Ministère de l'**E**ducation **N**ationale

Université **M**ontpellier II

FMIN 200

Rapport TER : Réalisation de Framework métier Phélix du Master 1 Informatique

Présenté Par Mrs:

Akram Ajouli

Mohammed Sami El Moutaoukil

Sami Kendri

El Mehdi Mouhtam

Encadré Par :

Pr Mr. Christophe Dony

Responsable du module

Mc. Mr. Michel Leclere

Promotion 2008 -2009

Remerciements

Au terme de ce Ter, nous tenons à remercier tout particulièrement Monsieur Christophe Dony pour les conseils avertis qu'il nous a procurés tout au long de la réalisation du projet, également pour sa disponibilité et son attachement à ce que nous réalisions un projet concret et bien fait et pour sa patience à notre égard et l'intérêt qu'il a montré lors du présent travail.

Nous remercions aussi les membres du jury de nous avoir honorés de leur présence.

Table des matières

Introduction Générale 1

1 Concepts de Framework et Workflow 4

1.1 Introduction 4

1.2 Le Framework 4

1.2.1 Définition d'un Framework 4

1.2.2 Description des framework 4

1.2.3 Présentation des Framework 5

1.3 WorkFlow 6

1.3.1 Définition 6

1.4 Schémas de réutilisation 7

1.6 Conclusion 8

2 État de l'art Framework 9

2.1 Introduction 9

2.2 Limitation de la programmation procédurale 9

2.3 Programmation à objet 10

2.4 Programmation par Framework 10

2.5 Principaux travaux de recherche sur les Frameworks 11

2.6 Principaux Frameworks 12

2.6.1 MVC 12

2.6.2 ATB 12

2.6.3 Éditalk 12

2.6.4 Actalk 12

2.6.5 The simulation Framework 13

2.7 Démarche utilisé pour la réalisation de Phélix 13

2.7.1 Spécification des deux applications 14

2.7.2 Déterminer les points communs 14

2.7.3 Spécification de l'architecture de Phélix 14

2.8 Conclusion 14

3 Framework

3.1 Introduction 13
3.2 Les classes abstraite 13
3.2.1 BusinessObject 15
3.2.2 Task 15
3.2.3 Actor 16
3.2.4 RessourceManager 16
3.2.5 Workflow 16
3.3 Les classes concrète 17
3.3.1 CreateBO 18
3.3.2 UpdateBO 18
3.3.3 SearcheBO 18
3.3.4 Role 18
3.3.5 Employe 19
3.3 Implémentation et points de paramétrage ou d'extension 20
3.3.3 BusinessObject 20
3.3.3 Task 22
3.3.4 Actor 24
3.3.5 RessourceManager 24
3.3.6 Workflow 25
3.4 Conclusion 26

4 Paramétrage du Framework

4.1 Introduction 27
4.2 Spécification du Workflow recrutement 27
4.2.1 Modélisation de recrutement 28
4.2.2 Implémentation et paramétrage 28
4.3 Spécification du Workflow Congé 32
4.3.1 Modélisation de congé 33
4.3.2 Implémentation et paramétrage 33
4.4 Conclusion 32

5 Tests

5.1 Introduction 37
5.2 JUnit 37
5.3 Pourquoi JUnit ? 38
5.4 Génération ou création de la classe du test unitaire 39
5.4.1 Ecriture du test unitaire 41
5.4.2 La classe Assert 43
5.4 TestSuite 44
5.4 Conclusion 45

Conclusion et perspectives **46**

Références bibliographies **48**

Table des figures

Figure. 1.1 Schéma d'un Workflow de publication de document sur intranet……. 05
Figure. 2.1 Démarche du framework 11
Figure. 3.1 Modèle version 1 …….. 14
Figure. 3.2 Modèle version 2 17
Figure. 4.1 Diagramme de classes recrutement. 28
Figure. 4.2 Processus demande Congé. 32
Figure. 4.3 Diagramme de classes gestion congé. 33
Figure. 5.1 Création d'un test 1 39
Figure. 5.2 Création d'un test 2. 40
Figure. 5.3 Création d'un test 3. 43
Figure. 5.4 Visualisation d'un test réussi. 42
Figure. 5.5 Visualisation d'un test échoué 43
Figure. 5.6 Création visualisation d'un TestSuite 45

Introduction Générale

Durant les vingt dernières années, le développement de logiciels a changé de manière significative. Son évolution a répondu en grande partie aux besoins des développeurs de produire les logiciels plus rapidement, et de prêter une attention plus soutenue aux utilisateurs finaux des systèmes.
Les frameworks sont aujourd'hui considérés par de nombreux spécialistes comme un des principaux avancements de la technologie à objets. Parce qu'ils mettent à disposition une infrastructure préétablie et une interface bien définie, ils permettent de pallier aux problèmes de la « simple » programmation à objets. Avec des frameworks bien conçus, il est en effet plus facile de développer des extensions, de mettre en facteur des fonctionnalités communes, de favoriser l'interopérabilité des composants, et d'améliorer la fiabilité et la maintenance des logiciels.

Réaliser un framework pour un nouveau domaine

L'idée de créer un framework pour workflow nous est venue suite à un stage effectué dans une entreprise, la mission était de paramétrer un framework d'application pour automatiser les tâches effectuées lors du déroulement du processus de recrutement comme la création d'un candidat dans le système, la mise à jour du bilan d'un entretien réalisé et la création d'une proposition d'embauche pour un candidat recruté. Bien que le framework des jeux de balles déjà écrit est un cas d'étude très intéressant et le but étant d'améliorer ces fonctionnalités et d'en ajouter de nouvelles, nous avons pris la décision de plonger dans le monde des frameworks orientées objets afin de se confronter à la complexité de la démarche pour réaliser un framework fiable et extensible qui grâce aux techniques de réutilisation d'une application semi-finie comme l'adaptation par spécialisation et/ou par composition, puisse générer des workflows en indiquant au développeurs du framework les classes abstraites et leurs méthodes qui doivent être redéfinies et les classes concrétés a instanciées ainsi que les points d'extension, le tout pour écrire un minimum de lignes de code dans un délais respectable.

But du projet

Le but du projet est la réalisation du framework métier «Phélix» pour les workflows d'entreprise au niveau interne comme le recrutement, les demande de congés, le remboursement des notes de frais, ou encore les demandes de crédit. Dans le souci d'atteindre notre objectif, nous avons analysés plusieurs workflows, puis nous avons choisis les workflows recrutement et demande de congés qui nous semblent appropriés au choix des workflows internes c'est à dire que les acteurs de ces processus sont internes à l'entreprise comme le directeur et les employés.

Domaine d'application : Workflow

Nous allons réaliser un framework dédié là un domaine de workflow nous avons donc :

Un workflow est un flux d'informations au sein d'une organisation, c'est aussi la modélisation et la gestion informatique de l'ensemble des tâches à accomplir et des différents acteurs impliqués dans la réalisation d'un processus métier. Nous avons choisis ce domaine d'application car les processus métier sont l'essence de l'organisation et la politique d'une entreprise dans le but d'atteindre certains de ces objectifs. De plus nous avons constaté durant nos recherches qu'il existe une multitude de processus métier dans une entreprise qui ont tous en commun des objets métier, des acteurs, des tâches et un manager qui distribue les différents rôles sur les acteurs, toutes ces entités suivent un enchaînement logique pour atteindre l'objectif du processus, dans les deux workflows étudiés l'objectif est de recruter des candidats et de délivrer des offres de congés aux employés d'une entreprise.

Ce rapport s'articule autour de cinq chapitres organisés de la façon suivante :

- Chapitre un, on présente dans ce chapitre les notions du Framework, du Workflow et des schémas de réutilisations.
- Chapitre deux, on présente l'état de l'art des framework et la démarche suivie pour réaliser le travail.
- Chapitre trois, on présente l'architecture du framework et les points de paramétrage
- Chapitre quatre, nous paramétrons le framework pour générer les deux workflows « recrutement » et « congé »
- Chapitre cinq, on présente JUnit tout en montrant la manière dont on a procédé pour réaliser des tests sur le Framework

Chapitre I

Concepts de Framework et Workflow

1.1 Introduction

Dans ce chapitre nous abordons tout d'abord le Framework, puis on va voir une idée sur les Workflow, ensuite on présente les principaux schémas de réutilisation.

1.2 Le Framework

1.2.1 Définition d'un Framework

Un Framework est une application logicielle extensible et adaptable, intégrant [12] :

- les connaissances d'un domaine,
- une architecture logicielle complète,
- le code du cœur générique de l'application et généralement le code d'un ou deux exemples spécifiques.

1.2.2 Description des Framework:

Un Framework fournit un ensemble de fonctions facilitant la création de tout ou d'une partie d'un système logiciel, ainsi qu'un guide architectural en divisant le domaine visé en modules. Un Framework est habituellement implémenté à l'aide d'un langage à objets, bien que cela ne soit pas strictement nécessaire : un Framework objet fournit ainsi un guide architectural en divisant le domaine visé en **classes** et en définissant les responsabilités de chacune ainsi que les collaborations entre classes. Ces classes peuvent être subdivisées en **classes abstraites**.

On trouve différents types de Framework :

1. Framework d'infrastructure système : pour développer des systèmes d'exploitation, des interfaces graphiques, des outils de communication. (exemple : Framework .Net, Eclipse, NetBeans, Struts)
2. Framework d'intégration intergicielle (*middleware*) : pour fédérer des applications hétérogènes. Pour mettre à dispositions différentes technologies sous la forme d'une interface unique. (exemple : Ampoliros avec ses interfaces RPC, SOAP, XML)
3. Framework d'entreprise : pour développer des applications spécifiques au secteur d'activité de l'entreprise.
4. Framework orientés Système de gestion de contenu

Les principaux avantages des Framework sont la réutilisation de leur code, la standardisation du cycle de vie du logiciel (Spécification, développement, maintenance, évolution).

1.2.3 Présentation des Framework :

Le Framework :

- Se résume à un ensemble de classes utilitaires.
- Ne traite qu'une problématique de développement.

En réalité, le terme « Framework » signifie aussi :

- Un cadre de conduite de projet.
- Un cadre d'analyse.
- Un cadre de conception.
- Un cadre de programmation.

Que peut-on attendre d'un Framework ?

- Une productivité accrue
- Une homogénéisation entre applications
- Une meilleure maintenabilité
- Une capitalisation du savoir-faire.

Les difficultés rencontrées

- Un temps de prise en main pour les développeurs
- Formation des équipes
- Evolution du Framework
- Maintien de la documentation [13]

1.3 Définition d'un WorkFlow :

Un ***workflow*** est un flux d'informations au sein d'une organisation, comme par exemple la transmission automatique de documents entre des personnes.

On appelle « workflow » (traduit littéralement « flux de travail ») la modélisation et la gestion informatique de l'ensemble des tâches à accomplir et des différents acteurs impliqués dans la réalisation d'un processus métier (aussi appelé processus opérationnel ou bien procédure d'entreprise). Le terme de « workflow » pourrait donc être traduit en français par « gestion électronique des processus métier ». De façon plus pratique, le workflow décrit le circuit de validation, les tâches à accomplir entre les différents acteurs d'un processus, les délais, les modes de validation, et fournit à chacun des acteurs les informations nécessaires pour la réalisation de sa tâche. Il permet généralement un suivi et identifie les acteurs en précisant leur rôle et la manière de le remplir au mieux.

L'exemple ci-dessous est une représentation très schématique de ce que pourrait être un workflow de publication de document sur un intranet à l'aide d'une interface de publication :

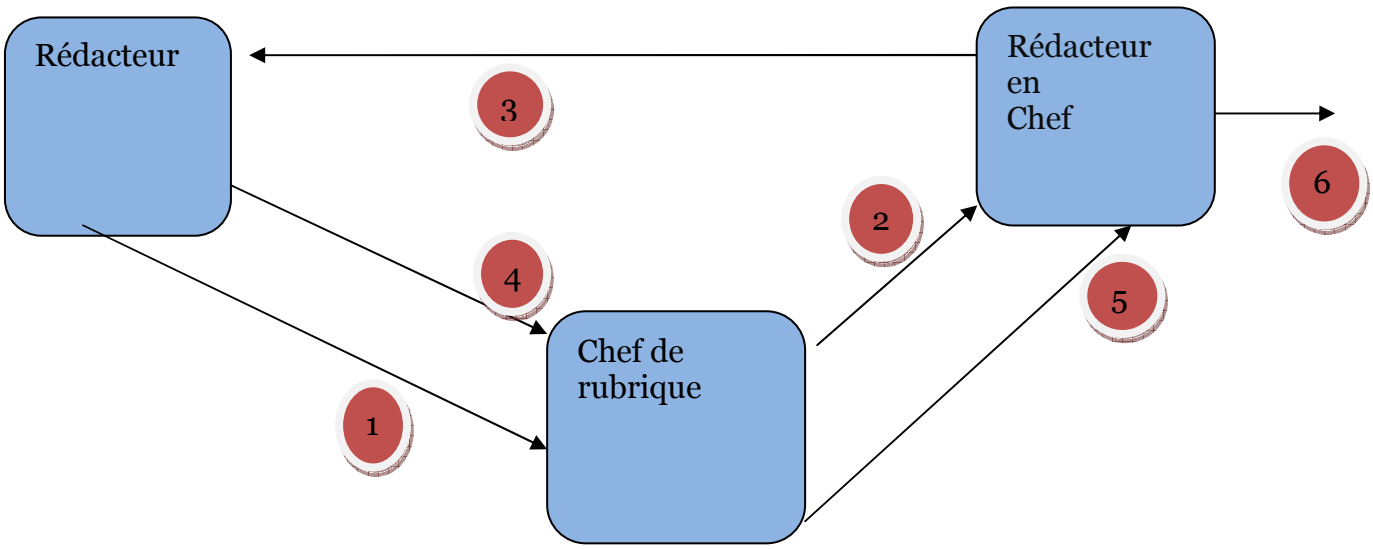

Fig.1.1 Schéma d'un Workflow de publication de document sur intranet.

