Rapport du Projet de Génie logiciel

BELASSAL Salma
BERTRAND Benjamin
ENNAJI Samia
ESPASA Kévin

December 2016

1 Architecture

1.1 Introduction

Pour l'implémentation de notre système, nous avons choisi de travailler avec Spark au lieu de faire une connection simple avec une base de données.
Spark est un framework open source de calcul distribué.
Il réalise une lecture des données au niveau du Cluster (grappe de serveurs sur un réseau), effectue toutes les opérations d'analyse nécessaires, puis renvoie les résultats au client lors de collect() ou collectToList(). Les données sont donc en mémoire sur une architecture distribue côté server. Ce système rend la première requête de chaque type relativement lente mais met en cache ces RDDs pour les requêtes futures En utilisant des Dataset, on peut conserver la structure de nos objets mais malheuresement, pour faire du CRUD [1] il aurait été préférable d'utiliser les tables de Spark SQL ou la persistence de Spring. Cependant comme nous pouvons nous attendre à des visites par millions et un personnel enseignant attentif, un système massivement en lecture fera l'affaire.
Pour mettre en œuvre notre application , nous avons utilisé quelques patrons architecturaux.

1.2 Vue physique

Parmi les intérêts de la vue physique :
— Analyse de performance (co-localisation des composants ayant de fortes communications...)
— Analyse de la fiabilité
— Sécurité ...

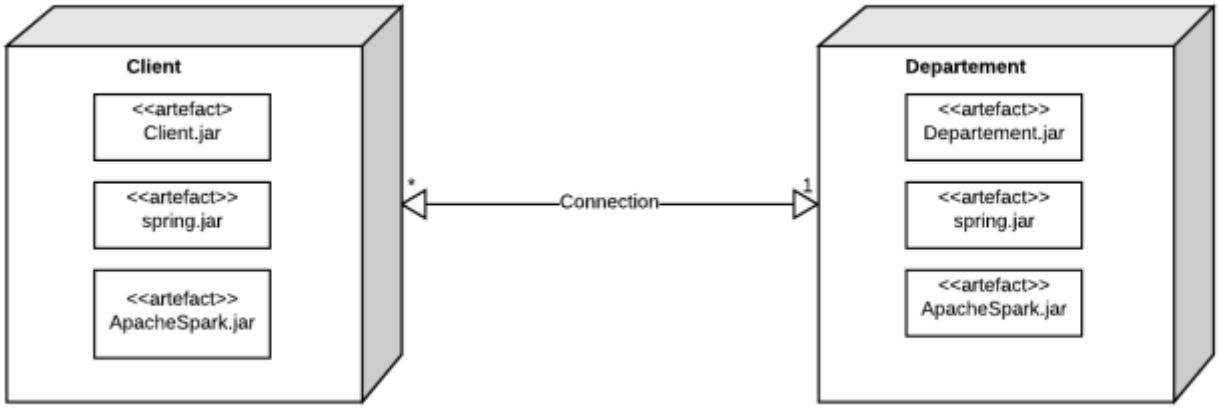

FIGURE 1 – Diagramme de déploiement du système "Gestion des services"

1. https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

1.3 Vue logique

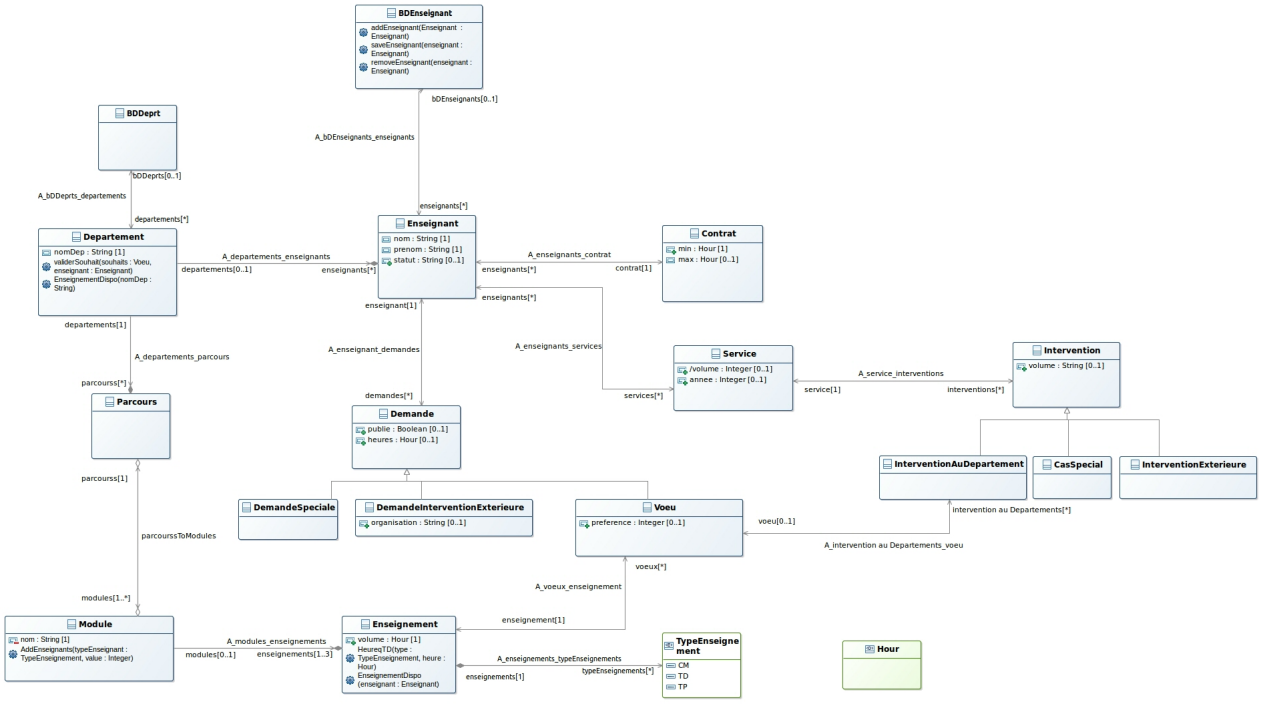

FIGURE 2 – Diagramme UML du logiciel

Le diagramme reprend celui donnée dans le sujet, en y ajoutant une classe bdd pour le Departement et une BDEnseignant.

1.4 Vue des processus

La vue des processus démontre :
— la décomposition du système en processus et actions
— les interactions entre les processus
— la synchronisation et la communication des activités parallèles ...
L'enseignant émet un souhait. Le souhait correspond à vœu sur un ensemble d'enseignements du département. Un enseignant est capable de visualiser les enseignements afin de consulter ou modifier ses souhaits.

1.5 Vue de la fiabilité

La base étant présente sur plusieurs machines en même temps, il y a peu de chance que l'ensemble du parc soit hors service et donc que l'on perde la base. Ce qui implique que l'on est pas obligé de persister sur le disque dur.

1.6 Réponses aux exigences non-fonctionnelles

1.6.1 Gestion de la concurrence

La concurrence peut être généré par des verrous, Spark n'est pas optimal pour celà, il faut donc le coupler avec le framework Hadoop.

