

Projet « Gestion des services »

Gerson Sunyé
`gerson.sunye@univ-nantes.fr`

Résumé

L'objectif de ce projet est de fournir un outil réparti capable de gérer les services de différents enseignants.

Table des matières

1	Introduction	4
1.1	Objectifs	4
1.2	Prérequis	4
1.3	Travail à rendre	4
1.4	Critères d'évaluation	5
1.5	Organisation	5
1.6	Préambule	5
2	Spécification des exigences logicielles	6
2.1	Introduction	6
2.1.1	Objectif	6
2.1.2	Portée du document	6
2.1.3	Définitions, acronymes et abréviations	6
2.1.4	Références	6
2.1.5	Vue d'ensemble	6
2.1.6	Organisation du chapitre	6
2.2	Description générale	6
2.2.1	Fonctions du produit	8
2.2.2	Caractéristiques et classes d'utilisateurs	8
2.2.3	Environnement opérationnel	9
2.2.4	Contraintes de conception et de mise en œuvre	9
2.2.5	Documentation utilisateur	10
2.2.6	Hypothèses et dépendances	10
2.2.7	Exigences reportées	10
2.3	Exigences fonctionnelles	10
2.3.1	UC—Affecter des enseignements	10
2.3.2	UC—Publier des souhaits	10
2.3.3	UC—Analyser des demandes	10
2.3.4	UC—Émettre un souhait	10
2.3.5	UC—Publier des souhaits	11
2.3.6	UC—Consulter des enseignements	11
2.4	Exigences non-fonctionnelles	11
2.4.1	Interface utilisateur	11
2.4.2	Interface matérielle	11
2.4.3	Interface logicielle	11
2.4.4	Interfaces ou protocoles de communication	11
2.4.5	Contraintes de mémoire	11
2.4.6	Hypothèses et dépendances	11
2.4.7	Correction	11
2.4.8	Interopérabilité	12

2.4.9	Portabilité	12
2.4.10	Exigences de performance	12
2.4.11	Maintenabilité	12
2.4.12	Exigences de sûreté	12
2.4.13	Exigences de sécurité	12
2.4.14	Attributs de qualité logicielle	12
2.5	Autres exigences	12
2.5.1	Persistance	12
2.5.2	Système réparti	13
2.5.3	Outils de développement	13
3	Analyse du domaine	15
3.1	Introduction	15
3.2	Définition des cas d'utilisation	15
3.2.1	UC—Affecter des enseignements	15
3.2.2	UC—Valider des souhaits et publication des possibles conflits	20
3.2.3	UC—Analyser des demandes	20
3.2.4	UC—Émettre un souhait	21
3.2.5	UC—Publier des souhaits	24
3.2.6	UC—Consulter des enseignements	24
3.3	Diagramme de classes métiers	25
4	Dictionnaire de données	27
5	Architecture	30
5.1	Introduction	30
5.2	Vue physique	30
5.3	Vue logique	30
5.4	Vue des processus	30
5.5	Vue de la fiabilité	31
5.6	Réponses aux exigences non-fonctionnelles	31
5.6.1	Gestion de la concurrence	31
5.6.2	Gestion de la persistance	31
5.6.3	Gestion de la sécurité	31
5.7	Architecture technique : traduction de UML en code source	31
5.7.1	Règles de traduction des types de base	31
5.7.2	Règles de traduction des classes	31
5.7.3	Règles de traduction des associations, agrégations composites et agrégations partagées	31
5.7.4	Règles de traduction des composants	31
5.7.5	Autres règles	31
5.8	Patrons architecturaux utilisés	31
5.8.1	Patron A	32
5.8.2	Patron B	32
5.8.3	Patron C	32
5.8.4	Patron D	32
5.8.5	Patron D	32
5.8.6	Patron E	32
5.9	Choix techniques - Distribution des processus	32
5.10	Conclusion	32

6	Spécification des composants	33
6.1	Introduction	33
6.1.1	Objectif	33
6.1.2	Organisation du chapitre	33
6.2	Description des composants	33
6.2.1	Le composant A	33
6.3	Interactions	33
6.3.1	Cas d'utilisation UC1	33
6.4	Spécification des interfaces	34
6.4.1	Interface A	34
6.4.2	Interface B	34
6.4.3	Interface C	34
6.5	Spécification des types utilisés	34
6.5.1	Task	34
6.6	Conclusion	34
7	Conception détaillée	35
7.1	Introduction	35
7.2	Répertoire des décisions de conception	35
7.2.1	Patron A	35
7.2.2	Patron B	35
7.2.3	Patron C	35
7.3	Spécification détaillée des composants	35
7.3.1	Composant A	36
7.3.2	Composant B	36
7.3.3	Composant C	36
7.3.4	Composant D	36
7.4	Spécification détaillée des classes	36
7.5	Conclusion	37

Chapitre 1

Introduction

1.1 Objectifs

L’objectif de ce projet est de concevoir et de mettre en oeuvre un logiciel réparti permettant la gestion des services de différents enseignants. Cette application pourra être utilisée de manière indépendante dans les différents départements de l’université, par les enseignants et les chefs de département, pour réaliser le choix des services de chacun des enseignants.

Pour répondre à ce besoin, vous disposez d’un document d’analyse du domaine, présentant de manière relativement complète le domaine “Gestion des services” au sein du département Informatique, d’un document de spécification des exigences logicielles et d’un dictionnaire de données, qui constitue une liste non-exhaustive de l’ensemble des termes relatifs au domaine étudié. Des informations complémentaires pourront être fournies dans le forum de discussions dédié au projet.

1.2 Prérequis

Pour réaliser ce projet, l’étudiant doit maîtriser les langages et outils suivants :

1. UML, le langage de modélisation unifié.
2. OCL, le langage d’expression de contraintes de UML.
3. Le langage Java, ou éventuellement un langage JVM, comme Kotlin ou Groovy.
4. Un outil de test unitaire de type xUnit : JUnit ou TestNG.
5. Un outil pour la gestion et l’automatisation de production des projets logiciels Java, comme Apache Maven ou Gradle.

1.3 Travail à rendre

Le projet se compose de deux livrables :

1. Un document PDF contenant la description de la conception.
2. Une archive contenant le code source.

La conception de l’application sera composée de trois parties :

1. L’architecture.
2. La conception préliminaire (spécification des composants) ;

3. La conception détaillée.

Les chapitres qui décrivent la conception doivent être organisés en fonction de la solution proposée (de ses fonctionnalités). Suivez le modèle proposé dans ce document.

1.4 Critères d'évaluation

L'évaluation sera faite exclusivement sur le respect des exigences logicielles décrites dans le Chapitre 2. Lisez attentivement toutes les exigences avant de commencer votre projet.

1.5 Organisation

Règles à respecter pendant la réalisation du projet :

1. La réalisation du projet sera faite par des groupes d'au moins 2 et au plus 4 étudiants. Les projets qui ne respectent pas cette règle ne seront pas évalués.
2. Chaque groupe déposera ses livrables sur le serveur Madoc : <http://madoc.univ-nantes.fr/>, **exclusivement**. Les projets envoyés par un autre moyen ne seront pas évalués. En cas de problèmes avec le dépôt d'un fichier, envoyez un message au responsable du module.
3. Pour éviter tout problème de compatibilité, les rapports seront rendus en format PDF, **exclusivement**.
4. Le code source et les tests seront rendus en une archive **tar.gz**, qui contiendra aussi un fichier **pom.xml** (Maven) ou **build.gradle** (Gradle) permettant la compilation du système et le lancement de la validation. Vous pouvez préciser s'il s'agit d'un projet Eclipse, IDEA ou un autre.
5. Les fichiers Java compilés (*.class) ainsi que les éventuelles bibliothèques utilisées ne doivent pas être rendus avec le code source.

1.6 Préambule

Les sources de ce document sont disponibles sur l'adresse suivante :

<https://github.com/sunye/tp-gestion-services>

Si vous avez des commentaires sur le document ou si vous souhaitez me communiquer des erreurs, merci de le faire directement sur Github. Vous pouvez aussi modifier directement les sources du document et m'envoyer ensuite un *pull request*.