1. Le rédacteur propose un article au chef de rubrique.
2. Le chef de rubrique regarde le document et le valide.
3. Le rédacteur en chef trouve que le document possède des éléments incompatibles avec l'actualité et retourne le document au rédacteur.
4. Le rédacteur revoit sa copie et la soumet au chef de rubrique.
5. Le chef de rubrique corrige quelques coquilles et transmet l'article au rédacteur en chef.
6. Le rédacteur en chef valide le document pour une publication en ligne [14] [15].

1.4 Schémas de réutilisation

Schéma 1 : Spécialisation ou Redéfinition

Définition d'une méthode de nom M sur une sous-classe SC d'une classe C où une méthode de nom M est déjà définie [12].

Schéma 2: Redéfinition partielle

Redéfinition faisant appel à la méthode redéfinie [12].

Schéma 3 : Adaptation par Spécialisation

Premier schéma de réutilisation permettant d'adapter une méthode à nouveaux besoins sans la modifier et sans dupliquer de code [12].

Schéma 4 : adaptation par composition

Code générique (méthode service) adapté par un plugin défini sur un composite [12].

Schéma 5 : adaptation par fonctions d'ordre supérieur

Fonction d'ordre supérieur : fonction réifiée, que l'on peut stocker dans une variable, référencer, passer en argument [12].

1.6 Conclusion

Ce chapitre représente un petit guide qui donne une idée générale sur les Framework et les WorkFlow qui représentent les deux grands concepts sur lesquels se base l'objectif de ce travail.

Chapitre II

État de l’art Framework

2.1 Introduction

Durant les vingt dernières années, le développement de logiciels a changé de manière significative. Son évolution a répondu en grande partie aux besoins des développeurs de produire les logiciels plus rapidement, et de prêter une attention plus soutenue aux utilisateurs finaux des systèmes. Malgré des gains certains, l’industrie du logiciel est encore confrontée à des cycles de production trop longs qui, de plus, aboutissent souvent à des logiciels qui ne répondent pas de manière satisfaisante aux préoccupations professionnelles.

Les limitations de la programmation procédurale traditionnelle, et des environnements de production logicielle associés, ont conduit l’industrie à s’intéresser à la technologie à objets, principalement pour son potentiel à augmenter la productivité des programmeurs. Comme le montre le nombre maintenant important d’expériences positives de sa mise en application, celle-ci s’est avérée en mesure d’améliorer de manière drastique le processus de développement.

Cependant, il apparaît qu’il est non seulement nécessaire d’utiliser la technologie à objets, mais qu’il est également important de s’intéresser à la manière de la mettre en œuvre, afin de réaliser pleinement les bénéfices qu’elle peut offrir.
En particulier, nous pensons que les frameworks -- qui peuvent a priori être envisagés comme des ensembles de classes d’objets intégrées permettant de réaliser des comportements calculatoires bien définis – constitue une des bases nécessaires pour voir se réaliser les plus belles promesses de la technologie à objets.

2.2 Limitations de la programmation procédurale

Dans un environnement de programmation procédurale, un programmeur écrit une application en faisant appel à des routines de bibliothèques fournies par le système, ou écrites au préalable par lui. Le code du programme développé s'appuie sur le code du système. Il peut accéder à tous les services du système, mais le système n'a aucune connaissance de celui-ci. Le programmeur est entièrement responsable de fournir l'architecture générale de l'application et le schéma de contrôle, ou schéma d'exécution de celle-ci. Le système ne fournit que l'opérationnalité.

Toutefois, les limitations de la programmation procédurale sont restées importantes, notamment en ce qui concerne :

- Les possibilités d'extension et de spécialisation des fonctionnalités d'un système.
- Celles de factorisation de fonctionnalités communes.
- L'interopérabilité de différents modules.
- Le coût important de la maintenance des systèmes développés.

Ces limitations se sont traduites par un faible niveau de réutilisation et de productivité. [1] [2] [3]

2.3 Programmation à objets

En opposition à la programmation procédurale, la programmation à objets est basée sur la notion de structures de données élémentaires, associées à des *méthodes* qui agissent sur les données [4] [5]. Son principe est de permettre de concevoir des *classes* d'objets représentant l'essentiel des caractéristiques d'un problème. Plutôt que d'essayer d'adapter un problème à l'approche procédurale, dans un langage informatique, la programmation à objets permet aux développeurs de modéliser et de représenter des éléments pertinents du monde réel, et d'y greffer ensuite le code de mise en œuvre des solutions aux problèmes qui y sont liés, et aussi de concevoir et de réaliser des programmes avec une meilleure productivité, la programmation à objets s'est avérée être une évolution bénéfique et significative par rapport aux techniques traditionnelles de programmation.

Cependant, même si la programmation est facilitée, notamment parce qu'elle permet de travailler à un niveau d'abstraction plus élevé, avec des objets et des bibliothèques de classes, elle ne permet pas de régler certains problèmes comme les suivants :

- Les développeurs sont toujours en charge de définir eux mêmes l'infrastructure d'une application.
- Ils ne disposent pas à priori de mécanisme élaboré pour définir des extensions de fonctionnalités.
- Ils doivent encore écrire beaucoup de code, notamment parce qu'ils ont à spécifier le schéma de contrôle de leur application.

2.4 Programmation par frameworks

Les frameworks sont aujourd'hui considérés par de nombreux spécialistes comme un des principaux avancements de la technologie à objets. Parce qu'ils mettent à disposition une infrastructure préétablie et une interface bien définie, ils permettent de pallier aux problèmes de la programmation à objets. Avec des frameworks bien conçus, il est en effet plus facile de développer des extensions, de mettre en facteur des fonctionnalités communes, de favoriser l'interopérabilité des composants, et d'améliorer la fiabilité et la maintenance des logiciels.

Les bénéfices que les frameworks permettent de réaliser sont liés à deux principes fondamentaux.

- Les frameworks définissent une infrastructure et représentent une conception.
- Ce sont les frameworks qui font en général appel au code du programmeur.

2.5 Principaux travaux de recherche sur les frameworks

Un certain nombre de travaux de recherche portent depuis la fin des années 80 sur la notion de frameworks au sens originel.

- Le pôle de recherche sur les frameworks le plus connu aujourd'hui est sans aucun doute celui du ***groupe Smalltalk de R. Johnson***, Ce groupe a contribué depuis quelques années au développement de plusieurs frameworks. Il mène des réflexions sur le développement de frameworks et en propose une vision systématisée (cf. la notion de ré-usinage d'applications : « refactoring »). Par ailleurs, il étudie depuis quelque temps les liens que les frameworks entretiennent avec les schémas de conception (design patterns). Il faut aussi citer comme principaux lieux d'activité sur les frameworks,
- le groupe de recherche sur les frameworks (***Frameworks Research Group***) mène des réflexions sur les liens éventuels entre frameworks et méthodes d'analyse / conception à objets.
- Le Groupe des Systèmes à Objets (***OSG***) a également travaillé sur les frameworks et a développé de l'outillage informatique pour leur mise en application, leur utilisation et leur composition.
- Des propositions commerciales ont déjà été envisagées, et la plus importante est sans conteste celle de Taligent .Le projet Taligent concerne l'élaboration d'un système d'exploitation et d'un environnement de développement entièrement basés sur les frameworks.

Les différents travaux en rapport avec les frameworks peuvent être classés suivant qu'ils concernent [6]

1. le *développement* de frameworks, et / ou
2. la *description* de frameworks, principalement en vue de faciliter leur utilisation, et / ou encore
3. leur *adaptation,* ou leur utilisation dans un développement logiciel spécifique.

2.6 Principaux Frameworks

2.6.1 MVC

MVC est un framework qui met en œuvre en Smalltalk 80 un paradigme d'implémentation d'interfaces utilisateur. C'est sans doute l'un des frameworks les plus référencés [7] [8].

MVC signifie *Model-View-Controller*. Ces trois notions définissent un modèle de la manière dont une interface utilisateur peut être réalisée. En Smalltalk 80, elles sont réifiées par trois classes abstraites (de mêmes noms) qui correspondent au noyau du framework. Un modèle MVC représente la structure interne d'une interface utilisateur. Il mémorise des données et réalise des comportements pour celle-ci. En général, il contient des liens vers des objets (du domaine) plus complexes de l'application à interfacer.

2.6.2 ATB (AckiaToolBox)

L'AckiaToolBox (pour ObjectWorks Smalltalk versions 2.5 à 4.1 [9]) est un framework constitué de classes utilitaires pour la définition synthétique d'interfaces utilisateur, respectant une norme de présentation précise, et essentiellement textuelles (à base de vues de liste, champs de saisie et autres composants élémentaires). Il est bien sur construit à partir du framework MVC standard et peut être vu comme en étant une extension. Il fournit des composants élémentaires branchables supplémentaires et complémentaires à ceux de la bibliothèque du MVC de base. Il comprend la mise en œuvre de contrôleurs particuliers pour des fenêtres de dialogues modales. Il comprend des réalisations d'autres contrôleurs pour gérer des événements en plus de ceux prévus par le MVC standard, notamment le double-clique à la souris. Il permet aussi de spécifier dynamiquement des boîtes de dialogue utilisateur dans un langage déclaratif.

2.6.3 Éditalk

Éditalk est un framework qui permet la création d'éditeurs de réseaux sophistiqués selon le paradigme MVC. Toutes les notions telles que les ancrages, les labels, les arcs contraints, etc. y sont traitées. L'utilisateur dispose de plus d'outils pouvant générer très rapidement des éditeurs d'un haut degré de complexité [10].

2.6.4 Actalk (pour la réalisation de systèmes multi-agents)

Le système Actalk est généralement présenté comme une plate-forme d'expérimentation de langages d'acteurs. En effet, différentes extensions de son « noyau » ont été élaborées pour implanter en Smalltalk les principaux modèles de langages d'acteurs de la littérature. Cependant, Actalk a aussi été utilisé pour réaliser des systèmes multi-agents de simulations. À ce titre, le système Actalk peut donc être considéré comme un framework de mise en œuvre de programmes de simulation multi-agents.

2.6.5 The Simulation Framework

Le *Simulation Framework* est sans doute, avec le framework MVC, un des premiers frameworks technologiques les mieux formalisés. Il propose un paradigme de réalisation en Smalltalk de programmes de simulations dirigées par les évènements (qui est un type de simulations à évènements discrets. Probablement du fait de son utilité moins nécessaire que le MVC, il

n'a pas connu le même engouement que celui-ci, et il a été plus ou moins oublié. Cependant, nous pensons que son noyau est constitué de manière aussi résistante que le MVC [11].

2.7 Démarche utilisé pour la réalisation de «Phélix»

La démarche suivie pour la réalisation de ce framework est de type ascendant, à partir des deux applications choisie (recrutement, gestion de congé) qui utilisent des workflow et après une étude détaillée, on a réussi de définir les principaux points communs qui nous aident à réaliser ce framework.
Le schéma suivant résume la démarche suivie :

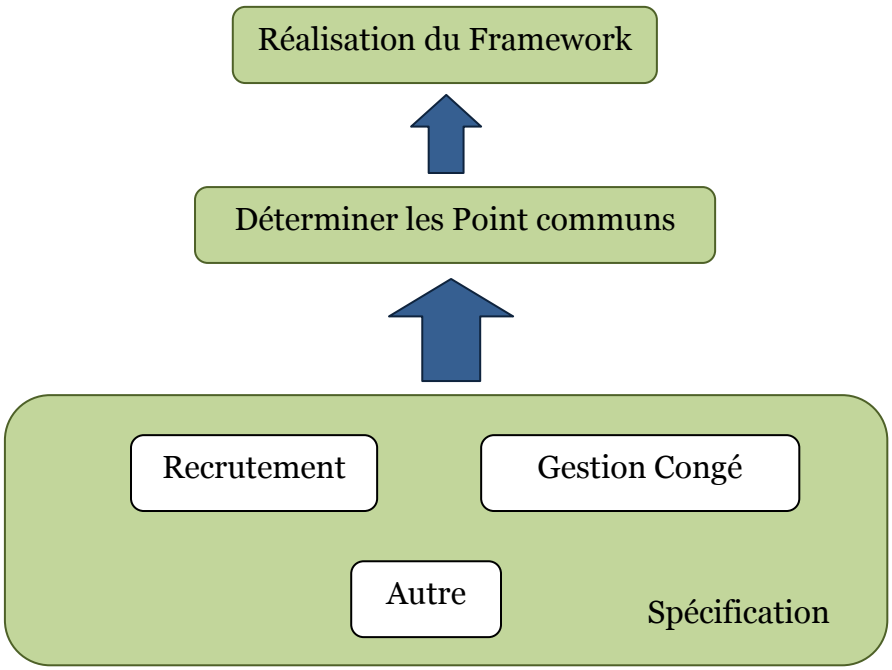

Figure 2.1 Démarche du framework

2.7.1 Spécifications globales de deux applications

La gestion de congé consiste à traiter les déférentes demandes déposées par les employés de l'entreprise, après une validation de la demande une acceptation ou un regret sera notifié. Le processus de recrutement nécessite la publication des annonces d'offres d'emploi, l'examen des candidatures reçues sous forme d'une lettre de motivation et d'un CV, par courrier ou par internet, ensuite la sélection sur CV, puis la sélection sur entretien, et enfin une proposition d'embauche au candidat sélectionné.

