



# Getting Things Done

Objets distribués

Le projet GTD d'un point de vue objets distribués

**BRUN Clémentine, LE NY Brendan, PONGE Myrtille,  
RAVENET Brian**

**02/02/2010**

## TABLE DES MATIERES

---

**Aucune entrée de table des matières n'a été trouvée.**

## INTRODUCTION

---

Au cours de la deuxième année du master ALMA, nous devons réaliser un projet concernant plusieurs modules distincts. Le travail à réaliser consiste en la construction d'une application outillant la méthode GTD<sup>1</sup>

---

<sup>1</sup> GTD = Getting Things Done, méthode de gestion de tâches créées par David Allen.



# Getting Things Done

## Livrable 2 : Spécifications des besoins

Documents de spécifications des besoins du client de l'application outillant la méthode GTD.

**BRUN Clémentine, LE NY Brendan, PONGE Myrtille,  
RAVENET Brian**

**08/10/2009**

## TABLE DES MATIÈRES

---

---

Introduction .....	1
I. Description générale.....	2
I.1. Perspectives et fonctions du produit.....	2
I.2. Environnement opérationnel et implémentation .....	3
I.3. Les utilisateurs.....	3
I.4. Exigences reportées.....	4
II. Fonctionnalités du logiciel.....	5
II.1. Cas d'utilisation <i>Collecter</i> .....	5
II.2. Cas d'utilisation <i>Vider le panier</i> .....	6
II.3. Cas d'utilisation <i>Agir</i> .....	8
II.4. Cas d'utilisation <i>Réviser</i> .....	12
II.5. Cas d'utilisation <i>Import/Export des données</i> .....	18
III. Exigences des interfaces externes .....	21
III.1. Interface utilisateur .....	21
III.2. Interface logicielle .....	21
III.3. Interface ou protocole de communication .....	21
III.4. Contraintes de mémoire .....	21
IV. Autres exigences non-fonctionnelles .....	23
IV.1. Utilisabilité .....	23
IV.2. Fiabilité .....	23
IV.3. Exigences de sécurité .....	23
IV.4. Exigences de performance .....	24
IV.5. Gestion de la persistance .....	25

## INTRODUCTION

---

L'objectif de ce document est d'énoncer, dans un langage compréhensible par tous les intervenants du projet, les exigences de l'application. Il peut s'agir d'exigences fonctionnelles, c'est à dire d'exigences qui doivent être assurées afin que l'application puisse être utilisée afin d'accomplir le but pour lequel elle a été développée. Il peut aussi s'agir d'exigences non-fonctionnelles, c'est à dire des caractéristiques qu'on attend de l'application afin de pouvoir l'utiliser simplement.

Cette spécification est écrite à destination aussi bien des développeurs que du client qui a commandé l'application. Il se doit donc d'être compréhensible par l'ensemble des acteurs intervenants autour du projet. Nous utiliserons un langage commun et des représentations accessibles par chacun. Ce document est essentiel dans la formalisation des demandes du client en vue d'une validation.

Nous ne décrivons dans ce document que ce qui concerne le sous-système dont nous avons la responsabilité : un client ainsi qu'une IHM. Nous ne présenterons donc pas la spécification des besoins du serveur.

Au cours de cette spécification des besoins, vous pourrez rencontrer l'acronyme GTD. Il désigne la méthode d'organisation personnelle « Getting Things Done™ » telle que décrite dans le livre de David Allen.

Nous présenterons tout d'abord une description générale du produit spécifié : sa place dans le système global et tous les éléments nécessaires pour détourner le périmètre de l'application. Une fois ce cadre définit, nous examinerons les besoins fonctionnels que le produit doit satisfaire selon les différents aspects de son utilisation. Enfin nous verrons les besoins qui devront être satisfaits pour garantir l'intégration de l'application et les divers besoins qui ne sont pas directement fonctionnels.

## I. DESCRIPTION GENERALE

L'application dont nous spécifions les besoins comprend de nombreuses fonctionnalités. Nous allons ici présenter brièvement le périmètre de l'application, ainsi que son environnement global.

### I.1. PERSPECTIVES ET FONCTIONS DU PRODUIT

Le produit spécifié dans ce document est un des éléments d'un système plus global. Il s'agit d'un client capable de se connecter à un serveur via CORBA. Sur le diagramme suivant, le produit est représenté en bleu.

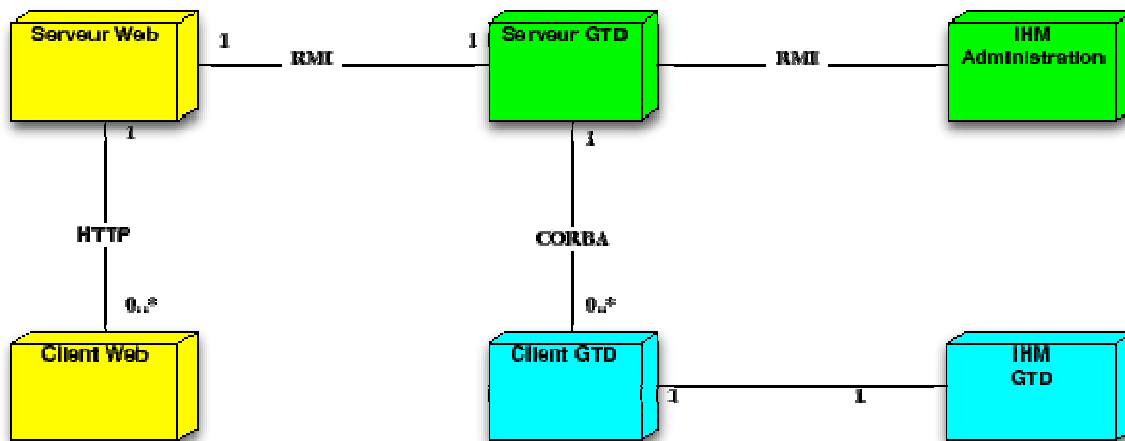


FIGURE 1 : REPRESENTATION GLOBALE DU SYSTEME

Les fonctionnalités principales proposées par le logiciel sont les suivantes :

- ✓ L'utilisateur pourra saisir, à la volée, des éléments dans un panier qu'il pourra prendre en compte plus tard pour créer entre des données. Plus loin on parle de « collecter »
- ✓ Gestion de projets et des tâches qui les composent.
- ✓ À tout moment, le produit doit être en mesure de permettre à l'utilisateur de savoir sur quelle tâche il peut agir. Lorsque l'utilisateur cherche à obtenir cette information, on parle de « agir »
- ✓ L'utilisateur peut passer en revue toutes ses tâches et éventuellement les modifier et vider le panier : l'utilisateur veut transformer ces notes du panier en tâches, projet, contexte ou contact. Dans ce cas, on parle de « réviser ».

## I.2. ENVIRONNEMENT OPERATIONNEL ET IMPLEMENTATION

---

Notre application cliente requiert une machine de type ordinateur de bureau ou ordinateur portable (pas de support du téléphone mobile, support limité des ultra-portables). Le système d'exploitation peut-être quelconque tant qu'une machine virtuelle Java est supportée. La configuration matérielle requise est celle nécessaire au fonctionnement d'une machine virtuelle Java.

Les données que l'utilisateur a entrées au fur et à mesure de l'utilisation régulière du logiciel sont stockées de façon pérenne. Étant donné qu'il doit être possible d'accéder à ces données depuis des clients déployés sur différentes machines (pas en même temps), les données devront être centralisées sur un serveur et c'est à chaque client de se connecter au serveur pour vérifier si aucune modification n'a été faite par ailleurs, via un autre client. Ceci impose la contrainte de centraliser les données sur un serveur qui doit être accessible depuis les machines clientes via les infrastructures réseaux habituelles. Cependant, le fonctionnement hors-ligne de l'application cliente est assuré. L'application devra, dès que l'occasion se présente, se synchroniser avec le serveur afin de présenter, autant que possible les données les plus à jour.

## I.3. LES UTILISATEURS

---

L'application cliente pourra être utilisée par des personnes qui maîtrisent les fondements de l'informatique bureautique. Un utilisateur ayant acquis la maîtrise d'applications bureautiques typiques devrait pouvoir prendre en main le logiciel en quelques minutes. L'application doit pouvoir être prise en main et utilisée quotidiennement, y compris par des personnes ignorant les principes de la méthode de GTD.

L'application outillant la méthode GTD doit être simple d'utilisation afin d'être accessible au plus large public possible. Afin de guider l'utilisateur dans le fonctionnement du logiciel, un guide d'utilisation sera fourni. Il reprendra les principales fonctionnalités du logiciel en détaillant comment y accéder et comment les mettre en œuvre. Ce guide sera accessible sous forme d'aide en ligne et sous forme de PDF fourni avec l'application.

L'objectif de l'application est de faire gagner du temps à l'utilisateur, il est donc fort probable qu'il ne désire pas lire la documentation avant de prendre en main le logiciel. Différentes aides devront donc être disponibles tout au long de l'utilisation de l'application. Nous utiliserons pour cela des info-bulles. Lors du survol d'un bouton ou d'un champ texte par la souris, une bulle apparaîtra indiquant précisément comment doit être utilisé l'élément survolé et son utilité. Pour



ne pas complexifier les actions de l'utilisateur, cette info-bulle disparaîtra automatiquement dès que la souris aura quitté l'élément concerné.

#### I.4. EXIGENCES REPORTEES

---

La plupart des problématiques d'adaptabilité de l'application cliente ne peuvent être envisagées dans cette version du logiciel. Toutefois, il peut être envisagé qu'elles soient développées dans une hypothétique version à venir. Dans cette perspective, le développement actuel du client n'intégrera aucune des fonctionnalités décrites ci-après mais préservera la possibilité d'intégrer ces fonctionnalités a posteriori avec facilité. En particulier, cela influera l'architecture globale de l'application qui devra proposer les points d'extensions.

L'application cliente pourra s'interfacer avec les différentes applications que l'utilisateur utilise régulièrement dans le cadre de son travail. Notamment, il peut s'agir d'un client de courrier électronique, un logiciel de calendrier, un carnet d'adresses (type LDAP), ERP ou CRM. L'application devrait permettre de rendre possible de développer des plugins permettant à l'application cliente de s'interfacer avec les outils métiers utilisés dans la structure.

L'utilisateur pourra adapter l'application à sa guise en agencant les différents éléments de l'interface graphique.

Étant donné que le client s'appuie sur la plate-forme Java pour fonctionner, il peut être fonctionnel sur des plateformes matérielles qui ne sont pas supportées lors du développement. Par exemple, c'est le cas des assistants personnels ou des téléphones portables. Cette compatibilité est possible mais pas assurée : dans une prochaine version, elle pourra être assurée.

L'application cliente sera développée en français et devrait fonctionner quelque-soit l'agencement clavier utilisé. Plus tard, l'application devrait permettre de traduire l'interface dans différentes langues.

## II. FONCTIONNALITES DU LOGICIEL

Lors de la commande d'une application, le client exprime ce pourquoi le logiciel devra être fait et surtout ce qu'il devra faire. Ces requêtes représentent les besoins fonctionnels. Ils permettent ainsi de délimiter les frontières du système. Formaliser ces besoins fonctionnels permet d'avoir un document référentiel pour l'équipe réalisant l'application et pour le client. Les développeurs connaissent donc précisément l'ensemble des fonctionnalités du logiciel, tandis que le client à l'assurance d'avoir bien fait comprendre ses besoins.

Le but de l'application présentée ici est d'outiller la méthode GTD. Les besoins fonctionnels sont donc au minimum les cas d'utilisation présentés au cours du premier livrable.

### II.1. CAS D'UTILISATION *COLLECTER*

L'utilisateur désirera régulièrement sauvegarder des éléments dans son panier. Le système développé doit donc être capable de proposer une interface adapté à la collecte à l'utilisateur, puis d'enregistrer les informations renseignées. Les éléments collectés doivent être consultables à tout moment, il est donc nécessaire de pouvoir gérer la persistance des données. Ce dernier point sera développé dans le chapitre IV.5.

Use Case	Collecter
Goal in context	L'utilisateur vient de recevoir une information ou pense à une idée. Il souhaite enregistrer cette nouvelle donnée.
Scope	La gestion du panier.
Level	Fonction primaire de l'utilisateur.
Pré-conditions	<ul style="list-style-type: none"><li>✓ L'utilisateur possède un compte sur l'application.</li><li>✓ L'utilisateur est identifié sur l'application.</li></ul>
Success end condition	👉 Une instance de l'élément est sauvegardée, sur le serveur ou en local
Failed end condition	👉 L'élément ne possède aucun instance correspondante ni sur le serveur, ni en local.
Primary actor	L'utilisateur.

Trigger	L'utilisateur envoi une demande d'ajout d'élément dans le panier au client GTD.
Main success scenario	<ol style="list-style-type: none"> <li>1. L'utilisateur remplit tous les champs du formulaire d'ajout d'élément dans le panier du client GTD.</li> <li>2. L'utilisateur demande au client GTD d'enregistrer ces informations.</li> <li>3. Le client GTD envoi une requête d'ajout de nouvel élément au serveur.</li> <li>4. Le serveur confirme la sauvegarde du nouvel élément dans la base de données.</li> <li>5. Le client confirme à l'utilisateur la sauvegarde du nouvel élément.</li> </ol>
Variations	<p>Le client n'est pas connecté au serveur, la sauvegarde du nouvel élément dans le panier se fera donc en mode local.</p> <ol style="list-style-type: none"> <li>1. L'utilisateur remplit tous les champs du formulaire d'ajout d'élément dans le panier du client GTD.</li> <li>2. L'utilisateur demande au client GTD d'enregistrer ces informations.</li> <li>3. Le client GTD enregistre les informations localement.</li> <li>4. Mise à jour de la date de modification de la base de données locale.</li> <li>5. Le client confirme à l'utilisateur la sauvegarde du nouvel élément.</li> </ol>
Frequency	Plusieurs fois par jour.

## II.2. CAS D'UTILISATION *VIDER LE PANIER*

Le logiciel doit permettre à l'utilisateur de vider le panier. C'est l'occasion pour l'utilisateur de transformer ses notes en projet, tâche, contexte ou d'effectuer les modifications à faire dans les données existantes ou de supprimer la note. A la fin de la purge du panier, le panier doit être vide. C'est l'utilisateur de prendre l'initiative de vider le panier, le logiciel ne fera pas de rappel du type "attention vous n'avez pas purgé votre panier depuis une semaine".

