Projet Multi-modules

Getting Things Done

Jérémy Braud Nicolas Bremard Cédric Krommenhoek Boris Lucas

Résumé

La méthode GTD est une démarche d'organisation personnelle applicable par chacun à l'ensemble de ses activités, tant professionnelles que privées. Décrite par David Allen dans son livre, la méthode GTD a pour objectif d'identifier avec sûreté ses priorité à tout moment, et d'agir immédiatement sur la priorité choisie. En effet, pour bien choisir sa priorité et s'y consacrer pleinement, il faut être certain de s'appuyer sur un système que l'on juge fiable.

Table des matières

Table des figures

Chapitre 1

Analyse du problème

Sommaire

1.1 Introduction

L'objectif du premier livrable est d'analyser la méthode GTD afin de pouvoir par la suite en développer un outil. Dans notre cas, il s'agira d'un client permettant l'exploitation de cette méthode. Nous devons pour cela identifier les différents cas d'utilisation, et les décrire à l'aide du canevas de Cockburn, ainsi qu'à l'aide des différents outils UML tels les diagrammes de séquences, de classes, etc.

1.2 Dictionnaire de données

Afin que les termes employés soient correctement interprétés par tous, nous les avons définis dans un dictionnaire de données.

Notion	**Définition**	**Traduit en...**	**Nom informatique**
Chose à faire	Toute idée, pensée, activité pouvant nécessiter une action de la part de l'**utilisateur** et devant être renseignée par la suite		
Contact	Ensemble d'attributs (nom, numéro de téléphone, email, etc.) définissant un contact	Classe	Contact
Contexte	L'endroit où l'on se trouve, les outils disponibles, les limites et possibilités de l'environnement	Classe	Contexte
Echéance	Date butoir jusqu'à laquelle la **tâche** peut être accomplie		
Fréquence de répétition	La fréquence à laquelle se répète la **tâche** (journalière, mensuelle, etc.)		
Priorité	Valeur comprise entre 1 et 5 traduisant l'urgence de la **tâche**		
Projet	Ensemble de **tâches** et de sous-projets, ayant une finalité propre et définissant un **contexte** par défaut	Classe	Projet
Tâche	Action atomique, pouvant être contenue dans un **projet**, et pouvant contenir un nom, une date de début, une **échéance**, des notes, une **priorité**, un **taux d'effort demandé**, une liste de **contacts**, une **fréquence de répétition**, une date d'arrêt de la répétition, des liens, des **tags**	Classe	Tache
Tags	Mots-clés liés à une **tâche**		
Taux d'effort demandé	Valeur comprise entre 0 et 99 définissant la complexité supposé de la **tâche**		
Utilisateur	Toute personne ayant recours à la méthode GTD	Acteur	User

Action	**Définition**	**Traduit en...**	**Nom informatique**
Collecter	Action qui permet de recenser toutes les **choses à faire** à traiter	Opération	Collect
Examiner	Action de mise à jour du système GTD	Opération	Review
Faire	Choix de la **tâche** à effectuer immédiatement	Opération	Do
Organiser	Action de classement des **tâches** par rapport à leurs attributs (**priorité**, **échéance**, etc.)	Opération	Organize
Spécifier une chose à faire	Spécifier la tâche correspondant à la chose à faire ou en faire un projet (ou sous-projet)	Opération	Specify
Traiter	Action de renseignement des tâches collectées	Opération	Process

1.3 Cas d'utilisation

Après une recherche approfondie sur la méthode GTD, nous avons retenu cinq cas majeurs d'utilisation :

1. Collecter des choses à faire.
2. Traiter ces choses à faire.
3. Organiser les tâches issues du traitement.
4. Examiner régulièrement ces tâches et ajouter celles en attente.
5. Sélectionner une tâche afin de la réaliser.

Nous obtenons alors le diagramme de cas d'utilisation de la figure 1.1.

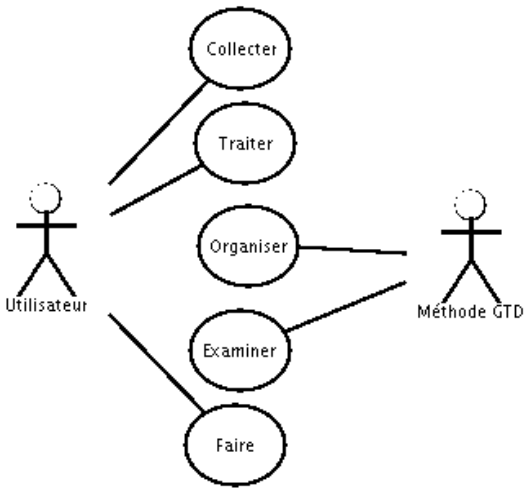


FIGURE 1.1 – Diagramme de cas d'utilisation

1.3.1 Collecter les informations

L'objectif de ce cas d'utilisation est de recenser tout ce à quoi on pense et qu'on peut être amené à réaliser par la suite. Ces idées doivent pouvoir être notées rapidement, car elles peuvent apparaître à un moment où l'on est occupé (au téléphone, en réunion, etc.).

Canevas de Cockburn

Use Case: 1 – Collecter des choses à faire

CHARACTERISTIC INFORMATION

Goal in context : Récupérer des choses à faire et les ajouter à la liste de choses à faire
Scope : La méthode GTD
Level : Summary
Pre-conditions : Avoir des choses à faire non classées dans GTD
Post-conditions : Aucune chose à faire perdue
Success End Condition : Toutes les choses à faire voulues ont été rentrées
Failed End Condition : Les choses à faire données par l'utilisateur ont été perdues
Primary actor : L'utilisateur, toute personne voulant être mieux organisée
Trigger : L'utilisateur a demandé à rentrer des choses à faire

MAIN SUCCESS SCENARIO

1. L'utilisateur demande à ajouter des choses à faire
2. La méthode GTD enregistre les choses à faire
3. L'utilisateur arrête la saisie

RELATED INFORMATION

Priority : Faible
Performance target : Quelques secondes par chose à faire
Frequency : Régulièrement (potentiellement tous les jours)
Channel to primary actor : Action directe de l'utilisateur

OPEN ISSUES

– Que se passe-t-il si le système perd une ou plusieurs choses à faire ?

Instantané

Avant application de ce cas d'utilisation (figure 1.2), on obtient une liste des choses à faire auquelles on a ajouté des éléments (figure 1.3).

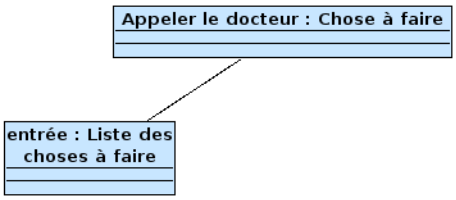


FIGURE 1.2 – Etat du système *avant* la collecte

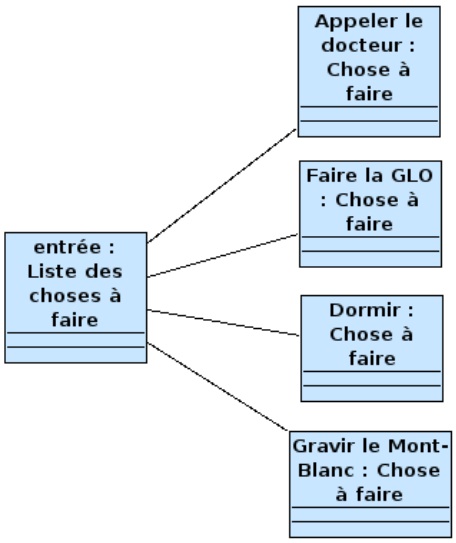


FIGURE 1.3 – Etat du système *après* la collecte

Contraintes OCL

Nous n'avons pas défini de contraintes OCL pour ce cas d'utilisation.

Diagramme de séquence

figure 1.4.

1.3.2 Traiter ces informations

Après la collecte des informations, il faut leur appliquer un traitement afin connaître certaines propriétés essentielles pour le système, telles le contexte de la tâche, la périodicité, l'échéance, etc.

Canevas de Cockburn

Use Case: 2 – Collecter des choses à faire

CHARACTERISTIC INFORMATION

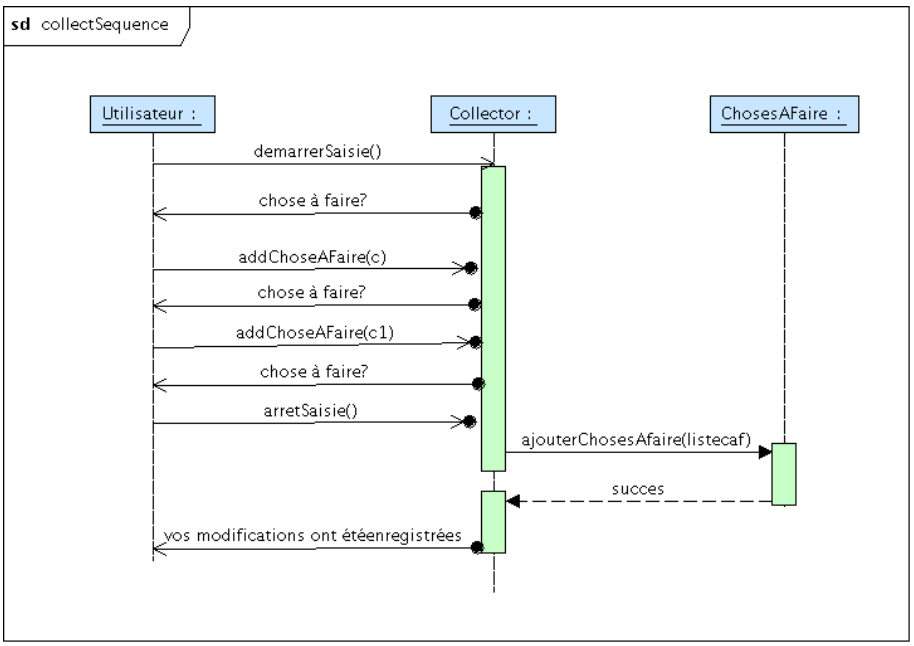


FIGURE 1.4 – Scénario de collecte des choses à faire

Goal in context : On effectue le traitement (création d'un projet contenant une décomposition de la tâche courante, ordonner à l'utilisateur de faire la tâche, la déléguer, la reporter, l'archiver) d'une ou plusieurs choses à faire de la liste

Scope : La gestion des priorités après traitement, la création de la liste de choses à faire

Level : Summary

Pre-conditions : Liste de choses à faire non-vide

Post-conditions : Les tâches traitées apparaissent dans les donnée stockée

Success End Condition : Le chose à faire supprimée de la liste a été traitée

Failed End Condition : Le traitement a échoué

Primary actor : L'utilisateur

Trigger : L'utilisateur

MAIN SUCCESS SCENARIO

1. L'utilisateur demande à lancer le traitement des éléments de la liste des choses à faire
2. La méthode GTD émet une demande de renseignements des attributs de la chose à faire, en vue d'en faire une tâche
3. L'utilisateur rentre ou non des informations
4. La méthode GTD applique le traitement à la tâche ou le projet
5. La méthode GTD supprime la chose à faire qui vient d'être traitée
6. Arrêt de la procédure de traitement

SUB-VARIATIONS

3a L'utilisateur rentre ou non des informations
3a1 L'utilisateur effectue la tâche immédiatement et retour à l'étape 2.
3a2 Il n'y a pas encore de contexte adapté à cette tâche, création d'un nouveau contexte.
6a L'utilisateur rentre ou non des informations et demande la chose à faire suivante
6a1 On remplace l'arrêt du traitement par le retour à l'étape 2.

RELATED INFORMATION

Priority : Moyenne
Performance target : Moins de deux minutes
Frequency : Régulièrement (une fois par semaine)
Channel to primary actor : Action directement effectuée par l'utilisateur

OPEN ISSUES

- Que se passe-t-il si la chose à faire n'est pas sauvegardée ?
- Que se passe-t-il si la chose à faire n'est pas supprimée ?

Instantané

Après traitement de données du système (figure 1.5), des tâches sont créés et associées à des contextes (figure 1.6).

Contraintes OCL

```
-- La taille de liste de choses a faire doit etre decrementee,
-- et au moins un conteneur de taches doit posseder des
-- elements supplementaires
context Traitement::traiterChoseAFaire()
pre Collector.getListeChoseAFaire().size() > 0
post Collector.getListeChoseAFaire().size()@pre >
                Collector.getListeChoseAFaire().size()
        and (NextAction.size() > NextAction.size()@pre
        or WaitFor.size() > WaitFor.size()@pre
        or Someday.size() > Someday.size()@pre
        or Archive.size() > Archive.size()@pre)
```

Diagramme de séquence

Dans le cas où la chose à faire correspond à une simple tâche, nous obtenons le diagramme de séquence de la figure 1.7.

1.3.3 Organiser les tâches issues du traitement

Canevas de Cockburn

Use Case: 3 – Organiser les éléments stockés dans les dossiers

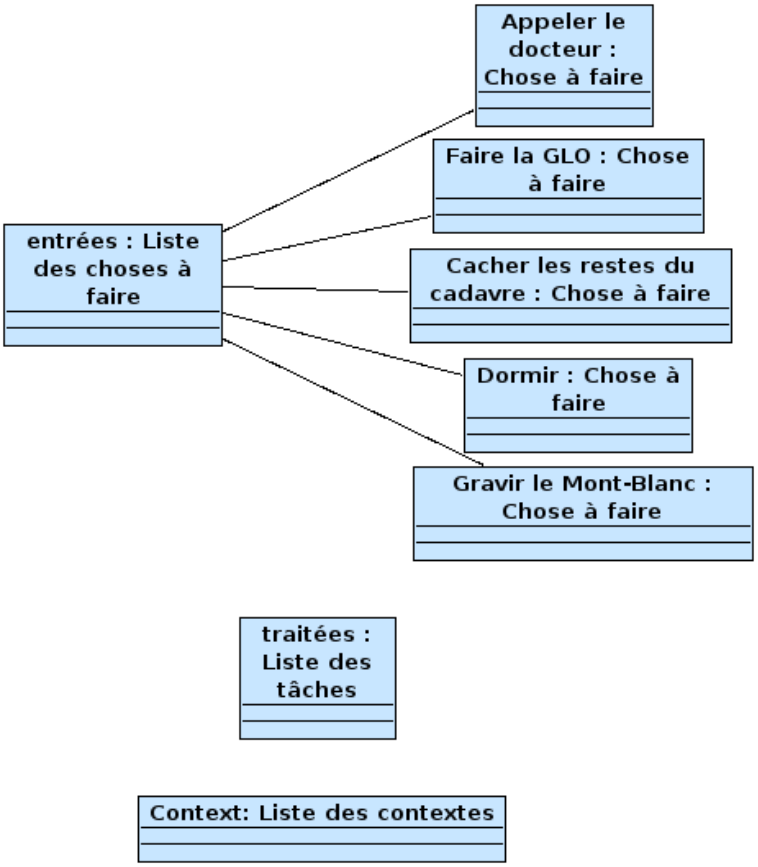


FIGURE 1.5 – Etat du système *avant* le traitement

CHARACTERISTIC INFORMATION

Goal in context : Gérer les priorités, les contextes et tout autres attributs pour définir l'ensemble des prochaines actions à partir des tâches dans les dossier.

Scope : La création des tâches, leurs actions et leurs attributs.

Level : Primary Task

Pre-conditions : /

Post-conditions : /

Success End Condition : Toutes les tâches ont été triées.

Failed End Condition : Tâches inorganisable, erreur système.

Primary actor : Le système

Trigger : Après le traitement/Après chaque modification

MAIN SUCCESS SCENARIO

1. Le système requiert de l'ordre

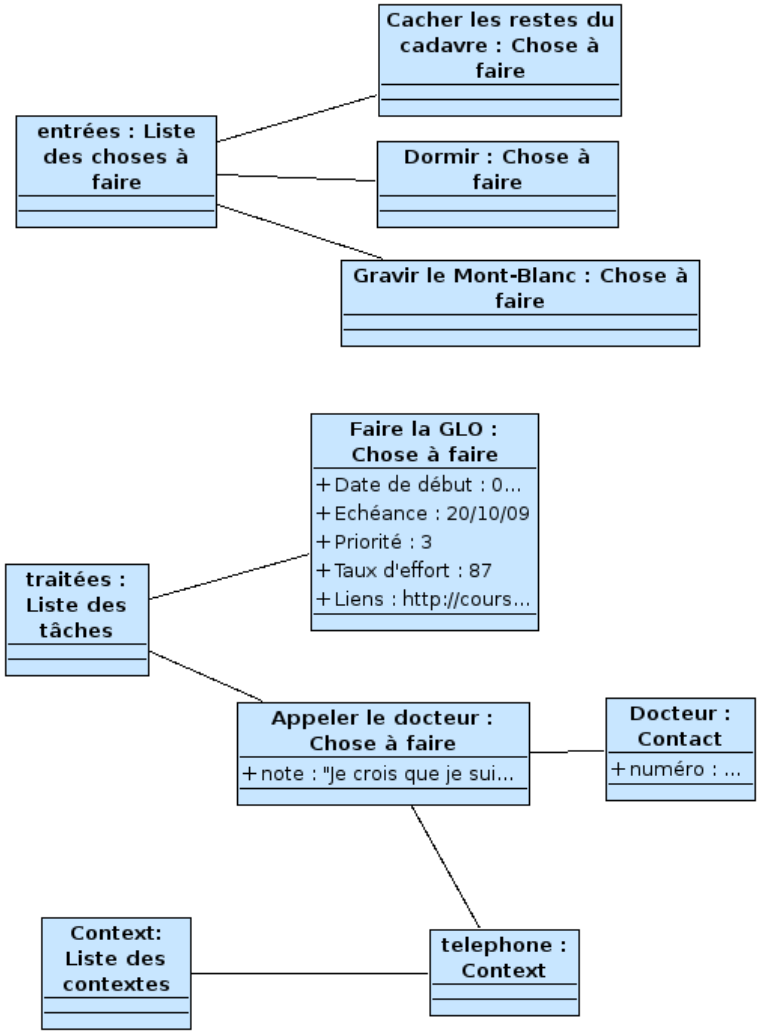


FIGURE 1.6 – Etat du système *après* le traitement

2. Les éléments mal-placés ou entrants sont déplacés au bon endroit

RELATED INFORMATION

Priority : Haute

Performance target : Invisible à l'utilisateur

Frequency : Après le traitement/Après chaque modification

OPEN ISSUES

– Que se passe-t-il dans le cas de conflit priorité/échéance ?

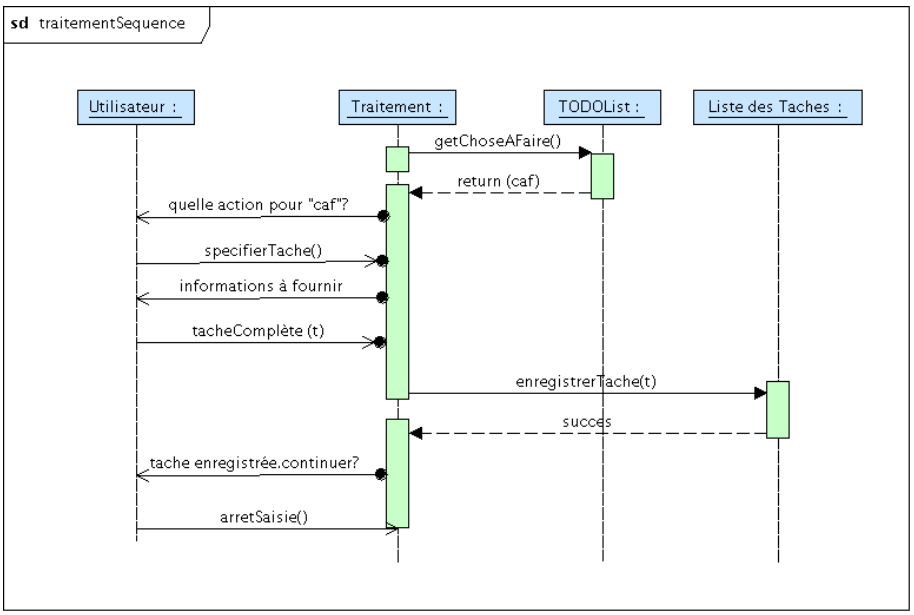


FIGURE 1.7 – Scénario de traitement d'une chose à faire

Instantané

Avant organisation (figure 1.8), nous avons une liste de tâches dont les éléments se font spécifier (figure 1.9).