1.6.2 Gestion la persistance

Une des meilleure capabilité de SPARK est la persistence (dataset in memory across operations), qui fonctionne de la manière suivante : chaque noeud stock ses partitions et les calculent en mémoire, puis les réutilise dans d'autres actions de la base de données, ce qui rend les actions à venir plus rapides. La gestion de la persistence est également présente par l'utilisation du Framework Spring qui facilite la gestion des exceptions, déclaration d'une source de données, assurant également une bonne gestion des transations, Classes de support pour JDBC

1.6.3 Gestion de la sécurité

Comme pour la concurrence, la sécurité avec Spark Apache est limité, il faudra donc ajouter une surcouche pour la gestion des utilisateur. Le framework Hadoop lui peut gérer la sécurité grâce à Kerberos. Le chien a trois tête garde sa grappe serveurs et garantis que les RDD stockées sur Spark ne soient pas corrompues

1.7 Architecture technique : traduction de UML en code source

1.7.1 Règles de traduction des types de base

Hormis l'utilisation de types prédéfinis en java et UML (String, Integer...), nous ne créons pas de nouveaux types, et cela car les Encoder ne supportent malheuresement pas encore les énumerations. [2] de même les Uuid seront représentés par des String

1.7.2 Règles de traduction des classes

La relation entre la classe enseignant et contrat est une relation bidirectionnelle , une relation qui peut assurer dans les deux sens la liaison entre les deux classes . La classe enseignant sur notre code prend un attribut contrat de type Contrat (pour chaque enseignant on affecte un contrat). Cependant,pour la partie gestion des interventions , on a 3 classes filles (InterventionAuDepartement , CasSpecial , Intervention Exterieure) qui héritent d'une classe mère Intervention définie comme une classe abstraite. Pour la gestion des demandes effectuées par l'enseignant,on a déclaré une classe demande avec 3 classes qui en héritent : DemandeSpeciale, DemandeInterventionExterieure, Voeu.

1.7.3 Règles de traduction des associations, agrégations composites et agrégations partagées

La relation entre la classe Enseignant et Souhait est une association bidirectionnelle. Dans la classe Enseignant , on a déclaré une liste de demandes de type Souhait , et un attribut transient Enseignant dans la classe Souhait. La relation entre la classe Departement et Enseignant est une association bidirectionnelle aussi. On déclare une liste d'enseignants de type Enseignant dans la classe Departement (un département contient plusieurs enseignants), et un attribut de type département

2. `http://dev.bizo.com/2014/02/beware-enums-in-spark.html`

dans la classe enseignant (un enseignant peut faire partie de 0 ou 1 département). La relation entre la classe Departement et Parcours est une relation de composition . En utilisant cette relation, la durée de vie du composant dépend du composite, c'est-à-dire la destruction du composite implique la destruction de tous ses composants. On a déclaré alors un attribut transient Departement dans la classe Parcours, et une liste de Parcours dans la classe Departement. La relation entre la classe parcours et module est une relation d'aggrégation de deux sens (chaque sens veut dire qu'il y a un critère d'appartenance d'une classe dans l'autre). On a déclaré un attribut de type parcours dans la classe module , et une liste de modules dans la classe parcours . La relation entre la classe service et intervention est une relation bidirectionnelle.Dans la classe intervetion on a déclaré un attribut de type service et une liste de services dans la classes interventions. Entre la classe service et enseignant on trouve une relation bidirectionnelle aussi. Dans la classe service, on a déclaré une liste d'enseignant , on a déclaré aussi une liste de service dans la classe enseignant. On trouve une relation bidirectionnelle entre la classe voeu et enseignement , on a donc déclaré une liste de voeu dans la classe enseignement , et un attribut de type enseignement dans la classe voeu(chaque voeu est affecté à un seul enseignant)

1.8 Patrons architecturaux utilisés

Dans cette partie , on va décrire les patrons architecturaux utilisés

1.8.1 Patron State

Nous avons choisi d'utiliser un pattern State pour modéliser l'état d'un voeu. Ce dernier peut être validé localement, c'est à dire que l'enseignant à choisi de le mettre sur sa base personnel, il peut être publié (rendu public aux autres utilisateurs), refusé ou affecté par le département.

1.8.2 Patron Decorateur

Afin d'étendre les différentes implementations de DAO simplement, nous créons un Decorateur MergingDAO, qui décore, non pas un mais plusieurs DAO. comme nous travaillons dans un environement massivement ditribué, il est donc possible que chaques Département posséde sa base et que l'utilisateur se connecte à plusieurs bases à la fois.

1.9 Choix techniques - Distribution des processus

Comme langage de programmation, nous avons choisi JAVA version 1.8, et l'environnement de développement choisi est l'IDE Eclipse de développement JAVA avec l'outil de construction automatique *Maven*.

La conception en langage de modélisation UML, en utilisant 'draw.io' de google et le logiciel 'Visual Paradigm'[1] pour la réalisation des diagrammes.

Le développement du code source du logiciel était fait en utilisant le système de gestion de versions Git.

Nous avons également utilisé *JUnit* pour les tests unitaires.

La documentation du code source JAVA avec l'outil *Javadoc* en enregistrant la configuration de génération dans un script *Ant*.

1.10 Spring

Le développement des composants se fait grâce au framework Spring.

@Configuration distingue cette classe pour que Spring l'utilise. Elle est initialisé lors de l'appel de :

```
ApplicationContext springContext = new AnnotationConfigApplicationContext("genielogiciel.config");
```

Les @Value peuvent provenir de config files, de paramètre -D passé au programme, ou d'autres sources encore specifiées par Spring.

Les @Bean sont chargés et sont uniques, c'est pour cette raison que l'annotation est mise dans la classe et pas dans la classe abstraite ou dans l'interface.

```
@Configuration
public class SparkConfig extends SparkDAOImpl implements SparkConfigI {

    @Value(value = "${remote.name:Remote}")
    private String APP_NAME;

    @Value(value = "${remote.master:local[2]}")
    private String master;

@Bean
public SparkSession sparkSession() { ... }

...
}
```

Les appels serveurs peuvent être "securisé" grace a @Transactional qui peut propager la transaction malgré les sous appel de fonctions. D'autres attributs sont disponibles

```
@Transactional(propagation=Propagation.REQUIRED)
```

2 Specification des composants

2.0.1 Objectifs

L'objectif de cette section est de décrire la décomposition du logiciel en composant, pourquoi nous avons fait ces choix et en quoi ils répondent à une spécification précise.

2.0.2 Organisation du chapitre

Dans un premier temps nous allons décrire l'intéraction entre les composants et le choix de cette décomposition. Dans un deuxième temps, nous allons parler plus précisément de chaque composant.

2.1 Description des composants

Nous avons choisi de décomposer le logiciel en six composants, le premier composant est le Département, il regroupe les classes en rapport avec les matières et les modules. Deux autres composants sont en rapport avec le Departement, le composant GUIDepartement et BDDépartement. L'un gère la base de données liées au Département et l'autre son interface. Un autre composant est Enseignant,

il regroupe les classes ayant un rapport avec un enseignant : son contrat, ses souhaits, ses affectations. Deux autres composants sont rattachés par des interfaces : GUIEnseignant et BDEnseignant. Une interface entre Département et Enseignant existe.

