WebTask : un applicatif web de gestion de tâches

Stéphane Guillebaud

le cnam

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

PARIS

MEMOIRE

Présenté en vue d'obtenir le

DIPLOME D'INGENIEUR CNAM

Spécialité : INFORMATIQUE

OPTION : Architecture et Ingénierie des Systèmes et des Logiciels

par

Stéphane Guillebaud

WebTask : Un applicatif web de gestion de tâches

Soutenu le 08 avril 2013

JURY

PRESIDENT : Yann Pollet (Professeur, CNAM)

MEMBRES : Jean-Michel Douin (Professeur, CNAM)
Yves Laloum (Professeur, CNAM)
Dominique Lablanche (Ingénieur CNAM)
Jean-Pierre Tixier (Ingénieur CNAM)

Remerciements

Je remercie Monsieur le Président du jury *Mr Pollet* ainsi que tous les membres du jury *Mr Douin*, *Mr Laloum*, *Mr Lablanche* et *Mr Tixier* pour l'attention qu'ils ont portée à mon travail.

Je tiens à remercier les professeurs de l'UE GLG 204 du CNAM *Jean-Michel Douin*, *Pascal Graffion*, *Marc Le Bihan* pour la qualité de leur cours et leurs conseils précieux.

Je remercie à nouveau *Jean-Michel Douin* pour son implication dans la rédaction de mon mémoire.

Je tiens à remercier mes *amis* ainsi que ma *famille* pour leur compréhension, leur patience et leur soutien.

Enfin je rends grâce à Dieu pour ce travail.

Abréviations

Voici, par ordre alphabétique, les abréviations fréquemment utilisées dans ce document :

AJAX : *Asynchronous JavaScript and XML.*

Explication : technique web permettant une interaction plus fluide lorsqu'une page web est mise à jour à la suite d'une action de l'utilisateur.

AJP : *Apache JServ Protocol.*

Explication : protocole permettant de coupler un serveur d'applications avec un serveur web.

CRUD : *Create, Read, Update, Delete.*

Explication : opérations de base (création, lecture, mise à jour, suppression) pour la persistance des données.

CSS : *Cascading Style Sheets.*

Explication : feuilles de style en cascade dont le rôle est de fournir la mise en forme des éléments HTML d'une page.

DAO : *Data Access Object.*

Explication : patron de conception.

DTO : *Data Transfer Object.*

Explication : patron de conception.

EJB : *Enterprise Java Bean.*

Explication : composant dont le cycle de vie est géré par un conteneur d'EJBs.

HTML : *HyperText Markup Language.*

Explication : langage utilisé pour créer des pages web.

IHM : *Interface Homme Machine.*

Explication : partie du logiciel qui permet à l'utilisateur d'interagir avec le programme informatique.

JDBC : *Java DataBase Connectivity.*

Explication : API Java permettant d'accéder à des bases de données relationnelles.

JNDI : *Java Naming and Directory Interface.*

Explication : API Java permettant d'accéder aux systèmes d'annuaires et de nommage.

JRMP : *Java Remote Method Protocol.*

Explication : protocole natif de l'API RMI.

JSF : *Java Server Faces.*
Explication : framework JEE pour le développement d'applications Web.

JSTL : *Java Standard Tag Library.*
Explication : bibliothèque de balises standard de Java.

JVM : *Java Virtual Machine.*
Explication : environnement d'exécution pour applications Java.

ORM : *Object-Relational Mapping.*
Explication : mapping objet/relationnel.

OSI : *Open Systems Interconnexion.*
Explication : modèle pour l'interconnexion des systèmes ouverts.

POJO : *Plain Old Java Object.*
Explication : simples classes Java respectant la norme JavaBean.

RIA : *Rich Internet Application.*
Explication : client riche fondé sur un navigateur.

RMI : *Remote Method Invocation.*
Explication : API de communication entre objets distants de Java.

RPC : *Remote Procedure Call.*
Explication : mécanisme de base dans les systèmes distribués.

VO : *Value Object.*
Explication : patron de conception également connu sous le nom de DTO.

Terminologie des stéréotypes

Les stéréotypes permettent de définir de nouveaux éléments de modélisation par extension d'éléments existants d'UML. Certains stéréotypes sont prédéfinis dans UML tandis que d'autres peuvent être définis par l'utilisateur. Ils se notent à l'aide d'un nom entouré par des guillemets : <<nom_stéréotype>>.

Le tableau ci-dessous présente les stéréotypes employés par la modélisation UML de l'application WebTask :

Stéréotypes employés par la modélisation UML de l'application WebTask.

Stéréotype	Description
<<entity>>	Objet métier qui encapsule les données du système.
<<control>>	Objet assurant une coordination d'autres objets.
<<boundary>>	Objet à la frontière entre le système et un acteur.
<<lifecycle>>	Objet responsable de trouver les objets <entity>>.
<<delegate>>	Objet réalisant les appels vers un service distant.
<<locator>>	Objet permettant de localiser un service distant.
<<sessionFacade>>	Objet représentant un service distant.

Sommaire

Introduction

Le projet présenté au cours de ce mémoire est une application web open source nommée *WebTask*. Elle est issue de l'unité d'enseignement GLG 204 du CNAM. L'objectif de cette unité est de maîtriser les concepts enseignés dans l'UE GLG 203, également au CNAM, en mettant en œuvre un projet informatique de son choix.

L'idée de concevoir et de développer l'application WebTask provient d'un stage que j'avais effectué, il y a quelques années, au CETIMA (Centre de Traitement de l'Information Médicale des Armées). L'objet de ce stage consistait principalement à réaliser un applicatif web ayant fonction d'annuaire. Cependant, à cette époque, le CETIMA avait envisagé de créer éventuellement dans le futur une application collaborative de gestion de tâches. On me demanda alors de réaliser une courte étude de faisabilité concernant ce projet afin de *sonder* brièvement les besoins. Je ne sais pas si ce logiciel a eu l'occasion de voir le jour. Toutefois l'application WebTask tente de répondre à cette nécessité de gestion de tâches en s'inspirant et en approfondissant le fonctionnel de cette étude de faisabilité sommaire.

Le but de l'application WebTask est également d'essayer de se frayer un chemin au milieu des outils de gestion de tâches disponibles sur le marché. Certains logiciels sont axés sur le travail collaboratif grâce à internet et proposent l'utilisation de *groupes de travail* auxquels des utilisateurs sont rattachés. Les groupes de travail offrent souvent un éventail de fonctionnalités (calendrier, forum, gestionnaire de fichiers etc.) mais minimisent la gestion des tâches par une simple liste (todo-list). D'autres logiciels sont pleinement dédiés à la gestion de tâches. Ils permettent par exemple de réaliser des diagrammes de Gantt, des diagrammes de réseau Pert etc. Cependant ils sont non collaboratifs et généralement de type desktop. L'application WebTask est un intermédiaire entre ces deux types de logiciels. Elle est collaborative et la plupart de ses fonctionnalités sont destinées à offrir à ses utilisateurs un processus de résolution de tâches.

Le mémoire décrit progressivement, à l'aide de huit chapitres, l'élaboration du projet :

- Le premier chapitre présente l'application WebTask de manière générale. Il fournit, entre autre, un aperçu de son fonctionnement global.
- Le deuxième chapitre présente l'expression des besoins qui, comme son nom l'indique, consiste à recueillir les besoins des acteurs d'un système. Des informations complémentaires sont disponibles à l'annexe 1.
- Le troisième chapitre décrit le graphisme des pages web de l'application.
- Le quatrième chapitre aborde l'analyse orientée objet.
- Le cinquième et le sixième chapitre présentent l'architecture et les choix technologiques.
- Le septième chapitre présente la conception orientée objet. Des informations complémentaires sont disponibles à l'annexe 2.
- Enfin, le huitième chapitre apporte des informations concernant la réalisation de l'application.

Chapitre 1. Présentation de l'application

1.1 Objectifs

Le site internet WebTask est un logiciel axé sur le travail collaboratif. Selon le dictionnaire *le Petit Robert* de 1995, une collaboration *est un travail en commun; un travail entre plusieurs personnes qui génère la création d'une œuvre commune.* L'objectif principal de l'application WebTask est de permettre aux différents salariés d'une entreprise de collaborer ensemble pour résoudre des tâches (cf. *A.1.6 Tâche* / cf. *1.2 Déroulement de la résolution d'une tâche*).

L'application WebTask simplifie et améliore la transmission de l'information entre les membres d'une entreprise. La dispersion géographique des salariés n'est pas un obstacle à cette transmission de l'information puisque l'application est accessible à distance via internet. Le partage aisé du savoir qui concerne les tâches en cours de résolution entraîne un gain de temps non négligeable.

1.2 Déroulement de la résolution d'une tâche

Voici un exemple simplifié, en dix étapes successives, de la manière dont l'application WebTask peut être utilisée pour résoudre une tâche :

- Un chef de projet dépose une tâche qui est à réaliser ultérieurement (cf. *A.1.1 Compte*). A cette étape, l'application attribue le statut *créé* à la tâche déposée (cf. *A.1.9 Statut d'une tâche*).
- Le chef de département, qui dirige le chef de projet, assigne cette tâche à un premier exécutant nommé Exécutant_1. A cette étape, l'application donne le statut *attribué* à la tâche puisqu'elle est assignée pour la première fois à un exécutant.
- Le chef de département assigne à nouveau cette tâche à un second exécutant nommé Exécutant_2.
- Exécutant_1 accepte de traiter la tâche.
- Exécutant_2 accepte également de traiter la tâche.
- Le chef de département met à jour le statut de la tâche en *à réaliser*. Le but est d'indiquer aux exécutants qu'ils peuvent désormais effectuer la tâche en tenant compte de son calendrier (cf. *A.1.8 Calendrier d'une tâche*).
- Lorsque la tâche est réalisée, un des exécutants met à jour son statut en *terminé*.
- Le chef de projet n'est pas satisfait de la réalisation de la tâche. Il met à jour son statut en *à revoir*.
- Lorsque la tâche a été revue par les exécutants, l'un d'entre eux met à jour son statut en *terminé*.
- Le chef de projet est satisfait de la réalisation de la tâche. Il met à jour son statut en *validé*.

La figure *1.1* présente sous forme graphique ces dix étapes. Elles correspondent chacune à un rectangle aux bords arrondis.

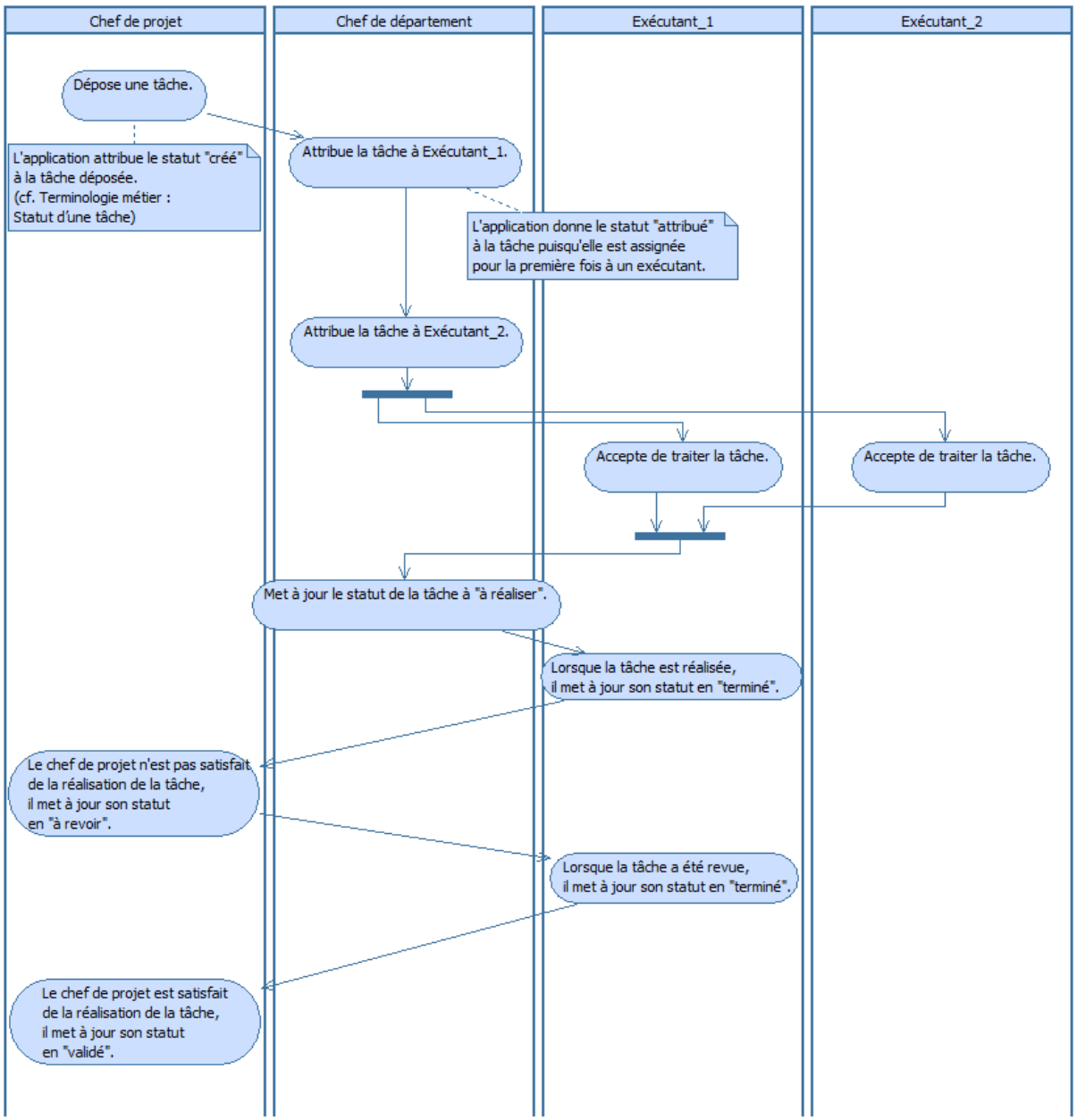

Figure 1.1, *Résolution d'une tâche avec l'application WebTask*.

1.3 Outils de conception

1.3.1 UML

UML signifie *Unified Modeling Language*, ou en français, *langage de modélisation unifié*. C'est un standard de notation graphique normalisé par l'OMG (*Object Management Group*) et destiné à la modélisation orientée objet. Il propose un panel de diagrammes tel que des diagrammes de cas d'utilisation, des diagrammes de classes, des diagrammes de séquences, des diagrammes d'activité etc. Comme son nom l'indique, UML est le résultat d'un processus d'unification de trois méthodes-objets (OMT, Booch et OOSE).

1.3.2 Merise

Merise est une méthode de conception de systèmes d'information qui est apparue en 1978. Auparavant, en 1977, le Ministère de l'Industrie avait lancé une consultation visant à choisir des SSII qui furent chargées de définir cette méthode. Merise se base sur la séparation des données et des traitements et s'appuie sur l'approche par niveau (conceptuel, organisationnel/logique, physique/opérationnel).

1.3.3 Processus *Arrington*

Le processus *Arrington* est un processus de développement logiciel qui a été élaboré, comme son nom l'indique, par C.T. Arrington. D'après Pascal Roques, *un processus de développement logiciel définit une séquence d'étapes, en partie ordonnées, qui concourent à l'obtention d'un système logiciel ou à l'évolution d'un système existant*.

C.T. Arrington décrit son processus, basé sur UML, dans le livre qu'il a écrit : *Enterprise Java with UML.* Il y détaille, entre autre, les différentes phases à suivre :

- Modélisation des besoins.
- Analyse orientée objet.
- Architecture.
- Choix technologiques.
- Conception.

Chapitre 2. Expression des besoins

2.1 Les acteurs de l'application

2.1.1 Diagramme de contexte statique

Le diagramme de classes (figure *2.1*), appelé *diagramme de contexte statique,* comporte une classe centrale et unique représentant le système WebTask. Cette classe est reliée à chaque acteur à l'aide d'une association. Les acteurs sont donc des entités externes qui interagissent avec un système, c'est-à-dire dans le contexte actuel, le système WebTask.

Le diagramme de contexte statique suit les bonnes pratiques en appliquant la représentation graphique du *stick man* pour les acteurs humains et la représentation rectangulaire avec le mot-clé <<*actor*>> pour les autres acteurs.

Les acteurs humains *Exécutant*, *Chef_de_projet* et *Chef_de_département* sont des acteurs primaires. D'après *Pascal Roques*, chacun d'eux est un *acteur pour qui le cas d'utilisation concerné produit un résultat observable.* Les acteurs non humains *Messagerie_entreprise* et *Source_de_données* sont appelés *acteurs secondaires*. Toujours d'après *Pascal Roque*, il s'agit d'un type d'acteur *qui n'est sollicité que par le système lors du cas d'utilisation concerné*.

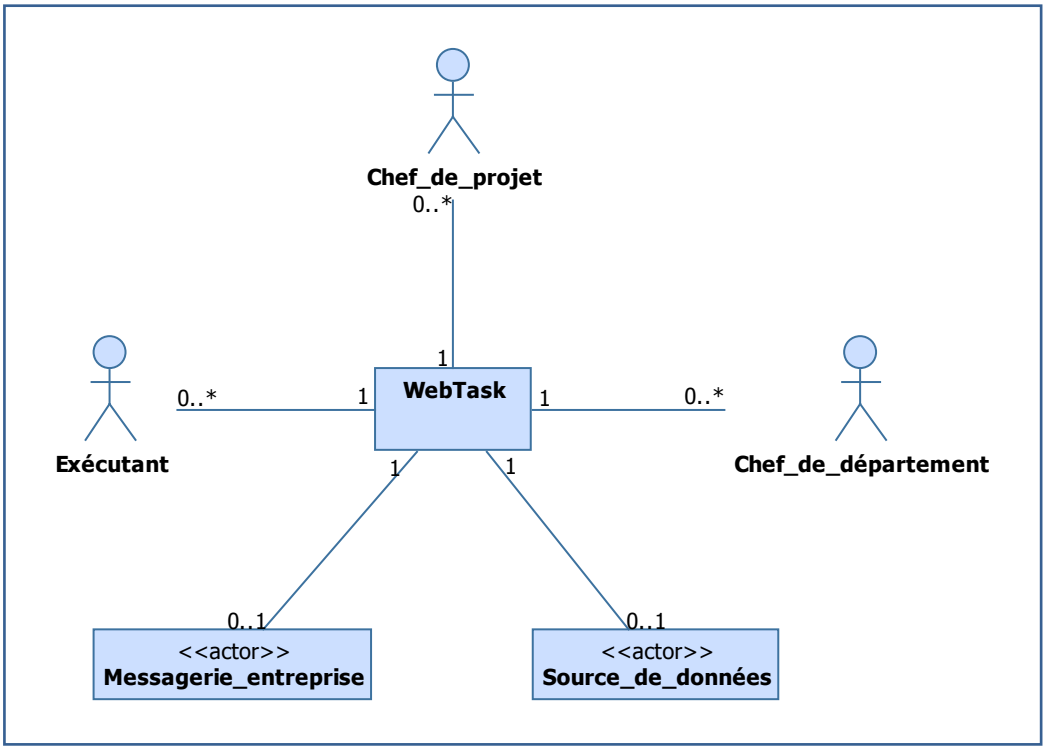

Figure 2.1, *Diagramme de contexte statique*.

2.1.2 Le chef de département

La mission de ce responsable consiste à diriger le département (achats, support, exploitation, production etc.) dont il a la responsabilité au sein d'une entreprise. Il est chargé de la gestion commerciale et budgétaire du département, de son développement et de son management.

Sa fonction requière indéniablement une grande disponibilité et une ouverture vis-à-vis de son personnel. Il en assure d'ailleurs le recrutement, la formation, l'évolution, l'animation et la répartition des tâches. Il trouvera un soutien concernant cette distribution des tâches à l'aide de l'application WebTask. Celle-ci lui apportera un gain de temps dans son quotidien.

2.1.3 Le chef de projet

L'objectif d'un chef de projet est d'assurer le pilotage, le suivi et l'exécution d'un projet. Son métier nécessite entre autre un sens de l'organisation, des aptitudes en matière de communication, le goût des contacts humains ainsi que des capacités à animer une équipe.

Son activité riche et variée sera soutenue par l'utilisation de l'application WebTask. Cette dernière lui permettra, par exemple, de déposer des tâches à exécuter. Lorsque ces tâches déposées auront été distribuées par un chef de département au personnel impliqué dans la réalisation (exécutants), le chef de projet les supervisera et les contrôlera.

2.1.4 L'exécutant

C'est un membre du personnel qui exécute des tâches. Il détient un savoir-faire important mis à la disposition d'un ou plusieurs départements (cf. *A.1.14 Département*). L'application WebTask lui offrira une vision claire des tâches qui le concerne et lui permettra de communiquer avec sa hiérarchie quant à leur réalisation.

2.1.5 La source de données et la messagerie d'entreprise

L'application WebTask nécessite une source de données ainsi qu'une messagerie d'entreprise. Ces acteurs secondaires seront détaillés ultérieurement (cf. *Chapitre 6. Choix technologiques*) car l'expression des besoins ne se préoccupe pas de l'aspect technique et technologique.

2.2 Définition des fonctionnalités du système

L'objectif de ce paragraphe est de structurer, énumérer et décrire les cas d'utilisation de l'application WebTask. Un cas d'utilisation, ou *use case* en anglais, désigne un contrat de comportement entre un système et les acteurs qui interagissent avec lui.

Dans un premier temps, un diagramme de packages des cas d'utilisation est présenté (cf. *2.2.1 Diagramme de packages des cas d'utilisation*). Il organise les cas d'utilisation en les rassemblant par domaine fonctionnel à l'aide d'un mécanisme de regroupement appelé *package.* Les packages du diagramme sont annotés avec le mot-clé *subsystem* (sous-système en français) car ils représentent des parties indépendantes du système. Les flèches de dépendance entre packages de cas d'utilisation synthétisent les relations (inclusions et extensions) entre les cas.

Dans un second temps les sous-systèmes, mentionnés dans le diagramme de packages des cas d'utilisation, sont détaillés. Chaque sous-système est développé en présentant les cas d'utilisation qu'il contient : à l'aide d'un diagramme de cas d'utilisation et à l'aide d'explications textuelles.

Le rôle d'un *diagramme de cas d'utilisation* est de montrer les relations entre les acteurs et les fonctionnalités d'un système. Les cas d'utilisation sont représentés par des ellipses.

Les explications textuelles décrivent l'utilité des différents cas d'utilisation. Le lecteur est invité à se reporter à l'annexe 2 s'il désir obtenir, pour chaque cas d'utilisation, des informations complémentaires qui sont en général les suivantes :

- Pré-conditions :
  Ce sont les conditions nécessaires pour démarrer un cas d'utilisation.
- Scénario nominal, scénarios alternatifs, scénarios d'erreur :
  Un scénario est une succession d'enchaînements qui s'exécute du début à la fin du cas d'utilisation. Trois types de scénarios sont habituellement rencontrés :
  - Le scénario nominal :
    Il décrit l'exécution du cas d'utilisation dans les conditions attendues.
  - Les scénarios alternatifs :
    Ils décrivent les enchaînements alternatifs du cas d'utilisation.
  - Les scénarios d'erreur :
    Ils décrivent les enchaînements qui terminent brutalement le cas d'utilisation en échec.
- Post-conditions :
  Etat du système après la réalisation du cas d'utilisation.
- Diagramme d'activité :
  La forme la plus simple d'un diagramme d'activité est constituée d'un nœud initial (rond noir), d'activités reliées entre elles par des transitions et d'un nœud final (rond noir entouré).

2.2.1 Diagramme de packages des cas d'utilisation

La figure *2.2* présente le diagramme de packages des cas d'utilisation :

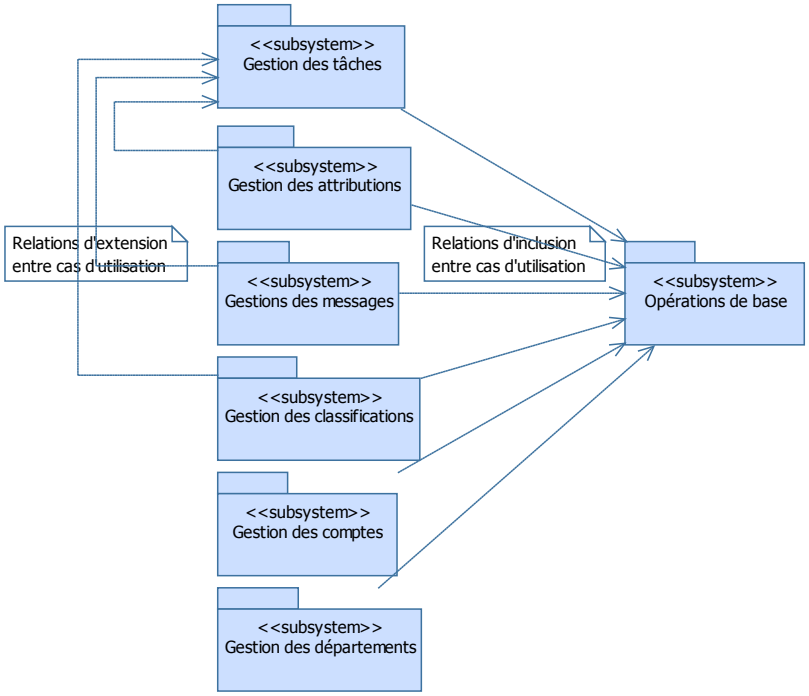

Figure 2.2, *Diagramme de packages des cas d'utilisation*.

2.2.2 Le sous-système *Opérations de base*

2.2.2.1 Diagramme de cas d'utilisation

La figure *2.3* présente le diagramme de cas d'utilisation du sous-système *Opérations de base :*

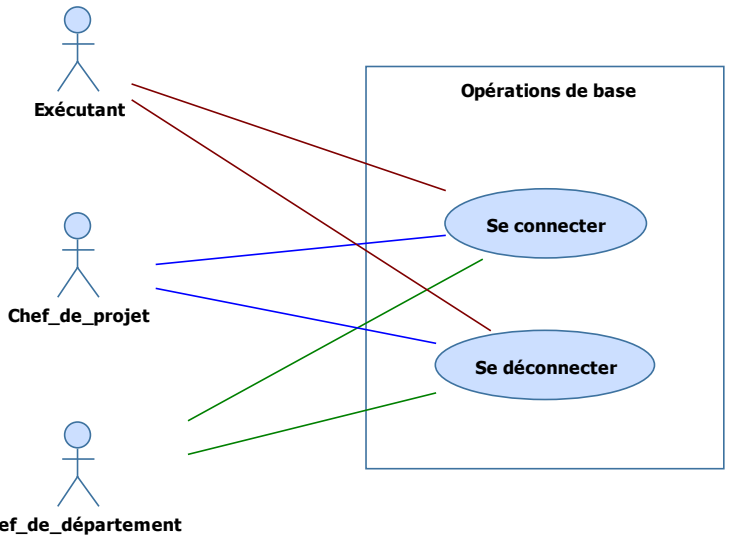

Figure 2.3, *Diagramme de cas d'utilisation du sous-système « Opérations de base ».*

2.2.2.2 Le cas d'utilisation *Se connecter*

(cf. *A.2.1.1 Le cas d'utilisation Se connecter*)

Ce cas d'utilisation permet à un acteur (exécutant, chef de projet, chef de département) de s'authentifier auprès de l'application afin d'accéder à son compte (cf. *A.1.1 Compte*).

2.2.2.3 Le cas d'utilisation *Se déconnecter*

(cf. *A.2.1.2 Le cas d'utilisation Se déconnecter*)

Ce cas d'utilisation permet à un acteur (exécutant, chef de projet, chef de département) de se déconnecter de l'application.

2.2.3 Le sous-système *Gestion des tâches*

2.2.3.1 Diagramme de cas d'utilisation

La figure *2.4* présente le diagramme de cas d'utilisation du sous-système *Gestion des tâches :*

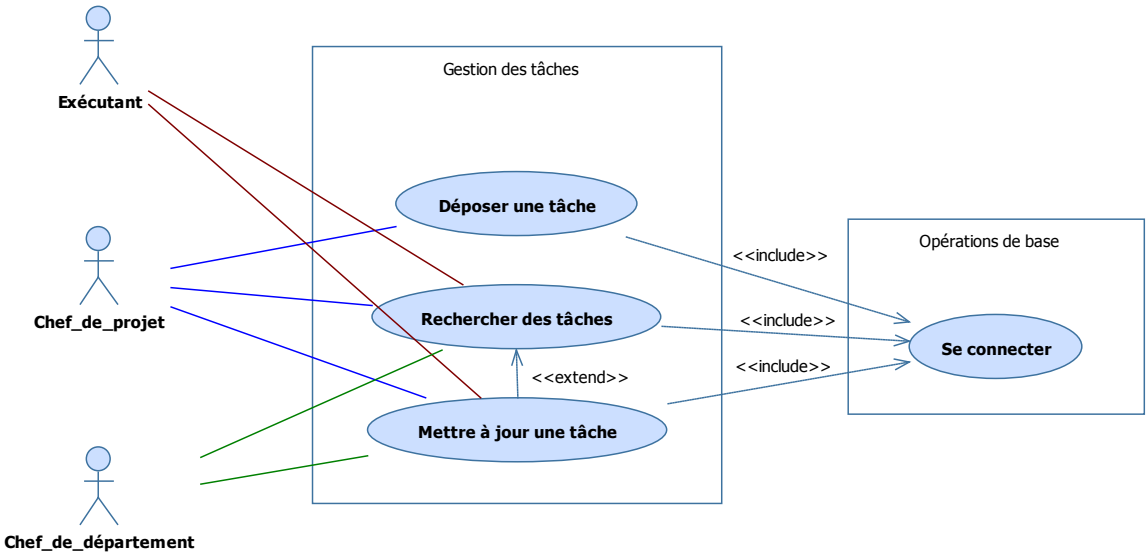

Figure 2.4, *Diagramme de cas d'utilisation du sous-système « Gestion des tâches ».*

2.2.3.2 Le cas d'utilisation *Déposer une tâche*

(cf. *A.2.2.1 Le cas d'utilisation Déposer une tâche*)

Ce cas d'utilisation permet à un chef de projet de déposer une tâche à réaliser ultérieurement (cf. *A.1.6 Tâche*) en renseignant les informations qui la caractérisent.

En effet, lors du dépôt d'une tâche, un chef de projet a l'obligation de renseigner le titre, la description et le niveau d'urgence (cf. *A.1.7 Niveau d'urgence d'une tâche*). Il peut également préciser, s'il le souhaite, une tâche liée (cf. *A.1.6 Tâche*) et un calendrier (cf. *A.1.8 Calendrier d'une tâche*). La date de création, l'émetteur (chef de

projet) et le statut *créé* (cf. *A.1.9 Statut d'une tâche*) sont mentionnés automatiquement par le système.

Le système se charge également d'envoyer un mail au chef de département pour l'alerter du dépôt d'une tâche effectué par un chef de projet qu'il dirige.

2.2.3.3 Le cas d'utilisation *Rechercher des tâches*

(cf. *A.2.2.2 Le cas d'utilisation Rechercher des tâches*)

Ce cas d'utilisation permet à un acteur (exécutant, chef de projet, chef de département) de consulter des tâches.

Les tâches consultables varient selon le type de l'acteur connecté :

- Un exécutant ne peut consulter que les tâches qu'il a acceptées de réaliser (cf. *2.2.4.3 Le cas d'utilisation « Accepter/refuser une tâche »*).
- Un chef de projet ne peut consulter que les tâches qu'il a déposées (cf. *2.2.3.2 Le cas d'utilisation « Déposer une tâche* »).
- Un chef de département ne peut consulter que les tâches déposées par les chefs de projet qu'il dirige.

La recherche des tâches est réalisée avec l'un des critères suivants, au choix :

- Avec leurs identifiants.
- A l'aide d'une partie du titre.
- Par catégorie (cf. A.1.13 *Catégorie d'une tâche* / cf*. 2.2.6.2 Le cas d'utilisation « Ranger une tâche dans une catégorie »*).
- Par statut.

2.2.3.4 Le cas d'utilisation *Mettre à jour une tâche*

(cf. *A.2.2.3 Le cas d'utilisation Mettre à jour une tâche*)

Ce cas d'utilisation permet à un acteur (exécutant, chef de projet, chef de département) de modifier les caractéristiques d'une tâche.

Une tâche qui a le statut *validé*, *rejeté* ou *fermé* ne peut pas être mise à jour.

Les informations modifiables d'une tâche varient selon le type de l'acteur connecté :

- Un chef de département ne peut, d'une part, modifier que le statut d'une tâche et, d'autre part, il ne peut effectuer cette modification qu'en le faisant passer de *attribué* à *à réaliser*. Le statut *à réaliser* permet d'indiquer aux exécutants qu'ils peuvent désormais réaliser la tâche en tenant compte de son calendrier.
- Un chef de projet peut changer le titre, la description, le niveau d'urgence et le calendrier des tâches qu'il a déposées. Il peut également mettre à jour le statut en tant que *fermé* à n'importe quel moment de l'avancement d'une tâche. Ce n'est que lorsque l'exécutant signale qu'il a terminé une tâche par le

statut *terminé* que le chef de projet a la possibilité de mettre à jour le statut en *validé*, *à revoir* ou *rejeté*.

- Un exécutant ne peut, d'une part, modifier que le statut d'une tâche, et d'autre part, il ne peut effectuer cette modification qu'en le faisant passer de *à réaliser* à *terminé* ou de *à revoir* à *terminé*. Cette mise à jour du statut lui permet d'indiquer que la réalisation de sa tâche est achevée.

Lorsqu'une tâche est modifiée par un acteur, l'application envoie un mail aux autres intervenants (chef de département et/ou chef de projet et/ou exécutant(s) suivant le cas) pour les alerter.

2.2.4 Le sous-système *Gestion des attributions*

2.2.4.1 Diagramme de cas d'utilisation

La figure *2.5* présente le diagramme de cas d'utilisation du sous-système *Gestion des attributions :*

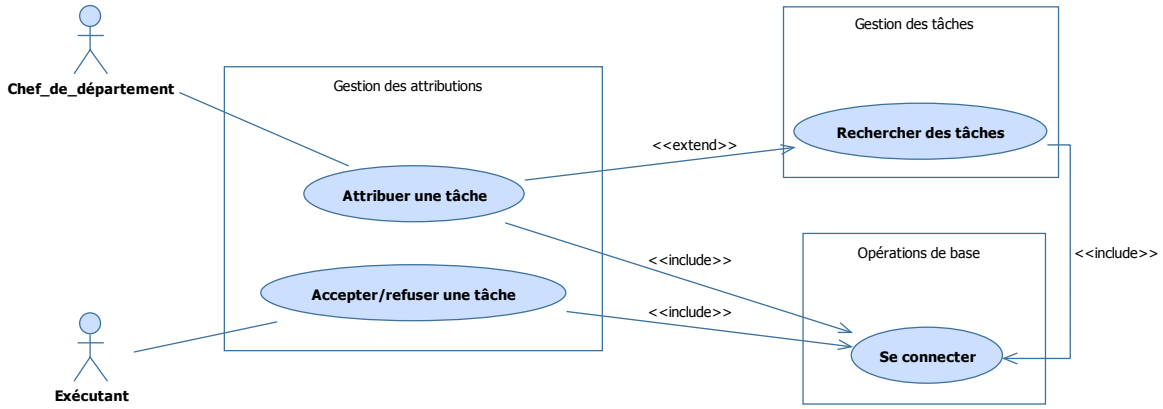

Figure 2.5, *Diagramme de cas d'utilisation du sous-système « Gestion des attributions »*.

2.2.4.2 Le cas d'utilisation *Attribuer une tâche*

(cf. *A.2.3.1 Le cas d'utilisation Attribuer une tâche*)

Ce cas d'utilisation permet à un chef de département d'assigner une tâche à un ou plusieurs exécutants. Il est à noter que l'opération d'attribution (cf. *A.1.12 Attribution d'une tâche*) est à renouveler autant de fois qu'il y a d'exécutants.

Dès qu'une tâche est assignée pour la première fois à un exécutant, l'application attribue à celle-ci le statut *attribué.* Si ensuite d'autres exécutants sont assignés, le statut *attribué* reste inchangé.

L'attribution ne peut être effectuée que sur les tâches qui ont un statut *créé* ou *attribué*.

Une fois que le chef de département aura obtenu une réponse positive des exécutants sollicités pour une tâche (cf. *2.2.4.3 Le cas d'utilisation « Accepter/refuser*

une tâche »), il mettra à jour le statut de cette dernière en tant que *à réaliser (cf. 2.2.3.4 Le cas d'utilisation « Mettre à jour une tâche »)*.

2.2.4.3 Le cas d'utilisation *Accepter/refuser une tâche*

(cf. *A.2.3.2 Le cas d'utilisation Accepter/refuser une tâche*)

Ce cas d'utilisation permet à un exécutant d'accepter ou de refuser une tâche.

Un exécutant ne doit réaliser la tâche qu'il a acceptée qu'à condition que le statut de cette dernière soit mis à jour en tant que *à réaliser*. Cette mise à jour (cf. *2.2.3.4 Le cas d'utilisation « Mettre à jour une tâche »*) est effectuée par le chef de département qui a proposé la tâche.

Lorsqu'un exécutant accepte ou refuse une tâche, le système envoie un mail aux autres acteurs (chef de département qui a assigné la tâche, chef de projet qui a déposé la tâche, exécutants ayant éventuellement déjà accepté de traiter la tâche) pour les alerter.

2.2.5 Le sous-système *Gestion des Messages*

2.2.5.1 Diagramme de cas d'utilisation

La figure *2.6* présente le diagramme de cas d'utilisation du sous-système *Gestion des messages :*

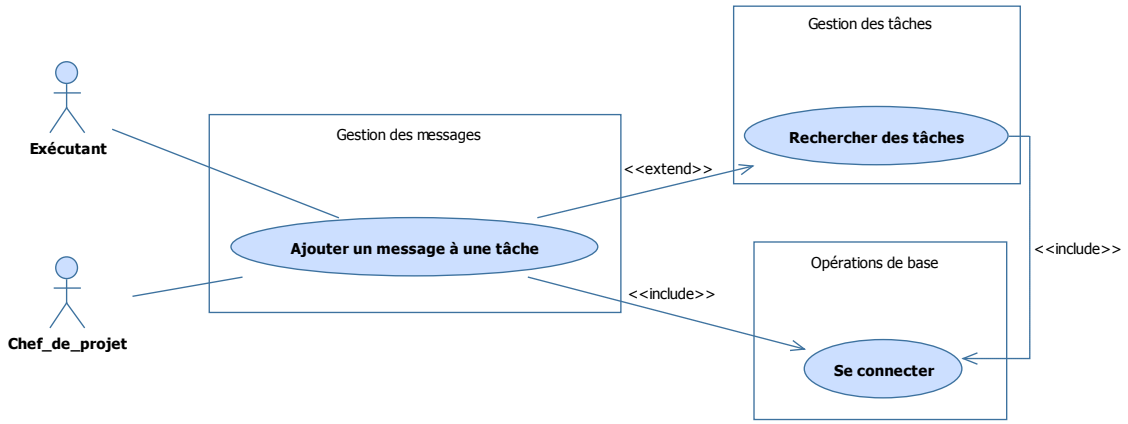

Figure 2.6, *Diagramme de cas d'utilisation du sous-système « Gestion des messages »*.

2.2.5.2 Le cas d'utilisation *Ajouter un message à une tâche*

(cf. *A.2.4.1 Le cas d'utilisation Ajouter un message à une tâche*)

Ce cas d'utilisation permet à un acteur (exécutant, chef de projet) de dialoguer avec les autres intervenants d'une tâche en déposant un message (cf. *A.1.11 Message d'une tâche*).

Cette fonctionnalité n'est pas disponible pour les tâches qui ont un statut *validé*, *rejeté* ou *fermé*.

Lorsqu'un message est ajouté par un acteur, l'application envoie un mail aux autres intervenants (chef de département, chef de projet et/ou exécutant(s) suivant le cas) pour les alerter.

2.2.6 Le sous-système *Gestion des classifications*

2.2.6.1 Diagramme de cas d'utilisation

La figure *2.7* présente le diagramme de cas d'utilisation du sous-système *Gestion des classifications :*

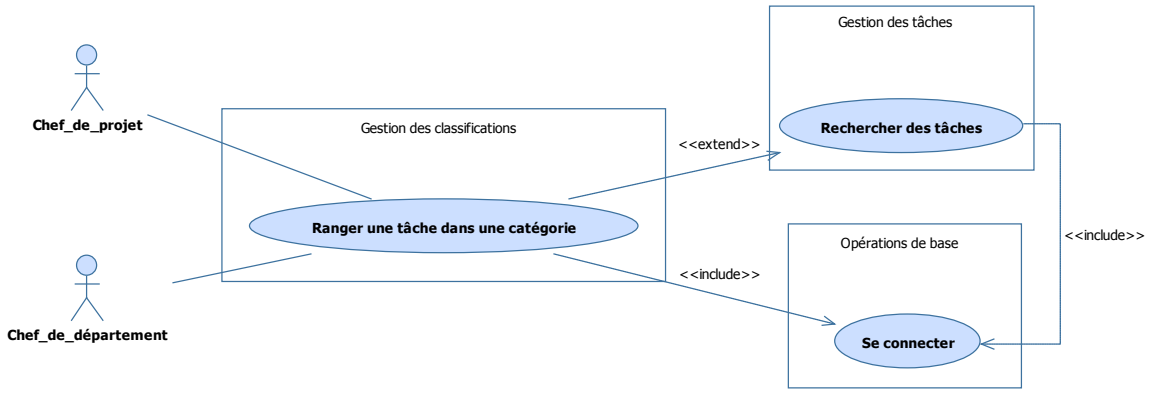

Figure 2.7, *Diagramme de cas d'utilisation du sous-système « Gestion des classifications ».*

2.2.6.2 Le cas d'utilisation *Ranger une tâche dans une catégorie*

(cf. *A.2.5.1 Le cas d'utilisation Ranger une tâche dans une catégorie*)

Ce cas d'utilisation permet à chaque acteur (chef de projet, chef de département) de classer les tâches qui le concernent dans des catégories. Le classement effectué par un acteur n'affecte que son compte.

Cette fonctionnalité aide les chefs de projet et les chefs de département à gérer simultanément un grand nombre de tâches.

2.2.7 Le sous-système *Gestion des comptes*

2.2.7.1 Diagramme de cas d'utilisation

La figure *2.8* présente le diagramme de cas d'utilisation du sous-système *Gestion des comptes :*

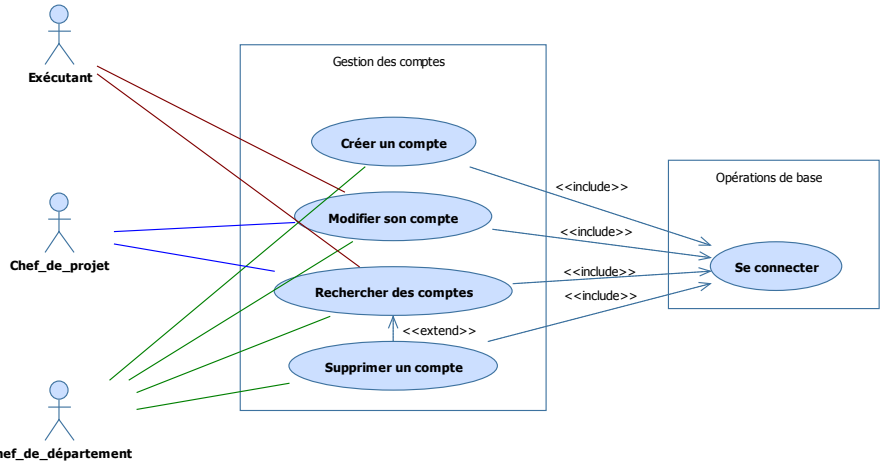

Figure 2.8, *Diagramme de cas d'utilisation du sous-système « Gestion des comptes»*.

2.2.7.2 Le cas d'utilisation *Créer un compte*

(cf. *A.2.6.1 Le cas d'utilisation Créer un compte*)

Ce cas d'utilisation permet à un chef de département de créer des comptes d'exécutant ou de chef de projet.

Lors de la création d'un compte, un chef de département a l'obligation de préciser les informations de sexe, nom, prénom, mail, poste (cf. *A.1.2 Poste d'un compte*), droit d'accès (cf. *A.1.4 Droit d'accès aux fonctionnalités d'un compte*), login et mot de passe.

Le système attribue automatiquement un département si le compte créé est celui d'un chef de projet (un chef de projet est affilié à un département). Le département octroyé correspond à celui du chef de département qui a créé le compte.

Il est important de préciser que l'application ne permet pas de créer des comptes de chef de département. Toutefois elle fournit dix comptes de chef de département utilisables, chacun correspondant à un des dix départements proposés.

2.2.7.3 Le cas d'utilisation *Modifier son compte*

(cf. *A.2.6.2 Le cas d'utilisation Modifier son compte*)

Ce cas d'utilisation permet à un acteur (exécutant, chef de projet, chef de département) de modifier les informations de son compte.

Un acteur peut préciser, par exemple, une absence (cf. *A.1.3 Absence d'un compte*) afin que son chef de département ne lui attribue pas de tâches (cf. *2.2.4.2*

Le cas d'utilisation Attribuer une tâche) lorsqu'il n'est pas présent. Les informations de poste, de département et de droit d'accès sont des données non modifiables.

2.2.7.4 Le cas d'utilisation *Rechercher des comptes*

(cf. *A.2.6.3 Le cas d'utilisation Rechercher des comptes*)

Ce cas d'utilisation permet à un acteur (exécutant, chef de projet, chef de département) de consulter les informations professionnelles des comptes des autres. Seules les informations de login et mot de passe ne sont pas consultables.

La recherche des comptes est réalisée avec l'un des critères suivants, au choix :

- A l'aide d'une partie du nom.
- A l'aide d'une partie du prénom.
- Par poste.
- Avec leurs identifiants.

2.2.7.5 Le cas d'utilisation *Supprimer un compte*

(cf. *A.2.6.4 Le cas d'utilisation Supprimer un compte*)

Ce cas d'utilisation permet à un chef de département de supprimer un compte.

2.2.8 Le sous-système *Gestion des départements*

2.2.8.1 Diagramme de cas d'utilisation

La figure *2.9* présente le diagramme de cas d'utilisation du sous-système *Gestion des départements :*

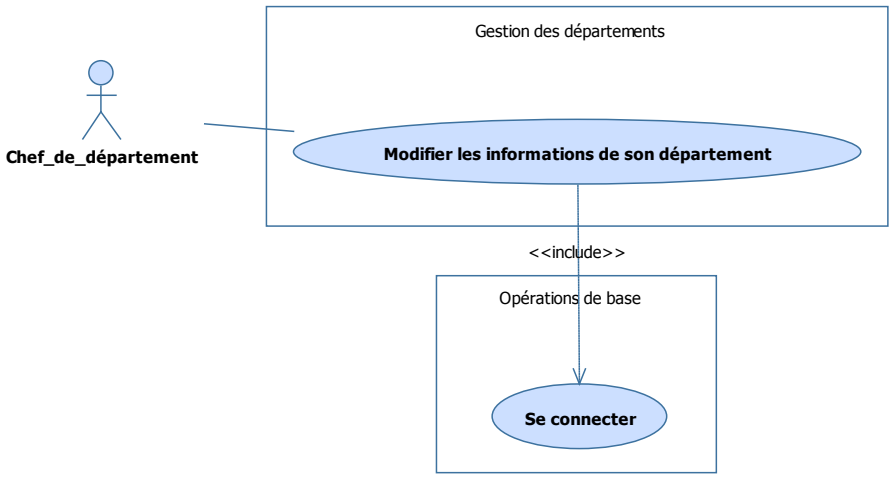

Figure 2.9, *Diagramme de cas d'utilisation du sous-système « Gestion des départements ».*

2.2.8.2 Le cas d'utilisation *Modifier les informations de son département*

(cf. *A.2.7.1 Le cas d'utilisation Modifier les informations de son département*)

Ce cas d'utilisation permet à un chef de département de modifier les informations du département qu'il dirige. Les informations modifiables sont l'activité et l'adresse (cf. *A.1.15 Adresse d'un département*).