Use case	Vider le panier
Goal in context	L'utilisateur passe en revue les éléments du panier

Scope	Gestion du panier
Level	Primary task
Preconditions	<ul style="list-style-type: none"> <li>✓ L'utilisateur possède un compte sur l'application.</li> <li>✓ L'utilisateur est identifié sur l'application.</li> </ul>
Success end condition	👉 Le panier ne contient aucun élément.
Failed end condition	👉 Il reste un ou plusieurs éléments dans le panier
Primaty actor	L'utilisateur
Trigger	Volonté de l'utilisateur
Main success scenario	<ol style="list-style-type: none"> <li>1. L'utilisateur demande au client de visualiser tous les éléments de son panier.</li> <li>2. A chaque élément visualisé, l'utilisateur choisi d'appliquer le comportement adapté : créer une tâche, un projet,... En remplissant les formulaires adaptés proposé par l'interface utilisateur du client.</li> <li>3. L'utilisateur demande au client de supprimer l'élément.</li> <li>4. Le client demande au serveur de supprimer l'élément.</li> <li>5. Le serveur confirme la suppression de l'élément.</li> <li>6. Le client confirme à l'utilisateur la suppression de l'élément.</li> </ol>
Extensions	<ul style="list-style-type: none"> <li>- si une tâche ne suffit pas : l'utilisateur peut créer un nouveau projet et créer les différentes tâches.</li> <li>- si besoin, l'utilisateur peut recourir à la création d'un nouveau contexte.</li> <li>- si l'élément est sans intérêt : l'utilisateur peut choisir de ne créer aucune tâche et de simplement supprimer l'élément.</li> <li>- si la nouvelle tâche peut être accomplie en moins de 2 minutes dans le contexte actuel, l'utilisateur ne créer pas de nouvelle tâche et accomplit le travail immédiatement.</li> <li>- si besoin, l'utilisateur peut recourir à la création d'un ou plusieurs contacts</li> </ul>

Variations	Le client n'est pas connecté au serveur distant. Il sauvegarde donc les modifications dans sa base de données locale et met à jour la date de dernière modification.
Frequency	Selon la volonté de l'utilisateur, plutôt une fois par jour

### II.3. CAS D'UTILISATION AGIR

L'utilisateur dispose de temps et veut agir sur un projet, sur une tâche ou dans un contexte particulier. Il recherche donc des actions à réaliser en fonctions de ces contraintes données. La méthode GTD propose plusieurs types de recherches différentes que nous allons définir.

#### II.3.A. CAS D'UTILISATION AGIR SUR UN PROJET

Une des spécificités de la méthode GTD est de proposer des visions différentes des tâches de l'utilisateur. L'une de ces visions est celle par projet. Nous pouvons reprendre l'exemple de l'utilisateur souhaitant réaliser le projet *construire sa maison*. Aujourd'hui il a du temps et voudrait faire avancer ce projet. Il va donc s'appuyer sur la méthode GTD pour trouver la prochaine action à réaliser au sein du projet concerné. Nous allons détailler ici les mécanismes permettant à l'utilisateur de faire avancer son projet grâce à la méthode GTD.

Use Case	Agir dans un projet.
Goal in context	L'utilisateur cherche une tâche à faire dans un projet particulier.
Scope	La gestion de projets.
Level	Fonction primaire de l'utilisateur.
Pré conditions	Le projet sur lequel l'utilisateur veut agir existe.
Success end condition	<ul style="list-style-type: none"> <li>👍 L'utilisateur a trouvé une tâche à réaliser.</li> <li>👍 Cette tâche est la plus urgente du projet indiqué.</li> <li>👍 La tâche trouvée est réalisable en fonction du temps et de l'énergie dont l'utilisateur dispose.</li> </ul>

Failed end condition	<ul style="list-style-type: none"> <li>☞ L'utilisateur n'a trouvé aucune tâche.</li> <li>☞ La tâche trouvée n'est pas la plus urgente du projet.</li> <li>☞ La tâche trouvée est irréalisable en fonction du temps et de l'énergie dont l'utilisateur dispose.</li> </ul>
Primary actor	L'utilisateur
Trigger	L'utilisateur demande une tâche à réaliser dans un projet.
Main success scenario	<ol style="list-style-type: none"> <li>1. L'utilisateur demande au client de rechercher une tâche à réaliser en fonction de données qu'il renseigne sur un formulaire. Ces données comprennent le taux d'effort et le temps dont il dispose.</li> <li>2. Le client demande au serveur l'ensemble des tâches du projet sélectionné par l'utilisateur.</li> <li>3. Le client parcourt les tâches en recherchant celle correspondant le plus possible aux demandes de l'utilisateur.</li> <li>4. L'utilisateur affiche à l'utilisateur la tâche sélectionné.</li> <li>5. L'utilisateur traite la tâche.</li> <li>6. L'utilisateur demande au client la suppression de la tâche.</li> <li>7. Le client transmet la demande de suppression au serveur.</li> <li>8. Le serveur confirme la suppression de la tâche.</li> <li>9. Le client confirme la suppression de la tâche à l'utilisateur.</li> </ol>
Extensions	Le projet ne contient aucune tâche à réaliser.
Variations	<ul style="list-style-type: none"> <li>- Le client n'est pas connecté au serveur, il travail donc avec sa base de données locale.</li> <li>- Après le traitement de la tâche, l'utilisateur ne souhaite pas la supprimer. Il met donc son état à jour, et demande sa sauvegarde.</li> </ul>

---

### II.3.B. CAS D'UTILISATION AGIR DANS UN CONTEXTE

---

La méthode GTD permet également d'avoir une vision par contexte des différentes tâches à réaliser. Nous pouvons prendre l'exemple suivant : pour le projet "organiser mon voyage en Antarctique" j'ai besoin du contexte "avoir internet". L'utilisateur est connecté à internet, il va demander à la méthode GTD quelle est la prochaine tâche à réaliser dans ce contexte. Nous

allons détailler le procédé permettant à l'utilisateur de travailler dans un contexte précis à l'aide de la méthode GTD.

Use Case	Agir dans un contexte
Goal in context	L'utilisateur cherche une tâche à faire dans un contexte particulier.
Scope	La gestion de contexte/projet
Level	Fonction primaire de l'utilisateur.
Pré conditions	Le contexte demander par l'utilisateur existe. Le contexte dans lequel l'utilisateur veut travailler contient au moins un projet ou une tâche.
Success end condition	<ul style="list-style-type: none"> <li>👉 L'utilisateur a trouvé une tâche à réaliser dans ce contexte.</li> <li>👉 Cette tâche est la plus urgente du contexte indiqué.</li> <li>👉 La tâche trouvée est réalisable en fonction du temps, de l'énergie dont l'utilisateur dispose.</li> </ul>
Failed end condition	<ul style="list-style-type: none"> <li>👉 L'utilisateur n'a trouvé aucune tâche dans le contexte.</li> <li>👉 La tâche trouvée n'est pas la plus urgente dans le contexte.</li> <li>👉 La tâche trouvée est irréalisable en fonction du temps et de l'énergie dont l'utilisateur dispose.</li> </ul>
Primary actor	L'utilisateur
Trigger	L'utilisateur demande une tâche à réaliser dans un contexte.
Main success scenario	<ol style="list-style-type: none"> <li>1. L'utilisateur recherche une tâche à faire dans un contexte.</li> <li>2. Toutes les tâches du contexte voulu sont parcourues, la tâche sélectionnée est celle dont la priorité est la plus haute et dont l'énergie et le temps nécessaire correspond à ce dont dispose l'utilisateur.</li> <li>3. L'utilisateur traite la tâche</li> <li>4. L'utilisateur supprime la tâche si elle est terminée</li> </ol>
Extensions	1 - Le contexte n'existe pas.

	2 - Aucune tâche n'est associée au contexte.
Variations	

### II.3.C. CAS D'UTILISATION *AGIR SUR UNE TACHE*

Le logiciel permet également au logiciel d'accéder à toutes les tâches enregistrées. L'utilisateur peut accéder à la tâche la plus urgente à faire et se mettre dans le bon contexte pour la réaliser.

Use Case	Agir sur une tâche.
Goal in context	L'utilisateur cherche une tâche à réaliser.
Scope	Gestion des tâches
Level	Fonction primaire de l'utilisateur.
Préconditions	L'utilisateur demande une tâche à réaliser.
Success end condition	<ul style="list-style-type: none"> <li>👍 L'utilisateur a trouvé une tâche à réaliser.</li> <li>👍 Cette tâche est la plus urgente.</li> <li>👍 La tâche trouvée est réalisable en fonction du temps, de l'énergie dont l'utilisateur dispose.</li> </ul>
Failed end condition	<ul style="list-style-type: none"> <li>👎 L'utilisateur n'a trouvé aucune tâche.</li> <li>👎 La tâche trouvée n'est pas la plus urgente.</li> <li>👎 La tâche trouvée est irréalisable en fonction du temps et de l'énergie dont l'utilisateur dispose.</li> </ul>
Primary actor	L'utilisateur
Trigger	L'utilisateur demande une tâche à réaliser.



Main success scenario	1. L'utilisateur recherche une tâche à faire. 2. Toutes les tâches du contexte voulu sont parcourues, la tâche sélectionnée est celle dont la priorité est la plus haute et dont l'énergie et le temps nécessaire correspond à ce dont dispose l'utilisateur. 3. L'utilisateur traite la tâche. 4. L'utilisateur supprime la tâche si elle est terminée.
Extensions	1. Aucune tâche n'est disponible.
Variations	

#### II.4. CAS D'UTILISATION *REVISER*

Le cas d'utilisation Réviser se divise en plusieurs sous cas d'utilisation qui sont, le nettoyage des projets, le nettoyage des contextes, la suppression d'un projet ou d'une tâche et la transformation d'une tâche en projet.

L'accès à chacune des ces fonctionnalités se fera à travers la *Révision*. C'est à dire que l'utilisateur doit demander au système d'accéder aux fonctionnalités de révisions, puis l'utilisateur doit pouvoir choisir quelle opération il souhaite réaliser.

##### II.4.A. CAS D'UTILISATION *NETTOYER DES PROJETS*

Le nettoyage des projets est une fonctionnalité importante du système. Il faut la considérer car c'est un des besoins primaires de l'application d'assurer que les éléments sauvegardés dans le système soient cohérents avec la réalité. Pour cela il est nécessaire de solliciter la participation de l'utilisateur afin de vérifier la validité des éléments. L'utilisateur devra donc être prévenu lorsque le dernier nettoyage remonte à trop longtemps. Nous allons vous présenter le cas général ainsi que le sous-cas qui est "Nettoyer un projet". Les autres sous-cas qui sont "Nettoyer les contextes" et "Supprimer un projet" ne nécessitent pas autant d'attention pour leur compréhension.

Use Case	Nettoyage des projets
Goal in context	Présenter chaque projet à l'utilisateur afin que celui-ci les révise si

	nécessaire
Scope	La gestion des projets.
Level	
Préconditions	L'utilisateur possède au moins un projet.
Success end condition	👉 Tous les projets ont été présentés à l'utilisateur.
Failed end condition	👉 Il reste des projets qui n'ont pas été présentés à l'utilisateur.
Primary actor	L'utilisateur
Trigger	L'utilisateur lance le nettoyage des projets.
Main success scenario	1. Le système présente chaque projet à l'utilisateur. 2. Pour chaque projet, l'utilisateur peut décider de nettoyer le projet
Extensions	Le projet peut être juste modifié. Le projet peut être supprimé
Variations	
Priorité	Haute
Fréquence	une à deux fois par semaine

Le système présentera à l'utilisateur tous les projets de son espace de travail, en lui proposant de supprimer, nettoyer ou de laisser tel quel chaque projet.

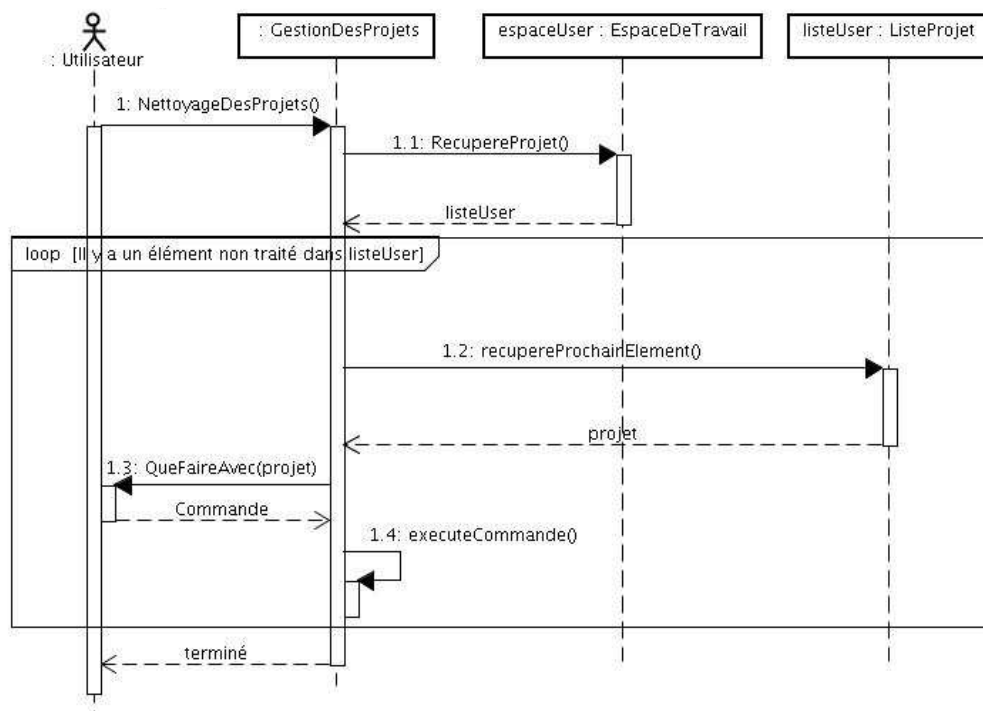


FIGURE 2 : SCENARIO NETTOYER LES PROJETS

Ce schéma nous montre que lorsque l'utilisateur demande à Nettoyer les projets, le système récupère la collection de projets associée à l'utilisateur (listeUser). Ensuite, pour chaque projet de cette liste, le système demande à l'utilisateur quelle commande il souhaite réaliser sur le projet (Modification, Suppression...). En ce qui concerne le développement, ce diagramme de séquence nous montre qu'il serait possible d'appliquer le Design Pattern Commande pour réaliser cette partie de l'application.

#### II.4.A.1. CAS D'UTILISATION NETTOYER UN PROJET

Il est important de bien expliciter ce qui attendu lors du nettoyage d'un projet, c'est pourquoi nous vous détaillons ce sous-cas d'utilisation de "Nettoyage des projets". Il s'agit de vérifier chacun de ses éléments et de demander à l'utilisateur s'il souhaite conserver cet élément, le modifier ou le supprimer. Si l'un de ses éléments est un sous-projet que l'utilisateur souhaite conserver, alors il doit être également nettoyé.