Chapitre 2

Spécification des exigences logicielles

2.1 Introduction

Ce document énumère les exigences du projet « Gestion des services ». Il suit la norme IEEE 830-1998.

2.1.1 Objectif

Ce document a pour objectif de décrire les exigences fonctionnelles et non-fonctionnelles du projet « Gestion des services ». La description du comportement utilisateur attendu du système fera référence à l'analyse du domaine, présentée dans le Chapitre [3](#)

2.1.2 Portée du document

Ce document s'applique seulement au développement de composants « métier » du système. Il ne concerne pas l'interface utilisateur.

2.1.3 Définitions, acronymes et abréviations

Acronyme	Description
CM	Cours magistral
IHM	Interface Home-Machine
REST	Representational State Transfer
TD	Travaux dirigés
TP	Travaux pratiques

2.1.4 Références

2.1.5 Vue d'ensemble

2.1.6 Organisation du chapitre

2.2 Description générale

Deux rôles se confrontent lors de la gestion des services : le département et les enseignants. Le département doit :

- Publier la liste des enseignants gérés par le département et donc y réalisant leur service. A chaque enseignant est associée le nombre d'heures qu'il doit réaliser. C'est en fait une fourchette MIN..MAX.

Cours	Semestre	CM		TD		TP	
		HE	nb	HE	nb	HE	nb
S4I0100	1	18	1	15	3	18	5
S3I0200	1	18	1	15	3	18	6
S1I0100	2	16	1	20	1	-	-
S5I0400	2	5	1	20	1	20	2

TAB. 2.1 : Exemple de cours disponibles dans le département

Num	Nom Module	Catégorie	Volume Etudiant	Nb groupes demandés	éqTD
1	S4I0200	CM	15	1	22,5
2	S4I0100	TD	18	1	18
3	S4I0100	TP	15	2	20

TAB. 2.2 : Exemple de demande de service

Ces nombres varient d'un enseignant à l'autre et sont exprimés dans une unité éqTD (heures équivalentes Travaux Dirigés). Pour information, 1h de cours représente 1h30 de TD, alors 1h30 de TP représente 1h de TD.

- Publier la liste des enseignements qui doivent être couverts, avec les informations suivantes (cf. exemple Table 2.1) :
 - Nom du module.
 - Publics concernés (année d'étude, parcours).
 - Volume horaire par étudiants (Cours, TD, TP).
 - Nombre de groupes (Cours, TD, TP).
 - Semestre dans l'année universitaire.
- Affecter les enseignants et visualiser leur nombres d'heures réalisées dans leur service (en éqTD)

Les enseignants doivent :

- Prendre connaissance des enseignements disponibles sur le département.
- Énoncer des préférences quant aux cours qu'ils souhaitent couvrir. Pour ce faire, un enseignant constitue la liste des enseignements qu'il souhaite couvrir en les classant du plus souhaité (en position 1) au moins souhaité (dernière position). Cf. Table 2.2 pour un exemple. Pour être acceptable, une demande doit correspondre à un volume horaire au moins égale à 1,5 fois le service que doit effectuer l'enseignant.
- Prendre connaissance des préférences énoncées par les autres enseignants pour essayer de résoudre les problèmes éventuels autant que faire se peut.
- Prendre connaissance des cours qui leur ont été affectés, les préparer, les assurer.

2.2.1 Fonctions du produit

Le système doit permettre aux enseignants de émettre des souhaits de service et au directeur du département de valider ces souhaits. Il n'est pas question de réaliser un outil permettant d'optimiser automatiquement les affectations d'enseignants. C'est un outil d'aide à la gestion qui doit être proposé aux départements et aux enseignants.

Fonctionnalités souhaitées pour le département :

- Mémorisation des services des années passées.
- Constitution de la liste des enseignements à couvrir.
- Récupération des choix des enseignants.
- Affichage des services suivant différent modes (globalement, par modules, par enseignants, par formation, seulement les souhaits, seulement les affectations effectives, les deux, etc.).
- Affectation des enseignants sur les enseignements.
- Alertes concernant :
 - les enseignements non couverts
 - les enseignements trop demandés
 - les conflits entre les demandes
 - les enseignants n'entrant pas dans le cadre de leur contrat de service statutaire.
- La possibilité d'effectuer des affectations et des modifications sans les rendre publiques.
- La possibilité de valider un état courant, c'est à dire de le rendre public.

Fonctionnalités souhaitées pour les enseignants :

- Mémorisation du service de l'enseignant des années passées, éventuellement ses choix.
- Récupération des choix des autres enseignants (globalement ou à la demande).
- Afficher les services suivant différent modes (globalement, par modules, par enseignants, par formation, seulement les souhaits, seulement les affectations effectives, les deux, etc.).
- Repérage facile des enseignements non encore demandés.
- Alertes concernant :
 - les conflits entre les demandes
 - les enseignants dépassant le minimum ou le maximum correspondant à leur contrat de service par rapport aux affectations réalisées. Plus particulièrement pour l'enseignant connecté.
- Alarmes lorsque l'affectation effective des services ne respecte pas les choix de l'enseignant.
- Publication du choix.

Attention, un enseignant doit pouvoir changer sa liste de choix à n'importe quel moment.

2.2.2 Caractéristiques et classes d'utilisateurs

Les utilisateurs du système seront des enseignants avec une certaine maîtrise de l'informatique.

2.2.3 Environnement opérationnel

Le système sera déployé sur les machines des enseignants du département d'informatique : des ordinateurs de bureau et portables, sous Windows, Linux, OSX et FreeBSD. Il sera utilisé lorsque les ordinateurs sont connectés sur un réseau ou déconnectés en mode nomade.

2.2.4 Contraintes de conception et de mise en œuvre

Langage de programmation

REQ 1 (*Java*)

L'application doit être mise en œuvre dans le langage de programmation Java (version ≥ 1.7).

Langage de conception

REQ 2 (*UML*)

Les diagrammes utilisés comme support à la conception doivent respecter la norme UML (version > 2.4) et le langage d'expression de contraintes OCL (version > 2.4).

Conception

La conception sera utilisée pour spécifier la mise en œuvre de la solution.

REQ 3 (*Conception claire*)

La conception doit être claire et précise. Les diagrammes de conception doivent être systématiquement accompagnés d'une description textuelle.

REQ 4 (*Conception pour la testabilité*)

La conception doit prioriser la testabilité : le code doit être facilement testable.

REQ 5 (*Cohérence entre diagrammes*)

Les diagrammes de conception doivent être cohérents entre eux.

Mise en œuvre

REQ 6 (*Cohérence entre la conception et le code*)

La mise en œuvre doit être cohérente avec la conception.

REQ 7 (*Lisibilité du code*)

Le code source doit être simple à comprendre et à évaluer : il doit être possible de les compiler et d'exécuter les tests unitaires à partir de la ligne de commandes.

Outils de construction

REQ 8 (*Construction automatique*)

L'utilisation d'outil de construction automatique est obligatoire : Maven ou Gradle.

Outils de développement

REQ 9 (IDE)

L'utilisation d'un environnement de développement (IDE) compatible avec Maven ou Gradle, comme IntelliJ IDEA ou NetBeans, est fortement recommandée.

Bibliothèques et composants logiciels

Les tests unitaires doivent utiliser JUnit (version > 4.10) ou son équivalent pour les autres langages de programmation.

2.2.5 Documentation utilisateur

2.2.6 Hypothèses et dépendances

2.2.7 Exigences reportées

REQ 10 (Import/export CSV)

Le système permettra l'importation et l'exportation de données au format CSV.

REQ 11 (Interface graphique)

Le système sera accessible grâce à une interface graphique simple et ergonomique.

REQ 12 (Interface Web)

Le système sera accessible à travers une interface web.

2.3 Exigences fonctionnelles

2.3.1 UC—Affecter des enseignements

REQ 13 (Affectation)

Confirmation, par le chef du département, d'un souhait d'un enseignant.

