Projet Génie Logiciel :
Gestion des services

Sarra BOUTAHLIL
Alexis DONNART

Décembre 2016

Résumé

Dans le cadre du cours de génie logiciel de Master 1 ALMA 2016-2017 dispensé à l'Université de Nantes par S. Gerson, il nous à été demandé de concevoir et de mettre en œuvre un logiciel réparti permettant la gestion des services de différents enseignants.
L'objectif était d'effectuer l'analyse, la conception préliminaire ainsi que la conception détaillé du système afin de pouvoir le mettre en œuvre en Java.

Table des matières

1 Architecture
- 1.1 Introduction
- 1.2 Vue physique
- 1.3 Vue logique
- 1.4 Vue de processus
- 1.5 Vue de fiabilité
- 1.6 Réponses aux exigences non-fonctionnelles
  - 1.6.1 Gestion de la concurrence
  - 1.6.2 Gestion de la persistance
  - 1.6.3 Gestion de la sécurité
- 1.7 Architecture technique : traduction de UML en code source
  - 1.7.1 Règles de traduction des types de base
  - 1.7.2 Règles de traduction des classes
  - 1.7.3 Règles de traduction des associations, agrégations composites et agrégations partagées
- 1.8 Patrons architecturaux utilisés
  - 1.8.1 Patron Façade
  - 1.8.2 Patron Etat

2 Spécification des composants
- 2.1 Introduction
- 2.2 Description des composants
  - 2.2.1 Le composant Emission
  - 2.2.2 Le composant Publication
  - 2.2.3 Le composant Affectation
- 2.3 Interactions
  - 2.3.1 Cas d'utilisation UC1
  - 2.3.2 Cas d'utilisation UC2
  - 2.3.3 Cas d'utilisation UC3
  - 2.3.4 Cas d'utilisation UC4
  - 2.3.5 Cas d'utilisation UC5
  - 2.3.6 Cas d'utilisation UC6
- 2.4 Spécification des interfaces
  - 2.4.1 Interface IEmissionSouhait
  - 2.4.2 Interface IPublicationSouhait
  - 2.4.3 Interface IAffectationSouhait
- 2.5 Spécification des types utilisés
  - 2.5.1 Données
  - 2.5.2 Service
  - 2.5.3 Département

3 Conception détaillée

3.1 Introduction . . . . . .
3.2 Répertoire des décisions de conception . . . . . .
3.2.1 Patron Façade . . . . . .
3.3 Spécification détaillée des composants . . . . . .
3.3.1 Composant Emission . . . . . .
3.3.2 Composant Publication . . . . . .
3.3.3 Composant Affectation . . . . . .

1 Architecture

1.1 Introduction

Dans un premier temps, nous nous attacherons à la description générale de l'architecture du système. Nous y présenterons les différentes vues (physique, logique, processus), ainsi que les réponses aux exigences non-fonctionnels.

1.2 Vue physique

La vue physique nous montre comment se comportera physiquement le système. La figure [1] présente le diagramme de déploiement du système basé sur deux grands composants matériels : le département et le client. Le département comportera la version de l'application prévue pour le chef de département, avec la persistance de données assurée. Le client lui sera conçu pour l'enseignant qui viendra utiliser un poste du département.
La figure [2] représente un exemple de déploiement du système, composé

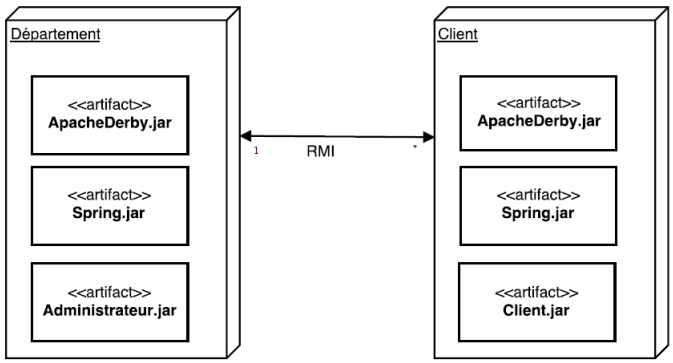

FIGURE 1 – Diagramme de déploiement

de plusieurs départements, et plusieurs clients.
Nous déploierons sur chaque département et client un ensemble d'outils pour répondre aux exigences non-fonctionnelles correspondants aux artefacts.

1.3 Vue logique

Nous nous intéressons à présent à un niveau plus détaillé : les paquetages UML. L'application sera décomposée en quatre grand paquetages [3], répondant chacun à une problématique. Le paquetage *Départements* regroupe les