Use Case	Nettoyer un projet
Goal in context	Nettoyer tout ce qui est inutile au sein d'un projet.
Scope	La gestion des projets.
Level	

Préconditions	Le projet à nettoyer appartient à l'utilisateur.
Success end condition	👍 Tous les éléments au sein du projet sont utiles.
Failed end condition	👎 Il reste des éléments inutiles ou obsolètes au sein du projet.
Primary actor	L'utilisateur
Trigger	L'utilisateur lance le nettoyage du projet.
Main success scenario	<ol style="list-style-type: none"> <li>1. L'utilisateur accède au projet.</li> <li>2. Le système présente à l'utilisateur chaque élément du projet.</li> <li>3. L'utilisateur supprime l'élément s'il est inutile.</li> </ol>
Extensions	<ul style="list-style-type: none"> <li>- Si les éléments du projet sont des sous-projets, l'utilisateur doit pouvoir choisir de nettoyer les sous-projets.</li> <li>- L'élément peut également être juste modifié au lieu d'être supprimé</li> </ul>
Variations	Le projet est vide, il n'y a donc pas d'élément à observer. Le projet est donc déjà nettoyé.
Priorité	Haute
Fréquence	une à deux fois par semaine

#### II.4.A.II. CAS D'UTILISATION *TRANSFORMER UNE TACHE EN PROJET*

Nous ne voulions pas nous attarder sur la mise à jour d'une tâche ou d'un projet qui consiste à modifier quelques informations, mais sur la transformation d'une tâche en projet qui est une fonctionnalité beaucoup plus intéressante. Ce cas d'utilisation est une extension de la mise à jour d'une tâche, et peut donc intervenir dans le cadre de la Révision ou lors d'un appel isolé.

Use Case	Transformer une tâche en un projet.
Goal in context	La transformation d'une tâche devenue trop complexe en un nouveau projet.

Scope	Gestion des tâches
Level	
Préconditions	✓ L'utilisateur possède au moins une tâche
Success end condition	<ul style="list-style-type: none"> <li>👉 Un nouveau projet est créé;</li> <li>👉 Le nouveau projet contient l'ensemble des éléments appartenant précédemment à la tâche à transformer;</li> <li>👉 Le nouveau projet doit être un sous-projet de celui qui contenait la tâche;</li> <li>👉 Le nouveau projet contient la tâche à transformer.</li> </ul>
Failed end condition	<ul style="list-style-type: none"> <li>👉 Aucun projet n'est créé.</li> <li>👉 Le nouveau projet ne contient pas les éléments appartenant précédemment à la tâche.</li> <li>👉 Le nouveau projet n'est pas un sous-projet de celui qui contenait la tâche</li> </ul>
Primary actor	L'utilisateur
Trigger	L'utilisateur appelle la fonction de transformation d'une tâche en projet offerte par le système.
Main success scenario	<ol style="list-style-type: none"> <li>1. L'utilisateur demande la transformation d'une tâche en projet au système.</li> <li>2. Un nouveau projet est créé dans le projet qui contient la tâche.</li> <li>3. Tous les éléments contenus dans la tâche sont déplacés dans le nouveau projet (contexte, notes, nom...).</li> <li>5. La tâche est ajoutée à ce nouveau projet.</li> </ol>
Extensions	
Variations	
Priorité	Moyenne

Fréquence	Environ une fois par semaine
-----------	------------------------------

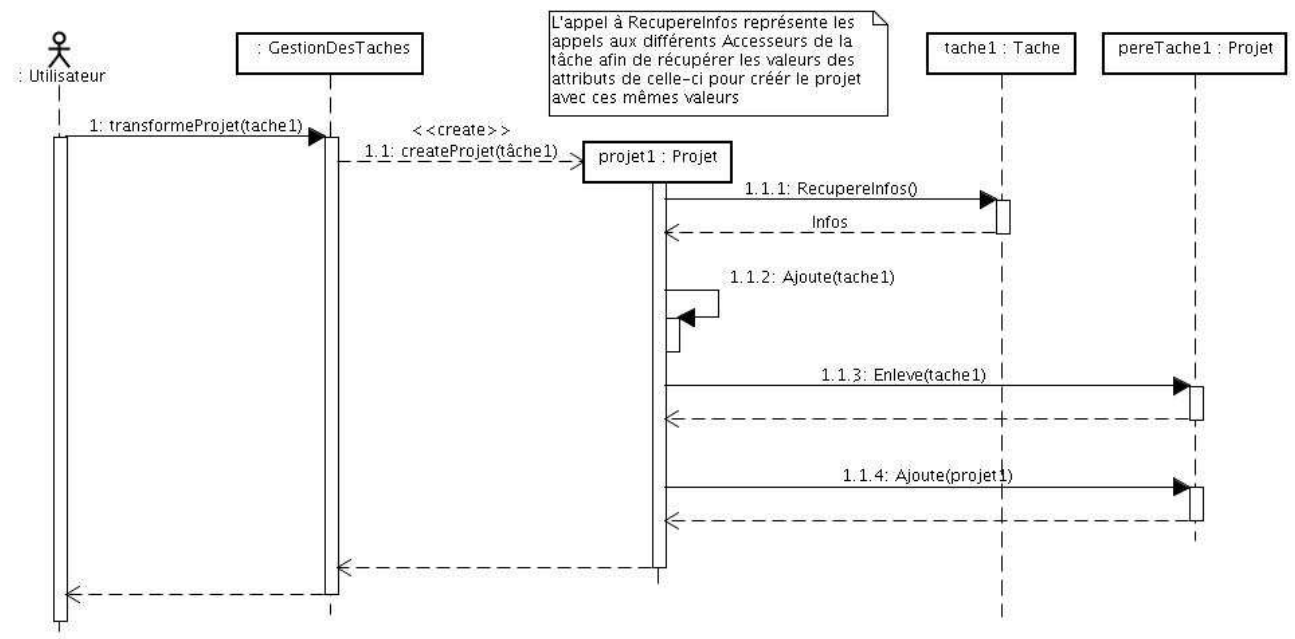


FIGURE 3 : SCENARIO TRANSFORMER UNE TACHE EN PROJET

Ce diagramme dynamique exprime bien les interactions entre les classes lors de la transformation de la tâche 1 en projet 1. La référence vers le projet qui contient tâche 1 (pereTache1) est obtenue grâce à l'appel à la méthode RecupereInfos.

Les exigences associées à ces fonctionnalités sont les suivantes :





- ✓ Le système doit posséder un mécanisme d'accès et de sauvegarde des éléments de l'utilisateur (tâches, projets...). Ceci sera réalisé par l'utilisation d'une base de données en local et sur le serveur, contacté via CORBA.
- ✓ Le système devra bien différencier les types de données (tâches, notes, contact, projet...); ceci pouvant être fait par l'utilisation des Classes JAVA.
- ✓ Le système doit posséder un mécanisme de gestion de session d'utilisateur, afin de s'assurer d'accéder uniquement aux éléments appartenant à l'utilisateur actif. Il faudra donc définir l'entité utilisateur dans les composants du système afin de pouvoir associer un utilisateur à ses éléments. Il y aura donc une Classe représentant l'utilisateur, pour l'instant il s'agit de la classe Espace\_De\_Travail dans l'analyse du problème.
- ✓ De même, les éléments devront donc posséder une référence (attribut ou autre) sur l'utilisateur auquel ils appartiennent.

## II.5. CAS D'UTILISATION *IMPORT/EXPORT DES DONNEES*

Le système propose à l'utilisateur une possibilité d'export de ses données. Nous présenterons les deux formats lui sont proposés, chacun répondant à des besoins distincts. De plus, l'utilisateur aura la possibilité d'importer des données dans l'application à partir d'un fichier spécifique.

### II.5.A. CAS D'UTILISATION EXPORT DES DONNEES

Le système propose à l'utilisateur une possibilité d'export de ses données. Deux formats lui sont proposés pour répondre à deux besoins distincts. Une première version que nous définirons comme « export papier », permettant à l'utilisateur d'imprimer ses données et ainsi de les transporter (en rendez-vous, en vacances, ...). La seconde version est un export dit de « sauvegarde ». Il respectera un format XML reconnu par l'application, ainsi l'utilisateur pour réimporter ses données. Cette version sera utilisée dans le cas d'un utilisateur possédant plusieurs machines et n'ayant pas accès au serveur. Nous pouvons par exemple imaginer un utilisateur utilisant notre application sur son PC à son domicile. A ce moment, il est encore relié au serveur. L'utilisateur sait qu'il va devoir séjourner dans un endroit sans connexion et avec un nouveau PC. Il souhaite pourtant continuer à utiliser son application avec ses données. Il exportera ainsi ses données sous le format XML, et pourra l'importer sur son nouveau PC.

Use Case	Export de sauvegarde des données.
Goal in context	L'utilisateur souhaite réaliser une sauvegarde de ses données sous forme d'un fichier, dans l'objectif de les réimporter dans l'application.
Scope	Export
Level	
Préconditions	✓ L'utilisateur possède un compte valide sur l'application.
Success end condition	 L'utilisateur possède un fichier d'export valide.  Toutes les données sont contenues dans le fichier d'export.
Failed end condition	 Il manque des données dans le fichier d'export.  L'utilisateur ne reçoit pas de fichier d'export valide.

Primary actor	L'utilisateur
Trigger	L'utilisateur utilise l'interface homme machine proposée par le client pour demande la génération d'un export de sauvegarde de ses données.
Main success scenario	1. L'utilisateur demande au client d'effectuer un export de sauvegarde de ses données. 2. Le client demande au serveur l'ensemble des données de l'utilisateur. 3. Le client formate les données reçues suivant le format XML défini. 4. Le client envoie le fichier à l'utilisateur.
Extensions	
Variations	
Priorité	Basse

L'export papier des données se déroule exactement de la même manière. Seul le format du fichier de sortie fourni à l'utilisateur change. Le type de fichier généré est imposé par le système : PDF. En effet, c'est un format lisible sur toutes les plateformes et dont la mise en forme est garantie.

---

#### II.5.B. CAS D'UTILISATION *IMPORT DES DONNEES*

---

Le système offre à l'utilisateur la possibilité d'importer des données dans l'application à partir d'un fichier de sauvegarde. Ce fichier aura été généré préalablement par le système.

Use Case	Import des données.
Goal in context	L'utilisateur souhaite importer des données à partir d'un fichier de sauvegarde.
Scope	Import.
Level	



Préconditions	<ul style="list-style-type: none"> <li>✓ L'utilisateur possède un compte valide sur l'application.</li> <li>✓ Le fichier d'import suggéré par l'utilisateur est valide, c'est-à-dire qu'il a été généré par le système lui-même.</li> </ul>
Success end condition	👉 L'ensemble des données définies dans le fichier sont sauvegardées par le client.
Failed end condition	👎 Il manque des données des données dans le système.
Primary actor	L'utilisateur
Trigger	L'utilisateur utilise l'interface homme machine proposée par le client pour demander l'import de données.
Main success scenario	<ol style="list-style-type: none"> <li>1. L'utilisateur demande l'affichage de l'écran concernant l'import de données.</li> <li>2. Le client demande à l'utilisateur de télécharger le fichier décrivant les données à sauvegarder.</li> <li>3. Le client analyse les données décrites dans le fichier, et crée toutes les instances correspondantes.</li> <li>4. Le client demande au serveur de sauvegarder toutes les instances créées.</li> <li>5. Le client confirme la bonne sauvegarde des données à l'utilisateur.</li> </ol>
Extensions	
Variations	<ul style="list-style-type: none"> <li>- Le client n'est pas connecté au serveur distant, il sauvegarde donc les informations dans la base de données locale et actualise la date de dernière modification.</li> </ul>
Priorité	Basse

### III. EXIGENCES DES INTERFACES EXTERNES

---

#### III.1. INTERFACE UTILISATEUR

---

L'application devra proposer les éléments généralement attendus dans toute application bureautique permettant de manipuler du texte. Il s'agit en particulier des fonctions permettant de couper, copier et coller du texte. Il peut également s'agir de mise en forme élémentaire du texte comme la couleur, et les différentes variations de caractères (taille, couleur, grasse).

#### III.2. INTERFACE LOGICIELLE

---

L'application outillant la méthode GTD est amené à être utilisée par une même personne dans des lieux différents et sur des plate-formes différentes. L'installation du logiciel doit donc être simple et rapide quelque soit le système d'exploitation possédé par l'utilisateur. Un logiciel développé en utilisant le langage Java permet un déploiement indifférencié sur l'ensemble des postes de travail. De plus, une telle application ne nécessite pas d'installation, un simple téléchargement suffit. Ce langage nous permet donc bien de répondre aux besoins de déploiement de l'application. C'est donc celui-ci que nous utiliserons pour le développement du logiciel outillant la méthode GTD.

Si on souhaite pouvoir accéder à la création d'un courriel à partir d'une adresse courriel présente dans l'application, il revient à l'utilisateur (ou à l'administrateur responsable du poste client) de faire le choix, l'installation, la configuration d'un client courriel ainsi que de souscrire aux services correspondants (compte courriel de type IMAP/SMTP fourni par un tiers). De même, il revient à l'utilisateur de configurer son système d'exploitation pour affecter un client courriel par défaut.

#### III.3. INTERFACE OU PROTOCOLE DE COMMUNICATION

---

Étant donné que l'application est un logiciel client, elle doit pouvoir se connecter au serveur via une interface réseau standard (type TCP/IP) configurée sur le poste client. Le produit utilisera une interface réseau mise à sa disposition par le système d'exploitation. Il appartient aux administrateurs du poste client de s'assurer que la configuration réseau du poste client et de l'infrastructure réseau permettant de relier le client au serveur est fonctionnelle.

#### III.4. CONTRAINTES DE MEMOIRE

---

L'empreinte mémoire de l'application doit être mesurée. En effet, le logiciel a vocation à être utilisé en même temps que d'autres applications. Au repos, le logiciel doit permettre aux autres applications de fonctionner sans que l'utilisateur ne constate de ralentissement.

## IV. AUTRES EXIGENCES NON-FONCTIONNELLES

---

### IV.1. UTILISABILITÉ

---

L'utilisabilité de l'application dépendra grandement de l'interface homme-machine choisie. Celle-ci fera l'objet d'un livrable indépendant de celui-ci, nous ne pouvons donc la présenter ici. Nous allons uniquement préciser les contraintes principales d'utilisation que l'interface utilisateur devra respecter.