2.3.2 UC—Publier des souhaits

REQ 14 (Publication des interventions)

Le système doit permettre au chef du département de publier des interventions, des souhaits et des conflits.

2.3.3 UC—Analyser des demandes

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.

2.3.4 UC—Émettre un souhait

REQ 16 (Expression de souhaits)

Le système doit permettre aux enseignants d'émettre ses souhaits d'enseignement.

2.3.5 UC—Publier des souhaits

REQ 17 (*Publication de souhaits*)

Le système doit permettre aux enseignants de publier leurs souhaits.

2.3.6 UC—Consulter des enseignements

REQ 18 (*Consultation des enseignements*)

Le système doit permettre aux enseignants de consulter les enseignements.

2.4 Exigences non-fonctionnelles

2.4.1 Interface utilisateur

Le système se résume à des composants sans interface utilisateur. Le bon fonctionnement des composants sera validé par des tests unitaires.

2.4.2 Interface matérielle

Non applicable.

2.4.3 Interface logicielle

REQ 19 (*Clarté des interfaces des composants*)

Le système doit être divisé en composants disposant d'interfaces logicielles claires et simples.

2.4.4 Interfaces ou protocoles de communication

REQ 20 (*Protocoles ouverts*)

La communication entre les différents composants du système doit utiliser des protocoles ouverts et disponibles sur différentes plateformes.

2.4.5 Contraintes de mémoire

Non applicable.

2.4.6 Hypothèses et dépendances

2.4.7 Correction

La gestion des conflits étant problématique, il convient d'avoir une application fortement exacte, notamment au niveau de la communication entre les différents utilisateurs (d'autant plus qu'il s'agit d'une application distribuée).

REQ 21 (*Correction*)

La correction de toutes les opérations de tous les classes du système doit être vérifiée par des tests unitaires. Plus particulièrement, les interfaces fournies par les composant doivent être testées de manière exhaustive.

2.4.8 Interopérabilité

Il est aussi difficile de supposer que toutes les applications fonctionnent sur les mêmes systèmes ou sont programmées dans les mêmes langages. Comme l'application doit interroger des bases distantes, elles-mêmes pouvant être gérées par un autre logiciel similaire, nous devons assurer une interopérabilité totale.

REQ 22 (*Interopérabilité*)

Le système doit être interopérable avec d'autres systèmes.

2.4.9 Portabilité

REQ 23 (*Portabilité*)

L'application doit pouvoir s'appliquer à tous les systèmes d'exploitation, donc, le logiciel doit être complètement portable.

2.4.10 Exigences de performance

L'application ne nécessite pas de performances particulières (du fait de l'utilisation par un homme) mais nous devons tout de même assurer des temps de réponse raisonnables aux demandes de consultation des bases distantes.

2.4.11 Maintenabilité

Conventions de code Java

REQ 24 (*Conventions de codage*)

Le code source Java doit respecter le style proposé par Google :
<https://google.github.io/styleguide/javaguide.html>

2.4.12 Exigences de sûreté

2.4.13 Exigences de sécurité

La sécurité influe beaucoup sur la performance d'exécution. Cette dernière n'étant pas prioritaire, nous pouvons demander une forte sécurité (bien que les données manipulées ne soient pas particulièrement sensibles).

REQ 25 (*Identification*)

Le logiciel doit permettre d'identifier et authentifier les différentes applications des différentes personnes.

2.4.14 Attributs de qualité logicielle

2.5 Autres exigences

2.5.1 Persistance

REQ 26 (*Persistance*)

Les données saisies par le département doivent être persistantes. En particulier, les demandes non-publiées des enseignants.

2.5.2 Système réparti

REQ 27 (*Informatique nomade*)

Un utilisateur doit pouvoir consulter et modifier l'état de sa déclaration de service avec son application où qu'il se trouve, même s'il n'est pas connecté à un réseau.

REQ 28 (*Localisation des applications*)

Il n'est pas raisonnable de supposer que toutes les demandes sont hébergées sur la même machine. Il est donc nécessaire de les localiser.

2.5.3 Outils de développement

REQ 29 (*Contrôle de version*)

Le développement du code source du logiciel doit utiliser un système de gestion de versions comme Git ou Subversion.

REQ 30 (*UML Designer*)

Les diagrammes de conception doivent être réalisés sur Eclipse, avec UML Designer de Obeo.

REQ 31 (*Spring*)

Le développement des composants doit se faire grâce au framework Spring.

Liste des exigences

1	REQ (Java)	9
2	REQ (UML)	9
3	REQ (Conception claire)	9
4	REQ (Conception pour la testabilité)	9
5	REQ (Cohérence entre diagrammes)	9
6	REQ (Cohérence entre la conception et le code)	9
7	REQ (Lisibilité du code)	9
8	REQ (Construction automatique)	9
9	REQ (IDE)	10
10	REQ (Import/export CSV)	10
11	REQ (Interface graphique)	10
12	REQ (Interface Web)	10
13	REQ (Affectation)	10
14	REQ (Publication des interventions)	10
15	REQ (Analyse)	10
16	REQ (Expression de souhaits)	10
17	REQ (Publication de souhaits)	11
18	REQ (Consultation des enseignements)	11
19	REQ (Clarté des interfaces des composants)	11
20	REQ (Protocoles ouverts)	11
21	REQ (Correction)	11
22	REQ (Interopérabilité)	12
23	REQ (Portabilité)	12
24	REQ (Conventions de codage)	12
25	REQ (Identification)	12
26	REQ (Persistance)	12
27	REQ (Informatique nomade)	13
28	REQ (Localisation des applications)	13
29	REQ (Contrôle de version)	13
30	REQ (UML Designer)	13
31	REQ (Spring)	13

Chapitre 3

Analyse du domaine

3.1 Introduction

Dans ce chapitre, nous présentons l'ensemble des cas d'utilisation que nous avons dégagés lors de l'analyse du domaine. Nous utiliserons pour cela le canevas proposé par Cockburn [2] que nous compléterons par des instantanés ainsi que par des post-conditions exprimées en OCL (Object Constraint Language) et quelques scénarii. Cette partie constitue une étape clé de la phase de spécification des besoins.

Nous présentons également le diagramme des classes métiers (i.e., le diagramme de classes au niveau analyse) que nous avons construit à partir de l'analyse réalisée jusqu'à présent. Ce diagramme fournit une vue statique et synthétique du domaine "Gestion des services" de notre application. Cette dernière partie constitue également une étape clé de la phase de spécification des besoins.

3.2 Définition des cas d'utilisation

Dans cette section, nous spécifions l'ensemble des cas d'utilisation relatifs au domaine "Gestion des services". Le diagramme présenté dans la Figure 3.1 résume l'ensemble des cas d'utilisation associés de notre système. Ceux-ci peuvent être effectués par deux acteurs différents :

- Le chef de département ;
- l'enseignant.

Dans les sections suivantes, nous fournissons une description plus détaillée de chacun des cas d'utilisation.

3.2.1 UC—Affecter des enseignements

Use Case : UC1 – Affecter des enseignements

CHARACTERISTIC INFORMATION

Goal in the context : Le chef de département réalise la confirmation d'un souhait d'un enseignant (validation) ou impose une intervention dans le département à un enseignant (affectation "forcée"). L'enseignant devra se soumettre aux décisions du chef de département que ce soit pour une validation d'un souhait ou une affectation imposée.