2.7.2 Déterminer les points communs

Sachant bien que nos deux applications appartiennent au domaine de la gestion du personnel nous percevons des points communs qui réunissent ces dernières et qui nous ont permis de concevoir et de réaliser le framework «Phélix».

2.7.3 Spécifications de l'architecture de «Phélix»

D'un point de vue technique, afin de concevoir «Phélix» nous utiliserons un ensemble d'entités logicielles réutilisables éventuellement extensibles et adaptables comme les classes les bibliothèques, les hiérarchies de classes, et les design patterns.
Concernant les techniques de réutilisations orientées objets nous utiliserons quartes schémas de réutilisation : la spécialisation de méthodes, la redéfinition partielle, l'adaptation par spécialisation et l'adaptation par composition qui sont tous des techniques (qui peuvent être combinées) appliquées à un framework pour le paramétrer. Enfin nous déciderons des points d'extensions et de paramétrages appelés aussi Hot spot, c'est à partir de ces points que le code de la nouvelle application sera appelé par le Framework.

2.8 Conclusion

Dans ce chapitre, on a présenté l’Etat de l’art des framework en citant les limitations de la programmation procédurale, la naissance des framework a partir de la programmation objet, on a aussi mis l’accent sur quelques framework qui existent sur le marché, ensuite on a expliqué la démarche suivie pour réaliser «Phélix».

Chapitre III

Le Framework

3.1 Introduction

Ce chapitre présente l'architecture du framework, dans un premier temps présentation des classes abstraites, puis les classes concrètes et enfin l'implémentation de ces classes.

3.2 Les classes abstraites

Suite à la réalisation des workflows '' Recrutement'' et ''Demande de congés'', nous avons constaté des abstractions qui concernent les deux applications et généralement tous les processus métier que nous avons étudié.
Cette analyse nous a aidés à élaborer le modèle statique ci-dessous, qui représente les classes abstraites et leurs associations.
Le workflow est composé de plusieurs tâches et d'un manager qui crée des acteurs. Ces derniers peuvent utiliser les différentes tâches pour *créer, mettre à jour ou chercher un objet métier.*

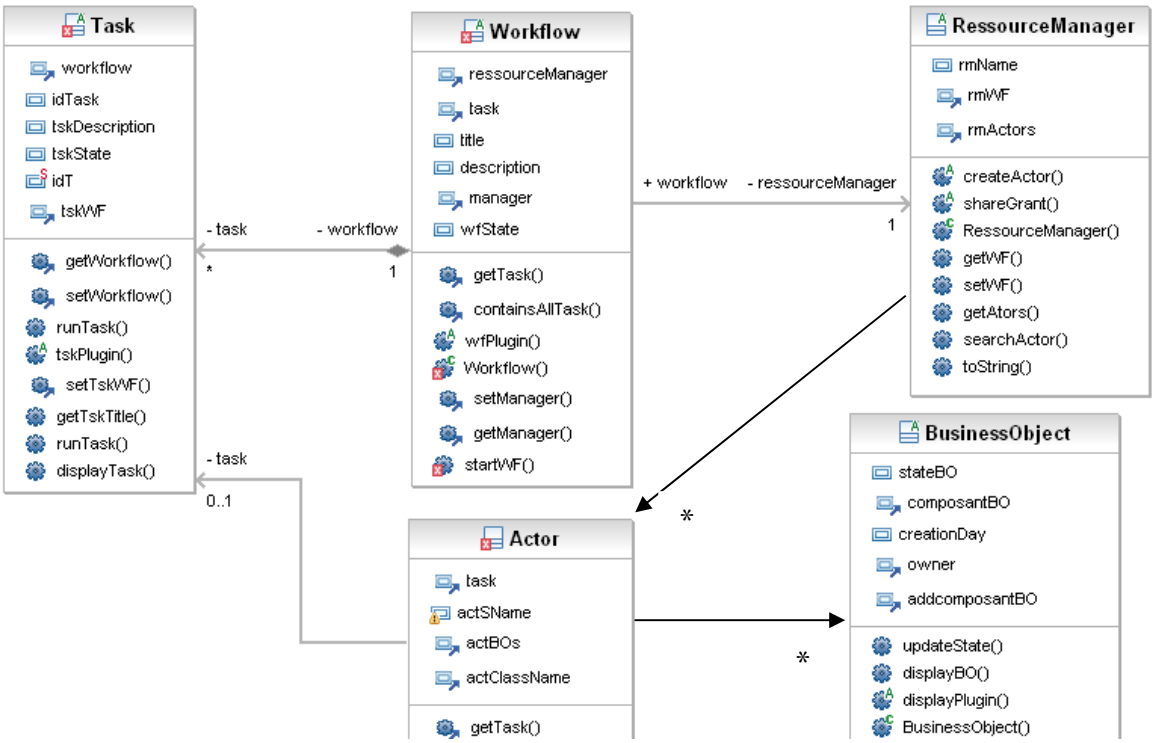

Figure : modèle version 1

3.2.1 BusinessObject

- Un objet métier est une abstraction ayant un sens pour des acteurs (partie prenante interne) de l'entreprise. Il permet de décrire les entités manipulées par les acteurs dans le cadre de la déscription du métier.
Exemple : "Mon métier consiste à gérer des CV et les entretiens des candidats"
Objet métier : CV, entretien.

- Les attributs sont des propriétés fixes ou variables portées par l'objet. Par exemple, l'objet Voiture possède l'attribut Age qui définit son âge.

- Les relations avec les autres objets métiers portent une information complémentaire à l'objet métier. Par exemple, l'objet candidat est caractérisé par une relation avec l'objet CV.

- Le comportement de l'objet métier décrit les actions et les réactions lorsqu'il est soumis à une opération. Il désigne la manière dont ses relations et ses attributs changent lors des interactions avec les autres objets métier au sein, par exemple, de Processus.
Exemple : L'attribut Etat de l'objet Candidat prend la valeur en entretien lorsqu'un entretien est prévu.

- L'identité ou identifiant. Cette identité le distingue des autres objets métiers qui peuvent être égaux sans toutefois être identiques. L'identité de l'objet métier est indépendante de son état.

3.2.2 Task

Une tâche est un concept abstrait qui défini une action ou une activité devant être réalisée par un acteur dans le cadre de son entreprise sur son système d'information.
Les principales tâches que nous avons recensées sont :

- La création
- La mise à jour
- La recherche d'un objet métier.

Exemple :
Les tâches d'un employé sont les suivantes :
1ère étape : Créer un candidat,
2ème étape : Créer un entretien avec un état initial « prévu ».
3ème étape : Après la réalisation effective de l'entretien, il met à jour le bilan de ce dernier.

Dans le même processus le Directeur cherche un candidat pour consulter ces entretiens et éventuellement créer une proposition d'embauche.

Cette classe dispose d'un point de paramétrage dans la méthode runTask() qui est surchargé pour prendre en compte les paramètres des trois tâches concrétées: la création, la mise à jour et la recherche d'un objet métier.

3.2.3 Actor

Un acteur est une abstraction d'une entité physique « employé » ou morale « service ou département » qui accomplit une tâche dans le cadre d'un processus métier.

L'association directe entre la classe Actor et Task signifie qu'un acteur peut effectuer les trois types de tâches qui sont la création, la mise à jour ou la recherche d'un objet métier.

Ainsi un Responsable des Ressources Humaines qui est un acteur, pourra étudier la demande de congé d'un employé et lui accorder des vacances en validant sa demande de congé préalablement créée.

Cette classe contient une collection d'objets métier qui sont manipulés par l'acteur qui les a créés.

3.2.4 RessourceManager

Un manager est une personne ou un système dont le rôle est de créer les acteurs du processus et leur donner leurs droits sur des objets métier.

Par exemple le responsable du recrutement manipule les objets ‘’ candidats’’ et ‘’ entretiens’’ alors que le responsable ressources humaines manipule les objets ‘’ demandes de congé’’ et ‘’ fiche de paye ‘’ etc....
Le manager fait en sorte que chaque acteur puisse manipuler les objets qui sont dans son métier.
La classe Actor contient une collection de noms de classes héritières de BusinessObject qui peuvent être manipulées par cet acteur.
La classe RessourceManager contient une collection d’acteurs du processus et deux méthodes abstraites createActor() pour créer les acteurs, shareGrant() pour distribuer les droits. Ces deux méthodes doivent être définies dans la classe qui spécialise RessourceManager.

3.2.5 Workflow

Un worklow est un processus métier automatisé. Ce dernier présente un ensemble de tâches réalisées par des acteurs qui s'enchaînent de manière chronologique afin d’atteindre un objectif, généralement qui est de délivrer un produit ou un service, dans le contexte d'une entreprise.
Le worklow est composé de plusieurs tâches et d'un manager qui crée les acteurs et leur distribue des droits.
La classe abstraite Workflow possède la méthode startWF() qui initialise le workflow. Elle contient un point d'extension wfPlugin() adaptable par spécialisation qui retourne un objet instance de la classe concrète héritière de la classe RessourceManager, et deux autres points de paramétrage adaptables par composition de l'attribut de type RessourceManager qui possède les deux méthodes abstraites createActor(), shareGrant().

3.3 Les classes concrètes

La particularité de cette deuxième version est l'ajout de la classe Role pour regrouper les tâches concrètes CreateBO, UpdateBO et SerachBO. Et aussi la classe Employe qui bénéficie des caractéristiques de la classe mère Actor.

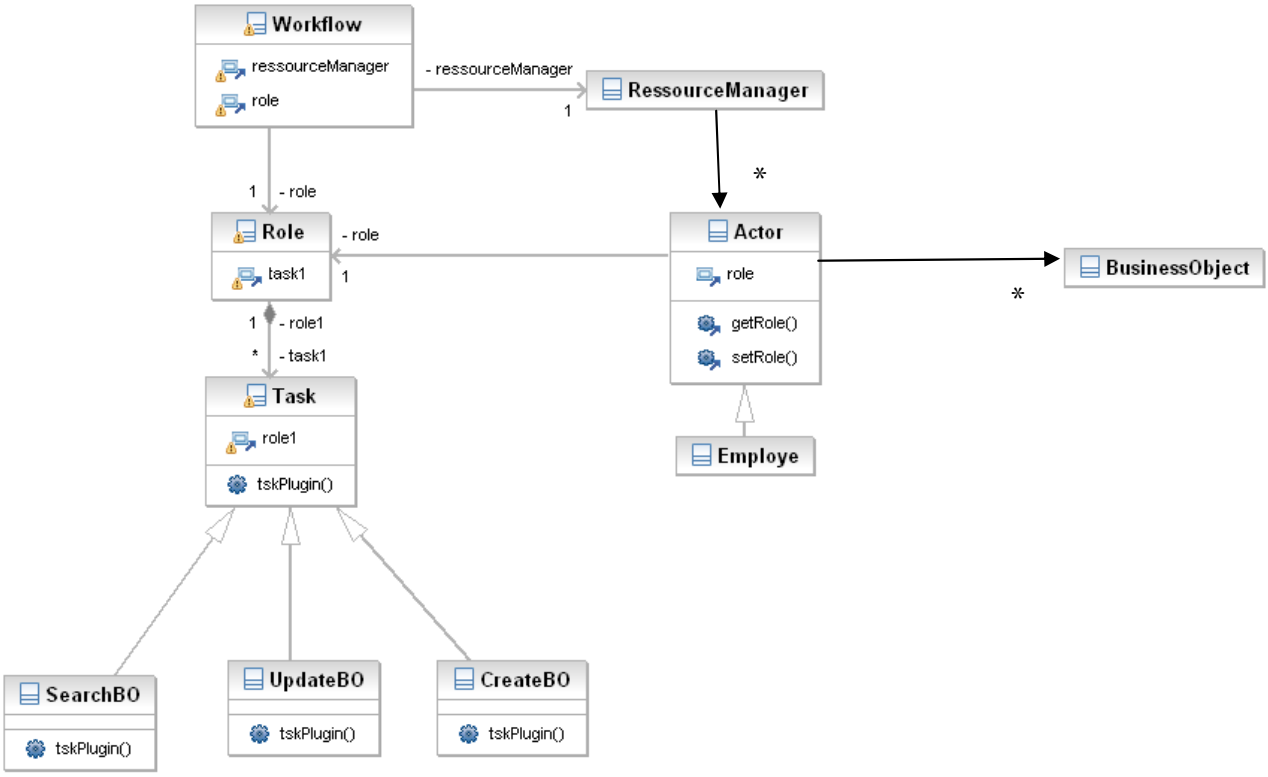

Figure : modèle version 2

3.3.1 CreateBO

Héritière de la classe Task, elle définit la méthode tskPlugin() qui prend en paramètre le nom de la classe héritière de BusinessObject et les arguments nécessaires à l'appel du constructeur de la classe Candidat par exemple ou de n'importe quelle autre classe qui hérite de BusinessObject. Cette classe prend un dernier paramètre de type Actor qui représente l'acteur qui a fait appel à cette tâche. Le but de la méthode tskPlugin() définie dans cette classe est d'utiliser la méthode newInstance(arglist) de la classe Class pour appeler le constructeur de l'objet métier qui va retourner un objet de type Object. Ce dernier est casté vers le type BusinessObject et ajouté à la collection des objets métier de l'acteur qui possède les droits de création sur ce type d'objet. Dans le cas contraire une exception est levée indiquant que l'acteur n'a pas la permission de créer ce type d'objet métier.