La méthode GTD est basée sur un enregistrement rapide d'éléments avant de les organiser. Les fonctionnalités de création et de sauvegarde d'un nouvel élément devront être disponibles à l'utilisateur immédiatement au lancement de l'application.

L'utilisateur souhaitera consulter plusieurs fois par jour les tâches qui lui sont possibles à effectuer immédiatement. La fonctionnalité de recherche de tâches réalisables doit également être disponible dès le lancement de l'application.

Quotidiennement, le panier de l'utilisateur devra être purgé. Cette fonctionnalité devra donc facilement être accessible sans pour autant être au premier plan de l'application. L'ajout de tâches, de projet, de contacts ou de contextes peut se faire aussi bien de manière directe que lors de la purge quotidienne du panier. L'utilisateur doit donc avoir accès à ces fonctionnalités à tout moment. Généralement plusieurs de ces actions seront réalisées successivement, elles devront donc être simples et rapides.

### IV.2. FIABILITE

---

L'application utilisant la méthode GTD que nous proposons n'est pas destinée à une utilisation critique. L'équipe de développement devra assurer un minimum de fiabilité. Plusieurs outils permettant de mesurer la fiabilité du logiciel seront utilisés au cours du développement. Nous fixons un objectif de couverture du code par les tests unitaires à 80%. Cette donnée pourra être calculée par PMD. L'application sera réalisée avec le langage Java, les tests unitaires seront donc réalisés grâce à JUnit. Nous considérons également que 100% des tests unitaires écrits devront être validés.

### IV.3. EXIGENCES DE SECURITE

---

Notre application est destinée au grand public. Les utilisateurs n'ont donc pas d'exigences de sécurité renforcée. Le chiffage des données qui circulent sur le réseau ne sera pas assuré par le logiciel. Seule la sécurité commune offerte par le réseau utilisé sera garantie.

L'application doit être également suffisamment robuste pour assurer la persistance des données même en cas d'interruption inopinée de l'application. Le démarrage de l'application doit systématiquement s'initialiser sur les données saines qui ont été sauvegardées. L'application préférera restaurer le maximum de données, y compris de façon incomplète que de ne pas pouvoir démarrer parce que l'intégralité des données ne peut être chargée.

La méthode GTD permet une gestion personnelle de ses tâches et ses projets. C'est pourquoi nous amenons la notion d'espace de travail individuel. Ainsi, l'utilisateur A n'aura aucun droit d'accès aux données de l'utilisateur B et réciproquement. Pour assurer cette individualité, les espaces de travail devront être liés à un unique login. De plus, l'aspect confidentiel de chaque compte est conservé grâce à un mot de passe.

Un administrateur doit être capable de créer de nouveaux espaces de travail. A cette occasion il initialisera le login et le mot de passe, et les fournira au seul utilisateur concerné. Il sera conseillé à l'utilisateur de modifier immédiatement ce mot de passe. Pour assurer une confidentialité durable dans le temps, l'utilisateur devra également modifier régulièrement ce mot de passe. Une fréquence mensuelle de changement paraît acceptable.

La complexité du mot de passe ainsi que la fréquence de sa modification ne sera pas vérifiée par l'application.

#### IV.4. EXIGENCES DE PERFORMANCE

---

La méthode GTD peut-être appliquée sur différents supports. Nous pouvons aussi bien imaginer l'utiliser avec un bloc note qu'avec un logiciel. Notre logiciel doit permettre à l'utilisateur de gagner du temps vis à vis de l'application de cette méthode à la main. Il devra donc être suffisamment réactif pour répondre à cette exigence.

L'application est basée sur un modèle client/serveur, ses performances sont donc intimement liées à celles du serveur et de la transmission de données. Nous estimons que le temps de réponse du serveur devra être de l'ordre des 10ms. Cette valeur pourra être vérifiée par le lancement d'un *ping*. Notre équipe n'est pas chargée du développement et de la gestion du serveur, nous n'avons donc aucun contrôle sur l'efficacité du serveur.

Le client que notre équipe devra développer possède aussi une grande part de responsabilité dans les performances de l'application. Les requêtes demandées au serveur devront être suffisamment rapides à exécuter pour que l'application ne montre pas de forts ralentissements lors de l'affichage d'informations. Pour l'ensemble des opérations usuelles (ajout d'un projet, visualisation des tâches, ...) le temps de réponse de l'application devra être inférieur à 200ms. De manière générale, l'ensemble des actions devront être réalisées en moins de 10 secondes. En effet, au delà de ce seuil l'utilisateur remet en question la fiabilité du système. Si le délai de réponse de l'application dépasse l'ordre de la seconde, une barre d'avancement devra rassurer l'utilisateur dans sa confiance envers le bon fonctionnement du logiciel.

L'application développée est supposée être utilisée plusieurs fois par jour. L'utilisateur peut donc désirer laisser le logiciel au repos en permanence et donc l'avoir à disposition dès qu'il le souhaite. L'application ne doit donc pas consommer trop de ressources afin de permettre à d'autres logiciels de fonctionner au même moment.

#### IV.5. GESTION DE LA PERSISTANCE

---

Les données entrées par l'utilisateur doivent être rendues persistantes par notre système. Il doit exister deux modes de fonctionnement de l'application : connecté ou hors connexion. En effet, l'utilisateur doit pouvoir gérer ses tâches à tout moment, et donc quelques soient les conditions de connexions à sa disposition.

---

##### IV.5.A. LE MODE CONNECTE

---

L'application en mode connecté fonctionne en interaction avec un serveur dont nous n'avons pas la charge du développement. La sauvegarde des instances est entièrement gérée par le serveur. Notre client effectue uniquement la demande de sauvegarde.

Pour montrer les interactions entre notre client et le serveur, nous prenons l'exemple de la sauvegarde d'un élément du panier initiée par l'utilisateur. Le comportement de notre client sera identique que ce soit pour la modification, la sauvegarde ou la suppression de n'importe quelles instances du système.

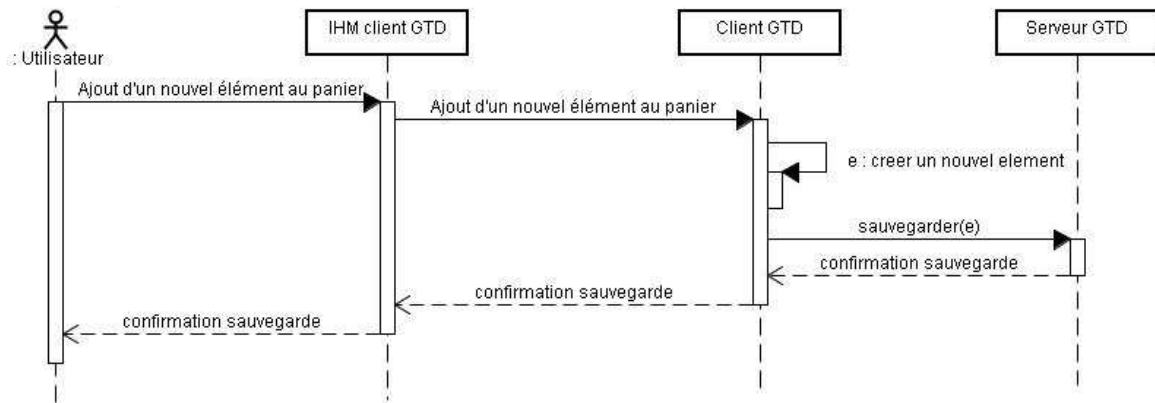


FIGURE 4 : SCENARIO DE SAUVEGARDE EN MODE CONNEXE

#### IV.5.B. LE MODE HORS-CONNEXION

Dans le cas de l'utilisation de notre système sans possibilité d'accès au serveur distant, on dit que l'on utilise le mode hors-connexion de l'application. Les données manipulées sont donc celles sauvegardées localement par le système. Ce mode doit être transparent pour l'utilisateur, il doit avoir accès exactement aux mêmes fonctionnalités.

Reprenons l'exemple de la sauvegarde d'un élément du panier utilisé précédemment, mais en mode hors-connexion cette fois-ci.

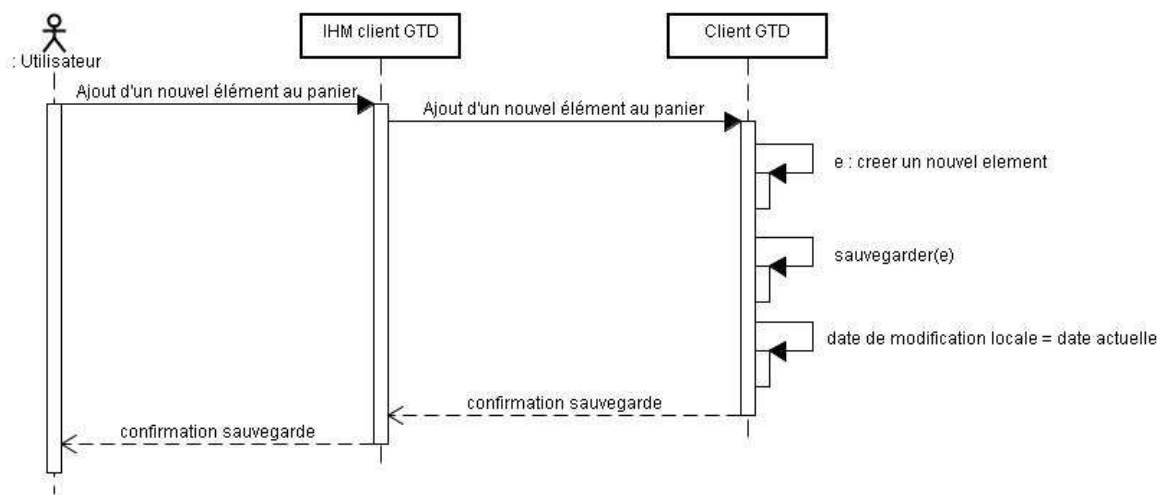


FIGURE 5 : SCENARIO DE SAUVEGARDE EN MODE HORS-CONNEXION

#### IV.5.C. SYNCHRONISATION DES DONNEES

Le système utilise des données provenant alternativement de sites différents : en local ou sur le serveur. Afin que le changement de base de données soit transparent pour l'utilisateur, il est essentiel qu'il y ait une synchronisation des données entre les deux sites.

Considérons tout d'abord quelques hypothèses de travail. Nous partons d'une hypothèse de travail forte : un utilisateur est connecté uniquement sur un seul poste au même moment. Nous interdisons les connexions multiples d'un même utilisateur. Cette hypothèse est cohérente, en effet, la méthode GTD permet une gestion personnelle de ses tâches. Il serait donc étrange de constater une double connexion d'un même utilisateur. Ainsi, une seule base de données peut être modifiée à la fois, soit la locale soit celle sur le serveur.

La base de données locale est jumelée avec une date de dernière modification. Celle-ci est actualisée par notre client dès qu'une modification intervient sur la base de données. De plus, notre système doit également posséder une date de dernière synchronisation entre la base de données locale et le serveur. Ces deux dates seront comparées afin de déterminer quelle base doit être mise à jour.

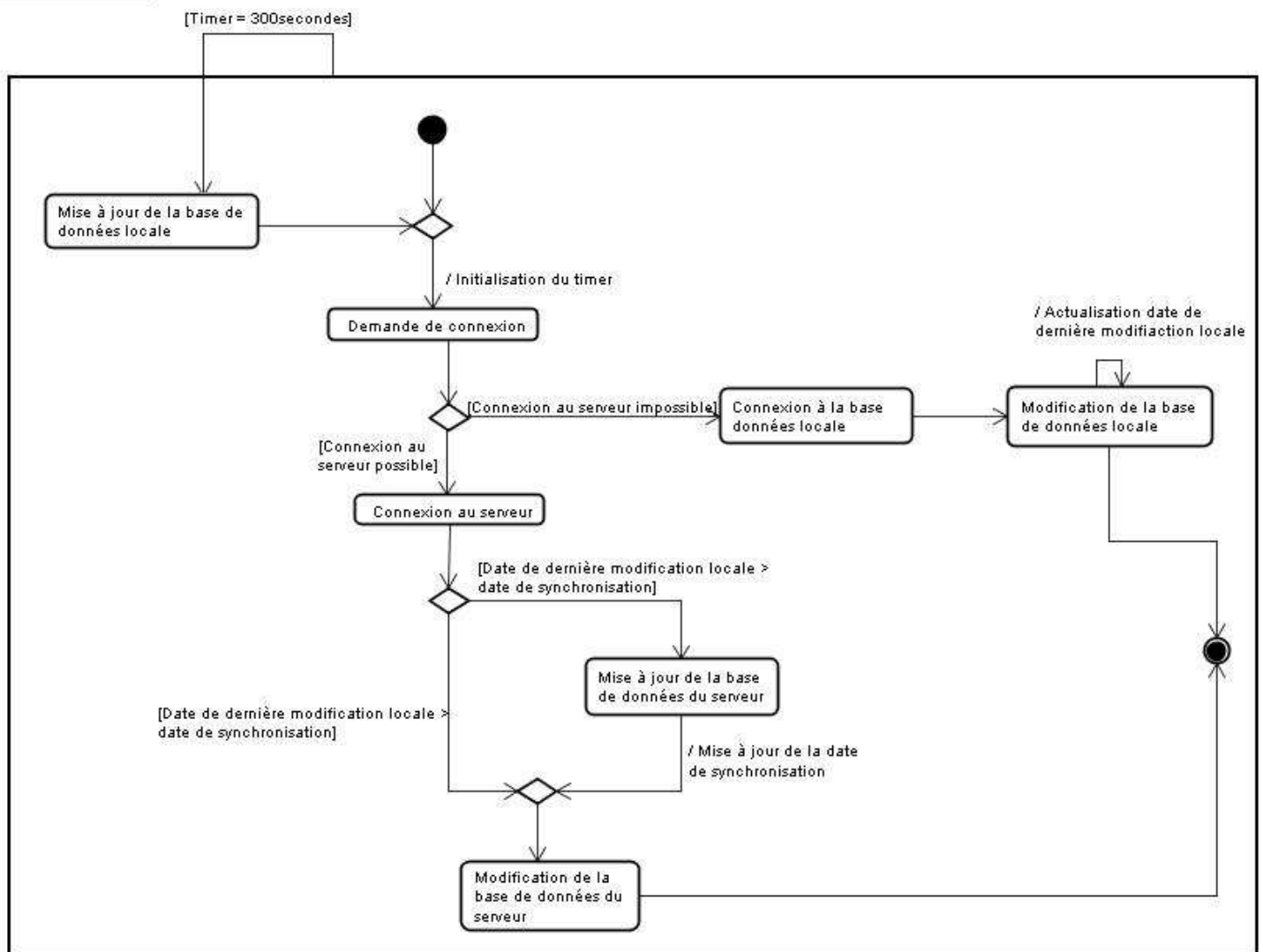


FIGURE 6 : DIAGRAMME D'ACTIVITE DE *SYNCHRONISATION*



Nous pouvons noter la présence d'un timer sur le diagramme d'activité, il permettra de synchroniser de manière automatique la base de données locale avec celle du serveur dans le cas d'une utilisation de l'application en mode connecté. Dans le cas d'un travail hors connexion le timer permet de retenter une connexion au serveur.