Chapitre 3. Conception graphique

3.1 Description graphique du Front Office

3.1.1 Entête

L'entête (figure *3.1*) est affiché sur toutes les pages du site. Il est constitué de trois bandeaux horizontaux :

- Un bandeau blanc comportant le logo WebTask. Un clic sur le logo provoque une redirection vers la page d'accueil.
- Un bandeau avec une image de la planète Terre.
- Un bandeau avec un dégradé bleu comportant le libellé *La gestion de tâches au service de vos projets*.

Figure 3.1, *Entête des pages web de l'application WebTask*.

3.1.2 Pieds de page

Le pied de page (figure *3.2*) est un bandeau avec un dégradé bleu. Il est affiché sur toutes les pages du site.

Figure 3.2, *Pieds de page des pages web de l'application WebTask*.

3.1.3 Bloc de gauche

Le bloc de gauche (figure *3.3*) est affiché sur toutes les pages du site. Il contient deux éléments :

- Un menu qui permet d'accéder aux différentes fonctionnalités de l'application. Lorsque l'utilisateur n'est pas connecté, le menu est désactivé. Il a une couleur noire et ses items ne sont pas accessibles. Lorsque

l'utilisateur est connecté, le menu est activé. Il devient bleu et ses items sont accessibles et cliquables.

- Une zone permettant de se connecter. Lorsque l'utilisateur est déconnecté, elle contient le libellé *Se connecter*, un champ de login, un champ de mot de passe et un bouton *Valider*. Lorsque l'utilisateur est connecté, le libellé *Se connecter* est transformé en libellé *Se déconnecter* et elle contient un bouton *Valider*.

Figure 3.3, *Bloque de gauche avant et après la connexion d'un utilisateur*.

3.1.4 Contenu

La zone de contenu varie d'une page à l'autre. Cependant elle contient toujours, en arrière plan, une image discrète de la planète Terre. La figure *3.4* présente le contenu de la page de recherche des tâches.

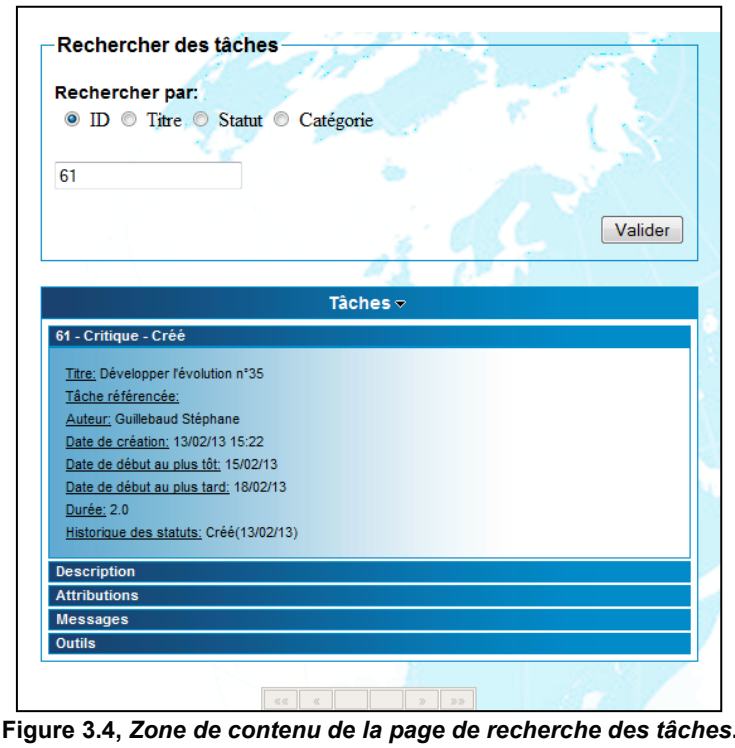

Figure 3.4, *Zone de contenu de la page de recherche des tâches*.

3.1.5 Vue d'ensemble

La figure *3.5* présente la vue d'ensemble de la page de recherche des tâches :

Figure 3.5, ***Vue d'ensemble de la page de recherche des tâches.***

3.2 Ergonomie web de l'application

Dans les années 1950, Alain Wisner donnait la définition générale suivante de l'ergonomie : *L'ensemble des connaissances scientifiques relatives à l'Homme nécessaires pour concevoir des outils, des machines et des dispositifs qui puissent être utilisés avec le maximum de confort, de sécurité et d'efficacité*. Cette description est applicable dans le cadre du web. En effet, l'ergonomie web prend en compte les critères de confort, de sécurité et d'efficacité pour atteindre son objectif. Ce dernier consiste à optimiser les interfaces homme-machine et à améliorer la communication entre l'utilisateur et la machine. L'ergonomie web s'appuie sur des règles de base que tente d'intégrer l'application WebTask.

Voici quelques unes de ces règles intégrées par l'application WebTask :

- Les informations d'un site doivent être rangées de telle manière que les utilisateurs trouvent rapidement ce qui les intéresse.
- Le contenu des pages doit être lisible, organisé et aéré.
- Les différentes pages d'un site internet doivent être cohérentes :
  - Cohérence des localisations des éléments.
  - Cohérence des appellations.
  - Cohérence des formats de présentation.
  - Cohérence des interactions.
  - Etc.
- Un site internet doit appliquer les conventions web.
  Ex : l'application WebTask redirige l'utilisateur sur la page d'accueil lorsque celui-ci clique sur le logo (cf. *3.1.1 Entête*).
- Un site internet doit informer l'internaute lorsque c'est nécessaire.
  Ex : l'application WebTask affiche un message de confirmation à un utilisateur lorsqu'un cas d'utilisation s'est déroulé correctement.
- Un site internet doit prévoir les erreurs des internautes.
  Ex : lorsque les données d'un formulaire ne sont pas correctes, l'application WebTask affiche des messages d'erreur.
- L'internaute doit toujours avoir le contrôle sur le site internet, du moins en apparence. Il ne doit pas se sentir contraint par le système.
- Satisfaire les internautes.

Chapitre 4. Analyse orientée objet

4.1 Objectifs

La phase d'analyse orientée objet a succédé à l'identification et à la documentation des besoins de l'application WebTask, réalisées au cours de la phase d'expression des besoins (cf. *Chapitre 2. Expression des besoins*). La documentation des besoins a requis des explications textuelles ainsi que l'utilisation de diagrammes UML de différents types :

- Diagramme de contexte statique.
- Diagramme de packages.
- Diagramme de cas d'utilisation.
- Diagramme d'activité.

L'analyse orientée objet est donc une phase qui s'appuie sur celle de l'expression des besoins. Son rôle est de découvrir les objets qui forment le système, de déterminer leur utilité et de préciser leurs interactions. Elle décrit ce que le système a besoin de faire en occultant les aspects technologiques. Ceci signifie donc que l'analyse ne se préoccupe pas des problèmes techniques (ex : performance), de l'architecture ou du langage de développement.

Le processus d'analyse orientée objet consiste à appliquer trois étapes sur chaque cas d'utilisation détaillé au cours de la phase d'expression des besoins :

- Découvrir les objets candidats.
- Décrire les interactions entre ces objets.
- Décrire les classes.

Les objets découverts lors de la *première étape du processus d'analyse* sont des objets de stéréotype <<*entity*>>, <<*boundary*>>, <<*control*>> et <<*lifecycle*>> (cf. *Terminologie des stéréotypes*). Ils représentent le point de départ de l'étape suivante.

La *seconde étape du processus d'analyse* décrit les interactions entre les objets précédemment découverts à l'aide de diagrammes de séquences (un diagramme par flux d'événements). Chaque interaction se présente sous la forme d'un message qui est envoyé par un objet et reçu par un autre. La réception d'un message provoque un événement dans l'objet récepteur.

La *troisième étape du processus d'analyse* comporte un diagramme de classes qui montre les objets qui participent au fonctionnement du cas d'utilisation. Chaque message, mentionné dans un des diagrammes de séquences réalisés à l'étape deux, doit correspondre à une méthode d'une classe présente dans le diagramme de classes.

L'analyse orientée objet de l'application WebTask n'a pas inclus la réalisation d'un diagramme de classes par cas d'utilisation, comme le mentionne l'étape trois *du processus d'analyse*. Cependant toutes les classes découvertes pendant l'analyse,

c'est-à-dire les classes de tous les cas d'utilisation, ont été regroupées par stéréotype à l'aide de diagrammes de classes.

Le chapitre quatre ne précise pas les objets (d'analyse) candidats des différents cas d'utilisation. Il ne montre pas non plus les différents diagrammes réalisés, à l'exception du diagramme de classes de stéréotype *<<entity>>* (cf. *4.3 Diagramme de classes de stéréotype <<entity>>)*.

4.2 Conventions de nommage

Au cours de la phase d'analyse orientée objet, des conventions de nommage ont été instaurées afin de nommer les objets découverts :

- Les noms des objets de stéréotype *<<boundary>>* ont été suffixés par *UI.*
- Les noms des objets de stéréotype *<<control>>* ont été suffixés par *Control.*
- Les noms des objets de stéréotype *<<lifecycle>>* ont été suffixés par *Manager.*
- Les noms des objets de stéréotype *<<entity>>* n'ont pas été préfixés ou suffixés par un groupe de lettres.

4.3 Diagramme de classes de stéréotype <<*entity*>>

Les figures *4.1* et *4.2* présentent le diagramme de classes de stéréotype <<entity>> :

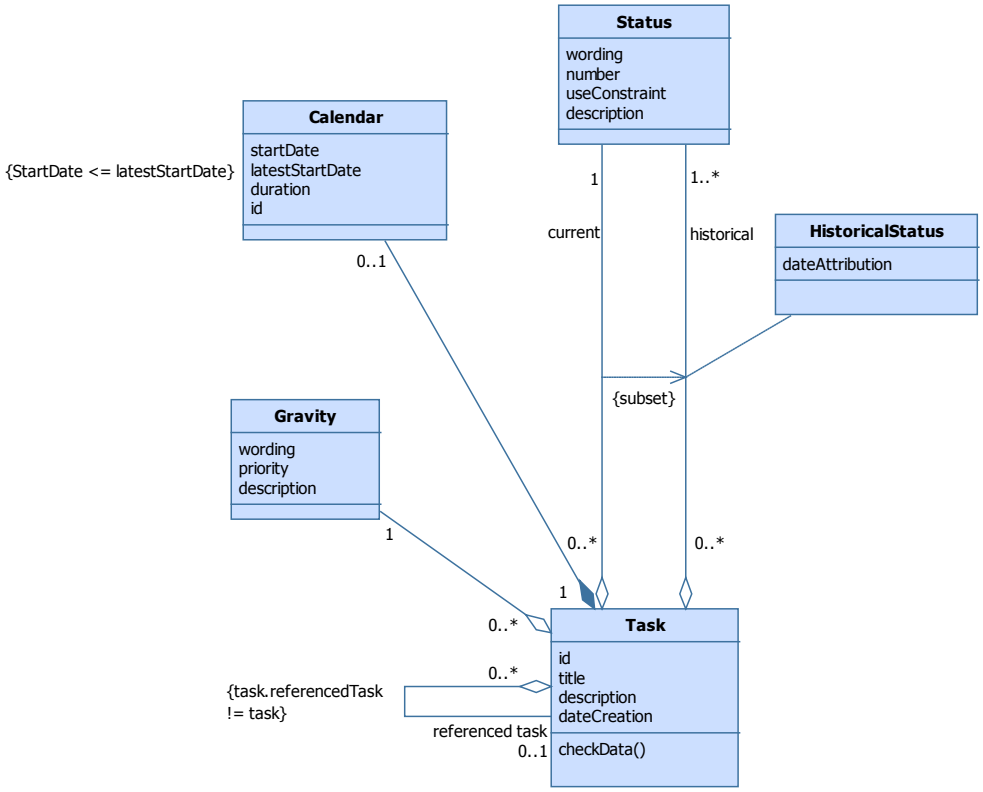

Figure 4.1, *diagramme de classes de stéréotype <<entity>> (partie 1)*.

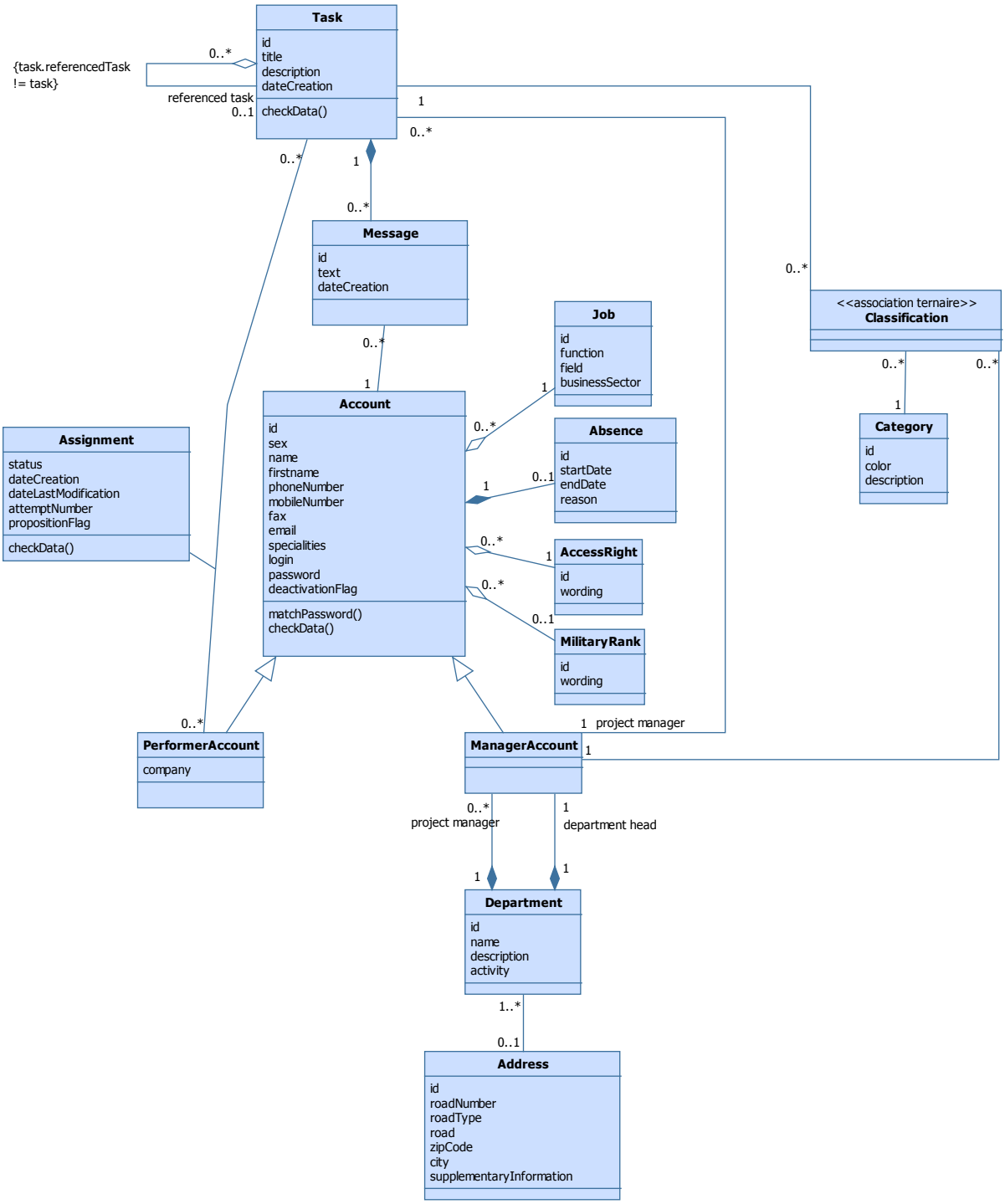

Figure 4.2, *diagramme de classes de stéréotype <<entity>> (partie 2)*.

Chapitre 5. Principes Architecturaux

Le terme *architecture* désigne, d'après le dictionnaire du Larousse, *l'art de concevoir et de construire un bâtiment selon des partis esthétiques et des règles techniques déterminées*.

La construction d'une application informatique et la construction d'un bâtiment présentent des similitudes. Par exemple dans le monde du génie civil, les termes *plan*, *matériaux de construction* et *outils* peuvent correspondre respectivement en architecture logicielle aux termes *diagramme*, *bibliothèques*, *outils de développement*.

Toutes ces similitudes ont entraîné l'usage courant du mot *architecture* dans le milieu informatique.

5.1 Architecture en couches logicielles

L'architecture en couches logicielles consiste à découper de façon "logique" une application en couches, indépendamment des considérations physiques.

Chaque couche constitue l'une des parties d'une application. Elle a ses propres responsabilités et fait appel à la couche immédiatement située en dessous d'elle. La communication entre couches s'effectue en établissant des contrats sous forme d'interfaces représentant, chacune, un service d'accès. L'utilisation des interfaces entraîne une limitation du couplage entre les couches d'une application (couplage lâche).

Réaliser un découpage en couches offre l'avantage de maîtriser la complexité des applications et de pouvoir substituer les implémentations des couches.

La figure *5.1* montre la décomposition logique de l'application WebTask *:*

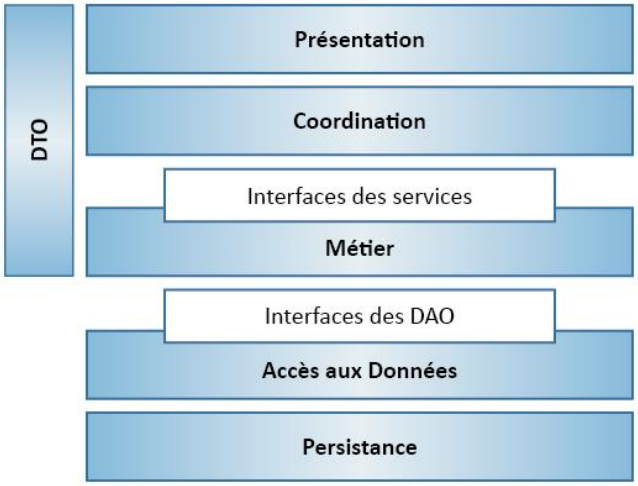

Figure 5.1, *Architecture en couches logicielles de l'application WebTask,* [COG 09].

Comme la figure précédente (figure *5.1*) a permis de l'observer, la structuration de l'application est basée sur une décomposition logique en cinq couches principales:

- La couche *Présentation*.
- La couche *Coordination*.
- La couche *Métier*.
- La couche *Accès aux Données*.
- La couche *Persistance*.

Il est à préciser que la couche *Métier* et la couche *Accès aux Données* ne sont connues que par les interfaces qu'elles exposent.

Les paragraphes qui suivent décrivent la fonction de chacune des couches constituant l'application WebTask.

5.1.1 La couche Présentation

La couche *Présentation* concerne la partie visible d'une application. Elle assure l'affichage de l'interface homme-machine (IHM) et permet l'interaction avec les utilisateurs. La couche *Coordination* est appelée par cette couche.

L'application WebTask propose une interface graphique de type client riche RIA (*Rich Internet Application*). Les clients riches sont un compromis entre les clients lourds et les clients légers. L'acronyme RIA désigne une catégorie de clients riches se fondant sur l'utilisation d'un navigateur et permettant d'avoir des interfaces web sophistiquées. Les technologies employées pour le développement des pages web sont présentées au chapitre *Chapitre 6. Choix technologiques*. La couche *Présentation* est nommée *Vue* dans le modèle de conception MVC (cf. *5.3.2.4 Le pattern Modèle Vue Contrôleur 2*).

Il est intéressant de préciser que la couche *Présentation* n'est pas toujours graphique. Par exemple, la couche *Présentation* d'un service web est la représentation XML des données retournées par celui-ci.

5.1.2 La couche Coordination

La couche *Coordination* est un intermédiaire entre la couche *Présentation* et la couche *Métier*. Elle est considérée comme le cerveau de l'application car son rôle est d'analyser les requêtes utilisateur et de contrôler l'enchaînement des tâches. Elle est notamment chargée de :

- Gérer la cinématique des pages.
- Réaliser l'invocation des appels métiers.
- Traiter les exceptions qui sont levées.
- Gérer l'état des sessions des clients connectés.

La couche *Coordination* représente la partie *Contrôleur* dans le modèle de conception MVC (cf. *5.3.2.4 Le pattern Modèle Vue Contrôleur 2*).

5.1.3 La couche Métier

La couche *Métier* est la couche principale d'un modèle à cinq couches puisqu'elle correspond à la partie métier ou fonctionnelle d'une application. Elle implémente la logique métier des cas d'utilisation et répond à un appel de la couche supérieure en retournant les résultats qu'elle a calculés. L'accès aux différentes données du système est réalisé en sollicitant les services de la couche inférieure, c'est-à-dire la couche *Accès aux Données*.

5.1.4 La couche Accès aux Données

La couche *Accès aux Données* est une couche qui, dans le contexte de l'application WebTask, est responsable de l'interaction avec une base de données relationnelle. Elle manipule les entités métier (ou objets du domaine) en réalisant des mappings objet-relationnel et relationnel-objet. Il est recommandé de mettre en œuvre dans cette couche le pattern de conception DAO (*Data Access Object*). Ce pattern est d'ailleurs employé par l'application WebTask et est présenté au cours du chapitre *5.3.1.5 Le pattern DAO*.

5.1.5 La couche Persistance

La couche *Persistance* fournit un service de stockage des données. Elle nécessite l'utilisation d'une technologie de persistance qui peut se matérialiser, par exemple, sous la forme d'un annuaire LDAP ou, dans le cas de l'application WebTask, d'une base de données relationnelle.

5.1.6 La couche DTO : une couche transverse

La couche DTO (*Data Transfer Object*), mentionnée par la figure *5.1*, est une couche transverse à la plupart des couches de l'application WebTask. En effet, elle contient des objets de transfert de données qui transitent à travers les autres couches. Le sigle DTO fait allusion à un pattern de conception qui est davantage présenté au chapitre *5.3.1.4 Le pattern DTO*.

5.2 Architecture en tiers

5.2.1 Un peu d'histoire : du 1-tiers au n-tiers

L'application WebTask possède cinq couches, toutefois une application informatique nécessite trois couches a minima :

- L'interface avec les utilisateurs.
- Les traitements.
- Les données.

L'architecture des applications a connu de nombreuses transformations depuis les premiers développements. Ces évolutions ont consisté à distribuer ces trois couches de différentes manières, entre plusieurs machines physiques. Leur répartition a fait naître au cours du temps les architectures applicatives suivantes :

- L'architecture à *un niveau* (ou *1-tiers*).
- L'architecture à *deux niveaux* (ou *2-tiers*).
- L'architecture à *trois nouveaux* (ou *3-tiers*).
- Les architecture à *n niveaux* (ou *n-tiers*).

Les paragraphes qui vont suivre reviennent brièvement sur ces différentes architectures.

5.2.1.1 L'architecture à un niveau

Dans une application à *un niveau* ou *1-tiers*, les trois couches applicatives s'exécutent sur le même ordinateur. Il est ici question d'informatique centralisée.

Historiquement, les applications sur site central ont été les premières à proposer un accès multi-utilisateurs. Dans ce type d'architecture, les utilisateurs se connectent aux applications exécutées par le serveur central (le mainframe) à l'aide de terminaux passifs (figure 5.2). Le serveur central prend donc en charge l'intégralité des traitements incluant l'affichage qui est déporté sur des terminaux.

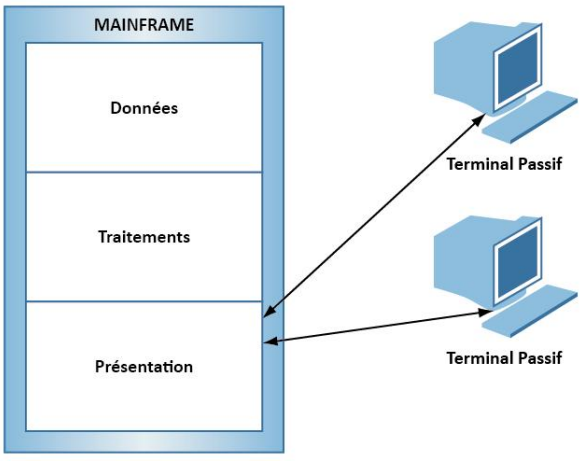

Figure 5.2, *Architecture à un niveau*, inspirée de [BIG 06].

L'architecture *1-tiers* avait notamment l'avantage de faciliter la gestion de la sécurité et d'offrir un faible coût d'administration. Cependant elle avait un certain nombre d'inconvénients comme par exemple une évolution difficile et une maintenance lourde.

5.2.1.2 L'architecture à deux niveaux

L'arrivée d'Internet et des réseaux d'entreprise ont ensuite permis aux applications d'évoluer et d'aboutir à des architectures dites *client/serveur*.

Dans l'approche *client/serveur* (figure *5.3*), un client émet une requête vers un serveur qui exécute un traitement et retourne une réponse.

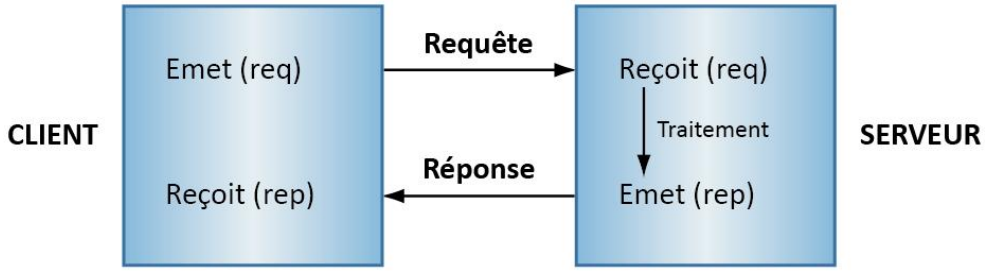

Figure 5.3, *Approche client/serveur*.

La Société Gartner Group a proposé une classification des architectures *client/serveur* (figure *5.4*) qui présente les différentes solutions de répartition des couches logicielles (interface, traitements, données).

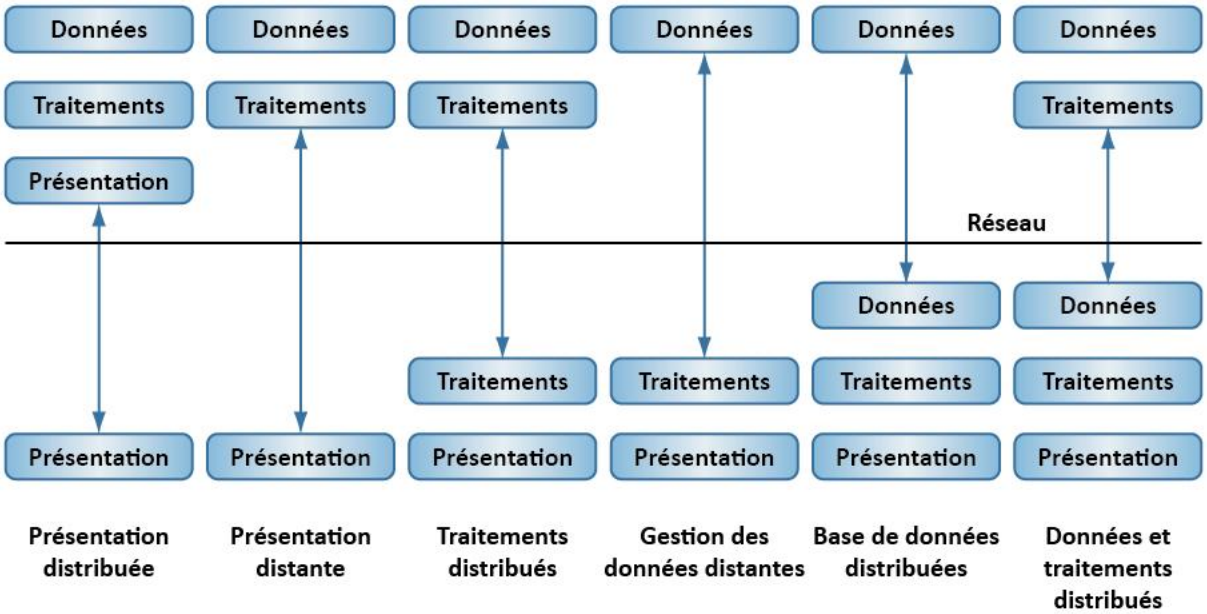

Figure 5.4, *Modèles d'architecture client/serveur*, [BIG 06].

La classification de la Société Garner Group distingue six types d'architecture *client/serveur* :

- Présentation distribuée :
  Il correspond à l'habillage graphique de l'affichage en mode caractères d'applications fonctionnant sur site central.
- Présentation distante :
  La couche *Traitements* et la couche *Données* sont exécutées par le serveur, le client ne prend en charge que l'interface utilisateur.
- Traitement distribué :
  Les traitements sont distribués entre le client et le serveur. Ce type d'application s'appuie sur un mécanisme d'appel de procédure distante ou *Remote Procedure Call* en anglais (RPC). Cette architecture permet d'optimiser la répartition de la charge de traitement entre machines.
- Gestion distante des données :
  Le serveur assure la gestion des données et le client dialogue avec celui-ci au moyen d'un mode d'accès aux données.
- Base de données distribuées :
  Il s'agit d'une variante de la gestion distante des données dans laquelle une partie des données est prise en charge par le client.
- Données et traitements distribués :
  Ce modèle, très puissant, permet de répartir au mieux la charge entre client et serveur.

La limite de l'architecture à *deux niveaux* provient, en grande partie, de l'absence de standardisation du protocole de communication entre le client et le serveur.

5.2.1.3 L'architecture à trois niveaux

L'architecture à *trois niveaux* est une évolution de l'architecture *client/serveur* qui a permis de pallier aux limites de cette dernière. Elle est divisée en trois tiers :

- Le tiers *Client* qui correspond à la machine sur laquelle l'application cliente est exécutée.
- Le tiers *Métier* qui correspond à la machine sur laquelle l'application centrale est exécutée.
- Le tiers *Données* qui correspond à la machine gérant le stockage des données.

La figure *5.5* présente un exemple d'architecture *3-tiers*. Un navigateur est utilisé pour représenter l'application sur la machine cliente, un serveur web pour la gestion de la logique applicative et un serveur de base de données pour le stockage des données. La communication entre le client et le serveur web s'effectue avec le protocole HTTP et la communication entre le serveur web et la base de données est réalisée avec l'API JDBC (dans le cas d'une application Java).

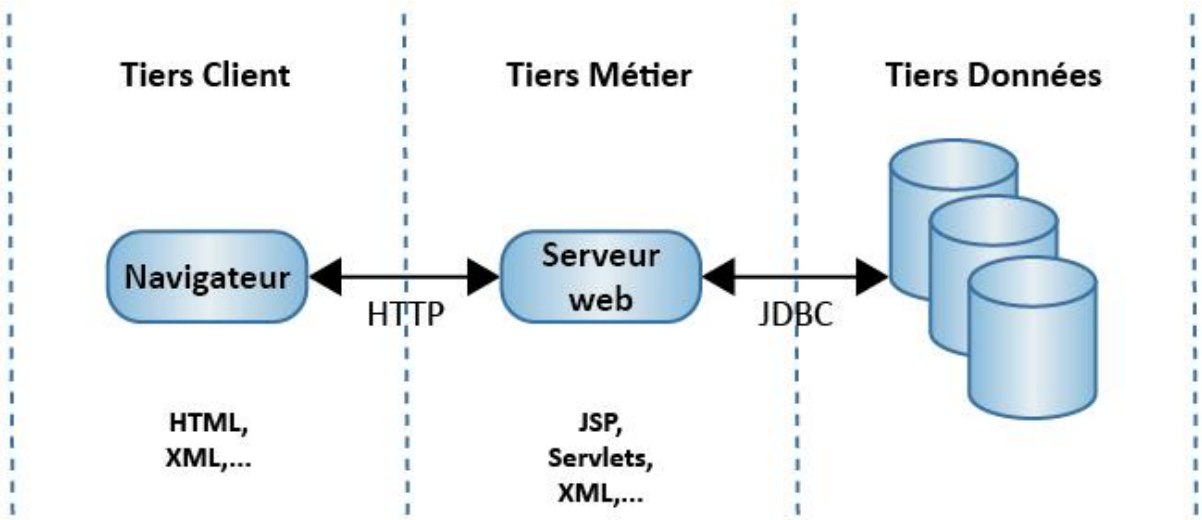

Figure 5.5, *Architecture à trois niveaux*, [CHU 06].

La séparation entre le client, l'application et le stockage, est le principal avantage de l'architecture à trois niveaux. Son inconvénient majeur est le manque de séparation au sein même de l'application.

5.2.1.4 L'architecture à n niveaux

L'architecture à *n niveaux* supprime tous les inconvénients des architectures précédentes. Elle met en œuvre une approche objet en disposant d'un tiers qui regroupe la logique métier de l'entreprise.

L'intérêt principal de l'architecture *n-tiers* est donc la réutilisation des développements. Toutefois il existe d'autres avantages :

- Modularité.
- Facilité d'évolution.
- Gain de temps.

Son inconvénient majeur est la complexité de mise en œuvre.

L'architecture à *n niveau* est le type d'architecture sur lequel se base l'application WebTask. Ceci est détaillé au cours du paragraphe suivant.

5.2.2 L'application WebTask : une architecture n-tiers

Le site internet WebTask s'appuie sur une architecture à *quatre niveaux* :

- Le tiers *Client* qui correspond à la machine sur laquelle un client exécute un navigateur.
- Le tiers *Web* qui correspond à la machine sur laquelle l'application web est exécutée.
- Le tiers *Logique Business* qui correspond à la machine sur laquelle s'exécutent les services distants regroupant la logique métier.
- Le tiers *Données* qui correspond à la machine gérant le stockage des données.

Chaque tiers héberge une, ou plusieurs, des cinq couches de l'application, présentées précédemment (figure *5.6*). La couche *Présentation* est répartie sur le tiers *Client* et le tiers *Web*. Ce dernier contient également la couche *Coordination*. La couche *Métier* et la couche *Accès aux Données* sont déposées sur le tiers *Logique Business*. Le niveau *Données* possède la couche *Persistance*. Quant à la couche transverse *DTO*, elle est distribuée sur les tiers *Client*, *Web* et *Logique Business*.

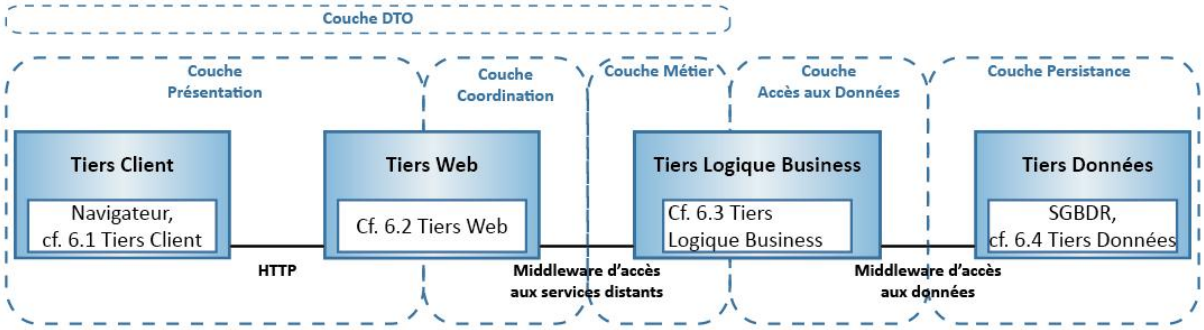

Figure 5.6, *L'architecture n-tiers de l'application WebTask.*

Un changement de niveau est identifié dès qu'une couche lui appartenant nécessite un intermédiaire de communication pour en invoquer une autre. Ce sujet est traité en détail dans le point abordé ci-après.

5.2.3 Interconnexion des tiers

La mise en œuvre de l'interconnexion des tiers, sur lesquels sont réparties les couches de l'application WebTask, se fonde sur la notion d'intergiciel (ou middleware en anglais).

Un intergiciel est un intermédiaire de communication organisant un dialogue entre applications de type *client/serveur*, indépendamment de l'hétérogénéité des machines, systèmes d'exploitation et langages de programmation. Il est le lien entre le software (les applications) et le hardware (l'environnement réseau), et masque la répartition des données et traitements. Le terme *middleware* désigne les outils de développement offrant aux applications une interface d'accès aux services du réseau (API, *Application Programming Interface*).

La figure *5.7* présente la position du middleware au sein d'une architecture répartie:

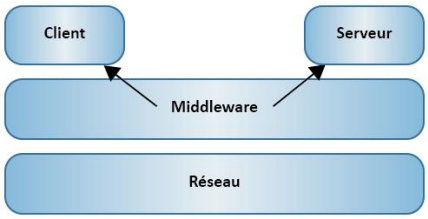

Figure 5.7, *Position du middleware*, inspirée de [SER 06].

Dans le modèle de référence *Interconnexion de Systèmes Ouverts* (OSI , *Open Systems Interconnexion*), le middleware correspond aux couches 5 (*Session*), 6 (*Présentation*) et 7 (*Application*).

L'application WebTask requiert différents types d'intergiciels. L'interconnexion des tiers *Web* et *Logique Business* est effectuée à l'aide d'un middleware d'accès aux services distants, et les tiers *Logique Business* et *Données* sont liés avec un middleware d'accès aux données. Le niveau *Client* et le niveau *Web* communiquent par l'intermédiaire du protocole standard HTTP.

5.2.3.1 Middleware d'accès aux services distants

L'interconnexion des tiers *Web* et *Logique Business* de l'application WebTask est réalisée à l'aide du middleware RMI (*Remote Method Invocation*). C'est une API de communication entre objets distants de Java, s'appuyant sur un mécanisme de type RPC. L'invocation de méthodes distantes repose sur le protocole JRMP (*Java Remote Method Protocol*).

La structure de l'API RMI se base sur trois couches (figure *5.8*). La première comprend les stubs et skeletons. Le stub, côté client, implémente un pattern de mandataire (cf. *5.3.2.3 Le pattern Proxy*). Il sérialise les paramètres transmis à la méthode distante, et désérialise la valeur de retour de cette même méthode distante. Le skeleton, côté serveur, désérialise les paramètres reçus pour les donner à la méthode appelée, et sérialise la valeur de retour. La seconde couche, RRL (*Remote Reference Layer*), permet d'obtenir une référence à l'objet distant à partir de la référence locale (le stub). La troisième couche, c'est-à-dire la couche de transport, est basée sur TCP/IP.

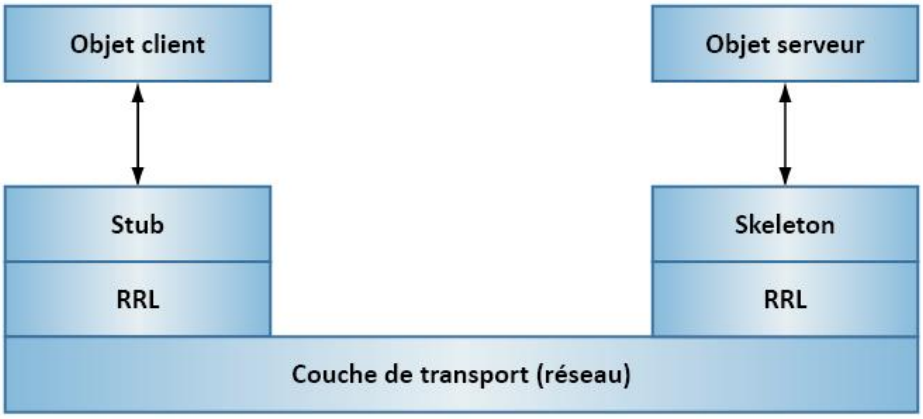

Figure 5.8, *Structure de l'API RMI*, [SIM 08].

Le développement de l'application WebTask ne fait pas appel directement à l'API RMI. L'utilisation du middleware est invisible car elle est incorporée au sein de la technologie implémentant les services distants (cf. *6.3.1 Services distants*).

5.2.3.2 Middleware d'accès aux données

La liaison entre les niveaux *Logique Business* et *Données* de l'application WebTask est accomplie avec l'API JDBC (*Java DataBase Connectivity*). Cette interface de programmation permet d'interagir avec des SGBDR (*Système de Gestion de Base de Données Relationnelle*) en leur envoyant des requêtes écrites en langage SQL.

L'accès à une base de données (figure *5.9*), par l'intermédiaire de JDBC, repose sur plusieurs étapes :

- Chargement du pilote – *(1)*.
- Obtention d'une connexion – *(2)*.
- Interaction (création et exécution d'une requête) – *(3)*.
- Traitement du résultat – *(4)*.
- Fermeture des objets – *(5)*.

```
import java.sql.*;
public class Exemple
{
   public static void main (String args []) throws SQLException
   {
// (1)
      // Chargement du pilote Oracle
      DriverManager.registerDriver (new oracle.jdbc.OracleDri-
      ver());
// (2)
      // Connexion avec la base
      String url = "jdbc:oracle:thin:@myhost:myport:mysid";
      String user = "scott";
      String passwd = "tigger";
      Connection cnx = DriverManager.getConnection (url, user,
      passwd);
// (3)
      // Interaction avec la BD
      Statement stmt = cnx.createStatement();
      ResultSet rset = stmt.executeQuery("SELECT prenom,
      salaire FROM employes");
// (4)
      // Traitement du résultat
      while (rset.next())
        System.out.println (rset.getString(1)+" - "+rset.getDou-
        ble(2));
// (5)
      // Fermetures des objets
      rset.close();
      stmt.close();
      cnx.close();
   }
}
```

Figure 5.9, *Programme JDBC*, [RAV 09].

Quatre types de pilotes peuvent être employés pour communiquer avec une base de données. Le middleware JDBC s'appuie, dans le cadre de l'application WebTask,

sur un pilote de type 4. Cette catégorie de driver offre généralement les meilleurs performances. Elle est notamment caractérisée par un mécanisme d'accès client codé entièrement en Java.

La manipulation des fonctionnalités de l'API JDBC est masquée car l'application WebTask utilise une couche d'abstraction de plus haut niveau pour l'accès au SGBD (cf. *6.3.2 Persistance des données*).

5.3 Imbrication des classes

L'imbrication des classes de l'application WebTask se fonde sur l'utilisation des design patterns, également nommés en français *modèles de conception* ou *patrons de conception*. Ils proposent des solutions éprouvées aux problèmes connus d'architecture et de conception des logiciels. Les modèles de conception sont des bonnes pratiques à appliquer dans le cadre de la programmation orientée objet.

Les design patterns sont classés selon trois catégories :

- Les patterns de construction :
  Ils abstraient les mécanismes d'instanciation des objets.
- Les patterns de structuration :
  Ils facilitent l'organisation de la hiérarchie des classes.
- Les patterns de comportement :
  Ils offrent des solutions pour organiser les interactions et pour distribuer les traitements entre les objets.

Cette section expose dans un premier temps les patterns volontairement choisis pour le développement de l'application WebTask. Dans un second temps, sont présentés les patterns associés aux choix technologiques (cf. *Chapitre 6. Choix technologiques*). Dans une troisième étape, il est proposé un exemple illustrant l'une des façons dont les patterns de l'application collaborent entre eux.

5.3.1 Patterns choisis pour le développement

5.3.1.1 Le pattern Singleton

Le pattern *Singleton* assure qu'une classe ne possède qu'une seule et unique instance pendant la totalité de la durée d'exécution d'une application. Ce modèle de conception fournit une méthode, souvent nommée *getInstance*, permettant d'accéder à l'instance unique.

5.3.1.2 Le pattern Business Delegate

Le pattern *Business Delegate* délègue à des objets *Delegate* les appels aux services distants. Ces objets correspondent à la partie *Modèle* du patron de conception MVC (cf. *5.3.2.4 Le pattern Modèle Vue Contrôleur 2*). Ils peuvent éventuellement traiter la mise en cache des résultats, convertir les exceptions réseau en exceptions de niveau application, ou même encore exécuter d'autres tentatives d'appel dans le cas d'une défaillance d'un service.

5.3.1.3 Le pattern Service Locator

Le rôle du pattern *Service Locator* est de fournir un localisateur de service distant qui encapsule le mécanisme de localisation. Généralement, ce mécanisme consiste notamment à réaliser une opération de recherche (lookup) dans un annuaire de nommage JNDI (*Java Naming and Directory Interface*). Le patron de conception *Business Delegate* emploie ce pattern pour accéder aux services distants.

5.3.1.4 Le pattern DTO

Un objet DTO, ou objet de transfert de données, possède une structure sans comportement, c'est-à-dire une structure uniquement constituée d'attributs, d'accesseurs et de mutateurs. Son rôle est de transporter des données en transitant entre les tiers d'une application distribuée. Un objet DTO encapsule les informations de l'objet métier qu'il représente. Ce pattern, également nommé *Value Object* (VO), permet d'améliorer les performances en limitant l'encombrement du réseau.

5.3.1.5 Le pattern DAO

Les DAO encapsulent le code d'accès aux données et masquent complètement la technologie sous-jacente. Ce sont des classes utilitaires qui, en proposant des fonctionnalités de création, de récupération, de mise à jour et de suppression, permettent de gérer la persistance des objets métier. Par ailleurs les DAO séparent l'accès aux données du reste du code.

5.3.2 Patterns associés aux choix technologiques

5.3.2.1 Le pattern Session Facade

Le pattern *Session Facade* est mis en œuvre à l'aide d'un session bean (cf. *6.3.1 Services distants*). Il se base sur le patron de conception *Facade* qui dissimule un ensemble d'objets en fournissant à l'extérieur une interface unifiée.

5.3.2.2 Le pattern Inversion of Control (IoC)

Ce design pattern permet de faire en sorte que le flot d'exécution d'un logiciel ne soit plus sous le contrôle direct de l'application elle-même mais sous celui du framework ou de la couche logicielle sous-jacente. L'une des utilisations de l'inversion de contrôle est l'injection de dépendances.

5.3.2.3 Le pattern Proxy

Le principe du pattern Proxy est de concevoir un mandataire qui encapsule un objet (le sujet réel) dans le but d'être utilisé à sa place et d'en contrôler l'accès. Ce pattern comporte de nombreuses variantes dont le pattern *Client Proxy* qui invoque des méthodes d'objets côté serveur.

5.3.2.4 Le pattern Modèle Vue Contrôleur 2

L'objectif principal du pattern MVC 2 est de diviser une application en trois parties distinctes:

- Le modèle :
  Il représente les données manipulées par une application.
- La vue :
  Elle permet la présentation des données du modèle.
- Le contrôleur :
  Cette partie est constituée d'un point d'entrée unique à toute l'application et de plusieurs entités de traitement. Lorsqu'une requête est traitée, le point d'entrée unique dirige les traitements vers l'entité appropriée.

L'ancêtre du pattern MVC 2 est le modèle de conception MVC 1.

5.3.3 Coopération de patterns pour appels aux services distants

Les patterns, présentés précédemment, collaborent ensemble dans l'objectif de structurer et de faciliter les appels aux services distants réalisés par l'application. Cette coopération est composée de six étapes (figure *5.10*) :

- Une requête émise par une page web est prise en charge par un contrôleur – *(1)*.
- Le contrôleur délègue à un objet Delegate l'appel à un service distant – *(2)*.
  - L'objet Delegate récupère un localisateur de service distant qui est instancié s'il est demandé pour la première fois – *(3)*.
- L'objet Delegate obtient, par l'intermédiaire du localisateur de service distant, un proxy client – *(4)*.