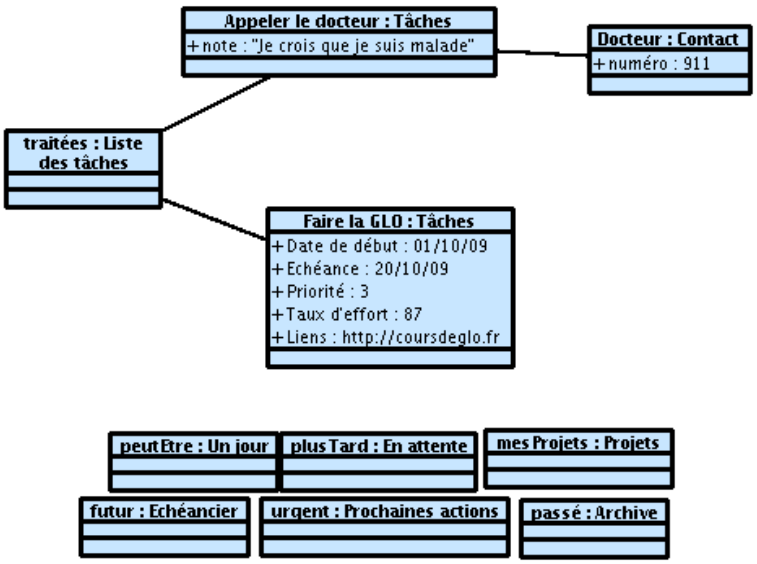


FIGURE 1.8 – Etat du système *avant* l'organisation

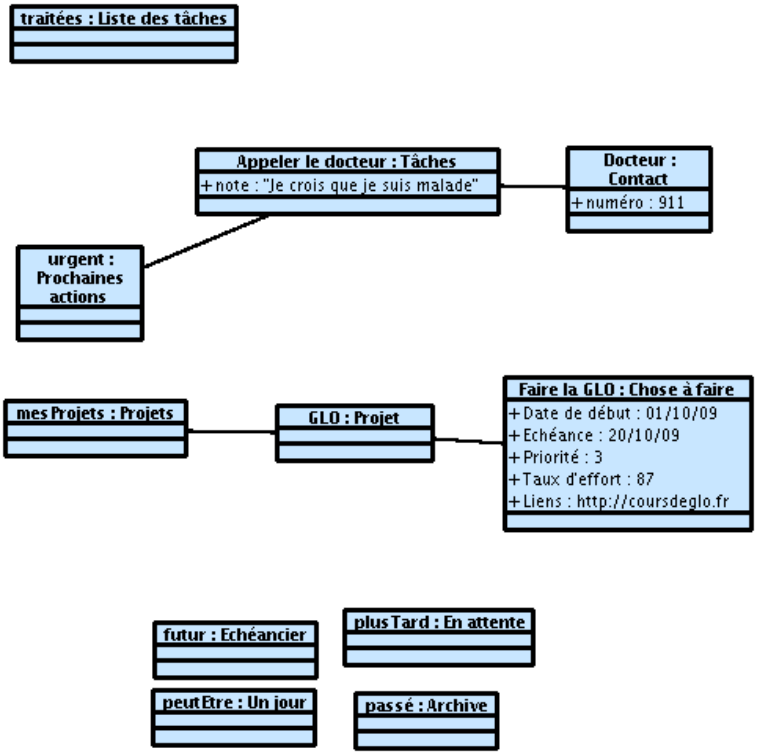


FIGURE 1.9 – Etat du système *après* l'organisation

Contraintes OCL

Nous n'avons pas défini de contraintes OCL pour ce cas d'utilisation.

Diagramme de séquence

Le scénario de l'organisation des tâches est décris par la figure 4.4.

1.3.4 Examiner régulièrement ces tâches et ajouter celles en attente

Canevas de Cockburn

Use Case: 4 – Vérification des données stockées

CHARACTERISTIC INFORMATION

Goal in context : Vérifier si aucune tâche dans En Attente ou dans Un Jour n'est passée à l'état actif et met l'échéancier à jour..

Scope : Les tâches

Level : Primary Task

Pre-conditions : Il y a des tâches à examiner.

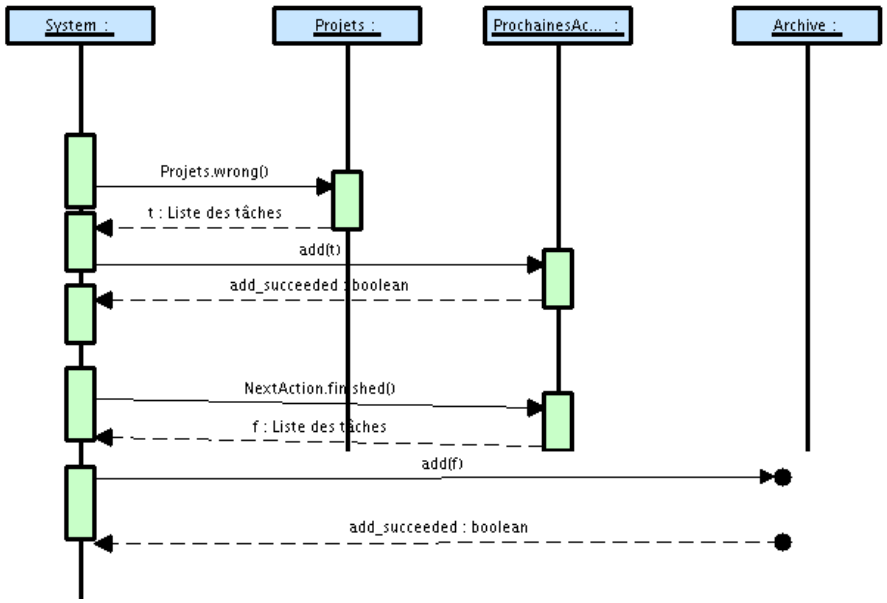


FIGURE 1.10 – Scénario d'organisation des tâches

Post-conditions : L'échéancier a été mis à jour.
Success End Condition : Toutes les tâches examinées ont été activées
Failed End Condition : Erreur système.
Primary actor : Le système
Trigger : Automatique

MAIN SUCCESS SCENARIO

1. Vérifier dans En Attente si des tâches sont devenues actives
2. Vérifier dans Un Jour si il est possible de réaliser une tâche
3. Vérification si il y a un jour1 dans l'échéancier, et, si c'est le cas, ajoute ses projets dans le dossier Projets (en ajoutant sa première tâche dans Prochaines Actions) .
4. Mettre à jour l'échéancier
5. Transférer les tâches dans Prochaines Action

RELATED INFORMATION

Priority : Haute
Frequency : Une fois par jour

Instantané

Avant l'examen des tâches (figure 1.11), nous avons une organisation qui va être remaniée par cette opération (figure 1.12).

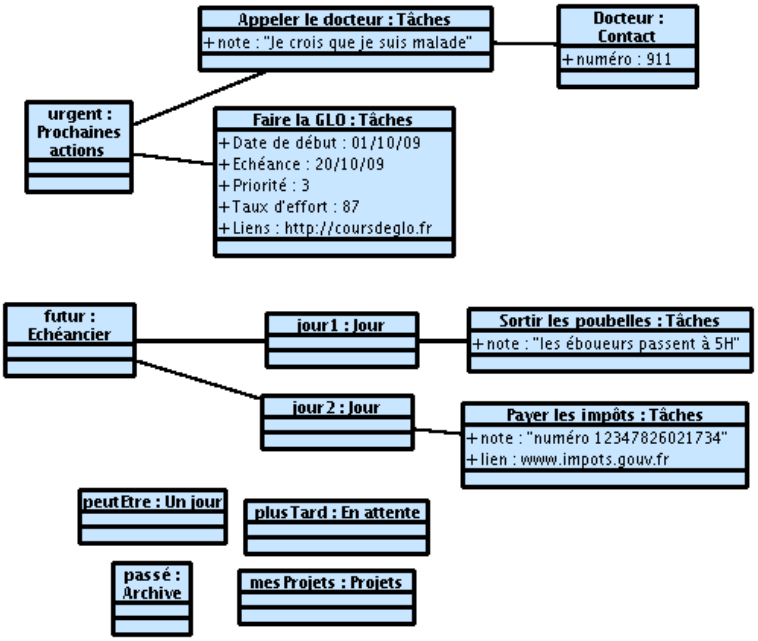


FIGURE 1.11 – Etat du système *avant* la mise à jour des tâches de l'échéancier

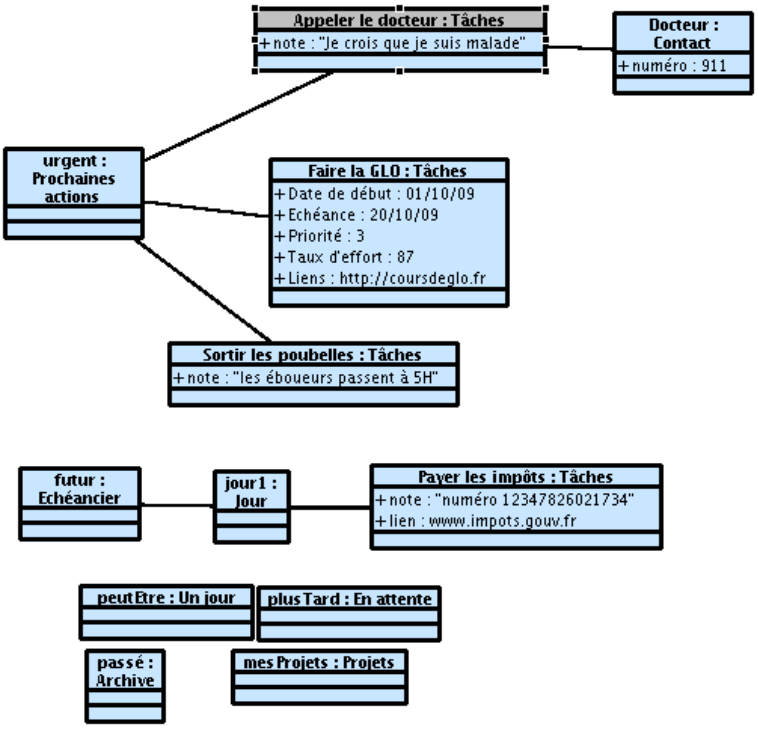


FIGURE 1.12 – Etat du système *après* la mise à jour des tâches de l'échéancier

Contraintes OCL

```
-- Il doit au moins y avoir une tache a examiner
context Examen::review())
pre WaitFor.size() > 0
        or Someday.size() > 0
        or Echeancier.size() > 0
post Echeancier.jour1.getEcheance() = Date.today()
```

Diagramme de séquence

Le scénario type de ce cas d'utilisation est décris par la figure 1.13.

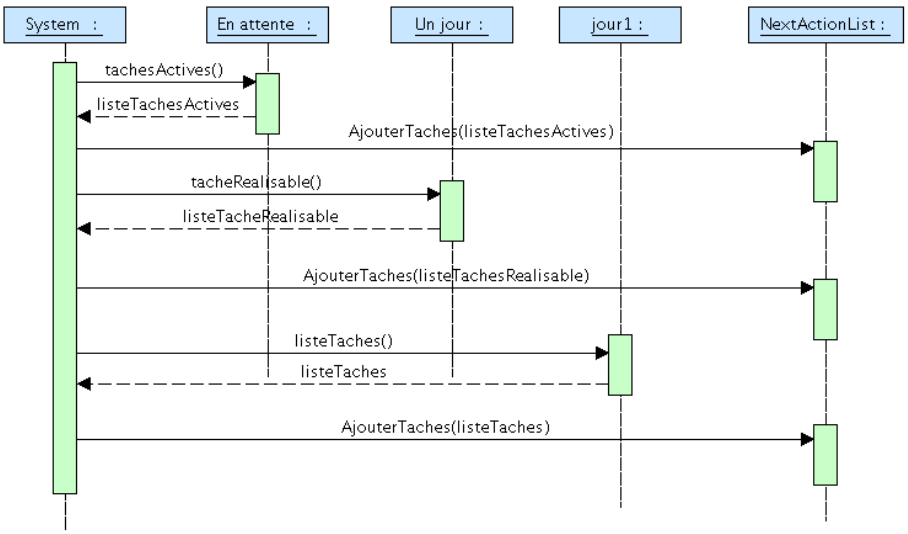


FIGURE 1.13 – Scénario du cas d'utilisation examen

1.3.5 Sélectionner une tâche afin de la réaliser

Canevas de Cockburn

Use Case: 5 – Réaliser une tâche

CHARACTERISTIC INFORMATION

Goal in context : Obliger l'utilisateur à effectuer la tâche la plus prioritaire dans le dossier Prochaines Actions dans son contexte.

Scope : Le contenu des Prochaines Actions

Level : PrimaryTask

Pre-conditions : L'utilisateur a du temps pour réaliser une tâche et il y a des tâches à effectuer.

Post-conditions : L'utilisateur à réalisée la tâche prédéfini.

Success End Condition : Ancienne tâche archivée et la seconde Prochaine Action devient la première.

Failed End Condition : La tâche n'a pas été supprimée ou l'utilisateur n'a pas réalisé son travail.

Primary actor : L'utilisateur

Trigger : L'utilisateur

MAIN SUCCESS SCENARIO

1. L'utilisateur demande à faire une tâche
2. L'utilisateur donne le contexte dans lequel il se trouve.
3. La méthode GTD choisit la tâche.
4. L'utilisateur indique qu'il a fini la tâche
5. La tâche est supprimée
6. Mise à jour des Prochaines Actions
7. L'utilisateur ne fait plus d'actions

EXTENSIONS

4a L'utilisateur n'a pas fini la tâche.

4a1 La méthode GTD ne modifie rien.

7a L'utilisateur demande une nouvelle tâche à faire

7a1 Retour à l'étape 3.

SUB-VARIATIONS

6.1 Si une tâche est En Attente de la tâche réalisée, le système doit en informer l'utilisateur comme Prochaine Action à réaliser.

RELATED INFORMATION

Priority : Haute

Frequency : Le plus souvent possible

OPEN ISSUES

– Que se passe-t-il si l'utilisateur ne finit jamais de tâches ?

Instantané

Avant la réalisation de la tâche (figure 1.14), la prochaine tâche devant être exécuté est (dans notre exemple) d'appeler un docteur. Cette tâche est alors désignée pour être terminée (figure 1.15).

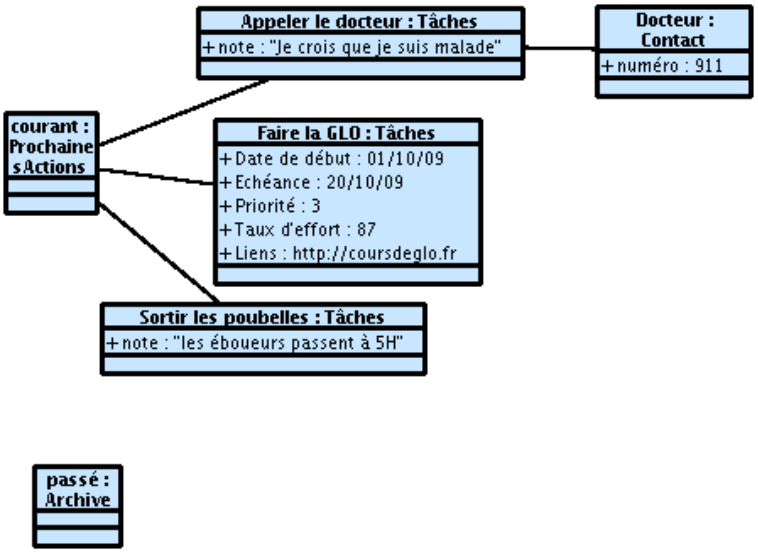


FIGURE 1.14 – Etat du système *avant* la réalisation d'une tâche

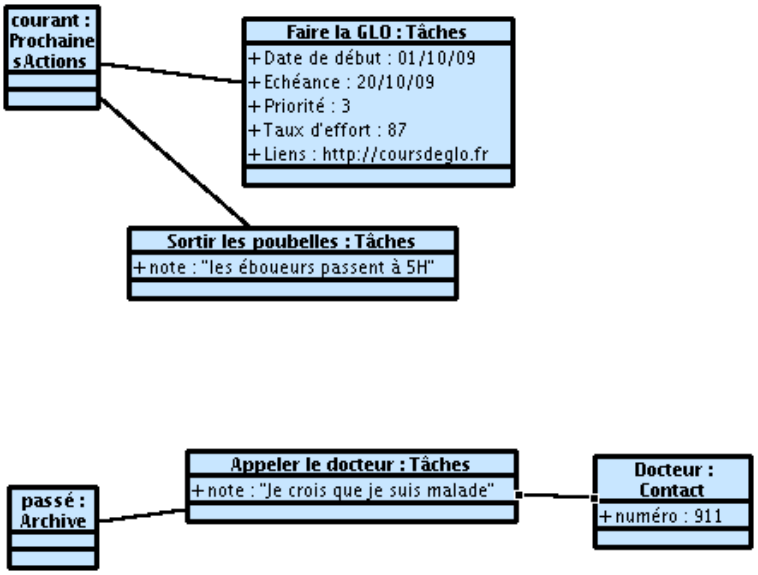


FIGURE 1.15 – Etat du système *après* la réalisation d'une tâche

Contraintes OCL

```
-- Une fois la tache "NextAction" accomplie, celle qui etait
-- en deuxieme sur la liste des actions devient la
-- nouvelle "NextAction"
context Do::do()
pre NextAction.size() > 0
post NextAction.second()@pre = NextAction.first()
```

Diagramme de séquence

Le scénario de la réalisation d'une tâche est décrit par la figure 4.6.

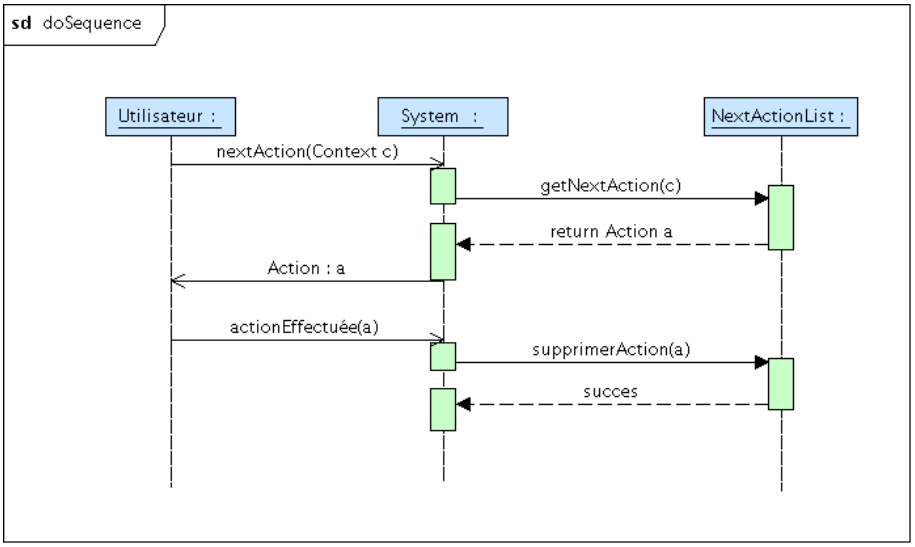


FIGURE 1.16 – Scénario de réalisation d'une tâche

Chapitre 2

Analyse finale

L'analyse de tous les cas d'utilisation du système nous a permis de concevoir le diagramme de classe de la figure 2.1.

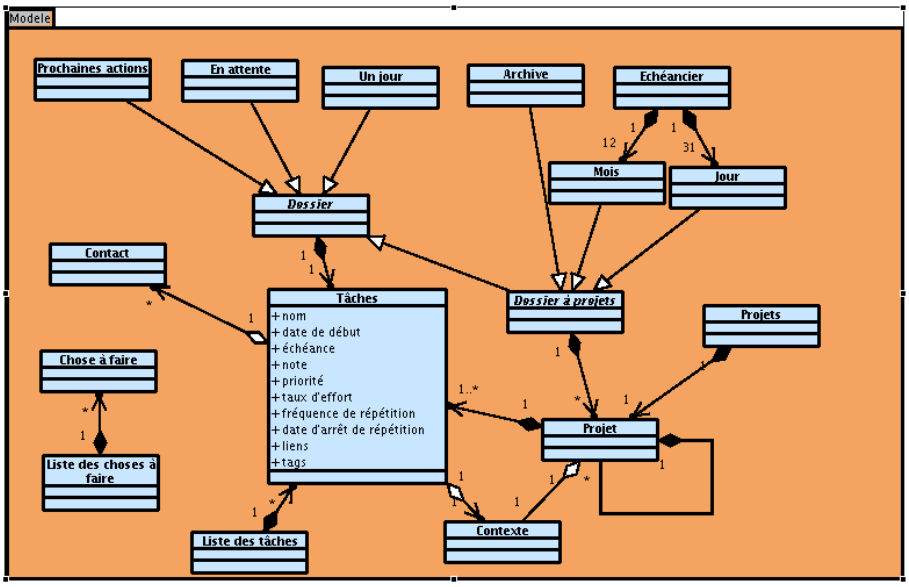


FIGURE 2.1 – Diagramme de classes du modèle

2.1 Conclusion

La modélisation du la méthode GTD nous a fait prendre conscience de certains aspects du problème auxquels nous n'avions pas pensé, et nous a fait réaliser à quel point chaque membre du groupe avait une vision foncièrement différente de la solution. En effet, il s'est avéré que nous n'étions pratiquement jamais en accord sur chaque cas d'utilisation, et il nous a fallu creuser le problème beaucoup plus en profondeur pour savoir qui avait l'idée la meilleure. Une fois toutes les notions clairement définies, l'avancement du projet a accéléré car nous pouvions alors travailler en équipe de manière beaucoup plus rigoureuse et efficace.