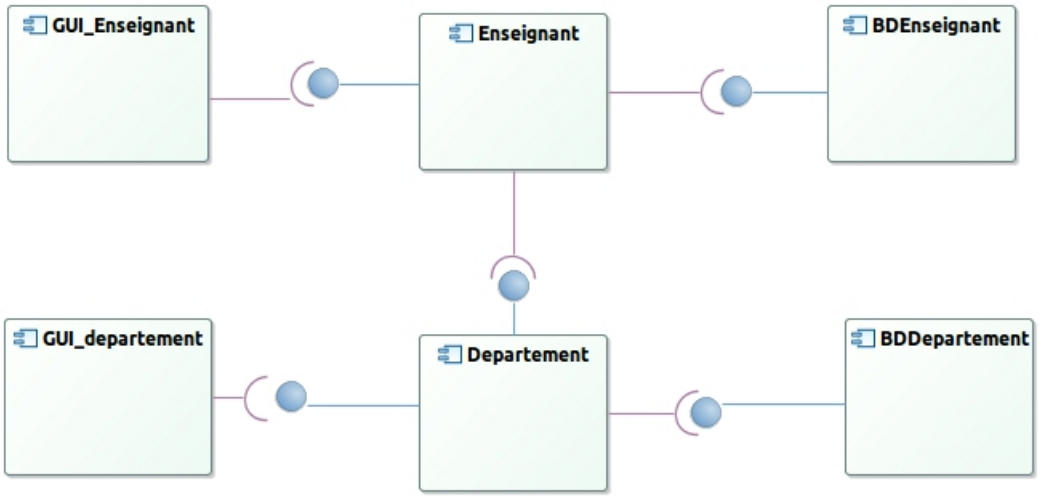

FIGURE 3 – Diagramme de composants du logiciel

2.1.1 Composants liés au Département

Nous avons fait le choix de décomposer en 3 composants la partie Département,ce choix est du à la volonté d'avoir des parties indépendantes les une des autres. On peut imaginer l'implémentation de plusieurs interfaces, ou de plusieurs types de bases de données. Ses modifications ne doivent pas avoir d'impact sur le composant Département en lui même.

FIGURE 4 – Diagramme de composants Département

2.1.2 Composants liés au Enseignant

On est dans le même cas que pour Département.

FIGURE 5 – Diagramme de composants Enseignant

2.2 Interactions

Dans cette section , on va décrire la collaboration entre les composants majeurs, pendant la réalisation des cas d'utilisation en utilisant des diagrammes de séquences et de communication.