3.3.2 UpdateBO

Deuxième héritière de la classe Task, elle définit aussi la méthode tskPlugin(). Cette fois-ci elle prend en paramètre l'objet métier à mettre à jour et une table associative attribut/valeur qui contient les attributs de l'objet métier qui vont prendre les nouvelles valeurs. TskPlugin() parcourt d'abord les attributs communs de tous les objets métier c'est-à-dire de la classe BusinessObject et les met à jour.
Signalons que les attributs de la classe BusinessObject qui sont boOwner pour l'acteur qui crée l'objet, boCreationDay pour la date de création, ne sont pas des attributs modifiables car ils ne possèdent aucun accesseur.
Ensuite la méthode parcourt les attributs de la classe héritière de BusinessObject pour les mettre à jour avec les nouvelles valeurs de la table associatives passé en argument. De même pour cette classe une exception est levée si l'acteur n'a pas la permission de mettre à jour ce type d'objet.

3.3.3 SearchBO

La dernière classe héritière de la classe Task, définit la méthode tskPlugin(). Elle prend en paramètre une collection d'objets métier et un identifiant. La méthode parcourt la collection d'objets métier et retourne l'objet qui possède l'identifiant passé en paramétre dans le cas échéant une exception est levée indiquant que l'objet est introuvable.

3.3.4 Role

Comme nous pouvons le voir sur le modèle cette classe est composée des tâches concrètes CreateBO, UpdateBO et SearchBO qui héritent de la classes Task. Elle sert à regrouper l'ensemble des tâches nécessaires dans le workflow.
Dans la première version du modèle l'association est entre Actor est Task, alors que dans la deuxième elle est entre Actor et Role.
Enfin la classe Workflow contient un attribut de type Role qui est instancié une seul fois dans la méthode startWF() puis les tâches qui sont créées sont regroupées dans ce rôle. Lors de la création d'un acteur par un manager l'attribut actRole de la classe Actor prend comme valeur l'objet instance de la classe Role.

Toutes ces classes sont instanciées une seul fois dans la méthode startWF() de la classe Workflow.

3.3.5 Employe

Cette classe hérite de la classe Actor et bénéficie de toutes les propriétés de cette dernière c'est-à-dire qu'un employé manipule un ensemble d'objets métier à travers des tâches qui sont regroupées dans son rôle. Ainsi un employé peut créer, mettre à jour ou rechercher les objets de son métier, et il possède des droits sur un ensemble d'objets métier qu'il peut manipuler. Les employés sont instanciés dans la méthode createActor() de la classe abstraite RessourceManager.

3.4 Implémentation et Points de paramétrage ou d'extension

3.4.1 BusinessObject

Etat

```
private Actor  owner = null;
private String creationDay = "14/05/2009"; //date du jour
private String stateBO = "BO-created";
protected Collection<BusinessObject> composantBO = null;
```

Un objet métier quelconque a un acteur qui l'a créé, une date de création et un attribut qui représente l'état que l'objet métier peut prendre durant sa durée de vie. Enfin un objet métier peut être un objet composite qui contient d'autres objets métier, par exemple l'objet Candidat contient un objet CV.

Comportement

```
public void setOwner(Actor a);
```

Affectation de l'acteur a à l'attribut owner.

```
public Collection<BusinessObject> getComosantBO();
```

Retourne la collection d'objets métier composite d'un objet métier.

```
public void updateState(String s);
```

Met à jour l'état d'un objet métier.

```
public void addcomposantBO(BusinessObject bo);
```

Ajoute un objet métier composite à l'objet métier courant.

```
public void displayBO()
```

Affiche l'objet métier et les objets métier qui le composent. La signature de cette méthode sera changée dans la prochaine version en String toString()

Points d'extension

```
abstract public int getIdPlugin();
public int getId(){
        return this.getIdPlugin();
}
```

Paramétrage pour récupérer l'identifiant d'un objet métier

```
abstract public void displayPlugin();
public void displayBO(){
System.out.println(this.owner+" - "+this.creationDay+" - "+this.stateBO);
System.out.println("----");
        this.displayPlugin();
        Iterator<BusinessObject> i = this.composantBO.iterator();
        int j = 0;
        while (i.hasNext()){
                System.out.println("* Composant "+j+":");
                i.next().displayBO();
                j++;
        }
}
```

La signature de la méthode displayPlugin() sera changée également dans la prochaine version, nouvelle signature String toString()

3.4.2 Task

Etat

```
private int idTask;
private String tskTitle = null;
private String tskDescription = null;
private String tskState;
private static int idT = 0;
```

Une tâche est représentée par un identifiant, un titre, sa description et son état.

Points d'extension

```
public abstract BusinessObject tskPlugin(String ncls, Object[] consArgs,
                                Collection<BusinessObject> cbo, int id,
                                BusinessObject boarg,
                                        HashMap<Object,Object> hmAttr,
                                Actor a);
```

Surcharge de la méthode runTask() :

-Pour la recherche d'un objet métier

```
/**reçoit une collection d'objets métier et un identifiant**/
public BusinessObject runTask(Collection<BusinessObject> cbo, int id,
                Actor a)
{
        BusinessObject bo = tskPlugin(null,null,cbo,id,null,null,a);
        this.tskState = "finished";
        return bo;
}
```

-Pour la création d'un objet métier

```
/** reçoit le nom de la classe et les arguments du constructeur**/
public BusinessObject runTask(String ncls, Object[] consArgs, Actor a)
{if(this.tskWF.getManager().searchActor(a.getActName()).
  searchClassName(ncls))
 {
        BusinessObject bo = tskPlugin(ncls,consArgs,null,0,null,null,a);
        this.tskState = "finished";
        return bo;
}
        this.tskState = "failed";
        System.out.println("pas de permission pour créer ce type d'objet
          métier");
        return null;
}
```

Le test effectué à l'entrée de la méthode assure que l'acteur qui a appelé la tâche de création a bien le droit de créer le type d'objet métier qui porte le nom ncls.

-Pour la mise à jour d'un objet métier

```
/**reçoit L'objet métier à mettre à jour et une table associative
"attribut/valeur" **/
public BusinessObject runTask(BusinessObject boarg,HashMap<Object,Object>
```

```
                    hmAttr, Actor a){
            if(this.tskWF.getManager().searchActor(a.getActName()).
                    searchClassName(boarg.getClass().getName()))
{
            BusinessObject bo = tskPlugin(null,null,null,0,boarg,hmAttr,a);
            this.tskState = "finished";
            return bo;
    }
    this.tskState = "failed";
    System.out.println("pas de permission pour mettre à jour ce type d'objet
            métier");
    return null;
    }
```

3.4.3 Actor

Etat

```
private String actFName = null;
private String actSName = null; //Ajouter les attributs.
private Role aRole = null;
private Collection<BusinessObject> actBOs = new
                                ArrayList<BusinessObject>();
private Collection<String> actClassName = new ArrayList<String>();
```

Un acteur possède un nom et un prénom, un rôle qui regroupe l'ensemble des tâches du workflow, une collection d'objets métier ainsi qu'une autre collection qui contient les noms de classes des objets métier qu'il peut créer et mettre à jour.

Comportement

Méthode pour manipuler la collection des noms de classe actClassName:

```
public String addClassName(String cn);
```

Ajout d'un nom de classe d'un objet héritier de la classe BusinessObject.

```
public void removeClassName(String cn);
```

Suppression d'un nom de classe.

```
public boolean searchClassName(String cn);
```

Recherche d'un nom de classe, retourne vrai si le nom de la classe existe dans la collection et faux sinon.

Ces trois méthodes sont appelées par le manager dans la méthode shareGrant().

3.4.4 RessourceManager

Etat

```
protected String rmName = null;
private Workflow rmWF = null;
protected Collection<Actor> rmActors = new ArrayList<Actor>()
```

Un manager possède un nom, le workflow auquel il appartient et une collection d'acteurs qui contient les acteurs créés.

Comportement

```
public Actor searchActor(String name);
```

Recherche d'un acteur par son nom dans la collection rmActors

Points d'extension

```
abstract public void createActor();
```

La classe RessourceManager doit être spécialisée pour définir la méthode createActor() qui va créer les différents acteurs du workflow, par exemple :

```
Actor a1 = new Employe( nom prénom);
this.rmActors.add(a1);

abstract public void shareGrant();
```

De même la méthode shareGrant doit être définie dans la classe qui spécialise RessourceManager. Elle donnera aux acteurs préalablement créés le droit de créer certains objets métier dont leurs classes héritent de la classe BusinessObject, par exemple :

```
this.searchActor(nom).addClassName(recrutement.Entretien.class.getName());
```

Ce qui signifie que l'acteur ‘’nom’’ pourra créer et mettre à jour les objets de type ‘’Entretien’’ qui hérite de la classe BusinessObject

3.4.5 Workflow

Etat

```
private String title;
private String description;
private Role wfRole = null;
private RessourceManager manager = null;
private String wfState =  "WF created";
```

Un workflow est représenté par un titre, une description, un rôle qui contient l'ensemble des tâches, et un manager.

Comportement

```
public void startWF();
```

Cette méthode initialise le workflow en créant un rôle unique et les tâches nécessaires comme la création, la mise à jour et la recherche d'un objet métier.

Points d'extension

```
abstract public RessourceManager wfPlugin();

public void startWF(){
this.wfRole = new Role("role");
  this.wfTasks.add(new CreateBO("c","Create businessobject",
                                        this.wfRole,this));
  this.wfTasks.add(new UpdateBO("u","Update business object",
                        this.wfRole,this));
  this.wfTasks.add(new SearchBO("s","Search business object",
                        this.wfRole,this));
  this.wfRole.setRoleTasks(this.wfTasks);
  this.manager = this.wfPlugin();
  /**Workflow est composé d'un manager**/
  this.manager.createActor(); /**Adaptation par composition**/
  this.manager.shareGrant();
 }
```

La méthode wfPlugin() doit être définie dans le workflow concret comme le ‘’recrutement’’ ou ‘’la demande de congé’’, elle retourne une instance de la classe qui spécialise RessourceManager, l'instance est affectée à l'attribut manager. Puisque le workflow est composé d'un manager les deux méthodes createActor() et shareGrant() de ce dernier sont appelées pour créer les acteurs et leur attribuer des droits.

3.5 Conclusion

Nous avons vu l’architecture de Phélix, et son implémentation, ainsi dans le prochain chapitre nous verrons comment paramétrer le framework pour générer des workflows.

Chapitre IV

Paramétrage du Framework

4.1 Introduction

Dans ce chapitre, on spécifie les deux applications étudiées (gestion de congés et recrutement) en détaillant les deux processus métiers et on montre les points de paramétrage en donnant quelques exemples.

4.2 Spécifications du workflow recrutement

Le processus de recrutement a pour objectif de sélectionner et d'embaucher les profils qui permettront à l'entreprise de réaliser ses objectifs stratégiques de croissance. Il est composé de trois étapes qui sont la sélection sur CV, sur entretien et la proposition d'embauche aux candidats sélectionnés. Le responsable de recrutement est chargé de la sélection sur CV, le responsable entretien est l'employé qui sélectionne les candidats après la réalisation de l'entretien et enfin le directeur qui fait la proposition d'embauche aux candidats sélectionnés sur entretien. A la demande du directeur un dernier entretien est réalisé avec les candidats avant de lui faire une proposition.

4.2.1 Modélisation du recrutement

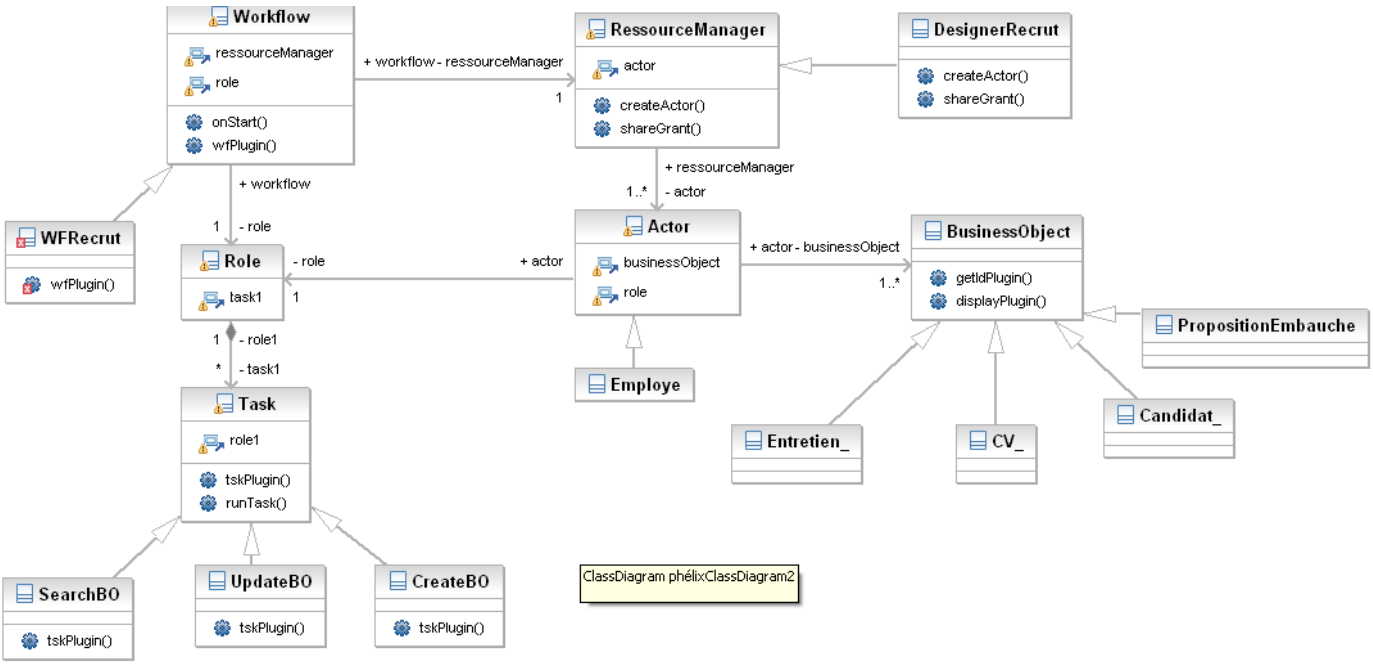

Fig 4.1 Diagramme de classes recrutement

- **La Classe WFRecrut**

Cette classe concrétise la classe abstraite Workflow en définissant la méthode *wfPlugin ()* qui retourne une instance de la classe DesignerRecrut, cette instance est affectée à la variable d'instance manager de l'objet instance de la classe **WFRecrut.**

- **La Classe DesignerRecrut**

Héritière de la classe RessourceManager, elle définie les méthodes *createActor ()* et *share-Grant ()* qui respectivement vont créer les acteurs et leur attribuer les droits sur les objets métiers qui héritent de BusinessObject.