Chapitre 3

Spécification d'exigences logicielles

Sommaire

3.1 Introduction

Dans ce chapitre, nous allons spécifier les fonctionnalités du logiciel permettant de mettre en pratique la méthode GTD pour un particulier. On utilisera donc l'analyse effectuée dans le chapitre précédent, en prenant en compte le fait que nous spécifions cette fois un logiciel, et non une méthode généraliste.

3.1.1 Objectif

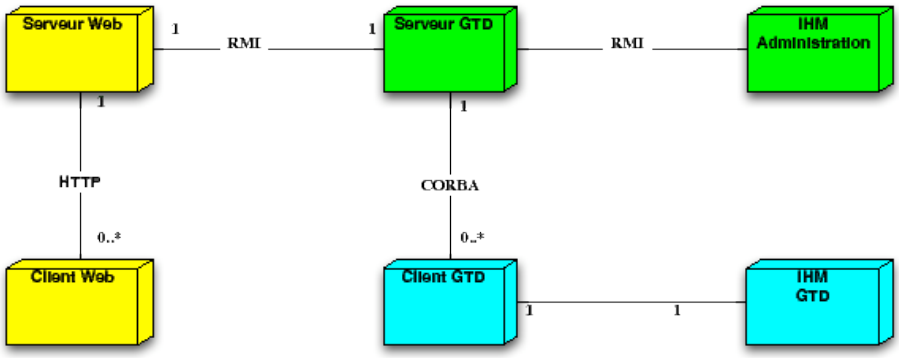
L'objectif poursuivi est donc de décrire de manière précise le comportement et les fonctionnalités du futur logiciel GTD. Nous allons également lister toutes les conditions qu'il doit vérifier, ainsi que tous les facteurs extérieurs desquels il dépend.

3.1.2 Conventions

Nous assumerons dans ce documents que l'utilisateur est capable d'utiliser un ordinateur et est au courant des exigences et effets liés à cette utilisation.

3.1.3 Audience

Ce document est essentiellement destiné aux développeurs du projet et au chef de projet, car ce sont eux qui devront mettre en place ce qui y est décrit. Cependant, le responsable de la relation client devra vérifier que c'est bien ce qui est attendu, et les testeurs devront s'y référer. Accessoirement, il doit être compréhensible, et apprécié par le correcteur.



3.1.4 Portée du document

Nous décrivons ici une application client instrumentant la méthode GTD, destinée à un usage professionnel et/ou personnel. L'application visée a pour but d'être accessible pour un assez large public, mais pourra éventuellement dépendre (dans le cas d'une utilisation professionnelle en particulier) d'un serveur distant. On devra donc également prendre en compte cette dépendance dans les spécifications. On décrira dans les parties suivantes des cas d'utilisation représentatifs des principales fonctionnalités offertes par le logiciel.

3.1.5 Définitions, acronymes et abréviations

Voir Dictionnaire de données (partie 1).

3.2 Description générale

Le logiciel que nous concevons devra respecter un certain nombre de spécifications provenant de différentes sources : utilisateur, système d'exploitation ou encore la méthode GTD. Ces spécifications concernent un large éventail d'aspects du logiciel :

3.2.1 Perspectives du produit

Le logiciel que nous développons s'inscrit dans la suite de logiciel de style GTD. Nous développons plus particulièrement le composant représentant la partie logiciel client d'un système client/serveur.
Le produit final fera l'interface entre un utilisateur et son compte GTD stocké sur le serveur. Le logiciel sera capable d'accéder aux informations du serveur (réception/envoi), mais aussi de fonctionner hors connexion (de faire la mise à jour sur le serveur plus tard ou de permettre une utilisation complètement locale).

3.2.2 Fonctions du produit

Le logiciel comportera différents sous-systèmes offrant chacun plusieurs fonctionnalités. Un système de gestion des "choses à faire/idées" : enregistrements de nouvelles idées à partir de l'utilisateur, demande à l'utilisateur des informations sur chaque idées pour appliquer un traitement.
Un système de gestion des tâches : place les tâches dans les catégories adéquates, vérifie si les tâches en attente doivent le rester, supprimer/archiver les tâches achevées. Il devra également disposer des éléments suivants :

- Un système de connexion utilisateur : connexion , déconnexion, préférences (changement du mot de passe, ...).
- Un système connexion au serveur : envoi (immédiat ou différé), réception (globale ou par paquets).
- Un système de cryptage/décryptage.

3.2.3 Caractéristiques et classes d'utilisateurs

Nous comptons orienter notre logiciel pour cibler les utilisateurs ayant de nombreuses choses à faire et qui ont besoin de s'organiser.
Le client pourra être aussi bien un homme d'affaire qu'un artisan, la seule nécessité est que le client ait un réel besoin d'organisation.
Nous projetons de développer le logiciel pour qu'un utilisateur lambda puisse l'utiliser. Un minimum de connaissance sera nécessaire mais pas plus que pour la méthode GTD elle-même.
Ce sont souvent les personnes occupées qui ont le plus besoin d'organisation, ces personnes utiliseront probablement le logiciel tous les jours car ils auront besoin d'ajouter ou de réaliser des tâches souvent.

3.2.4 Environnement opérationnel

Nous prévoyons que notre logiciel puisse être exécuter sur différents systèmes d'exploitation (MacOS, Linux, Windows). Il sera nécessaire que ce système d'exploitation possède une machine virtuelle Java.
Pour l'utilisation avec un serveur, une connexion (réseau ou internet) sera nécessaire pour la liaison avec lui.

3.2.5 Contraintes de conception et d'implémentation

Dans le but de déployer notre logiciel sur un grand nombre de plateformes, nous avons décidé de le développer en Java.
Pour une harmonie et un code propre, nous développerons un checkstyle [1] pour notre projet.

3.2.6 Documentation utilisateur

Nous comptons écrire trois types de documentation :

- une aide en ligne (dans le logiciel) en html ;
- un manuel (fichier texte décrivant les principales manipulations) en pdf ;
- un tutoriel vidéo sur YouTube [2].

3.2.7 Hypothèses et dépendances

On assume que le mode de connexion et le format des données pour communiquer avec le serveur sont normalisés avec l'équipe de développement du serveur GTD.

3.2.8 Exigences reportées

Etant soumis à des contraintes temporelles, nous avons choisi de reporter la spécification des intégrations avec des logiciels déjà présents sur la machine (calendrier, liste de contacts, etc.).

1. outil de contrôle de code, permet de vérifier le style d'un code source
2. `http://www.youtube.com/`

3.3 Fonctionnalités du logiciel

Dans cette section, nous reprenons les cas d'utilisation de la méthode GTD et nous les adaptons pour un système informatique.

3.3.1 Collecte des choses à faire

Le logiciel permet de recenser les choses à faire, de manière rapide et simple, à n'importe quel moment (y compris pendant l'exécution d'une autre tâche par le logiciel).

Description

Use Case: 6 – Collecter des choses à faire

CHARACTERISTIC INFORMATION

Goal in context : Récupérer des choses à faire et les ajouter à la liste de choses à faire
Scope : La méthode GTD
Level : Summary
Pre-conditions : L'utilisateur est identifié, et a des choses à faire non classées dans le système
Post-conditions : Toutes les choses à faire sont enregistrées.
Success End Condition : Toutes les choses à faire désirées ont été enregistrées pour l'utilisateur.
Failed End Condition : Les choses à faire données par l'utilisateur ont été perdues
Primary actor : L'utilisateur, toute personne voulant être mieux organisée
Trigger : L'utilisateur a demandé à rentrer des choses à faire

MAIN SUCCESS SCENARIO

1. L'utilisateur demande à ajouter des choses à faire
2. Le système enregistre les choses à faire tant que l'utilisateur en ajoute
3. L'utilisateur arrête la saisie

RELATED INFORMATION

Priority : Moyenne
Performance target : Quelques secondes par chose à faire
Frequency : Régulièrement (potentiellement tous les jours, ou même plusieurs fois par jour)
Channel to primary actor : Action directe de l'utilisateur

OPEN ISSUES

– Que se passe-t-il si le système perd une ou plusieurs choses à faire ?

Exigences fonctionnelles

EXIGE-1 : L'ajout de nouvelle chose à faire doit être disponible à tout moment, quel que soit la tâche effectuée par le système.

EXIGE-2 : Dès qu'une chose à faire est ajoutée, elle est sauvegardée de manière persistante, même si la saisie ne s'arrête pas.

3.3.2 Traitement

Le logiciel permet de passer en revue les choses à faire une à une, et offre la possibilité à l'utilisateur de choisir le traitement à effectuer : le faire tout de suite (implique suppression de la chose à faire), transformation en tâche (et spécification de la tâche), transformation en projet (et spécification de la première tâche de ce projet au moins) ou en sous-projet.

Description

Use Case: 7 – Traitement

CHARACTERISTIC INFORMATION

Goal in context : On effectue le traitement (création d'un projet contenant une décomposition de la tâche courante, l'utilisateur effectue la tâche, la délègue, la reporte, l'archive) d'une ou plusieurs choses à faire de la liste

Scope : La gestion des priorités après traitement, la création de la liste de choses à faire

Level : Summary

Pre-conditions : Liste de choses à faire non-vide

Post-conditions : Les tâches traitées apparaissent dans les donnée stockée

Success End Condition : Le chose à faire supprimée de la liste a été traitée (et la tâche a été effectuée, déléguée, transformée, ...)

Failed End Condition : Le traitement a échoué

Primary actor : L'utilisateur

Trigger : L'utilisateur

MAIN SUCCESS SCENARIO

1. L'utilisateur demande à lancer le traitement des choses à faire.
2. Le système lui affiche la première chose à faire
3. Le système demande le traitement adéquat à appliquer (voir Goal)
4. L'utilisateur entre des informations, afin de créer la tâche ou le projet correspondant à la chose à faire
5. Le système enregistre la tâche ou le projet
6. Le système supprime la chose à faire qui vient d'être traitée
7. Arrêt de la procédure de traitement

SUB-VARIATIONS

4a. L'utilisateur entre des informations, afin de créer la tâche ou le projet correspondant à la chose à faire

4a1. L'utilisateur effectue la tâche immédiatement, et saut à l'étape 6.

4a2. Il n'y a pas encore de contexte adapté à cette tâche, création d'un nouveau contexte.

7a. Arrêt de la procédure de traitement

7a1. On remplace l'arrêt du traitement par le retour à l'étape 2 (on passe à la chose à faire suivante).

RELATED INFORMATION

Priority : Moyenne

Performance target : Moins de deux minutes par chose à faire

Frequency : Régulièrement (une fois par semaine ?)

Channel to primary actor : Action directement effectuée par l'utilisateur

OPEN ISSUES

- Que se passe-t-il si la tâche créée n'est pas sauvegardée ?
- Que se passe-t-il si la chose à faire n'est pas supprimée ?

Exigences fonctionnelles

EXIGE-3 : Le traitement doit pouvoir être interrompu à tout moment, et on doit pouvoir le reprendre à l'endroit exact ou l'on était rendu.

EXIGE-4 : Une tâche ou un projet créé lors du traitement doit immédiatement être sauvegardé de manière persistante.

EXIGE-5 : Une fois qu'une chose à faire a été traitée, elle doit être supprimée de la liste.

EXIGE-6 : Lors d'une création (de tâche ou de projet), tous les champs obligatoire doivent être renseigné, afin de permettre l'organisation automatique des tâches.

3.3.3 Organisation

Description

Use Case: 8 – Organiser les éléments stockés dans les dossiers

---

CHARACTERISTIC INFORMATION

Goal in context : Gérer les priorités, les contextes et tout autres attributs pour définir l'ensemble des prochaines actions à partir des tâches existantes.

Scope : La création des tâches, leurs actions et leurs attributs.

Level : Primary Task

Pre-conditions : avoir des tâches déjà créées.

Post-conditions :

Success End Condition : Toutes les tâches ont été triées.

Failed End Condition : Tâches inorganisable, signalisation à l'utilisateur.

Primary actor : Le système

Trigger : Après le traitement/Après chaque modification

MAIN SUCCESS SCENARIO

1. Le système étudie une à une les tâches existantes et calcule leur priorité
2. Le système trie toutes les tâches en fonction de leur priorité (la plus forte priorité en tête)

SUB-VARIATIONS

2a. Deux tâches ont exactement la même priorité

2a1. Le système indique au sein de la tâche qu'il place avant l'autre qu'il existe également une autre tâche de même priorité ; l'utilisateur sera en conséquent averti en temps voulu

RELATED INFORMATION

Priority : Haute

Performance target : Invisible à l'utilisateur

Frequency : Après le traitement/Après chaque modification

OPEN ISSUES

– Que se passe-t-il dans le cas de conflit priorité/échéance ?

Exigences fonctionnelles

EXIGE-7 : L'organisation des taches doit pouvoir se faire sans intervention humaine

3.3.4 Revue

Le logiciel passe régulièrement en revue toutes les tâches, afin de redéfinir les priorités de chacune (par exemple, dans le cas ou une tâche arrive très bientôt à sa date d'échéance).

Description

Use Case: 9 – Vérification des données stockées

CHARACTERISTIC INFORMATION

Goal in context : Vérifier si aucune tâche dans En Attente ou dans Un Jour n'est passée à l'état actif et met l'échéancier à jour.

Scope : Les tâches

Level : Primary Task

Pre-conditions : Il y a des tâches à examiner.

Post-conditions : L'échéancier a été mis à jour.

Success End Condition : Toutes les tâches examinées ont été activées

Failed End Condition : Erreur système.

Primary actor : Le logiciel

Trigger : Au démarrage de l'application ou à minuit

MAIN SUCCESS SCENARIO

1. Vérifier dans En Attente si des tâches sont devenues actives
2. Vérifier dans Un Jour si il est possible de réaliser une tâche
3. Mettre à jour l'échéancier
4. Transférer les tâches dans Prochaines Action

RELATED INFORMATION

Priority : Haute

Frequency : A chaque fois qu'un utilisateur démarre le logiciel ou au moins une fois par jour

---

Exigences fonctionnelles

EXIGE-8 : Le passage en revue doit pouvoir être interrompu rapidement et à tout moment, et on doit pouvoir le reprendre à l'endroit exact ou l'on était rendu.

3.3.5 Faire

Le logiciel renvoie, en fonction d'un contexte donné par l'utilisateur, la (ou les) tâche(s) prioritaire(s).

Description

Use Case: 10 – Réaliser une tâche

CHARACTERISTIC INFORMATION

Goal in context : Proposer à l'utilisateur d'effectuer la tâche la plus prioritaire du dossier Prochaines Actions, dans le contexte ou il se trouve.

Scope : Le contenu des Prochaines Actions

Level : PrimaryTask

Pre-conditions : L'utilisateur a du temps pour réaliser une tâche et il y a des tâches à effectuer.
Post-conditions : L'utilisateur à réalisée la tâche prédéfini.
Success End Condition : Ancienne tâche archivée et la seconde Prochaine Action devient la première.
Failed End Condition : La tâche n'a pas été supprimée ou l'utilisateur n'a pas réalisé son travail.
Primary actor : L'utilisateur
Trigger : L'utilisateur

MAIN SUCCESS SCENARIO

1. L'utilisateur demande à faire une tâche
2. L'utilisateur donne le contexte dans lequel il se trouve.
3. Le logiciel choisit la tâche.
4. L'utilisateur indique qu'il a fini la tâche
5. La tâche est supprimée
6. Mise à jour des Prochaines Actions
7. L'utilisateur ne fait plus d'actions

EXTENSIONS

4a. L'utilisateur n'a pas fini la tâche.
4a1. Le logiciel ne modifie rien.
7a. L'utilisateur demande une nouvelle tâche à faire
7a1. Retour à l'étape 3.

SUB-VARIATIONS

6.1. Si une tâche est En Attente de la tâche réalisée, le système doit recalculer sa priorité, éventuellement la placer comme Prochaine Action à réaliser.

RELATED INFORMATION

Priority : Haute
Frequency : A chaque fois que l'utilisateur demande une tâche à effectuer

OPEN ISSUES

– Que se passe-t-il si l'utilisateur ne finit jamais de tâches ?

Exigences fonctionnelles

EXIGE-9 : Quand il demande une tâche à faire, l'utilisateur doit toujours récupérer la même tâche pour un contexte donné tant qu'elle n'est pas finie.

EXIGE-10 : Quand il demande une tâche à faire, l'utilisateur doit tout de même pouvoir sélectionner une tâche différente de celle proposée si il estime qu'elle ne convient pas.

3.3.6 Authentification de l'utilisateur

Canevas de Cockburn

Use Case: 11 – Authentification d'un utilisateur sur le logiciel

---

CHARACTERISTIC INFORMATION

Goal in context : Authentifier un utilisateur pour qu'il ait accès à ces données cryptées
Scope : Le logiciel
Level : Primary Task
Pre-conditions : L'utilisateur est enregistré
Post-conditions : L'utilisateur est authentifié
Success End Condition : L'utilisateur est authentifié
Failed End Condition : L'utilisateur n'est pas authentifié
Primary actor : L'utilisateur
Trigger : L'utilisateur

MAIN SUCCESS SCENARIO

1. L'utilisateur demande à s'authentifier
2. Le logiciel lui affiche une fenêtre d'authentification
3. L'utilisateur rentre ses identifiants
4. Le système vérifie si les identifiants sont corrects
5. L'utilisateur est authentifié

EXTENSIONS

4a. Les identifiants sont incorrects
4a1. Retour à l'étape 2

SUB-VARIATIONS

3.1. L'utilisateur a oublié son mot de passe ; le logiciel le lui renvoie par e-mail.

RELATED INFORMATION

Priority : Haute
Performance target : Obligatoire
Frequency : A chaque lancement du logiciel

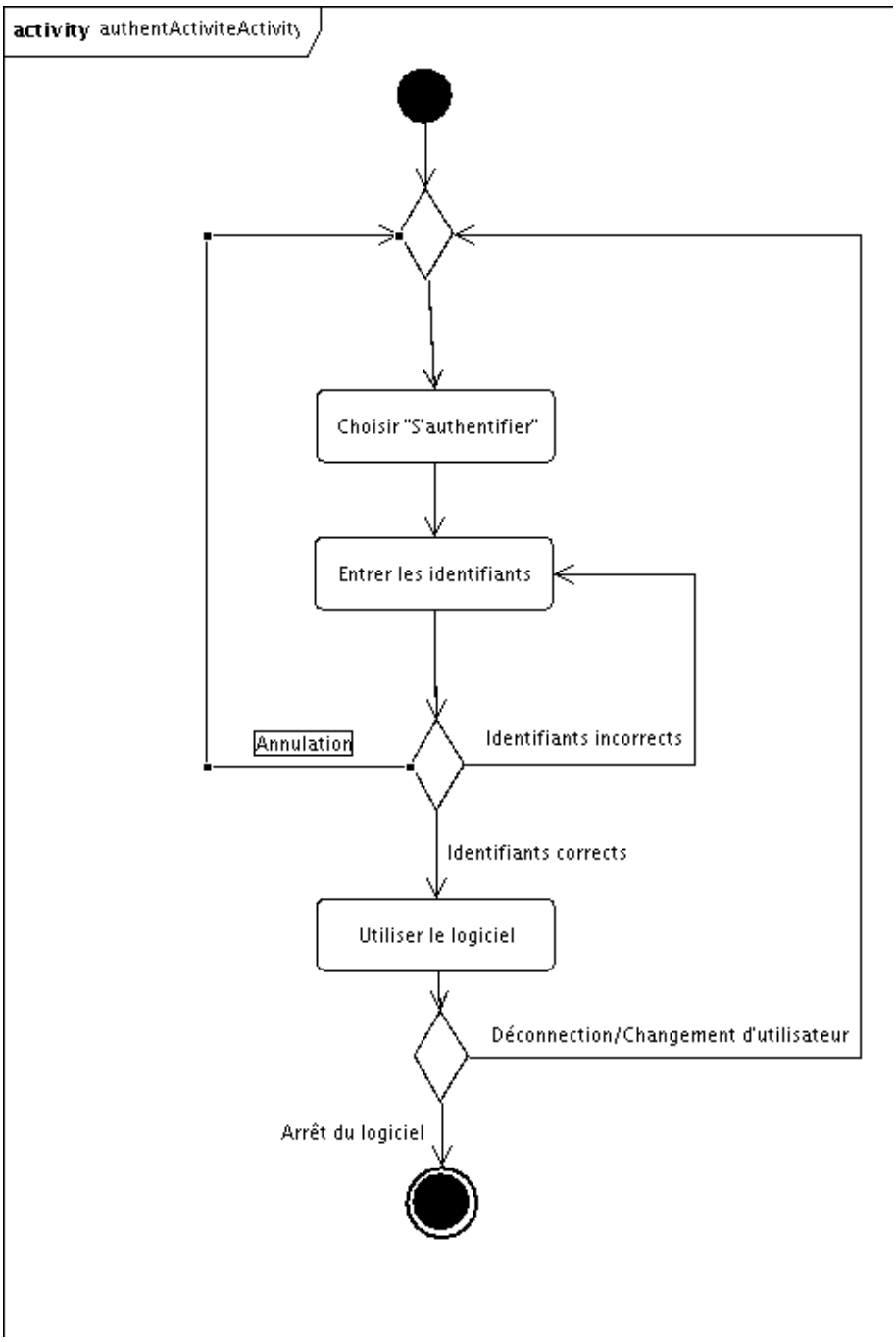


FIGURE 3.1 – Diagramme d'activité de l'authentification de l'utilisateur

Diagramme d'activité

L'authentification de l'utilisateur est représenté par le diagramme d'activité 3.1.