- Le proxy client permet ensuite à l'objet Delegate d'invoquer une méthode de l'objet côté serveur, implémentant le service distant à l'aide du pattern Session Facade – *(5)*.
- Le service distant effectue un traitement comportant généralement une ou plusieurs opérations dans la base de données. Ces opérations sont effectuées à l'aide d'objets DAO. Le résultat du service distant se matérialise sous la forme d'objets DTO – *(6)*.

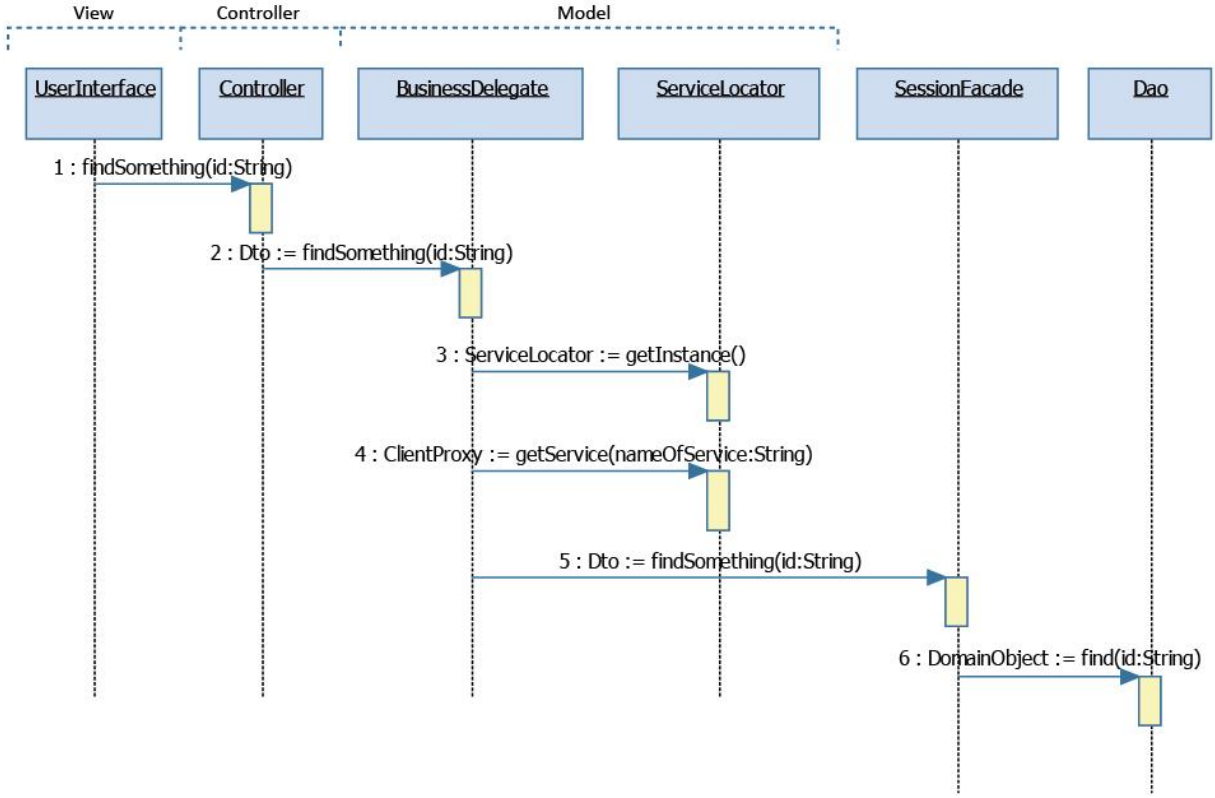

Figure 5.10, *Coopération de patterns pour appel à un service distant*.

5.4 Packages et dépendances

5.4.1 Packages de l'application WebTask

La notion de package correspond à un mécanisme de regroupement d'éléments qui permet, par exemple, de rassembler des classes, des interfaces ou des fichiers properties.

Les packages sont également des espaces de noms. Au sein d'un package, il n'est pas permis que deux éléments portent le même nom. En revanche, deux éléments associés à des packages différents peuvent avoir un nom identique.

Les packages de l'application, au nombre de neuf, ont systématiquement des noms commençant par le préfixe *fr.webtask* et sont souvent subdivisés en sous-packages. Une approche globale claire est fournie en regroupant, dans les packages, les classes suivant leur type d'activité (cf. *Terminologie des stéréotypes*).

5.4.1.1 Le package fr.webtask.web

Ce package est subdivisé en sous-packages contenant les classes nécessaires au tiers web :

Tableau 5.1, *Sous-packages du package fr.webtask.web*.

Sous-package	Fonction
fr.webtask.web.converter	Convertisseurs JSF.
fr.webtask.web.validator	Validateurs JSF.
fr.webtask.web.listener	Listeners JSF.
fr.webtask.web.component	Composants graphiques JSF personnalisés.
fr.webtask.web.control	Classe abstraite des contrôleurs JSF.
fr.webtask.web.control.impl	Contrôleurs JSF.
fr.webtask.web.filter	Filtres JSP.
fr.webtask.web.util	Classes utilitaires.
fr.webtask.web.storetext.configuration	Fichiers properties de configuration.
fr.webtask.web.storetext.messages.pages	Fichiers properties contenant les libellés et les messages des pages web (un fichier properties existe pour chaque page).
fr.webtask.web.storetext.messages.templates	Fichiers properties contenant les libellés associés aux templates (un fichier properties existe pour chaque template).
fr.webtask.web.storetext.util	Classes mappant les fichiers properties : elles permettent au code de l'application d'accéder aux valeurs des clés properties.

5.4.1.2 Le package fr.webtask.entity

Ce package est subdivisé en sous-packages contenant les classes de stéréotype <<*entity*>> :

Tableau 5.2, *Sous-packages du package fr.webtask.entity*.

Sous-package	Fonction
fr.webtask.entity.dto	Interface des objets de transfert de données.
fr.webtask.entity.dto.impl	Classes des objets de transfert de données.
fr.webtask.entity.domain	Interface des Entity beans.
fr.webtask.entity.domain.impl	Classes des Entity beans.

5.4.1.3 Le package fr.webtask.delegate

Ce package contient l'interface commune à toutes les classes de stéréotype <<*delegate*>>. Il est subdivisé en sous-packages :

Tableau 5.3, *Sous-packages du package fr.webtask.delegate*.

Sous-package	Fonction
fr.webtask.delegate.impl	Classes des objets Delegate.
fr.webtask.delegate.exception.impl	Classes des exceptions techniques spécifiques aux objets Delegate.

5.4.1.4 Le package fr.webtask.locator

Ce package contient l'interface commune à toutes les classes de stéréotype <<*locator*>>. Il possède également un sous-package :

Tableau 5.4, *Sous-package du package fr.webtask.locator.*

Sous-package	Fonction
fr.webtask.locator.impl	Classes des objets Locator.

5.4.1.5 Le package fr.webtask.service

Ce package est subdivisé en sous-packages contenant les classes correspondant aux services distants :

Tableau 5.5, *Sous-packages du package fr.webtask.service.*

Sous-package	Fonction
fr.webtask.service.sessionbean	Interface commune à toutes les classes de stéréotype <<*facade*>> + interfaces propres à chaque service.
fr.webtask.service.sessionbean.impl	Classes de stéréotype <<*facade*>>.
fr.webtask.service.storetext.configuration	Fichiers properties de configuration pour les services.
fr.webtask.service.storetext.util	Classes mappant les fichiers properties de configuration.
fr.webtask.service.util	Classes utilitaires.

5.4.1.6 Le package fr.webtask.dao

Ce package contient la classe abstraite commune à toutes les classes de stéréotype <<*life cycle*>> ainsi que les interfaces propres à chaque objet DAO. Il possède également un sous-package :

Tableau 5.6, *Sous-package du package fr.webtask.dao.*

Sous-package	Fonction
fr.webtask.dao.impl	Classes des objets DAO.

5.4.1.7 Le package fr.webtask.exception

Ce package contient les classes abstraites et l'interface utilisées par les classes d'exceptions techniques et fonctionnelles communes à l'application web et aux services. Il possède également un sous-package :

Tableau 5.7, *Sous-package du package f fr.webtask.exception.*

Sous-package	Fonction
fr.webtask.exception.impl	Classes des exceptions techniques et fonctionnelles.

5.4.1.8 Le package fr.webtask.enumeration

Ce package contient des énumérations java qui définissent, chacune, un ensemble fini de constantes.

5.4.1.9 Le package fr.webtask.validator

Ce package contient des classes permettant de vérifier le format des données. Il n'est pas à confondre avec le package *fr.webtask.web.validator* qui contient des validateurs JSF.

5.4.1.10 Le package fr.webtask.mail

Ce package contient la classe abstraite commune à toutes les classes des mails. Il possède également un sous-package :

Tableau 5.8, *Sous-package du package fr.webtask.mail*.

Sous-package	Fonction
fr.webtask.mail.impl	Classes des mails.

5.4.2 Packages externes requis par l'application WebTask

Le tableau *5.9* présente les dépendances directes identifiées :

Tableau 5.9, *Packages externes requis par l'application WebTask.*

Bibliothèque	Archive
EJB 3	jbossall-client.jar
JSF Sun RI 1.2	jsf-api.jar jsf-impl.jar
Richfaces 3.3	richfaces-api-3.3.3.Final.jar richfaces-impl-3.3.3.Final.jar richfaces-ui-3.3.3.Final.jar
JSTL	jstl-api-1.2.jar jstl-impl-1.2.jar
Apache Commons	commons-beanutils-1.8.3.jar commons-digester-1.8.1.jar commons-lang3-3.0.jar commons-logging-1.1.1.jar
Apache Tiles	tiles-core-2.1.4.jar tiles-api-2.1.4.jar tiles-servlet-2.1.4.jar tiles-jsp-2.1.4.jar
JSP 2.1 API	jsp-api.jar
Apache log4j	log4j.jar (provenant de JBOSS)
Driver JDBC de type 4 pour MySQL	mysql-connector-java-5.1.5-bin.jar
JavaMail	mail.jar
FreeMarker	freemarker.jar

5.4.3 Interfaces et classes abstraites

Certains sous-packages (ex : fr.webtask.entity.domain.impl) contiennent des classes qui implémentent ou étendent respectivement une interface ou une classe abstraite commune. Cette démarche permet de renforcer la structure de l'application, de factoriser du code et d'accentuer l'appartenance des classes à une utilité précise. Les sous-packages concernés sont ceux dont le nom se termine par *.impl*.

5.4.4 Diagramme de packages

Le diagramme (figure *5.11*) présente les principaux sous-packages. Ils correspondent aux différents stéréotypes et ont été précisés dans le but d'obtenir une plus grande clarté. Les packages seront ultérieurement répartis dans des artéfacts (cf. *5.6.1 Les artéfacts de l'application WebTask*).

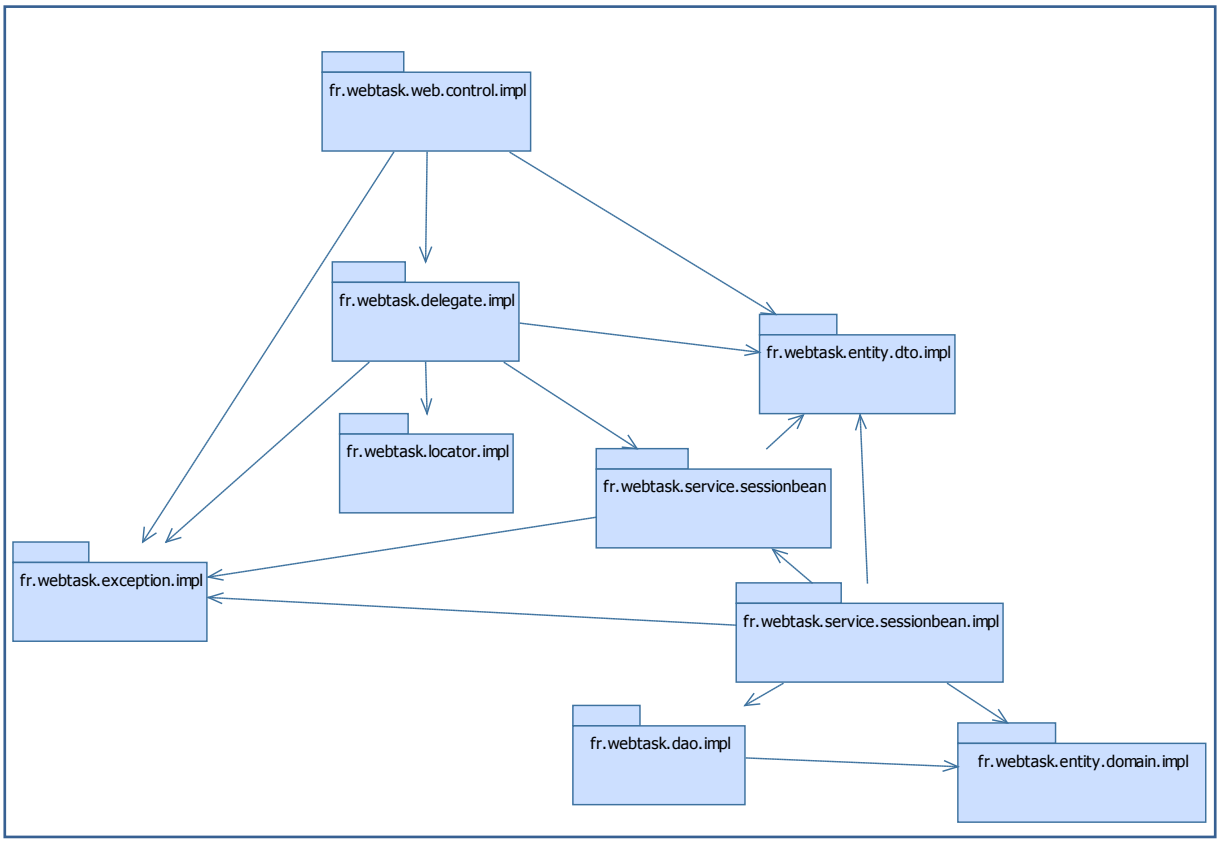

Figure 5.11, *Diagramme de packages*.

5.5 Sous-systèmes

L'architecture adoptée est orientée vers un découpage par couche applicative plutôt que par fonctionnalité. Les classes sont regroupées suivant leur type d'activité *(<<boundary>>, <<control >>, <<entity>>, <<delegate>>, <<lifecycle>>, <<facade>>, <<locator>>*). L'objectif est d'avoir une approche globale plus claire, au détriment toutefois du regroupement par sous-systèmes.

5.6 Déploiement

5.6.1 Les artéfacts de l'application WebTask

Un artéfact modélise une entité physique comme un fichier. Les artéfacts couramment rencontrés sont notamment :

- Des exécutables.
- Des librairies.
- Des fichiers source.
- Des fichiers de configuration.

L'application WebTask possède quatre artéfacts:

- L'archive *webtask.war* qui correspond à l'application web.
- L'archive *webtask-business.jar* qui correspond aux services distants.
- L'archive *webtask-common.jar* qui contient les packages communs à l'application web et aux services distants.
- L'archive *client-webtask-service.jar* qui peut être utilisée par toute application désirant contacter les services distants WebTask. Elle a des dépendances vers l'archive *webtask-common.jar*.

Ces quatre artéfacts sont ensuite décrits en précisant leur contenu (packages, fichiers XML, répertoires etc.).

5.6.1.1 L'archive webtask.war

5.6.1.1.1 Les packages

Les packages de l'archive sont listés dans le tableau *5.10* :

Tableau 5.10, *Packages de l'archive webtask.war.*

Package
fr.webtask.web.converter
fr.webtask.web.validator
fr.webtask.web.listener
fr.webtask.web.component
fr.webtask.web.control
fr.webtask.web.control.impl
fr.webtask.web.filter
fr.webtask.web.util
fr.webtask.web.storetext.configuration
fr.webtask.web.storetext.messages.pages
fr.webtask.web.storetext.messages.templates
fr.webtask.web.storetext.util

5.6.1.1.2 Les répertoires

Les répertoires de l'archive sont décrits dans le tableau *5.11* :

Tableau 5.11, *Répertoires de l'archive webtask.war.*

Répertoire	Fonction
/webtask/images	Images des pages web.
/webtask/css	Fichiers de style CSS.
/webtask/js	Fichiers JavaScript.
/webtask/pages	Pages JSP.
/webtask/fragments	Fragments des pages JSP.
/webtask/templates	Modèles des pages.

5.6.1.1.3 Les fichiers XML

Les fichiers XML de l'archive sont décrits dans le tableau *5.12* :

Tableau 5.12, *Fichiers XML de l'archive webtask.war.*

Fichier XML	Fonction
/webtask/WEB-INF/web.xml	Descripteur de déploiement.
/webtask/WEB-INF/jsf/faces-config.xml	Déclaration des convertisseurs/validateurs/resource-bundles JSF.
/webtask/WEB-INF/jsf/faces-navigation-config.xml	Déclaration des règles de navigation JSF.
/webtask/WEB-INF/jsf/faces-managedbean-config.xml	Déclaration des beans managés JSF.
/webtask/WEB-INF/jsf/faces-component-config.xml	Déclaration des composants/render-kit JSF.
/webtask/WEB-INF/tiles-defs.xml	Déclaration des modèles de pages avec le framework Tiles.
/webtask/WEB-INF/jboss-web.xml	Fichier XML utilisé par JBoss.

5.6.1.1.4 Représentation graphique

La figure *5.12* présente la représentation graphique de l'archive *webtask.war*.

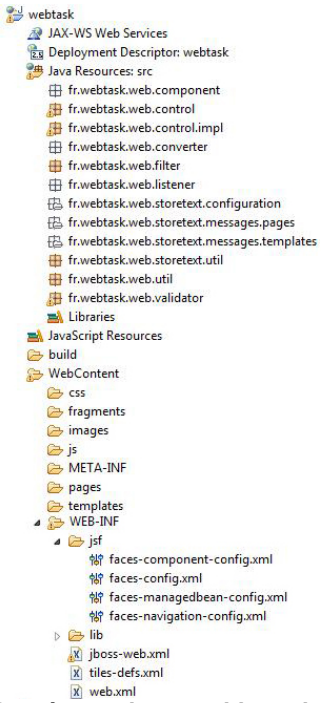

Figure 5.12, *Représentation graphique de l'archive webtask.war.*

5.6.1.2 L'archive webtask-business.jar

5.6.1.2.1 Les packages

La liste des packages est présentée dans le tableau *5.13* :

Tableau 5.13, *Packages de l'archive* webtask-business.jar.

Package
fr.webtask.entity.domain
fr.webtask.entity.domain.impl
fr.webtask.service.sessionbean.impl
fr.webtask.mail
fr.webtask.mail.impl
fr.webtask.service.storetext.configuration
fr.webtask.service.storetext.util
fr.webtask.service.util
fr.webtask.dao
fr.webtask.dao.impl

5.6.1.2.2 Les fichiers XML

Les fichiers XML de l'archive sont décrits dans le tableau *5.14* :

Tableau 5.14, *Fichiers XML de l'archive* webtask-business.jar.

Fichier XML	Fonction
/webtask-business/META-INF/persistence.xml	Déclaration de l'unité de persistance.

5.6.1.2.3 Représentation graphique

La figure *5.13* présente la représentation graphique de l'archive *webtask-business.jar*.

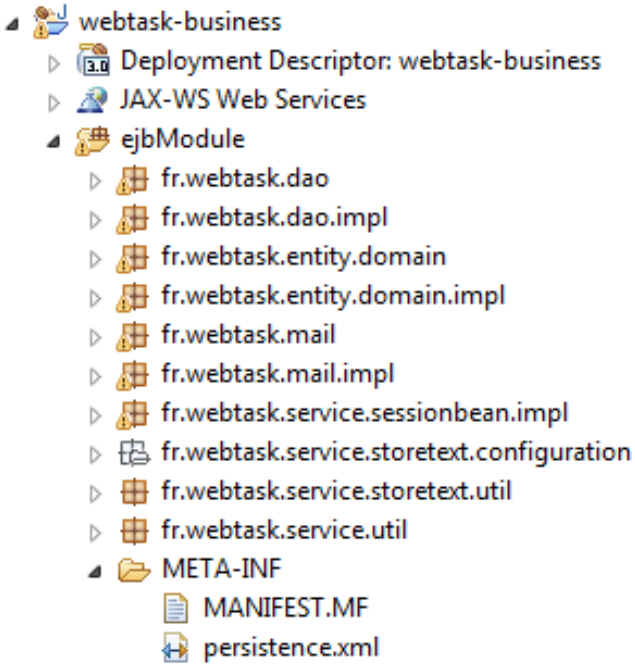

Figure 5.13, *Représentation graphique de l'archive webtask-business.jar.*

5.6.1.3 L'archive webtask-common.jar

5.6.1.3.1 Les packages

La liste des packages est présentée dans le tableau *5.15* :

Tableau 5.15, *Packages de l'archive webtask-common.jar.*

Package
fr.webtask.entity.dto
fr.webtask.entity.dto.impl
fr.webtask.service.sessionbean
fr.webtask.exception
fr.webtask.exception.impl
fr.webtask.enumeration
fr.webtask.validator

5.6.1.3.2 Représentation graphique

La figure *5.14* présente la représentation graphique de l'archive *webtask-common.jar*.

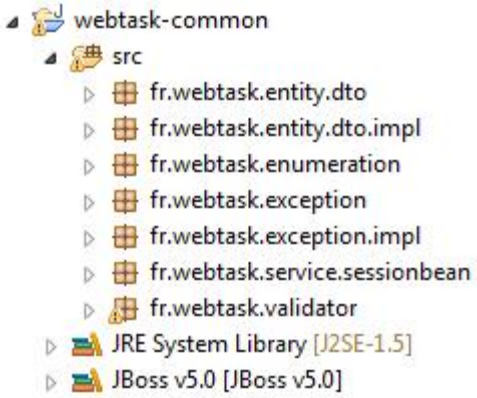

Figure 5.14, *Représentation graphique de l'archive webtask-common.jar.*

5.6.1.4 L'archive client-webtask-service.jar

5.6.1.4.1 Les packages

La liste des packages est présentée dans le tableau *5.16* :

Tableau 5.16, *Packages de l'archive client-webtask-service.jar.*

Package
fr.webtask.delegate
fr.webtask.delegate.impl
fr.webtask.delegate.exception.impl
fr.webtask.locator
fr.webtask.locator.impl

5.6.1.4.2 Représentation graphique

La figure *5.15* présente la représentation graphique de l'archive *client-webtask-service.jar*.

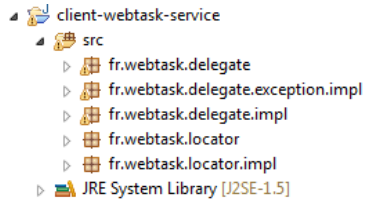

Figure 5.15, *Représentation graphique de l'archive client-webtask-service.jar*.

5.6.2 Diagramme de déploiement

La figure *5.16* présente un diagramme de déploiement. Son rôle est de décrire la disposition physique des matériels du système WebTask et de présenter la répartition des artéfacts sur ces matériels.

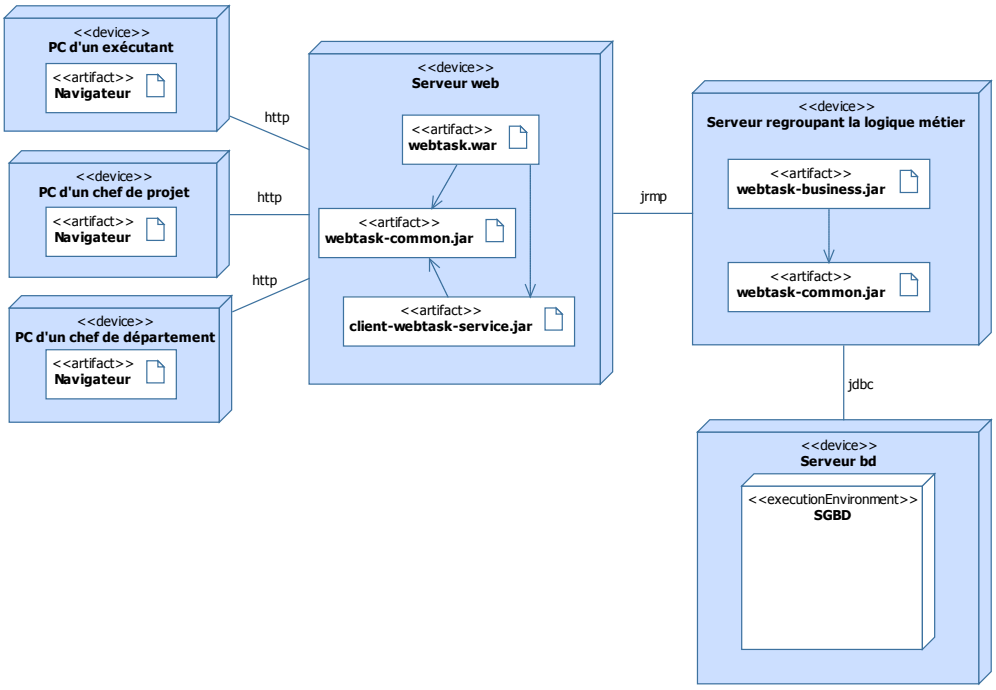

Figure 5.16, *Diagramme de déploiement*.

5.7 Base de données

5.7.1 Modèle conceptuel des données

Le modèle conceptuel des données (MCD) a été réalisé au cours de la phase d'analyse orientée objet. Il se matérialise sous la forme d'un diagramme de classes UML (cf. *4.3 Diagramme de classes de stéréotype <<entity>>*) où chaque classe possède le stéréotype <<*entity*>>. Pour que le MCD soit pleinement orienté *base de données*, il serait cependant nécessaire de le réadapter en supprimant les méthodes des classes.

5.7.2 Modèle physique des données

Le modèle physique des données (figure *5.17*), ou MPD, détaille la base de données d'un point de vue technique. Il indique les informations nécessaires à la création des scripts de génération de la base de données.

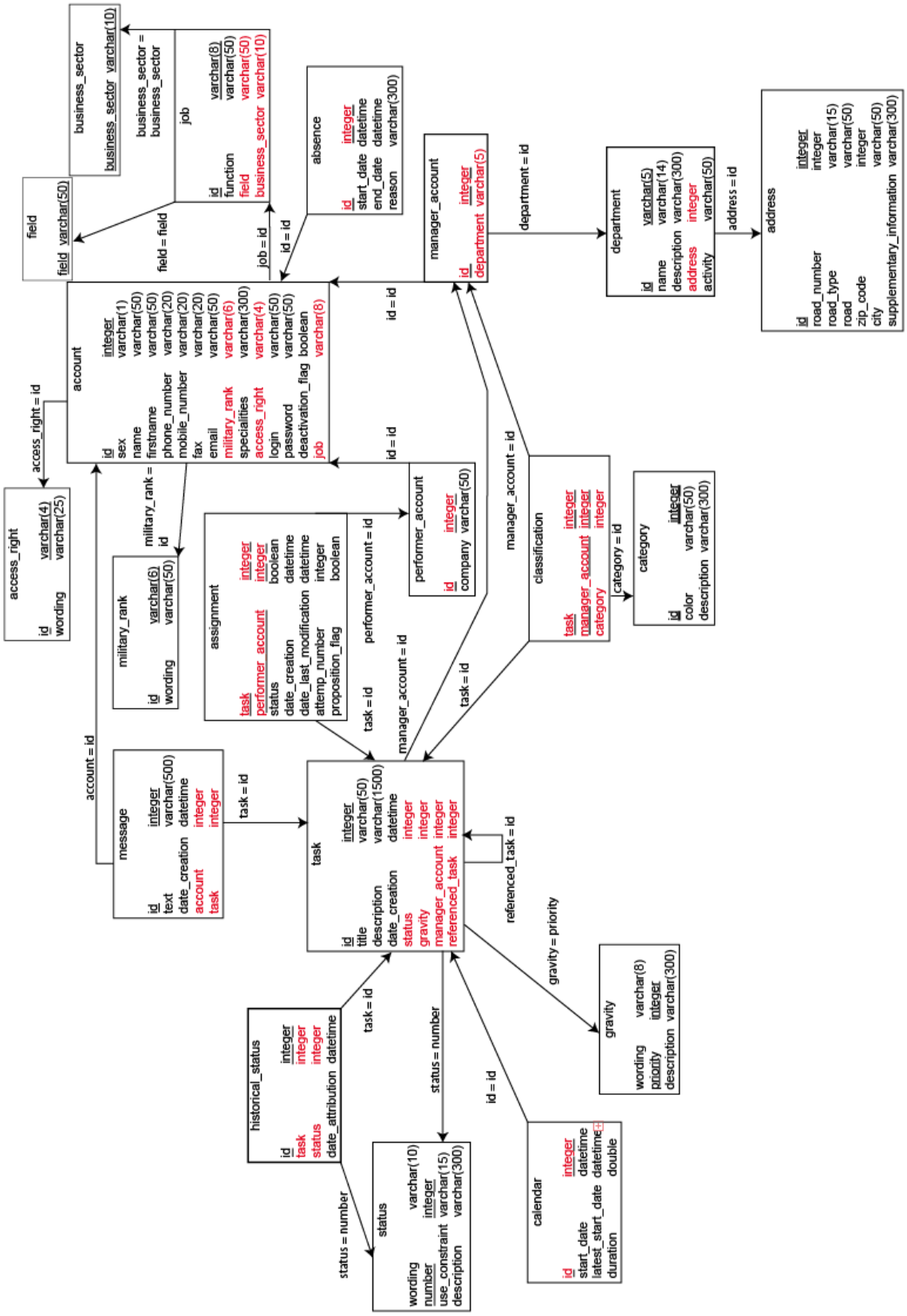

Figure 5.17, *Modèle physique des données*.

5.7.3 La vue *tasks_participants_view*

Une vue est une table virtuelle. Elle ne stocke pas les données mais elle fait référence à une ou plusieurs tables d'origine à travers une requête SELECT. Ses avantages sont, entre autre, de simplifier l'utilisation des tables, de masquer des jointures et de sauvegarder des requêtes fréquemment utilisées sous un nom.

La vue *tasks_participants_view* recense les participants (chef de département, chef de projet, exécutant(s)) de chaque tâche. Elle est constituée de quatre colonnes :

- Identifiant d'une tâche.
- Identifiant d'un compte.
- Email d'un compte.
- Droit d'accès d'un compte.

Cette vue permet à certains services distants de récupérer les différents participants d'une tâche pour ensuite leur envoyer un mail (cf. *7.2 Fonctionnement de l'envoi d'un mail aux participants d'une tâche*).

Chapitre 6. Choix technologiques

L'application WebTask s'appuie sur une plate-forme de développement Java EE 5. Le terme *Java EE* (*Enterprise Edition*) désigne la spécification d'un serveur d'applications EE. L'implémentation du serveur est à la charge des sociétés désirant utiliser cette spécification. Le rôle d'une spécification est de décrire la constitution et le fonctionnement d'un système.

La plate-forme EE est employée par de nombreuses entreprises à travers le monde car elle est reconnue pour sa stabilité, sa sécurité et sa robustesse.

Les différences entre une application Java SE (*Standard Edition*) et une application Java EE résident dans le fait que cette dernière contient des composants, suivant la spécification EE, et doit être déployée dans un serveur d'applications EE pour y être exécutée. La plate-forme EE utilise donc l'ensemble des fonctionnalités de la SE.

La figure *6.1* présente la plate-forme Java EE 5. Elle met en évidence l'implication de java SE au sein même de l'architecture EE.

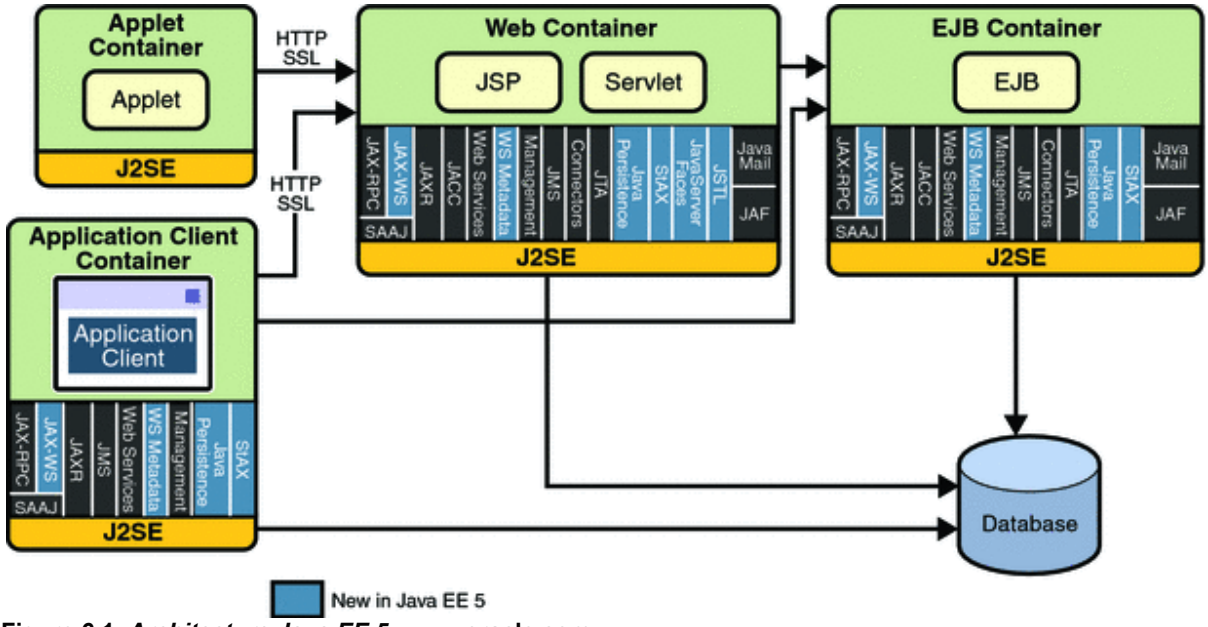

Figure 6.1, *Architecture Java EE 5*, www.oracle.com.

Ce chapitre présente les différents frameworks, bibliothèques et langages requis par le fonctionnement de l'application WebTask. Il introduit dans un premier temps les technologies du tiers *Client*, puis celles du tiers *Web* et du tiers *Logique Business* et enfin celles du tiers *Accès aux Données*.

6.1 Tiers Client

6.1.1 HTML et CSS

Le HTML (*HyperText Markup Language*) est le langage utilisé pour créer des pages web. Il est employé conjointement avec des feuilles de style en cascade (CSS : *Cascading Style Sheets*).

Le rôle des CSS est de fournir la mise en forme des éléments HTML d'une page. Le terme *en cascade* signifie que la mise en forme d'une page peut faire appel à plusieurs feuilles de style.

La démarche générale consiste à séparer le contenu (éléments HTML) de la mise en forme (feuilles de style CSS). L'avantage principal est la possibilité de définir des styles communs à plusieurs pages.

6.1.2 JavaScript

Le langage JavaScript a été développé conjointement par Netscape et Sun Microsystems puis a été standardisé par l'ECMA (ECMA-262).

C'est un langage de programmation de scripts principalement employé pour les pages web interactives. Il est interprété, c'est-à-dire que le script est lu, analysé et exécuté directement depuis le code source par le navigateur. Le terme *JavaScript* est maladroit car il sous-entend un lien de parenté qui n'existe pas avec le langage Java.

Le but de JavaScript est de réaliser des interactions entre un client et des pages web. Il offre la possibilité de contrôler et modifier le contenu du document affiché par le navigateur. Le code JavaScript peut aussi être relié à des événements de page web qui se manifestent lorsqu'un fait particulier se produit. Par exemple, lorsque l'utilisateur clique sur une page ou sur un lien hypertexte.

L'utilisation de JavaScript est devenue indispensable au cours du temps. Aujourd'hui la majorité des sites l'utilise pour aider le visiteur. C'est également le cas de l'application WebTask ne serait-ce qu'à travers l'emploi des composants JSF (cf. *6.2.3 Framework de présentation*) et RichFaces (cf. *6.1.3 Ajax*).

6.1.3 Ajax

Le terme Ajax (*Asynchronous JavaScript and XML*) a été introduit par Jesse James Garrett dans son article intitulé *Ajax, A New Approach to Web Applications ?*. Ajax propose à l'utilisateur un produit web dont l'interactivité est très proche de celle d'une application traditionnelle s'exécutant en local.

Il emploie le langage JavaScript afin d'interagir directement avec le serveur web. Le but est d'éviter le modèle d'application web classique dans lequel le navigateur est chargé d'envoyer des requêtes au serveur web et de traiter les requêtes en retour.

Lorsqu'une donnée est nécessaire, le code JavaScript effectue une requête asynchrone, le serveur retourne une réponse qui est ensuite analysée par le moteur Ajax et, enfin, l'interface utilisateur est mise à jour. Comme ce processus implique le transfert d'un plus petit nombre d'informations qu'avec le modèle web classique, les actualisations de l'interface utilisateur sont plus rapides au bénéfice de l'internaute. La figure *6.2* indique les différences entre le modèle d'application web traditionnel et le modèle d'application web Ajax.

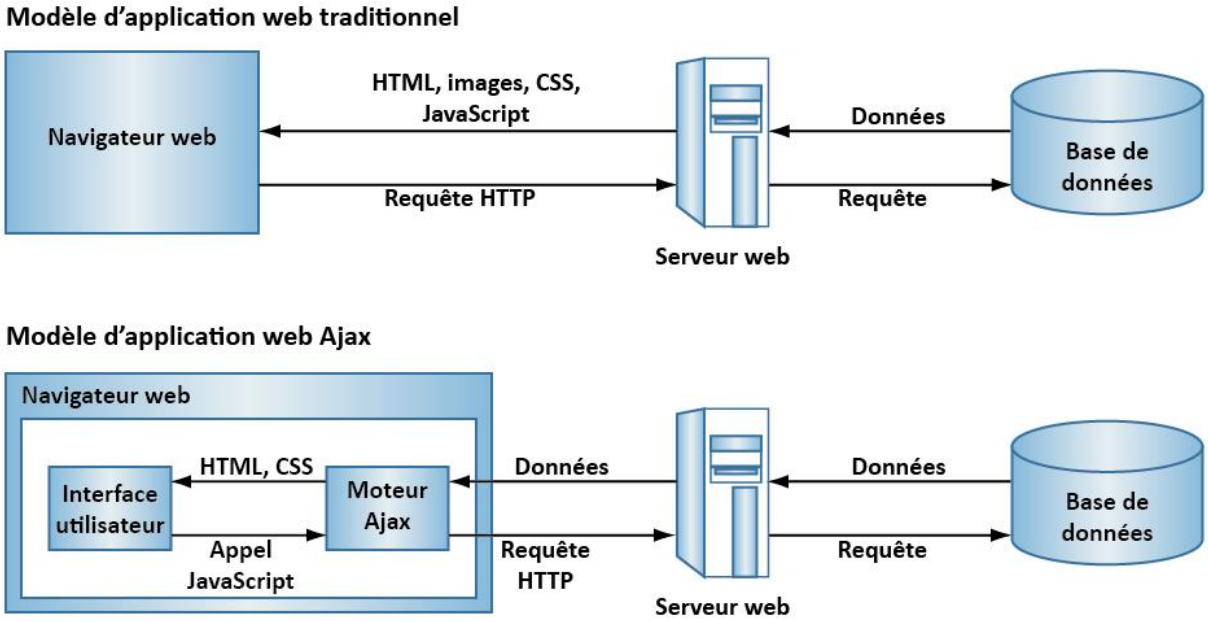

Figure 6.2, *Différences entre le modèle web traditionnel et le modèle web Ajax*, [ZAK 06].

L'application WebTask utilise la librairie **Richfaces version 3.3**. Cette bibliothèque fournit majoritairement des composants graphiques destinés à fonctionner avec Java Server Faces (cf. *6.2.3 Framework de présentation*). Ces composants exploitent les capacités d'Ajax énoncées ci-dessus et masquent l'implantation de l'objet *XMLhttpRequest*. A titre d'exemple, le composant *rich :suggestionbox* de Richfaces est employé par l'application afin de fournir la fonctionnalité d'auto-complétion à un champ (figure 6.*3*).

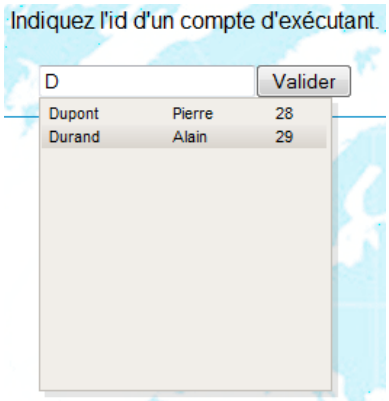

Figure 6.3, A*uto-complétion d'un champ de l'*application WebTask *avec Ajax.*

6.2 Tiers Web

6.2.1 Serveur d'applications

Les serveurs d'applications proposent un environnement d'exécution pour des applications s'appuyant sur les technologies Internet. Leur fonction principale est la gestion des composants.

Un composant est un programme qui s'intègre au sein d'un conteneur. On peut également l'envisager comme un objet qui vérifie quelques contraintes afin de s'y intégrer correctement. Les serveurs d'applications proposent plusieurs types de conteneurs, chacun étant adapté à certains types de composants.

Ces serveurs offrent également des services supplémentaires comme la gestion des transactions, la gestion de la sécurité, le support des connexions, l'équilibrage de charge (load balancing) etc.

Le tiers web quant à lui nécessite l'utilisation d'un conteneur de type *web*, offert par un serveur d'applications, et capable d'exécuter des pages JSP (sur lesquelles repose le framework Java Server Faces). La figure *6.4* ci-dessous présente l'algorithme d'exécution d'une page JSP.

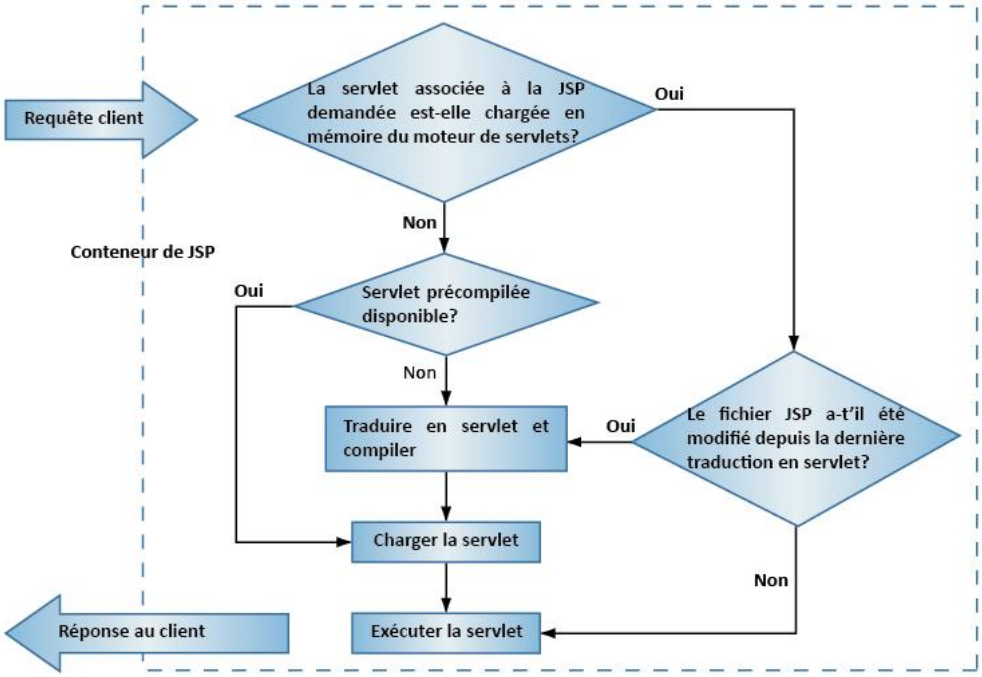

Figure 6.4, ***Exécution d'une page JSP.***

Ainsi, pour l'application WebTask, le conteneur web est fourni par l'intermédiaire du **serveur d'applications JBoss 5** écrit en Java et qui peut être employé sur tout système équipé d'une JVM (*Java Virtual Machine*).

JBoss a été développé au sein de la Société JBoss Inc., créée par Marc Fleury, le concepteur de la première version de JBoss. En avril 2006, Red Hat achète JBoss Inc. JBoss est l'un des serveurs open-source les plus populaires. Son architecture est synonyme de modularité et de performance.

La figure *6.5* ci-après décrit l'architecture du serveur d'application JBoss 5.

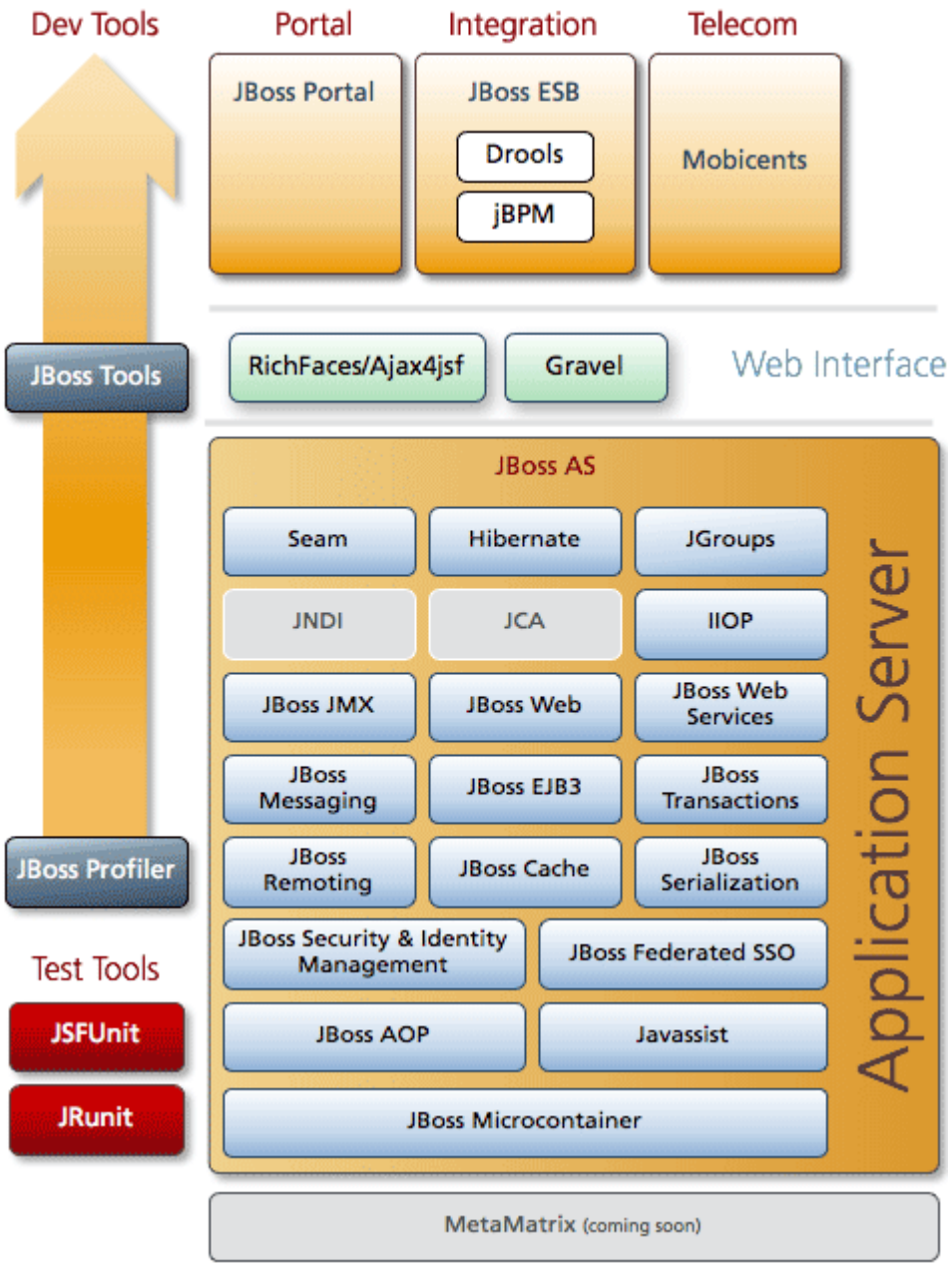

Figure 6.5, *Architecture du serveur d'application JBoss 5*, www.jboss.org.