4.2.2 Implémentation et paramétrage

- **WFRecrut**

```
public class WFrecrutement extends Workflow {

        public WFrecrutement(String t, String d) {
        //Le constructeur prend en arguments le titre du workflow et sa                //description
                super(t,d);

        }
```

```
        @Override
        public RessourceManager wfPlugin() {
                RessourceManager r = new DesignerRecrut("Manager du recrutement");
                r.setWF(this);
                return r;
        }
}
```

Cette méthode crée et retourne une instance de la classe DesignerRecrut. L'instance créée connait son workflow, donc le manager du recrutement va gérer le workflow recrutement.

❖ DesignerRecrut

```
public class DesignerRecrut extends RessourceManager {

        public Designer(String n){
                super(n);

        }
        public void setWF(){
                this.getWF().setManager(this);
                // le workflow connait sont manager
        }
        public void createActor() {
                //création des acteurs et affectation des rôles
                Actor a1 =  new Employe("Fred","Dust","Responsable recrut",2000);
                a1.setRole(this.getWF().getRole());
                Actor a2 =  new Employe("Alice","Dim","Responsable entretien",2000);
                a2.setRole(this.getWF().getRole());
                Actor a3 =  new Employe("Jean","Trio","Responsable entretien",4500);
                a2.setRole(this.getWF().getRole());
                //ajout dés acteurs crées à la collection rmActors du manager
                this.rmActors.add(a1);
                this.rmActors.add(a2);
                this.rmActors.add(a3);
        }
        public void shareGrant() {
                Actor a ;
                //fred peut manipuler des objets métier de type Candidat et CV
                a = this.searchActor("Fred");
                a.addClassName("recrutement.Candidat");
                a.addClassName("recrutement.CV");
                //Alice peut manipuler des objets métier de type Entretien
                a = this.searchActor("Alice");
                a.addClassName("recrutement.Entretien");
                //Jean peut manipuler des objets métier de type Entretien Prposition
                a = this.searchActor("Jean");
                a.addClassName("recrutement.PropositionEmbauche");
        }
```

```
}
```

❖ Client

1. *Création et initialisation du workflow recrutement*

```
Workflow wfr = new WFrecrutement("Recrutement","WF recrutement");
wfr.startWF();
```

Tâches et rôle crées
Le manager Designer va gérer le workflow
Les acteurs sont crées avec des droits sur les objets métier
Le workflow est initialisé

2. *Récupération d'un acteur par son nom*

```
Actor fred = wfr.getManager().searchActor("Fred");
//le type réel de fred est Employe
```

3. *Création et affichage d'un Candidat*

```
Object[] cndArgs = new Object[20];
cndArgs[0] = "mehdi"; cndArgs[1] = "mouh"; cndArgs[2] = "24";
cndArgs[3] ="adresse casa";
BusinessObject cndMehdi = fred.getRole().SearchTsk("c").
    runTask(Candidat.class.getName(), cndArgs, fred);
cndMehdi.displayBO();
```

Résultat

```
phelix.Employe@190d11 - 14/06/2009 - BO-created
----
10
Jeremy
Norman
24
570, route de Ganges, Montpellier
```

4. *Ajout d'un Objet métier composant CV à l'objet composite Candidat*

```
Object[] cvArgs = new Object[20];
cvArgs[0] = "Expert J2EE"; cvArgs[1] = "experience..."; cvArgs[2] =
        "formation ...";
BusinessObject cv = fred.getRole().SearchTsk("c").
                        runTask(CV.class.getName(), cvArgs,fred);
cndMehdi.addcomposantBO(cv);
```

Résultat

```
phelix.Employe@190d11 - 14/06/2009 - BO-created
----
10
Jeremy
Norman
24
570, route de Ganges, Montpellier
* Composant 0:
phelix.Employe@190d11 - 14/06/2009 - BO-created
----
20
poste
experience
formation
```

5. Mise à jour d'un entretien

```
HashMap m = new HashMap<Object,Object>();
m.put("stateBO", "Réalisé");
m.put("entBilan", "Très bon candidat");
fred.getRole().SearchTsk("u").runTask(entMehdi1, m, fred)
```

Résultat

```
phelix.Employe@190d11 - 14/06/2009 - Réalisé
----
400
12/12/2012
Très bon candidat
```

6. Recherche et affichage d'un candidat

```
fred.getRole().SearchTsk("s").runTask(fred.getActBOs(), 10, fred).displayBO();
```

Résultat

```
phelix.Employe@addbf1 - 14/06/2009 - BO-created
----
10
Jeremy
Norman
24
570, route de Ganges, Montpellier
```

4.3 Spécifications du workflow Congé

Le processus de traitement d'une demande de congés est composé de 5 étapes principales qui sont les suivantes :

- Saisie
- Validation
- Acceptation
- Mise à jour
- Notification

Dès lors qu'un employé émet une demande de congé, le processus lui demande de saisir les informations correspondantes. Les informations sont tout d'abord validées par le gestionnaire de congés, en accord avec les règles de l'entreprise. La demande est ensuite transmise au responsable hiérarchique de l'employé, lequel accepte ou refuse la demande. Une demande acceptée est enregistrée auprès du gestionnaire de congés. Enfin, quelque soit le résultat, il est communiqué à l'employé.

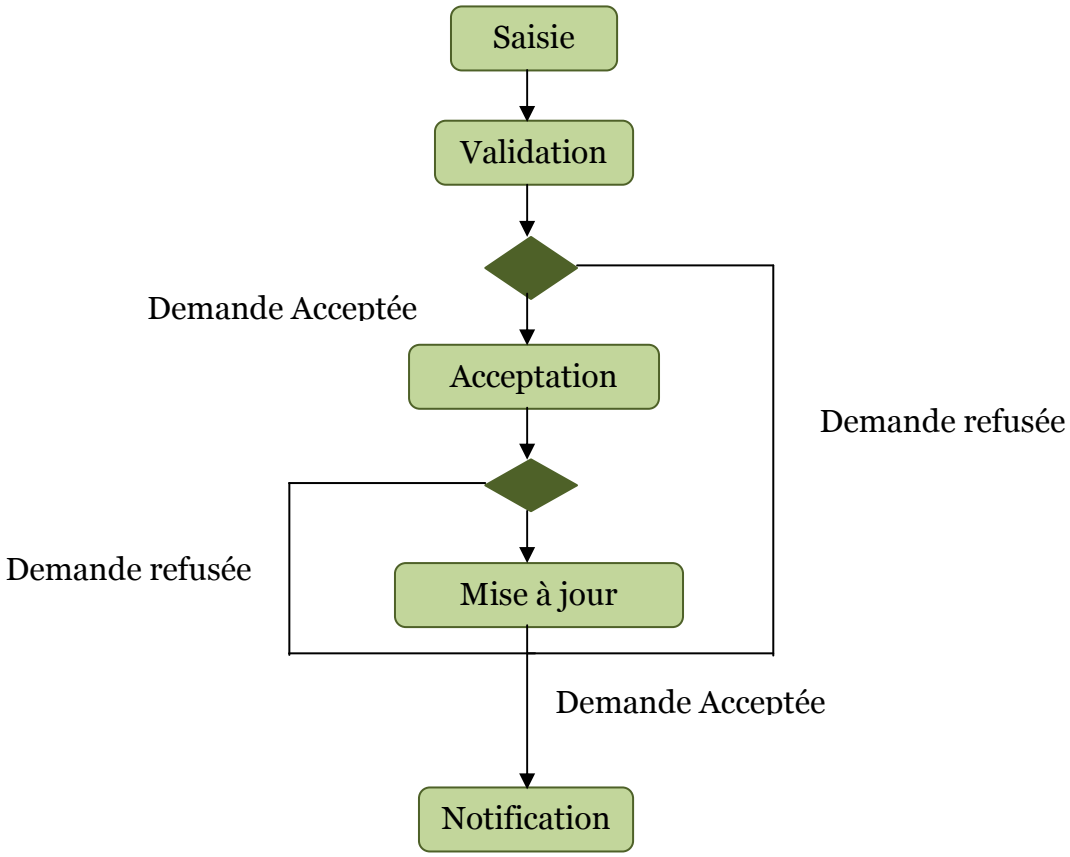

Fig 4.2 Processus demande Congé

4.3.1 Modélisation du Congé

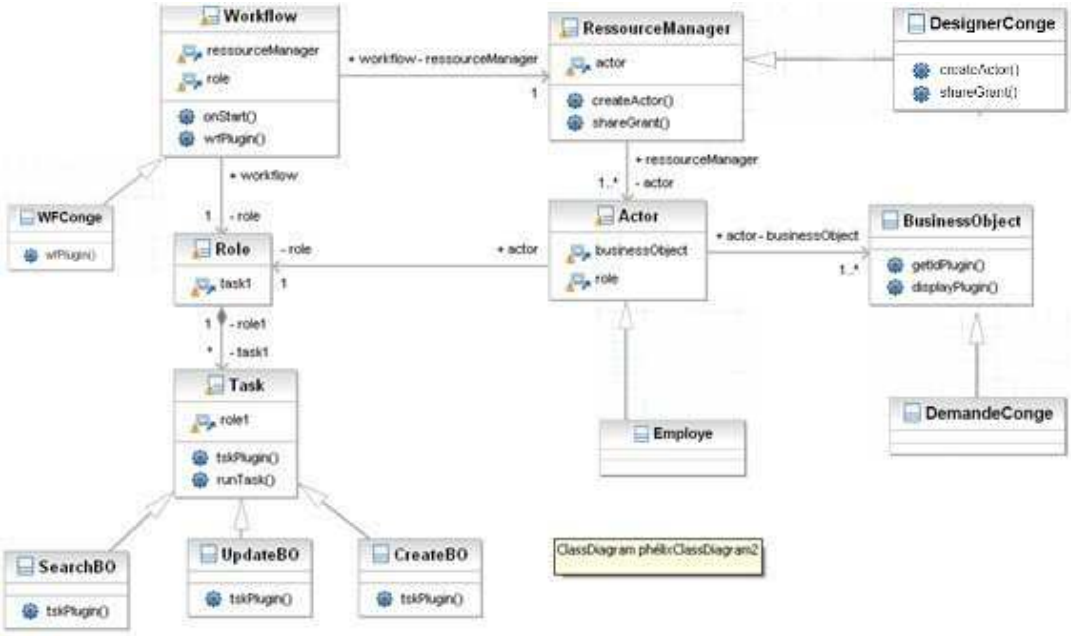

Fig 4.3 Diagramme de classes gestion congé

Comme dans le processus de recrutement le processus de demande de congé nécessite :

- **La Classe WFConge**

Cette classe concrétise la classe abstraite Workflow en définissant la méthode *wfPlugin ()* qui retourne une instance de la classe DesignerConge, cette instance est affectée à la variable d'instance manager de l'objet instance de la classe **WFConge.**

- **La Classe DesignerConge**

Définit les méthodes *createActor ()* et *shareGrant ()* qui respectivement vont créer les acteurs et leur attribuer les droits sur les objets métiers qui héritent de BusinessObject.