2.2.1 Cas d'utilisations *publier un souhait*

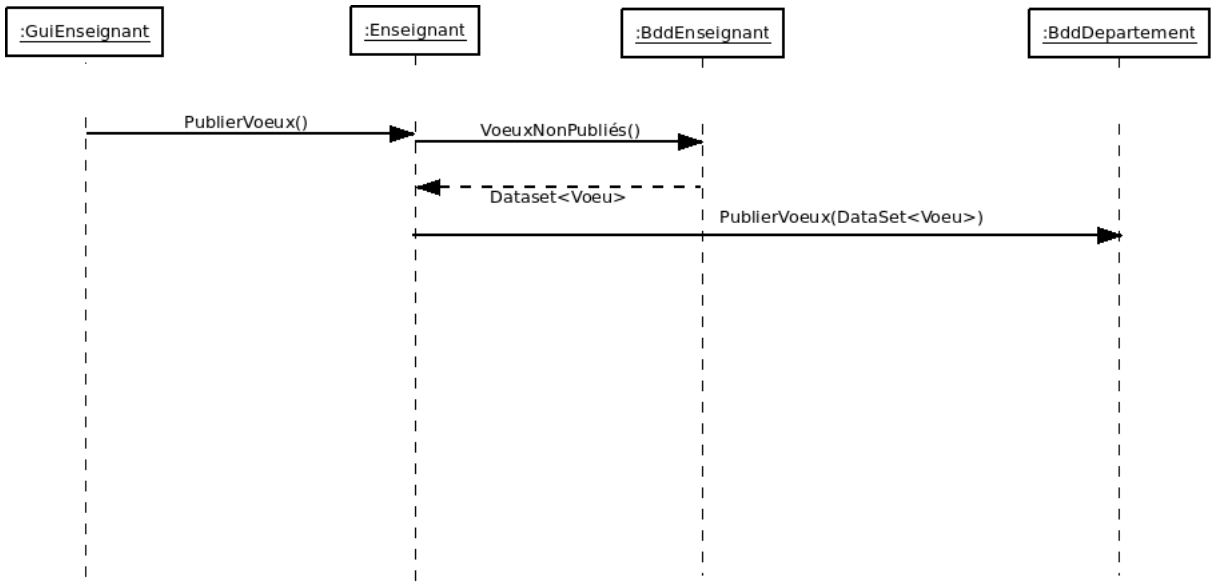

FIGURE 6 – Diagramme de séquence pour la publication d'un voeu

2.2.2 Cas d'utilisations *Consulter enseignement*

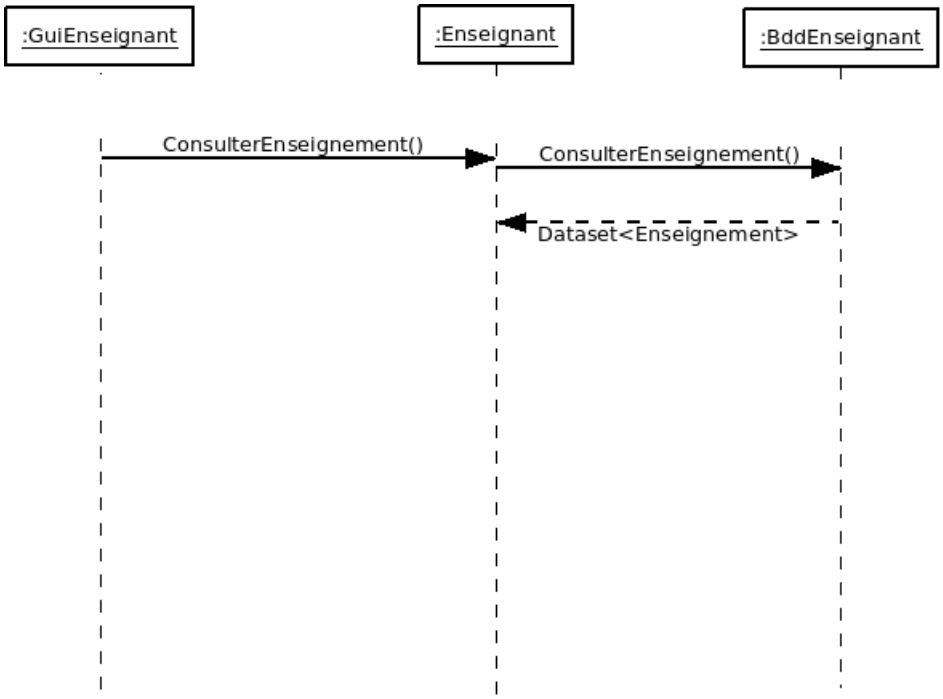

FIGURE 7 – Diagramme de séquence pour la consultation d'enseignement

2.2.3 Cas d'utilisations *Créer voeu*

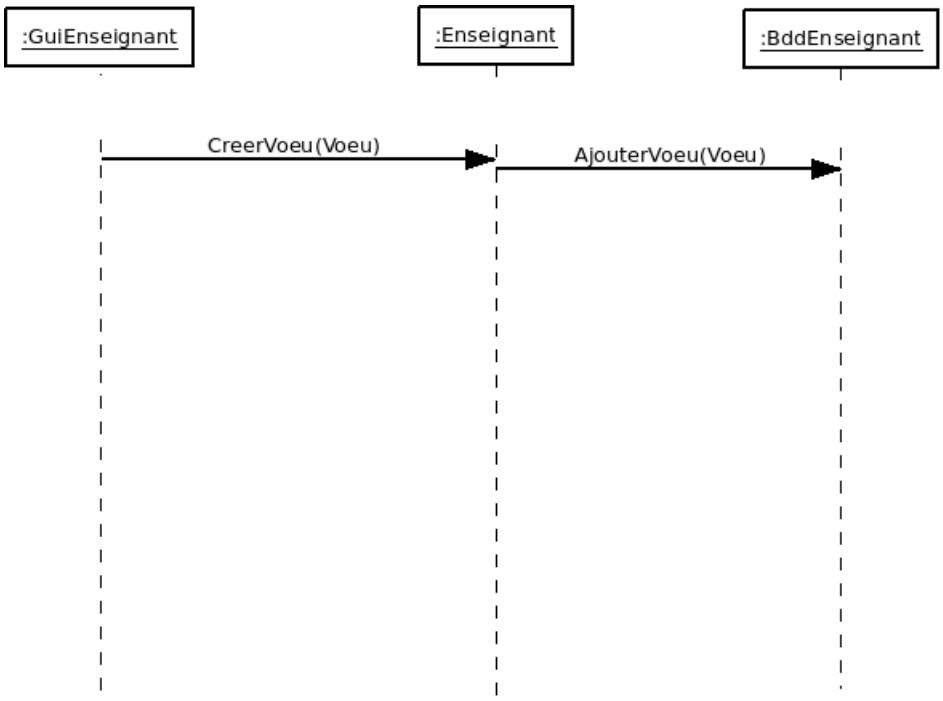

FIGURE 8 – Diagramme de séquence pour la création d'un voeu

2.2.4 Cas d'utilisations *valider voeu*

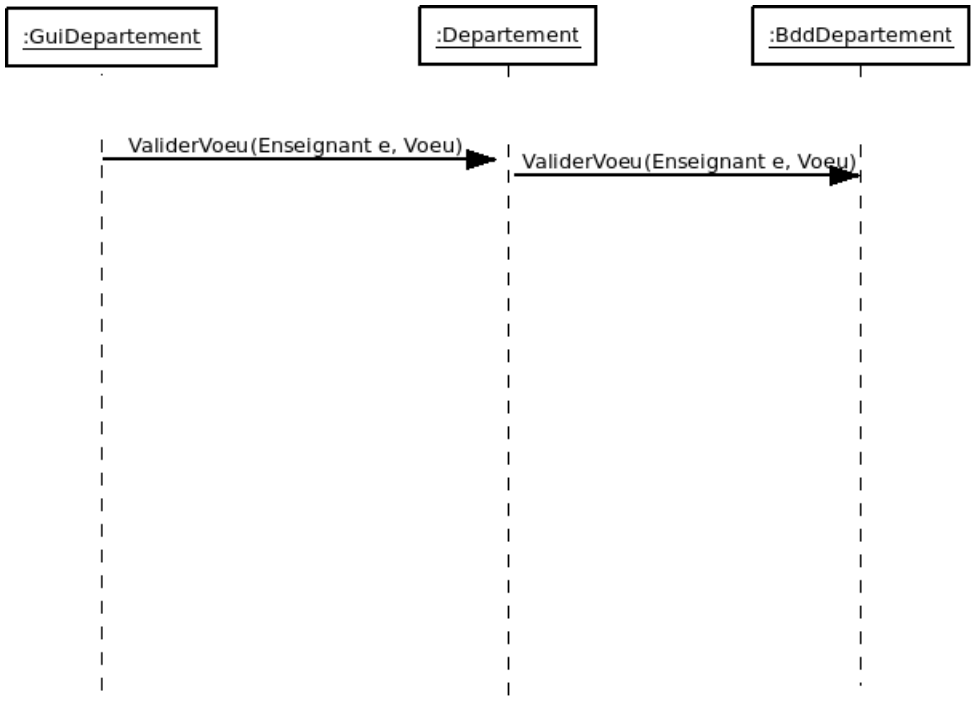

FIGURE 9 – Diagramme de séquence pour la validation d'un voeu

2.2.5 Cas d'utilisations *publier voeu département*

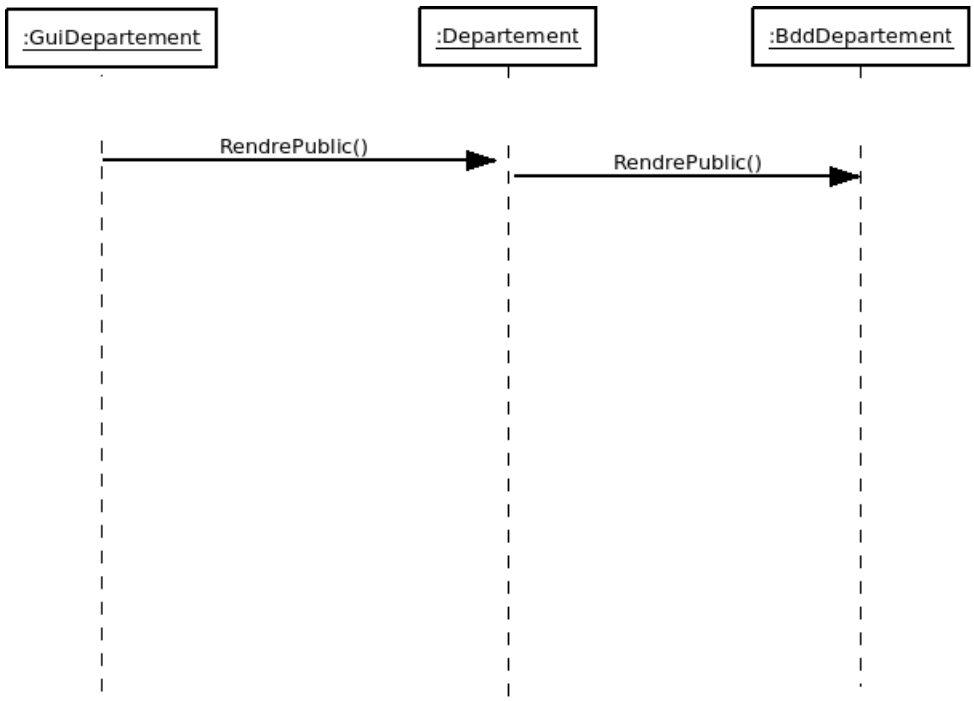

FIGURE 10 – Diagramme de séquence pour la validation d'un voeu de la part d'un département

2.2.6 Diagramme de communication

En utilisant un diagramme de communication pour illustrer certaines interactions entre les composants du système :