La mise à jour des données que ce soit sur le serveur ou en local s'effectue grâce à un fichier de LOG, enregistrant toutes les modifications de la base au fur et à mesure. La synchronisation s'effectue donc en exécutant toutes les transactions non réalisées sur le site voulue depuis la dernière date de synchronisation.



# Getting Things Done

## Livrable 3 : Spécifications des composants

Documents de spécifications des composants de l'application GTD

**BRUN Clémentine, LE NY Brendan, PONGE Myrtille,  
RAVENET Brian  
14/01/2010**

## TABLE DES MATIÈRES

Introduction .....	1
I. Division du problème en composants.....	2
I.1. Diagramme de composants simplifié .....	2
I.2. Comportement des composants .....	3
II. Interactions entre composants.....	5
II.1. Diagrammes de séquences et d'activité de la connexion .....	5
II.2. Diagrammes de séquences et d'activité de <i>collecter</i> .....	6
II.3. Diagramme de séquences et d'activité de <i>vider le panier</i> .....	8
II.4. Diagramme de séquences et d'activité de agir .....	13
II.5. Diagramme de séquences et d'activité de <i>reviser</i> .....	17
II.6. Diagramme de séquences de Sauvegarde.....	17
III. Spécifications des interfaces.....	20
III.1. Composant GestionnaireComptes .....	20
III.2. Composant IHMAdministrateur .....	20
III.3. Composant IHMConnexion.....	20
III.4. Composant IHMGTD.....	21
III.5. Composant GestionnaireConnexion.....	22
III.6. Composant GestionnairePanier .....	23
III.7. Composant GestionnaireRessources .....	23
III.8. Composants BDLocale et BDDistante.....	24
III.9. Composant Impression .....	28
III.10. Composant XML.....	29
III.11. Composant Persistance .....	29
IV. Diagramme de composants.....	30

## INTRODUCTION

---

Nous avons pour objectif de construire un logiciel outillant la méthode GTD (Getting Things Done). Au cours des livrables précédents, nous avons eu l'occasion de présenter cette méthode de gestion des tâches (Livrable 1 : Analyse du problème) ainsi que les besoins auxquels le logiciel répondra (Livrable 2 : Spécification des besoins). Pour résoudre un problème complexe, il faut le diviser en sous-problèmes. Nous allons procéder de la même manière pour réaliser l'application, en la découpant en composants.

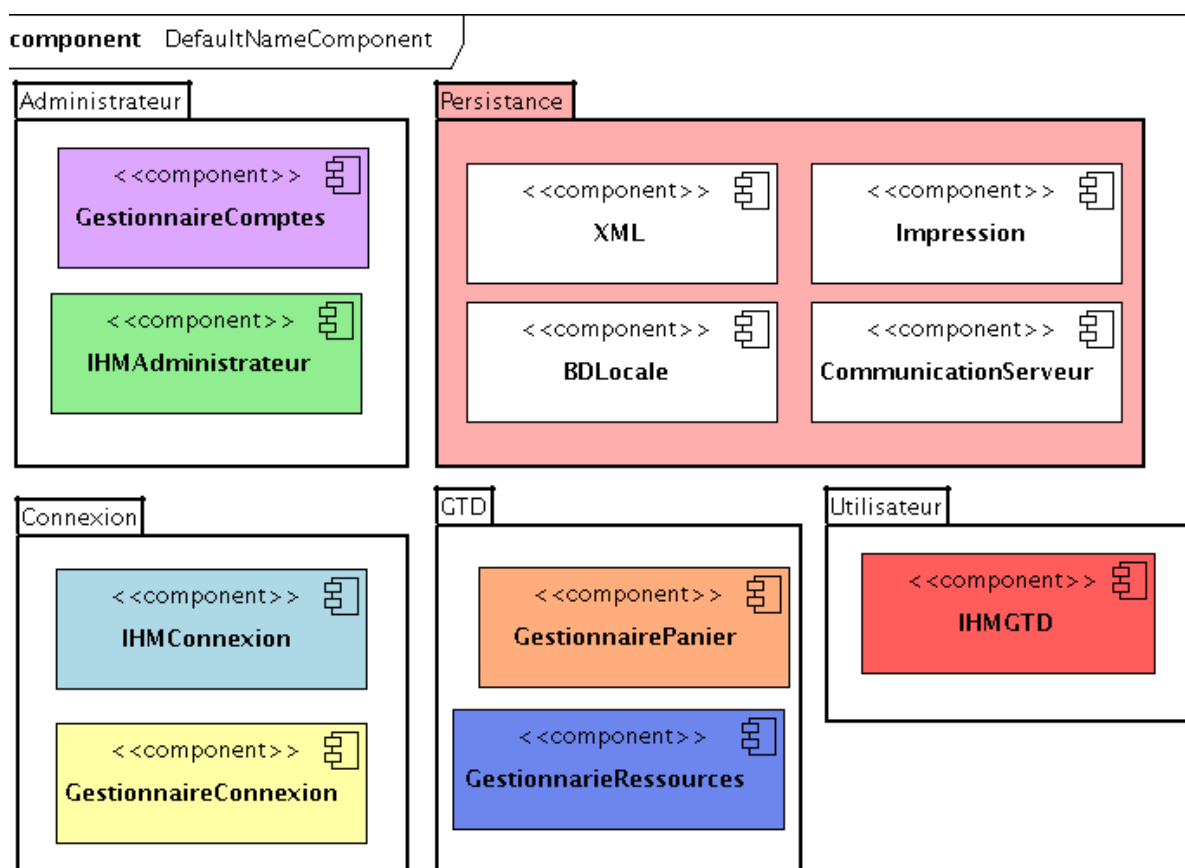
En plus de diviser le problème dans le but de le simplifier, le découpage du modèle en composant permet d'introduire une couche de généricité dans l'application. En effet, il sera relativement aisé de remplacer un composant par un autre du moment qu'il respecte les mêmes interfaces et le même comportement. L'évolution du logiciel s'en trouve donc simplifiée. Ce livrable présente la spécification des composants exhibés.

Nous avons construit notre rapport au fil de notre raisonnement. Ainsi, nous présentons tout d'abord le découpage du modèle en composants en repartant du diagramme de classe niveau analyse défini au cours du livrable 1. Nous mettons ensuite en évidence les différentes interactions qui ont lieu entre les composants. A partir de celles-ci, nous présentons finalement les interfaces que les composants offrent ou requièrent.

## I. DIVISION DU PROBLEME EN COMPOSANTS

Nous reprenons le diagramme de classe de niveau analyse défini dans le livrable 1 en y répercutant des modifications apportées entre temps. A partir de celui-ci, nous définissons le découpage du problème en composants. Enfin, nous spécifierons le comportement de chaque composant.

### I.1. DIAGRAMME DE COMPOSANTS SIMPLIFIE



Nous avons regroupé les composants en package afin de faciliter la compréhension. Les interfaces et les interactions entre les composants seront définies ultérieurement. Vous pourrez retrouver les couleurs utilisées comme toile de fond des composants sur le diagramme de classes. En effet cela permet de faire la correspondance entre les classes et les composants.

## I.2. COMPOTEMENT DES COMPOSANTS

---

### I.2.A. PACKAGE ADMINISTRATEUR

---

Le composant GestionnaireComptes regroupe l'ensemble des fonctionnalités liées aux comptes des utilisateurs de l'application. C'est à travers celui-ci, et en utilisant son IHM dédiée que l'administrateur de l'application pourra créer, modifier et supprimer des comptes. L'ouverture d'un compte correspond à la réservation d'un espace de travail pour un utilisateur donné. L'enregistrement d'un utilisateur se fait grâce à un mot de passe, un login et un identifiant permettant de définir le type d'utilisateur (administrateur ou utilisateur simple).

### I.2.B. PACKAGE CONNEXION

---

Le composant GestionnaireConnexion est utilisé dans le but d'ouvrir et de fermer les connexions avec l'application outillant la méthode GTD. L'IHM associée à ce composant permet l'affichage d'une page de connexion proposant à l'utilisateur de renseigner son login et son mot de passe en vue de se connecter. C'est également ce composant qui permettra à tout moment de connaître l'ensemble des utilisateurs connectés simultanément à l'application, et ainsi d'empêcher les connexions multiples.

### I.2.C. PACKAGE GTD

---

La méthode GTD offre à l'utilisateur l'utilisation d'un panier afin qu'il puisse collecter divers éléments. Ces éléments seront traités lors de la purge du panier. La gestion de ce panier est confiée au composant GestionnairePanier. Il devra fournir toutes les fonctionnalités exigées par la méthode GTD vis à vis du panier.

Le composant GestionnaireRessources devra donner accès à tous les éléments que l'utilisateur possède sur son espace de travail, hormis le panier. C'est à dire qu'il devra être capable d'offrir les fonctions de bases (CRUD : Create, Read, Update, Delete) sur les contacts, les contextes, les projets et les tâches d'un utilisateur.

Le composant de IHMGTD contient l'interface offerte à l'utilisateur pour manipuler les éléments de son espace de travail.

### I.2.D. PACKAGE PERSISTANCE

---

Le package Persistance regroupe tous les composants permettant la gestion de la sauvegarde des éléments de l'espace de travail de l'utilisateur. L'application outillant la méthode GTD offre

la possibilité d'utiliser de multiple modes de persistance. L'utilisation de l'un ou l'autre de ces modes sera transparent pour les autres composants de l'application grâce au composant Persistance. Celui-ci fait office de chef d'orchestre au sein du package. Le composant BDLocale offre une gestion locale de la persistance des données, et ainsi permet l'utilisation de l'application sans l'accès à une connexion réseau. L'utilisateur peut ainsi en toute circonstance avoir accès à ses données.

Le composant CommunicationServeur gère les comportement nécessaire pour accéder aux données distantes, c'est-à-dire qui ont été enregistrées sur le serveur.

L'application GTD offre la possibilité à l'utilisateur d'exporter ses données sous deux formats. Chacun est géré par un composant spécifique, XML pour la sauvegarde de la base de données au format XML en vue d'un futur import, et Impression pour une sortie papier dans un format lisible et compréhensible par un utilisateur humain.

