Université Paris 7 - Paris Diderot

Programmation Objet Concepts Avancés

eVote

Votalisk

Auteurs :
Josian Chevalier
Vladislav Fits
Quỳnh Nga Nguyễn
Thomas Salmon
Mohamed Skhiri

Enseignants :
M. Yann Régis-Gianas
M. Stefano Zacchiroli

16 janvier 2015

Table des matières

Partie 1

Interprétation du Sujet

L'objectif de ce framework SCALA est de fournir un environnement de gestion électoral, il ne s'agit donc que du noyau d'une application.

L'intérêt de cette application est donc d'offrir la possibilité d'étendre ses fonctionnalités afin d'implémenter n'importe quel type de vote. Pour cela, il faut donc offrir un système dont les bases seront les plus abstraites possibles. Cependant le système devra déjà modéliser les concepts d'un système électoral afin que l'implémentation d'un système électoral précis se passe de l'implémentation de tous les concepts immuables d'un système électoral.

Un système électoral peut s'appliquer à plusieurs échelles, que ce soit pour une élection au sein d'un comité restreint, ou pour un événement à échelle nationale. Il est donc essentiel que ce framework supporte le passage à l'échelle.

De plus, il existe de nombreux modes de scrutin, ce système doit donc permettre de mener à bien des élections de différentes manières. Le nombre de tours, le traitement des suffrages, le mode de calcul des résultats, sont autant de paramètres qui doivent pouvoir être changés en gardant le même code pour le noyau de l'application.

Le type d'électeurs, de candidats, le nombre de candidats élus, et la forme que revêt le suffrage sont également des paramètres susceptibles de changer

d'un système d'élection à l'autre.

Notre framework devra donc encapsuler chacun de ces concepts afin de permettre de les moduler indépendamment, et les traiter de manière suffisamment abstraite pour ne pas avoir à changer le code de base dans le cas ou l'on ajouterais de nouveaux modes de scrutins.