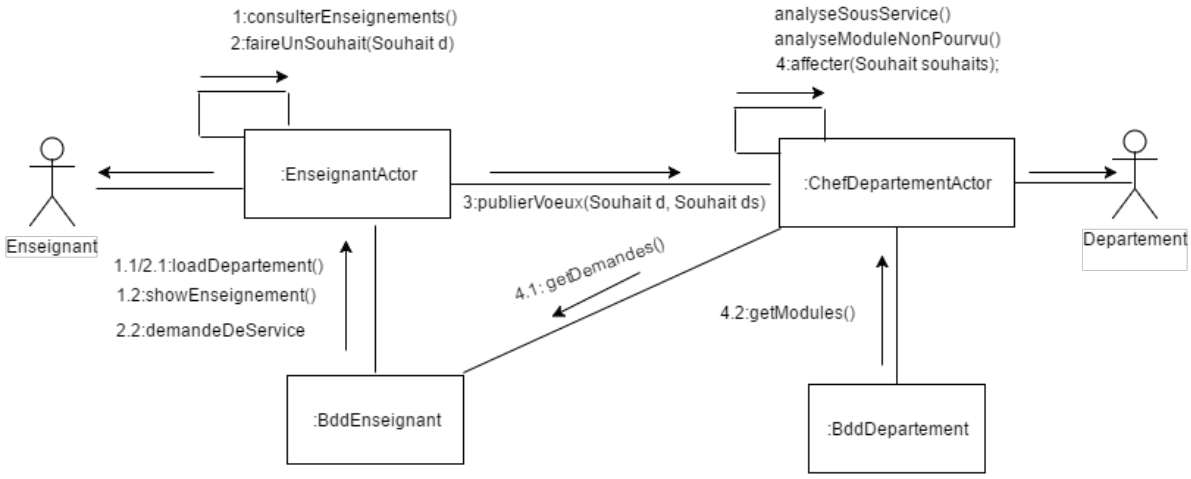

FIGURE 11 – Diagramme de communication

2.3 Spécifications des interfaces

2.3.1 Interface EnseignantActor

REQ 16 (Expression de souhaits) Le système doit permettre aux enseignants d'émettre ses souhaits d'enseignement.

```
public void faireUnSouhait(Souhait d);
```

Jouer Terme employé dans la description du domaine. Action, faite par un enseignant, de formuler un certain nombre de souhaits qu'il pourra décider, par la suite, de publier (de soumettre au chef de département et aux autres enseignants) ou pas.

```
public void jouer(Souhait d);
```

REQ 17 (Publication de souhaits) Le système doit permettre aux enseignants de publier leurs souhaits. une publication requiert une transaction avec le serveur de bdd

```
@Transactional(propagation=Propagation.REQUIRED)
void publierVoeux(Souhait d, Souhait... ds);
```

REQ 18 (Consultation des enseignements) Le système doit permettre aux enseignants de consulter les enseignements.

```
Dataset<Enseignement> consulterEnseignements();
```

2.3.2 Interface ChefDepartementActor

REQ 13 (Affectation) Confirmation, par le chef du département, d'un souhait d'un enseignant.

```
List<Intervention> affecter(Souhait... souhaits);
```

REQ 15 (Analyse) Le système doit aider le chef du département à analyser les souhaits pour détecter des enseignants en sous-services et des modules non pourvus.

```
    public Stream<Enseignant> analyseSousService();
    public Stream<Module> analyseModuleNonPourvu();
```

2.3.3 L'interface SparqConfigI

Interface pour la configuration de Spark contenant deux méthodes qui retournent la session de Spark et le SQL Context

```
     sparkSession : retourne SparkSession

     sqlContext : retourne le SQLContext
}
```

2.4 L'interface DAO

L'interface data access objet pour acceder aux données qui gere l'affectation des souhaits, les parcours, les demandes de services, enseignements, modules et l'affichage des enseignements..

```
les méthodes:
    parcours : retourne une liste des parcours

    modules: retourne une liste des modules

    enseignants : retourne une liste des enseigants

    souhaits: retourne une liste des souhaits

    intervention: retourne une liste des interventions

    enseignements: retourne une liste des enseignements

    showEnseignement: affiche tous les enseignements

    demandeDeService: affiche les demandes de service
```

2.5 L'interface SparkDAO

l'interface spécifique à Spark, pour la gestion des datasets des enseignants, departements, parcours et modules.

2.6 Conclusion

La spécification en composants nous a permis de mieux fragmenter l'application en morceau indépedants les uns et des autres tout en donnant un rôle, une spécification bien défini à chacun.

3 Conception detaillée

3.1 Introduction

Dans cette partie, nous allons développer la conception détaillée du logiciel.

3.2 Répertoire des décisions de conception

3.2.1 Patron state

Pour la gestion des voeux des enseignants , on peut remarquer qu'il y a un changement d'états d'un voeu et des transitions entre ces états. Alors on a choisi d'utiliser le pattern state. Le pattern state permet à un objet de modifier son comportement quand son état interne change .Tout se passera comme si l'objet changeait de classe .