3.3.7 Connection au serveur

Canevas de Cockburn

Use Case: 12 – Synchroniser les données avec le serveur

CHARACTERISTIC INFORMATION

Goal in context : Se connecter avec le serveur pour mettre à jour les données
Scope : Le serveur
Level : Summary
Pre-conditions : disposer d'une interface réseau active
Post-conditions : déconnexion du serveur réussie
Success End Condition : les données ont été synchronisé
Failed End Condition : la synchronisation a échouée
Primary actor : Le logiciel
Trigger : Fréquence déterminée dans les préférences utilisateur/ Déclenchement manuel de l'utilisateur

MAIN SUCCESS SCENARIO

1. Le logiciel envoi des données modifiées au serveur
2. Le serveur retourne les données qui ont été modifiées sur lui et pas sur le client

EXTENSIONS

2a. Des conflits de mises à jour apparaissent : on a besoin de l'utilisateur pour les gérer

SUB-VARIATIONS

1.1. La connexion avec le serveur est perdue ; une synchronisation ultérieure est programmée.
1.2. Le serveur demande une authentification ; le logiciel doit s'authentifier auprès du serveur.

RELATED INFORMATION

Priority : Faible
Performance target : Invisible à l'utilisateur
Frequency : Fréquence déterminée dans les préférences utilisateur/ Déclenchement manuel de l'utilisateur

OPEN ISSUES

– Que se passe-t-il dans le cas de conflit de date entre les données en ligne et locales ?

Diagramme d'activités

L'échange avec le serveur est décrit par le diagramme d'activités 3.2.

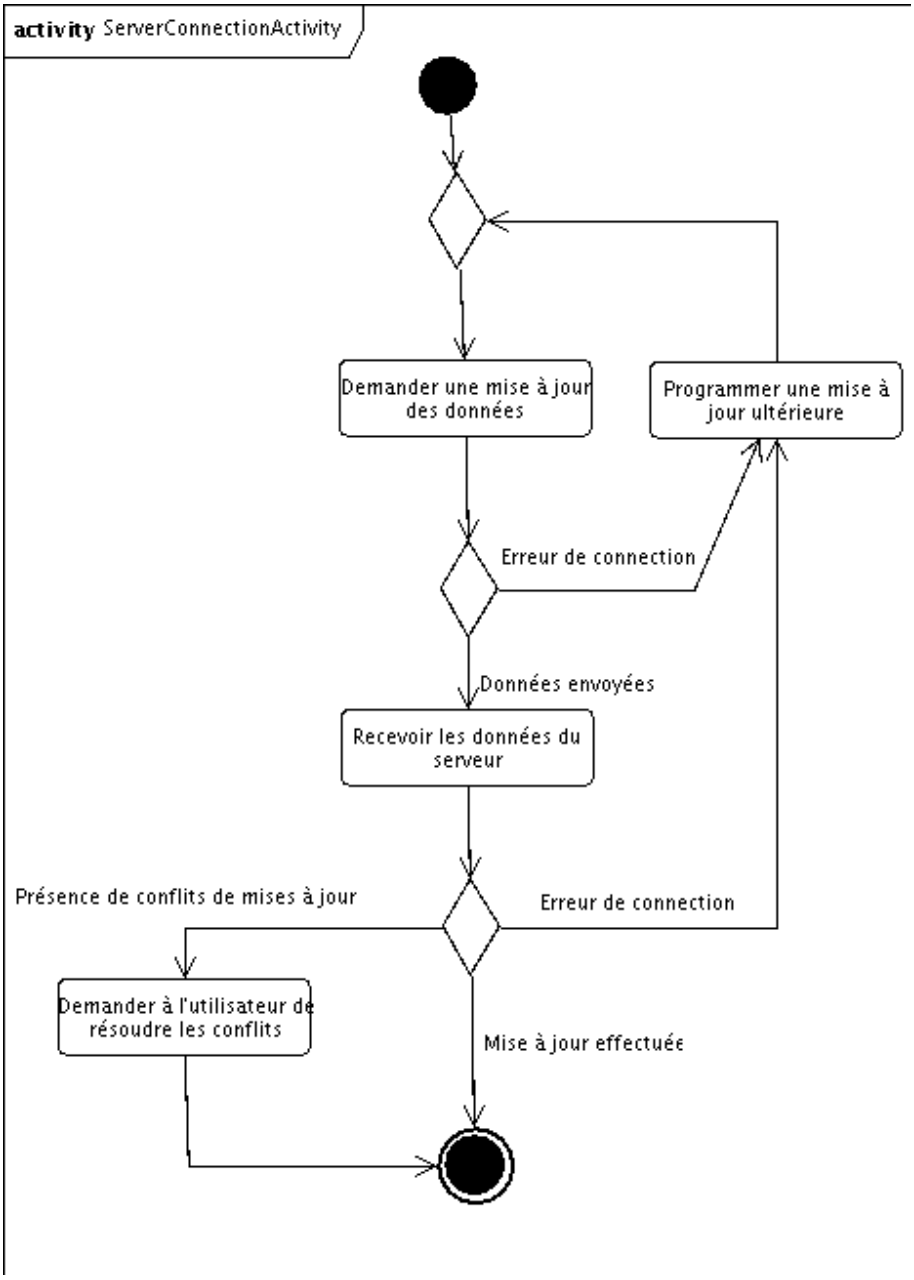


FIGURE 3.2 – Diagramme d'activité de la synchronisation avec le serveur

3.4 Exigences des interfaces externes

3.4.1 Interface utilisateur

IHM

L'IHM[3] doit être claire et non ambigüe. Elle doit disposer d'un bouton très accessible "Nouvelle chose à faire" ainsi que de "Traitement des choses à faire". Les tâches doivent être triées par contexte et accessibles de manière intuitive (onglets ou arborescence). Elle doit disposer d'un accès aisé à la prochaine action et aux informations la composant.

Si nous devions créer une première interface, notre premier jet pourrais ressembler au logiciel Getting Things Gnome[2]. (voir illustration 3.3)

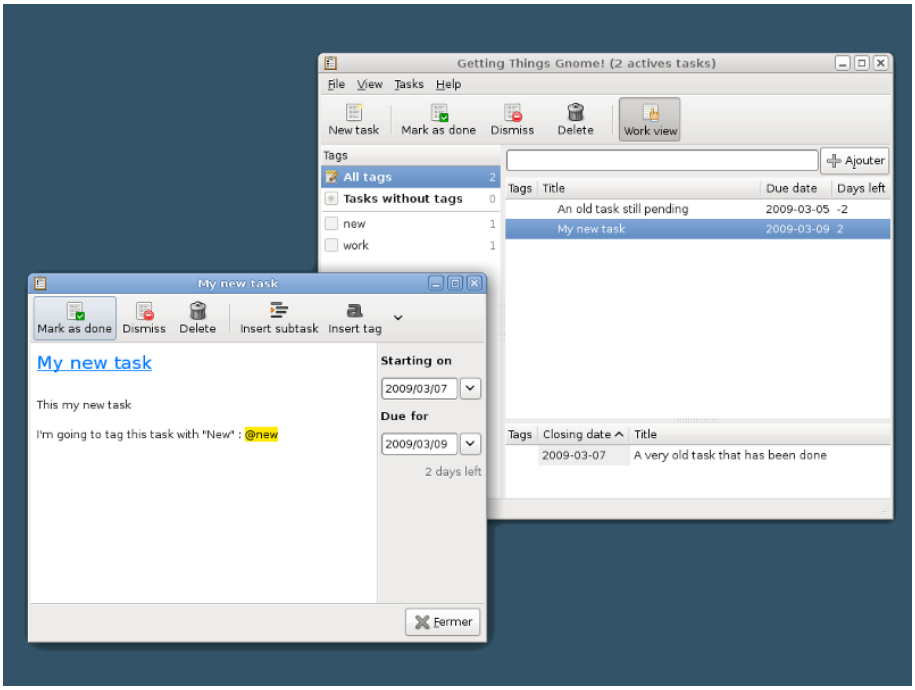


FIGURE 3.3 – Screenshot du logiciel Getting Things Gnome

Lors des prochains livrables, nous devons organiser l'IHM selon le canevas de présentation vu dans le cours du même nom.

Aides contextuelles

L'interface doit disposer d'aides contextuelle sur tous les éléments de menus et icônes, et d'éventuellement une aide en ligne. De plus, on peut ajouter des informations qui dépendent de l'expérience de l'utilisateur.

Documentation

Elle doit comprendre, de manière distincte :
– explications simples et concises de la méthode GTD.

3. Interface Homme Machine

– explications rapides des fonctionnalités de base de l'outil.
– explications complètes de toutes les fonctionnalités de l'outil.

Support de formation

L'utilisateur doit pouvoir apprendre par lui-même à utiliser l'outil, à l'aide d'un tutoriel concis.

3.4.2 Interface matérielle

La machine hôte doit disposer d'une connexion à internet, afin de pouvoir se connecter à un serveur distant. Le programme devra être installé sur un ordinateur, car elle ne sera pas défini pour un appareil mobile.

3.4.3 Interface logicielle

La machine hôte devra disposer d'un système d'exploitation pouvant supporter l'environnement d'exécution Java [4] car Java est le langage de programmation le plus portable, et permettant un développement très rapide. Nous ne savons pas encore quelles sont les bibliothèques ou bases de données qui seront utilisées. Ce point devra être éclairci lors des prochains livrables, voire lors de l'implémentation.

3.4.4 Interfaces ou protocoles de communication

Le programme doit offrir la possibilité de se connecter au serveur par l'utilisation de CORBA. L'interface de communication des objets doit nous être fournie par le serveur, et le programme doit pouvoir l'utiliser. Nous devrons donc nous mettre d'accord avec l'équipe chargée de l'implémentation de cette interface. Ce serveur pourra se trouver n'importe où, que ce soit sur une machine locale où sur internet. La communication doit se faire de manière sécurisée (cryptée) et acquittée, de manière à ne pas avoir de doublons ou de pertes d'information.

3.4.5 Contraintes de mémoire

Nous ne pouvons pas pour l'instant prédire combien de mémoire le programme utilisera ; nous compléterons ce point lors des prochains livrables. Cependant, il parait évident que notre logiciel devant pouvoir être utilisé par un maximum de personnes, il devra pouvoir s'exécuter sur une machine disposant de ressources mémoire moyennes, voire basses (par rapport à la moyenne des ordinateurs particuliers). Cela permettra également une réactivité accrue.

3.5 Autres exigences non-fonctionnelles

3.5.1 Utilisabilité

Outil simple

L'outil doit être aisément accessible à un utilisateur ne connaissant pas la méthode GTD ; cet utilisateur ne doit pas avoir à passer beaucoup de temps à se former sur l'outil avant d'être productif.

4. Le JRE actuel est la version 1.6

Outil complet

L'outil doit répondre aux attentes d'un utilisateur expérimenté à la fois sur la méthode GTD et sur l'utilisation de cet outil, c'est à dire qu'il doit être en mesure d'effectuer toutes les opérations qu'il désire, notamment celles qui sont récurrentes, de manière très rapide et efficace.

Outil respectant les standards

L'interface homme-machine doit respecter les standards des principaux développeurs de logiciels (Microsoft, Apple, etc.), afin d'être plus ergonomique et d'accélérer le processus d'adaptation de l'utilisateur à l'outil.

3.5.2 Fiabilité

Disponibilité de l'outil

L'outil nécessite de pouvoir fonctionner continuellement durant la période éveillée de son utilisateur, à cause de la saisie de choses à faire qui peut avoir lieu à n'importe quel moment.

Autres exigences de fiabilité

A ce niveau d'avancement du projet, les autres exigences de fiabilité de l'outil ne peuvent pas encore être spécifiées, à savoir :
- Durée moyenne de fonctionnement avant défaillance
- Durée moyenne de rétablissement
- Précision de l'outil
- Nombre maximum d'anomalies
- Criticité

3.5.3 Exigences de performance

Nous ne pouvons pas encore définir ce type d'exigences au niveau actuel d'avancement du projet.

3.5.4 Maintenabilité

Norme de codage

Afin de faciliter la relecture de notre code, nous avons choisi de respecter des critères de codage que nous définirons dans notre checkstyle.

3.5.5 Exigences de sûreté

Précautions à prendre

Cet outil présente les mêmes risques que n'importe quel autre logiciel, notamment vis-à-vis de l'épilepsie ou de la fatigue visuelle due à l'écran.

Sûreté vis-à-vis de la perte de données

Certaines données importantes de l'utilisateur n'existent qu'au sein de son outil ; il est donc primordial que ces données ne puissent pas être perdues. De même, en cas d'erreur du programme ou du système, les données doivent être récupérées.

3.5.6 Exigences de sécurité

Confidentialité des données

Afin de préserver la confidentialité des données, chaque compte d'utilisateur doit être protégé par mot de passe. Les fichiers assurant la persistance des données de l'utilisateur doivent également être sécurisé, par exemple à l'aide d'un cryptage. De même, les données transitant sur Internet (notamment pour la mise à jour entre le client et le serveur) doivent elles aussi être cryptées.

3.5.7 Attribut de qualité logicielle

Le logiciel, de par le fait qu'il doit être adressé à un assez large public, devra présenter une grande facilité d'apprentissage. Il devra néanmoins permettre, pour un utilisateur expérimenté, une utilisation fluide et efficace des fonctionnalités plus avancées. En outre, L'application devra bien sûr être la plus réactive possible (moins de deux secondes pour une action) pour offrir une utilisation agréable, et être assez robuste pour des systèmes (matériel et OS) peu efficaces.

3.6 Autres exigences

Nous ne pouvons pour l'instant définir les autre exigences du logiciel qui sera développé. Ce points sera à éclaircir lors des prochains livrables.

3.7 Classification des exigences fonctionnelles

Code	Exigence	Type
EXIGE-1	L'ajout de nouvelle chose à faire doit être disponible à tout moment, quel que soit la tâche effectuée par le système.	essentielle
EXIGE-2	Dès qu'une chose à faire est ajoutée, elle est sauvegardée de manière persistante, même si la saisie ne s'arrête pas.	essentielle
EXIGE-3	Le traitement doit pouvoir être interrompu à tout moment, et on doit pouvoir le reprendre à l'endroit exact ou l'on était rendu.	souhaitable
EXIGE-4	Une tâche ou un projet créé lors du traitement doit immédiatement être sauvegardé de manière persistante.	essentielle
EXIGE-5	Une fois qu'une chose à faire a été traitée, elle doit être supprimée de la liste.	essentielle
EXIGE-6	Lors d'une création (de tâche ou de projet), tous les champs obligatoire doivent être renseigné, afin de permettre l'organisation automatique des tâches.	essentielle
EXIGE-7	L'organisation des taches doit pouvoir se faire sans intervention humaine	souhaitable
EXIGE-8	Le passage en revue doit pouvoir être interrompu rapidement et à tout moment, et on doit pouvoir le reprendre à l'endroit exact ou l'on était rendu.	souhaitable
EXIGE-9	Quand il demande une tâche à faire, l'utilisateur doit toujours récupérer la même tâche pour un contexte donné tant qu'elle n'est pas finie.	souhaitable
EXIGE-10	Quand il demande une tâche à faire, l'utilisateur doit tout de même pouvoir sélectionner une tâche différente de celle proposée si il estime qu'elle ne convient pas.	essentielle

3.8 Conclusion

Maintenant que nous avons définis tous nos critères et toutes nos exigences, nous pouvons nous concentrer sur finir l'analyse pour pouvoir enfin réaliser notre logiciel.

Chapitre 4

Spécification des composants

Sommaire

4.1 Introduction

4.1.1 Objectif

Notre architecture étant conséquente, nous devons déterminer les différents composants du système et leur assemblage, afin d'en avoir une meilleur perception. Notre point de vue se situe à un haut niveau, afin de décrire les exigences de chaque composant. Il ne s'agit pas pour l'instant de savoir comment sera constitué chacun des composants.

4.1.2 Organisation du chapitre

Dans un premier temps, nous essaierons de présenter chacun des composants principaux du système. Nous aborderons ensuite les différentes interactions entre les composants par le biais de cas d'utilisation. Nous devrons enfin spécifier les interfaces des différents composants, et spécifier les types d'objets qu'il nous sera nécessaire de créer.

4.2 Description des composants

Une des partie les plus délicates est de définir les frontières entre les différents concepts.

Nous avons choisi de définir un composant ControleCentral qui régit l'ensemble du programme. Ce composant sera chargé d'utiliser les services fournis par d'autres composants internes. Nous appellerons ces différents composants : Authentification, Gestionnaire de persistance et MethodeGTD. Ces trois composants forment une base de l'application, qui devra néanmoins être étoffée pour obtenir un système complètement fonctionnel. Un composant doit nous permettre la communication avec le serveur. Nous le

nommerons donc CommunicationServeur. Nous aurons également besoin d'un autre composant qui sera chargé du cryptage des données et du hachage du mot de passe, que ce soit en local sur la machine, ou pour la communication avec le serveur distant [1].

Voyons maintenant cela plus en détail.

L'agencement des différents composants est décrit dans le schéma ci-dessous :

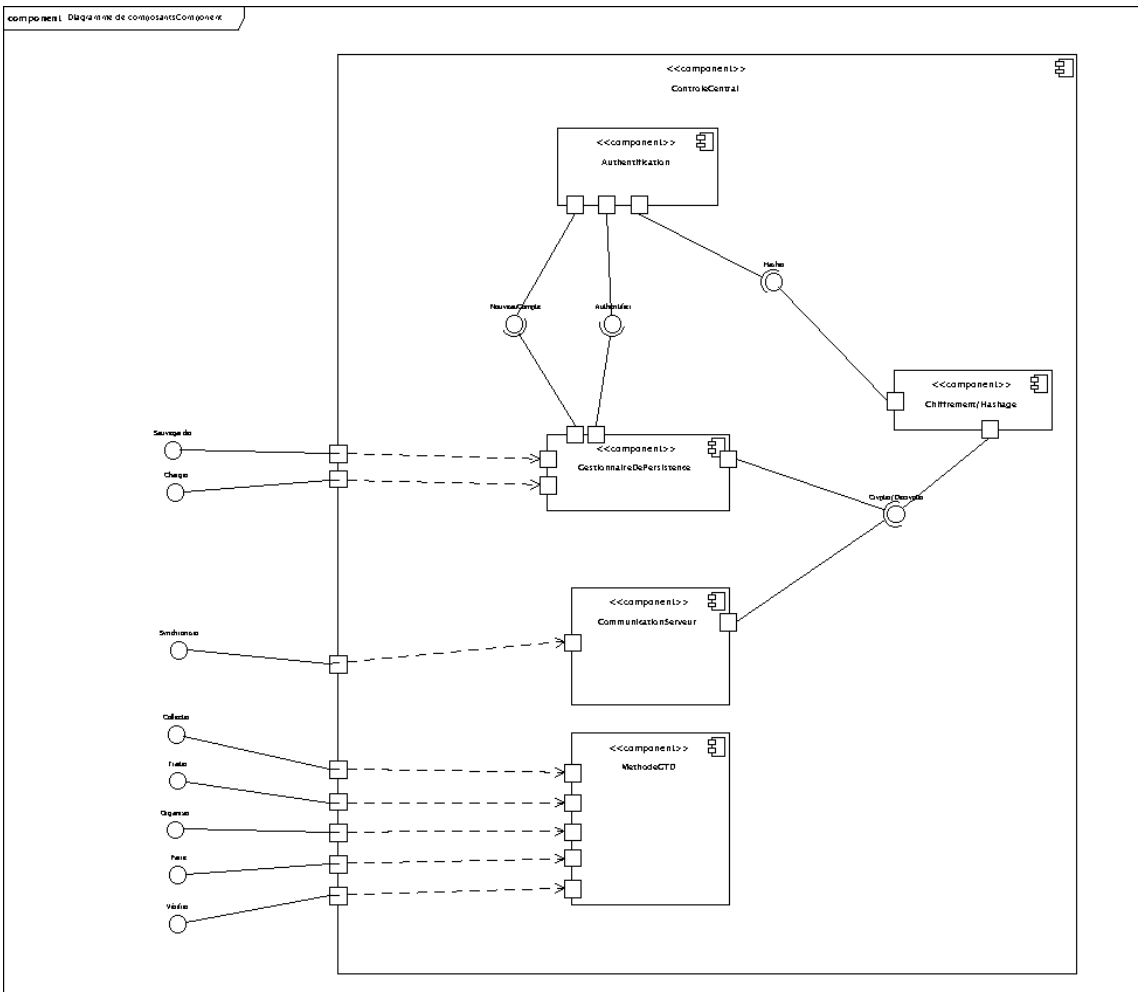


FIGURE 4.1 – Le composant ControleCentral et ses interfaces

4.2.1 Le composant ControleCentral

Voici le composant ControleCentral et les interfaces des services qu'il fourni. C'est par lui que devront transiter toutes les méthodes. Cela permettra de garder une gestion cohérente du système. Nous avons choisi de représenter ce composant global car il nous paraissait évident de le faire figurer, mais nous ne pouvions pas en faire un composant extérieur aux autres car il ne fourni aucune interface, il se contente d'organiser ses composants interne. Il s'agit donc plus d'une configuration qu'un composant, car il contient tous les autres composants du système.