Partie 2

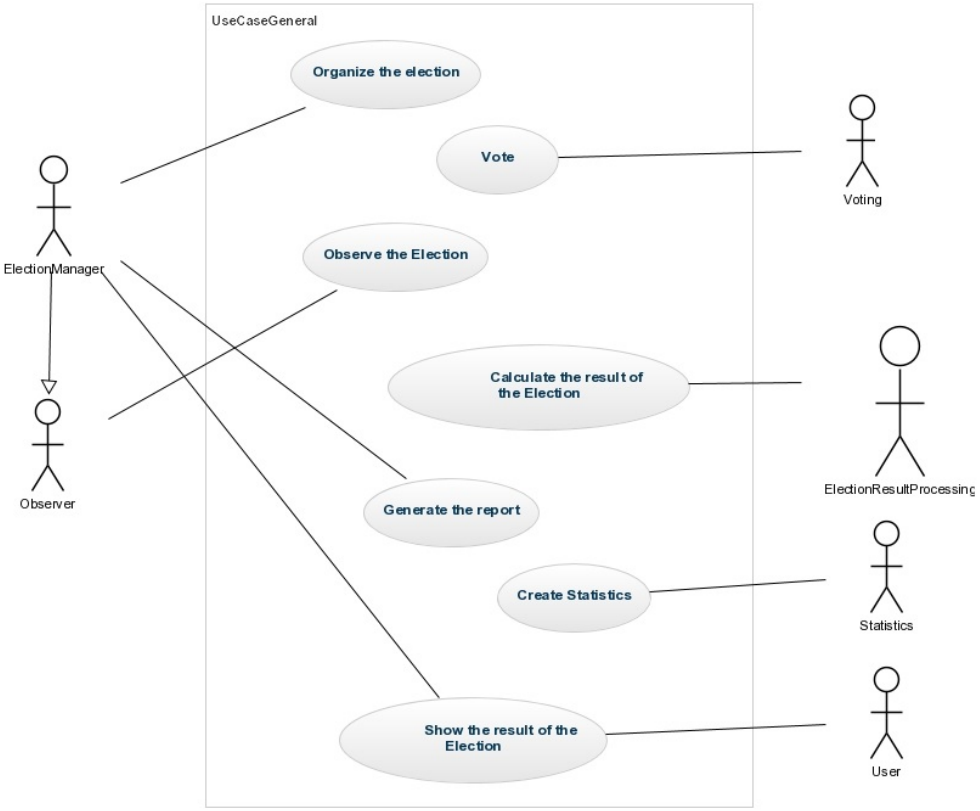
Spécifications fonctionnelles

2.1 Exigences fonctionnelles

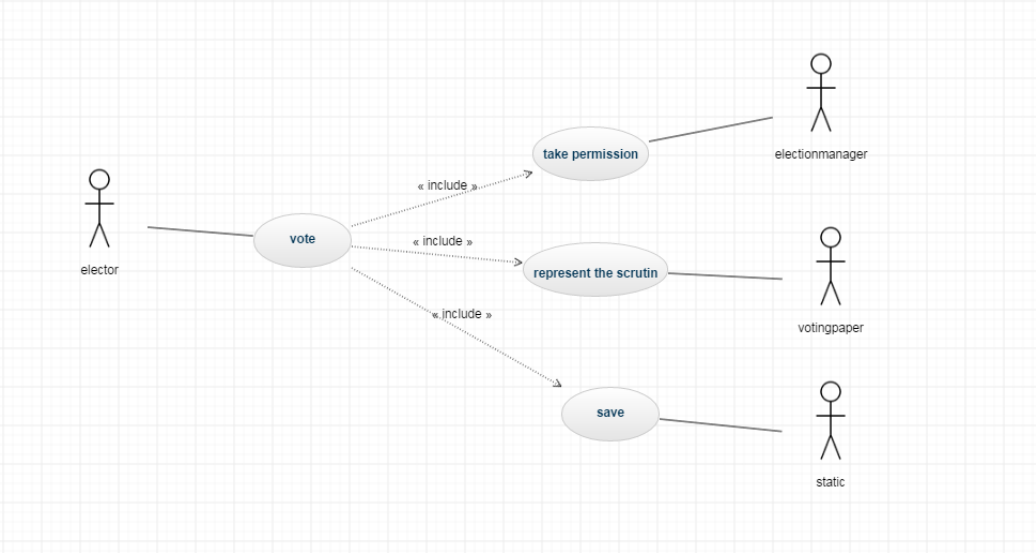
- Le système doit pouvoir servir de base à l'implémentation de n'importe quel système électoral.
- Le système doit être facilement intégrable dans les systèmes numériques d'élections existantes.
- Le système doit permettre d'implémenter des élections à plusieurs tours.
- Le système doit pouvoir implémenter des élection dans différents types de circonscriptions.
- Le système doit permettre de d'extraire des statistiques concernant les élections.
- Le traitement de données d'un scrutin peut se passer en temps réel, ou lorsque tous les scrutins ont été soumis.
- Le système doit pouvoir traiter les scrutins invalides.

2.2 Cas d'utilisation

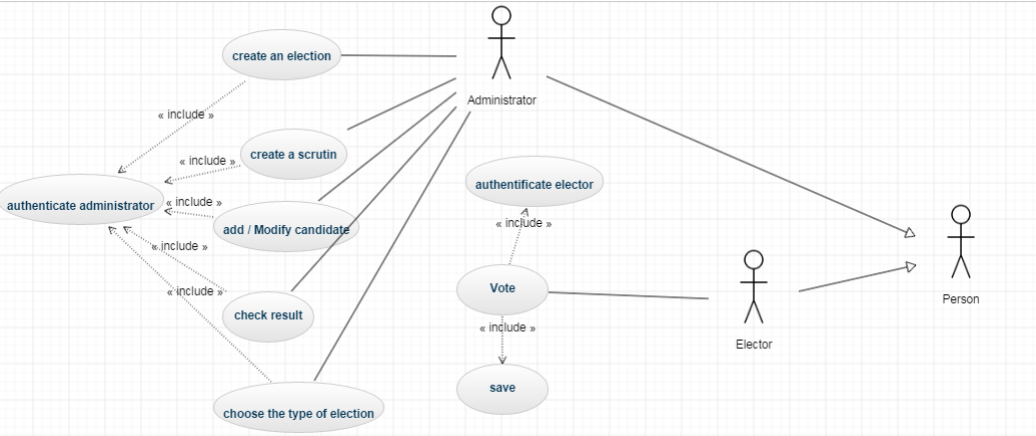
2.2.1 Diagramme de cas d'utilisation général