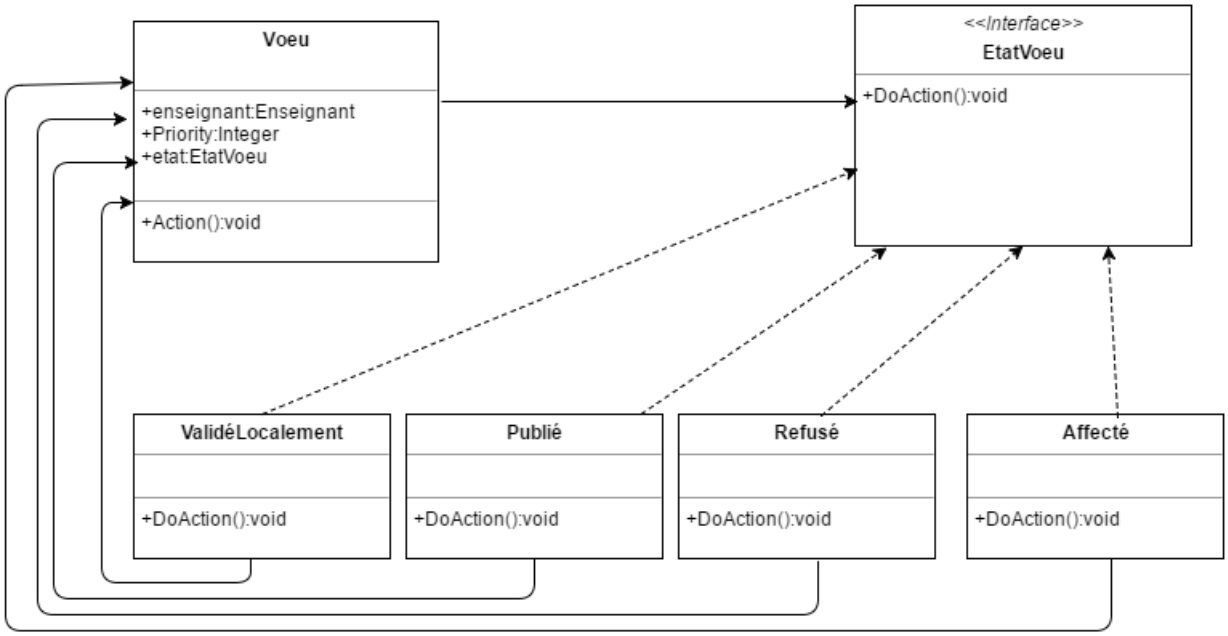

FIGURE 12 – Diagramme UML du pattern State

3.2.2 Patron Decorateur

On utilise le pattern Decorator entre MergingDAO et DAO

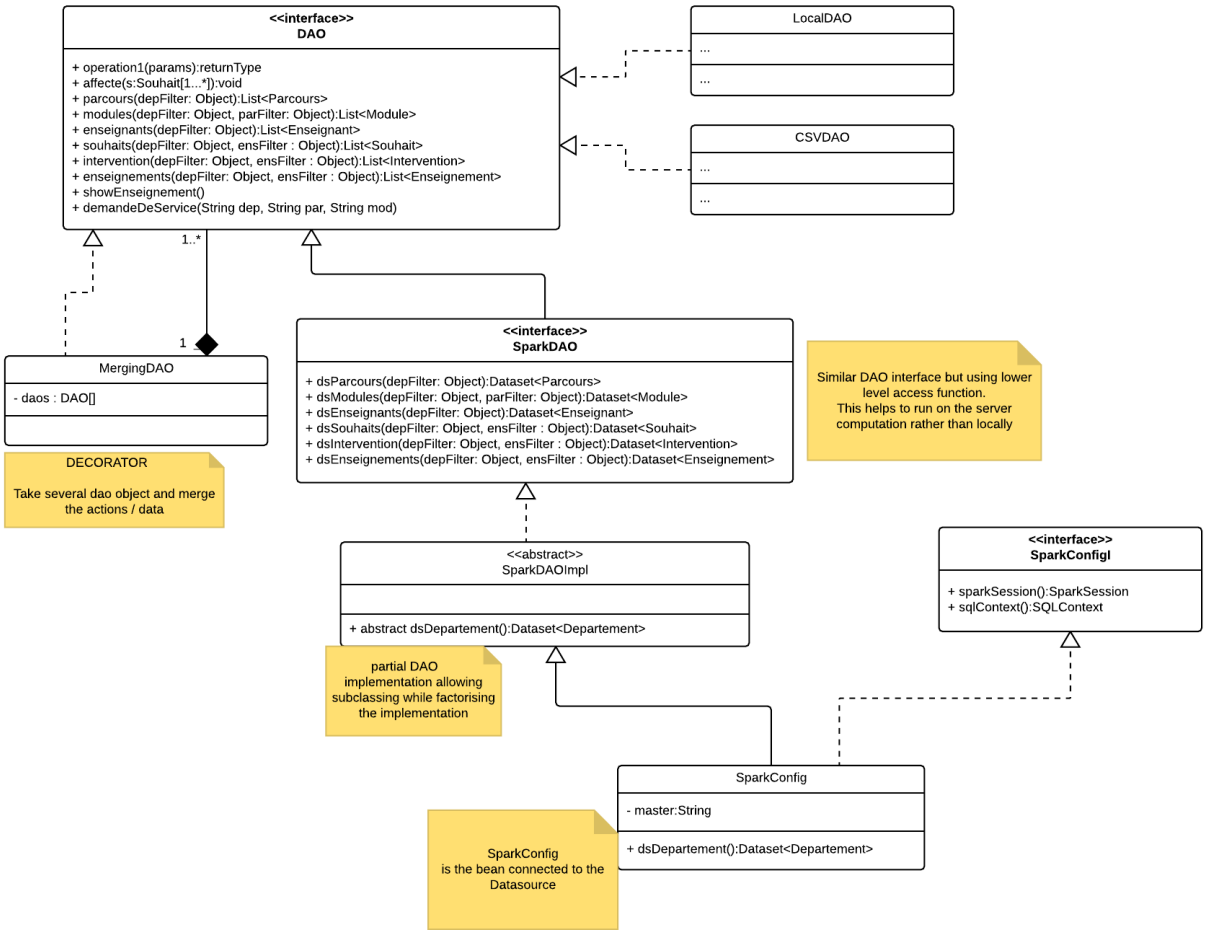

FIGURE 13 – Diagramme UML du pattern Decorator

Afin de pouvoir questionner les différentes sources de données de manière indépendante, les appels de fonctions à MergingDAO seront executés parallèlement et/ou agregés par cette interface.

Ainsi on peut voir s'exécuter demandeDeService ou showEnseignement différemment.

```
@Override
public void showEnseignement() {
    departements().stream().forEach(d -> System.out.println(d));
}

Departement: Licence Informatique
Parcours: L2 Info
    [Reseau:CM:15.0 , Reseau:TD:20.0 , Reseau:TP:15.0 , Reseau:TP:15.0]
    [Based de Donnee:CM:18.0 ....
```

```
@Override
public void showEnseignement() {
    dsDepartement()
        .selectExpr("name as D","explode(parcours) as p")
        .selectExpr("D", "p.name as P", "explode(p.modules) as m")
        .selectExpr("D", "P", "m.name as M", "explode(m.enseignements) as e")
        .selectExpr("D as Departement", "P as Parcours", "M as Module", "e.type as Type", "e.volume
        .groupBy("Departement", "Parcours", "Module")
        .pivot("Type", Arrays.asList("TD","TP","CM"))
        .sum("Volume")
        .show();
}
```

```
|        Departement|Parcours|        Module|  TD|  TP|  CM|
+-------------------+--------+--------------+----+----+----+
|Master Informatique| M1 ATAL|Genie Logiciel|20.0|30.0|15.0|
|Master Informatique| M1 ALMA|       Anglais|null|null|80.0|
|Master Informatique| M1 ALMA|          Test|15.0|15.0|20.0|
```

3.3 Spécification détaillée des composants

3.3.1 Composant Département

Le composant regroupe 4 classes. Une classe Département, une classe Parcours, une classe module et une classe Enseignement. La classe département possède une liste de parcours ainsi qu'une liste d'enseignant, elles sont initialisées avec Spark en allant chercher dans le fichier département.json. Le parcours, lui possède une liste de module et le module possède une liste d'enseignement. Dans la classe enseignement est calculé le vrai coup horaire d'un enseignement.

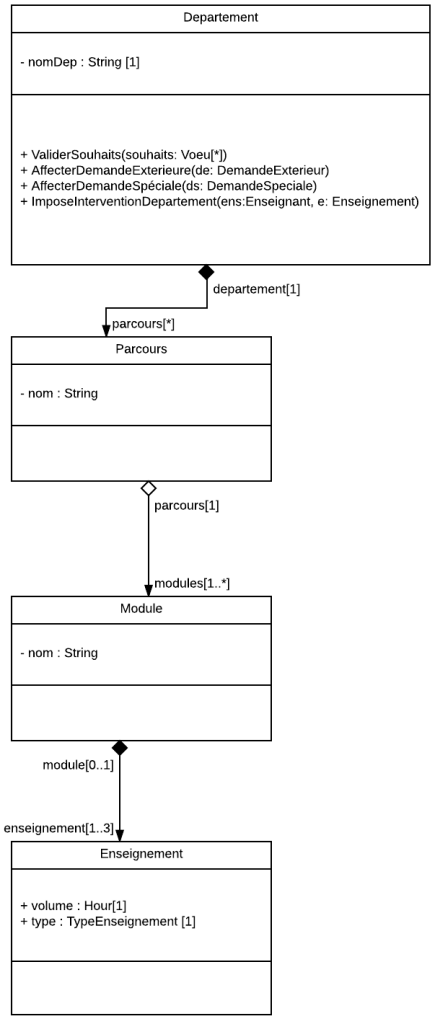

FIGURE 14 – Diagramme de composant Département

Le diagramme état-transition utilisé pour décrire le comportement qui dépendent de l'état interne de ce composant :

