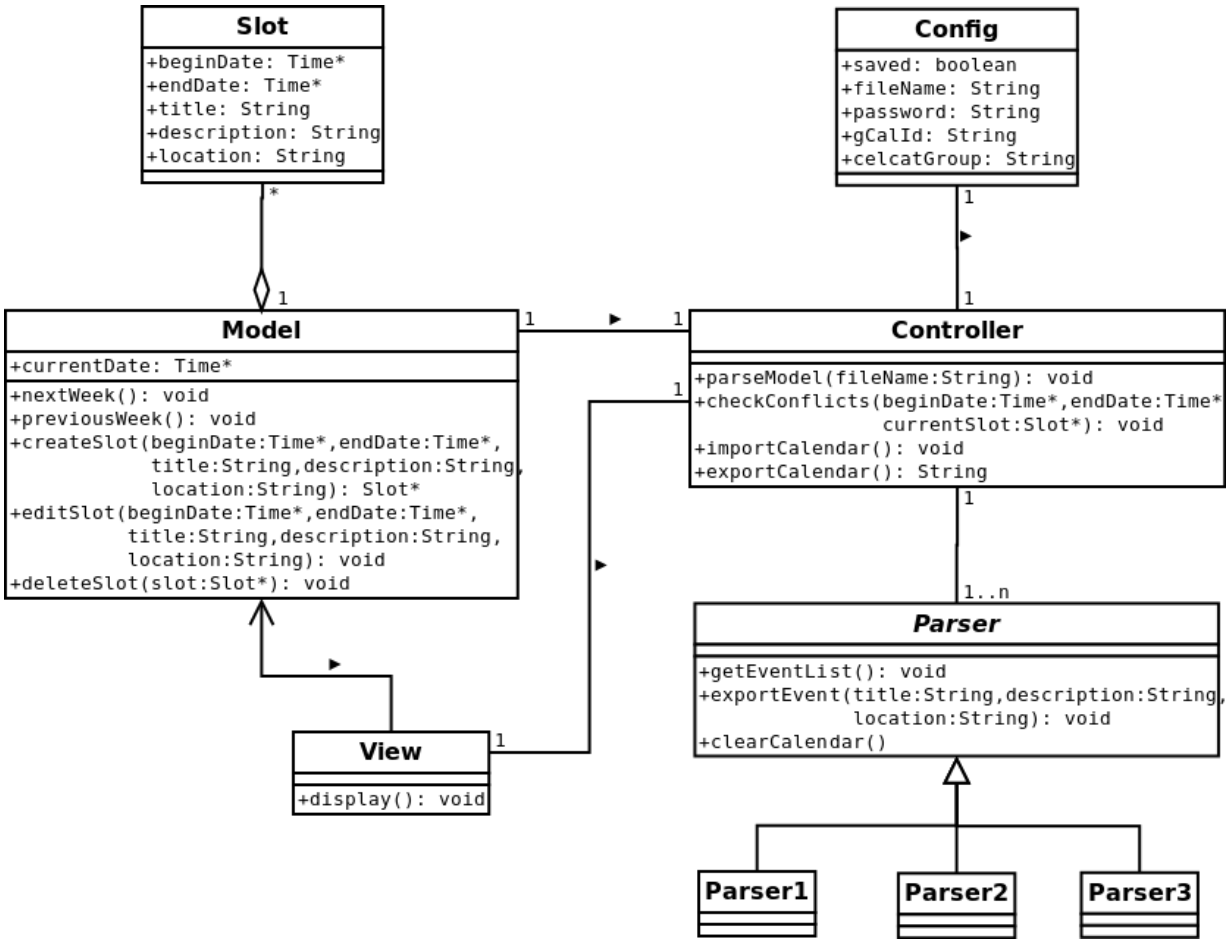

FIGURE 1.1 – Diagramme de classe général

2 Conception du modèle

3 Conception des interfaces de communication

Ce sous-système de l'application MyCalendar constitue, derrière le modèle local, son centre névralgique. En effet, c'est toute cette partie de l'application qui va gérer les communications avec l'extérieur, que ce soit pour récupérer un calendrier distant ou pour exporter en ligne les données locales.

Il existe deux types de communications : les communications entrantes (import d'un calendrier vers les données locales) et les communications sortantes (export des données locales vers un calendrier distant).