1. si le serveur nous permet d'utiliser ce genre de fonctionnalités

4.2.2 Le composant MethodeGTD

C'est le composant principal du système. Il permet d'effectuer les différentes opérations de la méthode GTD sur la structure de donnée.

4.2.3 Le composant Chiffrement/Hachage

Ce composant aura deux tâches bien distinctes :

1. il permettra le chiffrement des données pour la sauvegarde des fichiers en local, ainsi que pour la communication avec le serveur.
2. il permettra également le hachage des mots de passe, afin que ceux-ci ne puissent être dévoilés.

4.2.4 Le composant Authentification

Cet outil devant pouvoir être utilisé potentiellement par plusieurs utilisateurs, le système aura besoin d'un composant capable de gérer les authentifications de chacun d'eux. Il doit donc permettre d'indiquer si un mot de passe donné est bien celui correspondant au login de l'utilisateur.

4.2.5 Le composant GestionnaireDePersistence

Le système devant pouvoir être utilisé uniquement de manière locale, nous devons pouvoir gérer la persistance des données, de manière à ce que l'utilisateur puisse éteindre son ordinateur par exemple. Nous devrons alors enregistrer notre structure de donnée sous forme de fichier. Ce composant aura aussi la responsabilité de charger les données d'un utilisateur contenues dans la base, après avoir vérifié son login et mot de passe.

4.2.6 Le composant CommunicationServeur

Ce composant a été défini afin de pouvoir communiquer avec le serveur. Il devra permettre la synchronisation des données entre le client local et le serveur distant, par le biais d'une communication cryptée. La représentation des données n'étant pas forcément identique entre le client et le serveur, le composant devra donc effectuer un travail de conversion des données.

4.3 Interactions

Dans ce qui suit, nous allons décrire les interactions principales entre les principaux acteurs de notre système. On se focalisera en particulier sur les cas d'utilisation qui seront le plus souvent rencontrés.

Voici, à haut niveau, la manière dont les différents composants collaborent entre eux. // Toutes les actions sont initiées par le composant ControleCentral(avec lequel interagira l'utilisateur). Les opérations complexes sur le compte actif sont effectuées par le composant MethodeGTD, mais la persistance des modifications est lancée par ControleCentral, et effectuée par le gestionnaire de persistance. Le chargement d'un compte utilisateur est effectuée par le gestionnaire de persistance. Celui-ci utilise le composant Authentification, pour savoir quel compte charger, et le composant Chiffrement, pour décrypter les données stockées.// L'authentification utilise également le composant Chiffrement(ou du moins sa fonction de hachage), afin de comparer le login et le mot de passe entrés avec ceux existants. Enfin, le composant CommunicationServeur utilise également le Chiffrement pour crypter la communication distante.

4.3.1 Traiter une chose a faire

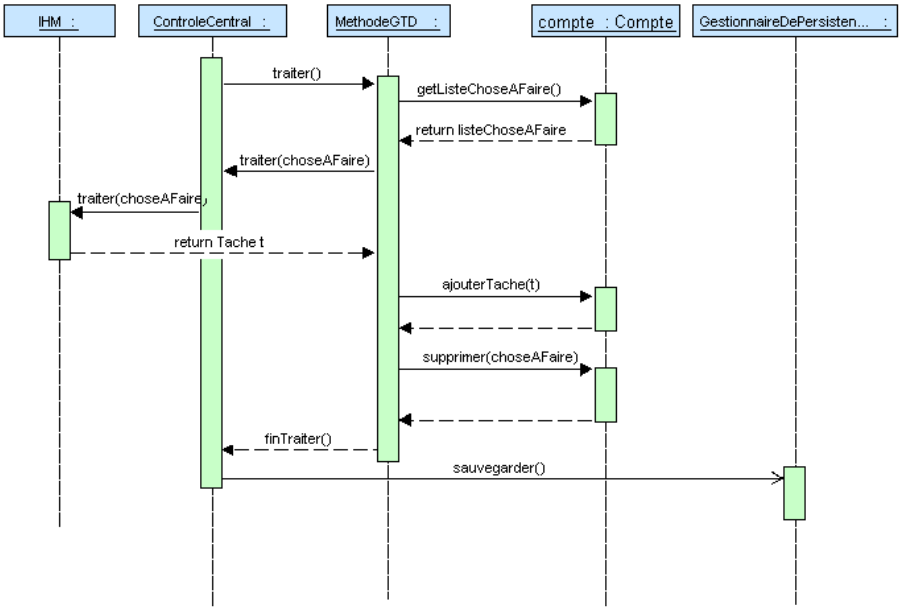


FIGURE 4.2 – Interaction – Traiter - Création d'une tâche

Dans le premier diagramme 4.2 , on peut voir la séquence d'appel intervenant lors du traitement d'une Chose A Faire, lorsque celle-ci donne lieu à la création d'une tâche.
Dans le diagramme suivant 4.3, on peut voir les différents chemins que peut prendre l'algorithme de traitement. On peut y voir les possibilités de création de projet, sous-projet, projet ordonné(dépendance entre les tâches) etc.

4.3.2 Organiser des tâches

Dans le diagramme 4.4, on peut observer une séquence du cas Organiser.

4.3.3 Examen des tâches

Le diagramme 4.5 représente l'examen des tâches.

4.3.4 Faire une tâche

Le diagramme 4.6 représente l'action de faire une tâche.

4.3.5 Authentification d'un utilisateur

Le diagramme d'activité 4.7 représente l'authentification des utilisateurs.

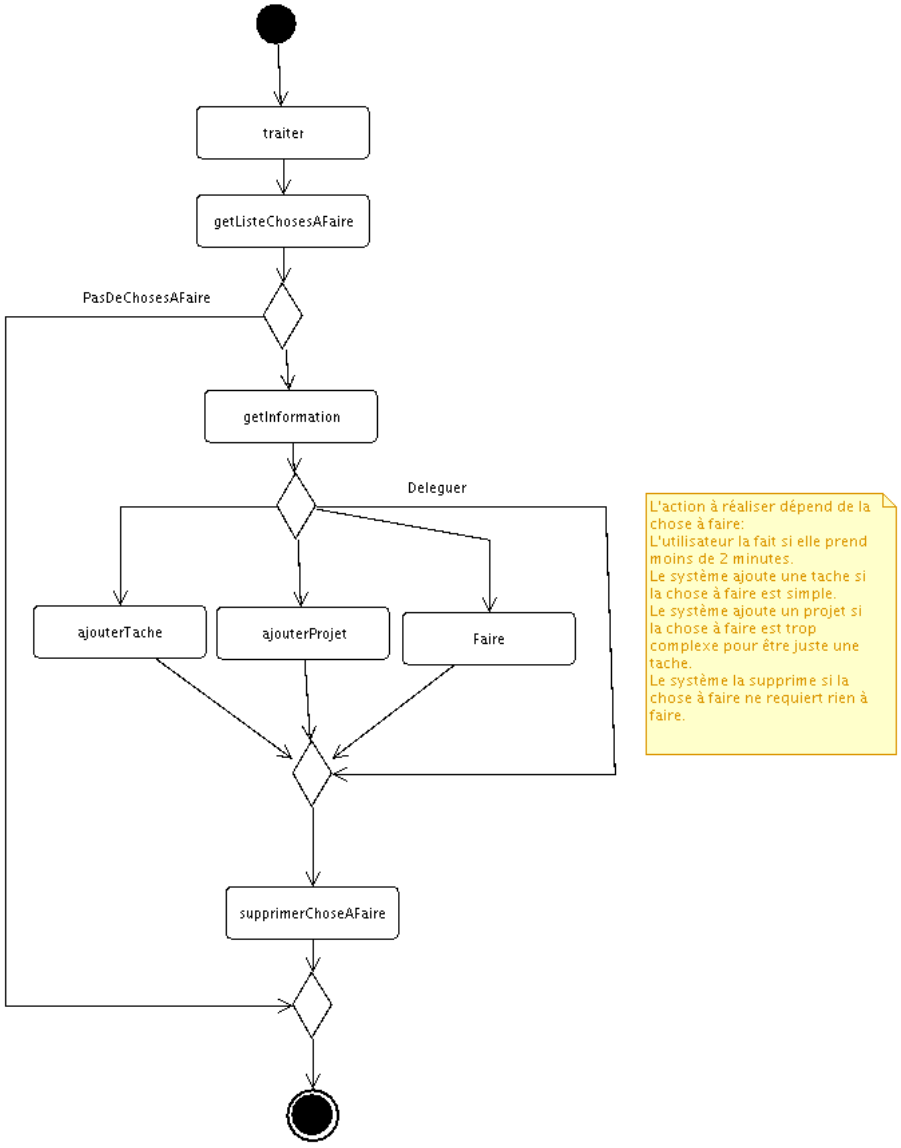


FIGURE 4.3 – Interaction – Traiter - vue générale

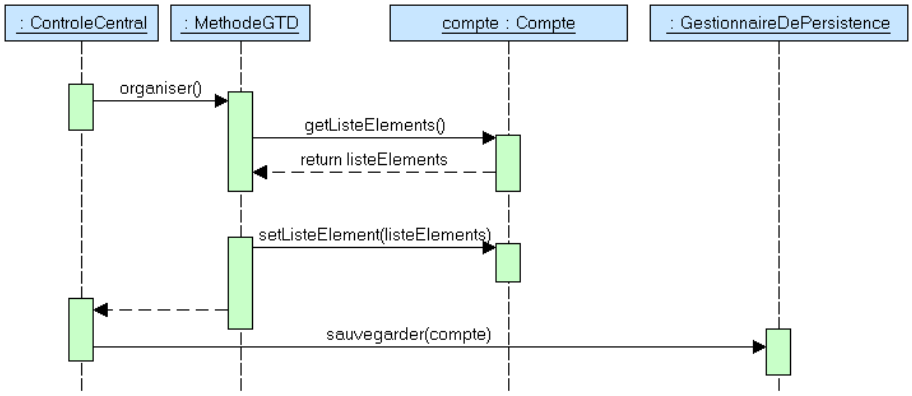


FIGURE 4.4 – Interaction – Organiser

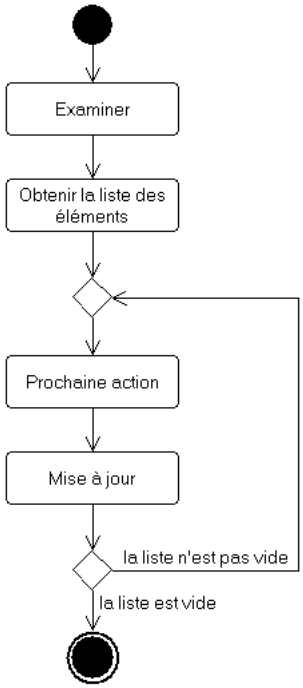


FIGURE 4.5 – Interaction – Examiner

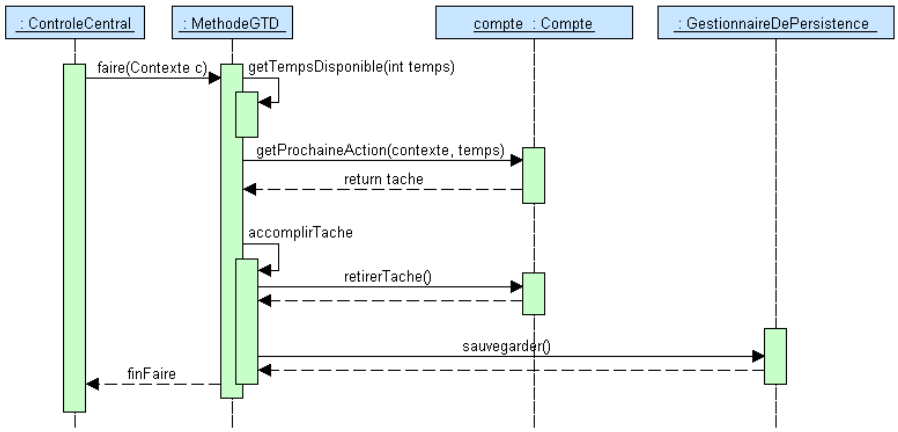


FIGURE 4.6 – Interaction – Faire

4.4 Spécification des interfaces

4.4.1 Interface du composant MethodeGTD

- **collecter (idee : String) : void**
  Permet de collecter un nouvelle idée

```
-- L'idee doit exister
context collecter(idee : String) : void
pre: idee <> null
        and idee.size()>0
```

- **traiter (idee : ChoseAFaire) : Tache**
  Renseigne une idée, pour la transformer en tâche.
- **organiser () : void**
  Calcule la priorité dynamique des tâches, et les ordonne selon cette priorité.
- **verifier (listeTache : Tache[*]) : void**
  Passe en revue une liste de tâches, pour mettre à jour la priorité dynamique.
- **faire (listeTache : Tache[*], contexte : Contexte) : Tache**
  Sélectionne la tâche la plus importante à faire selon un contexte donné.

4.4.2 Interface du composant GestionnaireDePersistance

L'interface du gestionnaire de persistance autorise le chargement/enregistrement d'un compte à partir/depuis un fichier à l'aide d'un login et d'un mot de passe.

- **sauvegarder() : void**
  Permet de sauvegarder le compte lié au login de l'utilisateur
- **charger(login : String, motDePasse : String) : Comptes**
  Charge et renvoie le compte lié au login si le mot de passe est correct.
- **enregistrerIdentifiants (tableCleValeur : Map<String, String>) : void**
  Enregistre la table des identifiants et mots de passe cryptés dans un fichier.
- **chargerIdentifiants () : Map<String, String>**
  Charge la table des identifiants et mots de passe cryptés depuis un fichier.

Nous avons définit la fonction sauvegarder comme ceci car le gestionnaire de persistance gardera en mémoire l'identifiant du compte chargé, ou bien pourra le récupérer depuis le contrôle central, qui devra

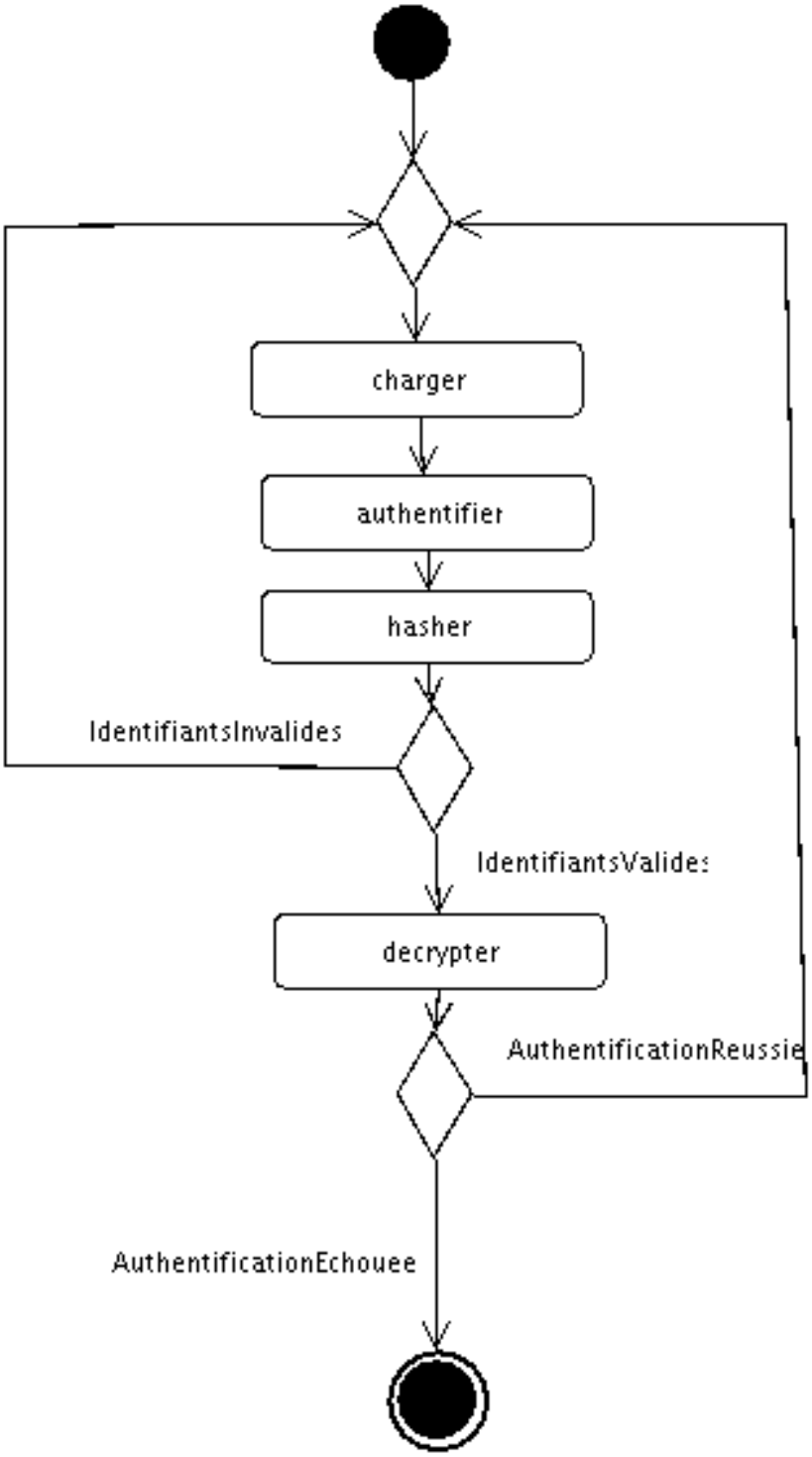


FIGURE 4.7 – Interaction – Authentifier

être un singleton. Nous verrons la meilleure façon de faire lors de l'implémentation.

4.4.3 Interface du composant Chiffrement/Hachage

L'interface du ce composant permet le chiffrement et le déchiffrement de fichier ainsi que le hachage (pour des mots de passe typiquement).

- **crypter(entree : Fichier) : Fichier**
  Crypte un fichier donné dans un autre fichier.
- **decrypter(entree : Fichier) : Fichier**
  Décrypte un fichier donné et restitue un fichier non crypté.
- **hasher(motDePasse : String) : String**
  Crypte le mot de passe de manière unidirectionnel.

Concernant le cryptage/décryptage, nous nous sommes basés sur des fichiers pour l'instant, bien que nous ne sachons pas comment ce système pourra être représenté. Il pourrait s'agir de transformer un Compte en chaine de caractères cryptée. Nous verrons cela plus en détail lors de l'implémentation.

4.4.4 Interface du composant Authentification

L'interface de ce composant permet l'authentification des utilisateur.

- **authentifier (nom : String, motDePasse : String) : boolean**
  Authentifie ou non un utilisateur suivant un mot de passe donné.
- **nouveauCompte(nom : String, motDePasse : String) : void**
  Associe un nouvel utilisateur et son mot de passe.

```
-- Le nom ne doit pas etre la valeur null.
-- La taille du nom est superieur a 0.
contexte nouveauCompte(nom, motDePasse) : void
pre: nom <> null
        and nom.size > 0
```

4.4.5 Interface du composant CommunicationServeur

- **synchroniser(compte : Comptes) : void**
  Permet de synchroniser l'ensemble des tâches avec le serveur.