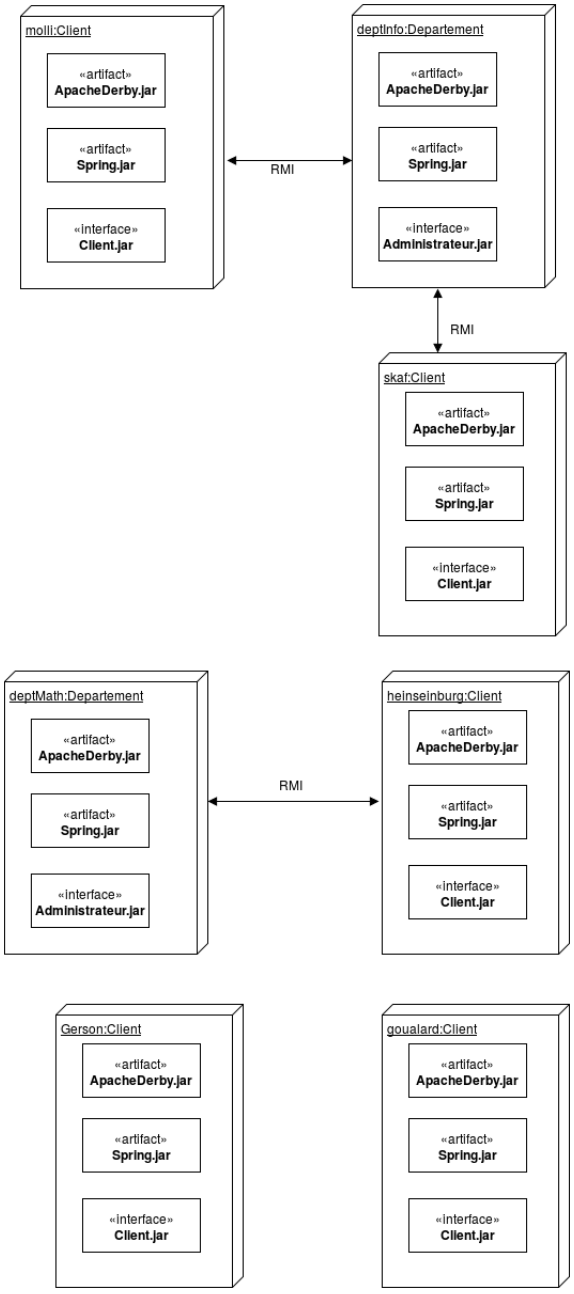

FIGURE 2 – Diagramme de déploiement d'instance

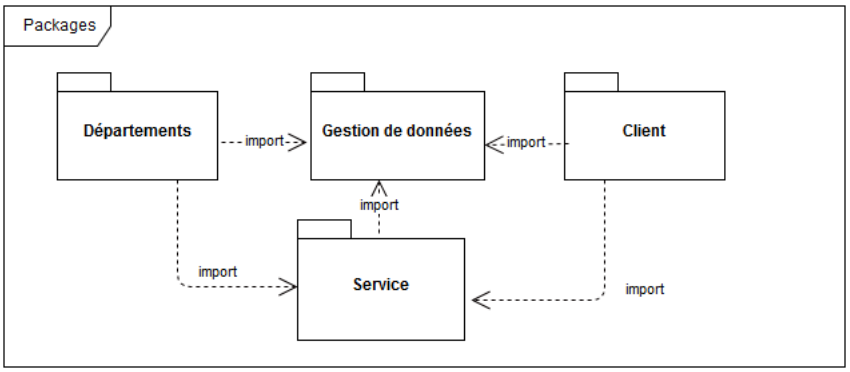

FIGURE 3 – Diagramme de paquetage

fonctionnalités propres aux département, à savoir la gestion par le chef de département, la gestion du serveur distant, et la persistance. Le *Client* de son côté gère tout les accès par le client au serveur ainsi que le stockage locale des données. Le paquetage *Service* offre les accès du client aux données et requêtes sur le département. Tous se basent sur les données du paquetage *Gestion de données*, décrivant le diagramme de classe de l'application.

1.4 Vue de processus

Le déroulement de l'application se base sur deux(ou plusieurs) processus parallèles correspondant aux départements et aux clients. Chaque processus aura un rôle précis dans le système : enseignant (emettre des voeux, etc.) ou chef de département (validation des voeux, etc.). Tout ces processus se recoupent dans un but : la gestion des services des enseignants [4].

1.5 Vue de fiabilité

Pour assurer la fiabilité du système, sa maintenabilité ainsi que son interopérabilité, nous utiliserons le framework *Spring*, permettant un développement et une maintenance facile. De plus ce framework permet la configuration simple de l'application, quelque soit la machine (utilisant la machine virtuelle java). Tout ces facteurs renforcent la fiabilité.

1.6 Réponses aux exigences non-fonctionnelles

Pour répondre aux exigences non-fonctionnelles, certains choix techniques se sont imposés.

1.6.1 Gestion de la concurrence

Pour assurer le bon fonctionnement du département et des clients en simultanés, nous avons choisi d'assurer la concurrence à l'aide d'un ser-

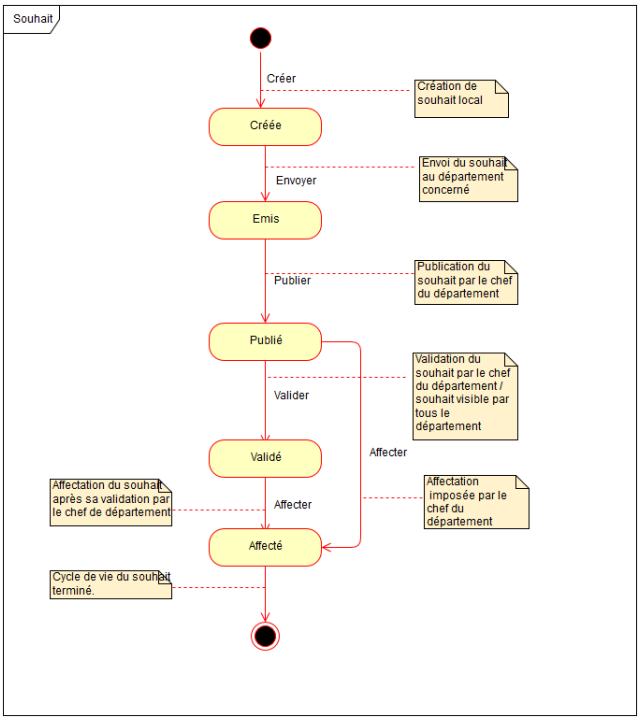

FIGURE 4 – Diagramme d'état transition global

veur *Spring RMI*. Chaque département possèdera son serveur, assurant que chaque enseignant pourra s'y connecter.
En plus de répondre à des questions de maintenabilité du code, RMI offre un développement concurrent très rapide.

1.6.2 Gestion de la persistance

En plus de pouvoir se connecter à distance sur le serveur du département, nous souhaitions pouvoir stocker nos données de façon persistante. Pour se faire, nous avons utilisé *Hibernate*, déployé sur le serveur du département. L'utilisateur client qui voudra stocker ses données devra le faire par le biais du serveur RMI à l'aide des *repositories*. De même que pour la concurrence, le choix de Hibernate s'est axé sur la simplicité d'utilisation et la facilité de maintenabilité du code.

1.6.3 Gestion de la sécurité

La sécurité est remise aux implémentations faites de Spring RMI et de Hibernate, aucune sécurité supplémentaire n'a été mise en place.

1.7 Architecture technique : traduction de UML en code source

Pour traduire la conception et l'architecture développé, nous avons établis des règles de base pour la traduction.

1.7.1 Règles de traduction des types de base

Pour traduire les types de base (Hour, TypeEnseignement), nous utiliserons des classes regroupant les informations nécessaires au type de base.

1.7.2 Règles de traduction des classes

Plusieurs règles s'appliquent pour les classes : toutes les classes UML deviennent des classes java, pas d'interface. Tous les attributs seront annotés à l'aide de hibernate pour respecter les relations. De même, si des instances sont nécessaires, ils seront injectés à l'aide de Spring.