Il n'est pas inutile de préciser que le rôle de conteneur web de JBoss est assuré par un service basé sur Tomcat, projet du groupe Apache.

6.2.2 Serveur web

Un serveur web est un processus dont la fonction principale consiste à répondre à des requêtes émises par des clients exécutant un navigateur.

L'application WebTask a recours au serveur HTTP **Apache (version 2)** réputé pour sa stabilité et sa fiabilité. Son maintient est assuré au sein de la fondation Apache. Selon le site internet *Netcraft.com*, Il est le serveur le plus répandu sur internet.

La figure 6.*6* le démontre en illustrant les parts de marché des principaux serveurs web disponibles actuellement.

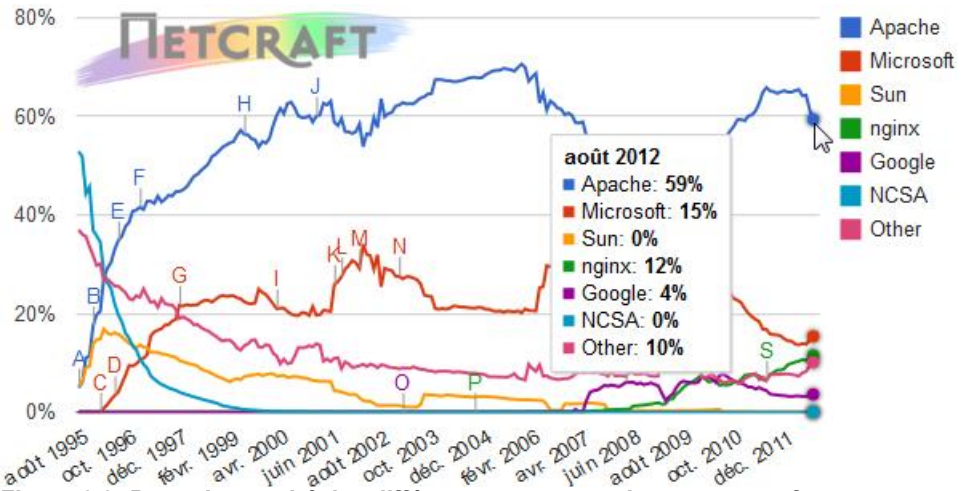

Figure 6.6, *Parts de marché des différents serveurs web*, news.netcraft.com.

Au sein du système WebTask, le serveur Apache est couplé au serveur d'application JBoss à l'aide du module mod_proxy qui inclut un protocole d'arrière-guichet AJP (*Apache JServ Protocol*).

L'utilisation d'un serveur web en frontal d'un serveur d'application est recommandée dans une architecture en production. L'intérêt principal se situe au niveau des performances, de la configurabilité et de la sécurité :

- Performances : le moteur HTTP d'un serveur web est beaucoup plus puissant et rapide que celui d'un serveur d'application.
- Configurabilité : un serveur web propose une plus grande palette de services qu'un serveur d'application.
- Sécurité : le conteneur web est isolé d'internet grâce au serveur web utilisé en frontal.

6.2.3 Framework de présentation

Le framework de présentation employé par l'application WebTask est **JSF** (*Java Server Faces*) **version Sun Ri 1.2**. Il respecte le pattern MVC 2 (cf. *5.3.2.4 Le pattern Modèle Vue Contrôleur 2*) et permet de faciliter et de standardiser la programmation d'application web avec Java. Son développement a tenu compte des différentes expériences acquises lors de l'utilisation de technologies standards (servlet, JSP, JSTL) et de différents frameworks comme Struts par exemple.

6.2.3.1 Concepts clés de JSF

Le cœur de la technologie JSF repose entre autres sur les concepts clés suivants :

- Les beans managés.
- La validation des données.

- Les conversions de données.
- La gestion des événements.
- Le cycle de vie à phases multiples d'une page JSF.

Ces différents concepts sont exposés succinctement au cours des prochains paragraphes afin de percevoir les capacités du framework JSF.

6.2.3.1.1 Les beans managés

Les beans managés sont des beans s'exécutant côté serveur et dont le cycle de vie est pris en charge par JSF. Ces beans sont exploités en association avec les composants graphiques contenus dans les formulaires web.

L'utilité principale des beans managés se résume aux actions suivantes :

- Validation de la donnée saisie dans un composant graphique.
- Prise en charge des événements générés par un composant.
- Réalisation du traitement permettant de définir vers quelle page la navigation doit se poursuivre après la validation d'un formulaire.

6.2.3.1.2 La validation des données

Dans la technologie Java Server Faces, la validation des données saisies dans un formulaire est assurée par des validateurs.

L'implémentation JSF fournit un jeu de validateurs par défaut répondant à quelques besoins courants. Ils permettent par exemple de contrôler la longueur d'une chaîne ou de vérifier qu'une valeur numérique est située dans un intervalle précis.

Ces validateurs standards ne traitent qu'une minorité des cas rencontrés lors des validations de données. De plus, ils s'avèrent souvent insuffisants dans des applications professionnelles où la logique métier impose des validations plus spécifiques et plus avancées. C'est la raison pour laquelle JSF offre la possibilité de créer des validateurs personnalisés.

6.2.3.1.3 Les conversions de données

La technologie JSF permet l'association entre un composant graphique, présent sur l'interface web, et une propriété d'un bean managé.

Lorsque ce genre d'association est mis en œuvre dans un processus de saisie, il est impératif qu'il y ait adéquation entre le type de la valeur saisie dans le composant et celui de la propriété du bean. De même, lorsqu'une telle association est utilisée dans un processus d'affichage, il faut que la valeur de la propriété soit affichable. C'est-à-dire que le type de la propriété du bean managé soit convertible en String.

Dans la plupart des situations, JSF applique une conversion automatique. Mais parfois il arrive que la conversion ne soit pas réalisable automatiquement. Cela se produit lorsqu'un composant graphique est lié à une propriété dont le type n'est pas standard. La mise en œuvre de la conversion nécessite alors l'emploi d'un convertisseur capable de réaliser une conversion spécifique.

L'implémentation JSF propose des convertisseurs standards permettant de convertir des valeurs numériques et des dates. Il est par ailleurs possible de créer des convertisseurs personnalisés.

6.2.3.1.4 La gestion des événements

Lorsqu'un utilisateur clique sur un bouton de soumission d'un formulaire par exemple, un événement particulier est déclenché. A ce moment, l'application JSF invoque la méthode de l'écouteur conçu pour prendre en charge l'événement en question.

6.2.3.1.5 Le cycle de vie à phases multiples d'une page JSF

En apparence, le cycle de vie d'une page JSF est identique à celui d'une page JSP : un client web émet une requête HTTP vers une page, puis le serveur répond en envoyant une page en HTML. Cependant, le cycle de vie d'une page JSF est divisé en phases afin de répondre efficacement aux attentes liées à cette technologie (validation, conversion, capture d'événements etc.). La figure *6.7* ci-après présente le cycle de vie à phases multiples d'une page JSF.

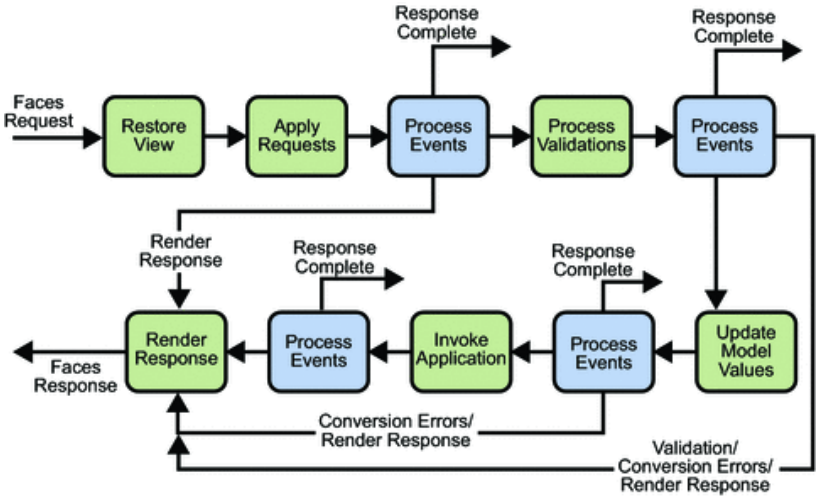

Figure 6.7, *Cycle de vie d'une page JSF*, www.oracle.com.

6.2.4 Templating

Les pages web de l'application WebTask possèdent un découpage sous forme d'entête, de menu, de contenu et de pied de page. Seule la zone de contenu change et les autres zones sont répétées à l'identique sur toutes les pages.

Pendant longtemps, les développeurs ont factorisé le code des zones communes aux différentes pages en faisant appel à des fragments JSP et à des directives ou tags JSP *include* (<%@ include %>, <jsp :include/>).Cette technique est limitée car, par exemple, la modification des noms des fragments entraîne des changements sur chaque page utilisatrice.

Pour y remédier, le site internet a recours au **framework Tiles (version 2 .1)**. A l'origine appelé *Components*, il a ensuite été renommé pour éviter des confusions. Ce framework permet de créer des templates qui factorisent les parties communes de l'interface graphique. La figure *6.8* ci-après expose les régions du template (*layout* dans la terminologie Tiles) d'une page web classique de l'application.

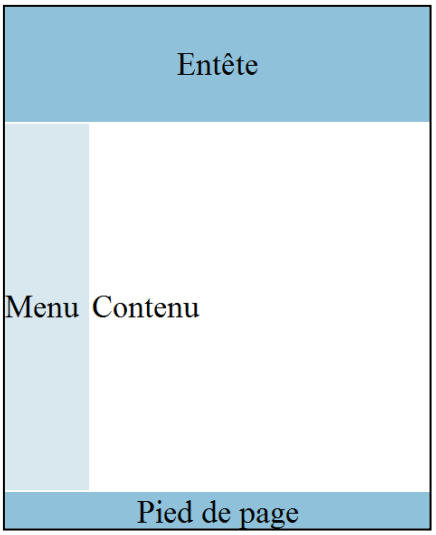

Figure 6.8, Régions d'un t*emplate Tiles* de l'application WebTask.

Le modèle de page présenté dans la figure *6.8* ci-dessus est constitué de quatre régions, chacune étant également appelée *tile* dans la terminologie du framework Tiles. Le layout et les tiles se matérialisent sous la forme de fichiers JSP. Le rôle du layout est de positionner les tiles aux emplacements appropriés.

6.2.5 Journalisation

La journalisation de l'application est prise en charge par l'**API Apache Log4j**. Cette librairie, très répandue dans les projets java, permet de debugger pendant et après l'étape de développement. La configuration est située dans le fichier *jboss-log4j.xml* de JBoss.

Les traces de l'application WebTask sont enregistrées dans un fichier qui tourne régulièrement. Chaque jour, elles sont inscrites dans le fichier de journalisation *webtask.log*. A minuit, ce fichier est dupliqué avec un nom de la forme *webtask.log.yyyy-MM-dd*. La journalisation de la journée suivante continue dans le fichier *webtask.log*.

Il est important de préciser que la journalisation n'est pas à confondre avec le test. Elle indique les erreurs de l'application en cours de fonctionnement tandis que le test offre une confiance relative dans le fonctionnement du système. La journalisation et le test sont donc deux démarches complémentaires.

6.3 Tiers Logique Business

6.3.1 Services distants

La prise en charge de la logique métier est attribuée à des services distants, mis en œuvre par des **composants EJB 3** (*Enterprise JavaBean*). Cette technologie repose sur la spécification JSR-220 qui est elle-même divisée en trois sous-spécifications :

- L'API EJB 3.
- L'API Java Persistence API.
- L'API Core Contracts and Requirements.

Au début des systèmes distribués, le standard d'objets distribués Corba (*Common Object Request Broker Architecture*) vit le jour. Il permet d'interconnecter des applications développées avec différents langages et de partager des objets entre celles-ci. Les inconvénients majeurs de Corba sont la lourdeur et la complexité de mise en œuvre. C'est pourquoi Sun a développé le modèle EJB qui est plus restrictif mais plus performant et plus simple à utiliser.

Les services distants se matérialisent sous la forme d'EJB Stateless Session Bean. Ce type d'EJB offre une collection de traitements dont chacun est représenté par une méthode. Lorsqu'une application cliente appelle une méthode d'un Session Bean, celui-ci l'exécute et retourne le résultat. Un Stateless Session Bean qui implémente un service distant doit adopter une visibilité elle aussi distante (*Remote*). Ce type d'accès signifie que l'EJB met à disposition ses méthodes à des clients s'exécutant sur des machines virtuelles différentes, et donc sur des machines physiques différentes. Les appels distants de méthodes sont réalisés par l'intermédiaire de la technologie RMI (cf. *5.2.3.1 Middleware d'accès aux services distants*) et nécessitent une sérialisation des arguments.

Le point *6.3.2 Persistance des données*, ci-dessous, décrit les technologies avec lesquelles les services distants réalisent des opérations CRUD (*Create : la création, Read : la lecture, Update : la mise à jour et Delete : la suppression*) auprès de la base de données relationnelle de l'application.

6.3.2 Persistance des données

La gestion de la persistance des données est traitée à l'aide des **EJB Entity Beans** et de l'**API JPA** (*Java Persistence API*).

Les Entity Beans établissent un pont entre la logique applicative et la base de données et se présentent sous la forme d'objets *simples* (POJO : *Plain Old Java Object*). Ces composants sont notamment employés lors de la réalisation d'un mapping objet/relationnel. Il s'agit d'une liaison automatique des propriétés d'un objet et d'une table d'une base de données relationnelle permettant de sauvegarder les informations de l'instance de l'objet.

L'API JPA est une composante essentielle de la spécification EJB 3. Elle fournit une couche d'abstraction de plus haut niveau que celle de JDBC (cf. *5.2.3.2 Middleware d'accès aux données*). L'implémentation sélectionnée pour l'application est Hibernate. Le framework de persistance JPA fournit entre autre les éléments suivants :

- Un mécanisme d'ORM (*Object Relational Mapping*) permettant le mapping objet/relationnel et vice versa.
- Une unité de persistance fournissant un gestionnaire d'Entity Beans (Entity Manager) gérant les opérations CRUD.

L'unité de persistance de l'application WebTask est configurée dans un fichier nommé *persistence.xml.* Le datasource qu'elle utilise est déclaré dans le fichier *mysql-ds.xml.*

6.3.3 Conteneur EJB

Les EJB Stateless Session Bean et les EJB Entity Bean sont des composants exécutés côté serveur et managés par un conteneur EJB.

Un conteneur EJB travail conjointement avec le serveur EJB dans lequel il est intégré. C'est une *boite noire* dont la fonction principale est de traiter le cycle de vie des EJB. Il les déploie, les stoppe tout en gérant leur conformité avec les spécifications du serveur. Un conteneur EJB propose également l'utilisation des services d'infrastructure disponibles au sein d'un serveur EJB (connexion à la base de données, gestion des transactions etc.).

La figure *6.9* illustre l'architecture globale d'un serveur EJB.

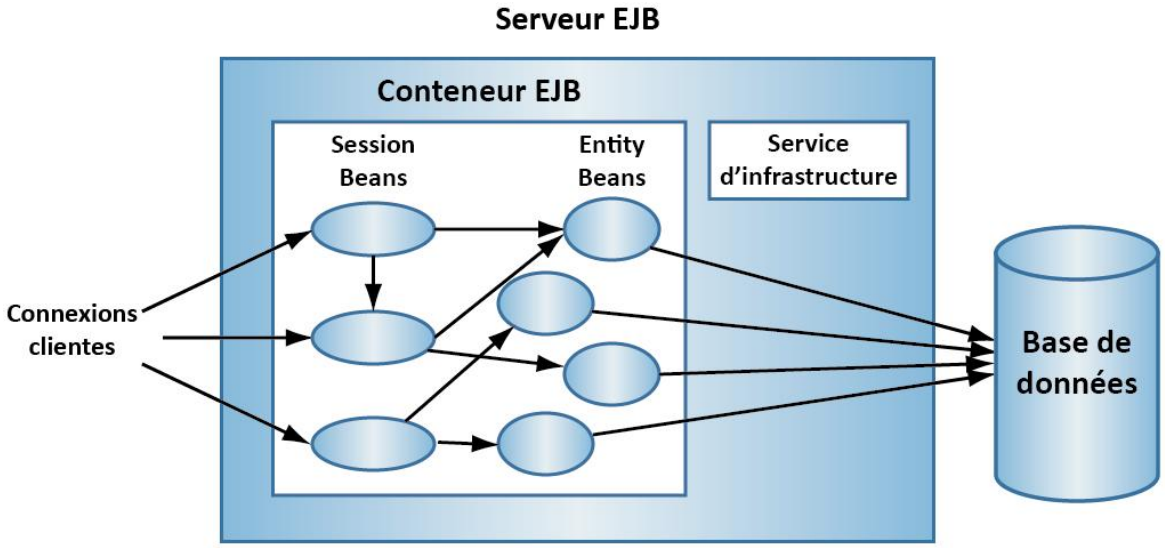

Figure 6.9, *Serveur EJB*, [CHU 06].

Le serveur EJB, utilisé pour le site internet WebTask, est le serveur d'application **JBoss 5**.Il intègre un conteneur EJB qui, par l'intermédiaire de l'injection de dépendance (cf. *5.3.2.2 Le pattern Inversion of Control*), réduit les couplages entre les objets de l'application.

6.3.4 Journalisation

La journalisation du tiers *Logique Business* fonctionne de la même manière que celle du tiers *Web*. Il est important de préciser que le fichier de journalisation ne se nomme plus *webtask.log* mais *webtask_services.log*.

6.3.5 Gestion des mails

Les données textuelles, contenues dans les corps des mails, sont générées par le moteur de templates FreeMarker. La figure *6.10* détaille le principe de cette librairie. L'envoi des messages électroniques se base sur l'utilisation de l'API JavaMail et du serveur de messagerie SMTP GMAIL.

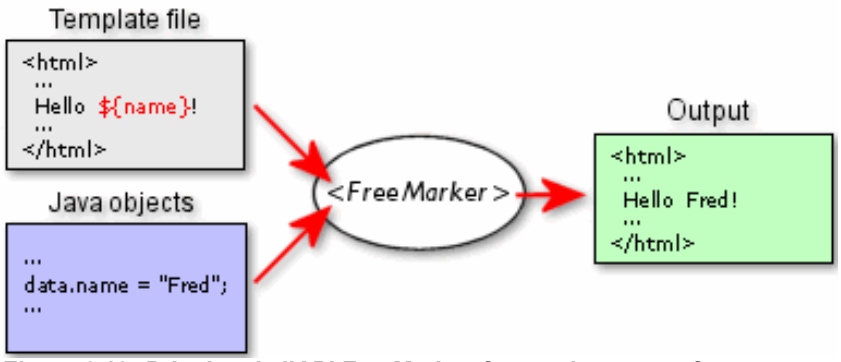

Figure 6.10, *Principe de l'API FreeMarker*, freemarker.sourceforge.net.

6.4 Tiers Données

6.4.1 Base de données relationnelle

Les données d'une base de données relationnelle sont décrites à l'aide d'un modèle de données appelé *modèle relationnel*. Il est apparu au début des années 1970 et doit son succès à sa grande simplicité.

Dans le modèle relationnel, les données sont structurées sous forme de relations (les tables), constituées d'attributs (les colonnes) et de tuples (les lignes).

Les bases de données relationnelles ont petit à petit remplacé les autres types de bases de données. Les systèmes de gestion de bases de données relationnelles sont actuellement des systèmes couramment utilisés. L'application WebTask a d'ailleurs recourt au SGBDR **MySQL**.

6.5 Récapitulatif

Le tableau *6.1* récapitule les choix technologiques :

Tableau 6.1, *Récapitulatif des choix technologiques.*

Tiers	Technologie
Client.	HTML, CSS, JavaScript, Ajax.
Web.	JBoss 5, Apache 2, JSF Sun RI 1.2, Tiles 2.1, Richfaces 3.3, log4j.
Logique Business.	EJB3, Hibernate, JBoss 5, log4j, JavaMail, FreeMarker.
Données.	SGBDR MySQL.

6.6 Enrichissement du diagramme de déploiement

Les choix technologiques étant désormais détaillés, il devient possible d'enrichir le diagramme de déploiement présenté au cours du chapitre *5.6.2 Diagramme de déploiement* :

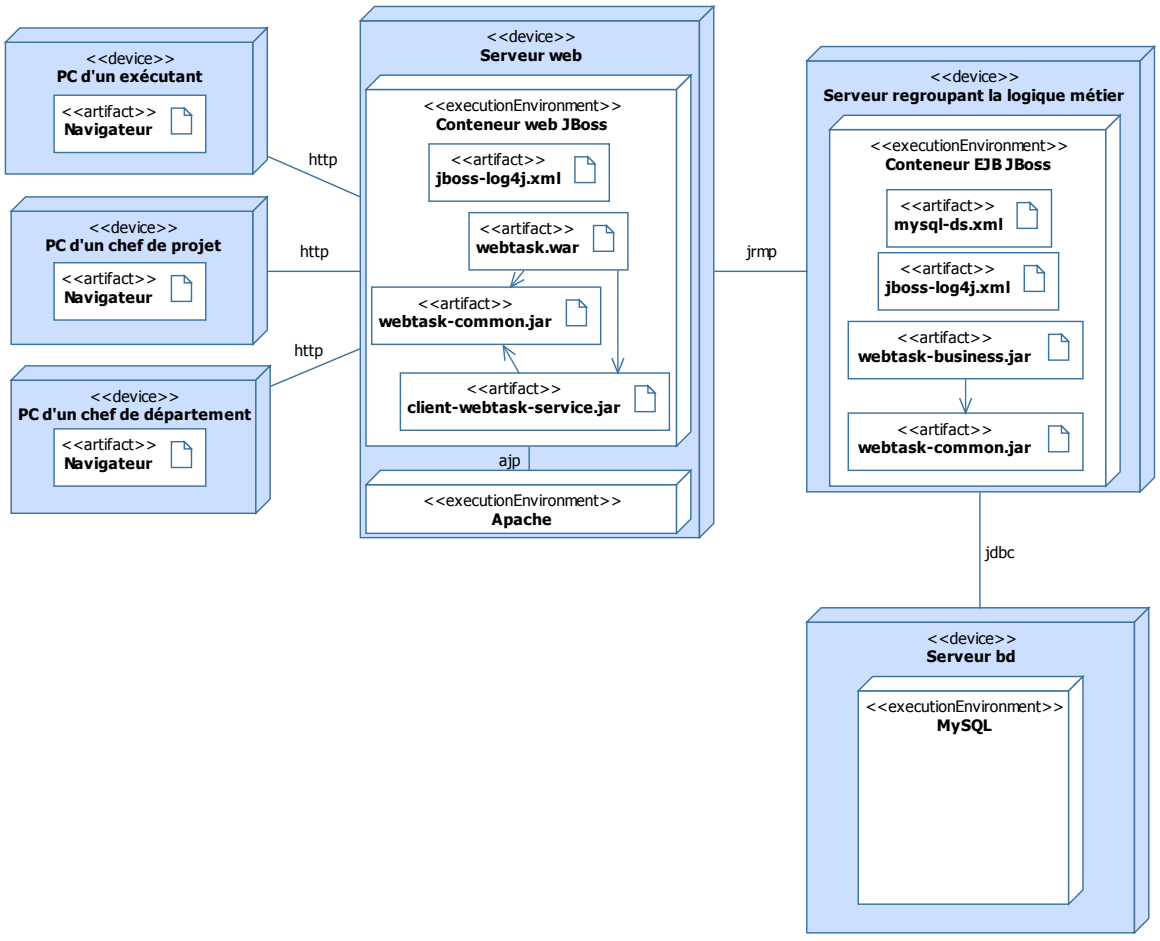

Figure 6.11, *Diagramme de déploiement enrichi.*

Chapitre 7. Conception orientée objet

La conception orientée objet consiste à établir une description détaillée des objets du système. Cette description tient compte des choix architecturaux (cf. *Chapitre 5. Principes Architecturaux*) et technologiques (cf. *Chapitre 6. Choix technologiques*) qui ont été préalablement effectués. Le rôle de la conception est de faire en sorte que la prochaine phase, c'est-à-dire la phase de réalisation, se déroule de la manière la plus efficace possible.

La conception emploie les même diagrammes UML que l'analyse orientée objet et dans les même buts. Ainsi pour chaque cas d'utilisation, plusieurs diagrammes de séquences et un diagramme de classes sont réalisés. Il est important de préciser qu'il existe un diagramme de séquences par flux d'événements important (nominal, alternatifs et erreurs). De même, le diagramme de classes montre les classes mentionnées dans les diagrammes de séquences précédemment créés.

Ce chapitre présente, dans un premier temps, les conventions de nommage employées par la conception orientée objet. Il explique ensuite, dans un second temps, comment certains services distants envoient des mails aux différents participants d'une tâche. Dans un troisième temps, il est mentionné pour quelques cas d'utilisation pris en exemple la liste des objets (de conception) candidats ainsi qu'un diagramme de séquences pour le flux d'événements nominal. Les diagrammes de séquences concernant les flux alternatifs et les flux d'erreur ne sont pas présentés. Pour les cas d'utilisation non pris en exemple, Le lecteur est invité à se reporter à l'annexe 3. La conception orientée objet de l'application WebTask n'a pas inclus la réalisation d'un diagramme de classes par cas d'utilisation. Cependant elle montre, dans un quatrième temps, les classes des différents cas d'utilisation regroupées par stéréotype à l'aide de diagrammes de classes.

7.1 Conventions de nommage

Les conventions de nommage instaurées au cours de la phase d'analyse orientée objet (cf. *Chapitre 4. Analyse orientée objet*) ont été modifiées. D'autres ont été également ajoutées. Voici les conventions utilisées pour la conception orientée objet :

- Les noms des objets de stéréotype <<*boundary*>> ont été suffixés par *ui.jsf* à l'exception de la page *index.jsf.*
- Les noms des objets de stéréotype <<*control*>> ont été suffixés par *Controller.*
- Les noms des objets de stéréotype <<*delegate*>> ont été suffixés par *Delegate.*
- Les noms des objets de stéréotype <<*locator*>> ont été suffixés par *Locator.*
- Les noms des objets de stéréotype <<*sessionFacade*>> ont été suffixés par *Service.*
- Les noms des objets de stéréotype <<*lifecycle*>> ont été suffixés par *DAOJpa à l'exception de l'objet SessionManager qui* gère les sessions.
- Les noms des objets de stéréotype <<*entity*>> n'ont pas été préfixés ou suffixés par un groupe de lettres.

7.2 Fonctionnement de l'envoi d'un mail aux participants d'une tâche

La description du fonctionnement de l'envoi d'un mail permet d'éviter de surcharger les diagrammes de séquences présentés dans le paragraphe *7.3 Exemples pour quelques cas d'utilisation* et dans l'annexe 3. En effet un diagramme de séquence indiquera désormais à l'aide d'une note textuelle qu'un mail est envoyé, puisque le fonctionnement de l'envoi d'un mail est explicité auparavant dans ce paragraphe-ci.

7.2.1 Liste des objets candidats

Le tableau *7.1* présente la liste des objets candidats pour l'envoi de mail :

Tableau 7.1, *Liste des objets candidats pour l'envoi de mail.*

Stéréotype	Objet
<<boundary>>	/
<<control>>	/
<<delegate>>	/
<<locator>>	/
<<sessionFacade>>	Un service distant quelconque qui envoie un mail.
<<lifecycle>>	TaskParticipantDAOJpa
<<entity>>	ObjectMail

7.2.2 Description des interactions entre objets

La figure *7.1* décrit les interactions entre objets pour l'envoi de mail :

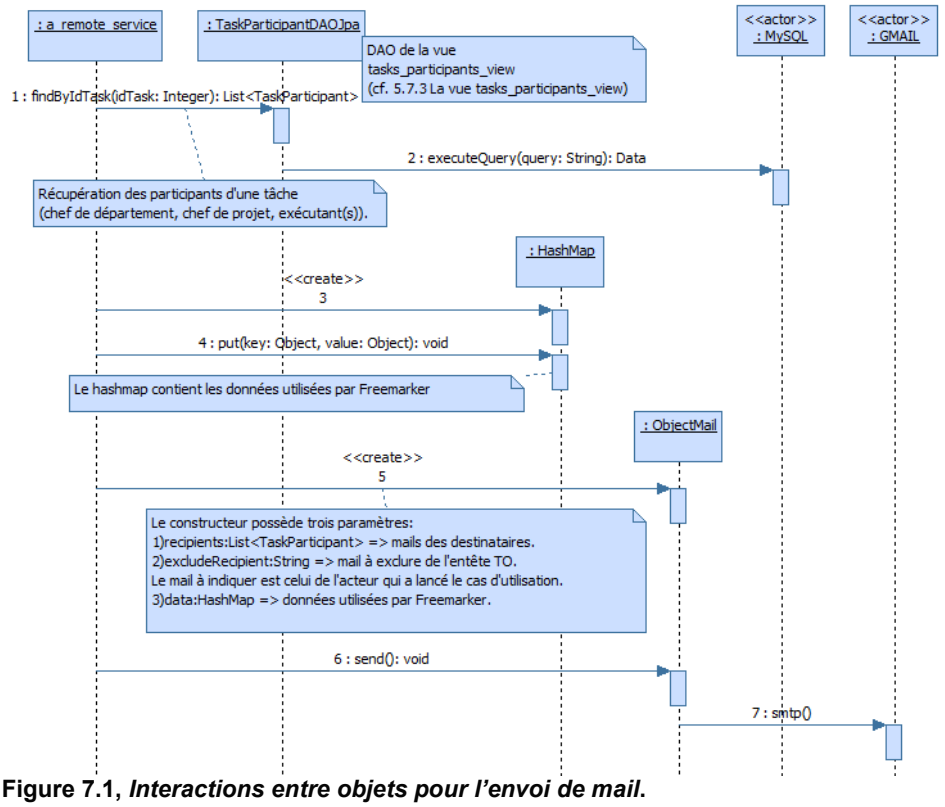

Figure 7.1, *Interactions entre objets pour l'envoi de mail.*

7.3 Exemples pour quelques cas d'utilisation

(Pour les autres cas d'utilisation, se reporter à l'annexe 3)

7.3.1 Le sous-système *Opérations de base*

7.3.1.1 Le cas d'utilisation *Se connecter*

7.3.1.1.1 Liste des objets candidats

Le tableau *7.2* présente la liste des objets candidats pour le cas d'utilisation *Se connecter :*

Tableau 7.2, *Liste des objets candidats pour le cas d'utilisation « Se connecter »*.

Stéréotype	Objet
<<boundary>>	index.jsf
<<control>>	LoginController
<<delegate>>	AccountDelegate, DepartmentDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	AccountService, DepartmentService
<<lifecycle>>	SessionManager, AccountDAOJpa, DepartmentDAOJpa
<<entity>>	Account, AccountDTO, Department, DepartmentDTO

7.3.1.1.2 Description des interactions entre objets

Les figures *7.2* et *7.3* décrivent, pour le cas d'utilisation *Se connecter*, les interactions entre objets :

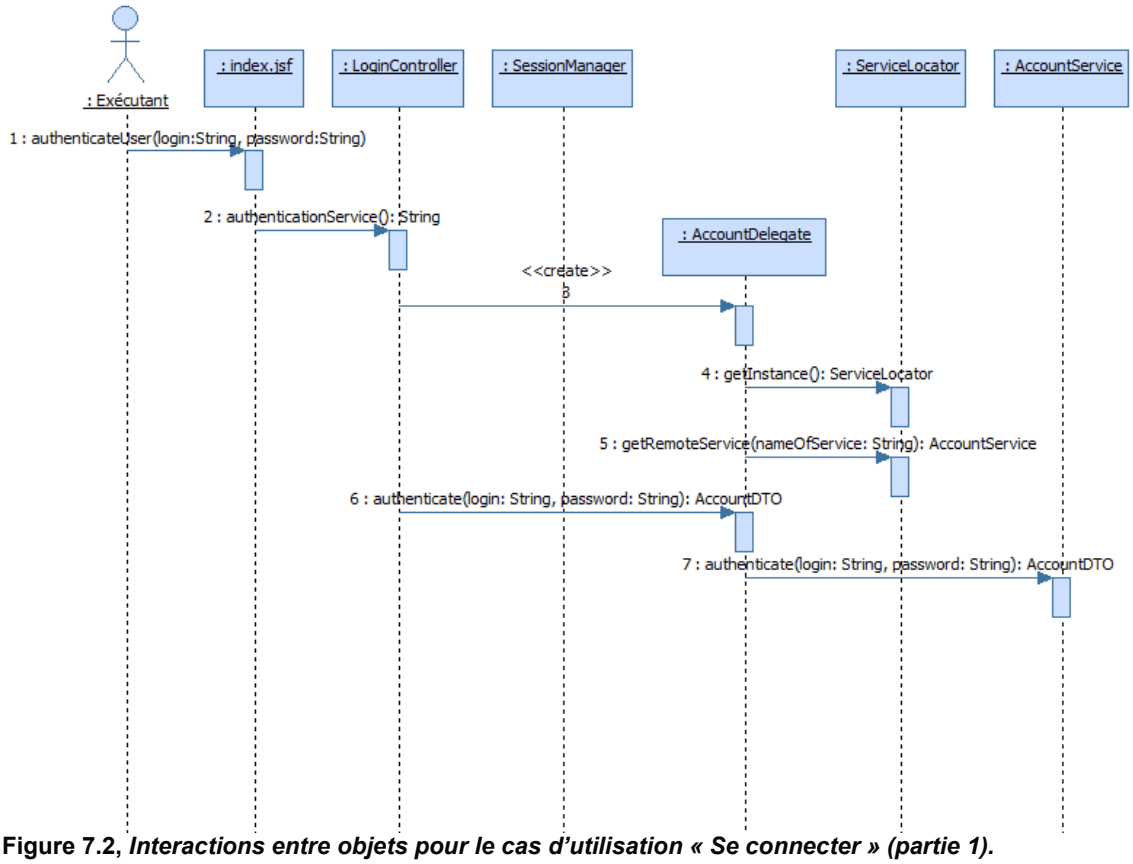

Figure 7.2, *Interactions entre objets pour le cas d'utilisation « Se connecter » (partie 1).*

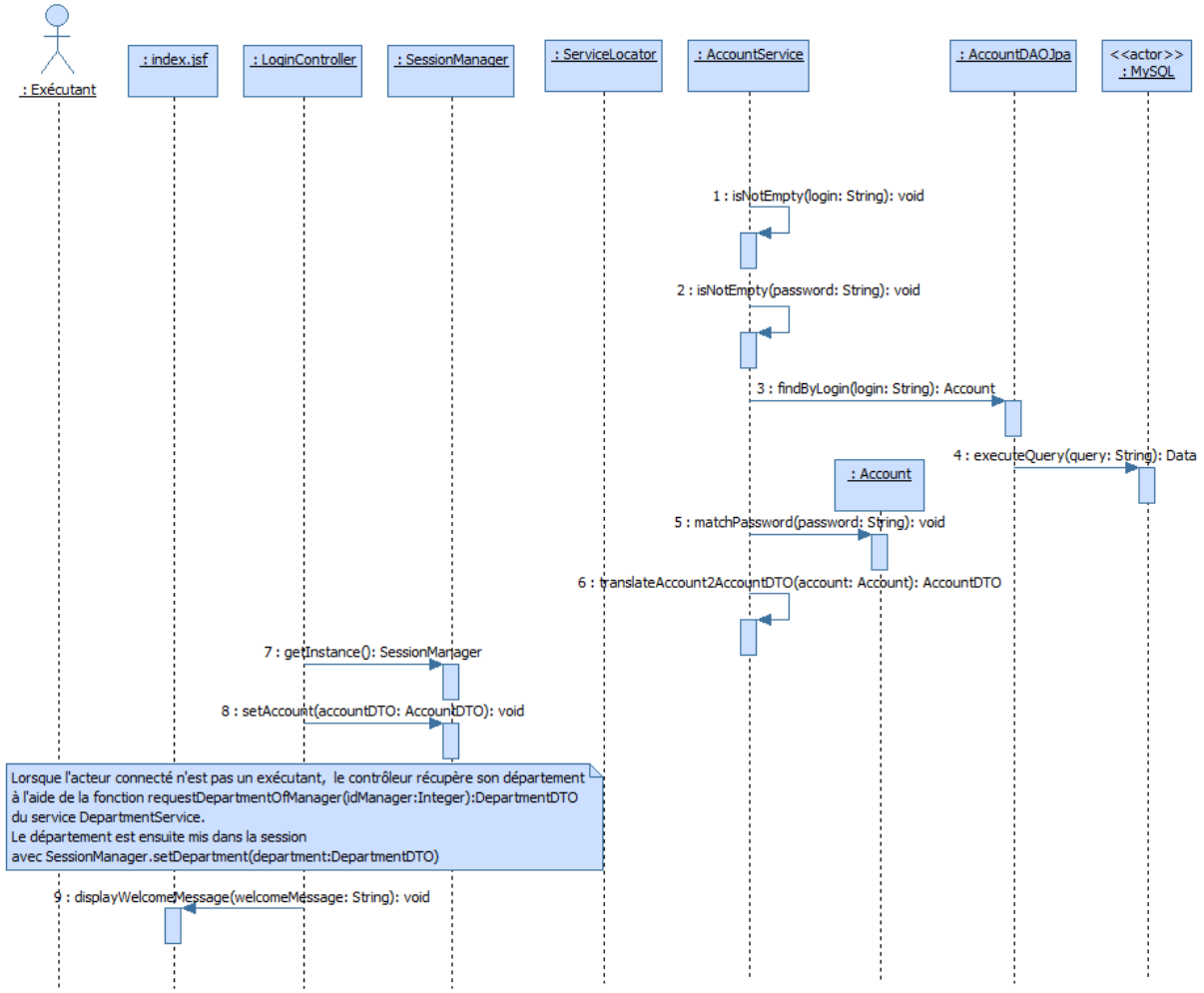

Figure 7.3, *Interactions entre objets pour le cas d'utilisation « Se connecter » (partie 2).*

7.3.1.2 Le cas d'utilisation *Se déconnecter*

7.3.1.2.1 Liste des objets candidats

Le tableau *7.3* présente la liste des objets candidats pour le cas d'utilisation *Se déconnecter :*

Tableau 7.3, *Liste des objets candidats pour le cas d'utilisation « Se déconnecter ».*

Stéréotype	Objet
<<boundary>>	consultation-task-ui.jsf, index.jsf
<<control>>	LoginController
<<delegate>>	/
<<locator>>	/
<<sessionFacade>>	/
<<lifecycle>>	SessionManager
<<entity>>	/

7.3.1.2.2 Description des interactions entre objets

La figure *7.4* décrit, pour le cas d'utilisation *Se déconnecter*, les interactions entre objets :

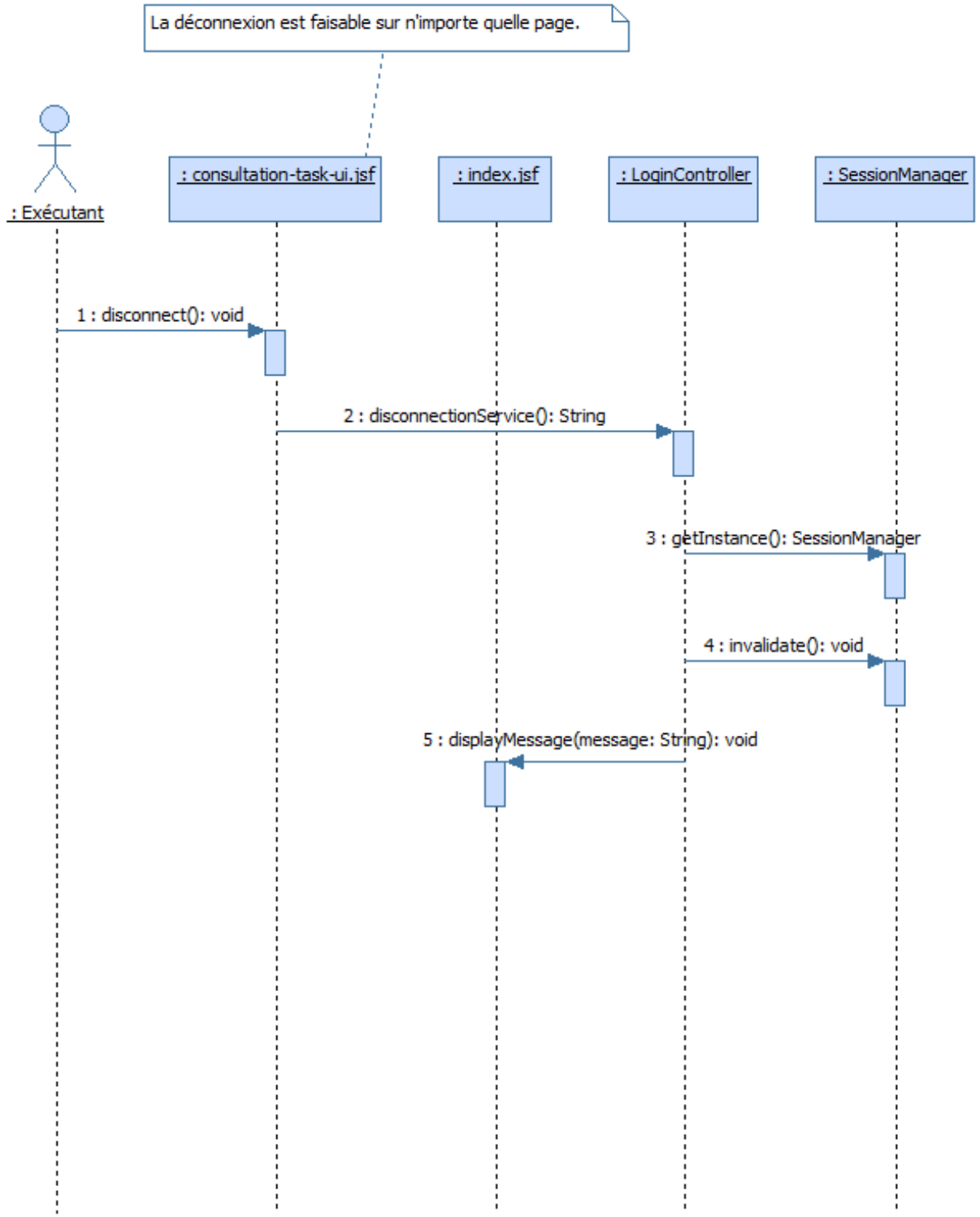

Figure 7.4, *Interactions entre objets pour le cas d'utilisation « Se déconnecter »*.

7.3.2 Le sous-système *Gestion des tâches*

7.3.2.1 Le cas d'utilisation *Déposer une tâche*

7.3.2.1.1 Liste des objets candidats

Le tableau *7.4* présente la liste des objets candidats pour le cas d'utilisation *Déposer une tâche :*

Tableau 7.4, *Liste des objets candidats pour le cas d'utilisation « Déposer une tâche ».*

Stéréotype	Objet
<<boundary>>	creation-task-ui.jsf
<<control>>	CreationTaskController
<<delegate>>	TaskDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	TaskService
<<lifecycle>>	TaskDAOJpa, HistoricalStatusDAOJpa
<<entity>>	Task, TaskDTO, HistoricalStatus

7.3.2.1.2 Description des interactions entre objets

Les figures *7.5* et *7.6* décrivent, pour le cas d'utilisation *Déposer une tâche*, les interactions entre objets :

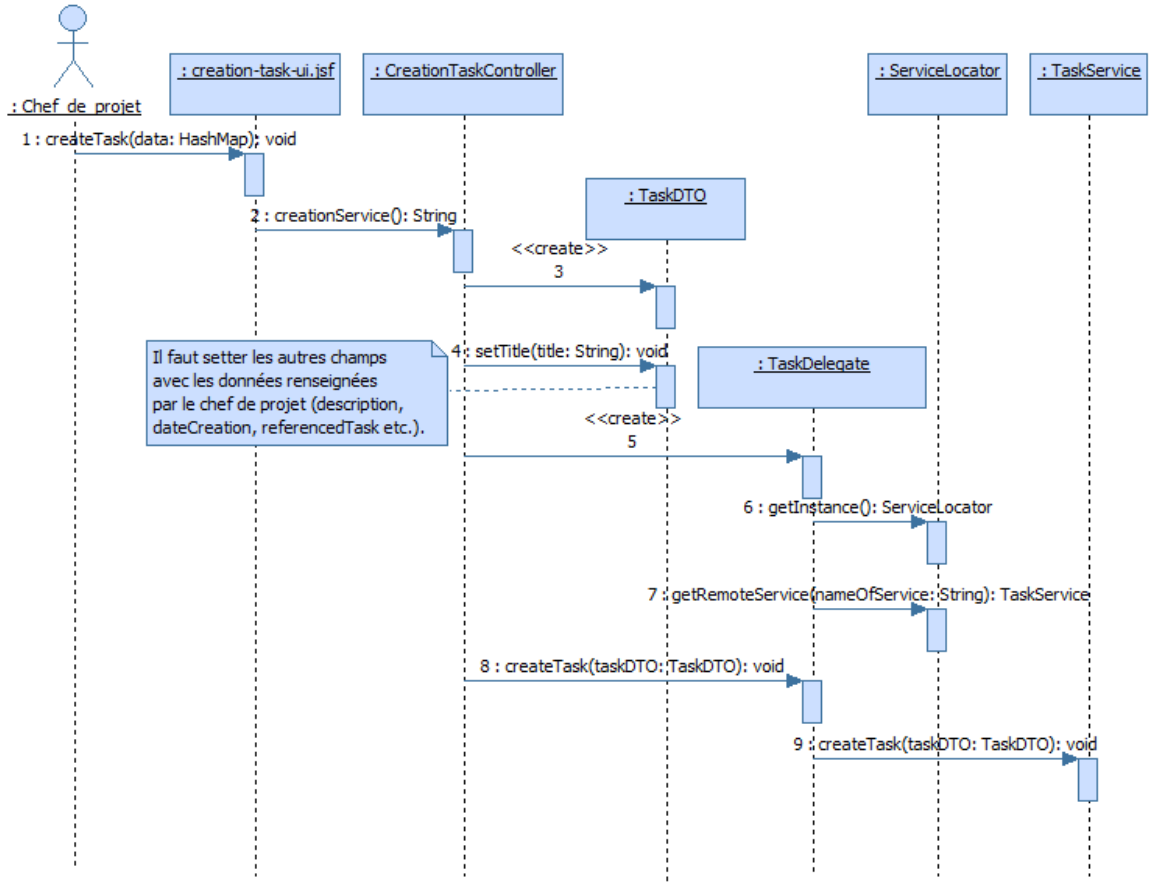

Figure 7.5, *Interactions entre objets pour le cas d'utilisation « Déposer une tâche » (partie 1).*

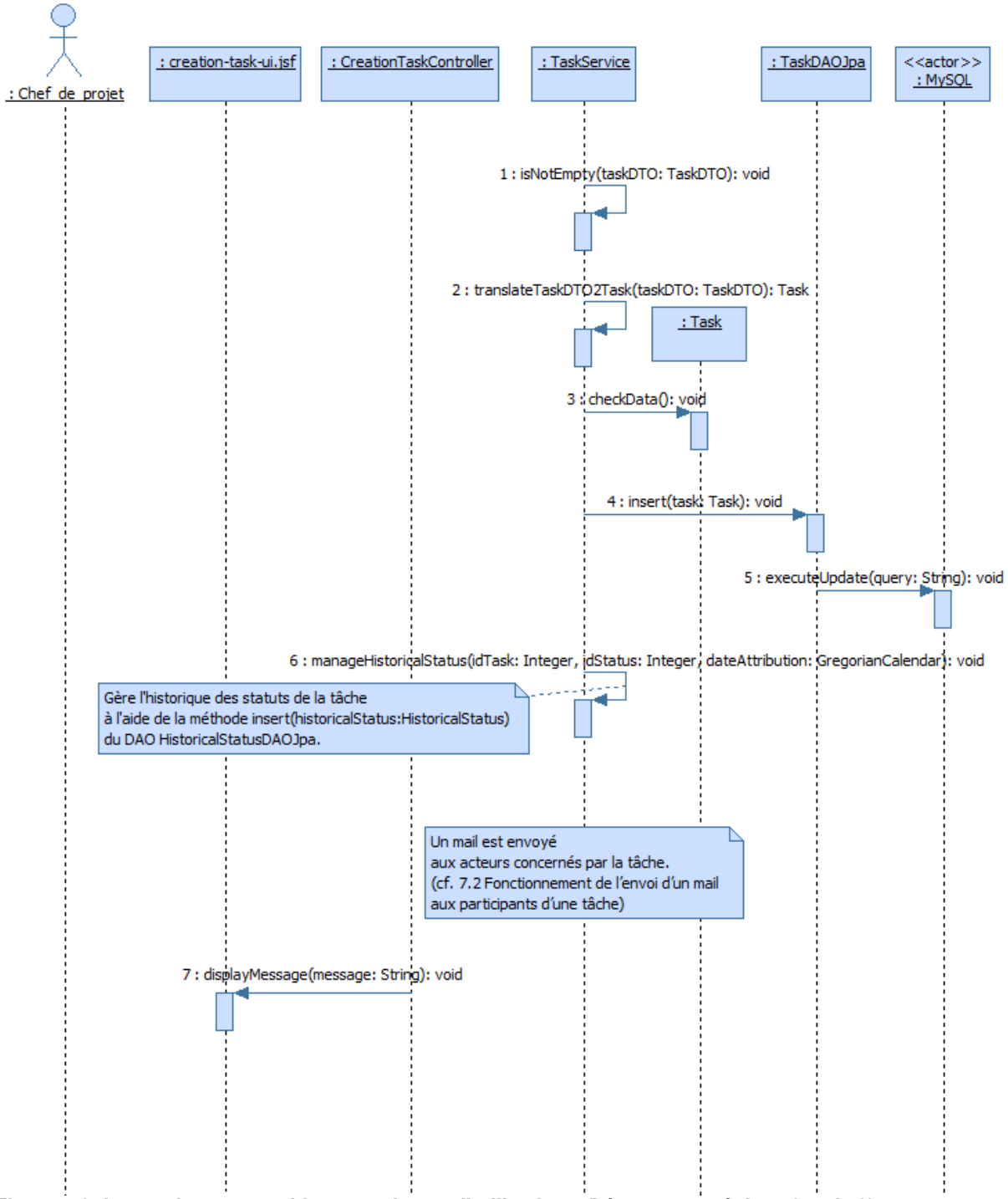

Figure 7.6, *Interactions entre objets pour le cas d'utilisation « Déposer une tâche » (partie 2).*

7.3.2.2 Le cas d'utilisation *Rechercher des tâches*

L'exemple traité dans ce paragraphe est celui d'une recherche de tâches dont le critère est le statut (cf. *2.2.3.3 Le cas d'utilisation Rechercher des tâches*).