4.5 Spécification des types utilisés

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

4.5.1 Element, Tache et Projet

Afin de permettre un traitement identique sur les différentes tâches et projets, nous avons utilisé le patron de conception composite.

Element

Nous avons donc défini une classe mère Element, qui contient les attributs et opérations commun aux classes Tache et Projet.

id : l'identifiant unique de la tâche.

nom : le nom de la tâche.

notes : la description de la tâche.

prioriteDynamique : la priorité calculée par la méthode GTD.

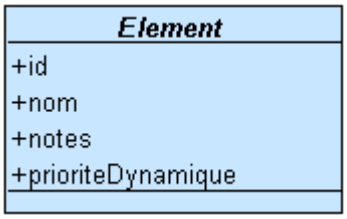


FIGURE 4.8 – La classe Element

Tache

La classe **Tache** représente une tâche. Elle possède les attributs suivants :

priorite : la priorité fixée par l'utilisateur.

tauxEffort : le taux d'effort demandé par la tâche.

liens : liste des liens internet liés à la tâche.

tags : la liste des tags associés à cette tâche.

isActive : détermine si la tâche est activée ou non.

FIGURE 4.9 – La classe Tache

Projet

Nous avons décidé de spécifier deux types de projets : ordonné ou non. Un projet ordonné permet de gérer des dépendances entre les tâches qui le composent (par exemple la deuxième tâche dépend de la première d'un même dossier). Nous avons choisi cette méthode car cela permet d'avoir des dépendances en arborescence, et donc évite les cycles de dépendance qu'il pourrait y avoir si la dépendance n'était qu'un attribut d'une tâche (ou ici d'un élément). Nous avons donc une classe abstraite Projet, et deux classes concrètes **ProjetOrdonne** et **ProjetNonOrdonne**. Chaque classe concrète peut alors choisir la représentation de sa liste interne.

La classe **Projet** représente un ensemble de tâches. Un projet possède les attributs suivants :

listeElement : les éléments contenus dans le projet.

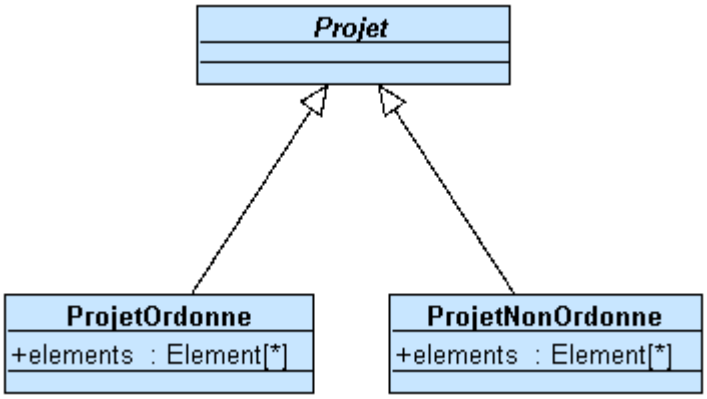


FIGURE 4.10 – La classe Projet

4.5.2 Contexte

La classe **Contexte** représente le contexte associé à une tâche (cf figure 4.11). Elle possède les attributs suivants :

nom : le nom du contexte.

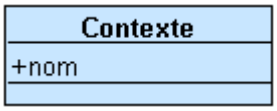


FIGURE 4.11 – La classe Contexte

4.5.3 Echeancier

La classe **Echeancier** représente l'échéancier d'une tâche (cf figure 4.12). Elle possède les attributs suivants :

dateDebut : La date du début de la tâche.

echeance : l'écheance de la tâche.

frequence : la fréquence de répétition de la tâche.

dateArret : la date d'arrêt de la répétition.

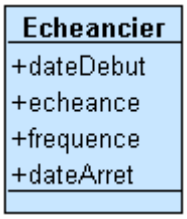


FIGURE 4.12 – La classe Echeancier

4.5.4 Contact

La classe **Contact** représente les contacts auxquelles les tâches peuvent être associées (cf figure 4.13). Elle possède les attributs suivants :

nom : le nom du contact.

eMail : son adresse e-mail.

telephone : son numéro de téléphone.

adresse : son adresse postale.

FIGURE 4.13 – La classe Contact

4.5.5 Compte

La classe **Compte** représente le compte auquel est associé un login. C'est cette classe qui contient l'ensemble des éléments (tâches et projets) d'un utilisateur (cf figure 4.14). Elle possède les attributs suivants :

nom : le login du compte.

mdp : le mot de passe du compte.

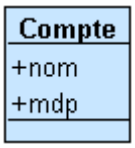


FIGURE 4.14 – La classe Compte

4.5.6 Ensemble de la structure de donnée

Voici donc la corrélation des divers éléments de notre structure de donnée (cf figure 4.15).

4.6 Conclusion

Maintenant que les frontières du système sont établies, nous allons pouvoir détailler chaque composant défini précédemment.

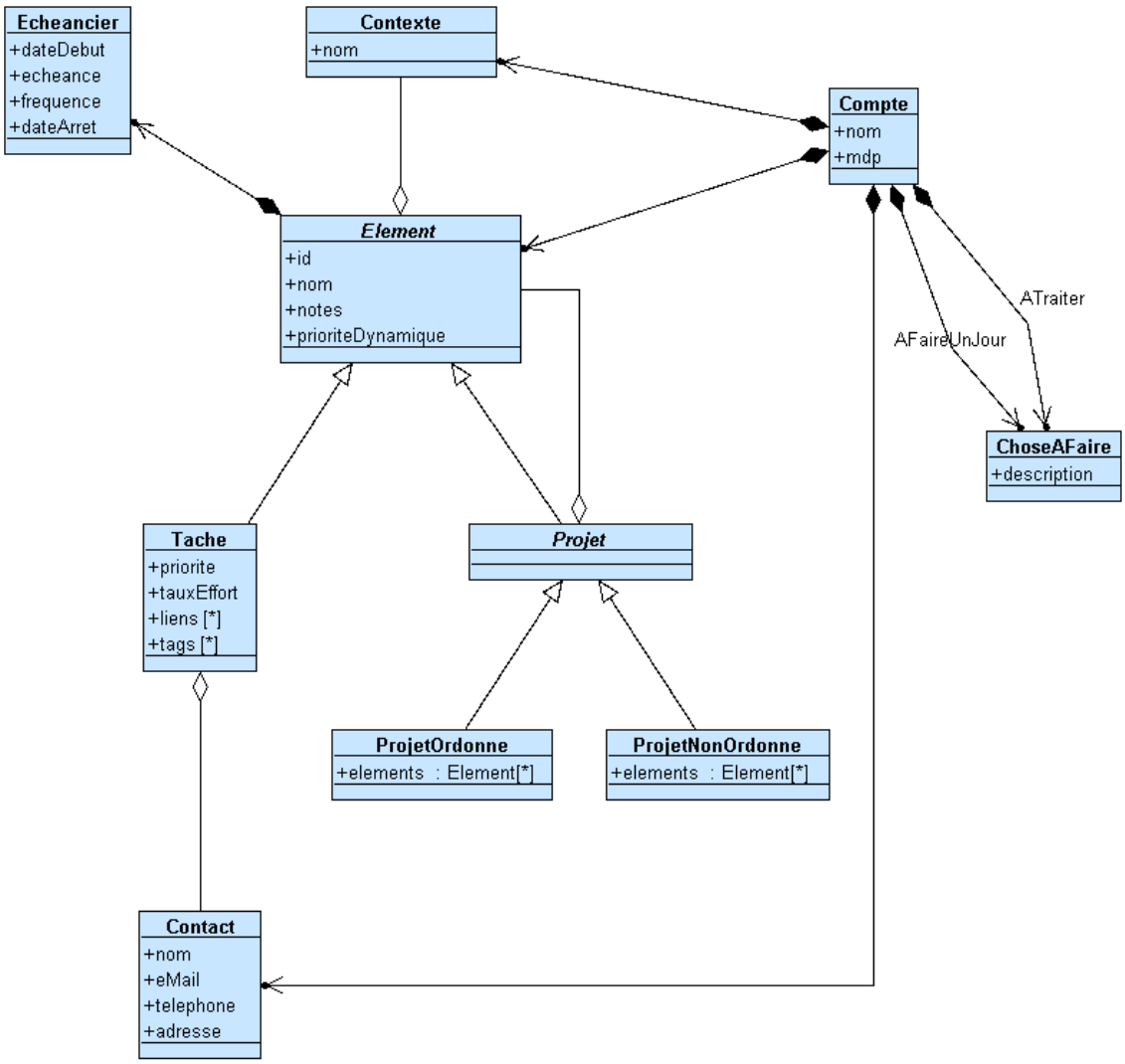


FIGURE 4.15 – La structure de donnée

Chapitre 5

Architecture

Sommaire

Introduction

Maintenant que tous les composants ont été défini, on peut se pencher plus en détails sur les différents niveaux composant notre logiciel et spécifier comment on passe du modèle architectural au modèle physique. Pour faire cela, on va définir sous quelle forme de tables relationnelles nos classes pourront persister, expliquer les patrons de conceptions que nous avons utilisé dans notre modèle de classes et comment on génère le code à partir de celui-ci.

5.1 Architecture physique

Le logiciel lourd que nous écrivons se découpe en différentes couches :

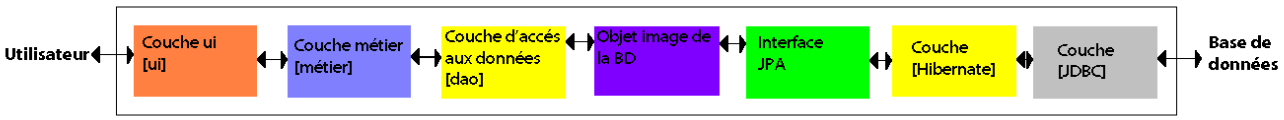


FIGURE 5.1 – Architecture de l'application

1. la couche [ui] (User Interface) est la couche qui dialogue avec l'utilisateur via une interface graphique. Elle a pour rôle de fournir des données provenant de l'utilisateur à la couche [2] ou bien de présenter à l'utilisateur des données fournies par la couche [2].
   – Représente les fenêtres Utilisateur.

2. la couche [2], appelée ici [metier] est la couche qui applique les règles dites métier, c.a.d. la logique spécifique de l'application, sans se préoccuper de savoir d'où viennent les données qu'on lui donne, ni où vont les résultats qu'elle produit.
   – Représente les différentes fonctions de notre programme : collecter traiter ...
3. la couche [3], appelée ici [dao] (Data Access Object) est la couche qui fournit à la couche [2] des données pré-enregistrées (fichiers, bases de données, ...) et qui enregistre certains des résultats fournis par la couche [2].
   – La couche dao rassemble les descripteurs des classes qui seront "persistées".
4. La couche [4] des objets-images de la BD est appelée "contexte de persistance". Elle fait le lien entre la couche [dao] qui s'appuie sur Hibernate pour faire des actions de persistance (CRUD, create - read - update - delete). Cette couche, fournie par la couche inférieure [5] représente la base de donnée : on n'utilise pas directement la base de données physique.
   – Appellée lors des ajouts suppressions ... sur la base de données.
5. L'interface [JPA] est un ensemble générique d'interfaces permettant des accès plus bas niveau (sur les couches physiques comme la base de données) sans avoir à mettre les mains dans le camboui. Elle permet de rendre toutes ses couches inférieures réutilisables.
   – Chaque classe qui doit être "persistée" doit posséder un classe Entity faisant justement partie de la couche JPA.
6. La couche [Hibernate] traduit les actions de persistence en ordres SQL.Pour les actions d'interrogation de la base (le SQL Select), Hibernate fournit au développeur, un langage HQL (Hibernate Query Language) pour interroger le contexte de persistance [4] et non la BD elle-même. Cette couche fait le pont entre le monde relationnel et le monde objet.
   – Hibernate est une couche transparente et fourni des outils, et donc n'apparait pas directement dans le code.
7. la couche [JDBC] est la couche standard utilisée en Java pour accéder à des bases de données. C'est ce qu'on appelle habituellement le pilote Jdbc du SGBD.
   – Représente la base de données physique.

Nous obtenons ainsi une chaîne reliant l'utilisateur (hors logiciel) jusqu'à la base de données (couche la plus physique) contenant ses informations. Chaque couche possède des règles et discute avec les couches l'entourant. Chaque couche modifie ou interprète les informations pour la couche inférieure/supérieure et permet ainsi le fonctionnement du logiciel.

5.2 Schémas des bases de données

5.2.1 Logins et mots de passe

La base de données des identifiants n'est jamais chargé. Quand un utilisateur entre un mot de passe, le programme chiffre celui-ci et parcourt le fichier pour comparer si le mot de passe chiffré qu'il vient de générer et celui stocké pour cet utilisateur sont les mêmes. Cette opération est simple et ne doit être faite que peu souvent, c'est pourquoi on peut se permettre de ne pas charger le contenu du fichier dans une base de données.

5.2.2 Java Persistence API

Les comptes sont stockés localement dans des fichiers et lorsque l'utilisateur a entré un mot de passe correct, le programme charge le compte lié à ce mot de passe dans la base de donnée pour y accéder et modifier les informations. C'est ainsi que la persistance est géré pour les comptes hors connexion ou dont la connexion est inactive.

Le compte est géré de manière différente selon le programme (classes) et la base de données (tables). On doit donc écrire des schémas de "traduction" de la base de données relationnelle aux classes. Nous avons choisi de créer une table par chemin d'héritage.

On commence en réalisant les schémas de toutes les classes isolées (ne dépendant pas de chemins d'héritage) :

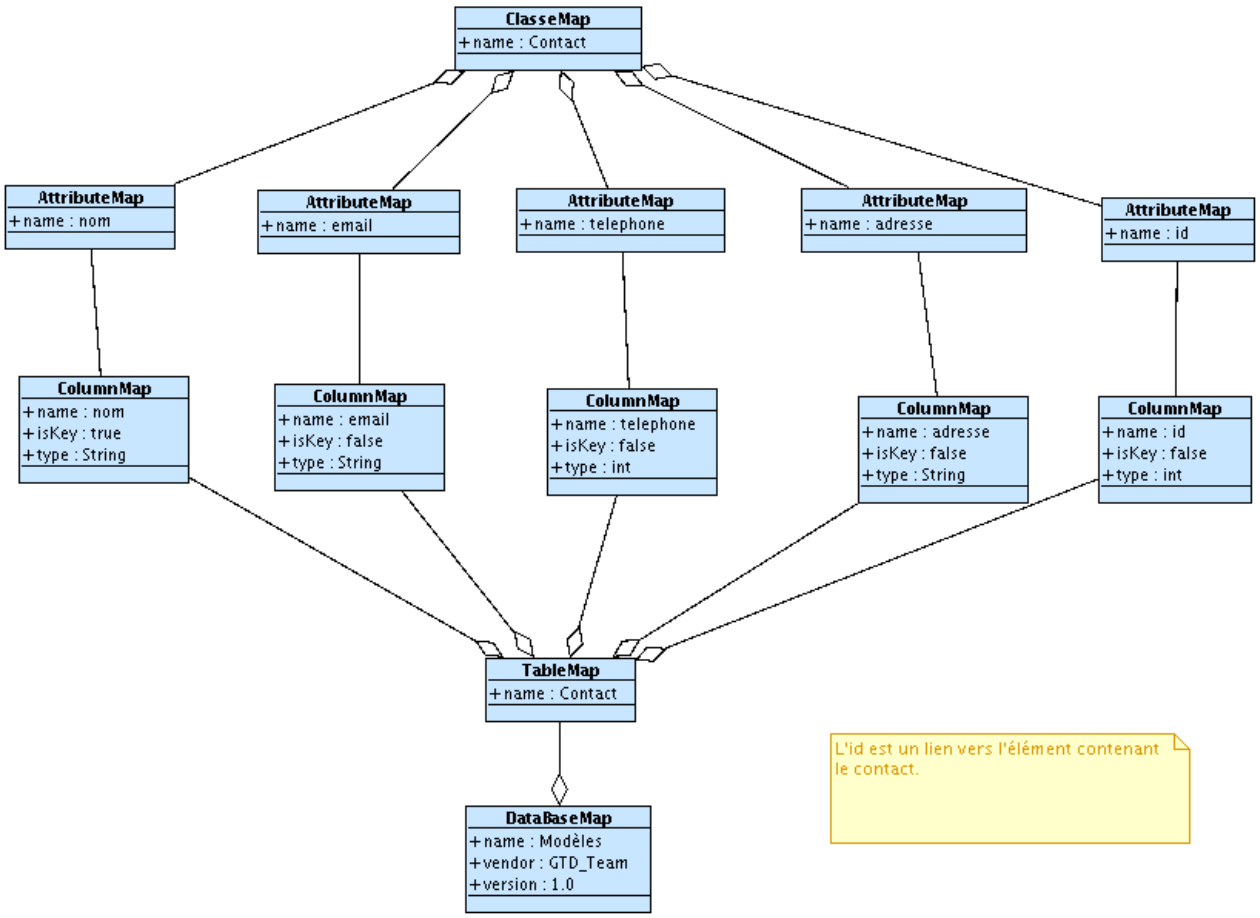


FIGURE 5.2 – Schéma de la base de données : Table Contacts

La table "Contact" 5.2 contient les colonnes liées aux attributs de la classe associée, plus l'id de la tâche, du projet ou directement du compte auquel elle est liée.

De même que la classe précédente, la table "Contexte" 5.3 contient l'id de l'élément qui l'utilise (de même pour les tables Tags 5.4, "Echeancier" 5.5 et "Participant" 5.6).

A la différence des tables précédentes, la table "ChoseAFaire" 5.7 contient une colonne pour mémoriser le participant de l'action (il peut ne pas y en avoir).