Scope : Département

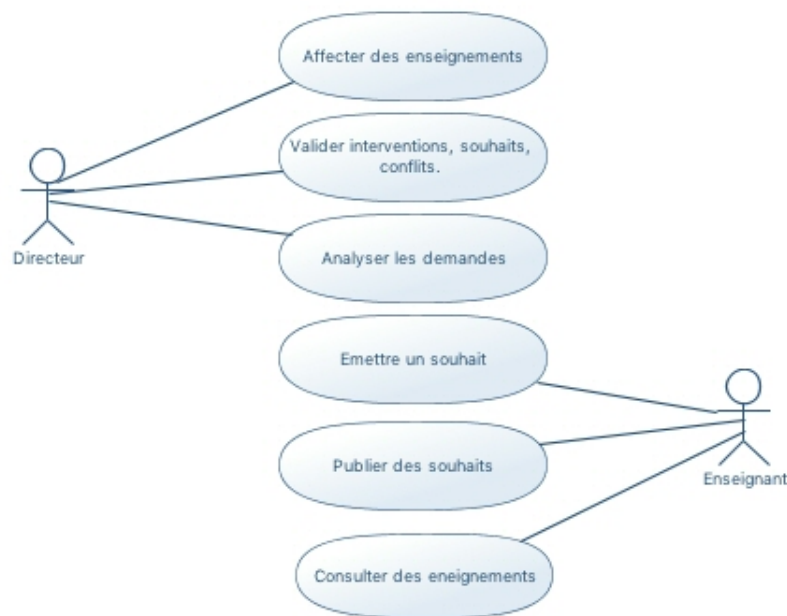


FIG. 3.1 : Cas d'utilisation du domaine "Gestion des services".

Level : Résumé (Summary)

Precondition : Le chef de département connaît les souhaits (seulement ceux déjà publiés par l'enseignant), les possibles conflits et les affectations de l'enseignant concerné.

Success End Condition : L'enseignant est affecté à un ou plusieurs enseignements (souhait validé ou intervention imposée). Le volume horaire total effectué par l'enseignant a été recalculé.

Failed End Condition : L'affectation n'a pas été réalisée (l'enseignement est déjà affecté à un autre enseignant : conflit).

Primary actor : Chef de département.

Trigger : Demande de réalisation d'une affectation faite par le chef de département.

MAIN SUCCESS SCENARIO

1. Le chef de département visualise l'ensemble des souhaits publiés par les enseignants.
2. Le chef de département sélectionne des souhaits, selon certains critères (l'enseignant, le module concerné, ...).
3. Le chef de département choisit un vœu d'un enseignant afin de le valider.
4. Le chef de département valide le vœu, l'enseignement associé au vœu est affecté à l'enseignant concerné : c'est-à-dire que le vœu donne lieu à une intervention effective.
5. Le volume horaire des enseignants concernés est recalculé.

VARIATIONS

- 4 Le chef de département impose une intervention à un enseignant dans son département (affectation imposée).
- 4 Le chef de département valide une demande d'intervention extérieure ou une demande spéciale (remplacement, encadrement de stage, congés, ...) de l'enseignant.

Spécification OCL

```

context UC::AffecterEnseignements(dep:Departement, vœux:Vœu[*])
post:
vœux→forAll(each:Vœu| let int = each.intervention in
  int.oclIsNew() and int.enseignant = each.enseignant and
  int.enseignement = each.enseignement)
-- l'affectation d'un vœu crée une intervention dont l'enseignant et l'enseignement sont les mêmes que ceux du vœu.

```

```

context UC::AffecterDemandeExterieur(dep:Departement, d:DemandeExterieur)
post: let i = d.intervention in
  i.oclIsNew() and t.enseignement = d.enseignant and
  dep.interventions→includes(d.intervention)
-- l'affectation d'une demande extérieure donne lieu à une intervention extérieure qui fait parti de
-- la liste d'affectation de l'enseignant (celui qui a émis cette demande).
-- Cette affectation est effectuée par le département.

```

```

\begin{oc}
context UC::AffecterDemandeSpeciale(dep:Departement, ds:DemandeSpeciale)
post: let speciale = ds.intervention in
  speciale.oclIsNew() and speciale.enseignant = ds.enseignant
-- La DemandeSpeciale réalisée donne lieu à un CasSpecial qui fait parti de la liste d'affectation
-- de l'enseignant (celui qui a émis cete DemandeSpeciale : de.emet).
-- Cette réalisation est effectuée par le département.

```

```

context Departement::imposeInterventionDepartement(dep:Departement, ens:Enseignant, e:Enseignement)
post: Intervention.allInstances()→exists(i : Intervention |
  i.oclIsNew() and i.enseignement = e and i.enseignant = e)
-- L'enseignement est lié à une unique InterventionDepartement qui appartient à la liste des affectations
-- de l'enseignant ens. C'est le département qui impose cette Intervention.

```

Exemple d'affectation de vœux

La Figure 3.2 présente un exemple d'affectation des 3 vœux d'Alice et Bob. L'affectation donne lieux à la création de trois instances d'intervention.

Exemple d'affectation à partir d'une demande extérieure

La Figure 3.3 présente l'affectation d'un vœu d'enseignement d'un intervenant extérieur, faite par l'enseignant Charles. L'affectation à partir d'une demande extérieure crée une nouvelle intervention extérieure associée à la demande extérieure et à l'enseignant demandeur.

Exemple d'affectation à partir d'une demande spéciale

La Figure 3.4 présente une affectation d'une demande spéciale de congé faite par l'enseignant Eve. L'affectation une instance de **CasSpécial** associé à l'enseignant et à la demande spéciale concernés.

Exemple d'affectation imposée à un enseignant

La Figure 3.5 présente une affectation imposée d'un enseignement de TP à l'enseignant Alice. L'imposition d'une intervention à un enseignant crée une instance de la classe **InterventionDepartement** associée à l'enseignant et à l'enseignement concernés. Les interventions (dans le département) imposées ne sont pas liées à un vœu.