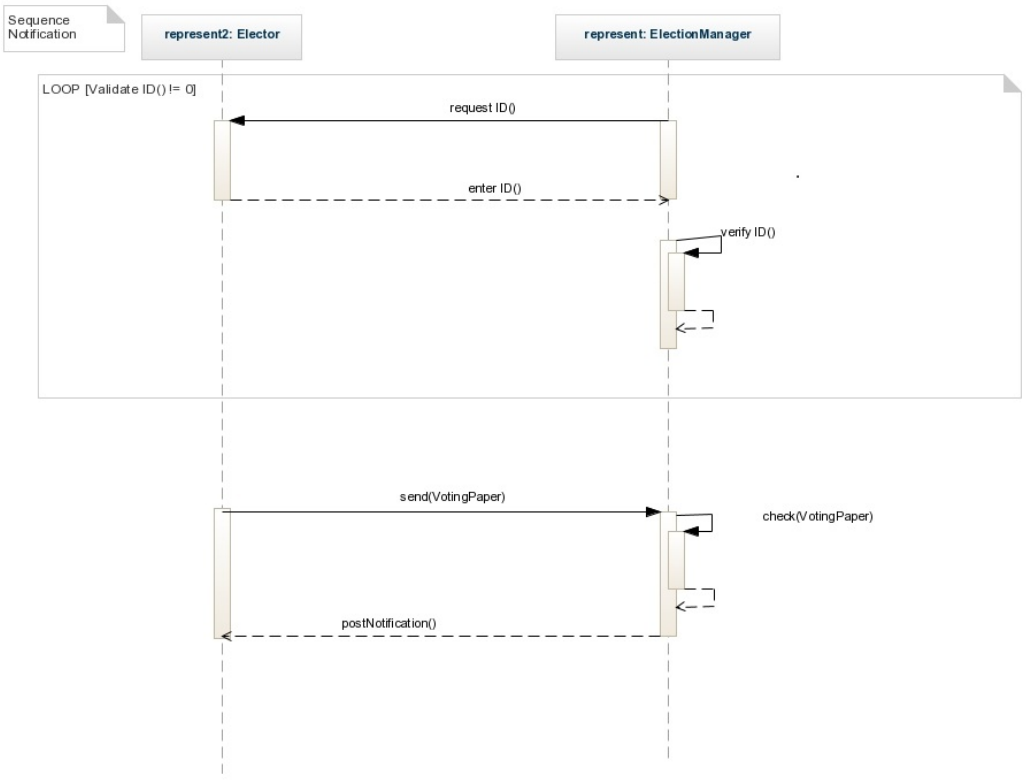
2.2.2 Vote



2.2.3 Création d'une élection



2.3 Diagramme de séquence



Partie 3

Concepts

Afin de mener à bien la réalisation de ce framework, il est nécessaire de s'intéresser aux différents concepts qui prennent part à un système électoral, et de définir lesquels sont susceptibles de changer d'un système à un autre.

3.1 Concepts utilisés

Nous décrirons ici les principaux concepts que nous avons utilisés pour modéliser le problème.

3.1.1 Candidat

Le concept de candidat est l'un des concepts les plus évidents du système électoral. Aucune élection ne saurait avoir lieu sans candidat, cependant la notion de candidat peut fortement varier d'une élection à une autre. Un candidat peut ainsi être une personne, un groupe de personnes, ou bien tout simplement n'importe quel type d'entité morale. Notons que l'on peut également décliner la notion de candidat afin d'utiliser le framework pour n'importe quel type de vote. Dans ce cas un candidat pourra être une idée, une option, une création, une loi, etc...

3.1.2 Électeur

Tout comme le concept de candidat, le concept d'électeur est indispensable à un système électoral, mais peut revêtir plusieurs formes. Bien que son rôle ne change pas, un électeur peut être une personne, un groupe de personnes, un parti, ou toute entité morale.