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

Pour traduire les relations, nous avons respecter ces principes :

- 1 Une relation 1 à n : Une liste pour représenter les n éléments, un attribut pour représenter le 1.
- 2 Une relation 1 à 1 : Une référence unidirectionnelle dans les deux sens, d'un objet à un objet.
- 3 Une relation n à n : Une liste des deux côtés pour représenter le lien bidirectionnel.

Dans tous les cas, nous prêtons attention aux problèmes d'intégrité référentielles.

1.8 Patrons architecturaux utilisés

Plusieurs patrons de conceptions pouvaient faciliter la conception de l'application.

1.8.1 Patron Façade

Notre système doit offrir une interface simple et clair pour l'utilisateur. Bien que simple d'utilisation, l'ajout de la persistance et de la concurrence complexifie la lisibilité pour l'utilisateur. Pour cibler les fonctionnalités de chaque partie, nous offrirons une *Façade* pour la vue client et une pour la vue département, n'offrant que des fonctionnalités de haut niveau. C'est tout le principe de la façade : regrouper des éléments complexes pour n'en offrir qu'une version facilité.

1.8.2 Patron Etat

Notre système sera décomposé en plusieurs composants qui viendront réaliser certaines actions : émission d'une demande, publication d'une demande puis affectation d'une demande. Ce cycle reste valide pour toute la durée d'utilisation de l'application, on pourra donc représenter ce cycle par un patron état qui permettrait de passer d'un état à l'autre facilement.

2 Spécification des composants

2.1 Introduction

Nous allons décrire les composants de notre système et leur fonctionnement au sein de l'application. Nous décrirons certains cas d'utilisation de ces processus.

2.2 Description des composants

Notre système est décomposé en trois composants.

2.2.1 Le composant Emission

Le composant Emission [5] offre les fonctionnalités nécessaires à la réalisation de l'émission d'un souhait pour l'enseignant. Ce composant se traduit par une interface proposant la fonctionnalité implémenté par une façade cachant le reste du système : l'utilisateur n'a ainsi accès qu'à l'unique fonctionnalité voulu : l'émission.

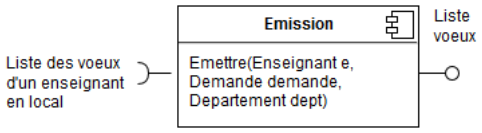

FIGURE 5 – Composant : émission

2.2.2 Le composant Publication

Sur le même principe, le composant Publication [6] offre une façade implémentant la fonction Publier, servant au chef de département à publier les voeux des enseignants.

2.2.3 Le composant Affectation

Finalement, le composant Affectation [7] offre la fonction Affecter au chef de département.

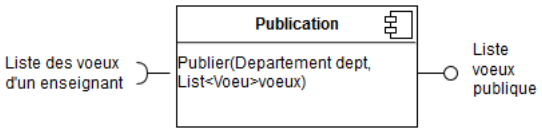

FIGURE 6 – Composant : publication

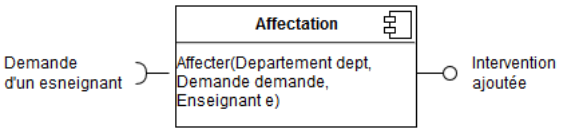

FIGURE 7 – Composant : affectation

2.3 Interactions

Avec les composants du système décrit, il est important maintenant de comprendre les interactions entre ces composants, nous allons pour cela étudier des diagrammes de séquences décrivant chaque composant et chaque cas d'utilisation.

2.3.1 Cas d'utilisation UC1

Après avoir bien étudié les scénarios du cas d'utilisation d'affectation des souhaits, nous obtenons trois cas possibles, comprenant un cas nominal et deux cas extra nominaux. **Le cas nominal** Le chef de département souhaite ici [8] récupérer la liste des voeux des enseignants pour pouvoir leur affecter des enseignements. Dans ce cas, le chef de département suit les vœux des enseignants. Une intervention au département sera créé.
Dans un premier cas extra-nominal [9], il s'agit d'affecter une demande extérieur suivant les souhaits d'un enseignant.

Dans un dernier cas [10], le chef du département, après avoir récupéré la liste des enseignants, impose une intervention à un enseignant.

2.3.2 Cas d'utilisation UC2

Dans ce deuxième cas d'utilisation [11], le chef de département visualise les demandes des enseignants. Une fois visualisés, il peut choisir d'en valider : les rendre publique aux autres enseignants.

2.3.3 Cas d'utilisation UC3

Dans le troisième cas d'utilisation [12], le chef du département visualise la liste des demandes publiés puis vient corriger les affectations des enseignants. Si un enseignant a été affecté par erreur à une intervention, le chef de département pourra le changer.