Premièrement, l'application est sensée, à terme, pouvoir gérer plusieurs types de services en ligne. Cela signifie qu'il faut non seulement permettre aux développeur de pouvoir ajouter la prise en charge d'un nouveau service facilement, mais aussi proposer aux utilisateurs une interface lui proposant de choisir lui-même quel service il souhaite utiliser pour l'import comme pour l'export.

Deuxièmement, un service de calendrier propose l'import de données en même temps que l'export de données. Cela ne sera pas vrai pour un emploi du temps universitaire, qui est sensé ne pas pouvoir être modifiable par un client quelconque, mais cela le sera normalement pour tout autre service distant.

Partant de ces deux conclusions, nous avons décidé de partir sur la conception d'une classe abstraite *Parser* déclarant toutes les méthodes permettant l'import et l'export de données. Pour implémenter un nouveau service distant, il faut ensutie étendre cette classe abstraite et de définir les méthodes qu'il faut. Enfin, pour l'utiliser, il suffira de modifier le controller (que nous avons vu précédemment) pour instancier ce parseur et utiliser les méthodes que l'ont veut.

Dans l'application *MyCalendar*, seulement deux parseurs ont été définis.

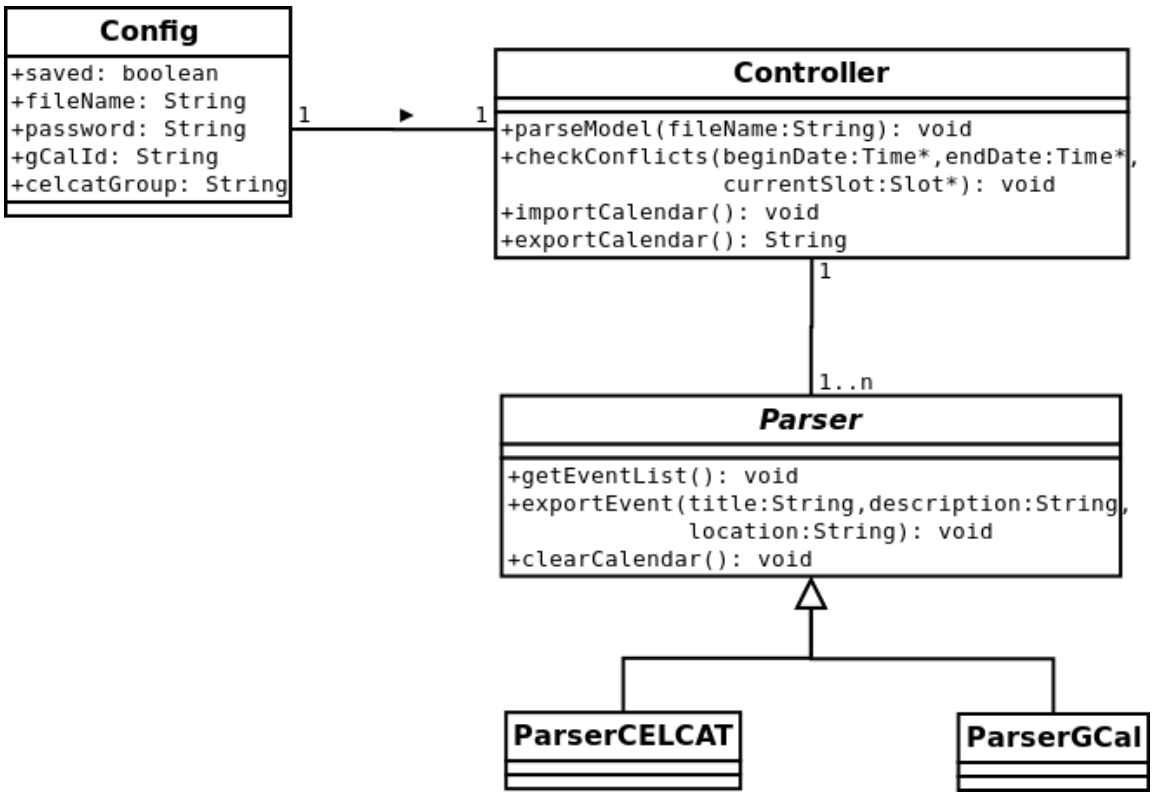

FIGURE 3.1 – Diagramme de classe des interfaces de communication

un seul service en ligne pris en compte, mais appli conçue de manière a pouvoir en ajouter facilement.

4 Conception de l'interface utilisateur