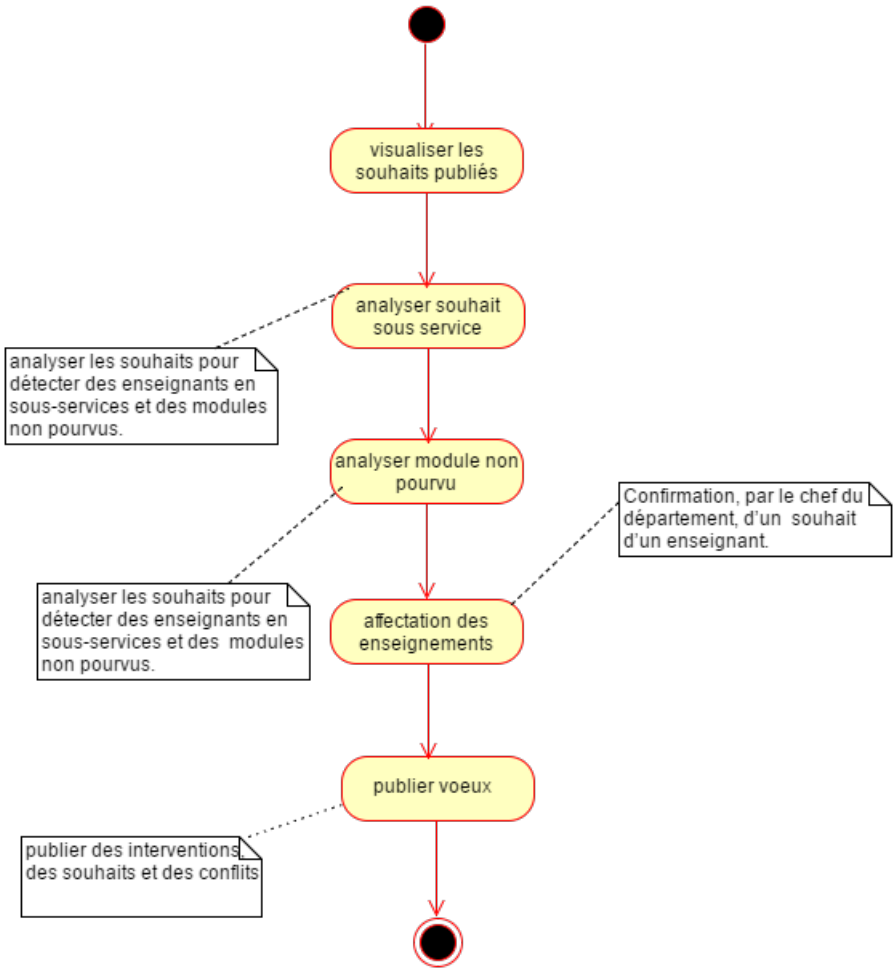

FIGURE 15 – Diagramme états-transitions Département

3.3.2 Composant Enseignant

Le composant Enseignant contient 12 classes. La classe "principale" est Enseignant, elle regroupe le nom de l'enseignant, son contrat, une liste de ses interventions et une liste de ses demandes. Demande (renommée Souhait) est une classe abstraite ayant 3 classe fille, Voeu qui représente une demande d'affectation au sein de l'établissement, DemandeExterieure qui regroupe les demandes d'intervention au sein d'un autre établissement ou d'un autre endroit. Et DemandeSpeciale qui concerne les demandes de congés. Intervention, à l'image de Demande possède 3 classes filles. Intervention au département, qui regroupe les affections qu'a recu l'enseignant une année donnée, une classe Intervention Exterieure regroupant les demandes extérieurs acceptées par le département et une Cas Speciale qui concerne les congés ou absence.

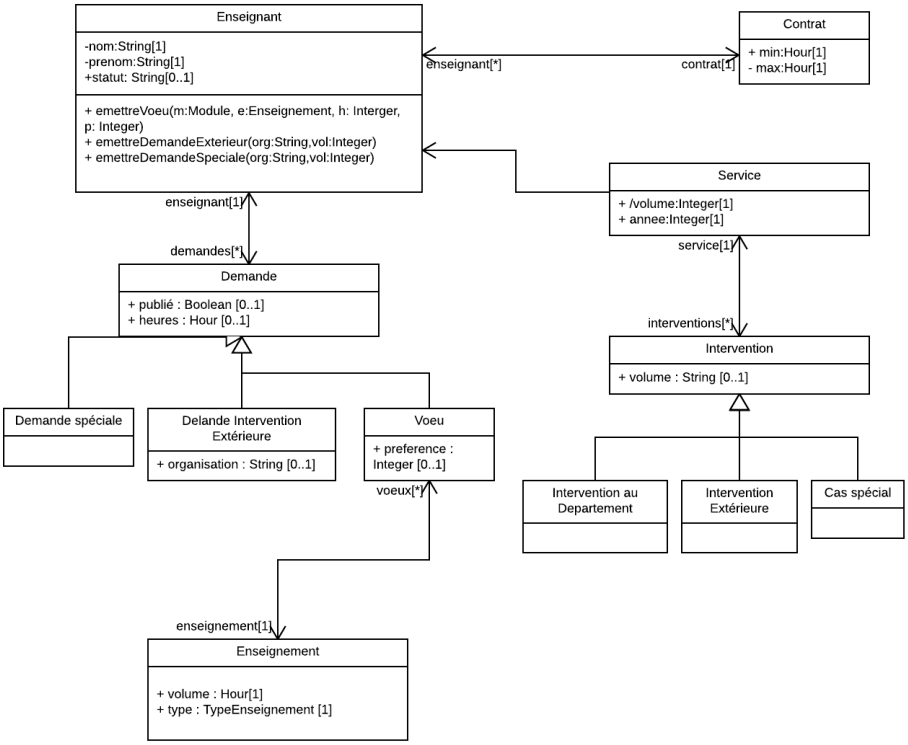

FIGURE 16 – Diagramme de composant Enseignant

Le diagramme état-transition utilisé pour décrire le comportement qui dépendent de l'état interne de ce composant :