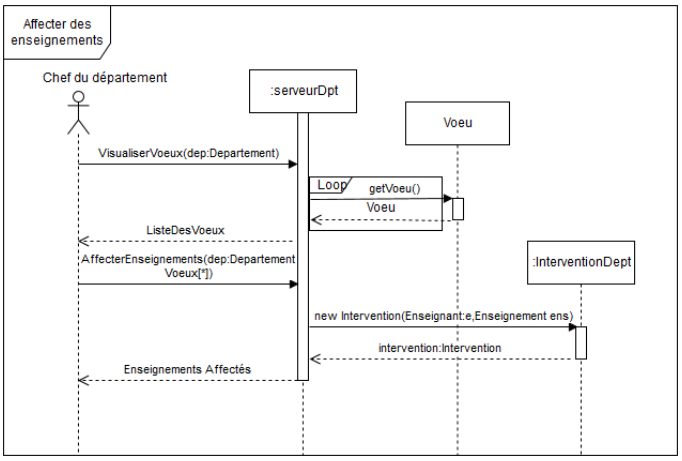

FIGURE 8 – Diagramme de Séquence UC1 : cas nominal

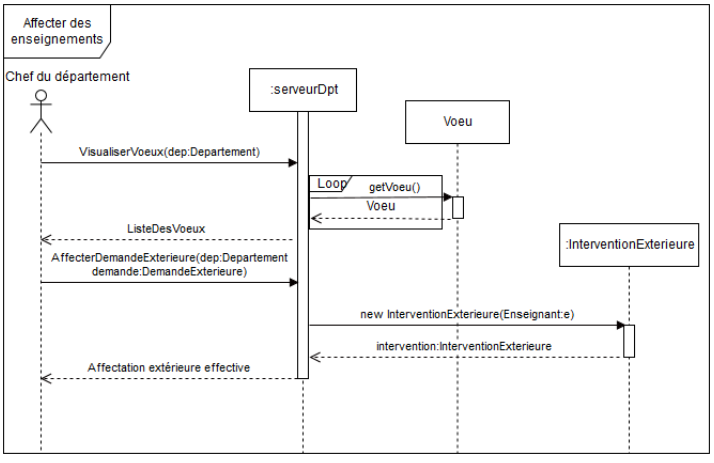

FIGURE 9 – Diagramme de séquence UC1 : cas extra nominal 1

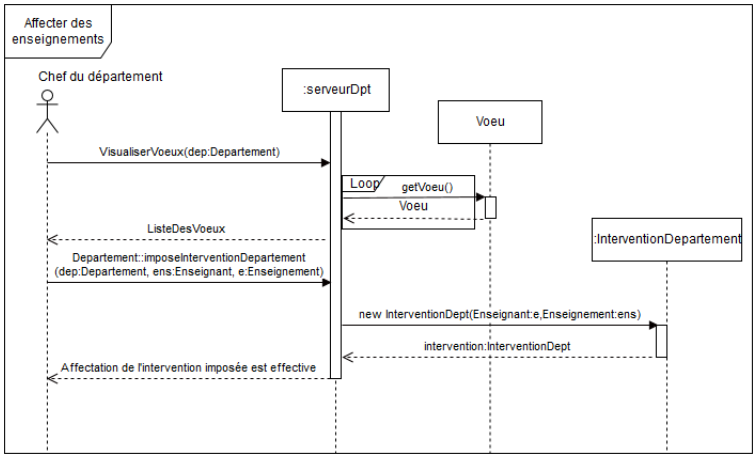

FIGURE 10 – Diagramme de séquence UC1 : cas extra nominal 2

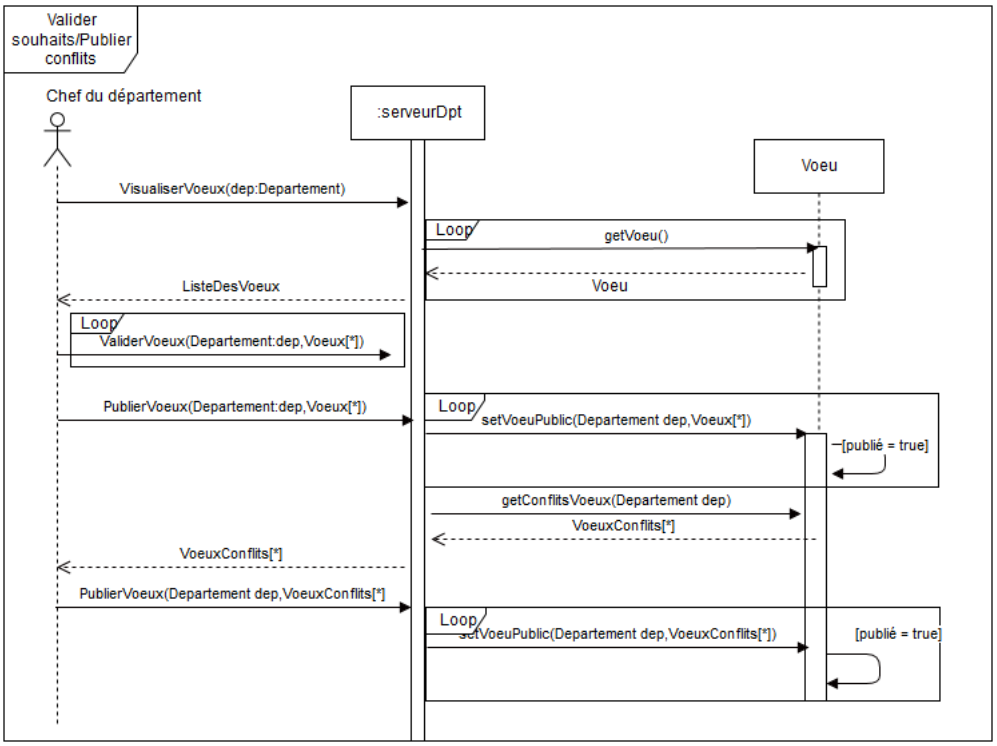

FIGURE 11 – Diagramme de séquence UC2 : cas nominal

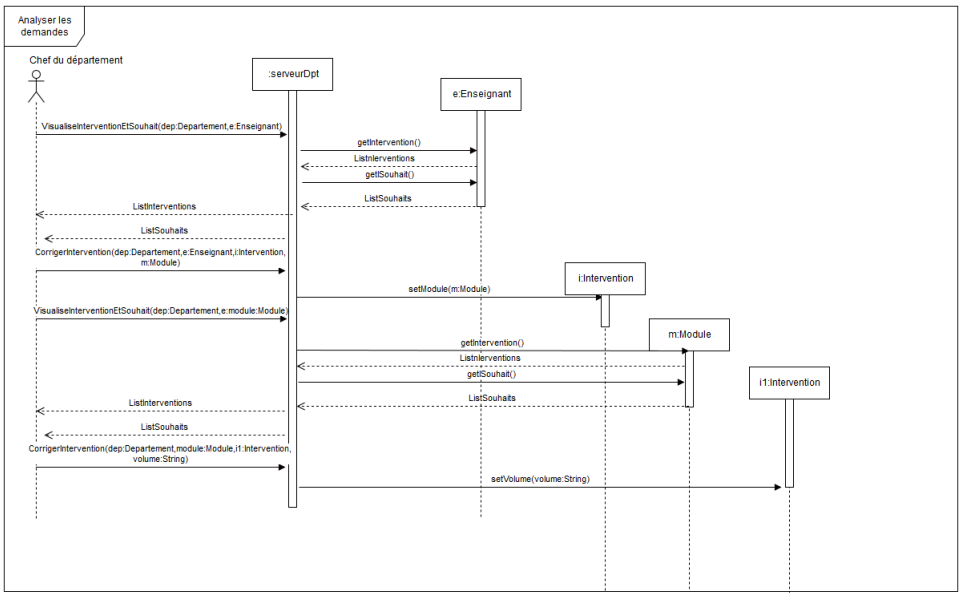

FIGURE 12 – Diagramme de séquence UC3 : cas nominal

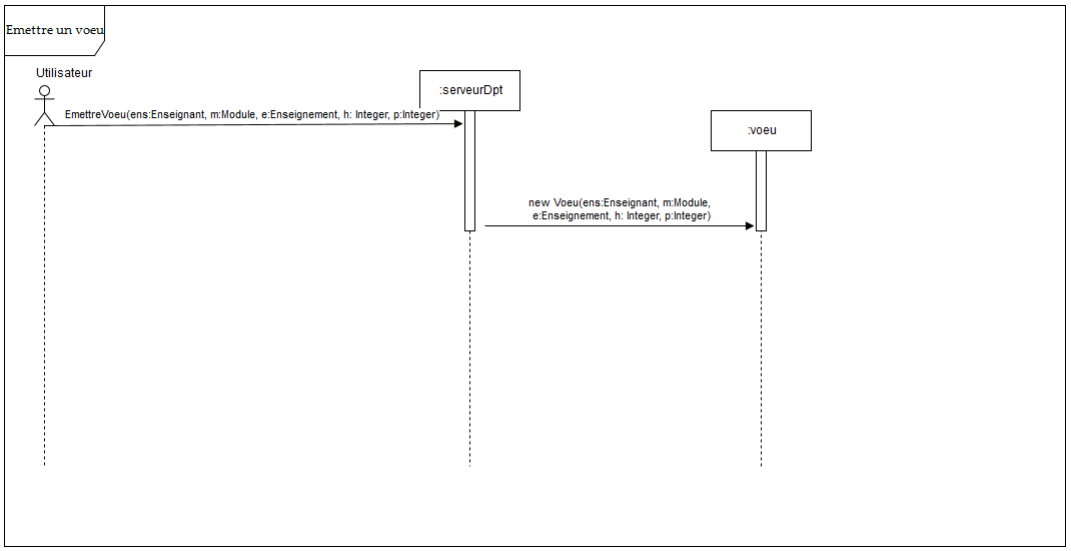

FIGURE 13 – Diagramme de séquence UC4 : cas nominal

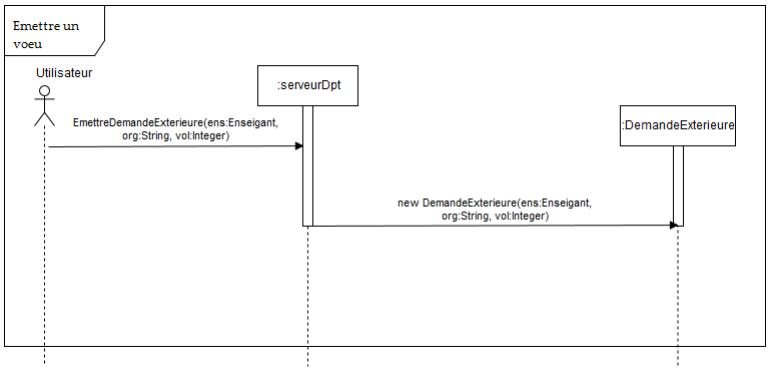

FIGURE 14 – Diagramme de séquence UC4 : cas extra nominal 1

2.3.4 Cas d'utilisation UC4

L'enseignant pourra émettre une liste de vœux [13] vers le département pour la rendre permanente. Les vœux ne seront publiques au reste des enseignants que si le chef de département les rends publiques [UC 11].

Le premier cas alternatif [14] correspond à l'émission d'une liste de demande extérieur.

Le deuxième cas alternatif [15] correspond à l'émission d'une liste de demande spéciale.

Le dernier cas alternatif [16] correspond à l'émission d'une liste de vœux avec détection de conflit. Lors de l'émission, l'enseignant sera prévenu à l'aide d'une exception du ou des vœux qui sont en conflit pour le corriger en local.