4.3.2 Implémentation et paramétrage

- **WFConge**

```
public class WFConge extends Workflow {

	public WFConge(String t, String d) {
	//Le constructeur prend en arguments le titre du workflow et sa 		//description
		super(t,d);
	}
	@Override
	public RessourceManager wfPlugin() {
		RessourceManager r = new DesignerConge("Manager Demande de congé");
		r.setWF(this);
		return r;
	}
}
```

- **DesignerConge**

```
public class DesignerConge extends RessourceManager {

	public Designer(String n){
		super(n);

	}
	public void setWF(){
		this.getWF().setManager(this);
		// le workflow connait sont manager
	}
	public void createActor() {
		//création des acteurs et affectation des rôles
	Actor a1 =  new Employe("ali","badou","ResponsableHiérarchique",2000);
		a1.setRole(this.getWF().getRole());

		Actor a2 =  new Employe("Mouloud","Achour","Manager",2000);
		a2.setRole(this.getWF().getRole());

		//ajout dés acteurs crées à la collection rmActors du manager
		this.rmActors.add(a1);
		this.rmActors.add(a2);
	}
	public void shareGrant() {
		Actor a ;
		//Mouloud peut manipuler des objets métier de type validation et Notification
		a = this.searchActor("Mouloud");
		a.addClassName("Conge.DemandeConge ");
		//Ali peut manipuler des objets métier de type Acceptation Mise à jour
		a = this.searchActor("Ali");
		a.addClassName("Conge.DemandeConge ");
```

```
        }
}
```

Client

1) ***Création et initialisation du workflow Congé***

```
Workflow WFC = new WFConge("Conge","WFConge");
WFC.startWF();
```

Tâches et rôle crées
Le manager Designer va gérer le workflow
Les acteurs sont crées avec des droits sur les objets métier
Le workflow est initialisé

2) ***Récupération d'un acteur par son nom***

```
Actor Ali = WFC.getManager().searchActor("Ali");
//le type réel de Ali est Employe
```

3) ***Création et affichage d'une demande de congé***

```
Object[] DemCongArgs = new Object[20];
DemCongArgs [0] = "000675"; DemCongArgs [1] = "Kendri"; DemCongArgs
[2] = "salim";  DemCongArgs [3] ="14/04/2009"; DemCongArgs [3]
="29/04/2009";
BusinessObject DemCongYan = Yan.getRole().SearchTsk("c").
runTask(DemandeConge.class.getName(),DemCongArgs, Yan);
DemCongYan.displayBO();
```

Résultat

```
phelix.Employe@190d11 - 14/06/2009 - BO-created
----
000675
Kendri
Salim
14/04/2009
29/04/2009
```

4) ***Mise à jour d'une Demande de congé***

```
HashMap m = new HashMap<Object,Object>();
m.put("stateBO", "Accepteé");
Ali.getRole().SearchTsk("U").runTask(DemCongeYan, m, Ali)
```

Résultat

phelix.Employe@190d11 - 14/06/2009 - Réalisé

00374
14/04/2009
Accepté

5) Recherche et affichage d'une demande de congé

```
Ali.getRole().SearchTsk("s").runTask(Ali.getActBOs(),0,Ali).displayBO();
```

Résultat

phelix.Employe@190d11 - 14/06/2009 - Réalisé

00374
14/04/2009
Accepté

4.4 Conclusion

Dans ce chapitre, on a présenté une démarche qui traite les différentes étapes de spécification des deux processus (recrutement et congé) en donnant des exemples sur les points de paramétrage.

Chapitre V

TESTS

5.1 Introduction

Les tests ont une grande importance. Tous les développeurs de logiciel le savent, mais (presque) aucun n'en fait. Le développeur se construit-il une batterie de préjugés et de bonnes raisons qui le confortent dans sa réticence à l'égard des tests et elles sont pratiquement toutes basées sur l'hypothèse que les tests coûtent du temps.
Si l'on croit à cela, on rentre dans un cercle vicieux : plus la livraison du logiciel est urgente, moins il y'a de tests. Moins il y'a de tests, plus le code est instable. Plus le code est instable, plus le client signale d'erreurs, et plus on a besoin de temps pour tester!

5.2 JUnit

La réalisation de programmes peut être une tâche complexe, et il n'est pas toujours évident de parvenir rapidement au but que l'on s'est fixé. Il existe heureusement des méthodes et des outils qui permettent de ne pas s'éloigner du but. La conception d'un cahier des charges fonctionnelles, avec la définition de "Besoins fonctionnels" - c'est à dire la définition des fonctionnalités que l'utilisateur attend du programme – est une des premières étapes du processus de développement. Tout au long du processus de développement, il faut veiller à une chose : parvenir à finaliser le programme, en répondant au cahier des charges fixé par le client.

JUnit fait partie de ces outils qui aident le programmeur dans ce sens. Il est tout d'abord bon de savoir que JUnit n'est pas une usine à gaze qui permet d'automatiser les tests de fonctionnalités macroscopiques (celles fixées par le client), il est quelque chose de très simple, destiné aux personnes qui programment.
JUnit est tellement petit et tellement simple que la seule comparaison possible en terme de rapport complexité d'utilisation / utilité est le bon vieux "println" ou les "assertions".
Pour savoir là où le framework JUnit peut nous aider à atteindre nos objectifs on ne pouvait avancer sans avoir répondu à une question assez importante :

Pourquoi JUnit peut-t-il à ce point révolutionner l'efficacité avec laquelle une personne programme ?
Un programmeur qui vient de programmer un petit morceau de programme buggué va s'aider d'un debuggeur ou de la sortie texte sur la console pour savoir ce que fait son petit morceau de code.
Un programmeur qui utilise JUnit va avant d'écrire le petit morceau de code buggué d'abord se demander ce que le petit morceau de programme doit faire, puis il va écrire un petit bout de code capable de vérifier si la tache a bien été réalisée, à l'aide du framework JUnit (le test unitaire).
Enfin il va réaliser le petit morceau de code qu'il devait faire, et il va constater que son code est buggué parce que le test unitaire ne passe pas. Le test unitaire constate en effet que la tache n'est pas complétée. Le programmeur qui utilise JUnit va alors débuguer son code jusqu'à ce que la tache soit correctement réalisée (Rq: Il va éventuellement utiliser un debugger ou des affichages sur la sortie texte pour y parvenir).

5.3 Pourquoi JUnit ?

- Parce que pour une raison ou pour une autre, on doit programmer efficacement. on ne peut pas se permettre de sortir des sentiers battus en programmant "hors sujet" ou en oubliant une fonctionnalité. notre seul objectif en programment est alors de passer les tests unitaires.
- Toujours dans un soucis d'efficacité, on ne peut pas passer non plus notre temps à chercher un bug (ou on n'aime pas faire ça...). Que se serait-il passé si on n'avait pas constaté le bug ? Si on développe un test unitaire, on s’ajoute une marge de sécurité. Donc on laisse moins de bugs derrière nous.
- Parce qu’on veut savoir si on n’est pas en train de faire régresser votre code. JUnit nous dira tout de suite si on a perdu une fonctionnalité en cours de route, par exemple après avoir supprimé 200 lignes de code en pensant bien faire. Ant peut faire appel à JUnit : on saura à chaque compilation s'il y a eu régression. Le refactoring devient moins difficile et aléatoire : on gagnera là encore en efficacité. A noter que JUnit est conçu à la base pour éviter ce genre de problème...
- Parce qu’on travaille à plusieurs sur un projet. On a (ou en tout cas on aura) une plus grande confiance dans un bout de code fourni avec son test unitaire que dans un bout de code fourni sans. Si voir la barre verte (signe d'un test passé avec succès) n'est pas suffisant pour gagner notre confiance, on a alors une seconde possibilité de vérifier le bon fonctionnement, qui est plus rapide que la lecture du code source : la lecture du test. Si le test nous inspire confiance et qu'il passe, alors pourquoi perdre du temps en lisant (pour ne pas dire décrypter !) le code source ? On peut enfin faire confiance au code des autres, et se concentrer sur la résolution de nos bugs à nous ! Là encore, on peut gagner en terme d'efficacité.

5.4 Génération ou création de la classe du test unitaire

La technique pour créer un test consiste à créer une nouvelle classe, portant pour des raisons de commodité le nom de la classe que vous voulez tester, suivi de "Test". Cette nouvelle classe doit hériter de la classe "TestCase".

```
public class ActorTest extends TestCase {}
```

Avec Eclipse, on ne pourrait hériter de TestCase qu'en ayant ajouté le jar de JUnit à notre build path.

❖ **Etape 1 :**

Choix d'un type de test. Dans un premier temps, on ne crée que des TestCase. Les TestSuite permettent seulement de rassembler des TestCase et d'autres TestSuite.

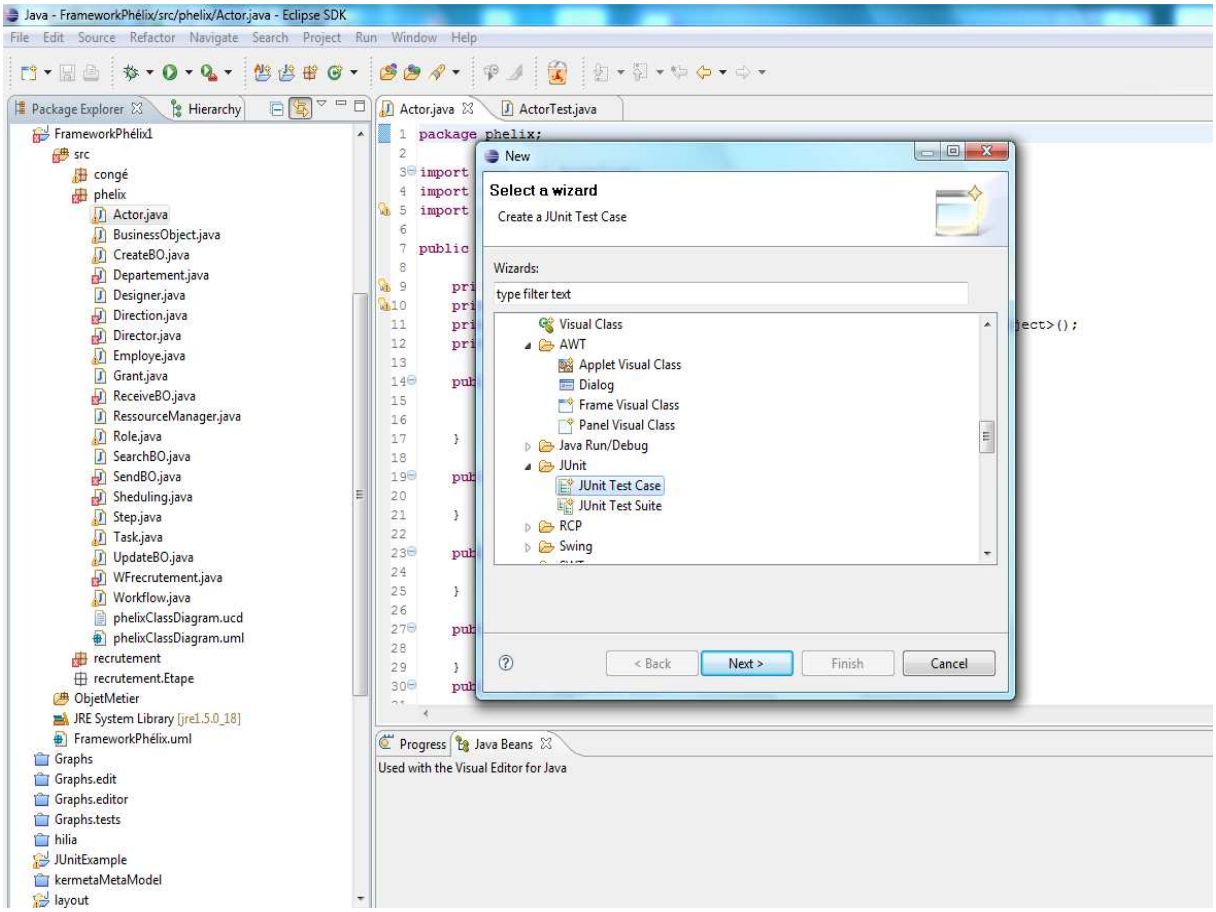

Fig 5.1 création d'un test 1

❖ **Etape 2 :**

On a pensé à séparer les sources des tests des sources normales, afin de faciliter la création d'un .jar pour notre client qui ne contienne pas les tests unitaires