7.3.2.2.1 Liste des objets candidats

Le tableau *7.5* présente la liste des objets candidats pour le cas d'utilisation *Rechercher des tâches :*

Tableau 7.5, *Liste des objets candidats pour le cas d'utilisation « Rechercher des tâches ».*

Stéréotype	Objet
<<boundary>>	consultation-task-ui.jsf
<<control>>	ConsultationTaskController
<<delegate>>	TaskDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	TaskService
<<lifecycle>>	SessionManager, TaskDAOJpa
<<entity>>	Task, TaskDTO, AccountDTO

7.3.2.2.2 Description des interactions entre objets

Les figures *7.7* et *7.8* décrivent, pour le cas d'utilisation *Rechercher des tâches,* les interactions entre objets *:*

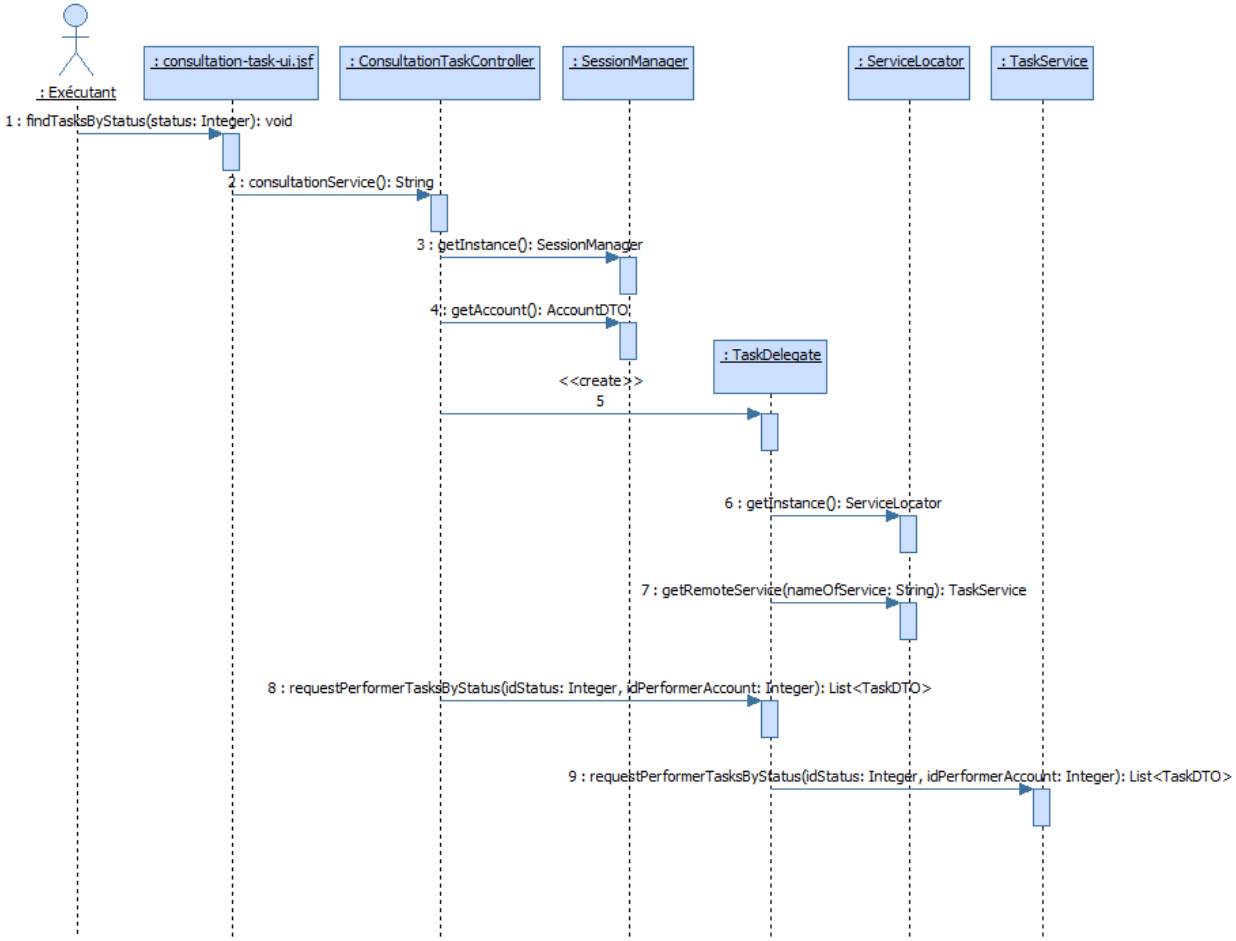

Figure 7.7, *Interactions entre objets pour le cas d'utilisation « Rechercher des tâches » (partie 1).*

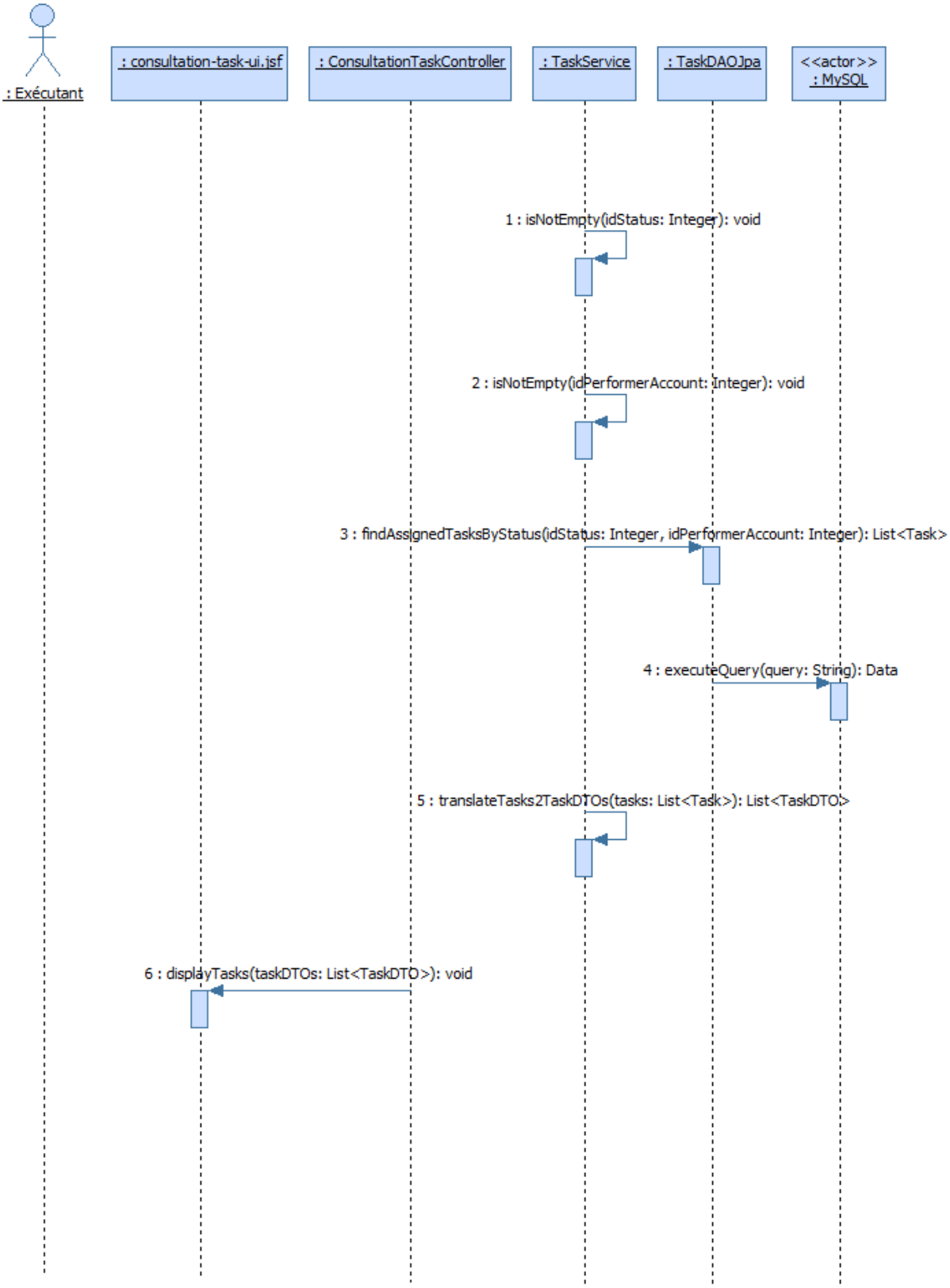

Figure 7.8, *Interactions entre objets pour le cas d'utilisation « Rechercher des tâches » (partie 2)*.

7.3.2.3 Le cas d'utilisation *Mettre à jour une tâche*

7.3.2.3.1 Liste des objets candidats

Le tableau *7.6* présente la liste des objets candidats pour le cas d'utilisation *Mettre à jour une tâche :*

Tableau 7.6, *Liste des objets candidats pour le cas d'utilisation « Mettre à jour une tâche »*.

Stéréotype	Objet
<<boundary>>	consultation-task-ui.jsf, update-task-ui.jsf
<<control>>	UpdateTaskController
<<delegate>>	TaskDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	TaskService
<<lifecycle>>	SessionManager, TaskDAOJpa, HistoricalStatusDAOJpa
<<entity>>	Task, TaskDTO, AccountDTO, HistoricalStatus

7.3.2.3.2 Description des interactions entre objets

Les figures *7.9* et *7.10* décrivent, pour le cas d'utilisation *Mettre à jour une tâche*, les interactions entre objets *:*

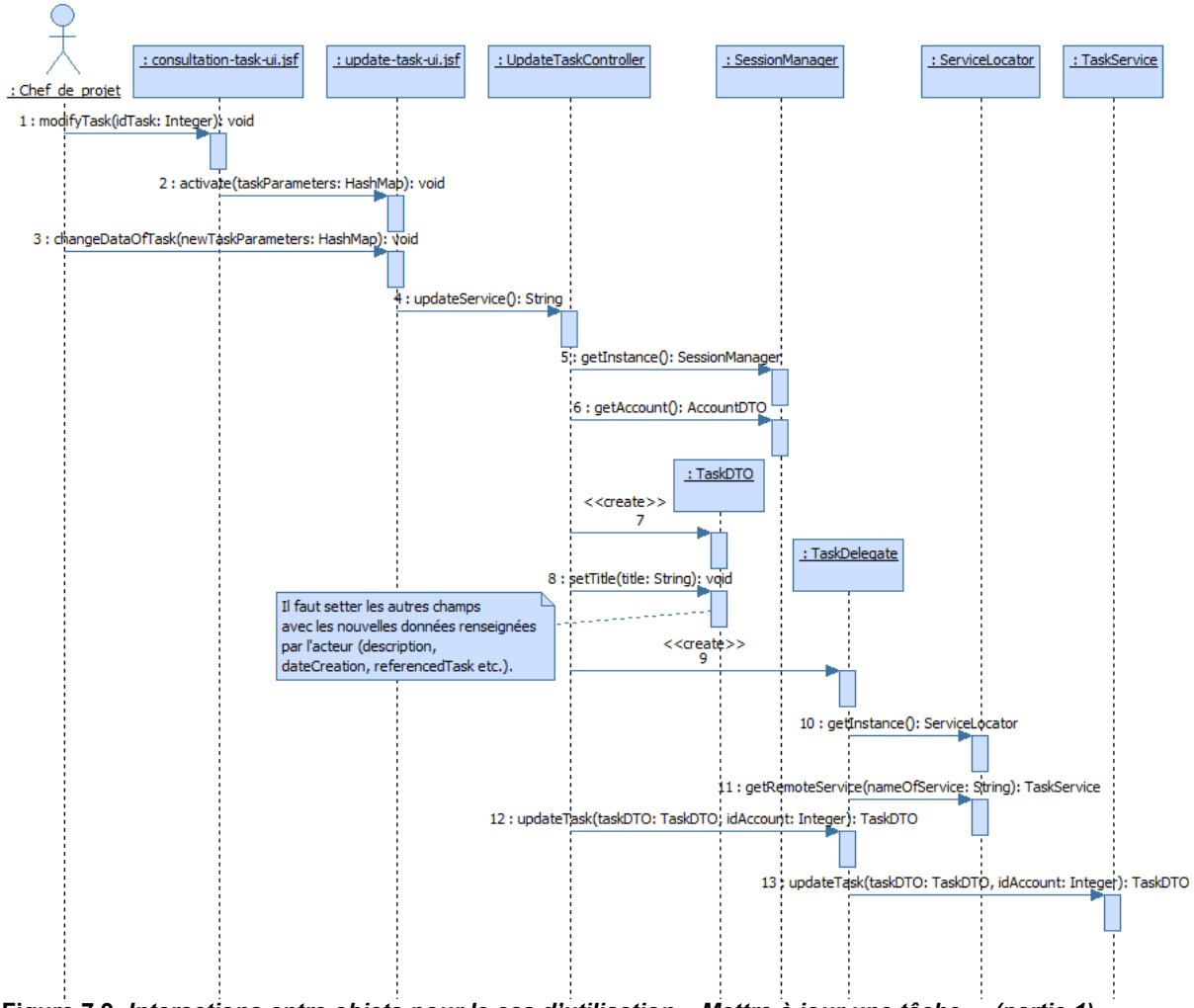

Figure 7.9, *Interactions entre objets pour le cas d'utilisation « Mettre à jour une tâche » (partie 1).*

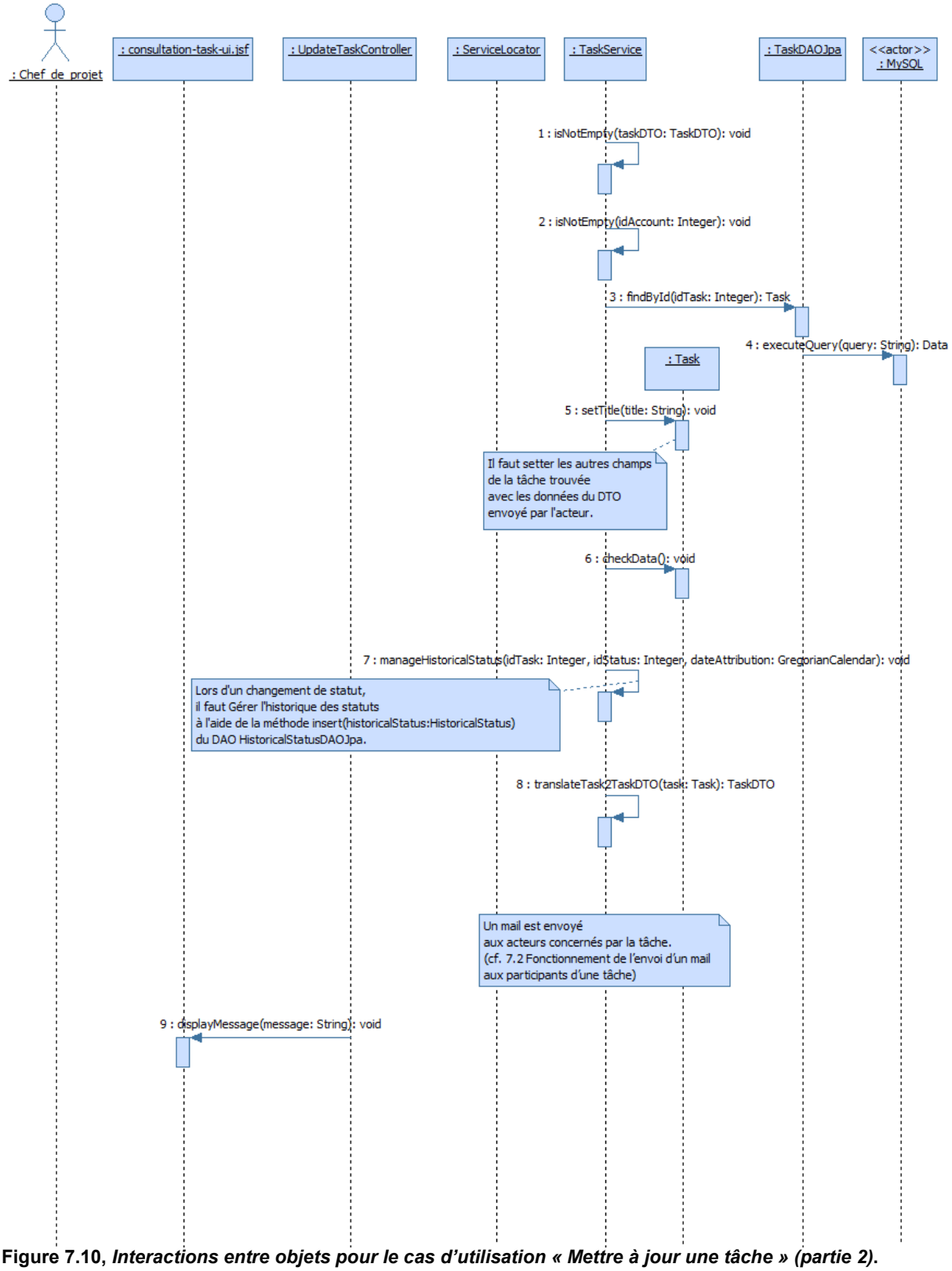

Figure 7.10, *Interactions entre objets pour le cas d'utilisation « Mettre à jour une tâche » (partie 2)*.

7.4 Regroupement des classes

7.4.1 Regroupement des classes de stéréotype <<boundary>>

La figure 7.27 présente le regroupement des classes de stéréotype <<boundary>> :

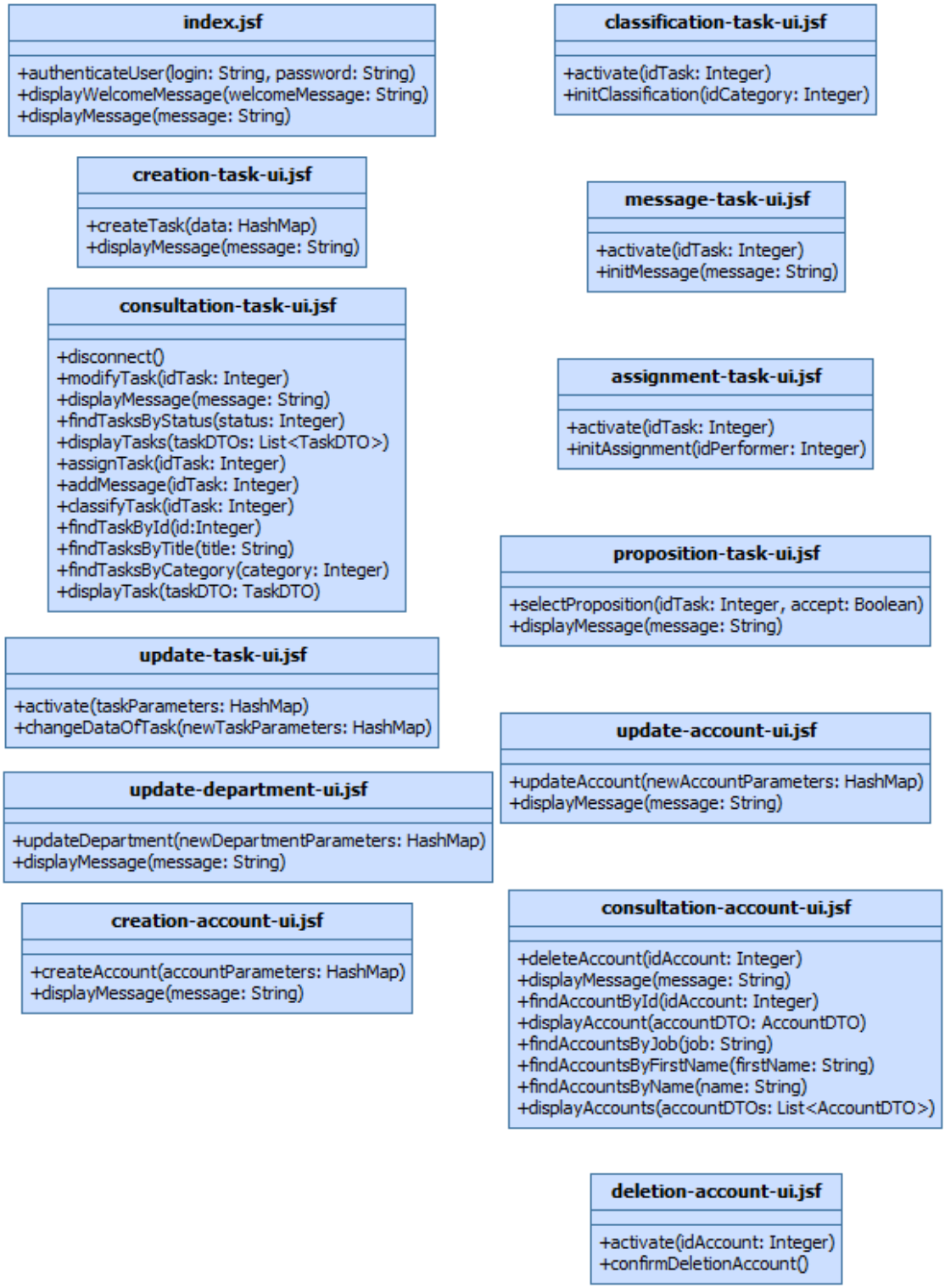

Figure 7.27, *Regroupement des classes de stéréotype <<boundary>>.*

7.4.2 Regroupement des classes de stéréotype <<control>>

La figure *7.28* présente le regroupement des classes de stéréotype <<control>> :

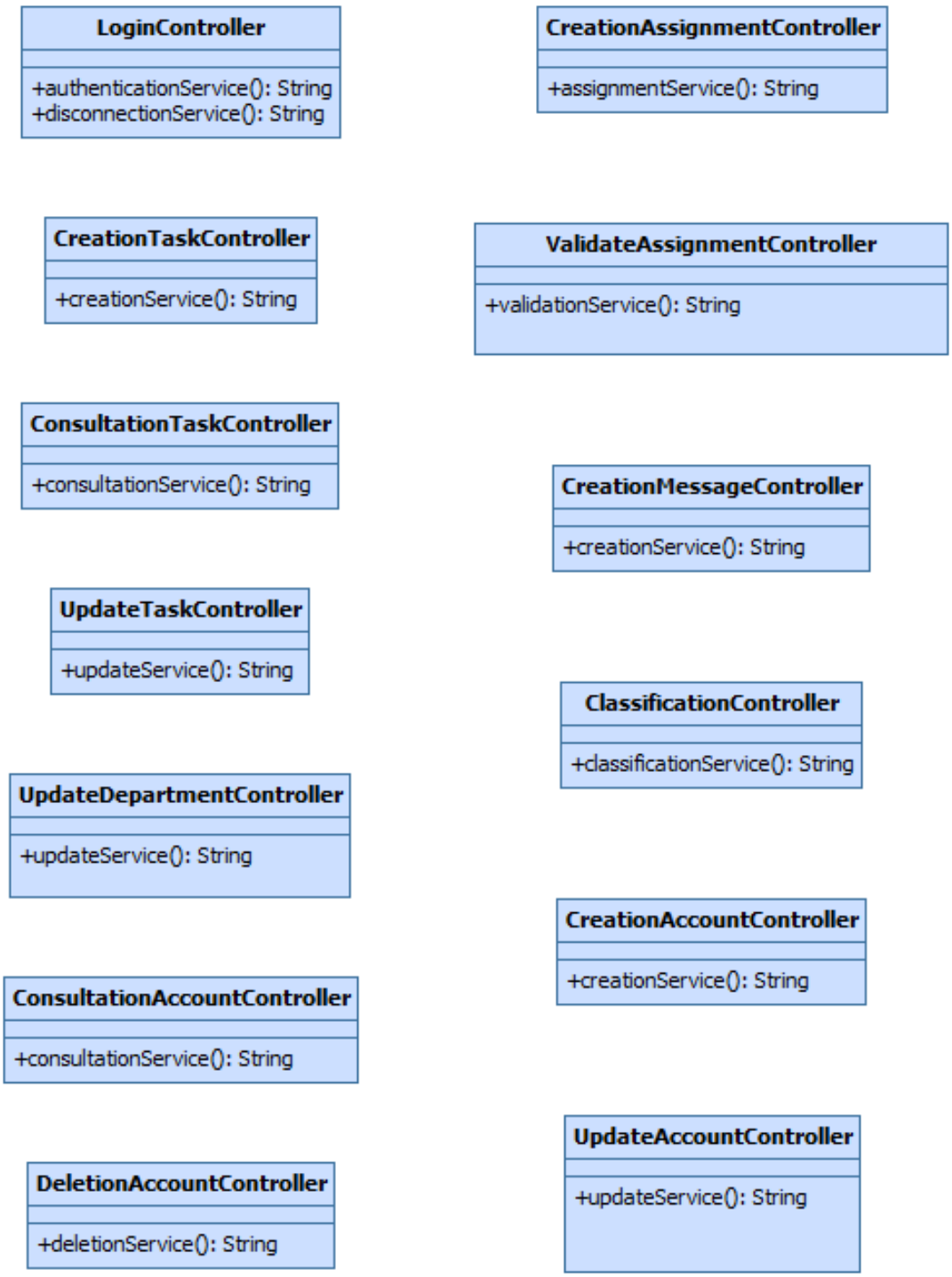

Figure 7.28, *Regroupement des classes de stéréotype <<control>>*.

7.4.3 Regroupement des classes de stéréotype <<delegate>>

La figure *7.29* présente le regroupement des classes de stéréotype <<delegate>> :

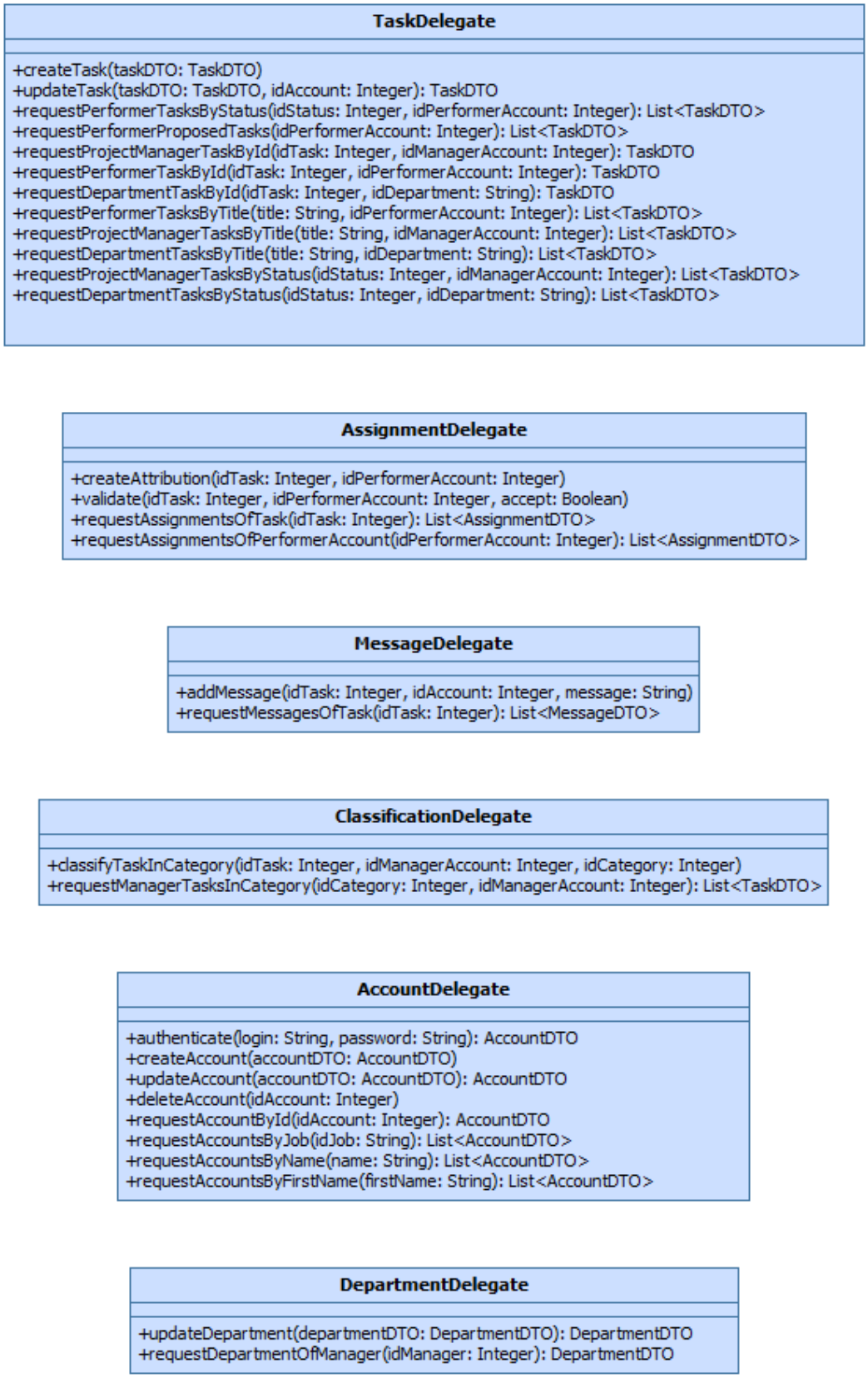

Figure 7.29, *Regroupement des classes de stéréotype <<delegate>>*.

7.4.4 Regroupement des classes de stéréotype <<locator>>

La figure *7.30* présente le regroupement des classes de stéréotype <<locator>> :

Figure 7.30, *Regroupement des classes de stéréotype <<locator>>*.

7.4.5 Regroupement des classes de stéréotype <<sessionFacade>>

La figure *7.31* présente le regroupement des classes de stéréotype <<sessionFacade>> :

Figure 7.31, *Regroupement des classes de stéréotype <<sessionFacade>>*.

7.4.6 Regroupement des classes de stéréotype <<lifecycle>>

La figure *7.32* présente le regroupement des classes de stéréotype <<lifecycle>> :

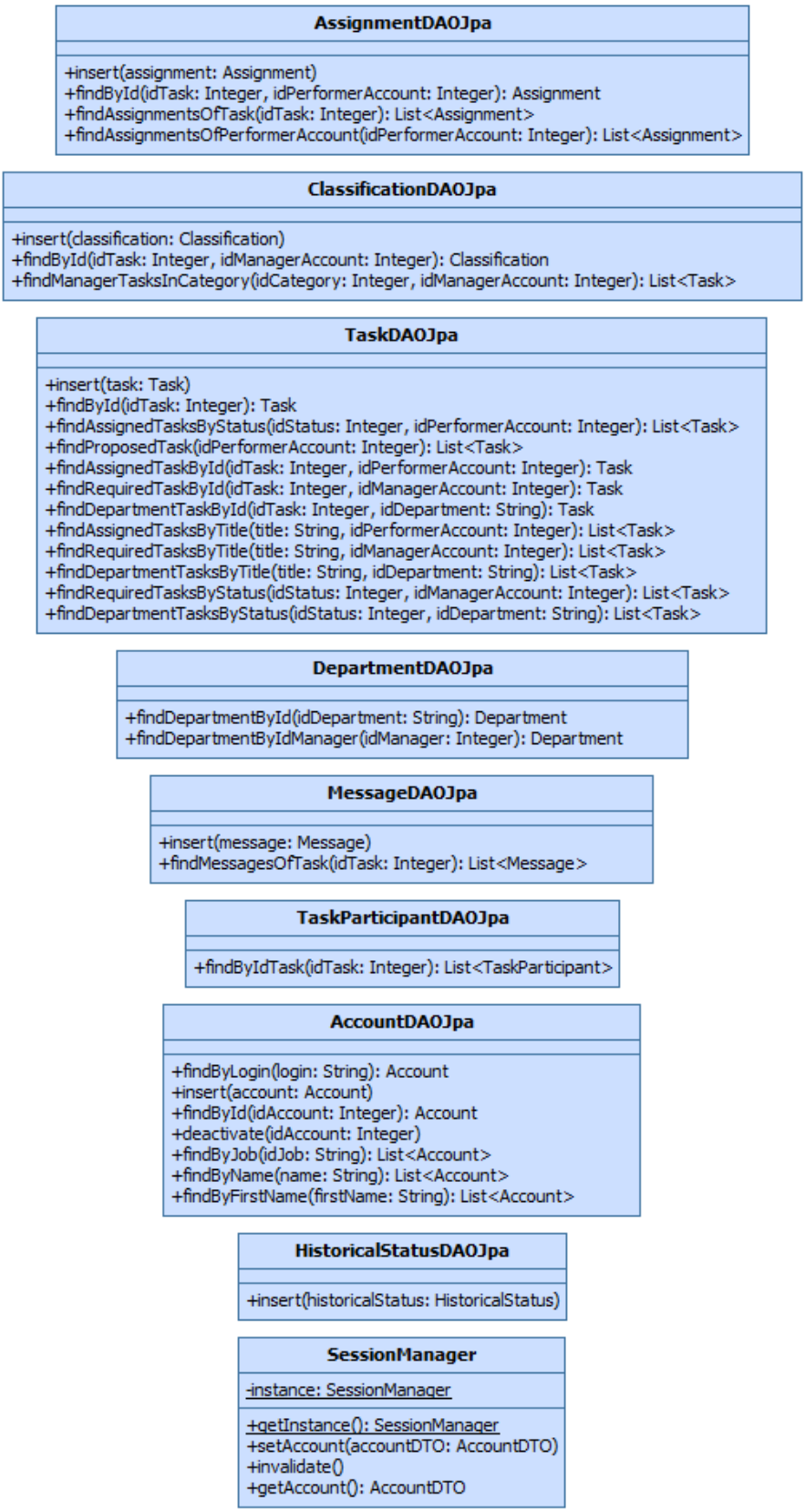

Figure 7.32, *Regroupement des classes de stéréotype <<lifecycle>>.*

7.4.7 Regroupement des classes de stéréotype <<entity>>

La figure *7.33* présente le regroupement des classes de stéréotype <<entity>>. Elle permet de percevoir l'utilisation des EJB puisqu'elle indique les relations bidirectionnelles entre entités ainsi que les classes des identifiants composites.

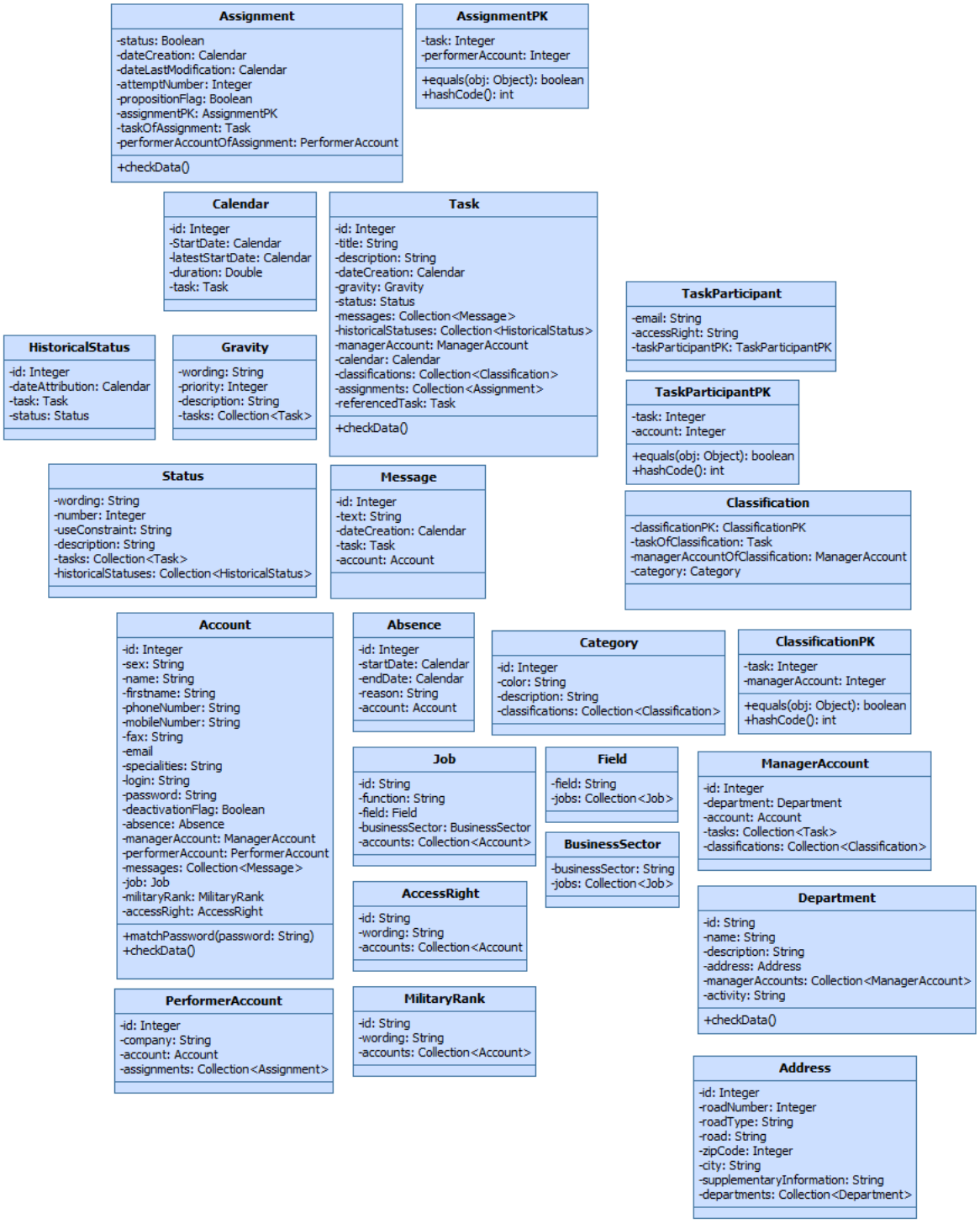

Figure 7.33, *Regroupement des classes de stéréotype <<entity>>*.

Il est important de préciser que l'application comporte, pour chaque classe de stéréotype <<entity>>, une classe DTO.

Chapitre 8. Réalisation

8.1 Environnement de développement

8.1.1 Eclipse

Eclipse est l'environnement de développement (IDE) qui a été employé pour programmer l'application WebTask. C'est un logiciel qui est gratuit, facile d'utilisation et puissant. En proposant différentes vues, il simplifie le développement avec la technologie JSF. Les vues les plus intéressantes à cet effet concernent la zone d'édition, l'explorateur de projets, l'aperçu arborescent d'une page web et l'affichage des propriétés.

8.1.1.1 La vue *Zone d'Edition*

La zone d'édition est la partie principale d'Eclipse. Elle présente, à un utilisateur, le contenu des fichiers mentionnés dans l'explorateur de projets. Cette zone est particulièrement adaptée à la modification et à la création des pages JSP (figure *8.1*). En effet, elle permet de les visualiser en mode *design* ou en mode *preview*. Le mode *design* montre le code source et comporte une palette d'outils autorisant le glissé-déposé de composants JSP et JSF. Quant au mode *preview*, il se charge de la prévisualisation.

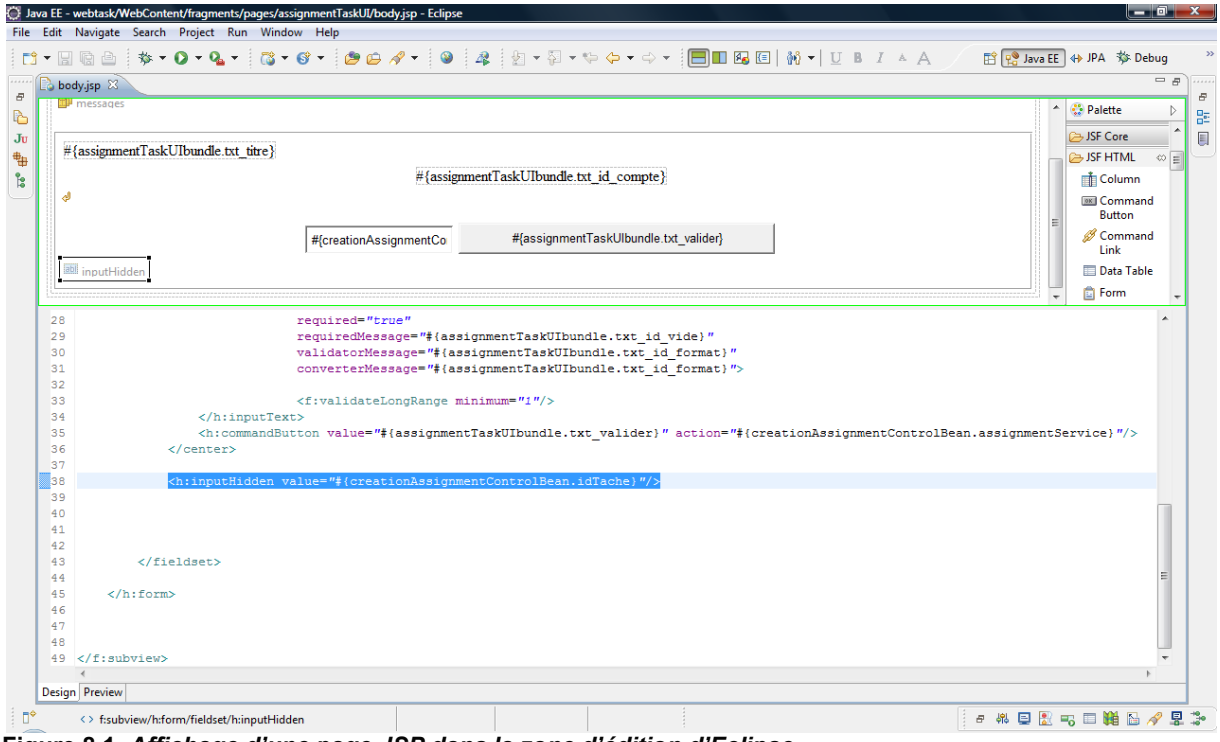

Figure 8.1, *Affichage d'une page JSP dans la zone d'édition d'Eclipse.*

Enfin, la zone d'édition facilite la modification des fichiers de configuration JSF. Elle fournit, en effet, différents formulaires de saisie qui permettent de déclarer des beans managés, de définir des règles de navigation, de préciser des validateurs etc.

8.1.1.2 La vue *Exploration de Projets*

Cette vue offre la possibilité de naviguer au sein des différents projets et d'accéder à leurs fichiers (JSP, classes Java, fichiers XML etc.).

8.1.1.3 La vue *Outline*

Cette vue (figure *8.2*) présente, sous la forme d'une arborescence, l'ensemble des composants de la page JSP en cours d'édition.

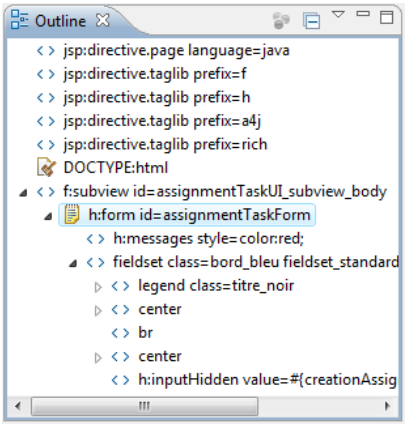

Figure 8.2, *La vue « Outline » d'Eclipse*.

8.1.1.4 La vue *Properties*

Cette vue liste les propriétés de l'élément sélectionné dans l'environnement de développement. Par exemple, lorsqu'un utilisateur clique sur un fichier ou un répertoire, elle affiche les diverses informations qui le concerne à l'aide d'une grille nommée *Info*.

Pour les composants JSF, la vue *Properties* présente une grille de saisie nommée *Attributes* (figure *8.3*). Cette grille permet de préciser les valeurs à attribuer aux différentes propriétés du composant sélectionné.

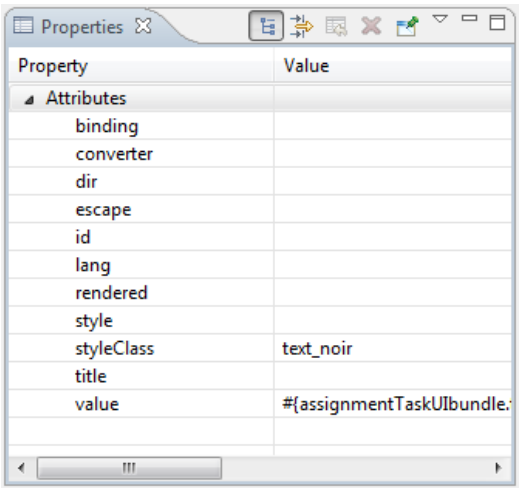

Figure 8.3, *Grille « Attributes » affichée par la vue « Properties » d'Eclipse*.