## II. INTERACTIONS ENTRE COMPOSANTS

### II.1. DIAGRAMMES DE SEQUENCES ET D'ACTIVITE DE LA CONNEXION

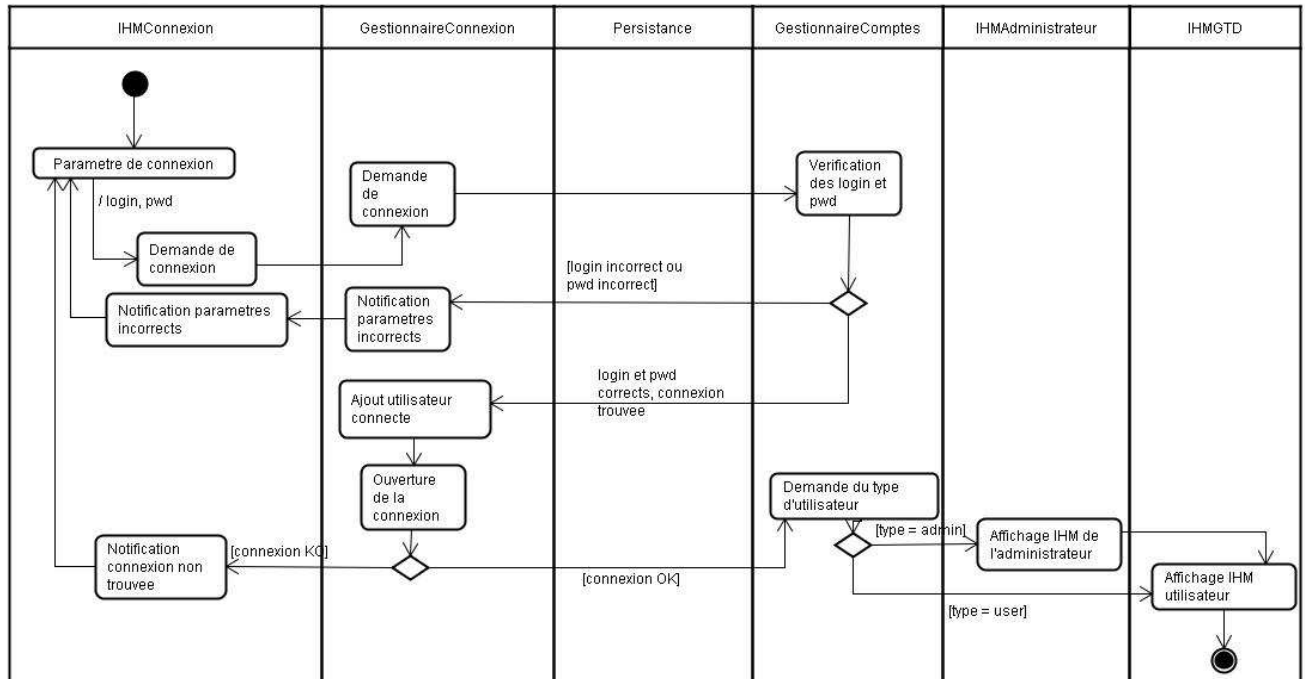


FIGURE 1 : CONNEXION

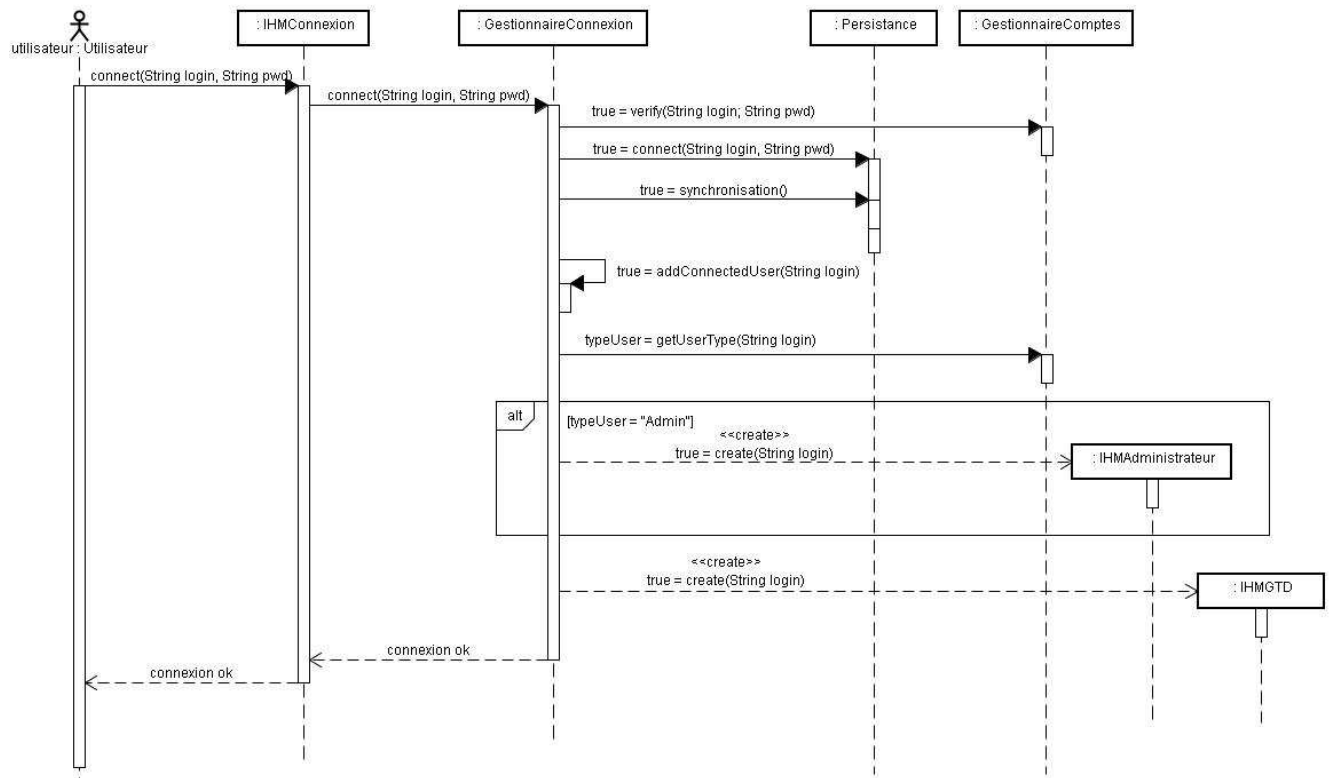


FIGURE 2 : CONNEXION



## II.2. DIAGRAMMES DE SEQUENCES ET D'ACTIVITE DE COLLECTER

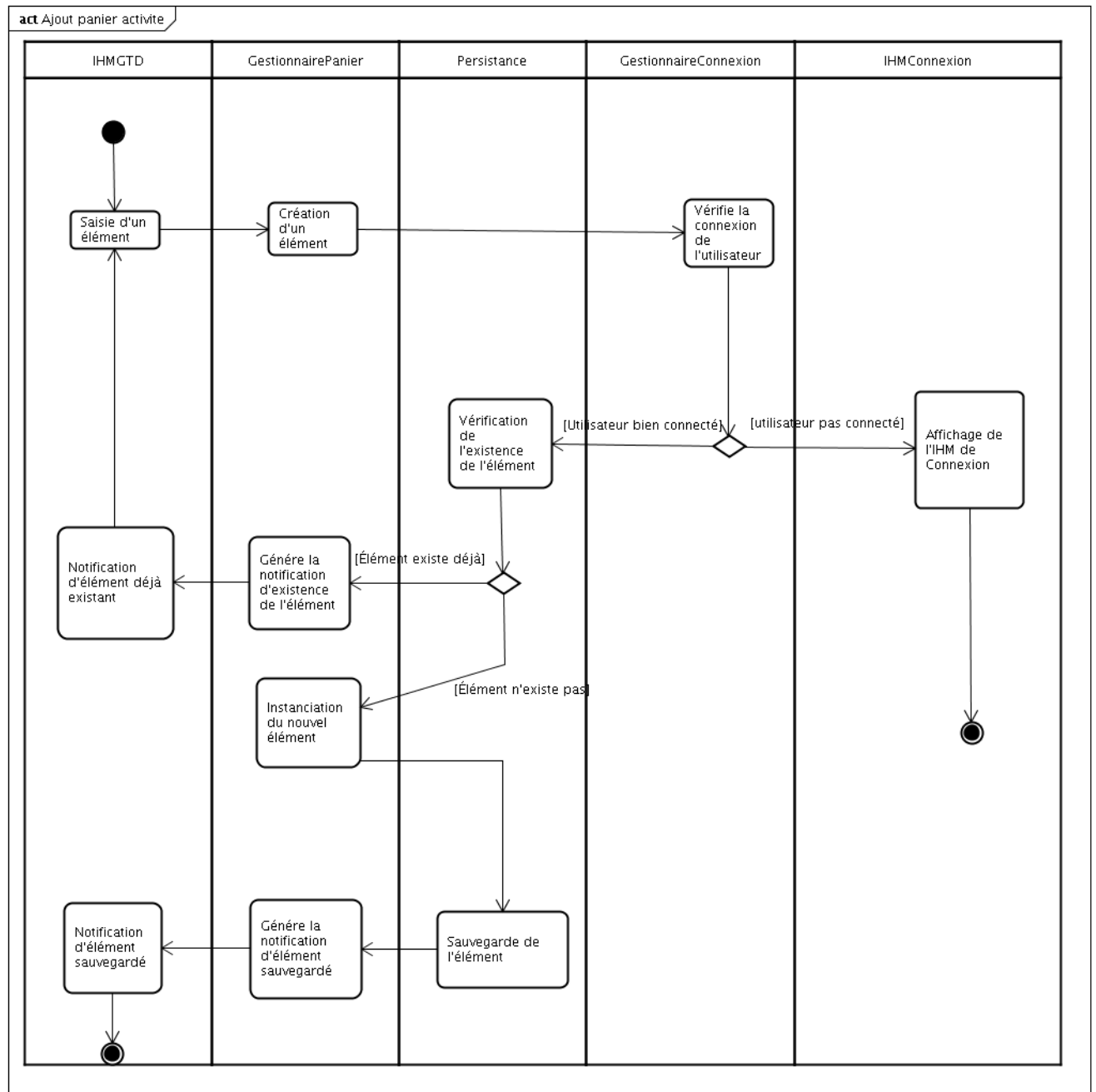


FIGURE 3 : COLLECTER

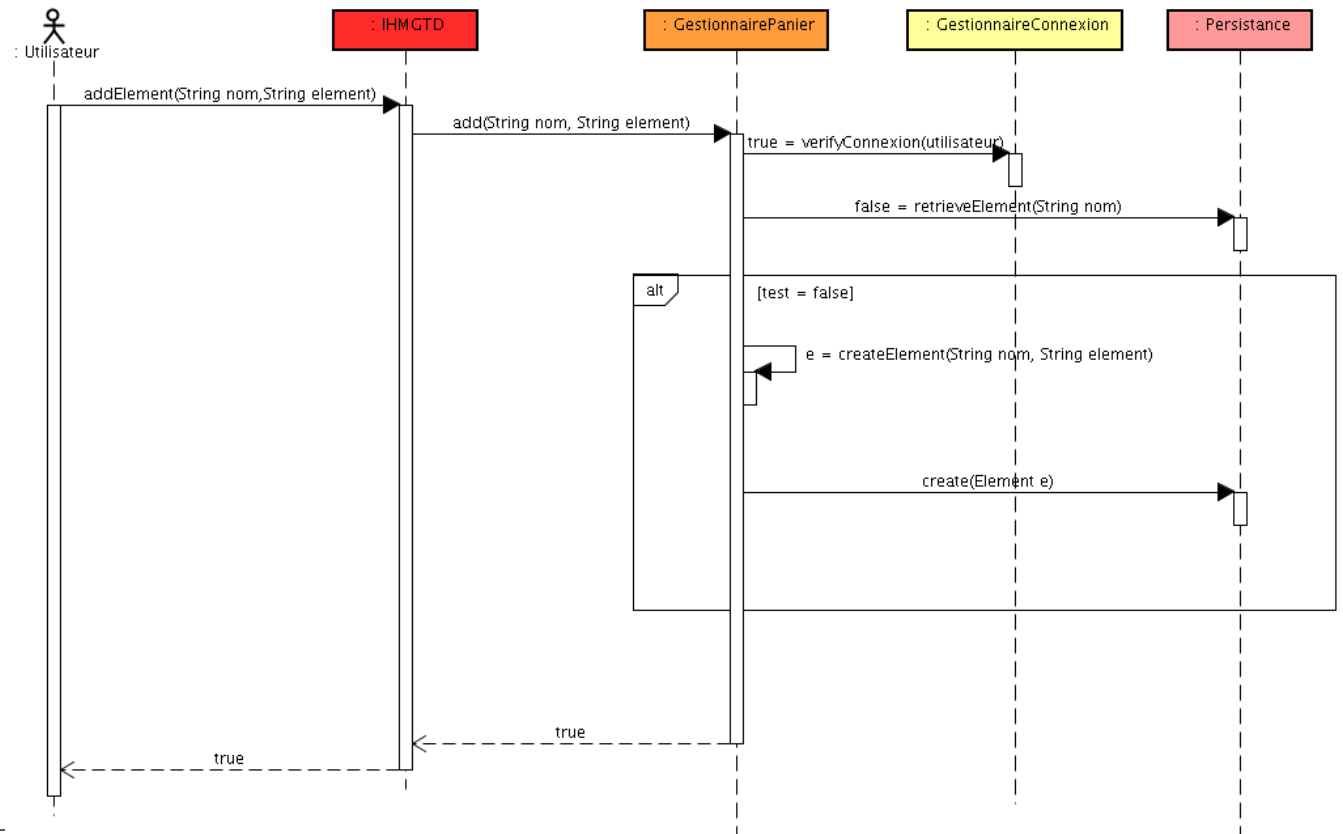


FIGURE 4 : COLLECTER

### II.3. DIAGRAMME DE SEQUENCES ET D'ACTIVITE DE *VIDER LE PANIER*

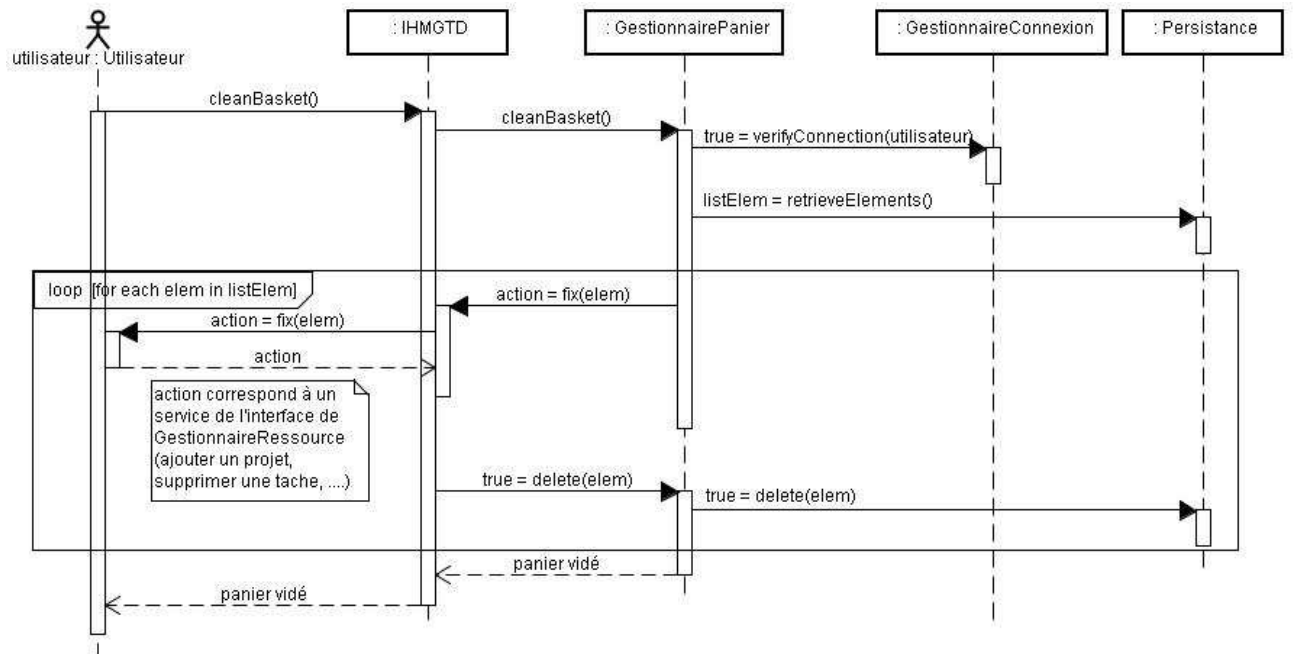