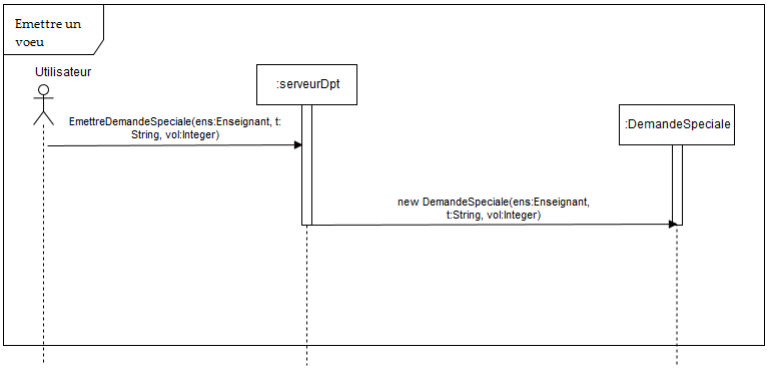

FIGURE 15 – Diagramme de séquence UC4 : cas extra nominal 2

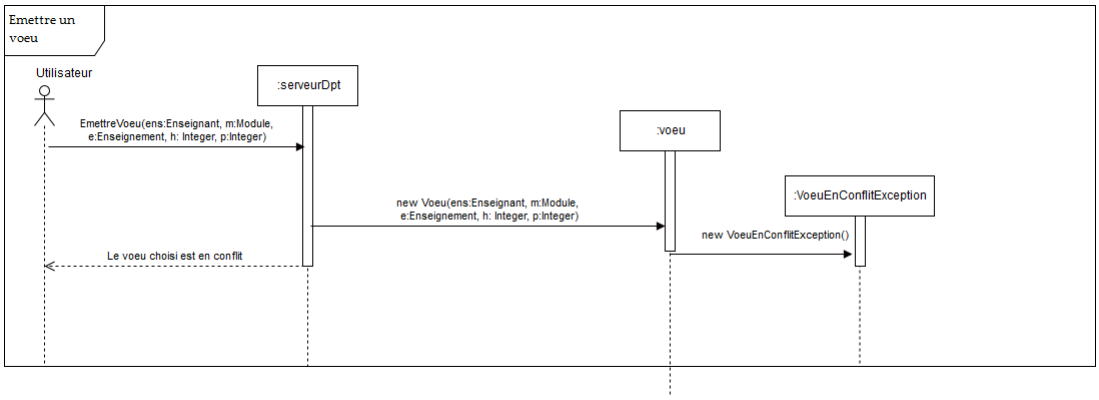

FIGURE 16 – Diagramme de séquence UC4 : cas extra nominal 3

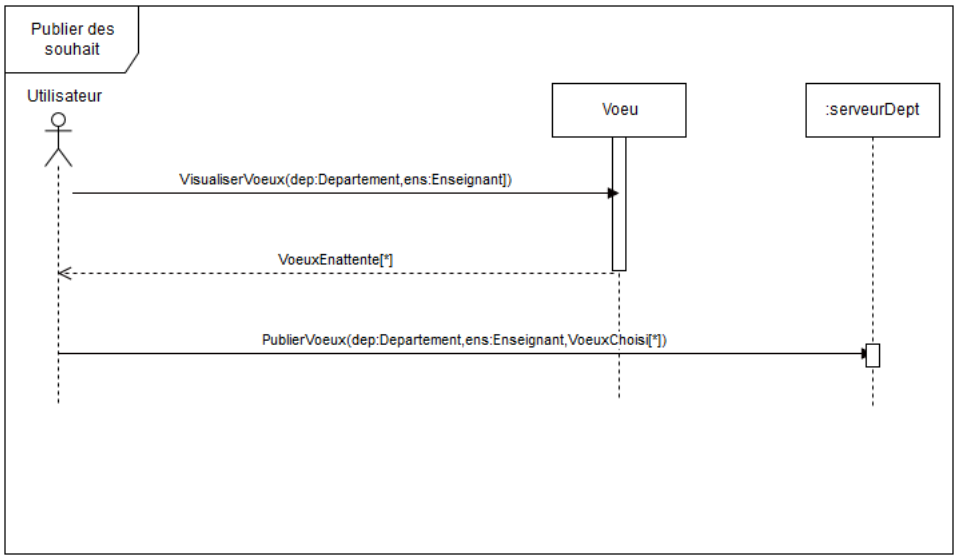

FIGURE 17 – Diagramme de séquence UC5 : cas nominal

2.3.5 Cas d'utilisation UC5

L'enseignant pourra transmettre au chef de département une liste de voeux préalablement sélectionnée [17]. Ce dernier pourra alors publier la liste.
Le premier cas alternatif [18] correspond à l'annulation d'une liste de souhait transmise : le chef de département n'en tiendra pas compte.
Le deuxième cas alternatif [19] correspond à l'émission d'une liste de voeux, mais les voeux ont déjà été affectés ou ont été supprimés.
Le dernier cas alternatif [20] correspond à l'émission d'une liste de voeux, mais la liste est vide, une exception *PasDeVoeuEnAttente* est renvoyé pour notifier l'enseignant.

2.3.6 Cas d'utilisation UC6

Le dernier cas d'utilisation [21] permet a un enseignant de consulter la liste des enseignements disponibles avant d'effectuer ses voeux.

2.4 Spécification des interfaces

Pour chaque composant, on disposera d'une interface implémentant les fonctions de base.