8.2 Gestion du scope *session*

La gestion du scope *session* est déléguée à la classe *SessionManager* qui implémente le pattern *Singleton* (cf. *5.3.1.1 Le pattern Singleton*). Le fonctionnement de la plupart des méthodes de cette classe repose sur deux étapes. La première étape consiste à obtenir, sous la forme d'un objet *javax.servlet.http.HttpSession*, la session courante associée à l'utilisateur effectuant une requête. Dans la seconde étape, une action est réalisée sur la session précédemment obtenue. Cette action se matérialise par l'appel d'une méthode de l'objet *HttpSession.* Elle peut consister, entre autre, à :

- Stocker des informations dans la session, à l'aide de la méthode *setAttribute(String name, Object value)* de l'objet *HttpSession*.
  Exemple : l'application WebTask stocke, dans la session, les informations du compte de l'utilisateur connecté.
- Retrouver des informations dans la session, à l'aide de la méthode *getAttribute(String name)* de l'objet *HttpSession*.
  Exemple : l'application WebTask retrouve, dans la session, les informations du compte de l'utilisateur connecté.
- Invalider la session à l'aide de la méthode *invalidate()* de l'objet *HttpSession*.
  Exemple : l'application WebTask invalide la session lorsque l'utilisateur se déconnecte.

Le principal intérêt d'utiliser une classe dédiée à la gestion du scope *session* est de pouvoir factoriser le code.

8.3 Gestion des exceptions

Une exception est générée par l'interpréteur Java lorsqu'un événement anormal ou inattendu survient. Les exceptions dérivent obligatoirement de l'une des classes suivantes :

- *java.lang.Error* qui concerne les erreurs graves conduisant à l'arrêt du programme.
- *java.lang.Exception* qui concerne les événements pouvant être traités sans provoquer l'arrêt du programme.

Les classes *Error* et *Exception* sont des sous-classes de la classe *java.lang.Throwable*.

De nombreuses exceptions prédéfinies sont fournies par Java. Cependant, l'application WebTask utilise également des exceptions qui lui sont propres. La lisibilité et la compréhension du code s'en trouve améliorées. Ces exceptions personnalisées sont soit de type *fonctionnel* soit de type *technique*. Elles sont contenues par les artefacts *webtask-common.jar* (cf. 5.6.1.3 *L'archive webtask-common.jar*) et *client-webtask-service.jar* (cf. *5.6.1.4 L'archive client-webtask-service.jar*). Ce dernier artéfact, contrairement au premier, possède uniquement des

exceptions personnalisées de type *technique*. Elles permettent aux objets *Delegate* de convertir les exceptions *réseau* des EJB en exceptions de niveau application.

La figure *8.4* présente la hiérarchie des classes d'exception spécifiques à l'application WebTask :

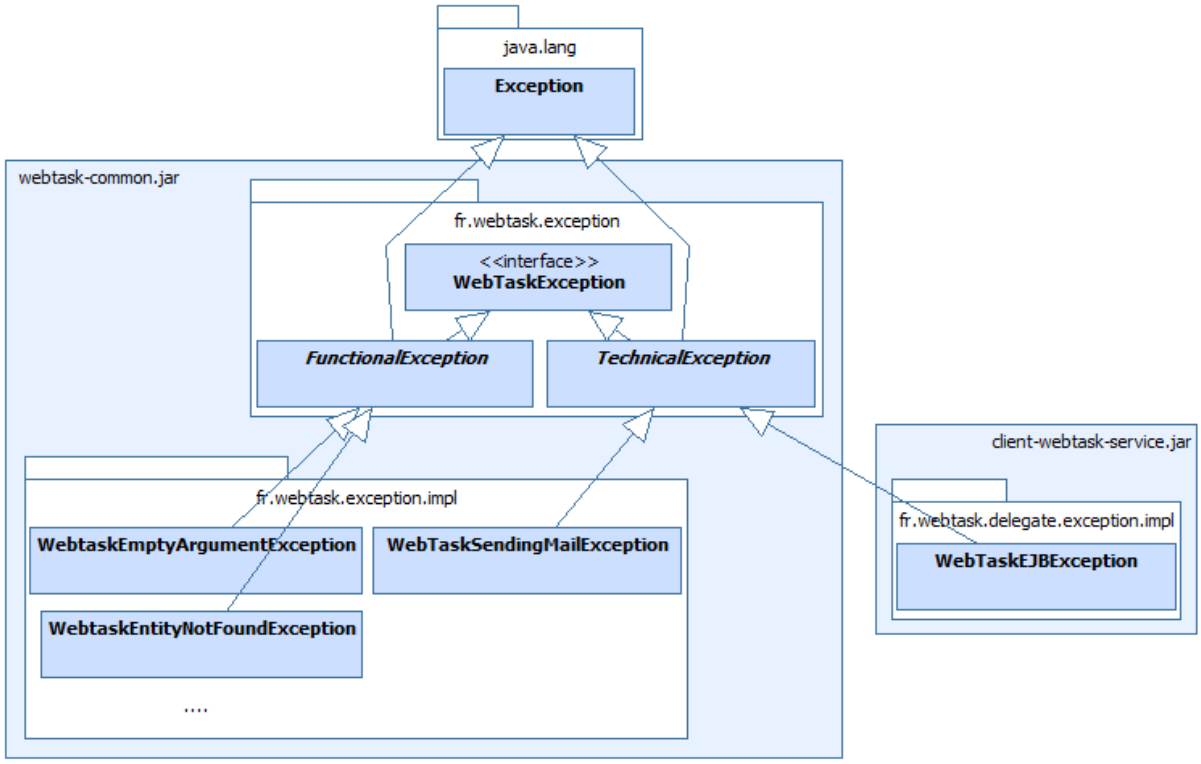

Figure 8.4, *Hiérarchie des classes d'exception spécifiques à l'application WebTask*.

8.4 Accès aux fichiers de propriétés

Les fichiers de propriétés sont des fichiers *texte* dont le nom possède l'extension *.properties*. Ils se placent dans des packages et contiennent des lignes constituées d'un identifiant, couramment appelé *clé*, puis d'une valeur.

Les fichiers de propriétés de l'application WebTask sont utilisés par les pages JSP, les beans managés et les services distants. Ils permettent de stocker, entre autre, des messages d'erreur, des messages à caractère informatif, des labels et des paramètres de configuration.

Deux techniques sont employées par l'application WebTask pour accéder aux fichiers de propriétés. Elles se basent toutes les deux sur l'utilisation de la classe *java.util.ResourceBundle* dont les instances représentent les couples clé/valeur d'un fichier properties. Voici ces techniques :

- Les pages JSP ont recours à la première technique qui correspond au mécanisme standard offert par JSF. Il consiste à déclarer, dans un des fichiers de configuration JSF, un ResourceBundle associé à un fichier de propriétés. Cette déclaration indique un nom de base et un identifiant unique employé dans les pages web.

- Les beans managés et les services distants ont recours à la seconde technique : des classes qui *mappent les* fichiers de propriétés. Lors de leur instanciation, elles obtiennent un ResourceBundle associé au fichier de propriétés qu'elles mappent. Ce ResourceBundle permet à chaque getter qu'elles possèdent de récupérer la valeur d'une propriété du fichier mappé.

8.5 Les principaux fichiers de configuration de l'application

8.5.1 Le descripteur de déploiement

Le descripteur de déploiement est un fichier XML portant le nom de *web.xml.* Il est placé dans le répertoire WEB-INF de l'application web (cf. *5.6.1.1 L'archive webtask.war*) et contient des informations de configuration. Le fichier *web.xml* permet à l'application WebTask de définir, entre autre, des servlets, des filtres, des listeners, le timeout des sessions etc.

8.5.2 Le fichier *tiles-defs.xml*

Le fichier *tiles-defs.xml* permet de déclarer les modèles de pages (cf. *6.2.4 Templating*) qui sont utilisés par le framework Tiles pour construire les pages JSP. Il est placé dans le répertoire WEB-INF de l'application web (cf. *5.6.1.1 L'archive webtask.war*). Voici une partie de son contenu :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE tiles-definitions PUBLIC
      "-//Apache Software Foundation//DTD Tiles Configuration 2.1//EN"
      "http://tiles.apache.org/dtds/tiles-config_2_1.dtd">
<tiles-definitions>
        <definition name="template1" template="/templates/template1.jsp">
                <put-attribute name="title" value="Webtask" />
                <put-attribute name="header" value="/fragments/templates/template1/header.jsp" />
                <put-attribute name="menu" value="/fragments/templates/template1/menu.jsp" />
                <put-attribute name="footer" value="/fragments/templates/template1/footer.jsp" />
        </definition>
        <definition name="creationAccountUI_def" extends="template1">
                <put-attribute name="body" value="/fragments/pages/creationAccountUI/body.jsp" />
        </definition>
        ……
</tiles-definitions>
```

La configuration présentée ci-dessus emploie différentes balises dont les fonctions sont décrites ci-après :

- La balise *tiles-definitions* : englobe les déclarations de templates.
- La balise *definition* : déclaration d'un template. L'attribut *extends* permet à une définition d'en étendre une autre.
- La balise *put-attribute* : détermine un attribut qui peut avoir comme valeur soit un chemin vers une page JSP soit une chaîne de caractères.

8.5.3 Le fichier *persistence.xml*

Le rôle du fichier *persistence.xml* est de paramétrer l'unité de persistance du système (cf. *6.3.2 Persistance des données*). Ce fichier est placé dans le répertoire META-INF de l'application EJB (cf. 5.6.1.2 *L'archive webtask-business.jar*). Voici son contenu :

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
        version="2.0">
        <persistence-unit name="webtaskPersistenceUnit" transaction-type="JTA">
                <provider>org.hibernate.ejb.HibernatePersistence</provider>
                <jta-data-source>java:/jdbc/WebtaskDataSource</jta-data-source>
                <properties>
                        <property
                              name="hibernate.dialect" value="org.hibernate.dialect.MySQLInnoDBDialect" />
                        <property name="hibernate.hbm2ddl.auto" value="validate" />
                </properties>
        </persistence-unit>
</persistence>
```

La configuration présentée ci-dessus emploie différentes balises dont les fonctions sont décrites ci-après :

- La balise *persistence-unit* : déclare une unité de persistance.
- La balise *provider* : détermine la classe d'implémentation du fournisseur de persistance utilisé dans l'application.
- La balise *jta-data-source* : détermine le nom JNDI du data source transactionnel à utiliser. Celui-ci est d'ailleurs paramétré dans le fichier *mysql-ds.xml* (cf. *8.5.4 Le fichier mysql-ds.xml*).
- La balise *properties* : détermine les attributs de configuration du fournisseur de persistance.

8.5.4 Le fichier *mysql-ds.xml*

Le serveur d'application JBoss permet de configurer des data sources en précisant leur paramétrage dans des fichiers suffixés avec *–ds.*xml. Un data source est une fabrique de connexions de type *java.sql.Connection*. Les fichiers suffixés avec *–ds.xml* doivent être placés dans le répertoire de déploiement de JBoss pour que ce dernier puisse les prendre en compte.

Ainsi, pour l'application WebTask, le fichier *mysql-ds.xml* configure un data source qui permet d'obtenir des connexions à la base de données MySQL (cf. *6.4.1 Base de données relationnelle*). Il est placé dans le répertoire de déploiement de JBoss, sur le serveur EJB. Son contenu est précisé sur la page suivante.

Voici le contenu du fichier *mysql-ds.xml* :

```
<datasources>
	<local-tx-datasource>
		<jndi-name>jdbc/WebtaskDataSource</jndi-name>
		<connection-url>jdbc:mysql://localhost:3306/webtaskdb</connection-url>
		<driver-class>com.mysql.jdbc.Driver</driver-class>
		<user-name>webtask</user-name>
		<password>webtask</password>
		<use-java-context>true</use-java-context>
		<min-pool-size>5</min-pool-size>
		<max-pool-size>20</max-pool-size>
		<metadata>
			<type-mapping>mySQL</type-mapping>
		</metadata>
	</local-tx-datasource>
</datasources>
```

La configuration présentée ci-dessus emploie différentes balises dont les fonctions sont décrites ci-après :

- La balise *local-tx-datasource* : déclare un data source avec support transactionnel.
- La balise *jndi-name* : détermine le nom JNDI avec lequel le data source sera lié.
- La balise *connection-url* : détermine l'URL de connexion.
- La balise *driver-class* : détermine le nom complet de la classe driver JDBC.
- La balise *user-name* : détermine l'identifiant de connexion à la base de données.
- La balise *password* : détermine le mot de passe de connexion à la base de données.
- La balise *use-java-context* : indique si le data source est lié au contexte *java:*.
- La balise *min-pool-size* : détermine le nombre minimum de connexions que doit contenir le pool de connexions.

  La balise *max-pool-size* : détermine le nombre maximum de connexions dans le pool de connexions.
- La balise *type-mapping* : détermine le type de mapping.

8.5.5 Le fichier *mailConfiguration.properties*

Le fichier *mailConfiguration.properties* contient les paramètres qui configurent l'envoi des emails. Ces différents paramètres sont, entre autre, le login et le mot de passe du compte GMAIL, l'activation/désactivation des envois de mail etc. Le fichier *mailConfiguration.properties* est placé dans le package *fr.webtask.service.storetext.configuration* de l'application EJB (cf. 5.6.1.2 *L'archive webtask-business.jar*).

8.5.6 Les fichiers *jboss-log4j.xml*

Deux fichiers *jboss-log4j.xml* sont utilisés :

- Le premier configure la journalisation de l'application web. Il est placé sur le tiers web dans le répertoire *conf* de JBoss (cf. *6.2.1 Serveur d'applications* / cf. *6.2.5 Journalisation*).
- Le second configure la journalisation de l'application EJB. Il est placé sur le tiers Logique Business dans le répertoire *conf* de JBoss (cf. *6.3.3 Conteneur EJB* / cf. *6.3.4 Journalisation*).

8.6 Bonnes pratiques de programmation

Le développement de l'application WebTask s'est appuyé sur les bonnes pratiques de programmation qui permettent de renforcer la qualité du code. Voici quelques unes des bonnes pratiques qui ont été appliquées :

- Commenter son code.
- Indenter son code.
- Pas de code en commentaire.
- Tout élément déclaré dans le code doit être utilisé.
- Respecter une taille de ligne maximale.
- Déclarer une seule variable par ligne.
- Mettre une seule instruction par ligne.
- Découper son code en fonctions.
- Une méthode doit avoir une sortie nominale et une seule.
- Chaque cas dans une instruction *switch* doit être terminé par une instruction *break*.
- Pas plus d'une instruction break dans une boucle.
- Utiliser la classe *java.lang.StringBuilder* plutôt que la classe *java.lang.String* lors de la concaténation de chaînes dans une boucle.
- Etc.

Conclusion : un bilan vraiment positif

Le projet WebTask, issu de l'UE GLG 204 du CNAM, a été un défi motivant et enrichissant. Il m'a offert la possibilité de concevoir et de programmer entièrement une application. Cette situation est plutôt rare dans le contexte professionnel puisque les projets nécessitent souvent la mise en place d'une équipe plus ou moins importante. Ainsi les différents membres d'une équipe se voient alors attribuer chacun une partie du travail à effectuer.

Concevoir et programmer entièrement cette application m'a permis de prendre en charge les différentes phases d'un projet informatique. La phase d'expression des besoins a été particulièrement intéressante car, dans mon métier de programmeur, je ne suis pas amené à m'en soucier. Approfondir les besoins de l'étude de faisabilité, que j'ai réalisée quelques années plus tôt au CETIMA, et déterminer de nouveaux besoins a été un peu laborieux. Toutefois, avec de la persévérance, je pense être parvenu à obtenir un fonctionnel cohérent et répondant convenablement à des besoins de gestion de tâches.

D'un point de vue compétence technique, ce projet m'a permis d'améliorer ma pratique de certaines API. Par exemple, dans le cadre professionnel, j'ai déjà employé les EJB Entity. Cependant, sans la réalisation de l'application WebTask, je n'aurais jamais eu l'occasion de les utiliser pour mapper entièrement une base de données. Ce projet m'a également permis d'utiliser d'autres API pour la première fois. Ce fut, entre autre, le cas pour l'API FreeMarker.

Enfin, il est important de préciser que je ne considère pas ce mémoire comme une finalité en soi. J'envisage de distribuer l'application et j'ai l'intention de continuer à l'enrichir en ajoutant de nouvelles fonctionnalités comme, par exemple, l'ajout de pièces-jointes à une tâche.

Annexe 1. Terminologie métier

L'objectif de la terminologie métier est de recenser les entités qui jouent un rôle dans le fonctionnel de l'application WebTask. Chaque entité est composée d'une liste de propriétés dont le format et la longueur sont précisés par la terminologie. Les formats employés sont les suivants : alphabétique (A), alphanumérique (AN), numérique (N), date (D), booléen (B).

A.1.1 Compte

Un compte permet à un utilisateur d'une part de se connecter et d'autre part d'accéder aux fonctionnalités de l'application qu'il peut utiliser. Il existe trois types de comptes : compte d'exécutant, compte de chef de projet et compte de chef de département. Il est important de préciser que l'application ne permet pas de créer des comptes de chef de département mais elle en fournit dix configurables. Le tableau ci-dessous précise les propriétés d'un compte :

Tableau A.1.1, *Propriétés d'un compte.*

Propriété	Format	Longueur
Identifiant	N	/
Sexe (*m=masculin*, *f=féminin*)	A	1
Nom	A	50
Prénom	A	50
Téléphone fixe (professionnel)	AN	20
Téléphone portable (professionnel)	AN	20
Fax	AN	20
Email	AN	50
Spécialités (compétences pointues dans un métier)	AN	300
Société (lorsqu'un exécutant est prestataire, le nom de son entreprise peut être précisé)	AN	50
Login	AN	50
Mot de passe	AN	50
Drapeau de désactivation (indique si le compte est désactivé ou non)	B	/

A.1.2 Poste d'un compte

Un compte est caractérisé par un poste, c'est-à-dire une profession. Le tableau ci-dessous précise les propriétés d'un poste :

Tableau A.1.2, *Propriétés d'un poste de compte.*

Propriété	Format	Longueur
Identifiant (*emp_1*, *emp_2*, *emp_3* etc.)	AN	8
Fonction (ex : développeur informatique)	AN	50
Domaine (ex : informatique, agriculture)	A	50
Secteur d'activité (*primaire*, *secondaire*, *tertiaire*)	A	10

A.1.3 Absence d'un compte

Une absence peut être mentionnée sur un compte. Elle correspond à une période pendant laquelle le propriétaire d'un compte n'est pas présent (vacances, congés maternité…). Le tableau ci-dessous précise les propriétés de l'absence d'un compte :

Tableau A.1.3, *Propriétés d'une absence de compte.*

Propriété	Format	Longueur
Identifiant	N	/
Date de début	D	/
Date de fin	D	/
Motif	AN	300

A.1.4 Droit d'accès aux fonctionnalités d'un compte

Chaque compte dispose d'un droit d'accès, c'est-à-dire un droit qui détermine quelles sont les fonctionnalités autorisées à un compte. Le tableau ci-dessous précise les propriétés d'un droit d'accès :

Tableau A.1.4, *Propriétés d'un droit d'accès.*

Propriété	Format	Longueur
Identifiant (*cdd*, *cdp*, *exec*)	A	4
Libellé (*chef de département*, *chef de projet*, *exécutant*)	A	25

A.1.5 Grade d'un compte

Un compte peut être caractérisé par un grade lorsque son propriétaire est un militaire. Le tableau ci-dessous précise les propriétés du grade d'un compte :

Tableau A.1.5, *Propriétés d'un grade de compte.*

Propriété	Format	Longueur
Identifiant (*mr_1*, *mr_2*, *mr_3 etc.*)	AN	6
Libellé (*Adjudant*, *Lieutenant*, *Major* etc.)	A	50

A.1.6 Tâche

Une tâche est un travail à faire sous des conditions et dans un temps fixés. Elle peut référencer une autre tâche afin d'établir un lien sensé et cohérent avec celle-ci. Le tableau ci-dessous précise les propriétés d'une tâche :

Tableau A.1.6, *Propriétés d'une tâche.*

Propriété	Format	Longueur
Identifiant	N	/
Titre	AN	50
Description	AN	1500
Date de création	D	/

A.1.7 Niveau d'urgence d'une tâche

Un niveau d'urgence indique le degré d'importance d'une tâche. Le tableau ci-dessous précise les propriétés du niveau d'urgence d'une tâche:

Tableau A.1.7, *Propriétés du niveau d'urgence d'une tâche.*

Propriété	Format	Longueur
Priorité (*1=critique*, *2=majeur*, *3=mineur*)	N	/
Libellé (*critique*, *majeur*, *mineur*)	A	8
Description	AN	300

A.1.8 Calendrier d'une tâche

Un calendrier indique les délais pour réaliser une tâche. Le tableau ci-dessous précise les propriétés du calendrier d'une tâche:

Tableau A.1.8, *Propriétés d'un calendrier de tâche.*

Propriété	Format	Longueur
Identifiant	N	/
Date de début au plus tôt	D	/
Date de début au plus tard	D	/
Durée de réalisation	N	/

A.1.9 Statut d'une tâche

Le statut permet d'indiquer l'état d'avancement d'une tâche. Le tableau ci-dessous précise les propriétés du statut d'une tâche:

Tableau A.1.9, *Propriétés d'un statut de tâche.*

Propriété	Format	Longueur
Libellé (*fermé*, *créé*, *attribué*, *à réaliser*, *terminé*, *validé*, *rejeté*, *à revoir*)	A	10
Numéro (*0=fermé*, *1=crée*, *2=attribué*, *3=à réaliser*, *4=terminé*, *5=validé*, *6=rejeté*, *7=à revoir*)	N	/
Contrainte d'utilisation (indique les numéros de statuts séparés par des «,» à partir desquels le statut en question peut être utilisé)	AN	15
Description	AN	300

La figure ci-dessous indique l'ordre de succession des statuts d'une tâche:

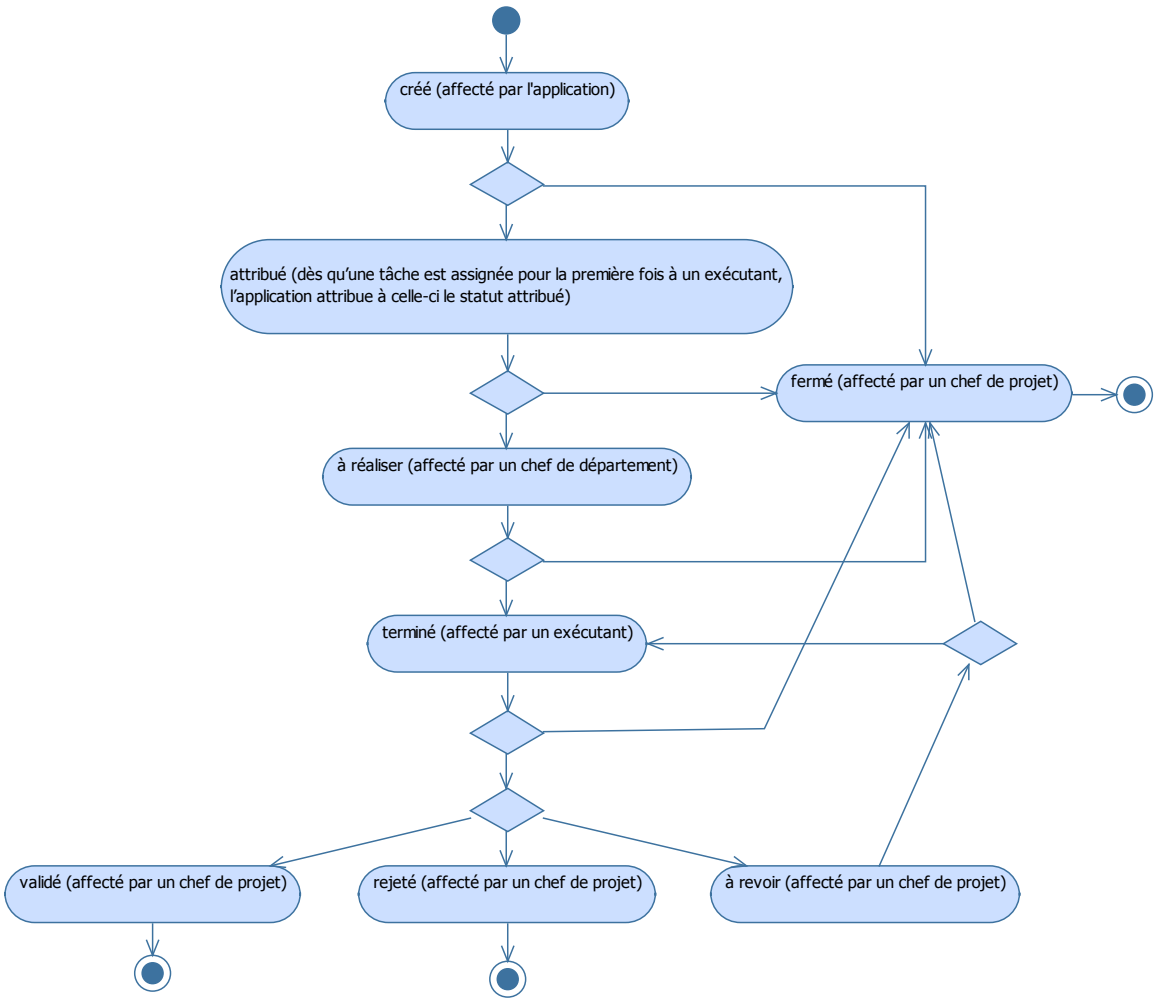

Figure A.1.1, *Succession des statuts d'une tâche.*

A.1.10 Historique des statuts des tâches

L'historique des statuts conserve en mémoire les statuts qui se sont succédés dans la vie de chaque tâche. Le tableau ci-dessous précise les propriétés de l'historique des statuts des tâches:

Tableau A.1.10, *Propriétés* de l'historique des statuts des tâches.

Propriété	Format	Longueur
Identifiant	N	/
Date d'attribution d'un statut	D	/

A.1.11 Message d'une tâche

Un message se présente sous une forme textuelle. En déposant un message, un acteur dialogue avec les autres intervenants d'une tâche. Le tableau ci-dessous précise les propriétés d'un message d'une tâche:

Tableau A.1.11, *Propriétés* d'un message de tâche.

Propriété	Format	Longueur
Identifiant	N	/
Text	AN	500
Date de création	D	/

A.1.12 Attribution d'une tâche

Une attribution représente l'assignation d'une tâche à un exécutant. Le tableau ci-dessous précise les propriétés de l'attribution d'une tâche:

Tableau A.1.12, *Propriétés* d'une attribution de tâche.

Propriété	Format	Longueur
Statut (il précise si l'attribution a été acceptée ou refusée par un exécutant)	B	/
Date de création	D	/
date de dernière modification	D	/
Nombre de tentatives d'attribution (un chef de département peut proposer à nouveau une tâche à un exécutant qui l'a précédemment refusée)	N	/
Flag de proposition (indique si l'attribution doit être proposée à l'exécutant concerné)	B	/

A.1.13 Catégorie d'une tâche

Les catégories permettent à un utilisateur de classer les tâches qu'il peut consulter. Elles sont représentées par des couleurs. Le tableau ci-dessous précise les propriétés d'une catégorie :

Tableau A.1.13, *Propriétés* d'une catégorie.

Propriété	Format	Longueur
Identifiant	N	/
Libellé de couleur	A	50
Description	AN	300

A.1.14 Département

Un département est une subdivision des services au sein d'une entreprise. L'application WebTask fournit dix départements, chacun correspondant à l'un des dix comptes de chef de département (cf. *A.1.1 Compte*) disponibles. Le tableau ci-dessous précise les propriétés d'un département :

Tableau A.1.14, *Propriétés* d'un département.

Propriété	Format	Longueur
Identifiant (*dp_1*, *dp_2*, *dp_3* etc.)	AN	5
Nom (Département 1, Département 2, Département 3 etc.)	AN	14
Description	AN	300
Activité (ex :recherche et développement)	AN	50

A.1.15 Adresse d'un département

Le tableau ci-dessous précise les propriétés d'une adresse :

Tableau A.1.15, *Propriétés* d'une adresse.

Propriété	Format	Longueur
Identifiant	N	/
Numéro de voie	N	/
Type de voie (*rue*, *avenue* etc.)	A	15
Nom de voie	AN	50
Code postal	N	/
Ville	AN	50
Complément d'adresse	AN	300

Annexe 2. Informations complémentaires concernant la phase d'expression des besoins

Cette annexe présente, pour chaque cas d'utilisation, des informations textuelles et graphiques complémentaires qui sont généralement les suivantes : pré-conditions, scénario nominal, scénarios alternatifs, scénarios d'erreur, post-conditions et diagramme d'activité.

A.2.1 Le sous-système *Opérations de base*

A.2.1.1 Le cas d'utilisation *Se connecter*

Scénario nominal

1 - L'acteur saisit son login et son mot de passe et s'authentifie.
2 - Le système vérifie les données (syntaxe, format, champs remplis).
3 - Le système vérifie l'existence du compte à l'aide de la source de données.
4 - Le système affiche un message de bienvenue à l'acteur.

Scénarios alternatifs

A1 : Vérification des données incorrecte.
L'enchainement A1 démarre au point 2 du scénario nominal.

3 - Le système indique à l'acteur que les données ne sont pas conformes aux règles imposées (syntaxe, format, champs remplis).
Le scénario nominal reprend au point 1.

A2 : Le login et/ou le mot de passe ne sont pas valides.
L'enchainement A2 démarre au point 3.

4 - Le système avertit l'acteur que son login et/ou son mot de passe ne sont pas valides.
Le scénario nominal reprend au point 1.

Scénarios d'erreur

E1 : La source de données n'est pas accessible.
L'enchainement E1 démarre au point 3.

4 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

L'acteur est authentifié. Il peut atteindre les fonctionnalités de l'application qu'il est autorisé à utiliser.

Diagramme d'activité

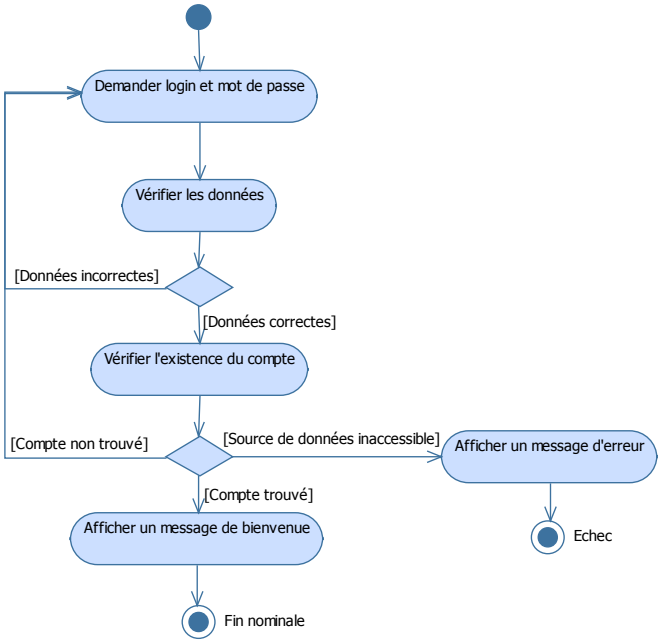

Figure A.2.1, *Diagramme d'activité du cas d'utilisation « Se connecter ».*

A.2.1.2 Le cas d'utilisation *Se déconnecter*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*).

Scénario nominal

1 - L'acteur réalise une demande de déconnexion.
2 - Le système invalide la session.
3 - Le système affiche un message confirmant la déconnexion.

Post-conditions

L'acteur ne peut plus atteindre les fonctionnalités de l'application qu'il était autorisé à utiliser.

Diagramme d'activité

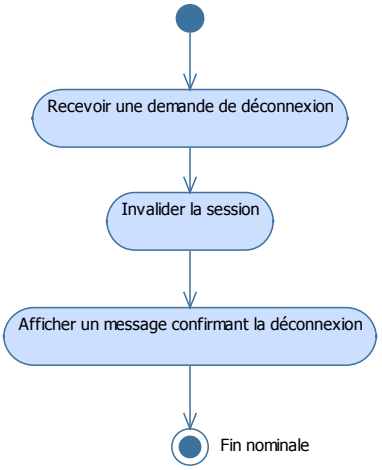

Figure A.2.2, *Diagramme d'activité du cas d'utilisation « Se déconnecter ».*

A.2.2 Le sous-système *Gestion des tâches*

A.2.2.1 Le cas d'utilisation *Déposer une tâche*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*).

Scénario nominal

1 - L'acteur renseigne les informations d'une tâche et la dépose.
2 - Le système vérifie les données (syntaxe, format, champs remplis).
3 - Le système enregistre la tâche dans la source de données.
4 - Le système envoie un mail au chef de département à l'aide de la messagerie d'entreprise.
5 - Le système affiche un message confirmant le dépôt de la tâche.

Scénarios alternatifs

A1 : *Vérification des données incorrecte*.
L'enchainement A1 démarre au point 2 du scénario nominal.

3 - Le système indique à l'acteur que les données ne sont pas conformes aux règles imposées (syntaxe, format, champs remplis).
Le scénario nominal reprend au point 1.

Scénarios d'erreur

E1 : *La source de données n'est pas accessible*.
L'enchainement E1 démarre au point 3.

4 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

E2 : *La messagerie d'entreprise n'est pas accessible*.
L'enchainement E2 démarre au point 4 du scénario nominal.

5 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Une tâche supplémentaire a été déposée et le chef de département en a été averti par l'envoi d'un e-mail.

Diagramme d'activité

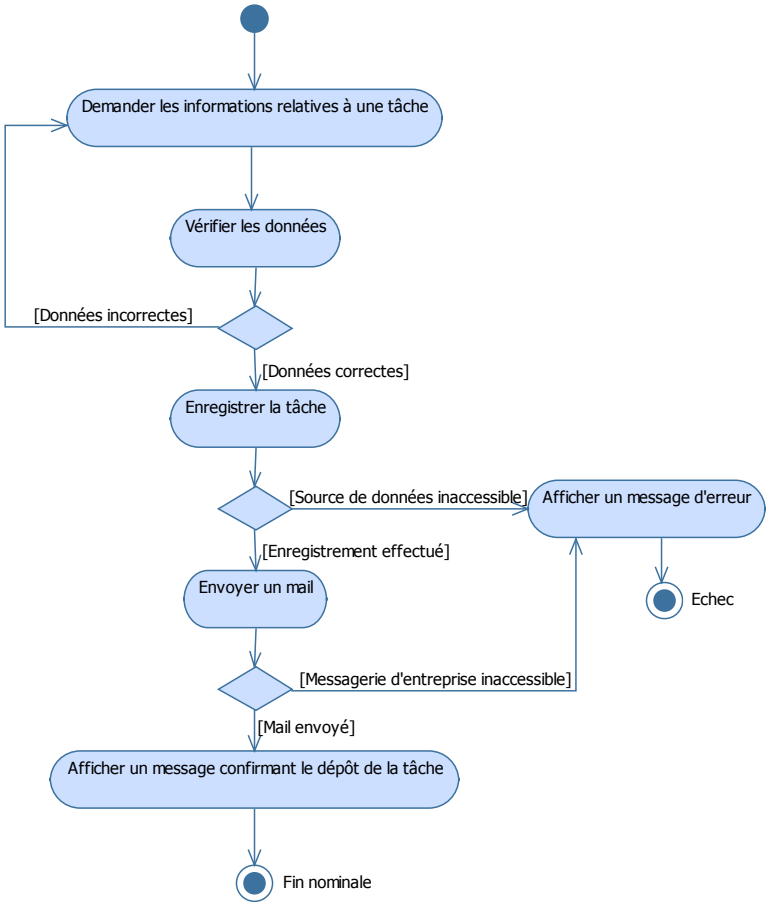

Figure A.2.3, *Diagramme d'activité du cas d'utilisation « Déposer une tâche ».*

A.2.2.2 Le cas d'utilisation *Rechercher des tâches*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*).

Scénario nominal

1 - L'acteur précise le critère, la valeur à rechercher, puis lance la recherche.
2 - Le système vérifie les données (syntaxe, format, champs remplis).
3 - Le système cherche les tâches dans la source de données.
4 - Le système affiche les tâches.

Scénarios alternatifs

A1 : *Vérification des données incorrecte*.
L'enchainement A1 démarre au point 2 du scénario nominal.

3 - Le système indique à l'acteur que les données ne sont pas conformes aux règles imposées (syntaxe, format, champs remplis).
Le scénario nominal reprend au point 1.

Scénarios d'erreur

E1 : *La source de données n'est pas accessible*.
L'enchainement E1 démarre au point 3.

4 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Les tâches sont affichées avec toutes les informations qui les concernent.

Diagramme d'activité

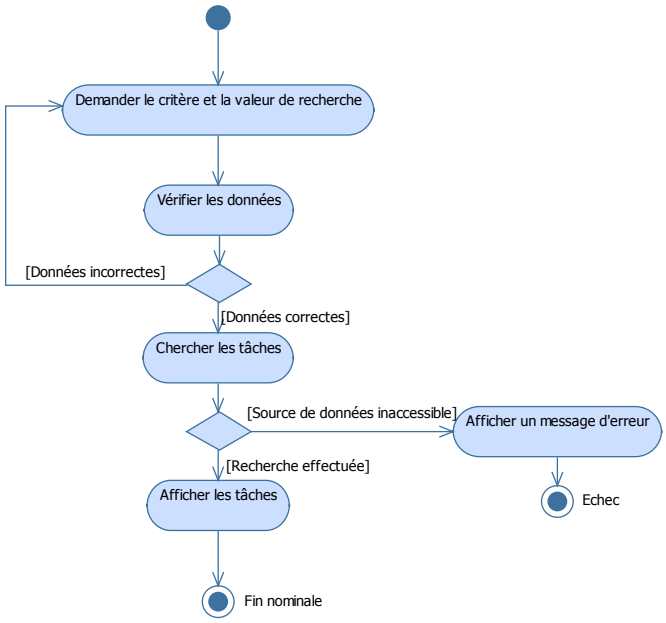

Figure A.2.4, *Diagramme d'activité du cas d'utilisation « Rechercher des tâches ».*

A.2.2.3 Le cas d'utilisation *Mettre à jour une tâche*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*) et doit avoir recherché des tâches (cf. *2.2.3.3 Le cas d'utilisation « Rechercher des tâches »*). La tâche qui sera mise à jour a été précédemment déposée par un chef de projet (cf. *2.2.3.2 Le cas d'utilisation « Déposer une tâche »*).

Scénario nominal

1 - L'acteur décide de modifier une des tâches qu'il a précédemment recherchées.
2 - Le système affiche les informations de la tâche choisie.
3 - L'acteur modifie les informations affichées.
4 - Le système vérifie les données (syntaxe, format, champs remplis).
5 - Le système enregistre les modifications dans la source de données.
6 - Le système envoie un mail aux intervenants (chef de département et/ou chef de projet et/ou exécutant(s) suivant le cas) à l'aide de la messagerie d'entreprise.
7 - Le système affiche un message confirmant la mise à jour de la tâche.

Scénarios alternatifs

A1 : *Vérification des données incorrecte*.
L'enchainement A1 démarre au point 4 du scénario nominal.

5 - Le système indique à l'acteur que les données ne sont pas conformes aux règles imposées (syntaxe, format, champs remplis).
Le scénario nominal reprend au point 3.

Scénarios d'erreur

E1 : *La source de données n'est pas accessible*.
L'enchainement E1 démarre au point 5.

6 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

E2 : *La messagerie d'entreprise n'est pas accessible*.
L'enchainement E2 démarre au point 6 du scénario nominal.

7 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Le système a mis jour une tâche.

Diagramme d'activité

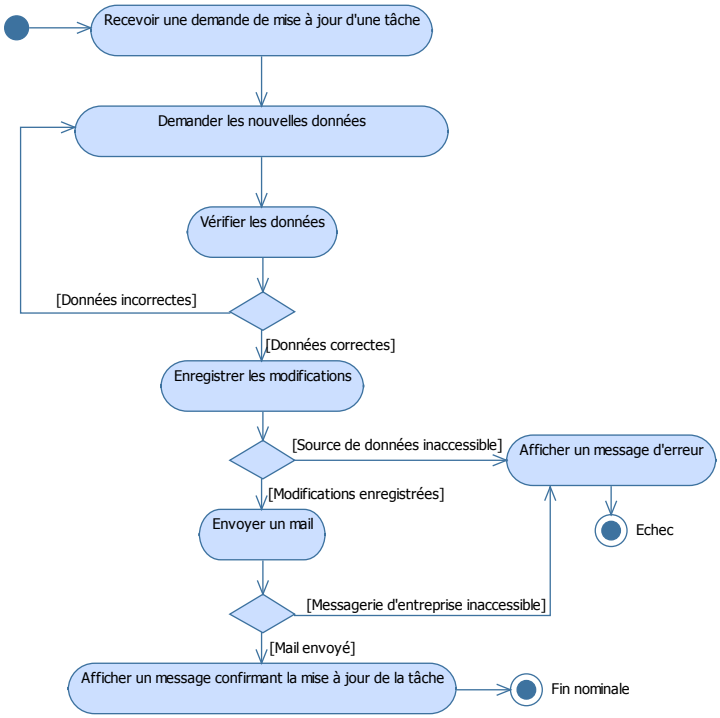

Figure A.2.5, *Diagramme d'activité du cas d'utilisation « Mettre à jour une tâche »*.

A.2.3 Le sous-système *Gestion des attributions*

A.2.3.1 Le cas d'utilisation *Attribuer une tâche*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*) et doit avoir recherché des tâches (cf. *2.2.3.3 Le cas d'utilisation « Rechercher des tâches »*). La tâche qui sera attribuée a été précédemment déposée par un chef de projet (cf. *2.2.3.2 Le cas d'utilisation « Déposer une tâche »*).

Scénario nominal

1 - Le chef de département décide d'assigner une des tâches qu'il a précédemment recherchées.
2 - Le chef de département indique un exécutant et lui propose une attribution.

3 - Le système vérifie les données (syntaxe, format, champs remplis).
4 - Le système enregistre l'attribution dans la source de données.
5 - Le système envoie un mail aux intervenants (chef de projet qui a déposé la tâche, exécutant à qui s'adresse l'attribution, exécutants ayant éventuellement déjà accepté de traiter la tâche) à l'aide de la messagerie d'entreprise.
6 - Le système affiche un message confirmant l'attribution de la tâche.

Scénarios alternatifs

A1 : *Vérification des données incorrecte*.
L'enchainement A1 démarre au point 3 du scénario nominal.

4 - Le système indique à l'acteur que les données ne sont pas conformes aux règles imposées (syntaxe, format, champs remplis).
Le scénario nominal reprend au point 2.

Scénarios d'erreur

E1 : *La source de données n'est pas accessible*.
L'enchainement E1 démarre au point 4.

5 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

E2 : *La messagerie d'entreprise n'est pas accessible*.
L'enchainement E2 démarre au point 5 du scénario nominal.

6 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Le système a enregistré une attribution de tâche.

Diagramme d'activité

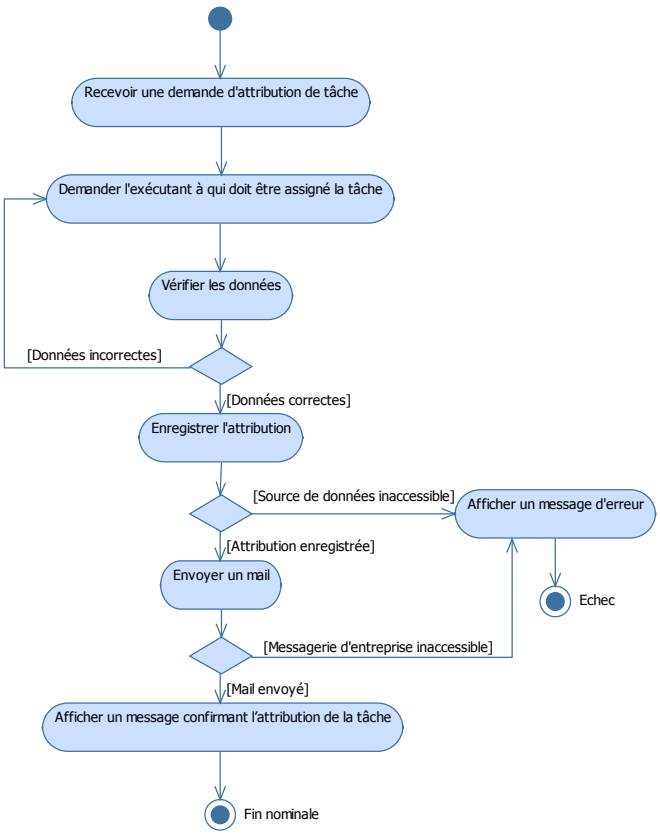

Figure A.2.6, *Diagramme d'activité du cas d'utilisation « Attribuer une tâche ».*

A.2.3.2 Le cas d'utilisation *Accepter/refuser une tâche*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*). La tâche qui sera acceptée ou refusée a été précédemment proposée par un chef de département (cf. *2.2.4.2 Le cas d'utilisation « Attribuer une tâche »*).

Scénario nominal

1 - Le système récupère, dans la source de données, les tâches qui sont proposées à l'exécutant connecté.
2 - Le système affiche les tâches proposées.
3 - L'exécutant accepte ou refuse une tâche de son choix.
4 - Le système enregistre l'acceptation ou le refus de la tâche dans la source de données.
5 - Le système envoie un mail aux autres acteurs (chef de département qui a assigné la tâche, chef de projet qui a déposé la tâche, exécutants ayant éventuellement déjà accepté de traiter la tâche) à l'aide de la messagerie d'entreprise.
6 - Le système affiche un message confirmant l'acceptation ou le refus de la tâche.

Scénarios d'erreur

E1 : *La source de données n'est pas accessible*.
L'enchainement E1 démarre au point 1.

2 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

E2 : *La source de données n'est pas accessible*.
L'enchainement E2 démarre au point 4.

5 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

E3 : *La messagerie d'entreprise n'est pas accessible*.
L'enchainement E3 démarre au point 5 du scénario nominal.

6 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Une tâche a été acceptée ou refusée par un exécutant.

Diagramme d'activité

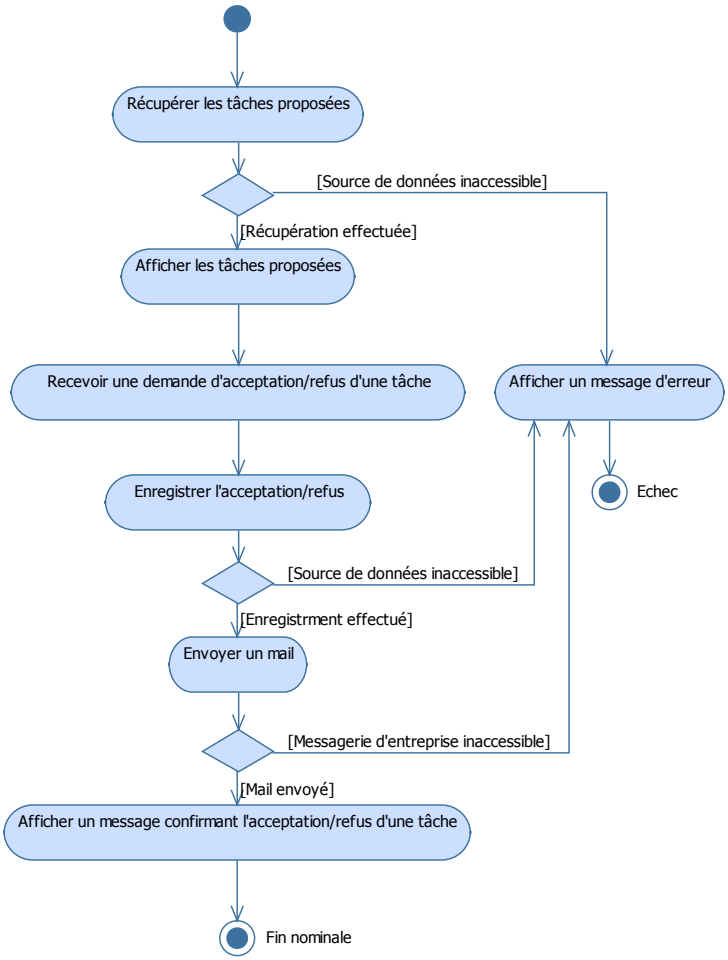

Figure A.2.7, *Diagramme d'activité du cas d'utilisation « Accepter/refuser une tâche ».*

A.2.4 Le sous-système *Gestion des Messages*

A.2.4.1 Le cas d'utilisation *Ajouter un message à une tâche*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*) et doit avoir recherché des tâches (cf. *2.2.3.3 Le cas d'utilisation « Rechercher des tâches »*). La tâche, à laquelle sera ajouté un message, a été précédemment déposée par un chef de projet (cf. *2.2.3.2 Le cas d'utilisation « Déposer une tâche »*).