FIGURE 5 : VIDER LE PANIER

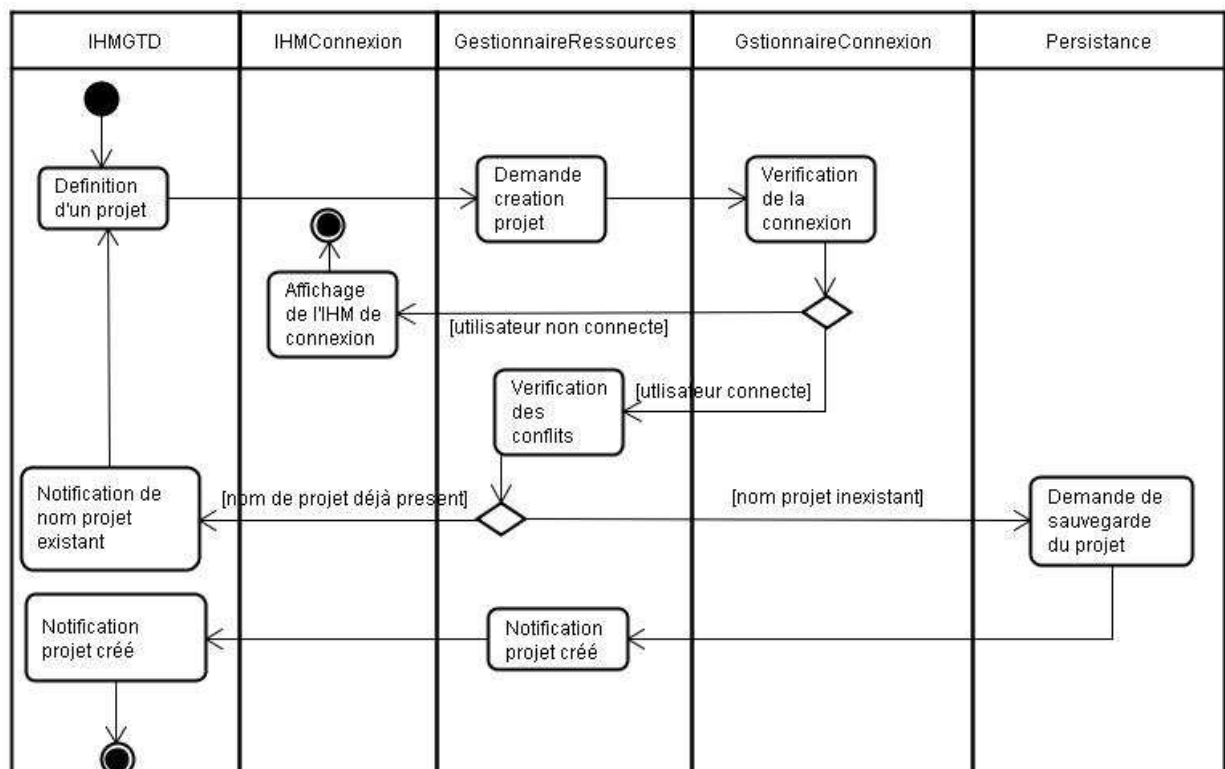


FIGURE 6 : CREER UN PROJET

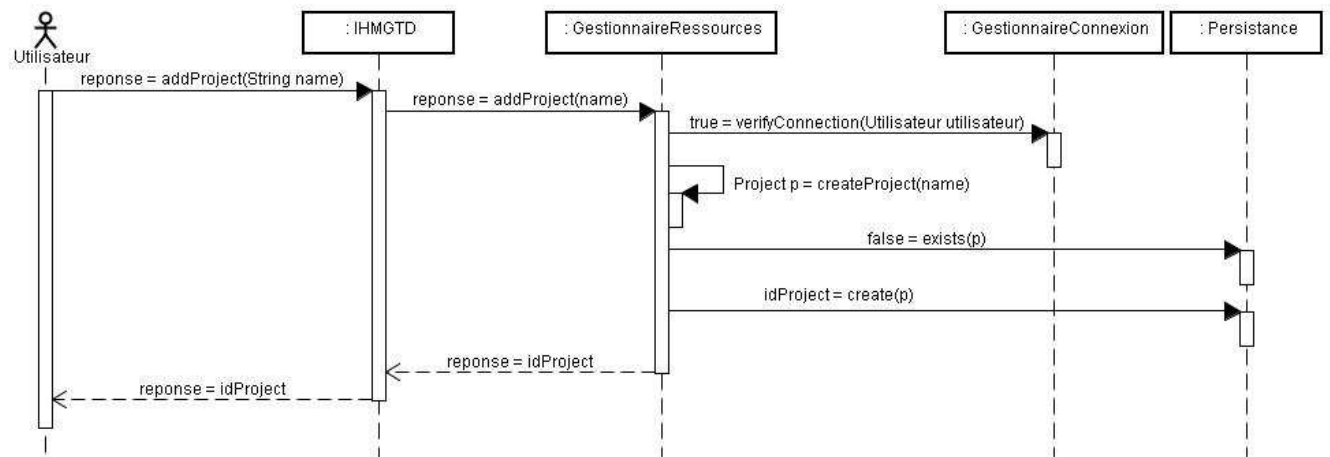


FIGURE 7 : CREER UN PROJET

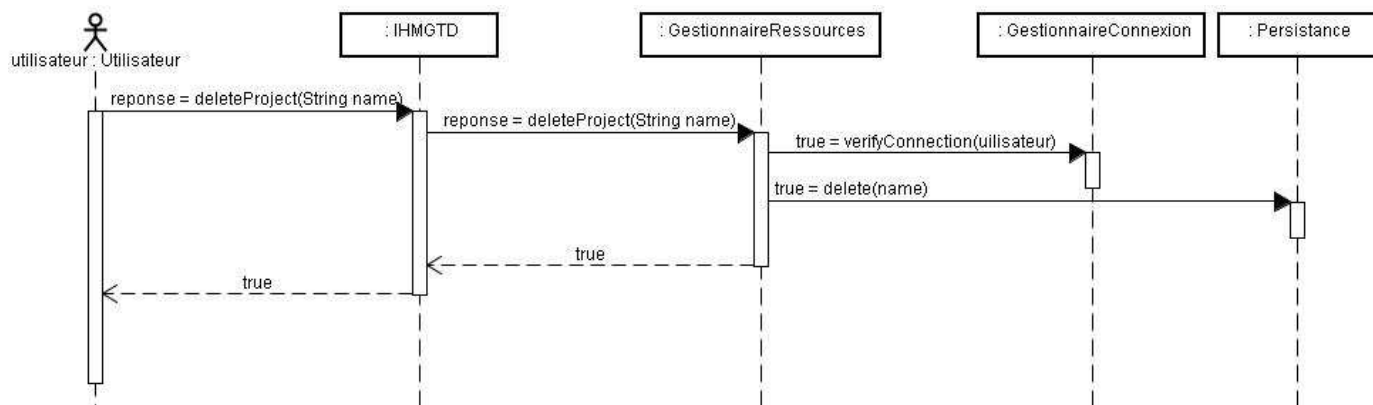


FIGURE 8 : SUPPRIMER UN PROJET

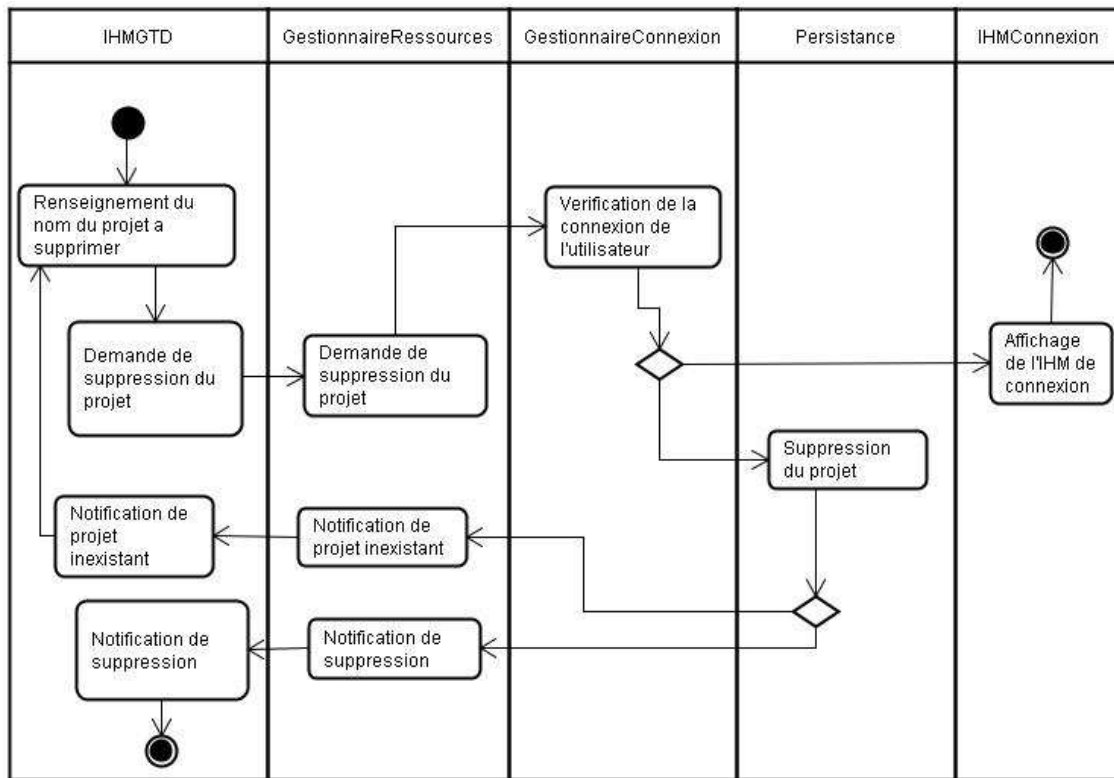


FIGURE 9 : SUPPRIMER UN PROJET

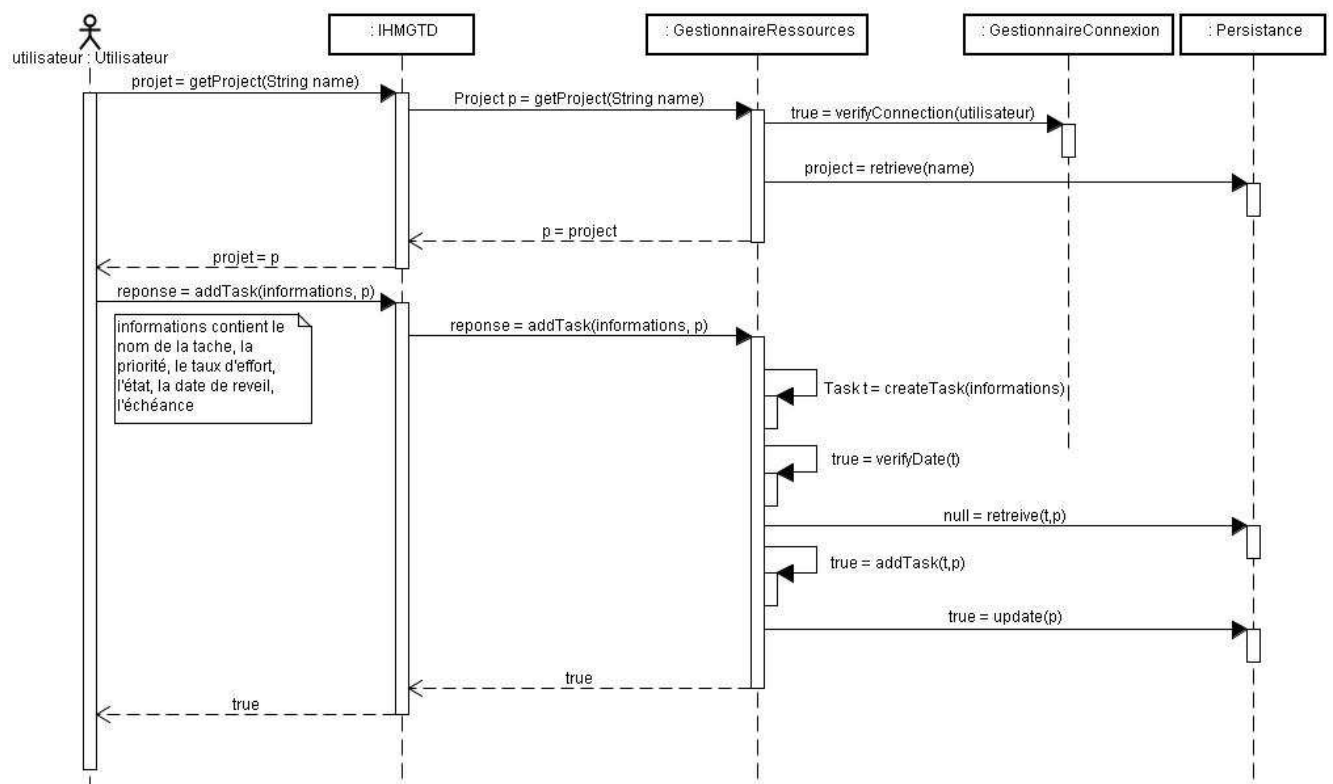


FIGURE 10 : AJOUT D'UNE TACHE A UN PROJET

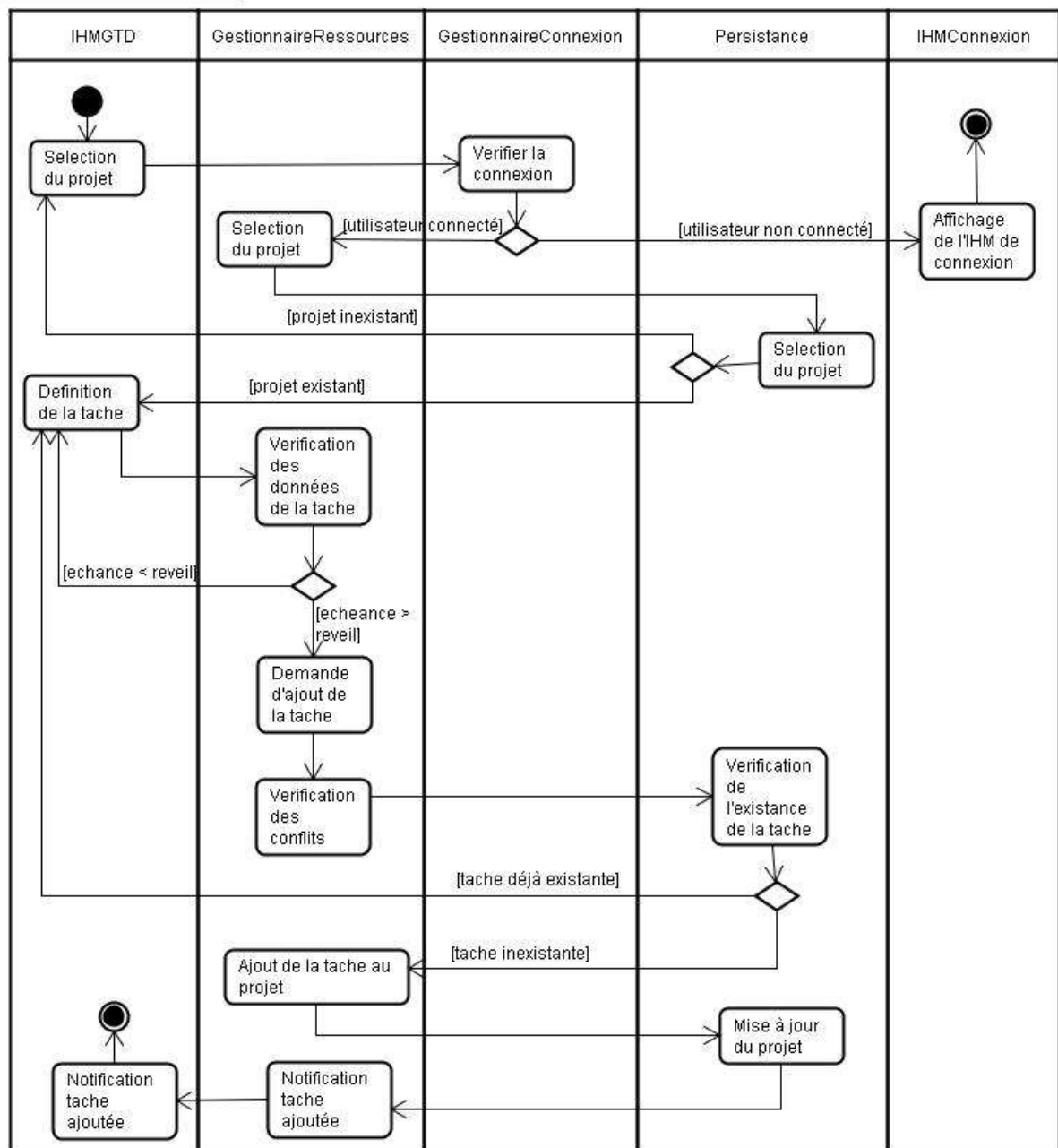


FIGURE 11 : AJOUT D'UNE TACHE A UN PROJET

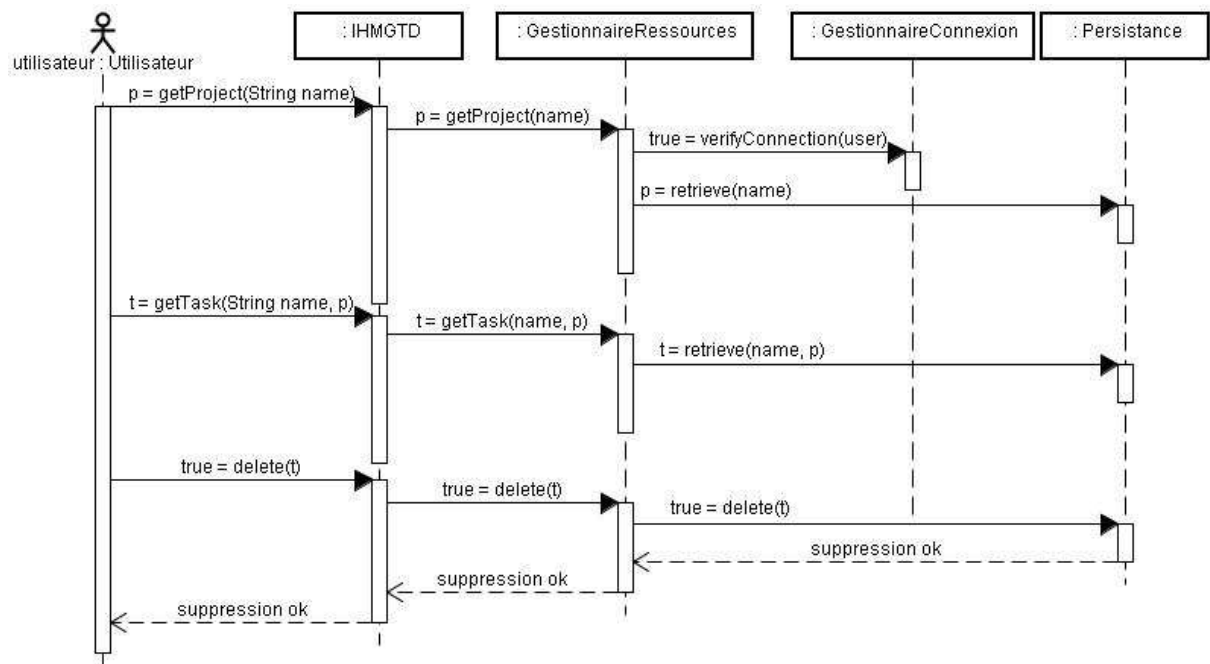