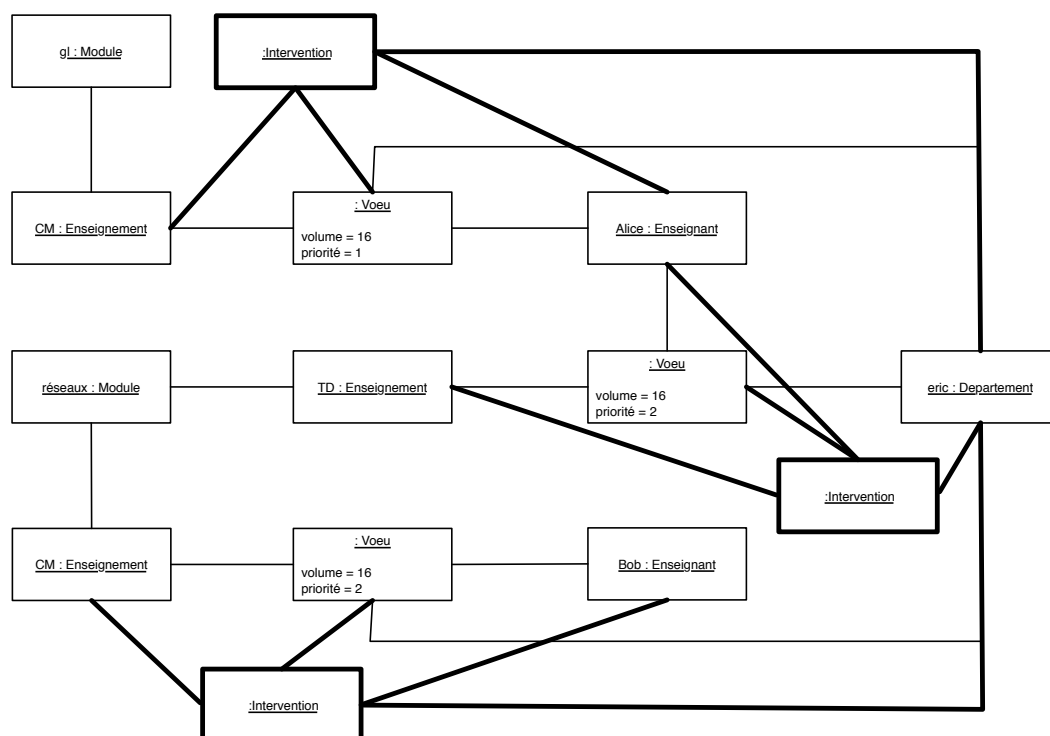


FIG. 3.2 : Diagramme d'objets : Affecter des vœux

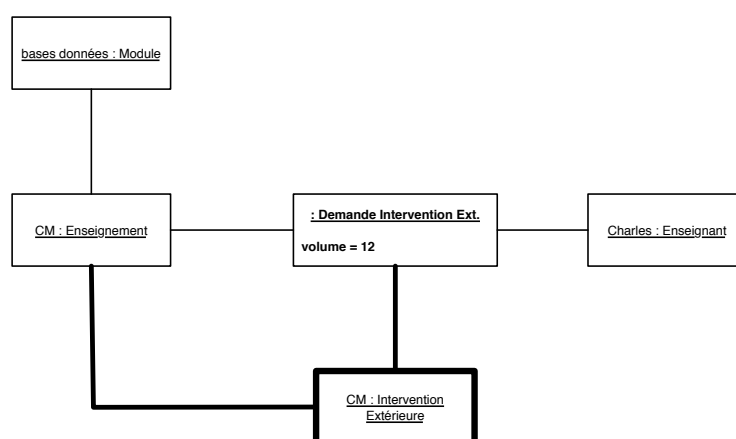


FIG. 3.3 : Diagramme d'objets : Affectation à partir d'une demande extérieure

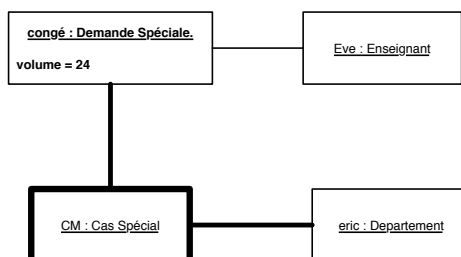


FIG. 3.4 : Diagramme d'objets : Affectation à partir d'une demande spéciale

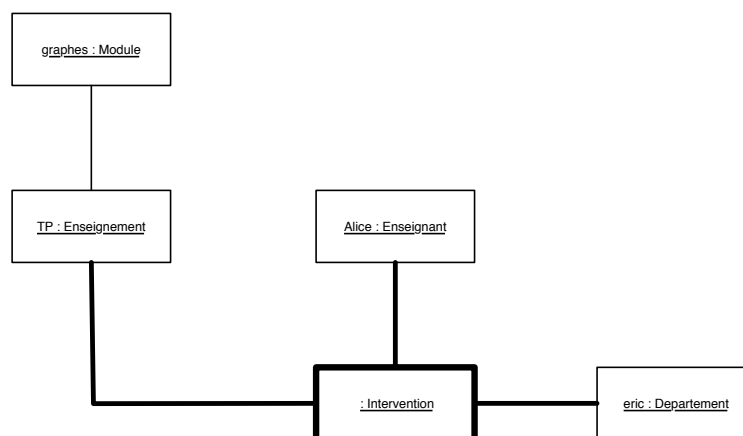


FIG. 3.5 : Diagramme d'objets : Imposition d'une intervention à un enseignant

3.2.2 UC—Valider des souhaits et publication des possibles conflits

Use Case : UC2 – Valider des souhaits et publication des possibles conflits

CHARACTERISTIC INFORMATION

Goal in the context : Le chef de département, par cette action, permet aux autres acteurs du système (en l'occurrence les enseignants) de visualiser les informations concernant l'état actuel des souhaits de l'ensemble des enseignants. L'exécution de ce cas d'utilisation peut faire apparaître de nouveaux conflits pour les enseignants.

Scope : Département

Level : Résumé

Precondition : aucune

Success End Condition : Les enseignants ont accès aux informations concernant l'état actuel des souhaits publiés. Des conflits peuvent apparaître localement pour chaque enseignant.

Failed End Condition : Echec de la publication : les enseignants n'ont toujours pas accès aux informations que le chef de département a tenté de publier.

Primary actor : Chef de département.

Trigger : Demande de publication des interventions, des souhaits et des conflits possibles faite par le chef de département.

MAIN SUCCESS SCENARIO

1. Le chef de département visualise l'ensemble des souhaits non publiés.
 2. Le chef de département sélectionne un ensemble de souhait et les valide.
 3. Le chef de département rend publics les souhaits validés.
 4. Le chef de département publie les nouveaux conflits qui sont apparus après la validation.
-

3.2.3 UC—Analyser des demandes

Use Case : UC3 – Analyser des demandes

CHARACTERISTIC INFORMATION

Goal in the context : Le chef de département effectue une analyse statistique permettant de déterminer la répartition des volumes horaires par enseignants ou la répartition des souhaits et affectations entre enseignants pour une année donnée.

Scope : Département

Level : Résumé

Precondition : /

Success End Condition : /

Failed End Condition : /

Primary actor : Chef de département.

Trigger : Demande de réalisation d'une analyse statistique faite par le chef de département.

MAIN SUCCESS SCENARIO

1. Le chef de département visualise les affectations (interventions) et les souhaits par rapport à une année donnée, selon un critère donné : par module, par enseignant, ou par le nombre d'heures effectives ou souhaitées.
 2. Le chef de département analyse et corrige les affectations d'enseignement.
-

3.2.4 UC—Émettre un souhait

Use Case : UC4 – Émettre un souhait

CHARACTERISTIC INFORMATION

Goal in the context : L'enseignant émet un souhait. Le souhait correspond à vœu sur un ensemble d'enseignements (avec une préférence) du département, une demande d'intervention extérieure au département ou une demande spéciale (arrêt maladie, remplacement,...)

Scope : Département

Level : Summary (Résumé)

Precondition : En cas de vœux sur des enseignements, ces derniers doivent exister et être non affectés à un autre enseignant.

Success End Condition : Le souhait de l'enseignant a été pris en compte mais n'est pas encore publié c'est-à-dire que seul l'enseignant peut le voir (i.e. cette modification est locale). Le volume horaire total (le volume horaire effectif plus la somme des volumes horaires des souhaits pas encore validés) de l'enseignant est modifié (recalculé) par rapport au volume horaire précisé dans le souhait.

Failed End Condition : L'enseignant n'arrive pas à émettre son souhait.

Primary actor : Enseignant.

Trigger : Demande d'émission d'un souhait faite par l'enseignant.

MAIN SUCCESS SCENARIO

1. L'enseignant recopie son souhait de l'année précédente.
2. L'enseignant visualise l'ensemble des enseignements disponibles (i.e. non affectés) dans son département.
3. L'enseignant fait son choix en sélectionnant un ensemble d'enseignements et indique son niveau de préférence pour ces enseignements (voulus ou seulement souhaités).

EXTENSIONS

- 3a. Le vœu génère un conflit (avec d'autres souhaits déjà rendus publics) : le système le notifie à l'enseignant.
- 3b. Le souhait génère un surplus d'heures pour l'enseignant (par rapport à son contrat de services) : le système le notifie à l'enseignant.

VARIATIONS

1. L'enseignant ne possède pas de historique de souhaits. Il commence avec un ensemble vide de souhaits.
3. L'enseignant fait une demande concernant une intervention extérieure au département, précise le volume horaires et l'organisme concerné.
3. L'enseignant décide de faire une demande spéciale souhaité en précisant le type (un congé, un remplacement, ...) et le volume horaire.

Spécification OCL

context UC::EmettreVoeu(ens:Enseignant, m:Module, e:Enseignement, h: **Integer**, p:**Integer**)

post:

ens.voeux→**exists**(v:Voeu | v.enseignement = e **and** v.volume = h **and** v.priorite = p)

-- Un nouveau voeu est ajouté à la liste des voeux de l'enseignant

context UC::EmettreDemandeExterieur(ens:Enseignant, org:**String**, vol:**Integer**)

post:

ens.demandes→**select**(each | each.**oclIsTypeOf**(DemandeInterventionExterieur))→
exists(d | d.**oclIsNew**() **and** d.volume = vol **and** d.organisation = org)

-- Une nouvelle demande d'intervention à l'extérieur est ajoutée à la liste de demandes de l'enseignant.

context UC::EmettreDemandeSpeciale(ens:Enseignant, t:**String**, vol:**Integer**)

post:

ens.demandes→**select**(each | each.**oclIsTypeOf**(DemandeSpeciale))→
exists(d | d.**oclIsNew**() **and** d.volume = vol **and** d.type = t)

-- Une nouvelle demande spéciale est ajouté à la liste de demandes de l'enseignant.