3.1.3 Circonscription

De nombreuses élections divisent les suffrages par circonscriptions, afin d'élire des candidats différents dans chaque circonscription. On peut également utiliser la notion de circonscription lorsque l'on élit un candidat commun, à des fins statistiques.

3.1.4 Suffrage

Un suffrage est un concept nécessaire à toute élection, mais dont la forme diffère d'une élection à l'autre. Certains systèmes électoraux limiteront chaque suffrage à un seul candidat, d'autre permettront une liste, dont l'ordre peut être ou ne pas être important.

3.1.5 Résultats de candidat

L'ensemble des suffrages définit un résultat différent pour chaque candidat. Ce sont ces résultats qui seront comparés afin de définir le ou les vainqueurs des élections. De plus, ces résultats pourront être utilisés à des fins statistiques.

3.1.6 Tours

Une élection peut se dérouler en plusieurs tours, ou parfois être composée de plusieurs élections. Par exemple, les élections présidentielles américaines sont composé de l'élection des grands électeurs et de l'élection du président par ces électeurs.

3.2 Invariants du système

Afin de garder un noyau abstrait le plus complet possible, il est important de bien identifier les éléments qui ne changeront pas d'un système électoral à l'autre, au sein des concepts abordés.

3.2.1 Candidats et électeurs

Si les candidats et les électeurs peuvent changer de type d'une élection à une autre, leur rôles ne changent jamais : les électeurs votent pour des candidats. Ainsi, si leurs attributs peuvent varier, on peut définir des ensembles de méthodes qui seront utilisées pour chaque type d'élections.

3.2.2 Circonscription

Le concept de circonscription peut être généralisé à tout type d'élection si on considère qu'une circonscription est une composition de circonscription. Ainsi cela permet de faire des votes au sein d'une circonscription composée de plusieurs autres circonscriptions, au sein d'une circonscription non composée, ou sans circonscription.

3.2.3 Résultat de candidat

Si la forme que prennent ces résultats change, une chose ne change pas : c'est ces résultats que permettent de définir les suffrages, et les résultats des élections sont définies par ces résultats.

3.2.4 Tours

Une solution pour abstraire la notion de tour de manière commune à toute type d'élections est de voir une élection comme une composition d'élections, ou un tour est considéré comme une élection à part entière. Ainsi, une élection peut n'être composée que d'elles même, ou de plusieurs élections qui seront vues comme ses tours, et dont les paramètres d'entrées de chacune dépendent des résultats de celle qui la précède.