Scénario nominal

1 - L'acteur décide d'ajouter un message à une des tâches précédemment recherchées.
2 - L'acteur rédige son message et l'ajoute.
3 - Le système vérifie les données (champ rempli).
4 - Le système enregistre le message dans la source de données.
5 - Le système envoie un mail aux autres intervenants (chef de département, chef de projet et/ou exécutant(s) suivant le cas) à l'aide de la messagerie d'entreprise.
6 - Le système affiche un message confirmant l'ajout.

Scénarios alternatifs

A1 : *Vérification des données incorrecte*.
L'enchainement A1 démarre au point 3 du scénario nominal.

4 - Le système indique à l'acteur que les données ne sont pas conformes aux règles imposées (champ rempli).
Le scénario nominal reprend au point 2.

Scénarios d'erreur

E1 : *La source de données n'est pas accessible*.
L'enchainement E1 démarre au point 4.

5 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

E2 : *La messagerie d'entreprise n'est pas accessible*.
L'enchainement E2 démarre au point 5 du scénario nominal.

6 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Les différents intervenants d'une tâche peuvent consulter le message ajouté.

Diagramme d'activité

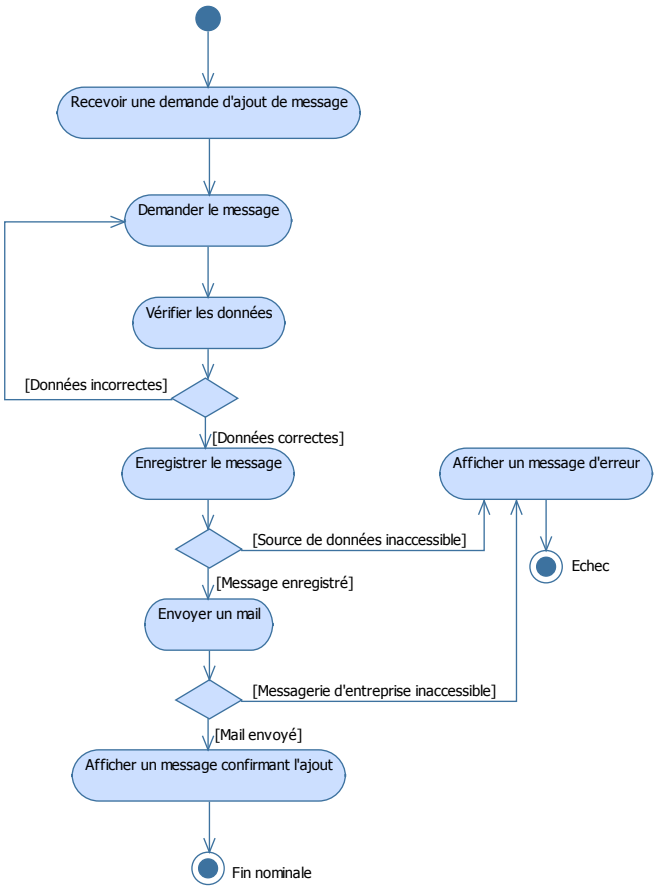

Figure A.2.8, *Diagramme d'activité du cas d'utilisation « Ajouter un message à une tâche ».*

A.2.5 Le sous-système *Gestion des classifications*

A.2.5.1 Le cas d'utilisation *Ranger une tâche dans une catégorie*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*) et doit avoir recherché des tâches (cf. *2.2.3.3 Le cas d'utilisation « Rechercher des tâches »*). La tâche qui sera classée a été précédemment déposée par un chef de projet (cf. *2.2.3.2 Le cas d'utilisation « Déposer une tâche »*).

Scénario nominal

1 - L'acteur décide de ranger, dans une catégorie, une des tâches précédemment recherchées.
2 - L'acteur renseigne la catégorie et la soumet.
3 - Le système enregistre le classement dans la source de données.
4 - Le système affiche un message confirmant le rangement de la tâche dans une catégorie.

Scénarios d'erreur

E1 : *La source de données n'est pas accessible*.
L'enchainement E1 démarre au point 3.

4 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Un acteur pourra retrouver les tâches qu'il a classées à l'aide d'une recherche se basant sur les catégories.

Diagramme d'activité

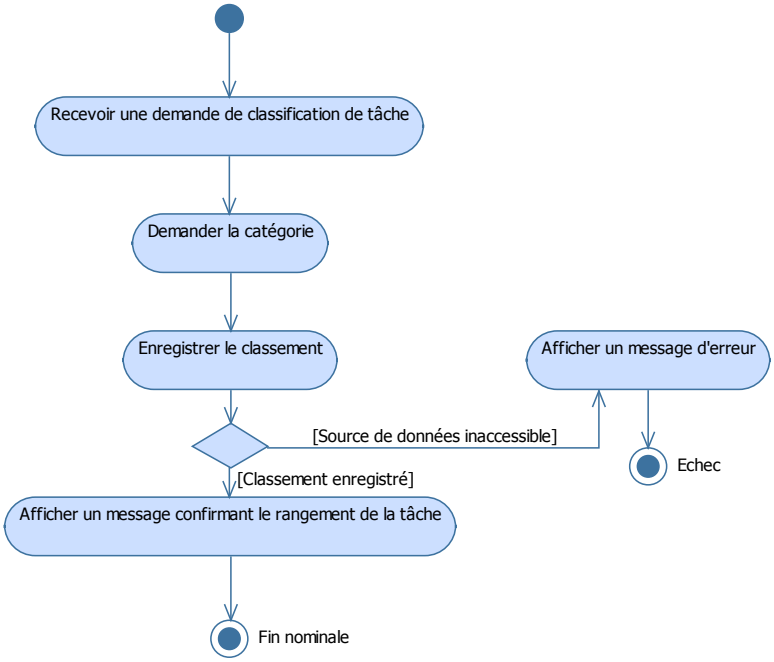

Figure A.2.9, *Diagramme d'activité du cas d'utilisation « Ranger une tâche dans une catégorie ».*

A.2.6 Le sous-système *Gestion des comptes*

A.2.6.1 Le cas d'utilisation *Créer un compte*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*).

Scénario nominal

1 - Le chef de département renseigne les informations relatives à un compte et le crée.
2 - Le système vérifie les données (syntaxe, format, champs remplis).
3 - Le système enregistre le compte dans la source de données.
4 - Le système envoie un mail au propriétaire du compte créé, à l'aide de la messagerie d'entreprise.
5 - Le système affiche un message confirmant la création du compte.

Scénarios alternatifs

A1 : *Vérification des données incorrecte*.
L'enchainement A1 démarre au point 2 du scénario nominal.

3 - Le système indique à l'acteur que les données ne sont pas conformes aux règles imposées (syntaxe, format, champs remplis).
Le scénario nominal reprend au point 1.

Scénarios d'erreur

E1 : *La source de données n'est pas accessible*.
L'enchainement E1 démarre au point 3.

4 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

E2 : *La messagerie d'entreprise n'est pas accessible*.
L'enchainement E2 démarre au point 4 du scénario nominal.

5 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Un compte a été créé par le système.

Diagramme d'activité

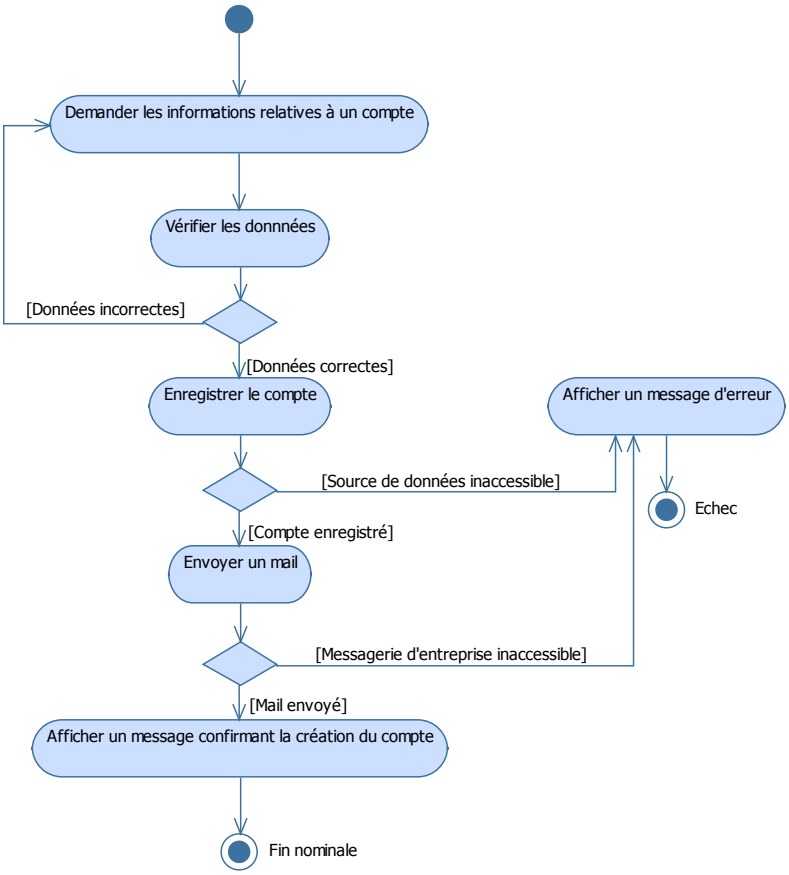

Figure A.2.10, *Diagramme d'activité du cas d'utilisation « Créer un compte ».*

A.2.6.2 Le cas d'utilisation *Modifier son compte*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*). Le compte qui sera modifié a été précédemment créé (cf. *2.2.7.2 Le cas d'utilisation « Créer un compte »*).

Scénario nominal

1 - Le système affiche les informations actuelles du compte.
2 - L'acteur modifie les informations affichées et les soumet.
3 - Le système vérifie les données (syntaxe, format, champs remplis).
4 - Le système enregistre les modifications dans la source de données.
5 - Le système affiche un message confirmant la modification du compte.

Scénarios alternatifs

A1 : *Vérification des données incorrecte*.
L'enchainement A1 démarre au point 3 du scénario nominal.

4 - Le système indique à l'acteur que les données ne sont pas conformes aux règles imposées (syntaxe, format, champs remplis).
Le scénario nominal reprend au point 2.

Scénarios d'erreur

E1 : *La source de données n'est pas accessible*.
L'enchainement E1 démarre au point 4.

5 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Le compte d'un acteur a été modifié.

Diagramme d'activité

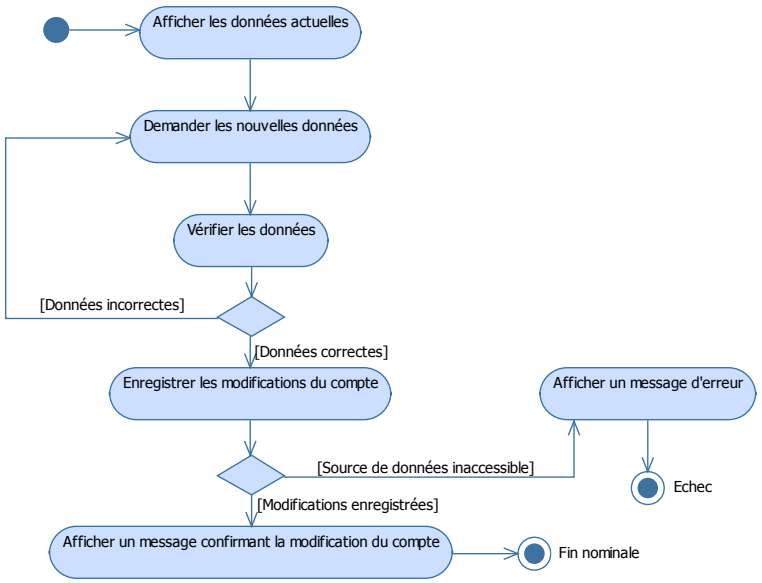

Figure A.2.11, *Diagramme d'activité du cas d'utilisation « Modifier son compte »*.

A.2.6.3 Le cas d'utilisation *Rechercher des comptes*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*).

Scénario nominal

1 - L'acteur précise le critère, la valeur à rechercher, puis lance la recherche.
2 - Le système vérifie les données (syntaxe, format, champs remplis).
3 - Le système cherche les comptes dans la source de données.
4 - Le système affiche les comptes.

Scénarios alternatifs

A1 : *Vérification des données incorrecte*.
L'enchainement A1 démarre au point 2 du scénario nominal.

3 - Le système indique à l'acteur que les données ne sont pas conformes aux règles imposées (syntaxe, format, champs remplis).
Le scénario nominal reprend au point 1.

Scénarios d'erreur

E1 : *La source de données n'est pas accessible*.
L'enchainement E1 démarre au point 3.

4 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Les comptes sont affichés avec toutes les informations qui les concernent.

Diagramme d'activité

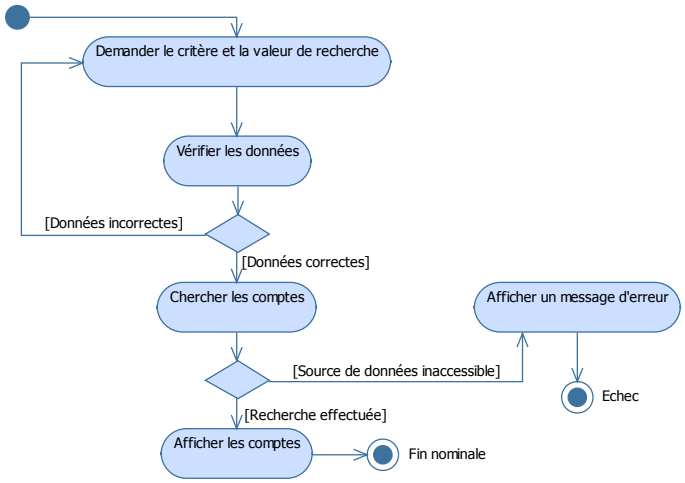

Figure A.2.12, *Diagramme d'activité du cas d'utilisation « Rechercher des comptes »*.

A.2.6.4 Le cas d'utilisation *Supprimer un compte*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*) et doit avoir recherché des comptes (cf. *2.2.7.4 Le cas d'utilisation « Rechercher des comptes »*). Le compte qui sera supprimé a été précédemment créé (cf. *2.2.7.2 Le cas d'utilisation « Créer un compte »*).

Scénario nominal

1 - Le chef de département décide de supprimer un des comptes précédemment recherchés.
2 - Le chef de département confirme la suppression du compte.
3 - Le système supprime le compte dans la source de données.
4 - Le système affiche un message confirmant la suppression du compte.

Scénarios alternatifs

A1 : *L'acteur ne confirme pas la suppression*.
L'enchainement A1 démarre au point 2.

3 - Le système revient à la vue précédente, c'est-à-dire l'interface de recherche.
Le scénario nominal reprend au point 1.

Scénarios d'erreur

E2 : *La source de données n'est pas accessible*.
L'enchainement E2 démarre au point 3.

4 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Le compte d'un acteur a été supprimé par le système.

Diagramme d'activité

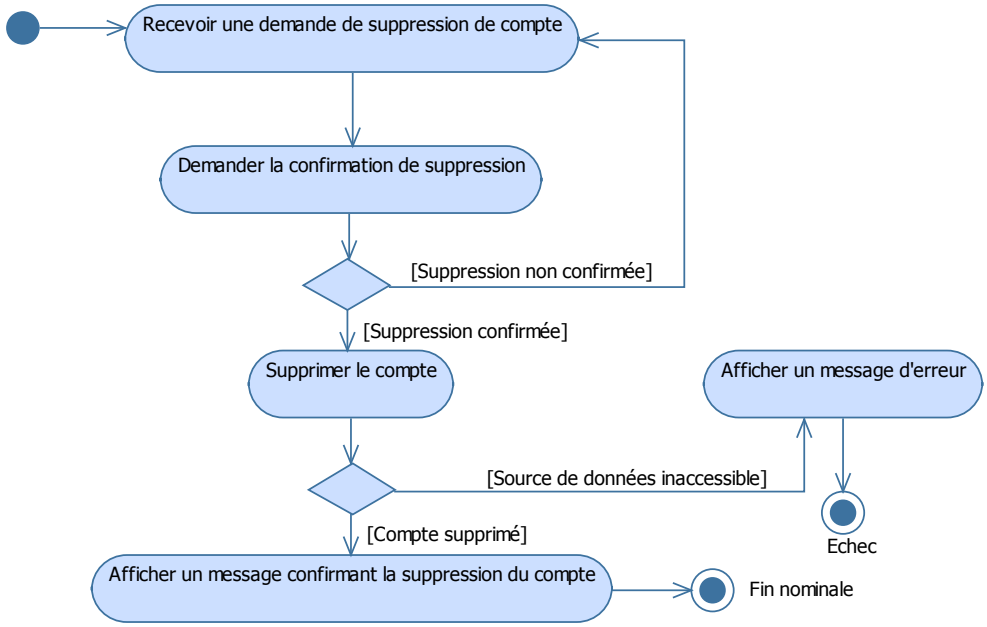

Figure A.2.13, *Diagramme d'activité du cas d'utilisation « Supprimer un compte »*.

A.2.7 Le sous-système *Gestion des départements*

A.2.7.1 Le cas d'utilisation *Modifier les informations de son département*

Pré-conditions

L'acteur doit être préalablement connecté (cf. *2.2.2.2 Le cas d'utilisation « Se connecter »*).

Scénario nominal

1 - Le système affiche les données du département.
2 - L'acteur modifie les informations affichées.
3 - Le système vérifie les données (syntaxe, format, champs remplis).
4 - Le système enregistre les modifications dans la source de données.
5 - Le système affiche un message confirmant la mise à jour du département.

Scénarios alternatifs

A1 : *Vérification des données incorrecte*.
L'enchainement A1 démarre au point 3 du scénario nominal.

4 - Le système indique à l'acteur que les données ne sont pas conformes aux règles imposées (syntaxe, format, champs remplis).
Le scénario nominal reprend au point 2.

Scénarios d'erreur

E1 : *La source de données n'est pas accessible*.
L'enchainement E1 démarre au point 4.

5 - Le système affiche un message d'erreur à l'acteur.
Le cas d'utilisation se termine en échec.

Post-conditions

Le système a permis de modifier les informations d'un département.

Diagramme d'activité

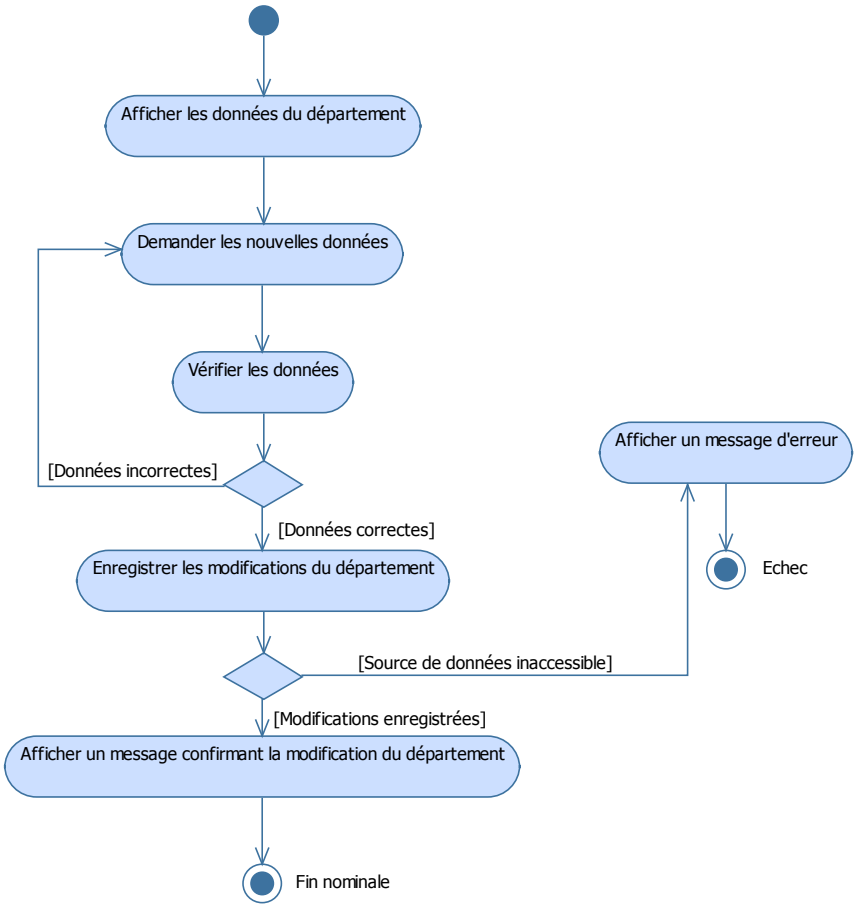

Figure A.2.14, *Diagramme d'activité du cas d'utilisation « Modifier les informations de son département »*.

Annexe 3. Informations complémentaires concernant la phase de conception orientée objet

L'annexe 3 présente la liste des objets candidats et la description des interactions entre objets pour les cas d'utilisation qui n'ont pas été pris en exemple au cours du chapitre *Chapitre 7. Conception orientée objet*.

A.3.1 Le sous-système *Gestion des attributions*

A.3.1.1 Le cas d'utilisation *Attribuer une tâche*

A.3.1.1.1 Liste des objets candidats

Le tableau *A.3.1* présente la liste des objets candidats pour le cas d'utilisation *Attribuer une tâche :*

Tableau A.3.1, *Liste des objets candidats pour le cas d'utilisation « Attribuer une tâche ».*

Stéréotype	Objet
<<boundary>>	consultation-task-ui.jsf, assignment-task-ui.jsf
<<control>>	CreationAssignmentController
<<delegate>>	AssignmentDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	AssignmentService
<<lifecycle>>	AssignmentDAOJpa, HistoricalStatusDAOJpa
<<entity>>	Assignment, HistoricalStatus

A.3.1.1.2 Description des interactions entre objets

Les figures *A.3.1* et *A.3.2* décrivent, pour le cas d'utilisation *Attribuer une tâche*, les interactions entre objets *:*

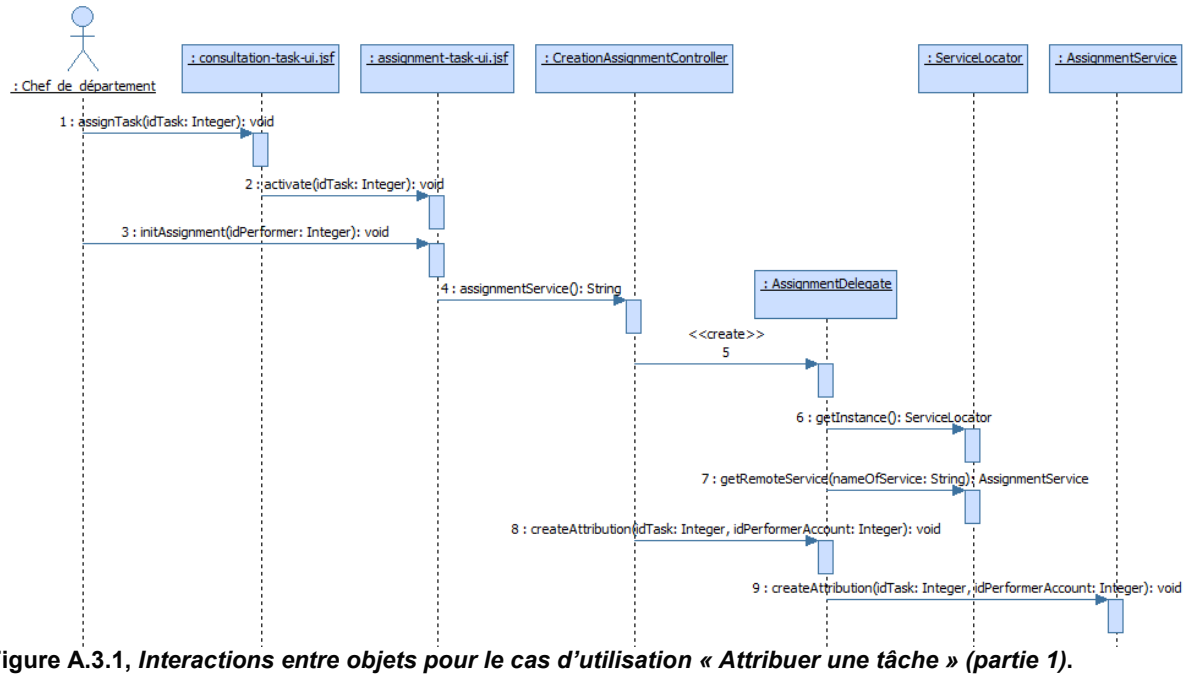

Figure A.3.1, *Interactions entre objets pour le cas d'utilisation « Attribuer une tâche » (partie 1).*

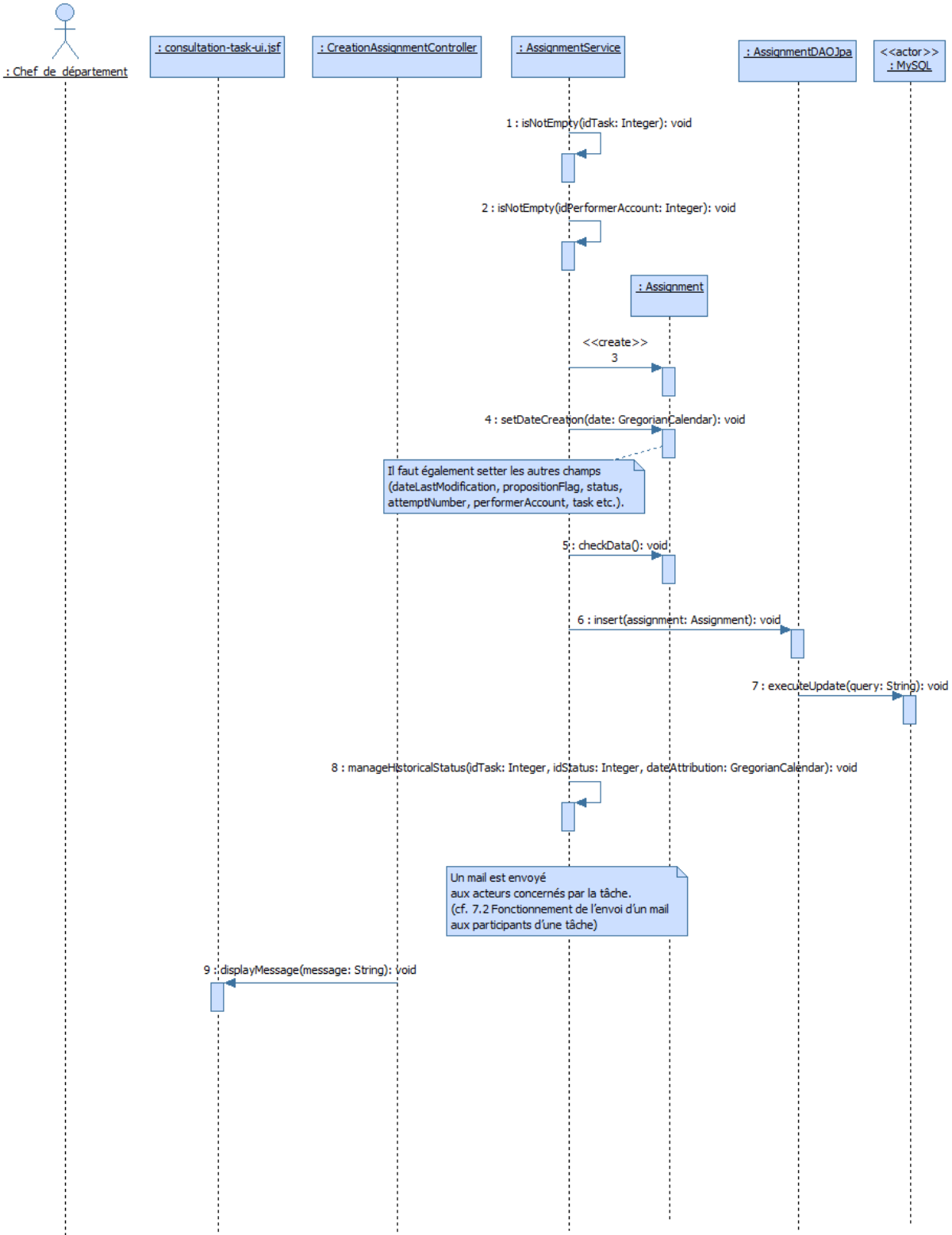

Figure A.3.2, *Interactions entre objets pour le cas d'utilisation « Attribuer une tâche » (partie 2)*.

A.3.1.2 Le cas d'utilisation *Accepter/refuser une tâche*

A.3.1.2.1 Liste des objets candidats

Le tableau *A.3.2* présente la liste des objets candidats pour le cas d'utilisation *Accepter/refuser une tâche :*

Tableau A.3.2, *Liste des objets candidats pour le cas d'utilisation « Accepter/refuser une tâche ».*

Stéréotype	Objet
<<boundary>>	proposition-task-ui.jsf
<<control>>	ValidateAssignmentController
<<delegate>>	AssignmentDelegate, TaskDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	AssignmentService, TaskService
<<lifecycle>>	SessionManager, AssignmentDAOJpa, TaskDAOJpa
<<entity>>	Assignment, AccountDTO, Task, TaskDTO

A.3.1.2.2 Description des interactions entre objets

Les figures *A.3.3* et *A.3.4* décrivent, pour le cas d'utilisation *Accepter/refuser une tâche*, les interactions entre objets :

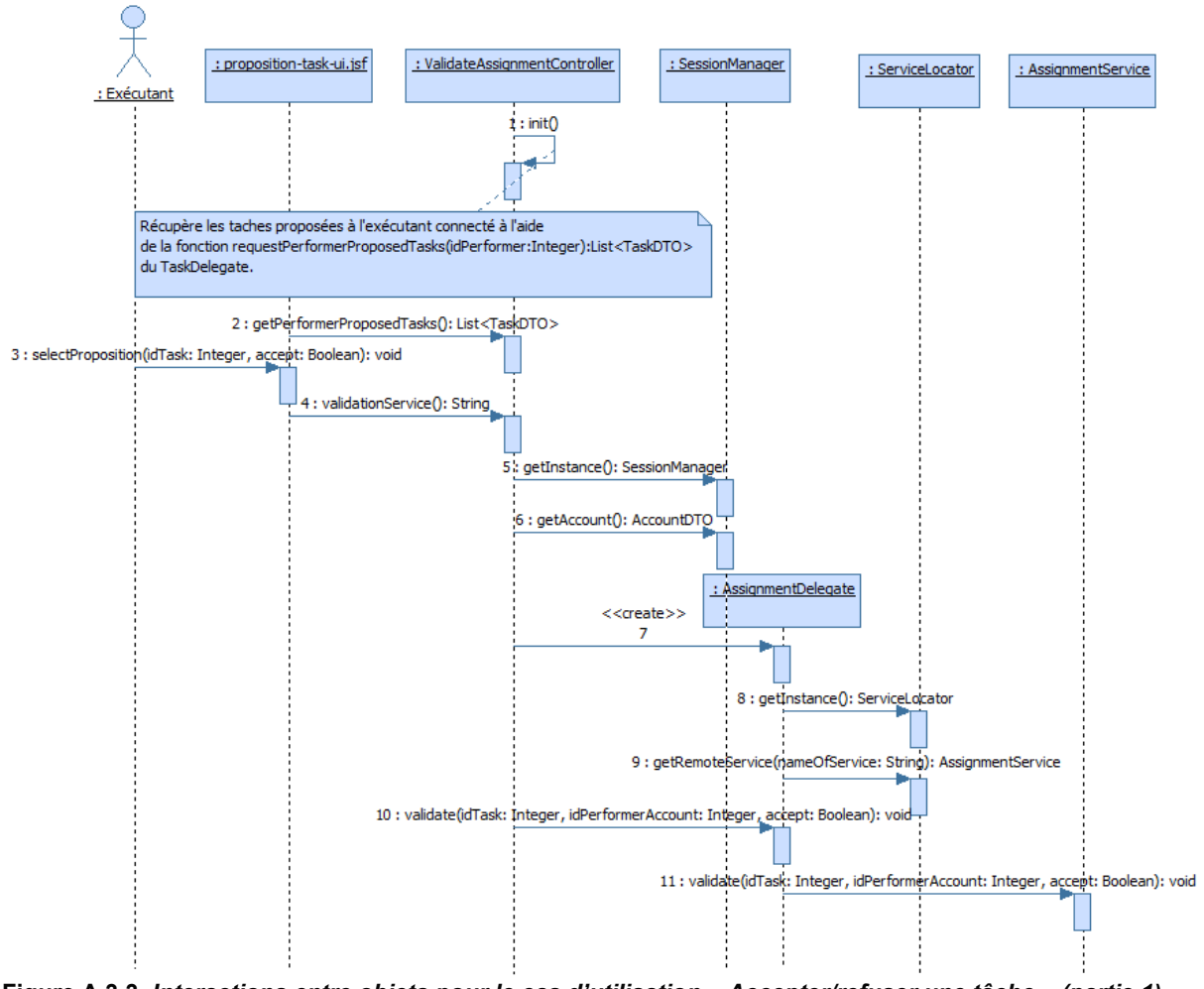

Figure A.3.3, *Interactions entre objets pour le cas d'utilisation « Accepter/refuser une tâche » (partie 1).*

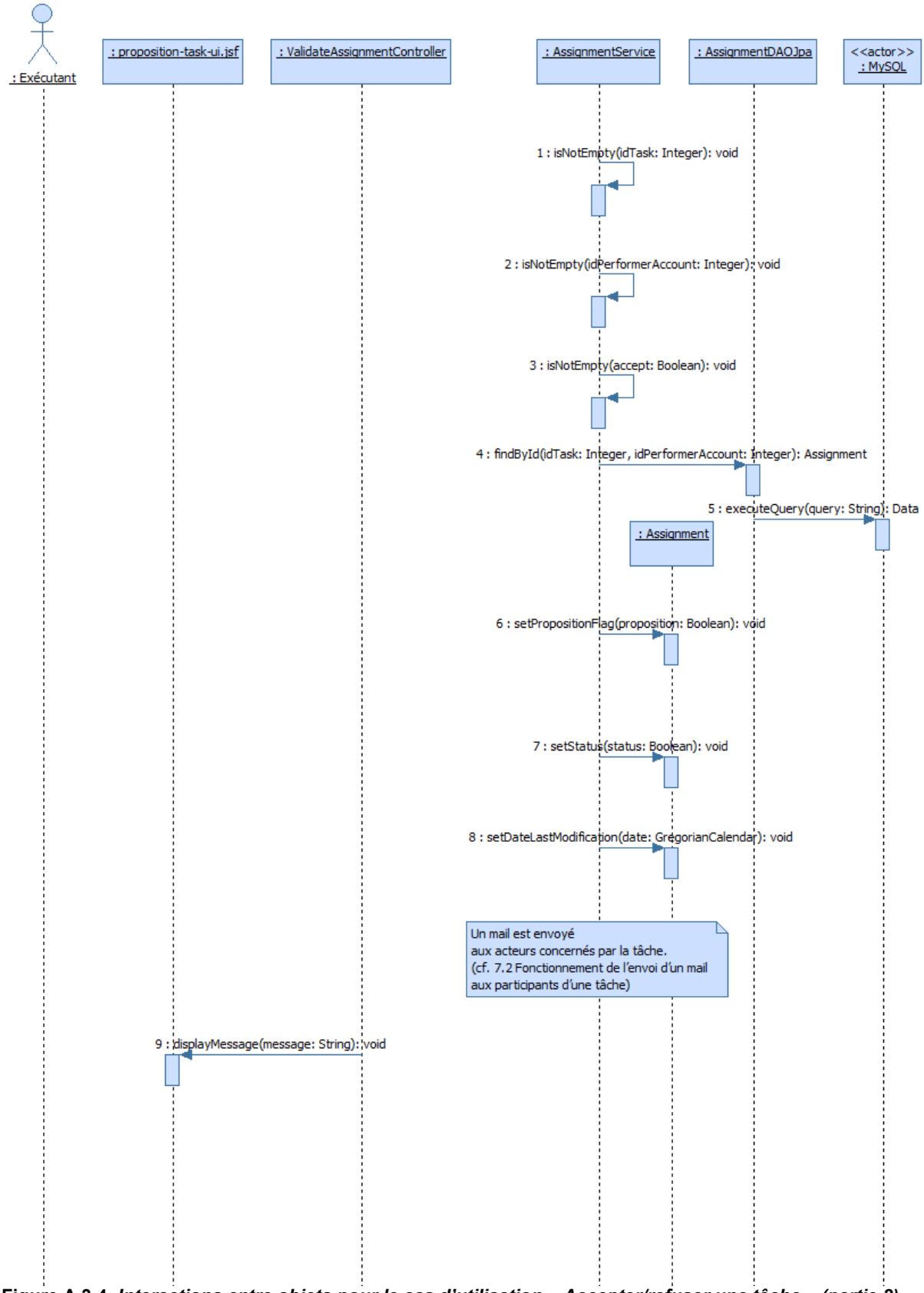

Figure A.3.4, ***Interactions entre objets pour le cas d'utilisation « Accepter/refuser une tâche » (partie 2).***

A.3.2 Le sous-système *Gestion des Messages*

A.3.2.1 Le cas d'utilisation *Ajouter un message à une tâche*

A.3.2.1.1 Liste des objets candidats

Le tableau *A.3.3* présente la liste des objets candidats pour le cas d'utilisation *Ajouter un message à une tâche :*

Tableau A.3.3, *Liste des objets candidats pour le cas d'utilisation « Ajouter un message à une tâche ».*

Stéréotype	Objet
<<boundary>>	consultation-task-ui.jsf, message-task-ui.jsf
<<control>>	CreationMessageController
<<delegate>>	MessageDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	MessageService
<<lifecycle>>	SessionManager, MessageDAOJpa
<<entity>>	Message, AccountDTO

A.3.2.1.2 Description des interactions entre objets

Les figures *A.3.5* et *A.3.6* décrivent, pour le cas d'utilisation *Ajouter un message à une tâche*, les interactions entre objets *:*

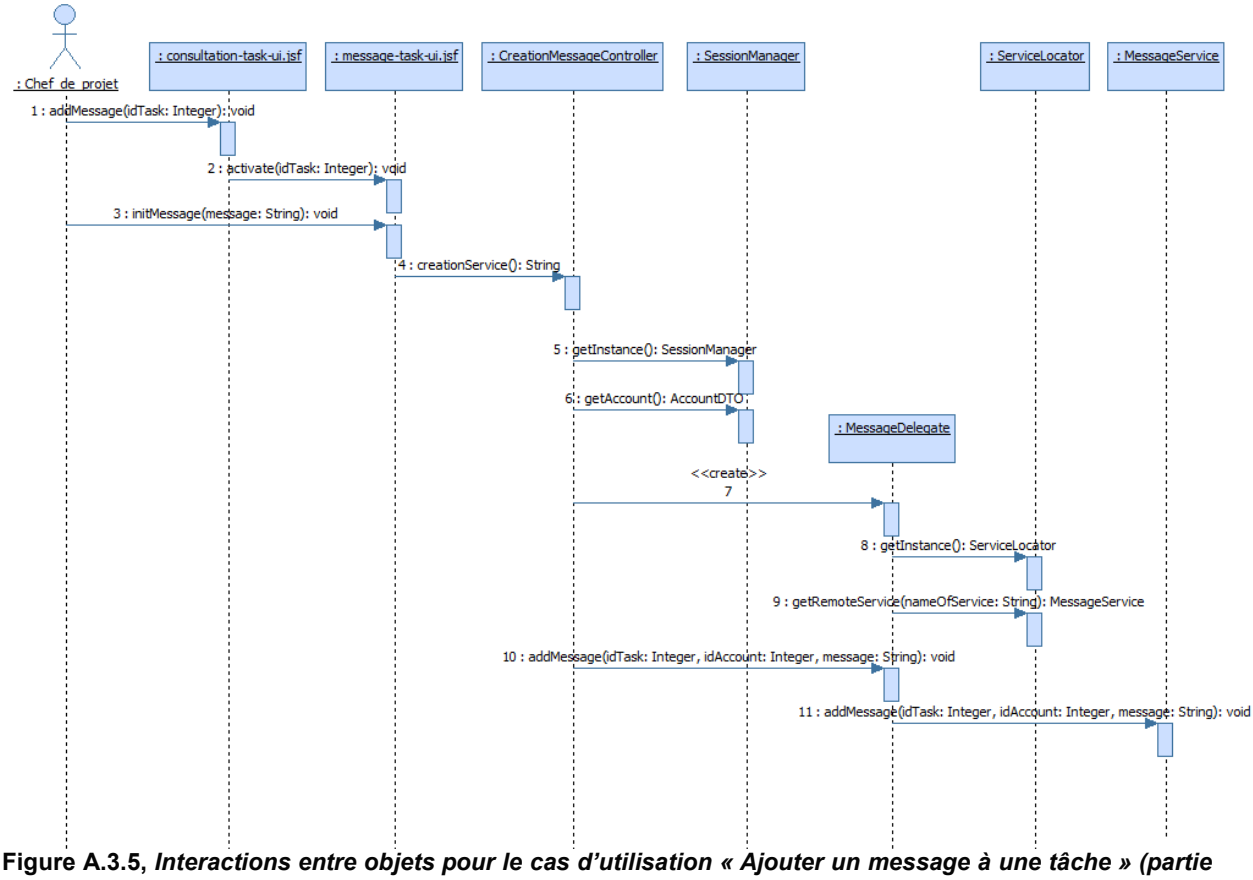

Figure A.3.5, *Interactions entre objets pour le cas d'utilisation « Ajouter un message à une tâche » (partie 1).*

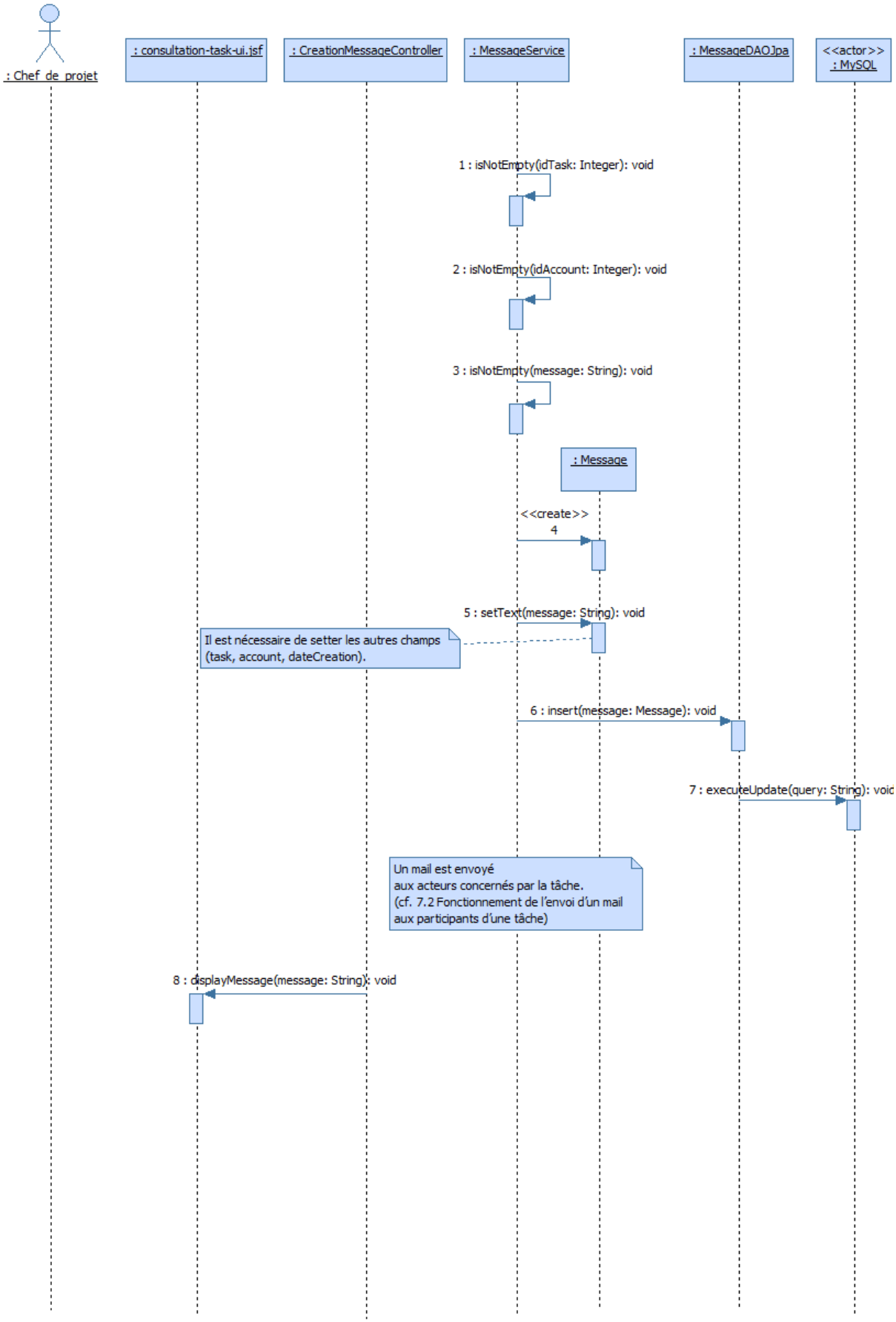

Figure A.3.6, ***Interactions entre objets pour le cas d'utilisation « Ajouter un message à une tâche » (partie 2).***

A.3.3 Le sous-système *Gestion des classifications*

A.3.3.1 Le cas d'utilisation *Ranger une tâche dans une catégorie*

A.3.3.1.1 Liste des objets candidats

Le tableau *A.3.4* présente la liste des objets candidats pour le cas d'utilisation *Ranger une tâche dans une catégorie :*

Tableau A.3.4, *Liste des objets candidats pour le cas d'utilisation « Ranger une tâche dans une catégorie »*.