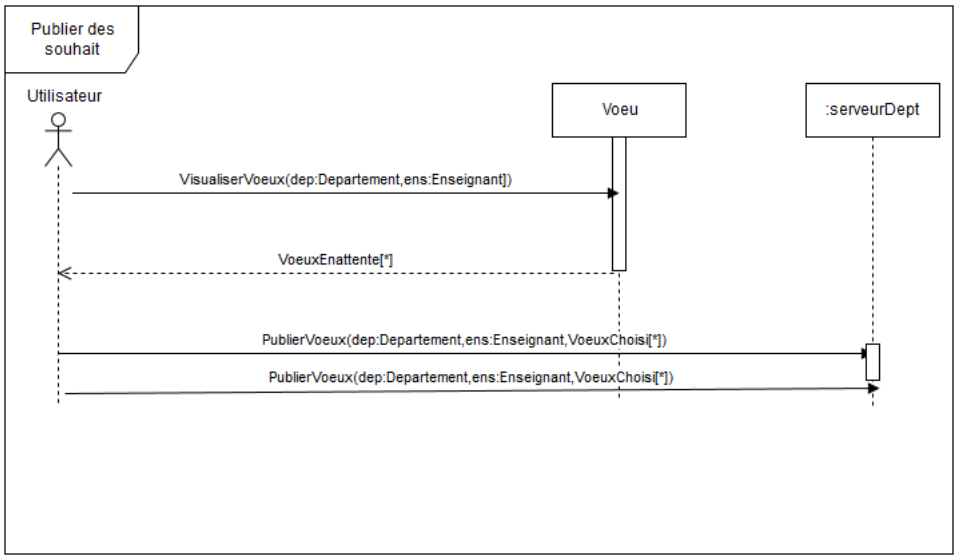

FIGURE 18 – Diagramme de séquence UC5 : cas nominal extra nominal 1

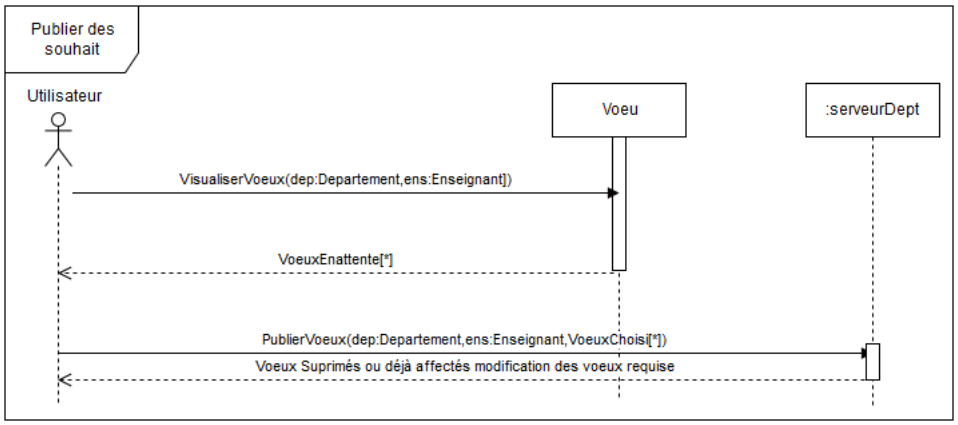

FIGURE 19 – Diagramme de séquence UC5 : cas nominal extra nominal21

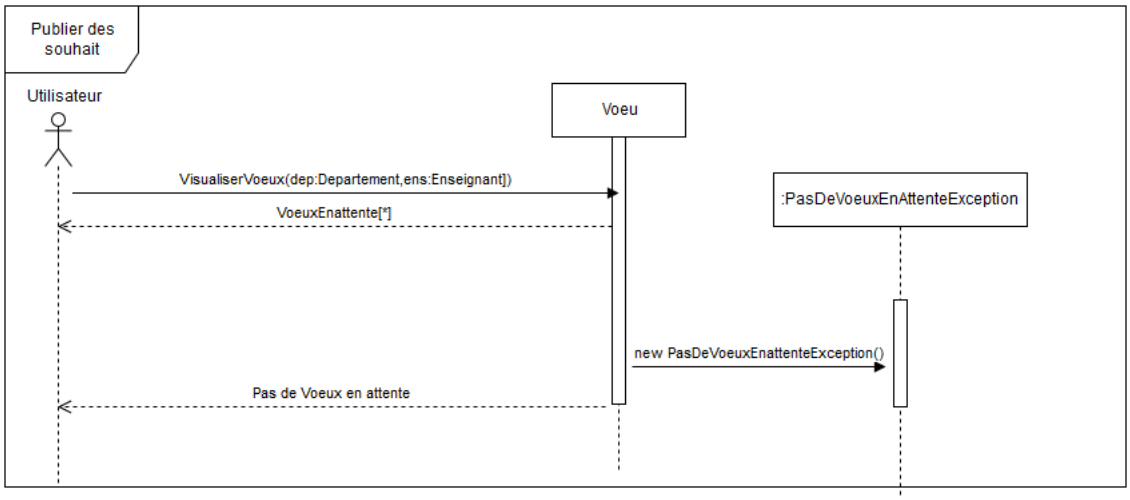

FIGURE 20 – Diagramme de séquence UC5 : cas nominal extra nominal 3

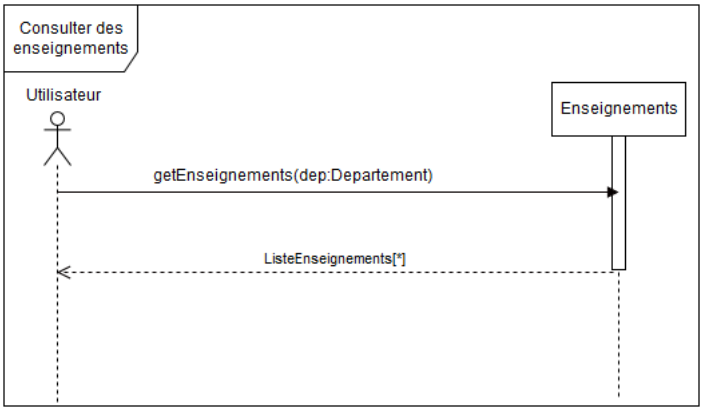

FIGURE 21 – Diagramme de séquence UC6 : cas nominal

2.4.1 Interface IEmissionSouhait

Le composant *Emission* implémente l'interface IEmissionSouhait décrivant la méthode *Emettre(Enseignant, Demande, Département)*. Cette fonction implémentera l'émission d'une demande d'un enseignant appartenant à un département vers le serveur de ce département par cet enseignant.

2.4.2 Interface IPublicationSouhait

Le composant *Publier* implémente l'interface IPublicationSouhait décrivant la méthode *Publier(Département, List<Demande>)*. Cette fonction implémentera la publication d'une liste de demandes d'un enseignant appartenant à un département sur le serveur de ce département par le chef de ce département.

2.4.3 Interface IAffectationSouhait

Le composant *Affecter* implémente l'interface IAffectationSouhait décrivant la méthode *Affecter(Département, Demande, Enseignant)*. Cette fonction implémentera l'affectation d'une demande d'un enseignant appartenant à un département sur le serveur de ce département par le chef de ce département

2.5 Spécification des types utilisés

Pour décrire l'application, les types ont été divisés en plusieurs catégories.

2.5.1 Données

Un type a été utilisé pour chaque classe de données (**Enseignant**, **Demande**, **Département**, etc). Ces classes ont pour attributs ceux spécifiés par le diagramme UML. Chaque attribut a été annoté pour l'utilisation.

2.5.2 Service

Pour l'utilisation combiné de Spring RMI et de Hibernate, nous avions besoin d'un type pour gérer la communication :

- **Service** : contient les Repositories Hibernate en tant qu'attributs, et offre les méthodes d'accès via Spring RMI pour le client.

2.5.3 Département

Le département contient les classes qui seront utilisés par le serveur du département pour gérer la persistance des données ainsi que le serveur.

– **Persistance** : toutes les classes de Repository Hibernate, contenant chaque les accès aux données persistantes.
– **Serveur** : classe de lancement du serveur du département.

3 Conception détaillée

3.1 Introduction

Après avoir effectué l'analyse puis la conception préliminaire, nous avons suffisamment d'éléments pour la conception détaillé. Nous tâcherons de décrire le comportement détaillé des composants ainsi que des choix de conception.

3.2 Répertoire des décisions de conception

Nous avons évoqué plus tôt des patrons pour concevoir notre système, revenons sur l'implémentation.

3.2.1 Patron Façade

Le patron de conception façade est utilisé pour offrir à l'utilisateur (enseignant ou chef de département) les fonctions utiles à son utilisation.
Chaque composant (Publication, Emission, Affectation) possède sa façade, offrant un ensemble de fonction utile à la réalisation de la tâche prévu. Cette façade masque l'utilisation du **Service** pour communiquer avec Spring RMI. Des fonctions d'ajout, de suppression et de mise à jour des données sont disponibles point de vue chef de département (Publication et affectation). Pour l'enseignant (Emission), seul les fonctions d'émission et d'authentification sont disponibles.

3.3 Spécification détaillée des composants

Détaillons maintenant les composants implémentés.

3.3.1 Composant Emission

Structure

Le composant Emission[22] sera présenté par la façade masquant l'utilisation des classes de données et des fonctionnalités de Spring/Hibernate. Cette façade implémente l'interface IEmissionSouhait offrant la fonction Emission à l'enseignant.