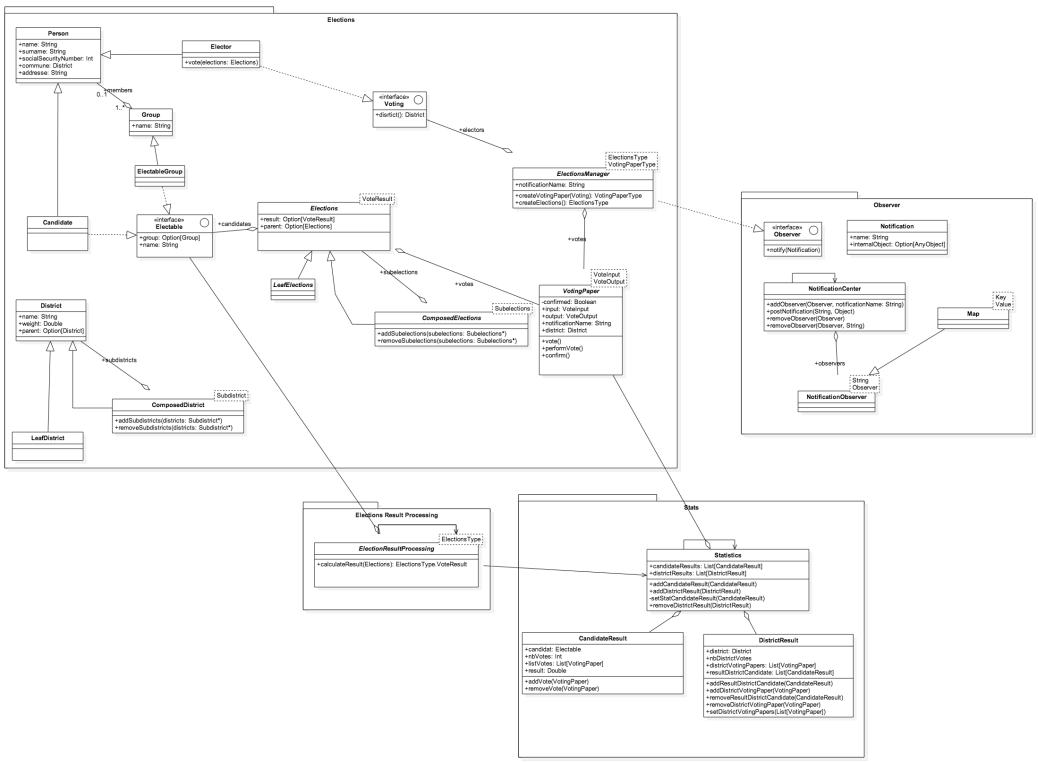
3.2.5 Suffrage

Si la composition du suffrage peut varier d'un système électoral à un autre, son traitement reste le même : un électeur soumet son vote au travers d'un suffrage, et le parcours de l'ensemble des suffrages permet de définir les résultats de chaque candidat.

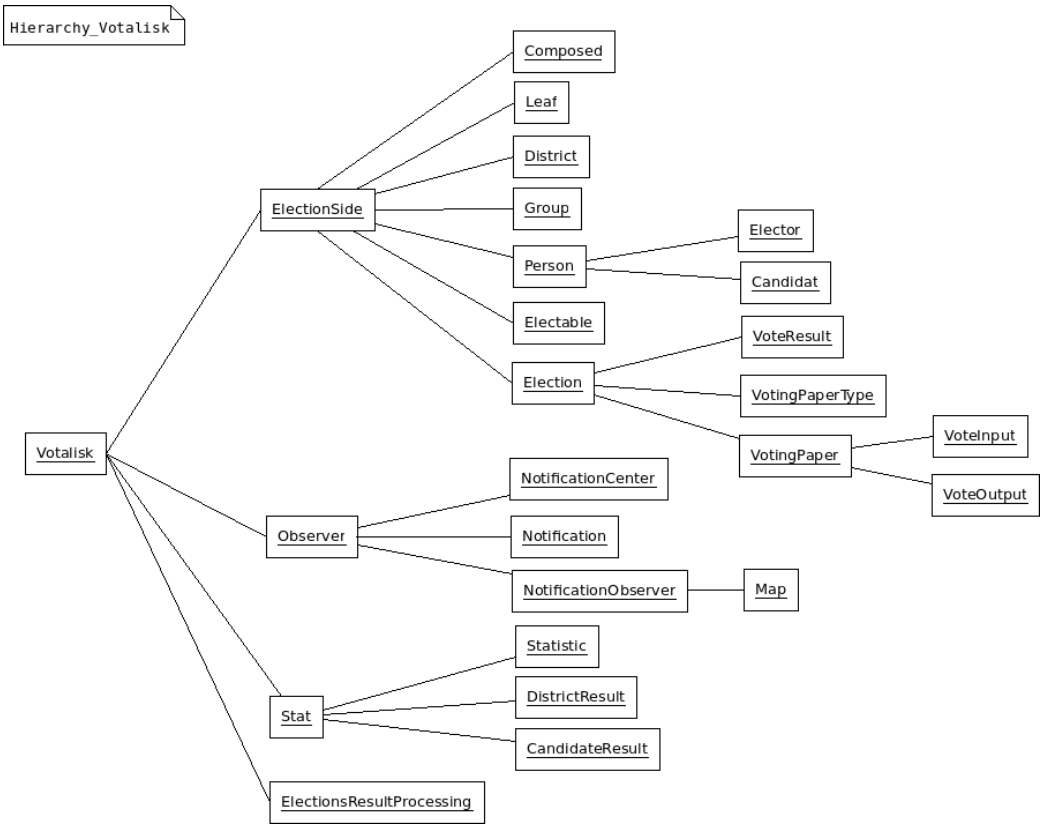
Partie 4

Architecture

Nous avons fait notre conception selon une approche objet standard, et non spécifique à SCALA. Après avoir identifié les principaux concepts et invariants du systèmes, nous avons donc établi la modélisation suivante :



De cette modélisation nous pouvons déduire un diagramme de hiérarchie, dont nous utiliseront les composants afin de décrire le système.