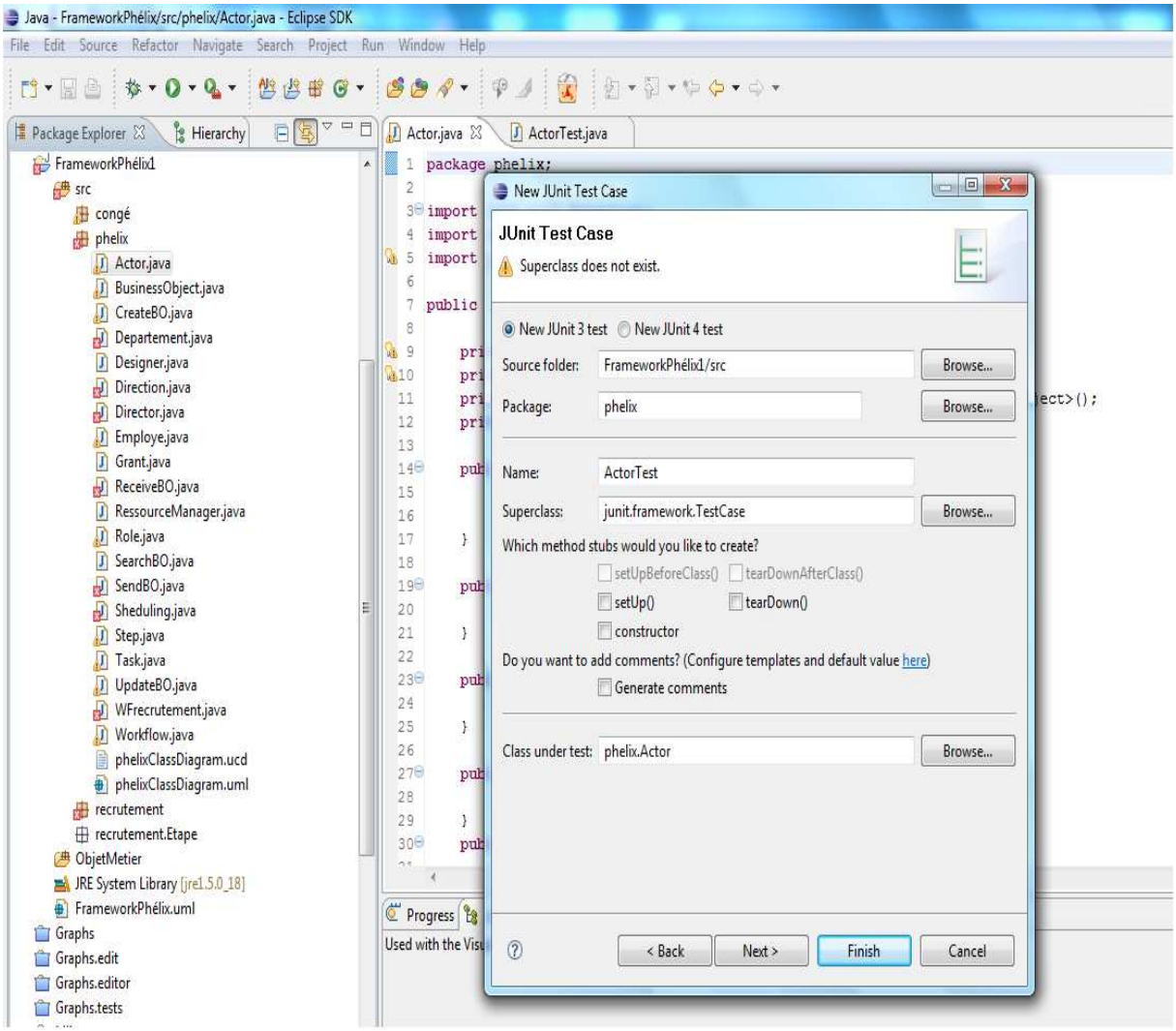

Fig 5.2 création d'un test 2

❖ **Etape 3, facultative :**

Les racines des méthodes à tester peuvent être auto-générées

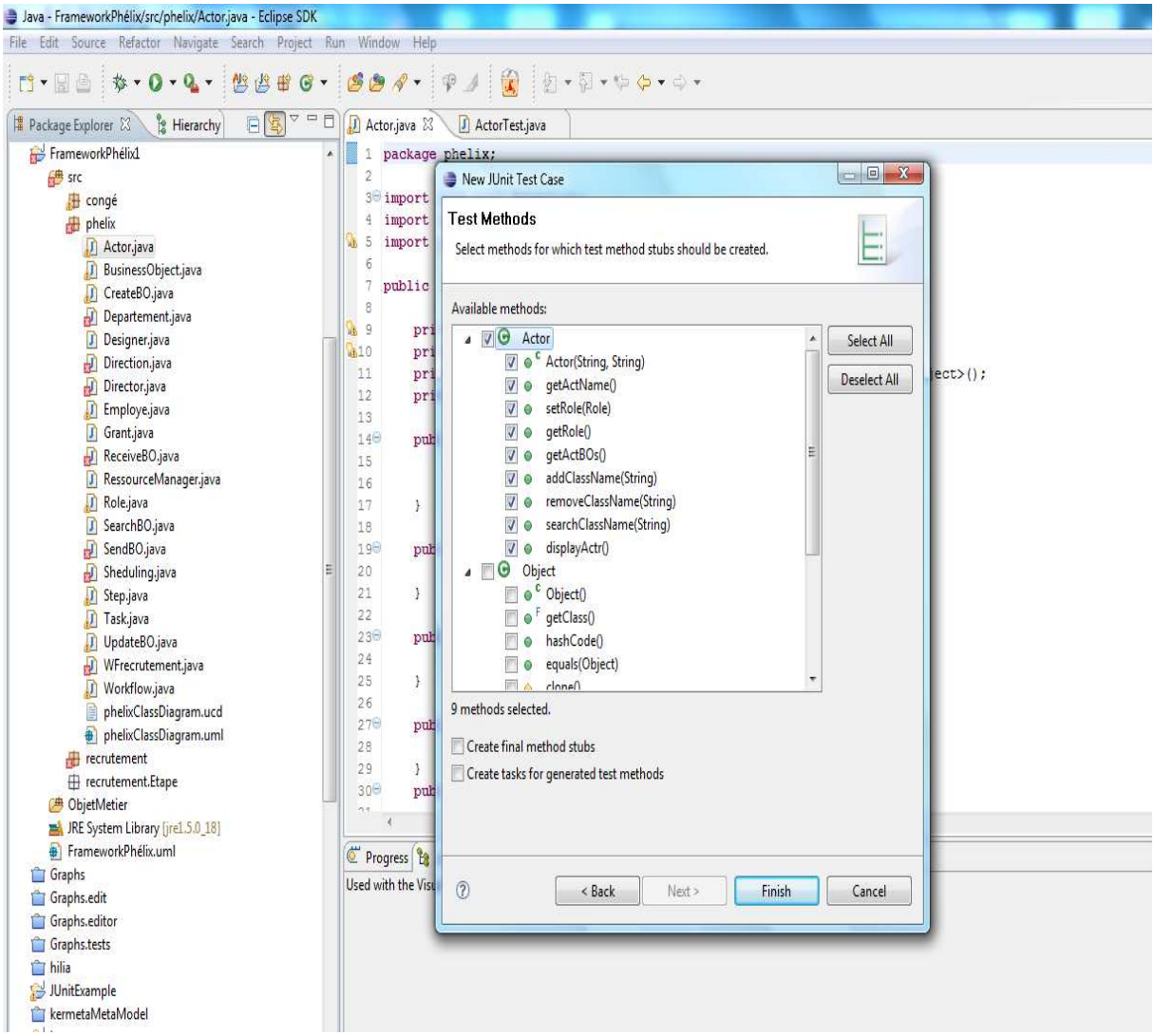

Fig 5.3 création d'un test 3

5.4.1 Ecriture du test unitaire

On réalise maintenant un "test de bon fonctionnement" d'une des méthodes de la classe testée (c'est à dire le test unitaire). Toujours pour des raisons de commodité, il est bon de reprendre le nom de la méthode testée. Cependant il arrive quelques fois (il est recommandé que ça arrive quelques fois) qu'une méthode ait plusieurs tests qui lui soient associés, dans ce cas, **on devrait trouver des noms permettant de comprendre tout de suite d'où vient le problème** (si le test échoue, c'est ce nom qui nous sera présenté, plus éventuellement des informations supplémentaires comme nous allons le voir). Points très importants :

- La méthode doit avoir un nom débutant par "test".
- Elle doit être déclarée public, et ne rien renvoyer (void).

Voici un exemple d'une telle méthode :

```
public void testGetActName() {
            Actor a =  new Actor("Christophe","Dony");
            assertEquals("Christophe",  a.getActName());
      }
```

Ce test "passe" si les deux arguments de assertEquals sont égaux . Généralement on met la valeur de référence attendue par le test en premier argument, et en second argument, la valeur obtenue auprès de la méthode testée.
Notre cas de figure avait favorablement répondu c'est-à-dire en disant qu'il n'y a pas eu d'échec au moment du test et donc la méthode elle retourne bien le prénom de l'acteur alors elle fait exactement ce qu'on lui a demandé de faire.
On aura à la fin une barre verte indiquant la réussite du test (l'image ci-dessous) :

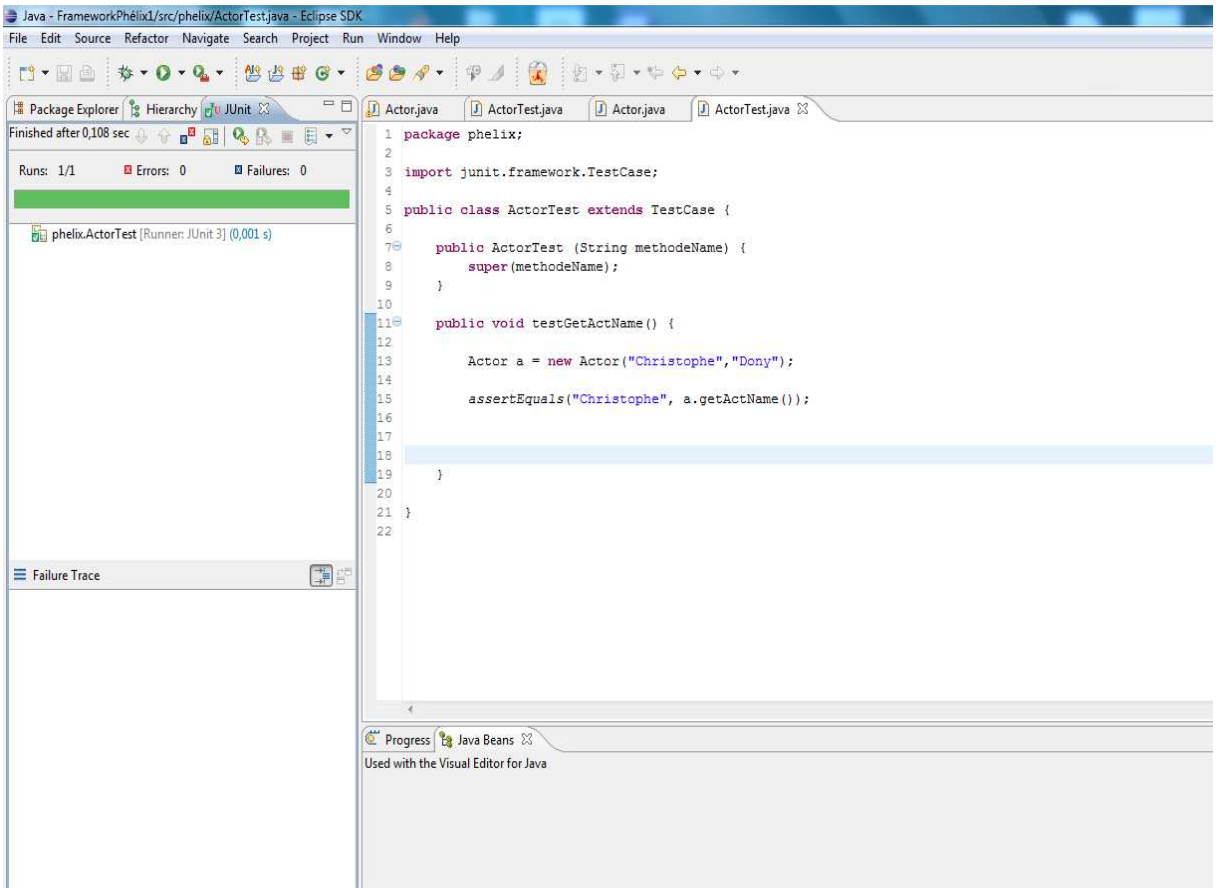

Fig 5.4 Visualisation d'un test réussi

Pour pouvoir visualiser le cas opposé c'est-à-dire un échec, il suffit juste par exemple de mettre dans le champ ASSERT « Christoph » au lieu de « Christophe » en enlevant le « 'e' de la fin et on assistera à l'affichage d'une barre rouge plutôt que verte comme la montre d'ailleurs l'image ci-dessous.

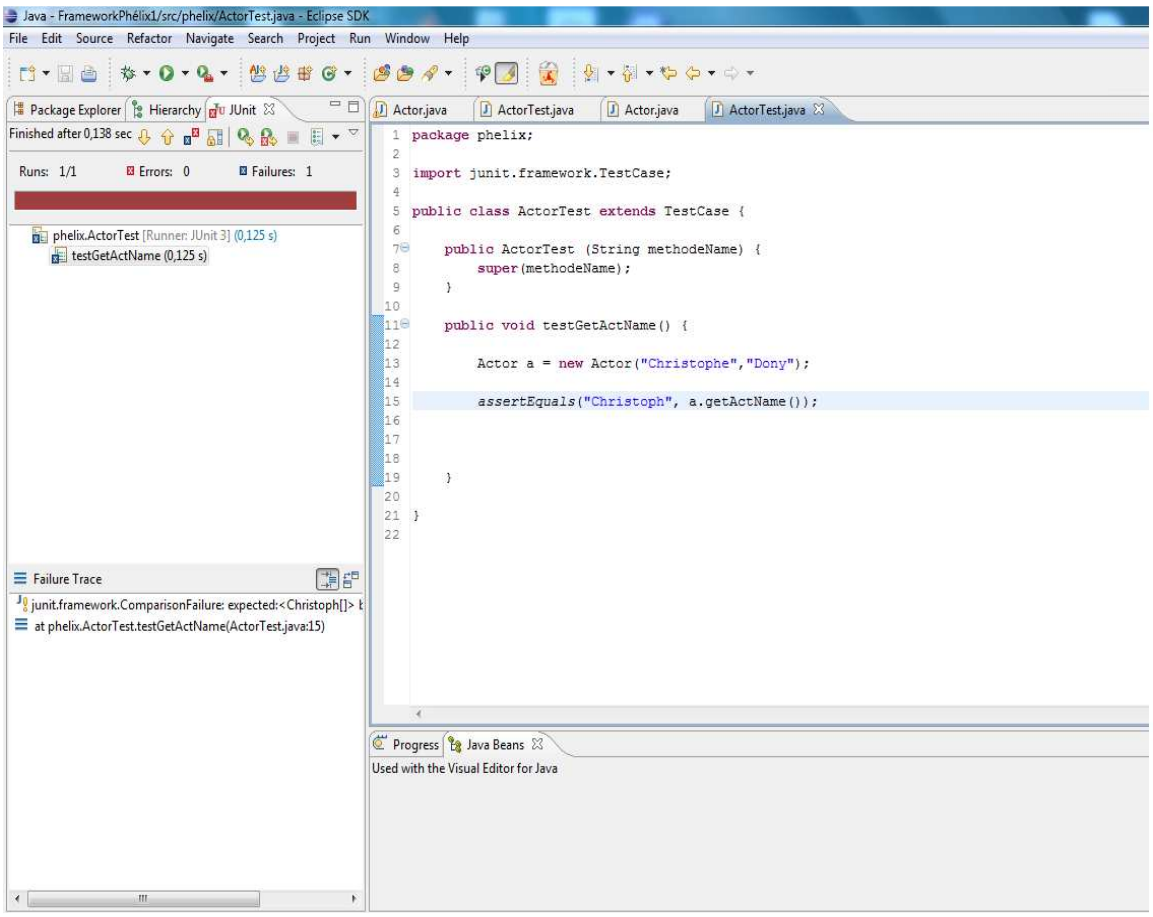

Fig 5.5 Visualisation d'un test échoué

5.4.2La classe Assert

Le propre d'un test unitaire est d'échouer quand le code testé ne fait pas ce qui est prévu. Pour faire échouer un test JUnit (c'est à dire une des méthodes testXXX d'un TestCase), il faut utiliser une des méthodes de la classe junit.framework.Assert, qui sont toutes accessibles au sein d'un TestCase.