Exemple d'émission d'un souhait

Dans cet exemple, l'enseignant Alice émet le souhait de réaliser deux enseignements :

1. Enseignement de 16h de CM du module Génie Logiciel ;
2. Enseignement de 16h de TD du module Réseaux.

Le diagramme d'objets présenté dans la Figure 3.2.4 illustre ce souhait. Après l'émission du souhait, une instance de la classe **Voeu** est créée pour chaque vœu, ainsi que deux liens, avec l'enseignant et l'enseignement concernés.

Exemple d'émission d'une demande extérieure

Dans cet exemple, l'enseignant Bob fait une demande d'intervention de 12h à l'Institut Mines-Télécom Atlantique. Le diagramme d'objets présenté dans la Figure 3.7 illustre cette demande. L'émission d'une demande extérieure crée une nouvelle demande extérieure associée à l'enseignant.

Exemple d'émission d'une demande spéciale

Dans cet exemple, l'enseignant Charles fait une demande de congés paternité, qui équivaut à 12h de décharge de service. L'émission d'une demande spéciale crée la demande spéciale associée à l'enseignant.

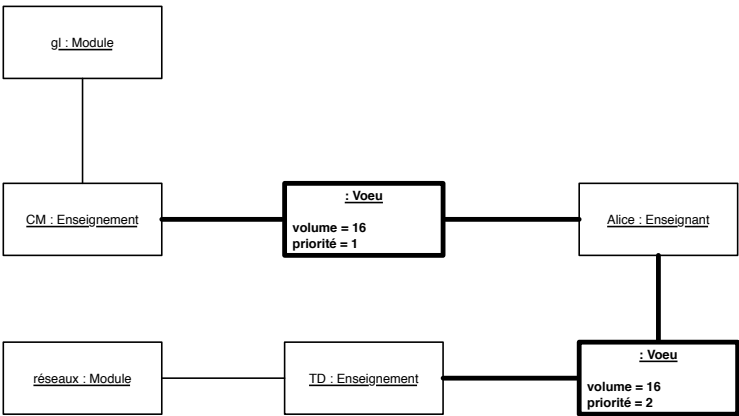


FIG. 3.6 : Diagramme d'objets : Alice émet son souhait de service

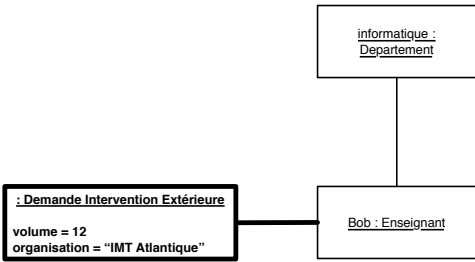


FIG. 3.7 : Émission d'une demande d'intervention extérieure

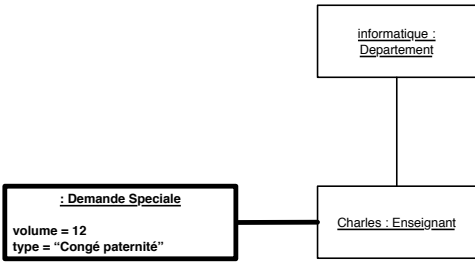


FIG. 3.8 : Émission d'une demande spéciale

3.2.5 UC—Publier des souhaits

Use Case : UC5 – Publier des souhaits

CHARACTERISTIC INFORMATION

Goal in the context : L'enseignant transmet ses souhaits (préalablement sélectionnés) au chef de département qui sera lui même apte, dans un deuxième temps, à les rendre visibles à l'ensemble des enseignants. Un souhait concerne un vœux fait sur des enseignements disponibles (i.e. non affectés), une intervention extérieure au département ou bien encore une demande caractérisée comme étant spéciale (congé, remplacement, ...).

Scope : Département

Level : Résumé

Precondition : Le ou les vœux concernés ont été préalablement réalisés et validés localement par l'enseignant.

Success End Condition : Le chef de département est capable de visualiser le/les souhaits publiés par l'enseignant.

Failed End Condition : Le département ne reçoit pas les souhaits de l'enseignant.

Primary actor : Enseignant.

Trigger : Demande de publication de souhaits faite par l'enseignant.

MAIN SUCCESS SCENARIO

1. L'enseignant choisit de visualiser l'ensemble des souhaits qu'il a réalisé localement mais qu'il n'a pas encore publié.
2. L'utilisateur choisit, parmi ces souhaits, ceux qu'il désire publier et les transmet au chef de département.
3. Le chef du département reçoit les vœux de l'enseignant.

EXTENSIONS

- 2a. Le système lui signifie qu'il n'existe pas de souhaits qui sont en attente d'être publiés.
- 3a. L'enseignant décide de ne publier aucun de ces souhaits et annule sa demande de publication.
- 4a . Le système signifie à l'utilisateur que l'un de ces souhaits (en l'occurrence un vœu) concerne un ou plusieurs enseignements qui ont été affectés ou supprimés et lui demande de modifier son vœu.

3.2.6 UC—Consulter des enseignements

Use Case : UC6 – Consulter des enseignements

CHARACTERISTIC INFORMATION

Goal in the context : L'enseignant doit être capable de visualiser les enseignements afin de consulter ou de modifier ses souhaits. Ainsi, des critères de tri sont mis à sa disposition pour personnaliser sa vue des enseignements dans le système "Gestion des services" : par module, par enseignement d'un module, par enseignant, ou par nombre d'heures.

De plus, il doit choisir les critères de sélection des enseignements qu'il souhaite visualiser : Les enseignements concernés par des vœux publiés ou non ; les enseignements concernés par des vœux déjà validés (affectations) ; ou les enseignements affectés et ceux non affectés.

L'enseignant peut choisir de visualiser les enseignements des années précédentes mais il ne pourra effectuer aucune modification sur ces années. Par défaut, la visualisation se fera sur l'année en cours. L'utilisateur choisit l'ensemble des critères de tri et de sélection sur les enseignements qu'il souhaite. Il peut donc entièrement paramétrer sa vue des enseignements.

Scope : Département

Level : Résumé

Precondition : /

Success End Condition : /

Failed End Condition : /

Primary actor : Enseignant.

Trigger : Demande de consultation des enseignements selon différents critères faite par l'enseignant.

MAIN SUCCESS SCENARIO

1. L'enseignant décide de visualiser l'ensemble des enseignements selon ses critères.
2. Le système renvoie l'ensemble des enseignements correspondants aux critères demandés.

EXTENSIONS

- 2a. Le système ne renvoie rien puisqu'aucun enseignement ne correspond aux critères demandés.

3.3 Diagramme de classes métiers

Afin d'achever cette phase de spécification des besoins, nous fournissons une modélisation statique et synthétique du domaine de notre application. La figure 3.9 représente ce domaine "Gestion des services" :

Le domaine "Gestion des services" est composé d'un ensemble de départements identifiés par leur nom (ex : département "Informatique", département "Mathématiques", ...). Chacun de ces départements est constitué de parcours et possède un certain nombre d'enseignants qui lui sont rattachés. Un parcours a un nom (ex : "Licence informatique", "Master Alma", etc.).

Un module est défini par un nom, une année d'étude (ex : 2^e année) et un semestre (1^e ou 2^e). Il se décompose en plusieurs enseignements avec pour chacun un volume horaire et un coefficient : il peut s'agir de cours magistraux (CM), de travaux dirigés (TD) ou de travaux pratiques (TP). Chaque module a pour responsable un enseignant qui se doit, en principe, d'assurer un certain nombre d'enseignements dans ce module (le plus souvent les cours magistraux).