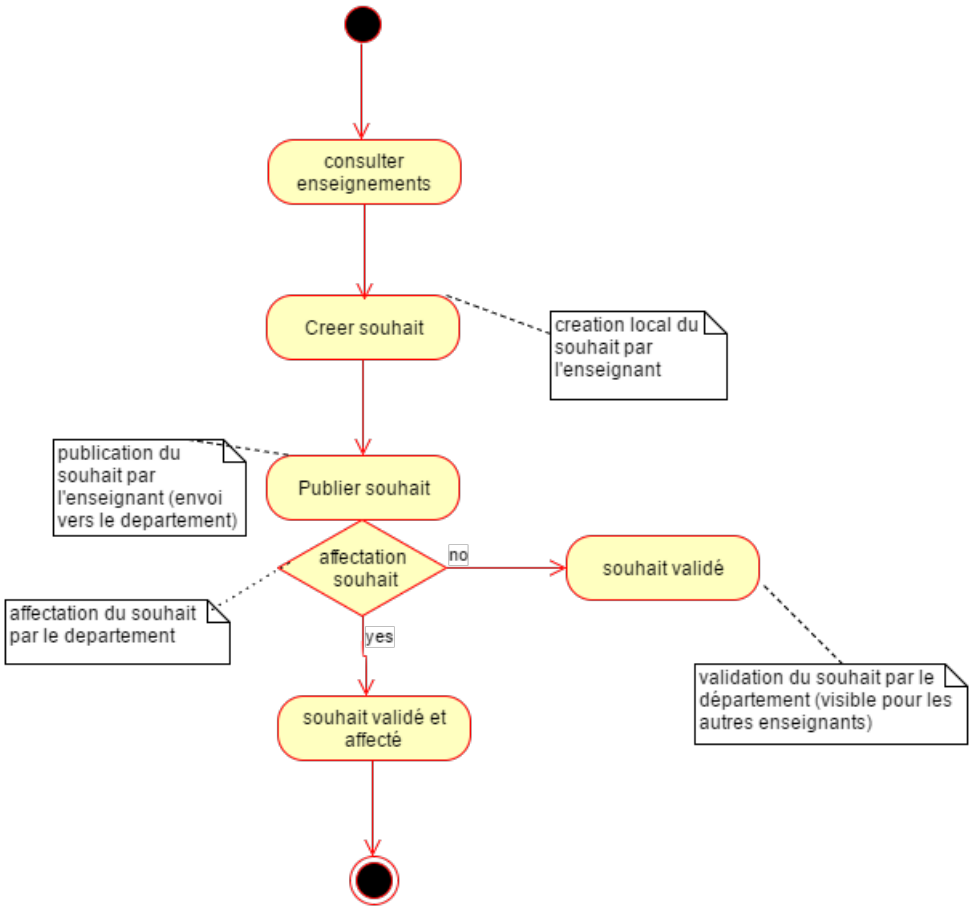

FIGURE 17 – Diagramme etats-transitions Enseignant

3.3.3 diagramme de classe du projet

le diagramme de classe de notre package *genielogiciel.model*

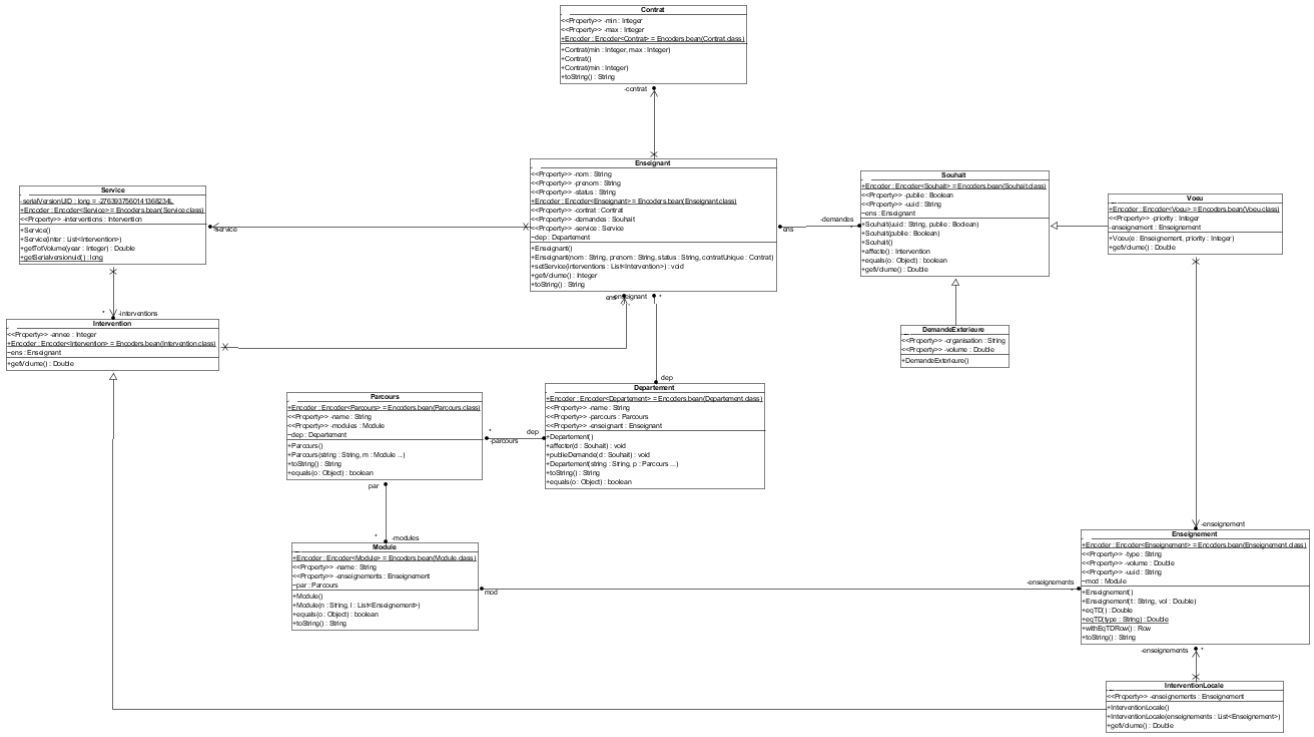

FIGURE 18 – Diagramme de classe

Pour le diagramme de classe du package de configuration de *Spark* ,le gestion du *DAO* et la gestion des interfaces, regarder la figure 13

diagramme de classe pour la classe du *ChefDepartement* et l'interface *ChefDepartementActor*

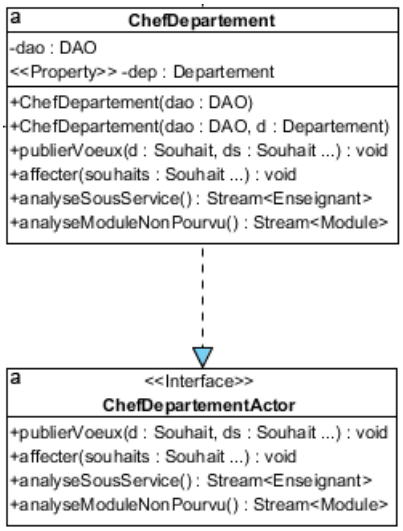

FIGURE 19 – *ChefDepartement* et *ChefDepartementActor*

3.3.4 Les classes de tests unitaires

Nous avons utilisé JUnit pour effectuer quelques tests unitaires pour tester le fonctionnement de nos classes, notamment les classes :

* `Enseignant`
* `Enseignement`
* `Voeu`
* `Intervention`
* `Departement`
* `Souhait`

3.4 Conclusion

Le projet a été trés enrichissant , il nous a aidé à mieux comprendre et approfondir nos connaissances en ce qui concerne l'utilisation des diagrammes uml , l'utilisation des designs patterns.. Ce projet nous a permis aussi de mieux comprendre la notion d'analyse de domaine vu cette année en cours de génie logiciel

Table des matières

Table des figures

Références

[1] Visual Paradigm. Visual paradigm for uml modeling. lien vers le siteweb du logiciel.