Stéréotype	Objet
<<boundary>>	consultation-task-ui.jsf, classification-task-ui.jsf
<<control>>	ClassificationController
<<delegate>>	ClassificationDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	ClassificationService
<<lifecycle>>	SessionManager, ClassificationDAOJpa
<<entity>>	Classification, AccountDTO

A.3.3.1.2 Description des interactions entre objets

Les figures *A.3.7* et *A.3.8* décrivent, pour le cas d'utilisation *Ranger une tâche dans une catégorie*, les interactions entre objets *:*

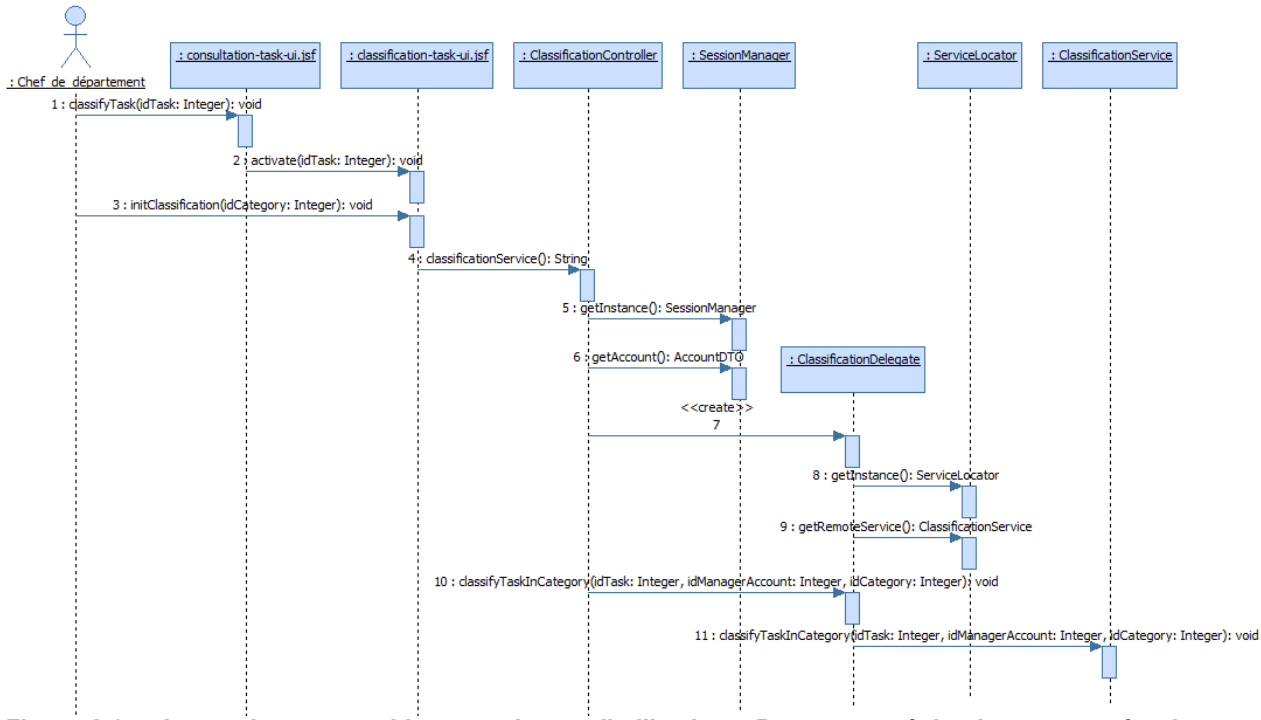

Figure A.3.7, *Interactions entre objets pour le cas d'utilisation « Ranger une tâche dans une catégorie » (partie 1).*

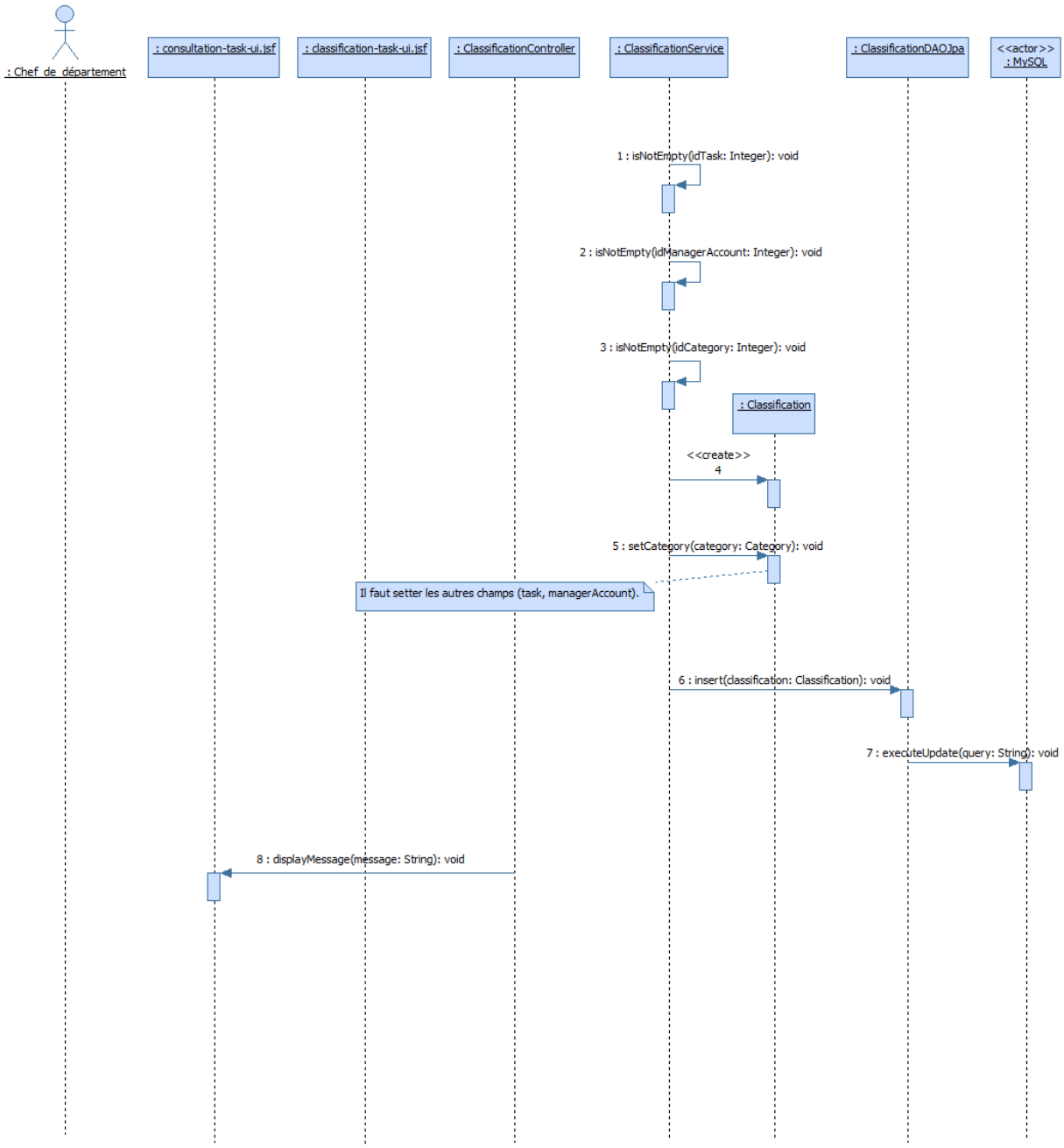

Figure A.3.8, ***Interactions entre objets pour le cas d'utilisation « Ranger une tâche dans une catégorie » (partie 2).***

A.3.4 Le sous-système *Gestion des comptes*

A.3.4.1 Le cas d'utilisation *Créer un compte*

A.3.4.1.1 Liste des objets candidats

Le tableau *A.3.5* présente la liste des objets candidats pour le cas d'utilisation *Créer un compte :*

Tableau A.3.5, *Liste des objets candidats pour le cas d'utilisation « Créer un compte ».*

Stéréotype	Objet
<<boundary>>	creation-account-ui.jsf
<<control>>	CreationAccountController
<<delegate>>	AccountDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	AccountService
<<lifecycle>>	AccountDAOJpa
<<entity>>	Account, AccountDTO

A.3.4.1.2 Description des interactions entre objets

Les figures *A.3.9* et *A.3.10* décrivent, pour le cas d'utilisation *Créer un compte*, les interactions entre objets *:*

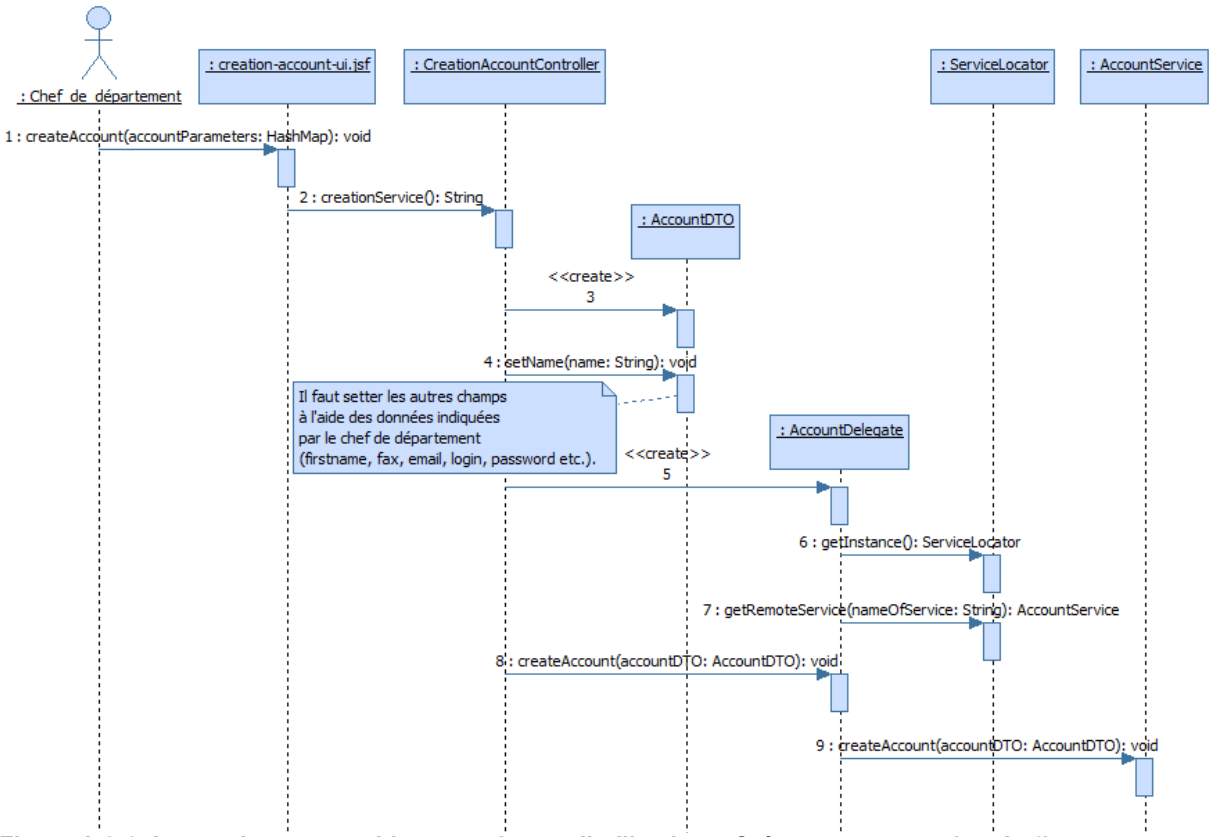

Figure A.3.9, *Interactions entre objets pour le cas d'utilisation « Créer un compte » (partie 1).*

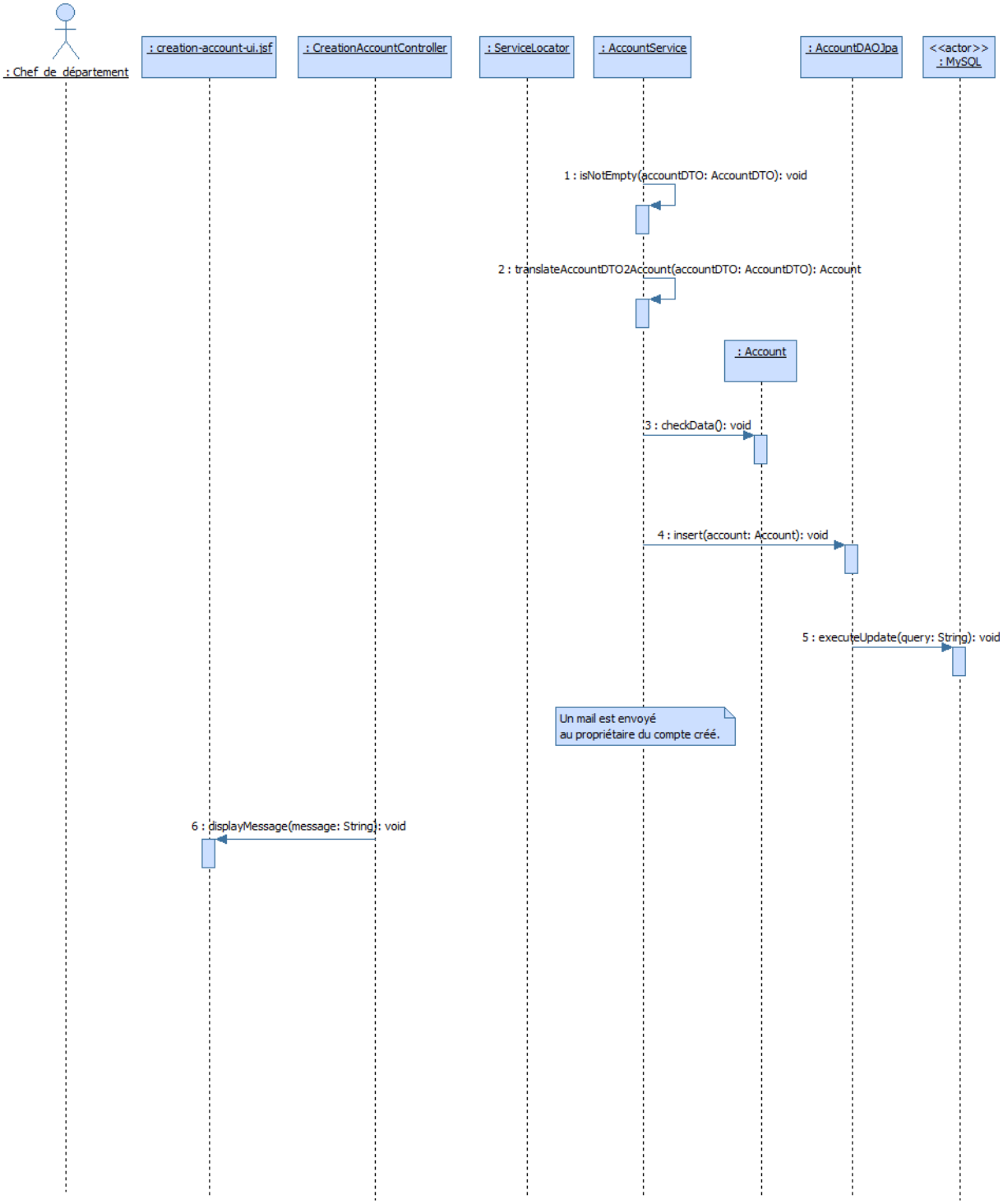

Figure A.3.10, ***Interactions entre objets pour le cas d'utilisation « Créer un compte » (partie 2).***

A.3.4.2 Le cas d'utilisation *Modifier son compte*

A.3.4.2.1 Liste des objets candidats

Le tableau *A.3.6* présente la liste des objets candidats pour le cas d'utilisation *Modifier son compte :*

Tableau A.3.6, *Liste des objets candidats pour le cas d'utilisation « Modifier son compte ».*

Stéréotype	Objet
<<boundary>>	update-account-ui.jsf
<<control>>	UpdateAccountController
<<delegate>>	AccountDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	AccountService
<<lifecycle>>	SessionManager, AccountDAOJpa
<<entity>>	Account, AccountDTO

A.3.4.2.2 Description des interactions entre objets

Les figures *A.3.11* et *A.3.12* décrivent, pour le cas d'utilisation *Modifier son compte*, les interactions entre objets *:*

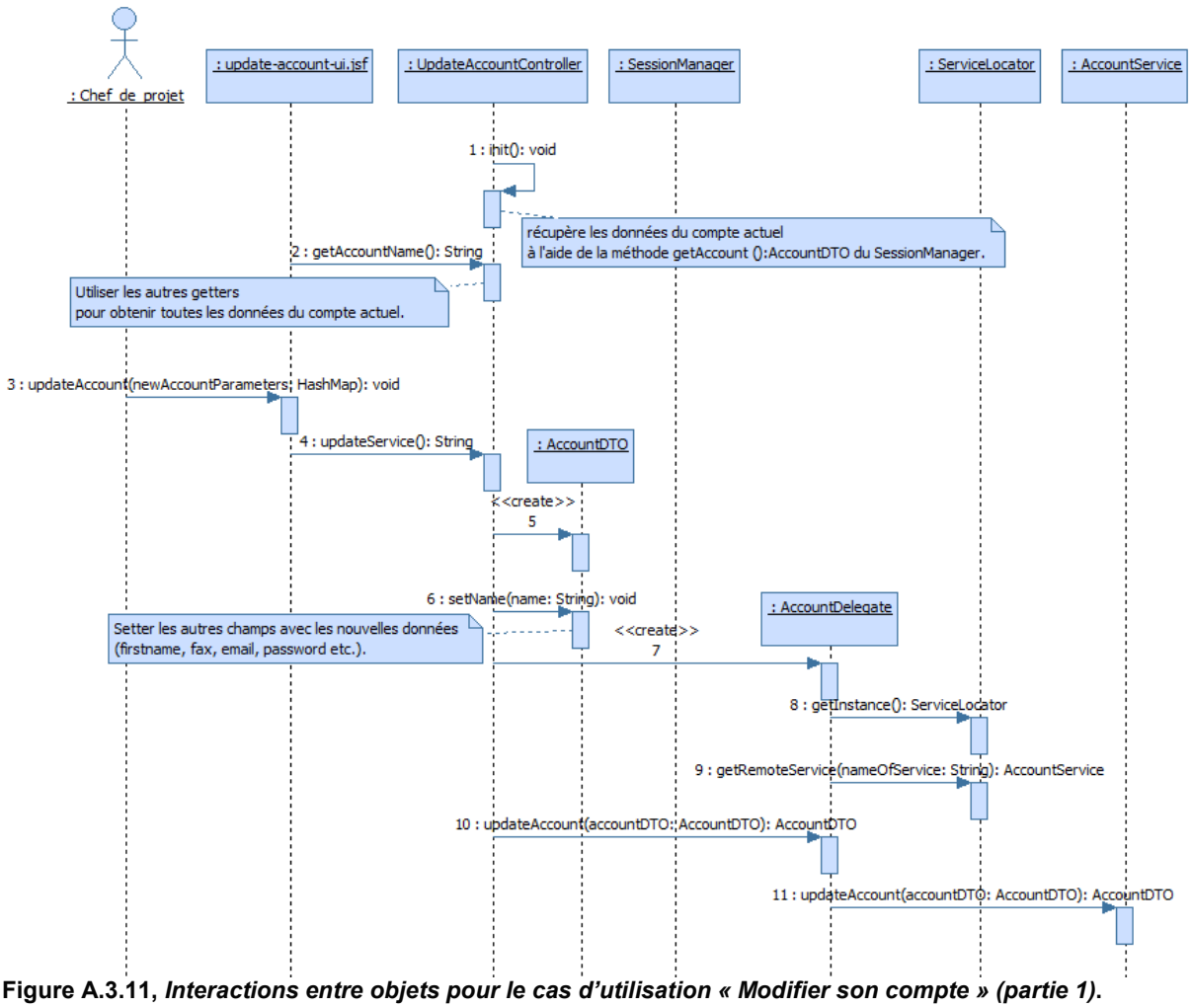

Figure A.3.11, *Interactions entre objets pour le cas d'utilisation « Modifier son compte » (partie 1).*

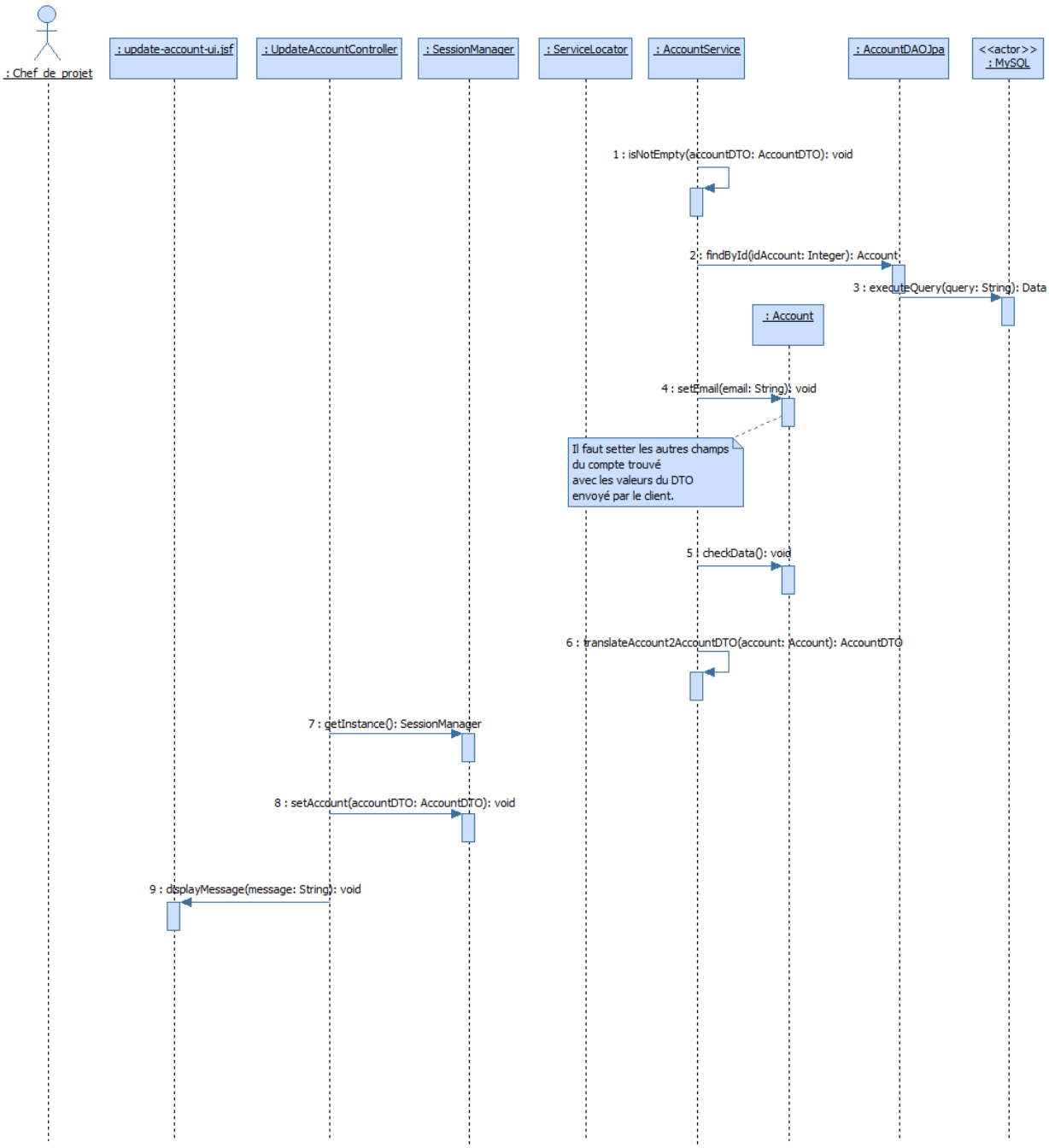

Figure A.3.12, *Interactions entre objets pour le cas d'utilisation « Modifier son compte » (partie 2)*.

A.3.4.3 Le cas d'utilisation *Rechercher des comptes*

L'exemple traité dans ce paragraphe est celui d'une recherche de comptes dont le critère est l'identifiant d'un compte (cf. *2.2.7.4 Le cas d'utilisation Rechercher des comptes*).

A.3.4.3.1 Liste des objets candidats

Le tableau *A.3.7* présente la liste des objets candidats pour le cas d'utilisation *Rechercher des comptes :*

Tableau A.3.7, *Liste des objets candidats pour le cas d'utilisation « Rechercher des comptes ».*

Stéréotype	Objet
<<boundary>>	consultation-account-ui.jsf
<<control>>	ConsultationAccountController
<<delegate>>	AccountDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	AccountService
<<lifecycle>>	AccountDAOJpa
<<entity>>	Account, AccountDTO

A.3.4.3.2 Description des interactions entre objets

La figure *A.3.13* décrit, pour le cas d'utilisation *Rechercher des comptes*, les interactions entre objets *:*

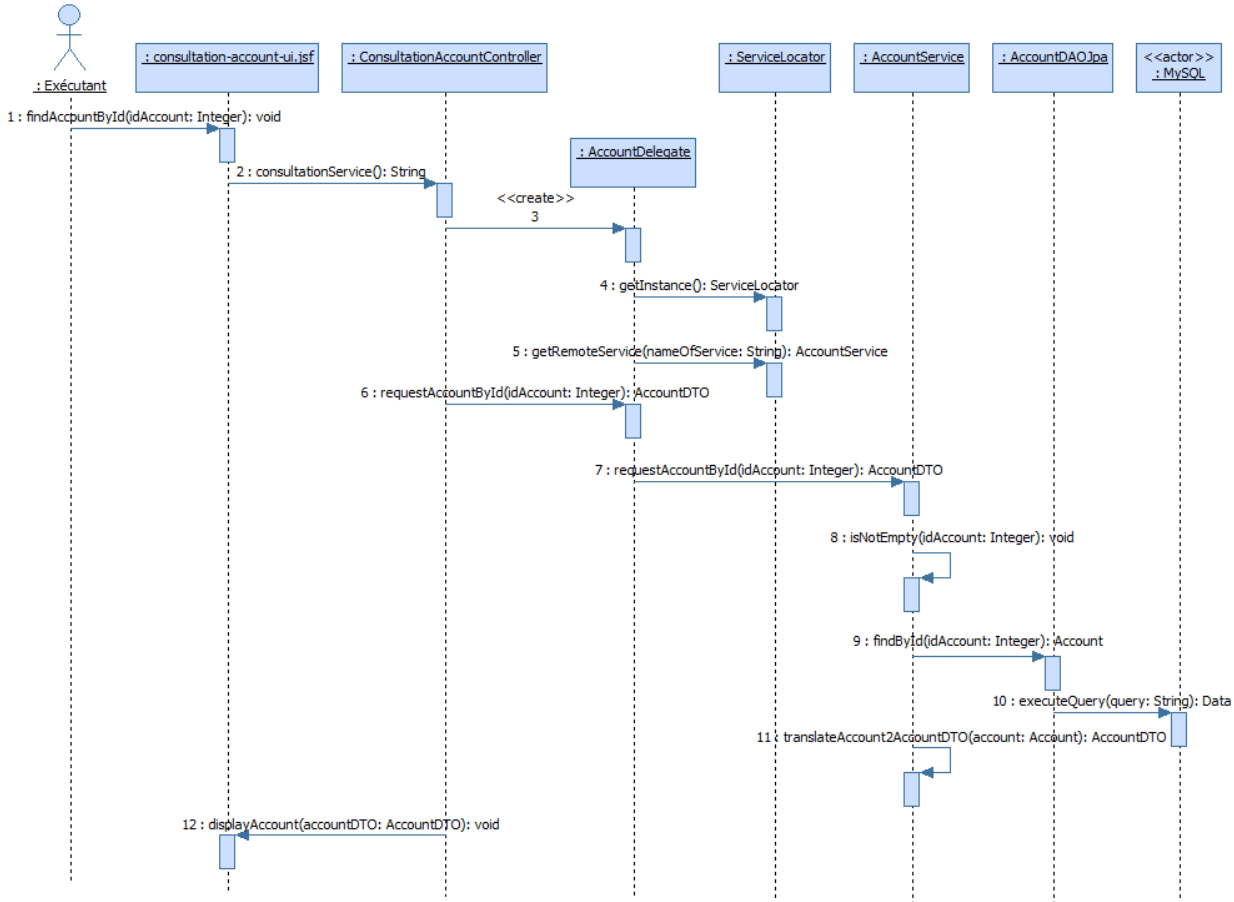

Figure A.3.13, *Interactions entre objets pour le cas d'utilisation « Rechercher des comptes ».*

A.3.4.4 Le cas d'utilisation *Supprimer un compte*

A.3.4.4.1 Liste des objets candidats

Le tableau *A.3.8* présente la liste des objets candidats pour le cas d'utilisation *Supprimer un compte :*

Tableau A.3.8, *Liste des objets candidats pour le cas d'utilisation « Supprimer un compte ».*

Stéréotype	Objet
<<boundary>>	consultation-account-ui.jsf, deletion-account-ui.jsf
<<control>>	DeletionAccountController
<<delegate>>	AccountDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	AccountService
<<lifecycle>>	AccountDAOJpa
<<entity>>	Account

A.3.4.4.2 Description des interactions entre objets

La figure *A.3.14* décrit, pour le cas d'utilisation *Supprimer un compte*, les interactions entre objets *:*

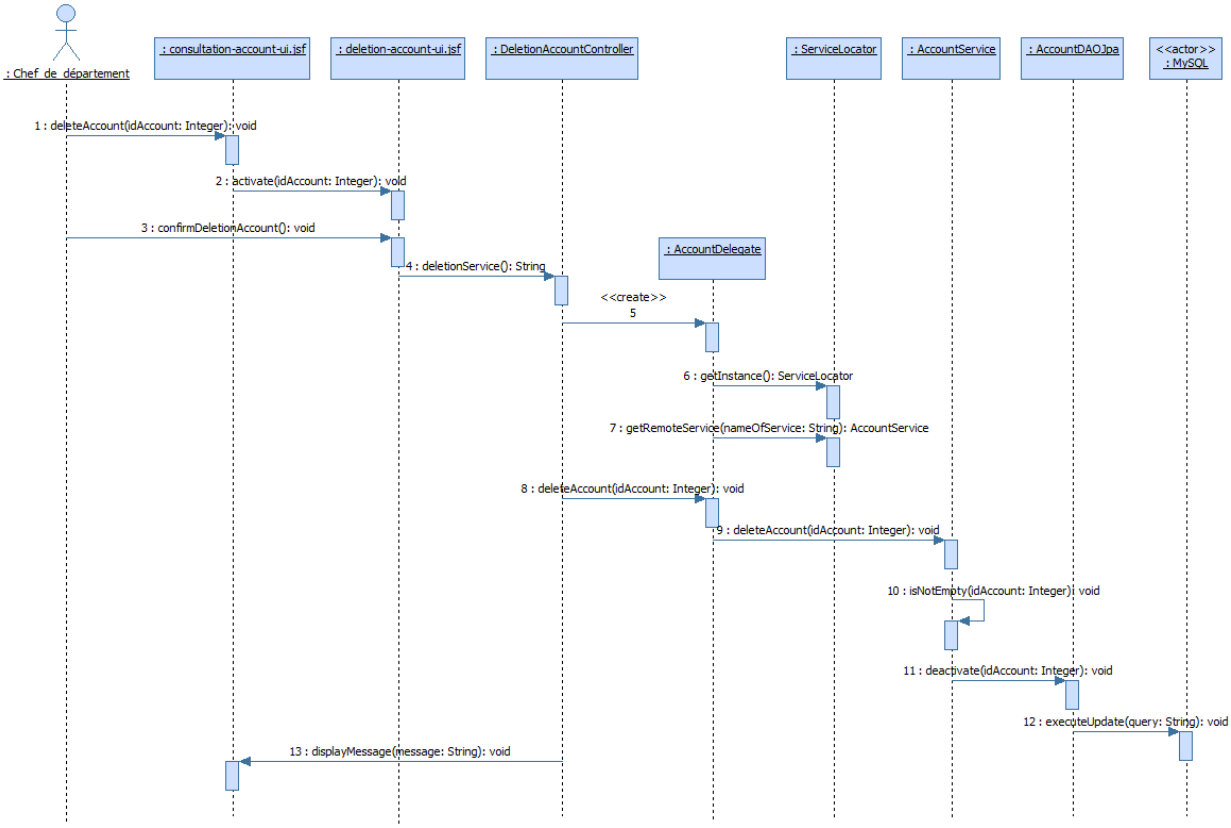

Figure A.3.14, *Interactions entre objets pour le cas d'utilisation « Supprimer un compte ».*

A.3.5 Le sous-système *Gestion des départements*

A.3.5.1 Le cas d'utilisation *Modifier les informations de son département*

A.3.5.1.1 Liste des objets candidats

Le tableau *A.3.9* présente la liste des objets candidats pour le cas d'utilisation *Modifier les informations de son département :*

Tableau A.3.9, *Liste des objets candidats pour le cas d'utilisation « Modifier les informations de son département »*.

Stéréotype	Objet
<<boundary>>	update-department-ui.jsf
<<control>>	UpdateDepartmentController
<<delegate>>	DepartmentDelegate
<<locator>>	ServiceLocator
<<sessionFacade>>	DepartmentService
<<lifecycle>>	SessionManager, DepartmentDAOJpa
<<entity>>	Department, DepartmentDTO

A.3.5.1.2 Description des interactions entre objets

Les figures *A.3.15* et *A.3.16* décrivent, pour le cas d'utilisation *Modifier les informations de son département*, les interactions entre objets *:*

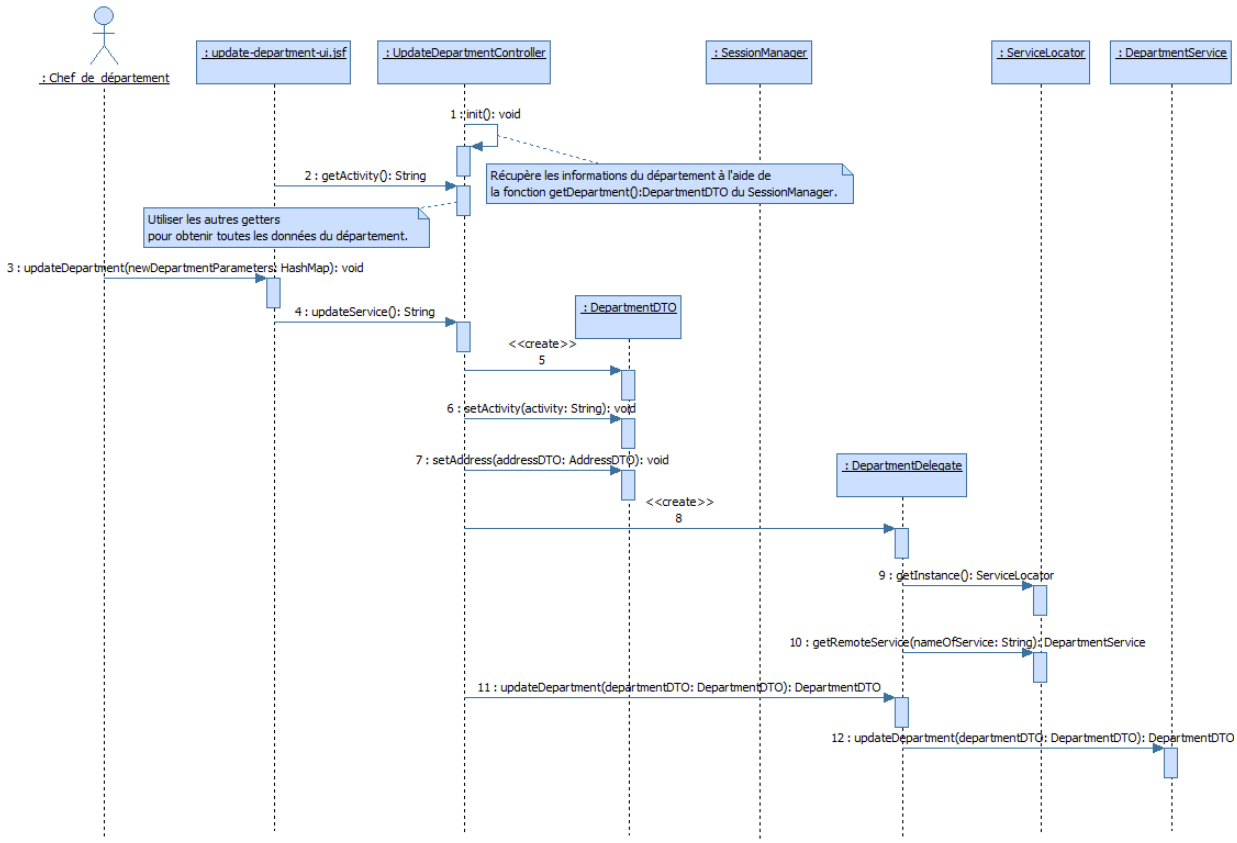

Figure A.3.15, *Interactions entre objets pour le cas d'utilisation « Modifier les informations de son département » (partie 1)*.

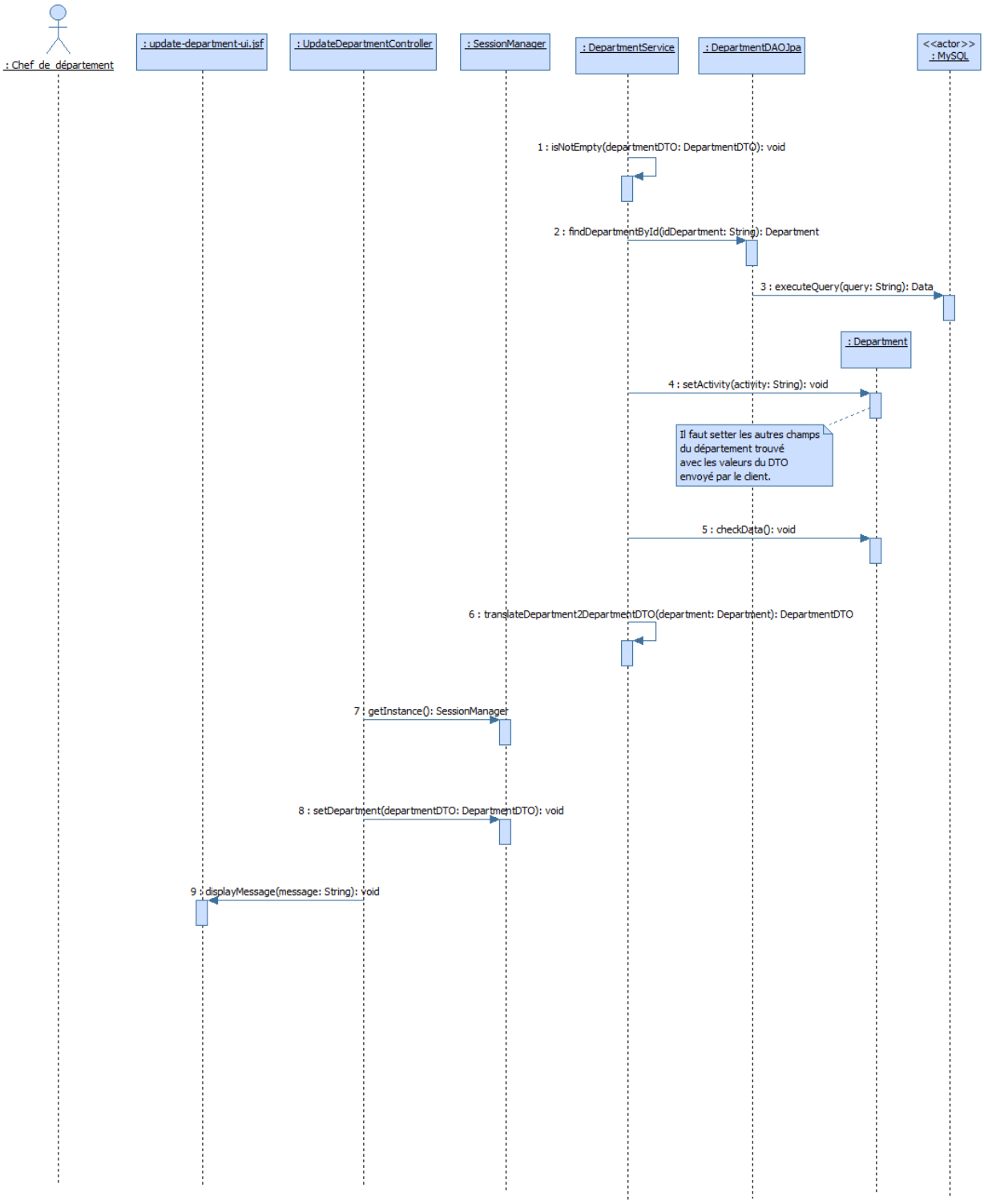

Figure A.3.16, ***Interactions entre objets pour le cas d'utilisation « Modifier les informations de son département » (partie 2).***

Bibliographie

Sites internet

- http://logging.apache.org/log4j/1.2/ : Site de Apache log4j™ 1.2.
- http://freemarker.sourceforge.net/ : Site de FreeMarker.
- http://www.oracle.com/index.html : Site de Oracle.
- http://www.jboss.org/ : Site de JBoss.
- http://news.netcraft.com/ : Site de la société Netcraft.
- http://www.jmdoudoux.fr/accueil.html : Site de Jean-Michel DOUDOUX.
- http://java.cnam.fr/iagl/glg203/index.html : Site de l'UE GLG 203 du CNAM.
- http://java.cnam.fr/iagl/glg204/index.html : Site de l'UE GLG 204 du CNAM.
- http://jfod.cnam.fr/NSY102/ : Site de l'UE NSY 102 du CNAM.

Encyclopédie internet *Techniques de l'ingénieur* (http://www.techniques-ingenieur.fr/)

- [CAB 11], CABANAC G., TESTE O., TUFFERY M., *Architecture client-serveur : modes d'accès aux bases de données*, ref : h3865, 22p, 2011.
- [CHE 04], CHEVASSUS M., *Enterprise Application Integration : EAI*, ref : h2915, 22p, 2004.
- [MER 00], MERLE P., RIVEILL M., *Programmation par composants*, ref : h2759, 26p, 2000.
- [RAV 09], RAVAT F., PUJOLLE G., TESTE O., *Bases de données relationnelles*, ref : h3860, 22p, 2009.

Ouvrages

- [ARR 01], Arrington C., *Enterprise java with UML*, WILEY, 450p, 2001.
- [AVE 01], Avedal K., Ayers D., Briggs T., *JSP Professionnel*, EYROLLES, 943p, 2001.
- [BAP 12], Baptiste J., *Merise Guide pratique: Modélisation des données et des traitements, langage SQL*, ENI, 280p, 2012.
- [BIG 06], Bigand M., *Conception des systèmes d'information: Modélisation des données, études de cas*, TECHNIP, 172p, 2006, http://books.google.fr/books?id=6ThJUVaps7kC&printsec=frontcover&dq=%22Conception+des+syt%C3%A8mes+d%27information:+Mod%C3%A9lisation+des+donn%C3%A9es,+%C3%A9tudes+de+cas%22&hl=fr&sa=X&ei=oUvXULzxI9TI0AW2-oGYAQ&ved=0CDkQ6AEwAA.
- [BOU 11], Boucher A., *Ergonomie web*, EYROLLES, 355p, 2011.

- [CHO 05], Chopra V., Eaves J., Jones R., Li S., Bell J., *Layout Management with Tiles - Beginning JavaServer Pages*, Chapitre 20, WROX, 2005, http://books.google.fr/books?id=Yy4Eyjy1qmkC&printsec=frontcover&dq=Beginning+JavaServer+Pages&hl=fr&sa=X&ei=AT_XUO2aEuin0AXWzIFg&ved=0CDQQ6AEwAA.
- [CHU 06], Chuong F., Corgeron C., Renaux J., Vialette M., *EJB 3*, DUNOD, 334p, 2006.
- [COG 09], Cogoluègnes A., Templier T., Dubois J., Retaillé J., *Introduction à Spring – Spring par la pratique*, Chapitre 1, EYROLLLES, 2009.
- [CON 10], Constantinidis Y., *Les cas d'utilisation - Expression des besoins pour le système d'information*, Chapitre 9, EYROLLES, 2010.
- [CRA 07], Crawford W., Kaplan J., *J2EE Design Patterns*, O'REILLY, 304p, 2007.
- [DEB 09], Debrauwer L., *Design Patterns pour Java*, ENI, 363p, 2009.
- [DES 10], Deslandes A., Grosjean J., Morel M., *Interfaces applicatives : des RIA aux portails de nouvelle génération - Le poste de travail Web*, Chapitre 8, DUNOD, 2010.
- [DRA 11], Draillard F., *Premiers pas en CSS et HTML: CSS 3 et HTML 5*, EYROLLES, 342p, 2011, http://books.google.fr/books?id=APRnkEXi03IC&printsec=frontcover&dq=%22Premiers+pas+en+CSS+et+HTML:+CSS+3+et+HTML+5%22&hl=fr&sa=X&ei=vkfXUO63NuLF0QXUyIDgAw&ved=0CDkQ6AEwAA.
- [FEL 06], Félicité J., *Outils open-source et évolution - Développement Java sous STRUTS (version 1.2)*, Chapitre 8, ENI, 2006.
- [FRO 07], Fron A., *Architectures réparties en Java*, DUNOD, 208p, 2007.
- [GAB 09], Gabillaud J., *SQL - Oracle 11g*, Chapitre 2, ENI, 2009.
- [GON 11], Goncalves A., *Java EE 5 3ème édition*, EYROLLES, 340p, 2011.
- [HON 09], Hondermarck O., *Introduction - JavaScript: Le guide complet*, Chapitre 1, Micro Application, 2009, http://books.google.fr/books?id=csQfVW3fnAwC&printsec=frontcover&dq=%22JavaScript:+Le+guide+complet%22&hl=fr&sa=X&ei=vUHXUK0CqY3QBaq0gOgM&ved=0CDcQ6AEwAA#v=onepage&q=%22JavaScript%3A%20Le%20guide%20complet%22&f=false.
- [LAF 09 A], Lafosse J., *Les moteurs de templates - Struts 2*, Chapitre 18, ENI, 2009.
- [LAF 09 B], Lafosse J., *Java EE: Guide de développement d'applications web en Java*, ENI, 610p, 2009.
- [LAF 11], Lafosse J., *Développements n-tiers avec Java EE*, ENI, 902p, 2011.
- [LAN 06], Langlet E., *Apache Tomcat 5*, ENI, 475p, 2006, http://books.google.fr/books?id=jrGqdunlNiAC&pg=PT85&dq=%22Apache+Tomcat+5%22&hl=fr&sa=X&ei=PEXXUNPlHuS80QX0xIGoBQ&ved=0CDoQ6AEwAA.
- [LOP 06], Lopez D., Blanco J., *Apache l'essentiel du code et des commandes*, CAMPUS PRESS, 227p, 2006.

- [MIL 06], Miles R., Hamilton K., *Learning UML 2.0*, O'REILLY, 290p, 2006.
- [OLS 07], Olson S., *Ajax pour Java*, O'REILLY, 216p, 2007.
- [PAT 08], Patricio A., *Java Persistence et Hibernate*, EYROLLES, 364p, 2008.
- [ROQ 09], Roques P., *UML 2 par la pratique 7ème édition*, EYROLLES, 396p, 2009.
- [ROQ 11], Roques P., *Processus et architecture - UML 2 en action: De l'analyse des besoins à la conception*, Chapitre 2, EYROLLES, 2011, http://books.google.fr/books?id=Wn6Oj_XsyW0C&printsec=frontcover&hl=fr&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false.
- [SEN 09], Sennesal F., *Java Server Faces avec Eclipse*, ENI, 299p, 2009.
- [SER 06], Servin C., *TCP/IP et les applications - Réseaux & télécoms 2ème édition*, Chapitre 12, DUNOD, 2006, http://books.google.fr/books?id=7D6drjlUc6IC&printsec=frontcover&dq=%22R%C3%A9seaux+%26+t%C3%A9l%C3%A9coms+-+2%C3%A8me+%C3%A9dition%22&hl=fr&sa=X&ei=aUnXUNXxBcOxhAex4ICwBw&ved=0CDcQ6AEwAA.
- [SIM 08], Simon F., *JBoss Développement, déploiement et sécurisation d'applications JEE*, ENI, 399p, 2008.
- [VOL 05], Völter M., Kircher M., Zdun U., *Remoting Patterns*, WILEY, 424p, 2005.
- [ZAK 06], Zakas N., McPeak J., Fawcett J., *Présentation d'Ajax - Ajax pour développer des applications Web à interface riche*, Chapitre 1, FIRST INTERACTIVE, 2006.