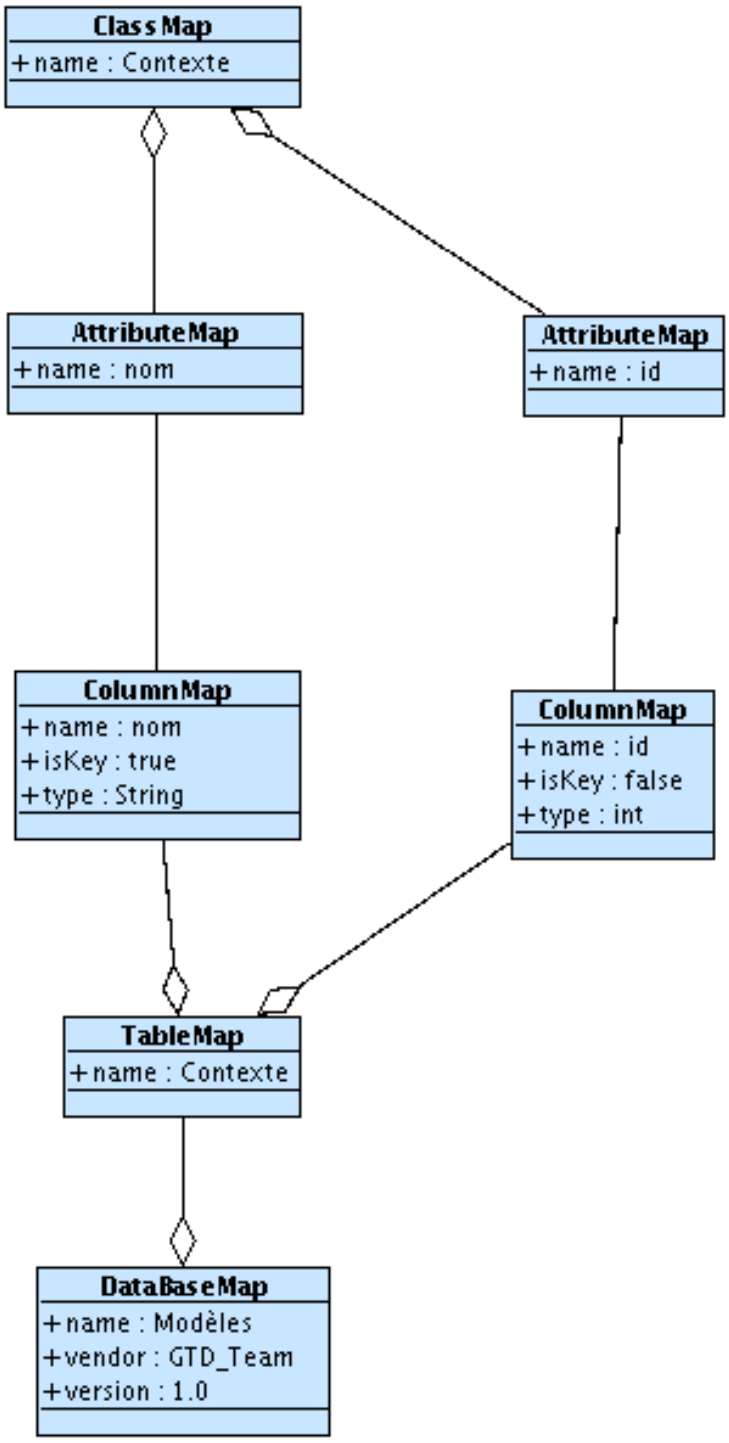


FIGURE 5.3 – Schéma de la base de données : Table Contexte

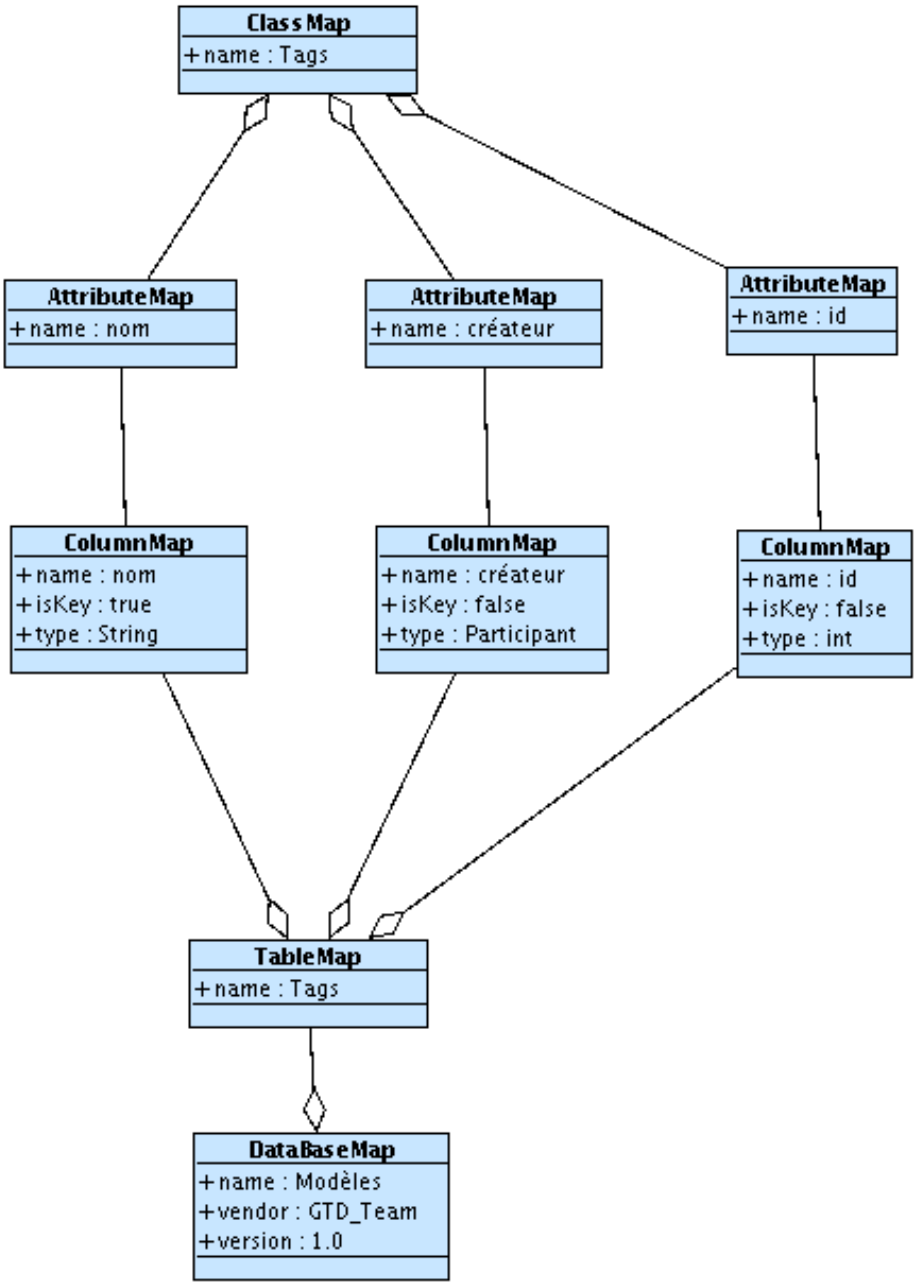


FIGURE 5.4 – Schéma de la base de données : Table Tags

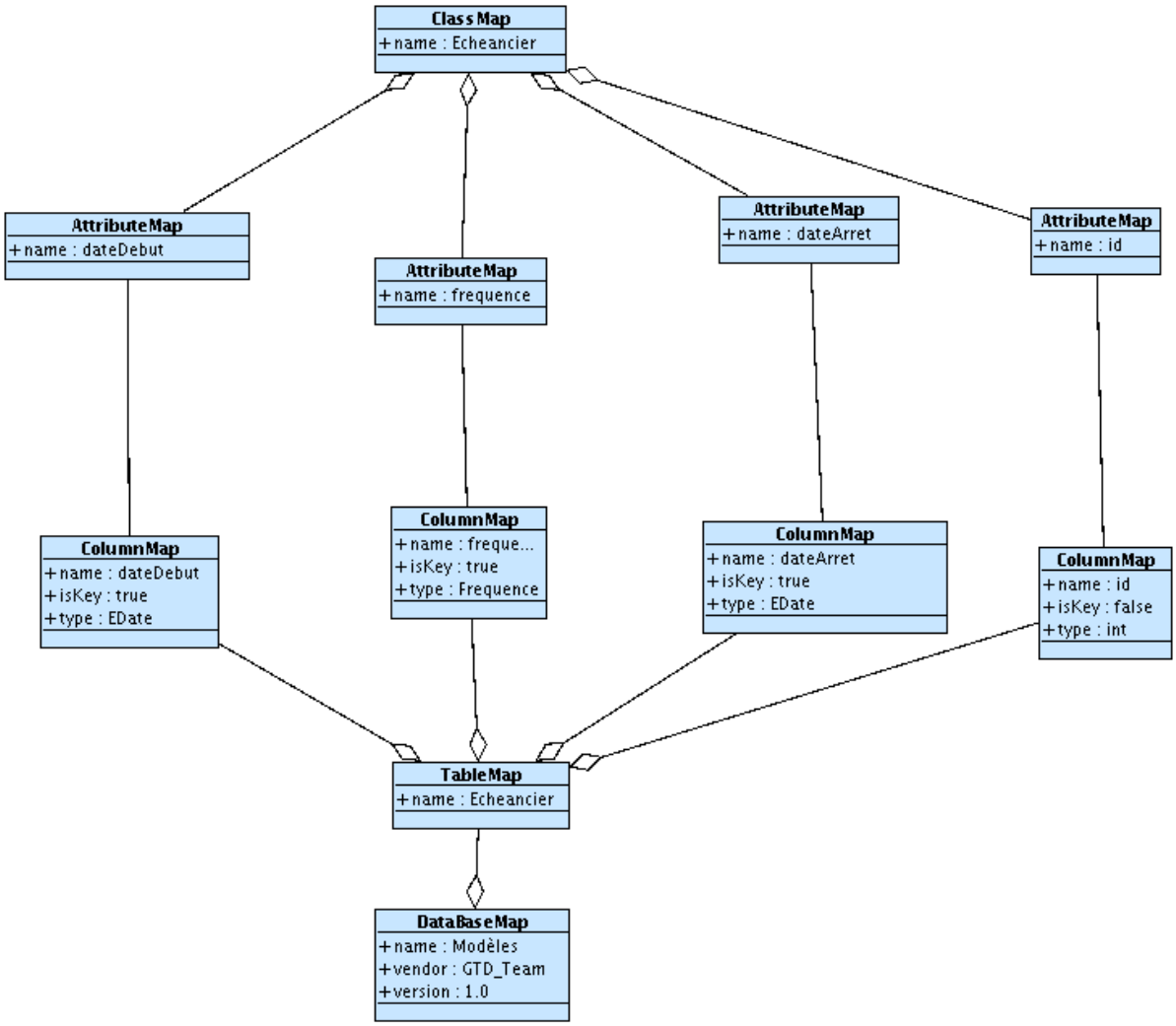


FIGURE 5.5 – Schéma de la base de données : Table Echeancier

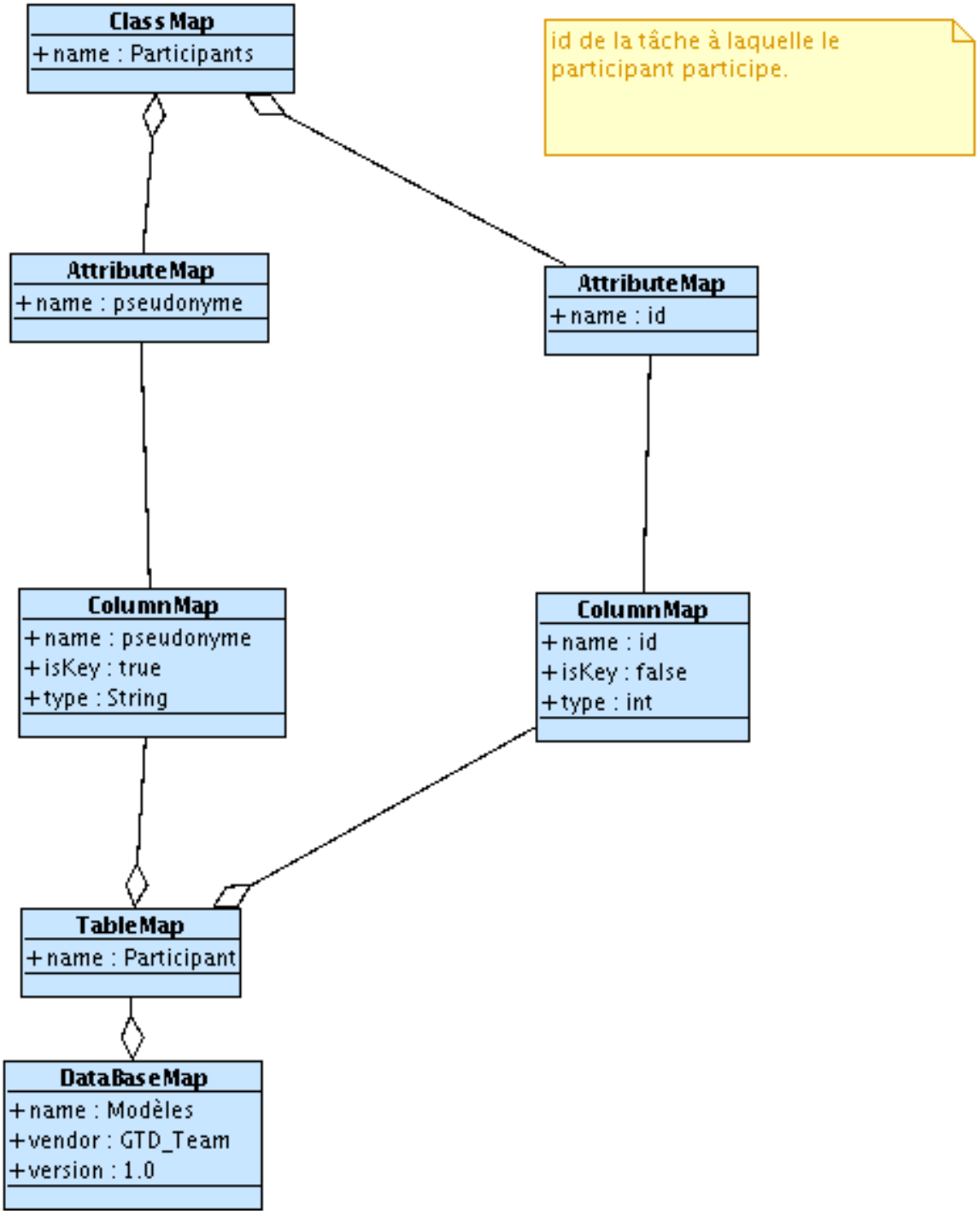


FIGURE 5.6 – Schéma de la base de données : Table Participant

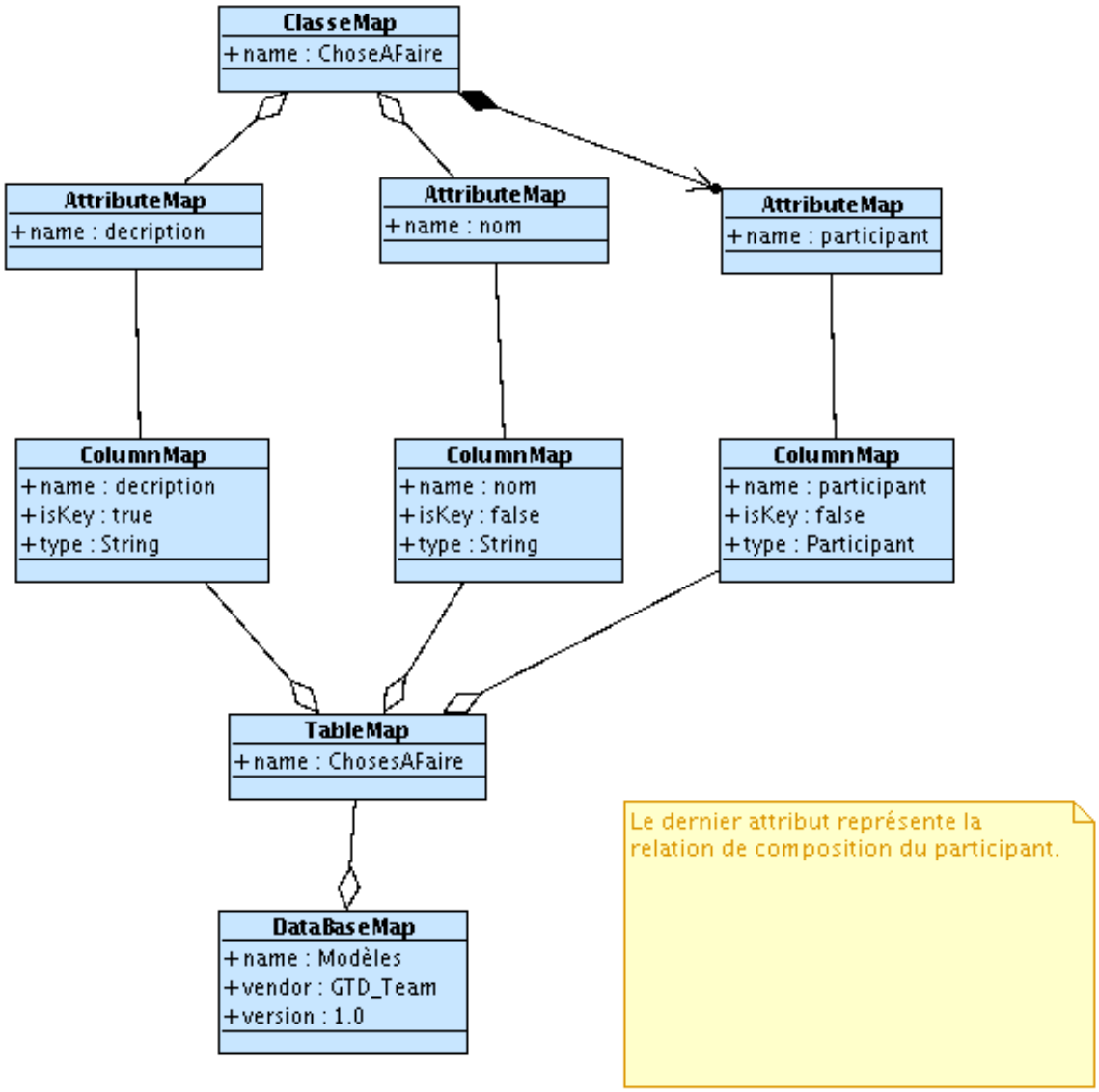


FIGURE 5.7 – Schéma de la base de données : Table Chose à faire

Puis on termine par les classes héritant d'attributs communs provenant de la classe "Élément" :

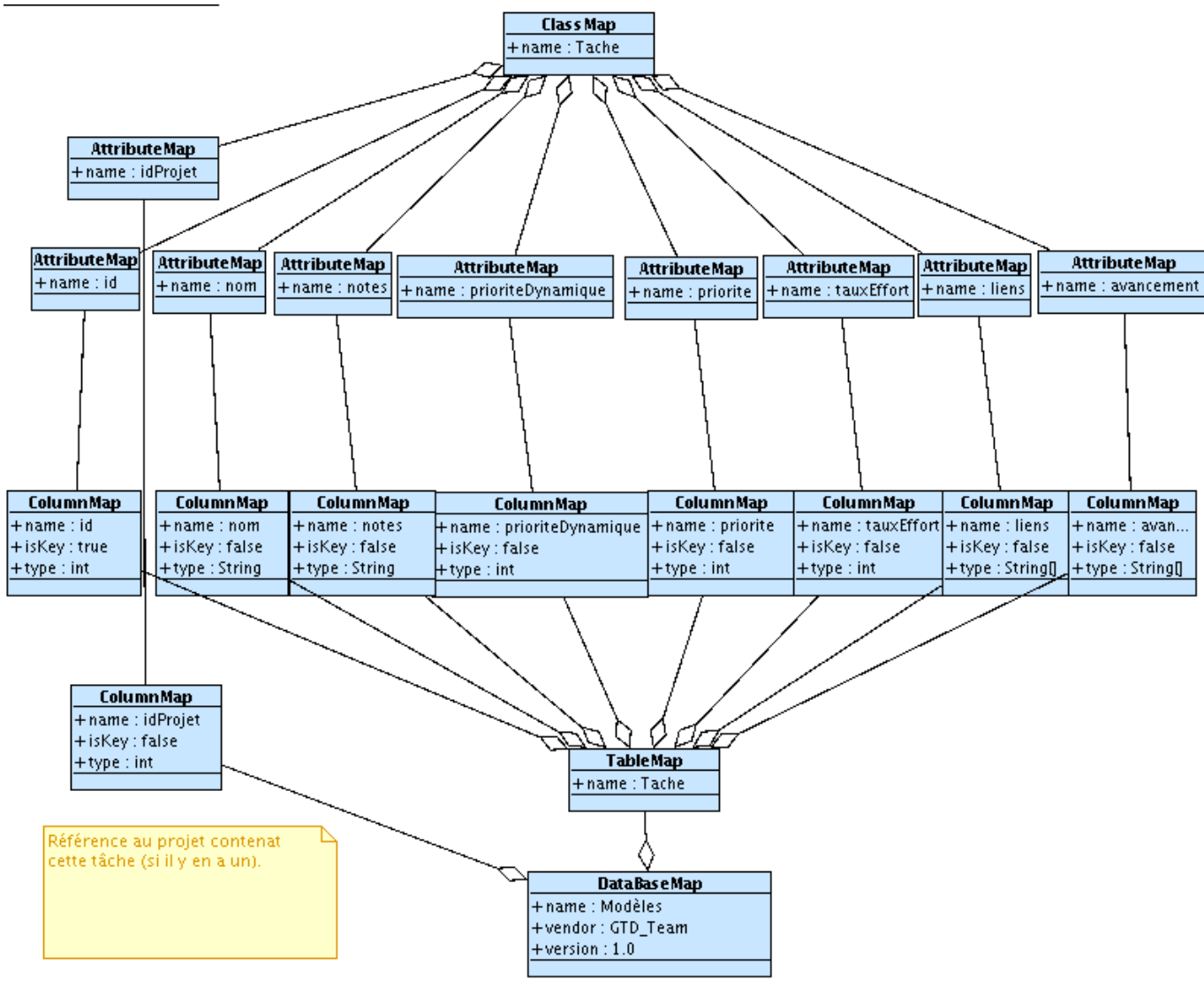


FIGURE 5.8 – Schéma de la base de données : Table Tâches

On retrouve donc dans la table "Tâche" 5.8 tous les attributs de la classe "Élément" et ceux de la classe "Tâche", plus un attribut référençant le projet contenant cette tâche (s'il y en a un).