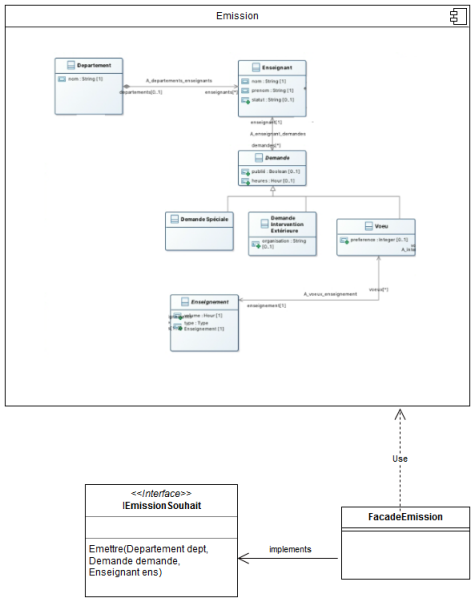

FIGURE 22 – Diagramme de classe du composant Emettre

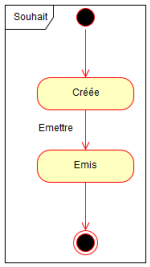

FIGURE 23 – Diagramme état transition du composant Emission

Comportement

Le comportement interne du composant Emission est décrit à l'aide des diagrammes état-transition [23] et d'activité [24].
Le comportement implémente toutes les situations décrites par ses cas d'utilisations.

3.3.2 Composant Publication

Structure

Le composant Publication[27] sera également présenté par sa façade, masquant l'utilisation des classes de données. Cette façade implémente l'interface IPublicationSouhait offrant la fonction Publier au chef du département.

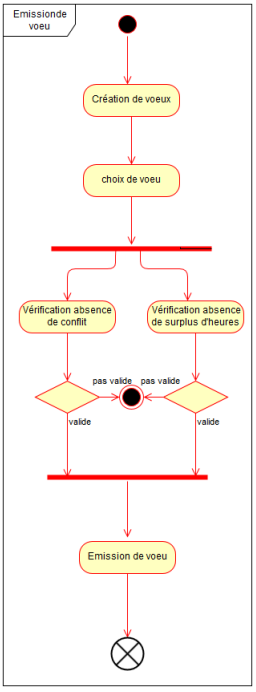

FIGURE 24 – Diagramme d'activité de l'opération Emettre

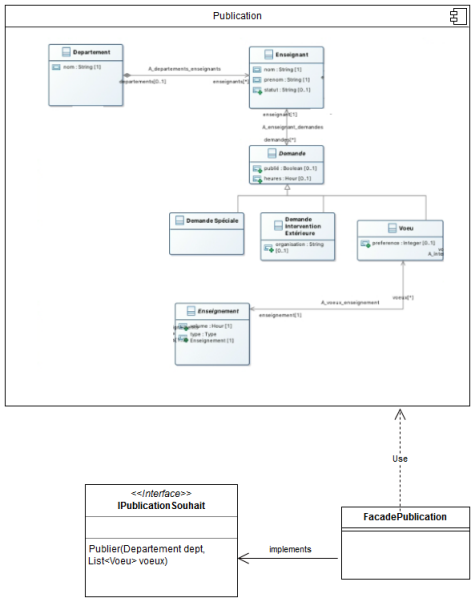

FIGURE 25 – Diagramme de classe du composant Pulication

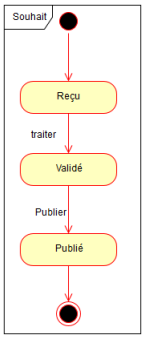

FIGURE 26 – Diagramme état transition du composant Publication

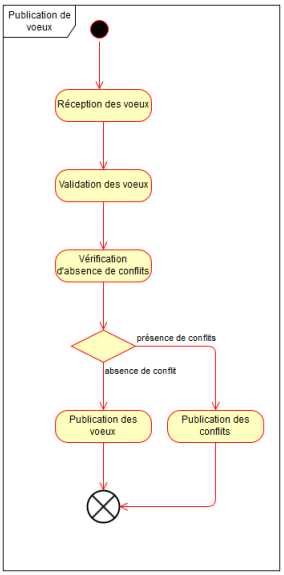

FIGURE 27 – Diagramme d'activité de l'opération Publier

Comportement

Le comportement interne du composant Publication est décrit à l'aide des diagrammes état-transition [26] et d'activité [**??**].
Le comportement implémente toutes les situations décrites par ses cas d'utilisations.

3.3.3 Composant Affectation

Structure

Finalement, le composant Affectation[28], sur le même principe sera présenté par sa façade. Cette façade implémente l'interface IAffectationSouhait offrant la fonction Affecter au chef du département.

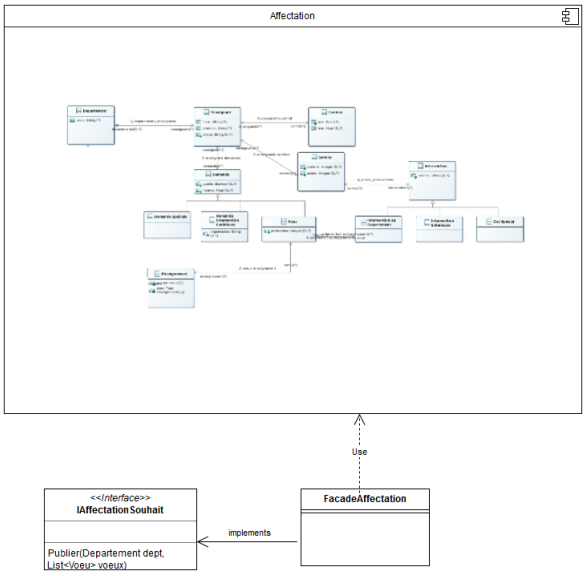

FIGURE 28 – Diagramme de classe du composant Affectation

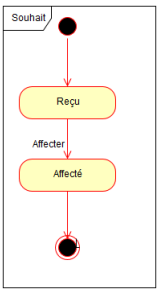

FIGURE 29 – Diagramme état transition du composant Affectation

Comportement

Le comportement interne du composant Affectation est décrit à l'aide des diagrammes état-transition [29] et d'activité [30].
Le comportement implémente toutes les situations décrites par ses cas d'utilisations.

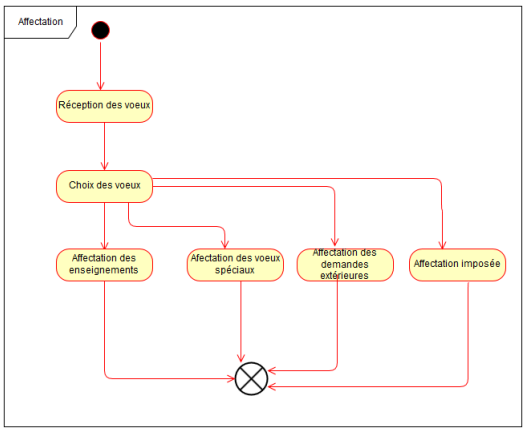

FIGURE 30 – Diagramme d'activité de l'opération Affecter