4.1 ElectionSide

Ce module contient les classes nécessaires afin de mettre en place une élection et de recueillir les votes des électeurs.

4.1.1 Voting et Electable

Afin de pouvoir abstraire les électeurs et les candidats du type d'électeur/candidat, tout en gardant les méthodes qui nous permettront de les utiliser dans tous les types de systèmes électoraux, nous avons fait de ces

concepts des interface (ou des traits en SCALA). Nous avons créé également une classe Person, ainsi qu'une classe Group, composée de personnes, dont des classes filles implémentent ces interfaces. Nous pouvons cependant représenter n'importe quel type d'entité de cette manière afin d'en faire un candidat ou un électeur.

4.1.2 Voting Paper

Les suffrages sont représentés par la classe VotingPaper. Les différents modes de scrutins autorisant différents types de suffrages, nous avons choisi d'abstraire les types d'entrée et et sortie, ce qui nous permet de modifier le contenu du scrutin sans rien changer à son mode de fonctionnement (pour la soumission de scrutin, etc.) ou aux informations qui y sont liées. La fonction associée à la soumission d'un suffrage est également abstraite, afin de pouvoir facilement ajouter un comportement au moment de cette soumission. Cela peut s'apparenter à un pattern Strategy. Par défaut lors de la validation d'un suffrage, aucune information sur le votant n'est envoyée, mais nous avons la possibilité d'envoyer des informations personnelles dans une extension, dans le cas ~~du vote Russe~~ d'un vote publique, comme au sein d'un parlement, ou les votes sont connus.

4.1.3 ComposedElections

Nous avons appliqué un pattern composite au concept d'élection, afin de permettre de créer au travers de la classe ComposedElections une élection à plusieurs tours, chacune étant un objet Election, tous les tours suivant le premier dépendant des résultats de celui qui le précède.

4.1.4 ElectionManager

Cette classe est une factory qui créé des VotingPaper pour les électeurs et qui créé un objet Election lorsque l'on souhaite obtenir les résultats pour

les suffrages exprimés. C'est cette classe qui fait office d'interface pour l'application qui embarque notre framework

4.1.5 District

Un pattern composite a été applique également au concept de circonscription (ou District), afin de pouvoir utiliser des Districts composés de plusieurs autres Districts.

4.2 Observer

Ce module est l'implémentation d'un pattern observer, utilisé ici pour notifier ElectionManager lorsqu'un VotingPaper est validé.

4.3 Stats

C'est au sein du module de statistiques que les suffrages sont analysés afin d'en extraire les résultats de chaque candidats, et les résultats des différents districts.

4.3.1 Statistics

Ce singleton reçois les VotingPapers qui ont été validé, et met à jour les objets CandidateResult et DistrictResult en conséquence.

4.3.2 CandidateResult

Chaque candidat a un objet CandidateResult qui lui correspond. Il est doté d'une fonction qui lui permet d'actualiser les résultats du candidat auquel il correspond en fonction des VotingPapers qu'il reçoit. Le type de don-

née de sortie de ces suffrages variant d'une élection à l'autre, et la manière de les interpréter également, il s'agit d'une fonction abstraite.

4.3.3 DistrictResult

Chaque district composant le district principal dans lequel a lieu l'élection a des résultats qui lui sont propres, et qui dépendent des résultat de chaque candidat par rapport aux suffrages exprimés par les électeurs qui dépendent de ce district.

4.4 ElectionResultProcessing

Ce singleton calcule les résultats finaux des élections à l'aide des divers résultats du module stats, afin de les transmettre à un objet Election.

Partie 5

Extentions

5.1 Implémentations concrètes

Afin de facilité l'implémentation concrètes des différents types de systèmes, nous avons spécialisé la classe de calcul des résultats afin de fournir à l'utilisateur une liste de classe offrant déjà des implémentations concrètes des principaux types de calcul des résultats. Ainsi donc, l'utilisateur peut utiliser la classe 'ElectionResultProcessing' s'il souhaite implémenter sa propre façon de calculer le gagnant à partir des votes, ou alors utiliser ses classes filles offrant des implémentations concrète pour le calcul par calcul majoritaire, proportionnel, ou mixte.