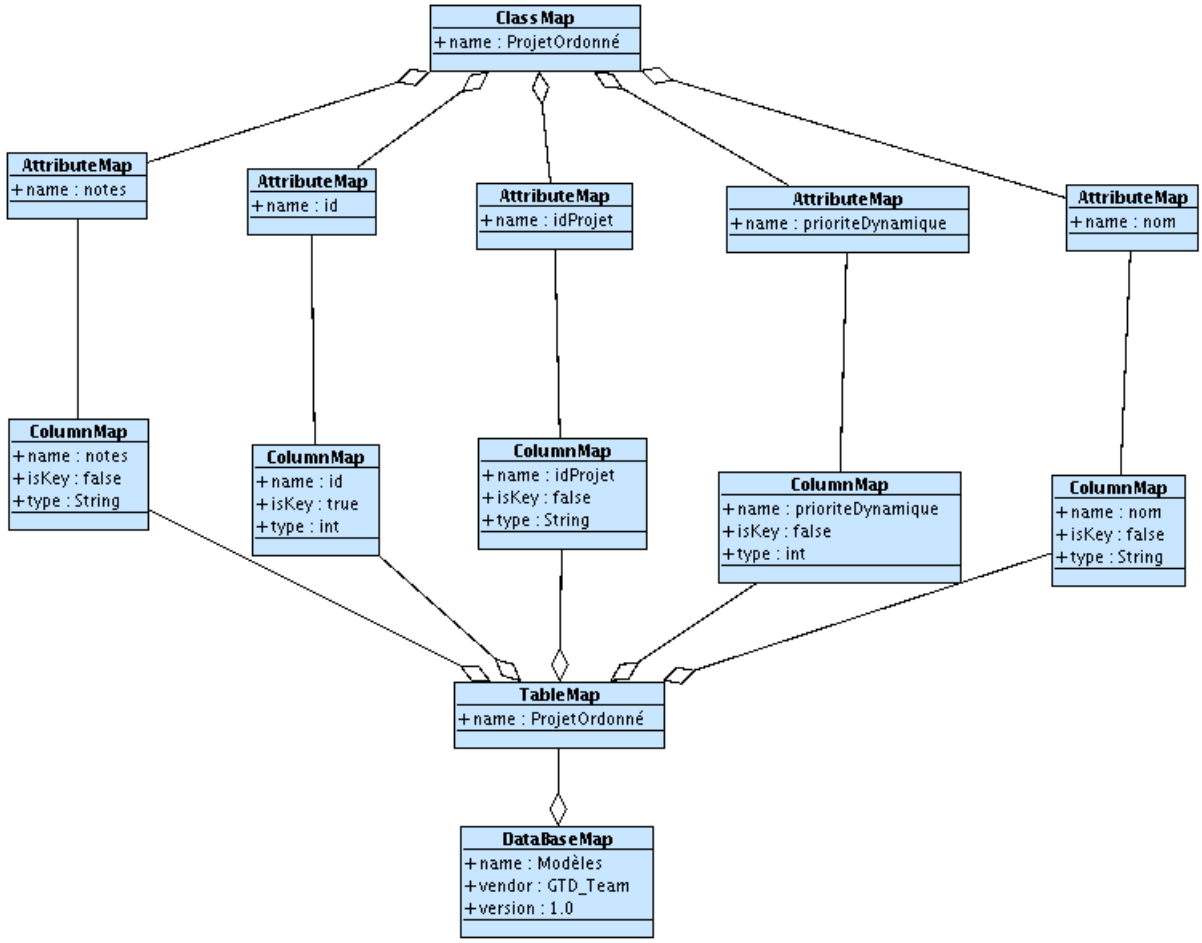


FIGURE 5.9 – Schéma de la base de données : Table Projets Non Ordonnés

De même pour un "Projet Non Ordonné" 5.9 qui peut être le sous-projet d'un autre projet.

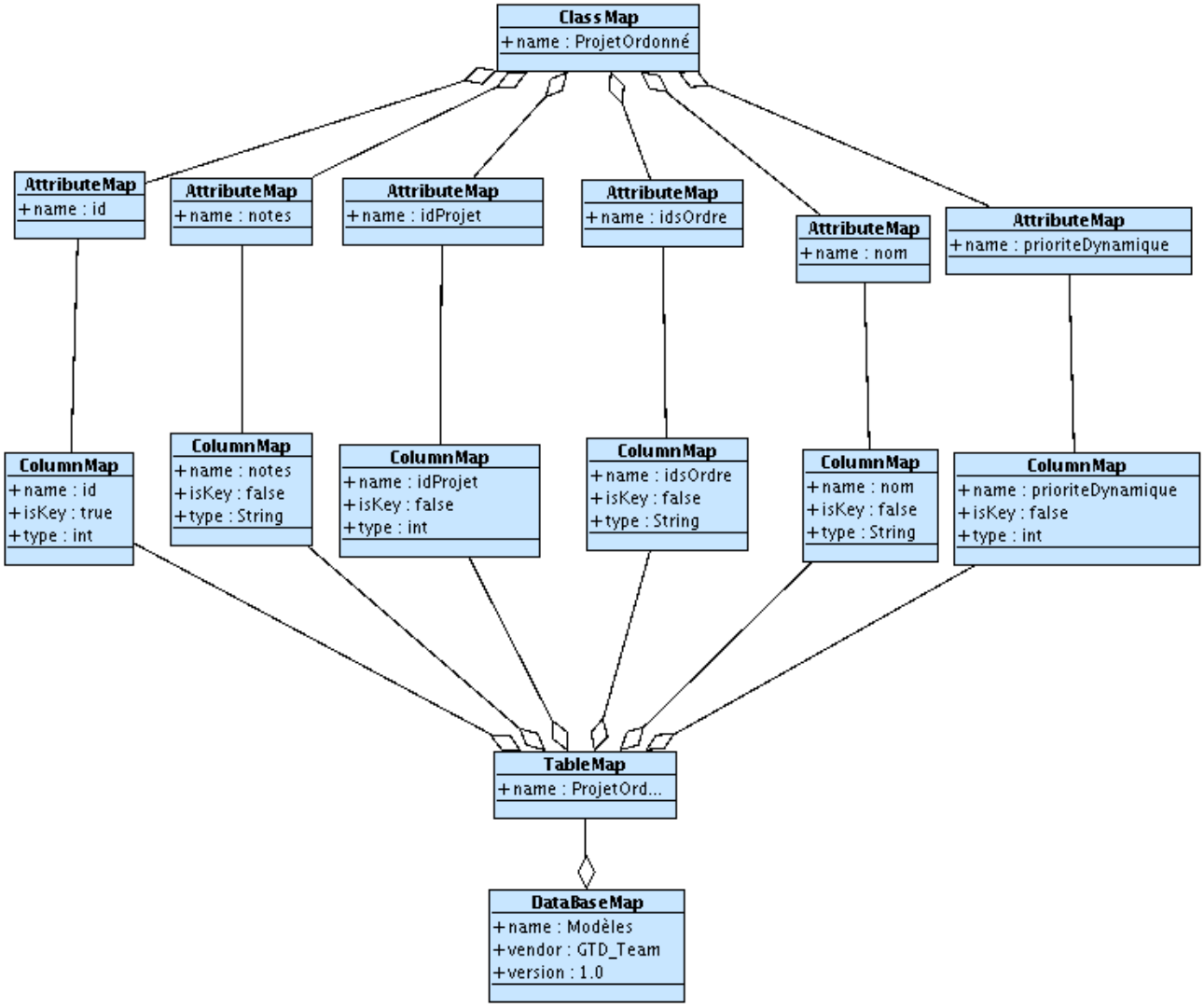


FIGURE 5.10 – Schéma de la base de données : Table Projets Ordonnés

Les "Projets Ordonnés" 5.10 peuvent aussi être des sous-projets mais doivent se "souvenir" de l'ordre des tâches qu'il contienne d'où la chaîne de caractères contenant les ids ordonnés des tâches d'un "Projet Ordonné".

5.3 Stéréotypes et étiquettes

Pour pouvoir implémenter proprement notre logiciel, nous avons utilisé différents patrons de conception pour créer son architecture.

5.3.1 Modèle-Vue-Contrôleur

Dans un premier temps, nous avons découper nos classes en 3 parties. Ce qui nous a permis d'y voir un peu plus clair dans ce que fait chaque composants logiciels. Dans notre diagramme de classes, la partie Vue n'apparaît pas car elle est plus extérieure ; mais elle est bien présente dans notre logiciel.

5.3.2 Projets : patron Composite

Selon notre diagramme de classes, un Projet est un Élément. Seulement nous avons défini depuis le début qu'un Projet peut contenir des sous-Projets ; c'est pourquoi nous avons utilisé ici le patron Composite.

5.3.3 Méthode GTD : patron Stratégie

La partie Organisation de la méthode GTD est une partie complexe et nous avons choisi d'utiliser le patron Stratégie pour pouvoir changer facilement son code. Ce qui nous donne la possibilité de changer cette méthode à tout moment pour l'améliorer ou changer complètement son code métier.

5.3.4 Comptes : patron Visiteur

On utilise le patron Visiteur pour pouvoir accéder facilement au différentes valeurs stockées dans le Modèle.

5.4 Règles de traduction

On utilise les règles suivantes de traduction de l'UML vers Java avec Acceleo :
- Chaque package dans le diagramme de classe se transforme en *Package Java* ;
- Chaque classe génère une *Class Java* ainsi que son constructeur ;
- Chaque interface génère des *Interfaces Java* ;
- Une propriété donne une méthode ;
- Un attribut est privé et on génère des *Getters/Setters* ;
- Pour les attributs avec une multiplicité supérieur, on place l'attribut Java dans une *LinkedList* et on ajoute des méthodes d'ajout et de suppression dans cette liste ;
- Si une classe hérite d'une autre, on rajoute le *extend* ;
- Une classe qui implémente une interface se voit ajouter *implements*et on génère les méthodes de l'interface que la classe implémente.

Ces quelques règles nous permettent de générer du code sur lequel nous pourrons "greffer" le code métier de notre application.

Conclusion

Puisque nous avons défini l'architecture globale de notre logiciel, nous pouvons dès à présent nous intéresser aux détails des composants et ainsi définir tous les détails précédant la partie de codage visant à remplir le code généré.

Chapitre 6

Conception détaillée

Introduction

Le dernier pas dans la conception de notre logiciel est de spécifier en détails tous les composants et leur rôles. Nous aurons de ce fait décrit tous les aspects de notre programme et pourrons le développer sans ambiguïtés.

6.1 Répertoire de décisions

Mais avant de rentrer dans les détails de la conception, nous devons préciser les modifications apporter au modèle durant le développement.
Pour rappel, le modèle après analyse était la figure 6.1.

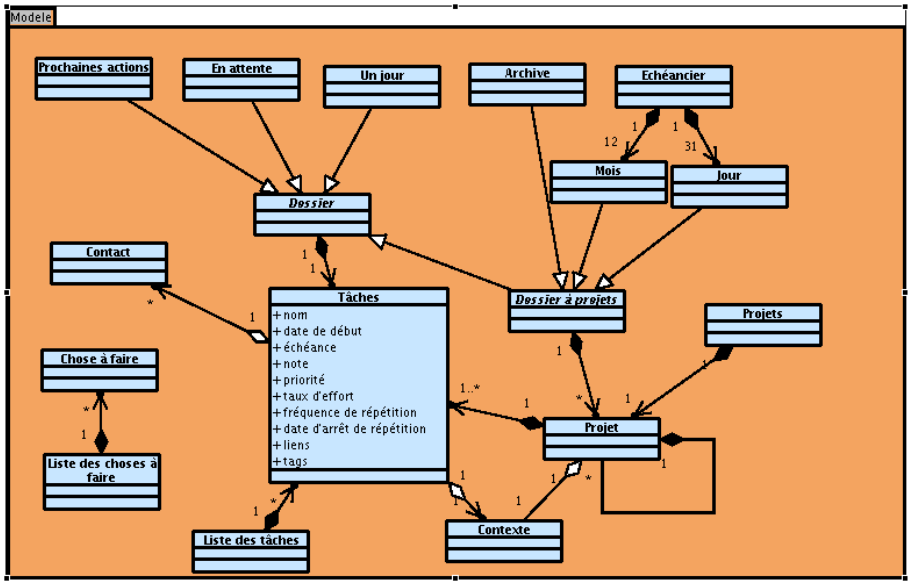


FIGURE 6.1 – Diagramme de classes du modèle après analyse

Et maintenant, le diagramme de classe est représenté par la figure 6.2.

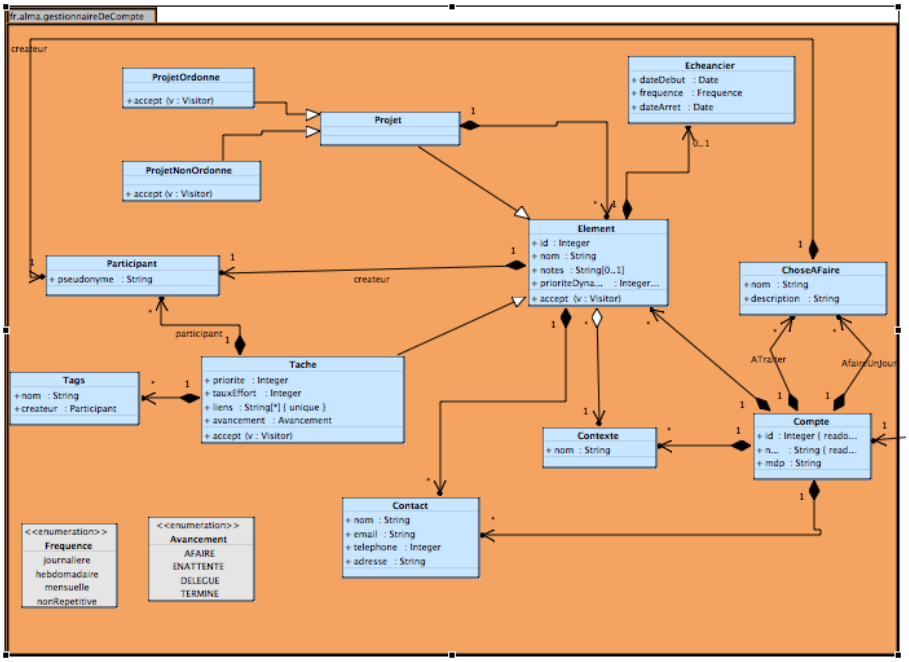


FIGURE 6.2 – Diagramme de classes du modèle à la fin du projet

Les modifications sont nombreuses :

- Suppression de la classe Mois (Les échéances sont stockées dans l'échéancier et les tâches ou projets qui l'utilisent pointent dessus) ;
- Suppression de la classe Jour (comme Mois) ;
- Suppression de Archive (les éléments contiennent un état pour renseigner si ils sont actifs ou non) ;
- Suppression des "dossiers" Dossier/Dossier à projets/Projets (la version informatique rend obsolète l'utilisation de dossiers) ;
- Suppression de Prochaines Actions (élément visuel, les tâches sont afficher en fonction des choix de l'utilisateur) ;
- Suppression de Liste des Tâches (affichage des tâches, pas besoin de les stocker ensemble) ;
- Modification de Liste des Choses à faire (attribut d'une autre classe) ;
- Modification de Contacts et Tags (ajout d'attributs) ;
- Modification de En Attente (contenu dans l'état de l'élément) ;
- Modification de Un Jour (attribut d'une autre classe) ;
- Ajout de Participants (nécessaire pour les éléments ordonnés et connaître les dépendances) ;
- Ajout de Projet Ordonné et de Projet Non Ordonné (sous classes de Projet) ;
- Ajout de Éléments (pour regrouper les attributs commun de Tâche et Projet, et implémenter le patron Composition)
- Ajout de Compte (contient toutes les autres classes).

Ainsi nous obtenons le diagramme de classe de la figure 6.2 et pouvons ainsi commencer la conception détaillée.

6.2 Diagrammes statiques et dynamiques

Le diagramme de classe du modèle n'est pas le seul à avoir changé, le reste du diagramme de classe a lui aussi eu des modifications(Voir figure 6.3).

On peut voir les différentes interactions entre les composants décrits dans le chapitre "Spécification des Composants".

6.3 Spécification des classes

Cette section comporte une liste de toutes les classes en spécifiant pour chacune d'elles ses invariants.

Package : Gestionnaire de Compte

Classe : Compte

- Nom jamais vide
- Mot de passe jamais vide

Classe : Chose A Faire

- Nom jamais vide
- description jamais vide

Classe : Contexte

- Nom jamais vide

Classe : Contact

- Le couple nom/adresse unique
- Nom jamais vide
- Adresse jamais vide

Classe : Élément

- Nom jamais vide

Classe : Participant

- Id unique
- Pseudonyme jamais vide

Classe : Échéancier

- Triplet d'attributs unique

Classe : Tag

- Nom non vide
- Nom unique

Conclusion

Toutes les étapes de l'analyse étant terminées, nous pouvons finalement commencé l'implémentation qui est facilitée grâce à l'analyse poussée que nous avons réalisé. La génération de code à partir de notre modèle nous fera gagner du temps et l'utilisation de frameworks, tels Hibernate, nous facilitera la tâche.

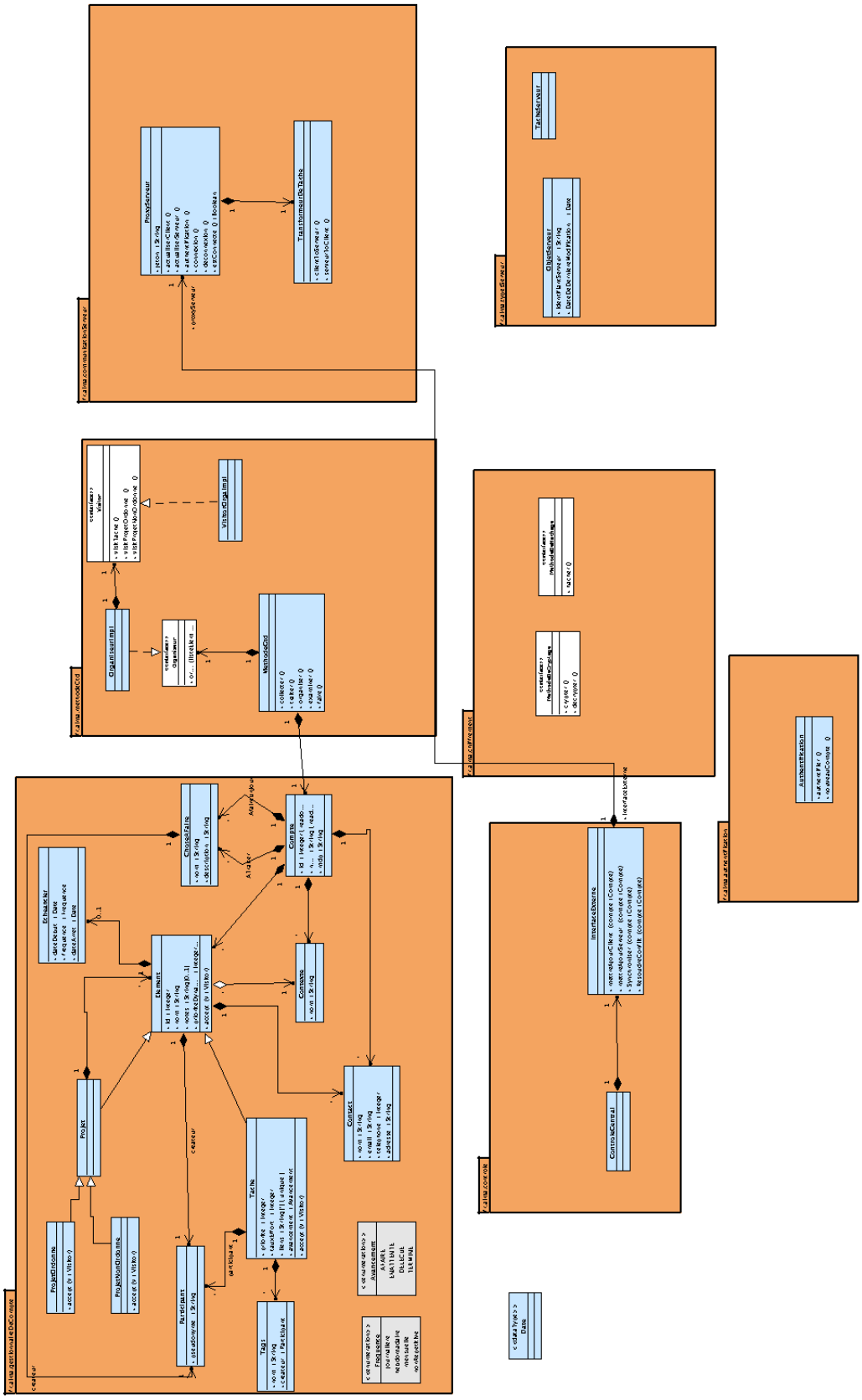


FIGURE 6.3 – Diagramme de classes final

Conclusion

Le développement de cette application nous a permis d'aborder de nouvelles méthodes telles que la génération de code, et d'approfondir des méthodes vu dans d'autres modules telles que la gestion de la persistance, les EJB.. Nous avons ainsi pu utiliser pleinement de nouveaux frameworks et de nouvelles méthodes de travail. Nous avons aussi pu mieux maîtriser la gestion et la répartition du travail au sein de notre groupe en même temps que de travailler sur un projet de gestion du travail à faire. L'interaction entre les modules s'est révélée aisé grâce aux professeurs qui nous ont guider. Ce projet nous a par ailleurs détailler niveau par niveau les différents pas pour donner vie à une idée (passage de l'idée jusqu'au produit final).

Bibliographie

[1] Wikipedia, Getting Things Done :
http://en.wikipedia.org/wiki/Getting_Things_Done

[2] Getting Things GNOME ! :
http://gtg.fritalk.com/