- assertEquals

Permet de tester si deux types primitifs sont égaux (boolean, byte, char, double, float, int, long, short). L'égalité de deux objets peut être testée également (attention, ce n'est pas un test sur la référence). Pour les 'double' et les 'float', il est possible de spécifier un delta, pour lequel le test d'égalité passera quand même.

- assertFalse et assertTrue

Teste une condition booléenne.

- assertNotNull et assertNull

Teste si une référence est non nulle.

- assertNotSame et assertSame

Teste si deux Object se réfèrent on non au même objet.

- Fail

Fait échouer le test sans condition. En cas d'utilisation de fail, il est encore plus conseillé que pour les autres méthodes de faire figurer un message expliquant pourquoi le test a échoué

5.5 TestSuite

Réaliser un groupement de tests est simple. La classe dite "TestSuite" n'hérite pas en fait de TestSuit. Elle définit une méthode publique, nommée "suite", renvoyant un objet de type Test qui peut contenir un TestCase ou un TestSuite. Pour réaliser nos "TestSuite", on devrait reprendre le même prototype de fonction. A noter que là encore, Eclipse propose un Wizard automatisant la création du TestSuite.

La méthode addTest(..) permettra donc d'ajouter des tests dans une suite. Chaque cas de test est généré par le biais du constructeur de la classe cas de test, lequel réclame comme paramètre le nom de la méthode de test.

Voici un exemple de Testsuite qu'on lui a donné deux tests de cas ActorTest et Demandecongé :

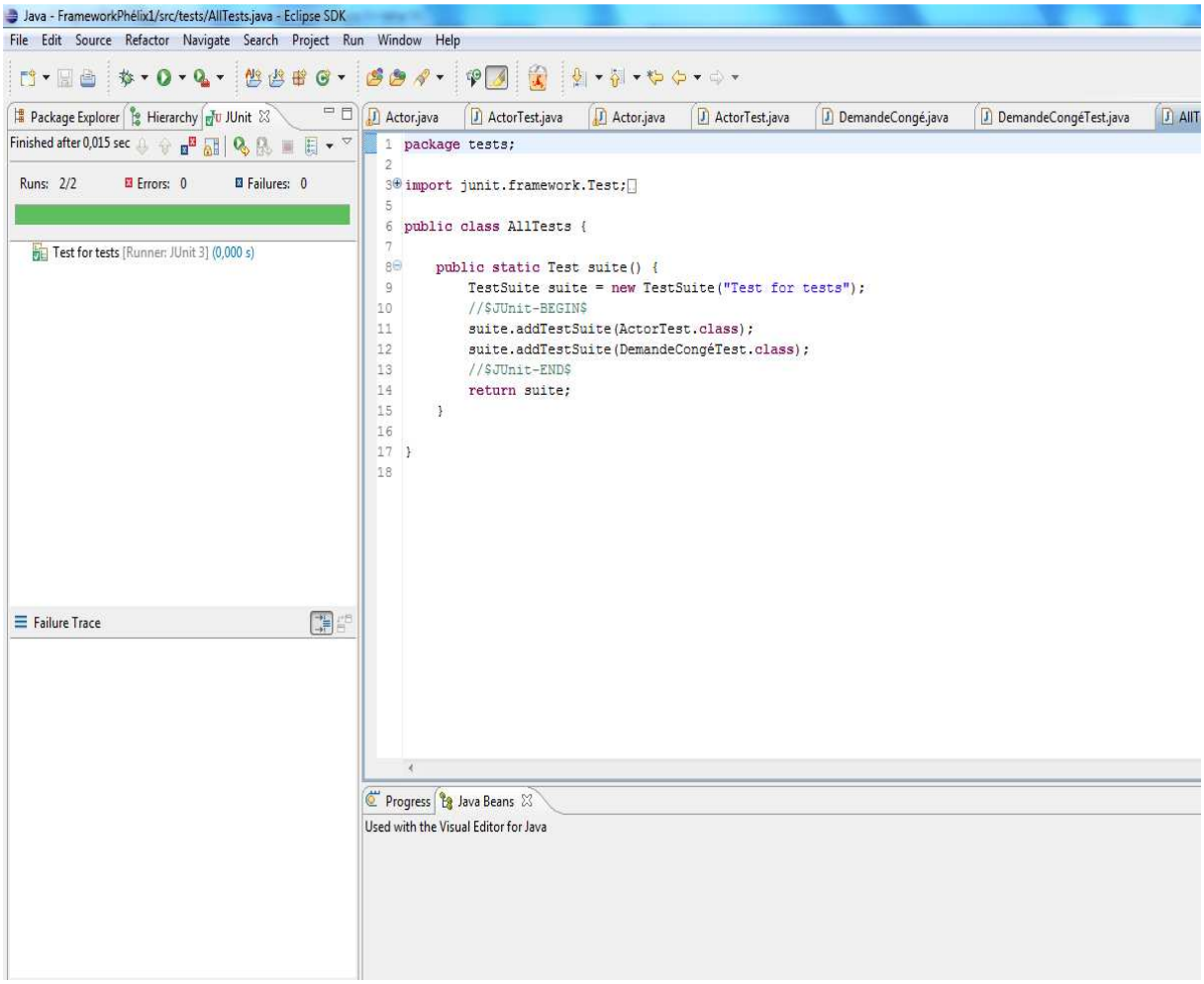

Fig 5.4 Création visualisation d'un TestSuite

Le résultat indique qu'il n'y avait pas eu de d'échec de la suite de tests.

Dans le cas ou on aurait obtenu un échec, l'interface qui se trouve à gauche nous le signalera (Barre rouge) en précisant exactement le TestCase ou on a le problème à régler.

5.6 Conclusion :

Avec le JUnit on avait assuré le bon fonctionnement des méthodes des classes du Framework et on a même gagné du temps pour connaitre la source des bugs rencontrés dans la phase programmation en testant à chaque fois tous les bouts de codes récemment ajoutés.

Conclusion

1 La solution à la problématique

Dans ce travail de TER, nous avons réalisé un framework dédié au domaine de workflow nous avons donc :

- Etudié les frameworks orientées objets ;
- Trouvé un domaine d'application original pour lequel cette technologie est applicable : le développement de workflow d'entreprises ;
- Imaginé deux applications typiques de ce domaine : recrutement et gestion des demandes de congés ;
- Réalisé successivement :Une implantation de l'application recrutement ,ensuite une implantation de l'application gestion des demandes de congés, puis à partir de ces deux solutions nous avons découvert des abstractions qui nous ont permis de modéliser le diagramme de classes du framework en utilisant la notation UML et le framework J2SE pour implémenter un workflow générique qui a été paramétré selon les besoins des deux workflows concrets ''recrutement'' et ''demande de congé''. Enfin nous avons choisi le nom «Phélix» car il est proche du nom du framework Éclipse.

2 Difficultés rencontrées

Au niveau de la conception du workflow générique nous avons modélisé une première version du diagramme de classe qui a été mis à jour à cause de quelques associations incompatibles qui devaient changer au niveau de la direction et la multiplicité. Aussi, le choix des classes abstraites et des points de paramétrage n'était pas évident à la phase de la modélisation il a fallu entrer dans la phase de l'implémentation pour décider du sort des classes abstraites et des points d'extension. Enfin pour implémenter les classes CreateBO et UpdateBO héritières de la classe Task qui possède la méthode abstraite tskPlugin(...), nous nous sommes familiarisés avec le package java.reflect pour réussir à appeler le constructeur d'une classe héritière de la classe BusinessObject ou de mettre à jour l'état d'un objet héritier de la classe BusinessObject.

3 Tâches réalisées et moyens de communication

Pour faciliter la communication entre les membres du groupe nous avons utiliser les mails et la messagerie instantanée, nous avons aussi mis en place une page web a l'adresse http://sites.google.com/site/sam6pro/ pour partager le code des deux applications et celui du framework. Enfin, en fonction de nos emplois du temps, nous faisions des réunions une fois par semaine soit au Lirmm si nous avons rendez-vous avec Monsieur Dony, à la BU Montpellier 2 ou encore dans les salles de travail de la cité u la Colombière. Voici la listes des tâches que nous avons réalisé dans le cadre du TER :

- Analyse de l'état de l'art des deux workflows et des framework OO.
- Réalisation du workflow recrutement en parallèle avec le workflow demande de congé
- Analyse et conception de «Phélix» (découverte d'abstractions)
- Implémentation de «Phélix» en Java
- Paramétrage de «Phélix» pour la réalisation des deux workflows
- Test, itération et étude qualitative

4 Perspective

Dans la perspective d'améliorer la productivité de «Phélix» et de lui donner un comportement flexible nous envisageons d'atteindre les avantages d'une utilisation conjointe entre framework et patterns. Les schémas de conception diminuent la complexité d'un logiciel en encapsulant le niveau d'abstraction.
L'intelligibilité d'une application semi-finie est améliorée puisque le framework n'est plus composé d'une multitude de classes interconnectées mais de quelques patterns. Ainsi nous utiliserons le pattern **State** pour la classe BusinessObject qui a un attribut boState. Les programmeurs pourront implémenter la machine à état associée à une classe héritière de BusinessObject, par exemple dans le processus ‘’recrutement’’ le candidat est un objet métier qui a un état initial en attente puis passe à l'état sélectionné sur ‘’CV’’ et sélectionné sur ‘’entretien’’ et enfin à l'état recruté ou refusé, durant chaque état le comportement du candidat devra changer.
Rappelons que le pattern State permet, lorsqu’un objet est altéré de changer son comportement.
Le deuxième schéma que nous utiliserons est le pattern **Singleton** dont l'objet est de restreindre l'instanciation des classes Role, CreateBO, UpdateBO, SearchBO et RessourceManager à un seul objet.

Enfin nous souhaiterons utiliser le pattern **MVC** pour organiser l'interface homme-machine du framework ce qui permettra aux acteurs d'un workflow de se connecter avec un identifiant et un mot de passe pour effectuer les tâches requises sur différentes interfaces graphiques ergonomiques et conviviales.

Bibliographie

[1] C. Gane & T. Sarson. Structured Systems Analisys: Tools and Techniques. Prentice Hall.
Englewood Cliffs, NJ. 1979.
[2] N. Wirth, Systematic progrmming : an introduction, Prentice Hall, 1973.
[3] O.J. Dahl, E.W. Dijkstra, C.A.R. Hoare, Structured programming, APIC studie on data processing n° 8, London Academic Press, 1972.
[4] Cox B.J., Novobilski A.J., Object Oriented Programming: an Evolutionary Approach, Addison-Wesley
Publishing Company, 1991.
[5] G. Masinin, A. Napoli, D. Colnet, D. Leonard, K. Tombre. Les langages à objets. InterÉditions,
1989.
[6] Wirfs-Brock R. J. & Johnson R. E., Surveying current research in Object-Oriented design, CACM Vol. 33, No. 9, pp. 105-124, Sept. 1990.
[7] Goldberg A. & Robson D., Smalltalk 80: the language and its implementation. Addison-Wesley, 1983.
[8] Krasner G.E. & Pope S.T. A cookbook for using the Model-View-Controller User Interface
Paradigm in Smalltalk 80. J. of Object-Oriented Program. Vol. 1, No. 3, pp. 26-49, Aug.-Sept. 1988.
[9] ACKIA, Ackia Tool Box, Rapports internes ACKIA, 1991 à 1993 (cf. note 2, p. 6).
[10] R. Voyer, Editalk - Documentation technique, Rapport interne Ackia, 1992.
[11] J. Leroudier, La simulation à évènements discrets, Monographies d'informatique de l'AFCET,
publiées sous la direction de E. Gelenbe, Éditions hommes et techniques, 1980.
[12] Pr. Christophe Dony Notes de cours Schémas de Réutilisation Objet Frameworks Patterns Notes de cours - 2003-2008 Université Montpellier-II

Bibliographie Web

[13] http://guidecms.com/dossiers-cms/supports-de-formation/presentation-copix

[14] http://fr.wikipedia.org/wiki/Workflow

[15] http://www.commentcamarche.net/contents/entreprise/workflow.php3