5.2 Interface Graphique

Une interface graphique peut être générée automatiquement pour les bulletins de votes, permettant de pouvoir cocher une seule option ou plusieurs, selon le type de vote, plutôt que de voter par la ligne de commande. Enfin, un bouton permet de valider le vote et de laisser la place au votant suivant, et ainsi de suite jusqu'à l'affichage du résultat final.

5.3 Récépissé

Nous avons chercher à ajouter à notre framework la possibilité d'ajouter un système de récépissé anonyme. Nous avons pu inscrire la plus grande majorité de ce système au sein du framework, afin d'offrir à un utilisateur du framework tous les outils nécessaires à l'implémentation concrète de ce système. Afin de garder l'anonymat des récépissés, plutôt que d'enregistrer au sein d'une classe fille de 'Voting Paper'des informations permettant de remonter à l'émetteur du vote afin de lui faire parvenir une confirmation, nous avons décidé d'utiliser le module Observer déjà en place. Ainsi, nous utilisons une classe 'VotingPaperReceipt', dont l'usage varie un peu : lors d'une implémentation concrète le votant devra être passé en paramètre de la fonction 'confirm', lors de son vote, et cette fonction enregistrera ce votant comme observer du traitement du bulletin de vote. La définition dudit votant dépend de l'implémentation concrète faite par un utilisateur, il suffit que le 'Voting' ai le trait 'Observer', et une fonction notify qui permettra de traiter la confirmation de la prise en compte du vote selon ce que l'on souhaite en faire. Notons que si l'utilisateur souhaite pouvoir recalculer les résultats à partir des récépissés, la fonction notify devra également enregistrer des 'Receipt' dans la 'ReceiptList', afin de pouvoir recalculer en utilisant ceci. Un Receipt n'est autre qu'un bulletin de vote déjà validé.

Une implémentation concrète du système de récépissé se retrouve dans le module DoodleReceipt, qui est une version légèrement modifiée de Doodle.

Ainsi, une bonne utilisation de framework nous permet d'ajouter cette fonctionnalité en un minimum d'efforts sous une forme abstraite, afin de laisser plus de liberté quand à l'utilisation faite par l'utilisateur.

5.4 Doodle

Le module Doodle permet à un votant de choisir un ou plusieurs créneaux horaire. Les extentions 'Single' et 'Multiple' permettent de choisir si l'on peut

choisir un seul ou plusieurs créneaux. L'anonymat du vote peut également être altéré au travers du trait 'Public'. Enfin, une option 'IfNeedBe' permet de choisir un créneaux en indiquant que l'on pourra venir si notre présence est indispensable. Ces fonctions sont implémentées au travers de traits pour garder une bonne modularité. Afin de laisser la possibilité de choisir un ou plusieurs créneaux horaires, nous avons choisis de tirer partie de l'opportunité offerte par notre framework de définir les types d'entrée des bulletins de vote, les types de sortie, et le traitement de leur résultat. Ces choix de conception ont grandement simplifié l'implémentation du module Doodle.

Le module Doodle ne va pas jusqu'au calcul du résultat mais implémente toute la partie de vote, tout en respectant la modularité du framework.

5.5 Extentions envisageables

- Nous pourrions pour commencer implémentes les méthodes d'élections de différents pays et organisations (France, Union Européenne, etc)
- Nous avons également la possibilité d'ajouter de nouveaux types de vote (par exemple par préférence, etc). Le bulletin de vote étant une structure abstraite, et la façon dont les suffrages sont traité étant abstraite également au sein du module Stats, nous pouvons facilement mettre de nouveaux types de vote en place.
- Il est possible d'utiliser ce framework pour tout type de vote grâce au caractère abstrait des candidats. Ainsi, il est possible de remplacer des candidats par des concepts, des idées, des lois, des options, et d'organiser des votes d'assemblée, des votes de concours, des référendum, etc.
- Il est possible d'envoyer des informations concernant le votant en faisant une classe fille de la classe suffrage.
- Nous pouvons implémenter également des votes au suffrage indirect, grâce aux Districts qui peuvent représenter des groupes de votants.