Les enseignants sont caractérisés par leur nom et leur statut et doivent chacun remplir un contrat de service par année, c'est-à-dire effectuer un certain nombre d'heures minimum d'enseignement par année (calculé à partir des coefficients des enseignements). Il est important de noter que le contrat de service est uniquement fonction du statut de l'enseignant et est indépendant du département auquel il est rattaché.

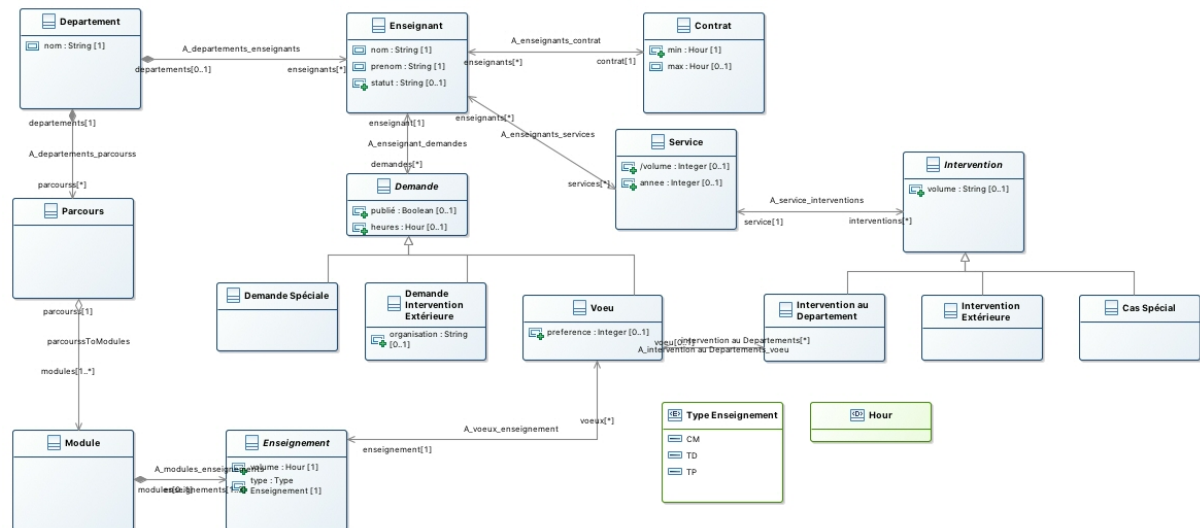


FIG. 3.9 : Diagramme de classes métiers du domaine “Gestion des services”

Chaque enseignant peut émettre un certain nombre de souhaits afin d’indiquer la manière dont il désire remplir son contrat de service. Ces souhaits peuvent être des vœux (indications du ou des enseignements que l’enseignant souhaite dispenser ainsi que ses préférences les concernant), des demandes d’intervention extérieure (dans une entreprise, une autre école...) ou des demandes spéciales (congrés, arrêts maladie, encadrements de stages ou de TER...).

Il est ensuite du ressort du chef du département concerné de décider de valider ou non ces souhaits, i.e. d’affecter ou non à l’enseignant concerné les interventions correspondantes aux souhaits émis. Toutefois, pour régler certains conflits, le chef d’un département se réserve le droit d’imposer une affectation à un enseignant sans que cet enseignant en ait auparavant formulé le souhait.

Chapitre 4

Dictionnaire de données

Dans une première partie, nous fournissons le dictionnaire des données que nous avons construit suite à notre analyse du cahier des charges. Il s'agit d'un listing de l'ensemble des termes relatifs au domaine étudié ('à savoir le domaine "Gestion des services") ainsi que leur définition précise.

Notion	Définition	Traduit en	Nom informatique
Affectation	Action de déterminer une intervention , i.e. d'associer un enseignant à un enseignement donné. C'est le chef de département qui est chargé de déterminer les différentes affectations en fonction, le plus souvent, des vœux réalisés par les divers enseignants .		
Chef de département	Acteur du système. Il est le responsable d'un département : il gère les modules (ainsi que les enseignements associés), les enseignants et leurs interventions pour son département .	Acteur	ChefDepartement
Conflit	Fait que deux vœux soit incompatibles (i.e. que deux enseignants aient émis les mêmes choix concernant un ou des enseignements). C'est au chef de département de régler les conflits en réalisant les affectations.		
Contrat de service	Nombre d'heures minimum (et parfois nombre d'heures maximum) d' enseignements à effectuer pour un enseignant donné. Il est indépendant des départements dans lesquels l' enseignant intervient : il est unique et seulement déterminé par le statut de l' enseignant .	Classe	ContratDeService
Département	Entité administrative (d'une université) identifiée par un nom. Il comprend un ensemble de modules et d' enseignants qui lui sont rattachés. Chaque département a pour responsable un chef de département . Plusieurs enseignants peuvent donner des enseignements pour le compte de chaque département .	Classe	Departement

Enseignant	Personne "physique" travaillant pour le compte d'un département et identifiée par son nom, son prénom et son statut. Un enseignant peut "intervenir" dans différents départements pour dispenser un certain nombre d' enseignements . C'est un également un acteur du système puisqu'il peut effectuer des vœux concernant les enseignements qu'il désire donner.	Classe et Acteur	Enseignant
Enseignement	Entité administrative représentant un cours dans un module donné. Il existe trois types d'enseignement avec, pour chacun, un coefficient différent en terme de volume horaire : <ul style="list-style-type: none"> • CM : cours magistral se déroulant le plus souvent dans un amphithéâtre, un CM vaut 3/2 d'un TD ; • TD : travaux dirigés se déroulant dans une salle de cours classique ; • TP : travaux pratiques se déroulant dans des salles particulières dédiées à cet effet (salles machines, laboratoires...), un TP vaut 2/3 d'un TD (ou 1 TD, cela dépend du statut de l'enseignant concerné). 	Classe	Enseignement, CM, TD, TP
Intervention	Fait qu'un enseignant intervienne dans un département donné. Elle a pour origine, la plupart du temps, un souhait formulé par un enseignant . Cette intervention peut être de différentes natures : <ul style="list-style-type: none"> • il peut s'agir d'une intervention dans le département, c'est-à-dire qu'un enseignement soit affecté à un enseignant pour un volume horaire donné ; • il peut s'agir d'une intervention extérieure (dans une entreprise, une autre école...), d'un volume horaire donné ; • il peut s'agir d'un cas spécial (congés, maladies, encadrement d'un stage ou d'un TER...), toujours d'un volume horaire donné. 	Classe	Intervention, Intervention Département, Intervention Extérieure, Cas Spécial
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.		
Module	Entité administrative (d'un département) regroupant un ensemble d' enseignements concernant un sujet donné. Chaque module est identifié par un nom (intitulé du module) et caractérisé par une année d'étude, un nom de parcours et un semestre.	Classe	Module

Publication	Action de rendre public (i.e. accessible à tous les utilisateurs du système, qu'ils soient enseignant ou chef de département) un ou des souhais formulés par un ou des enseignants .	Attribut	"publié" dans la classe Souhait
Souhait	Fait qu'un enseignant effectue une demande d'un certain type concernant les interventions qu'il souhaite réaliser pour remplir son contrat de service . Ce souhait peut être : <ul style="list-style-type: none"> • un vœu; • une demande d'intervention extérieure (dans une autre école, une entreprise...) d'un volume horaire donné ; • une demande spéciale (congé, arrêt maladie, encadrement d'un stage ou d'un TER...) d'un volume horaire donné. <p>Un souhait peut être à l'origine d'une intervention affectée (déterminée) par le chef de département (ou plusieurs dans le cas d'un vœu).</p>	Classe	Souhait, Vœu, DemandeInterventionExtérieure, DemandeSpéciale
Vœu	Choix fait par un enseignant , pour un département donné, indiquant quels enseignements il désire dispenser ainsi que ses préférences concernant ces enseignements . Les préférences pour un enseignement sont déterminées de la manière suivante : <ul style="list-style-type: none"> • 1 : si cet enseignement est réellement souhaité par l'enseignant; • 0 : si cet enseignement est toléré par l'enseignant. <p>Un vœu pourra, après validation par le chef de département, faire l'objet d'une ou plusieurs interventions.</p>	Classe	Vœu

Chapitre 5

Architecture

5.1 Introduction

Dans ce chapitre, vous devez fournir une description générale de l’architecture de votre système et décrire comment vous répondez aux exigences non-fonctionnelles énoncés précédemment dans la Section 2.4.

5.2 Vue physique

Utilisez un diagramme de déploiement UML pour décrire l’architecture physique de l’application : les nœuds logiques, les protocoles de communication, le déploiement des artefacts logiciels (bibliothèques, autres logiciels, etc.). N’oubliez pas de décrire votre diagramme.

Si l’architecture physique répond à une ou des exigences logicielles, n’oubliez pas de le mentionner.

FIG. 5.1 : Diagramme de déploiement du système “Gestion des services”

Utilisez également le diagramme de déploiement, mais au niveau instance, pour fournir des exemples de déploiement du système.

FIG. 5.2 : Diagramme d’instances modélisant un déploiement possible du système “Gestion des services”

5.3 Vue logique

Utilisez le diagramme de paquetages UML pour décrire l’organisation du code source de votre application. N’oubliez pas de le commenter.

Si l’architecture logique répond à une ou des exigences logicielles, n’oubliez pas de le mentionner.

5.4 Vue des processus

Présentez ici les différents processus d’exécution dans votre système.

5.5 Vue de la fiabilité

Énumérez ici les choix architecturaux faits pour assurer la fiabilité du système.

5.6 Réponses aux exigences non-fonctionnelles

Expliquez, dans cette section, la réponse de votre solution aux exigences non-fonctionnelles. Les sous-sections présentées ici ne constituent pas une liste exhaustive.

5.6.1 Gestion de la concurrence

5.6.2 Gestion de la persistance

5.6.3 Gestion de la sécurité

5.7 Architecture technique : traduction de UML en code source

Expliquez, dans cette partie, l'ensemble de règles que vous utiliserez par la suite pour traduire nos diagrammes UML (découlant de notre analyse et de notre conception) en code source (classes d'implémentation).

5.7.1 Règles de traduction des types de base

5.7.2 Règles de traduction des classes

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

5.7.4 Règles de traduction des composants

5.7.5 Autres règles

5.8 Patrons architecturaux utilisés

Fournissez, dans cette dernière partie, une liste exhaustive des patrons de conception que vous allez utiliser pour mettre en œuvre notre application. Pour chacun de ces patrons, donnez une courte description ainsi que les raisons pour lesquelles vous avez choisi de les mettre en œuvre.

5.8.1 Patron A**5.8.2 Patron B****5.8.3 Patron C****5.8.4 Patron D****5.8.5 Patron D****5.8.6 Patron E****5.9 Choix techniques - Distribution des processus**

Nous allons ici expliciter les différents choix techniques que nous avons choisis et les réponses technologiques aux différentes contraintes que notre système implique. Nous allons donc vous présenter l'environnement général de développement puis énoncer les 4 contraintes que nous avons déterminées de notre logiciel.

5.10 Conclusion

Chapitre 6

Spécification des composants

6.1 Introduction

6.1.1 Objectif

Préciser les objectifs de ce chapitre.

6.1.2 Organisation du chapitre

Cette section décrit le contenu du reste du chapitre et explique comment le document est organisé.

6.2 Description des composants

Établir les frontières du système.

Division du système en composants.

Décrire le comportement souhaité des composants.

6.2.1 Le composant A

FIG. 6.1 : Le composant A et ses interfaces

6.3 Interactions

Décrivez, à haut-niveau, la collaboration entre les composants majeurs, pendant la réalisation des cas d'utilisation. Utilisez des interactions, c'est à dire, des diagrammes de séquence et des diagrammes de communication, pour décrire les cas nominaux et extra-nominaux des cas d'utilisation.

Ne vous limitez pas à une seule interaction par cas d'utilisation

6.3.1 Cas d'utilisation UC1

FIG. 6.2 : Interaction – Création d'une tâche

FIG. 6.3 : Interaction – Création d'une tâche répétitive

6.4 Spécification des interfaces

6.4.1 Interface A

Présentation de l'interface en UML (ou HUTN). Description du comportement de chaque opération. Spécification éventuelle des pré-conditions en OCL.

- **getTasksForDate(Date) : Task[*]**
Retrouve un ensemble de tâches pour une date donnée.

```
getTasksForDate(Date) : Task[*]
pre:
-- Le package \emph{listings} n'est pas compatible avec UTF8.
-- Donc, n'utilisez pas d'accent.
```

- **removeAllTalks()**
Efface toutes les tâches, éteint le serveur et débranche son câble de la prise électrique.
- **getTaskForName(String, Number) : Task**
Retrouve une tâche à partir de son nom et son numéro de téléphone.

6.4.2 Interface B

6.4.3 Interface C

6.5 Spécification des types utilisés

Spécifiez ici les types utilisés par les interfaces (seulement ceux qui ne font pas partie des types de base UML).

6.5.1 Task

La classe **Task** représente une Tâche. Elle possède les attributs suivants :

name : le nom de la tâche.

isActive : détermine si la tâche est activée ou non.

FIG. 6.4 : La classe Task

6.6 Conclusion

Chapitre 7

Conception détaillée

7.1 Introduction

7.2 Répertoire des décisions de conception

Cette section doit contenir un répertoire regroupant l'ensemble de nos décisions de conception concernant votre système.

Si vous utilisez des patrons de conception, ou *design patterns*, décrivez leur utilisation dans les sections suivantes.

7.2.1 Patron A

Description de l'utilisation.

7.2.2 Patron B

Description de l'utilisation.

7.2.3 Patron C

Description de l'utilisation.

7.3 Spécification détaillée des composants

Cette section doit contenir une description détaillée de chacun des composants du système, dont l'interface a été présentée dans le Chapitre 6.

Décrivez, pour chaque composant, sa structure sous la forme d'un diagramme statique (en l'occurrence un diagramme de classes avec paquetages) et son comportement interne sous la forme de diagrammes dynamiques.

Comme UML propose plusieurs types de diagrammes statiques, pensez à bien choisir le diagramme qui permet de mieux spécifier ce que vous souhaitez décrire :

1. Les diagrammes d'activités sont utiles à description d'algorithmes non-triviaux, comme la résolution de conflits, par exemple.
2. Les diagrammes état-transition sont utilisés à décrire des comportements qui dépendent de l'état interne d'un composant ou d'une classe.

3. Les itérations sont utiles à la description d'enchaînement de messages entre objets. Ils sont très utilisés pour décrire des comportements nominaux et extra-nominaux, qui peuvent se traduire en tests.

N'oubliez pas que les deux premiers diagrammes représentent des classes et des opérations, alors que le dernier représente des objets et des messages envoyés entre ces objets. Ils ne sont pas au même niveau.

7.3.1 Composant A

Structure

FIG. 7.1 : Diagramme de classes du composant A

Comportement

FIG. 7.2 : Diagramme état-transition du composant A

Explications détaillées.

FIG. 7.3 : Diagramme d'activités de l'opération **a()** du composant **A**

Explications détaillées.

FIG. 7.4 : Diagramme de séquences de l'opération **a()** du composant **A** (cas nominal)

Explications détaillées.

FIG. 7.5 : Diagramme de séquences de l'opération **a()** du composant **A** (cas extra-nominal)

Explications détaillées.

7.3.2 Composant B

7.3.3 Composant C

7.3.4 Composant D

7.4 Spécification détaillée des classes

Décrivez, dans cette section, les classes ou les opérations dont la mise en œuvre est complexe. Il ne s'agit pas de spécifier toutes les classes et opérations, mais seulement celles dont la traduction en code

n'est pas triviale.

Utilisez le langage OCL et des diagrammes dynamiques pour les décrire.

7.5 Conclusion

Bibliographie

- [1] Persistance xml avec hibernate. <http://www.xml-persistence.net/file/xmlpersistence.pdf>.
- [2] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 2000.