FIGURE 12 : SUPPRIMER UNE TACHE D'UN PROJET

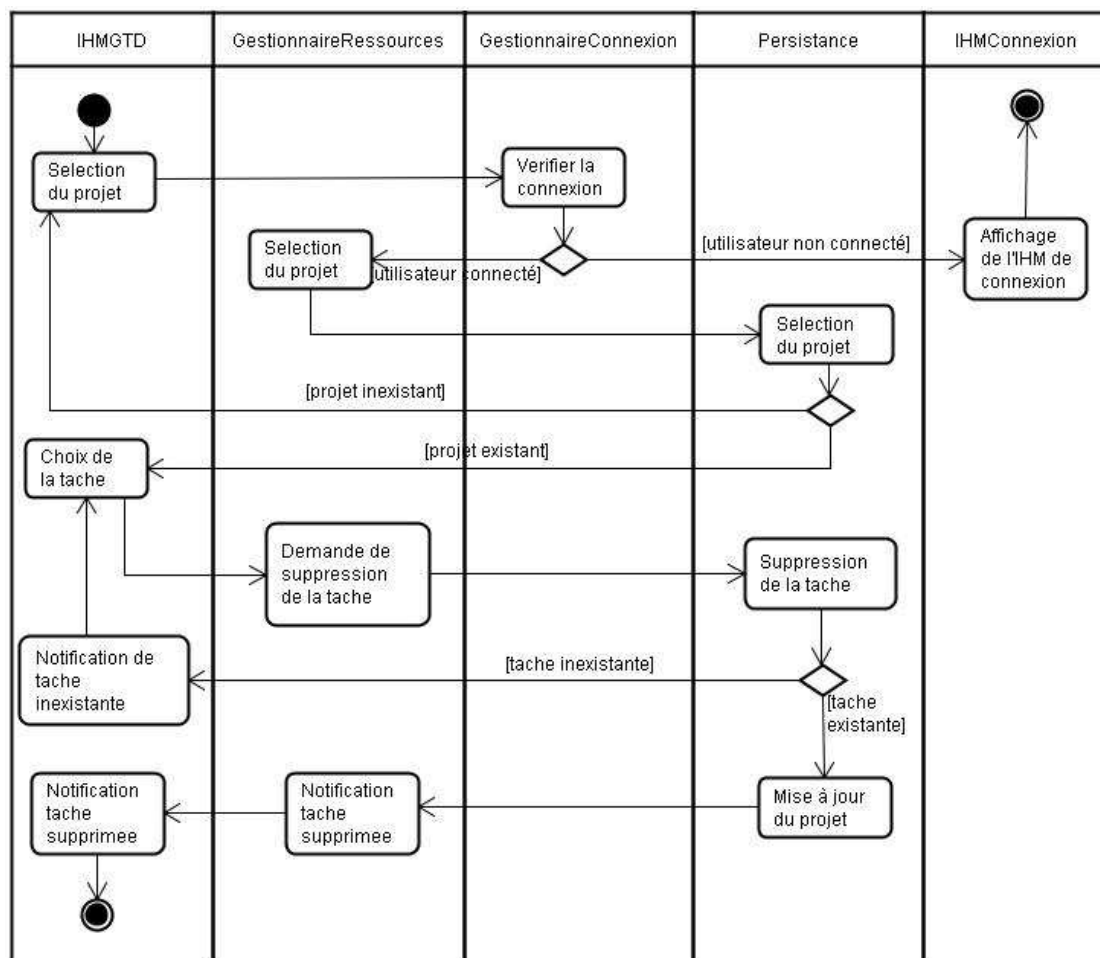


FIGURE 13 : SUPPRIMER UNE TACHE D'UN PROJET

## II.4. DIAGRAMME DE SÉQUENCES ET D'ACTIVITÉ DE AGIR

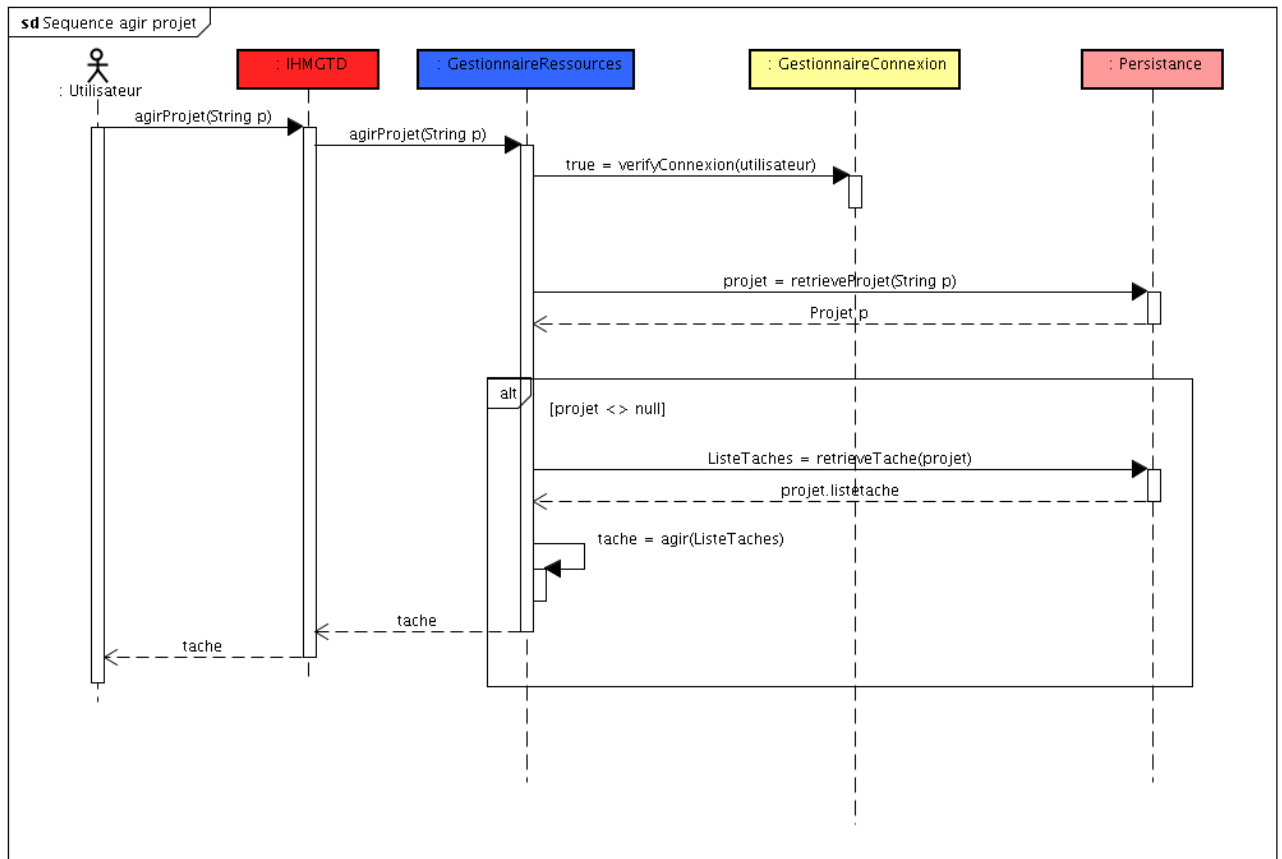


FIGURE 14 : AGIR SUR UN PROJET



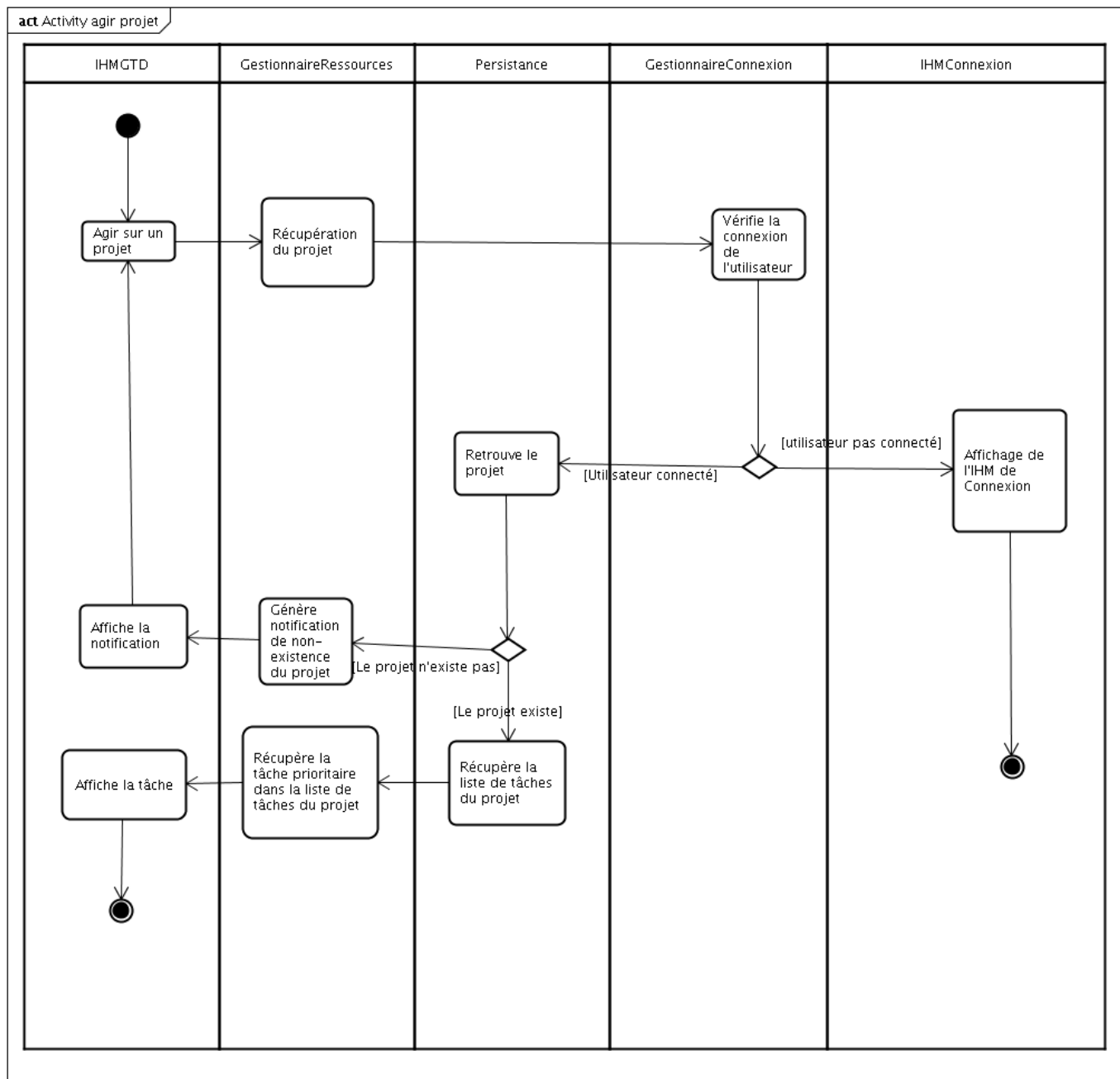


FIGURE 15 : AGIR SUR UN PROJET

Agir sur un contexte fonctionne de la même manière que Agir sur un projet. En effet la seule différence est l'appel de la méthode `retrieveTache()` du composant Persistence avec comme paramètre un contexte au lieu d'un projet.

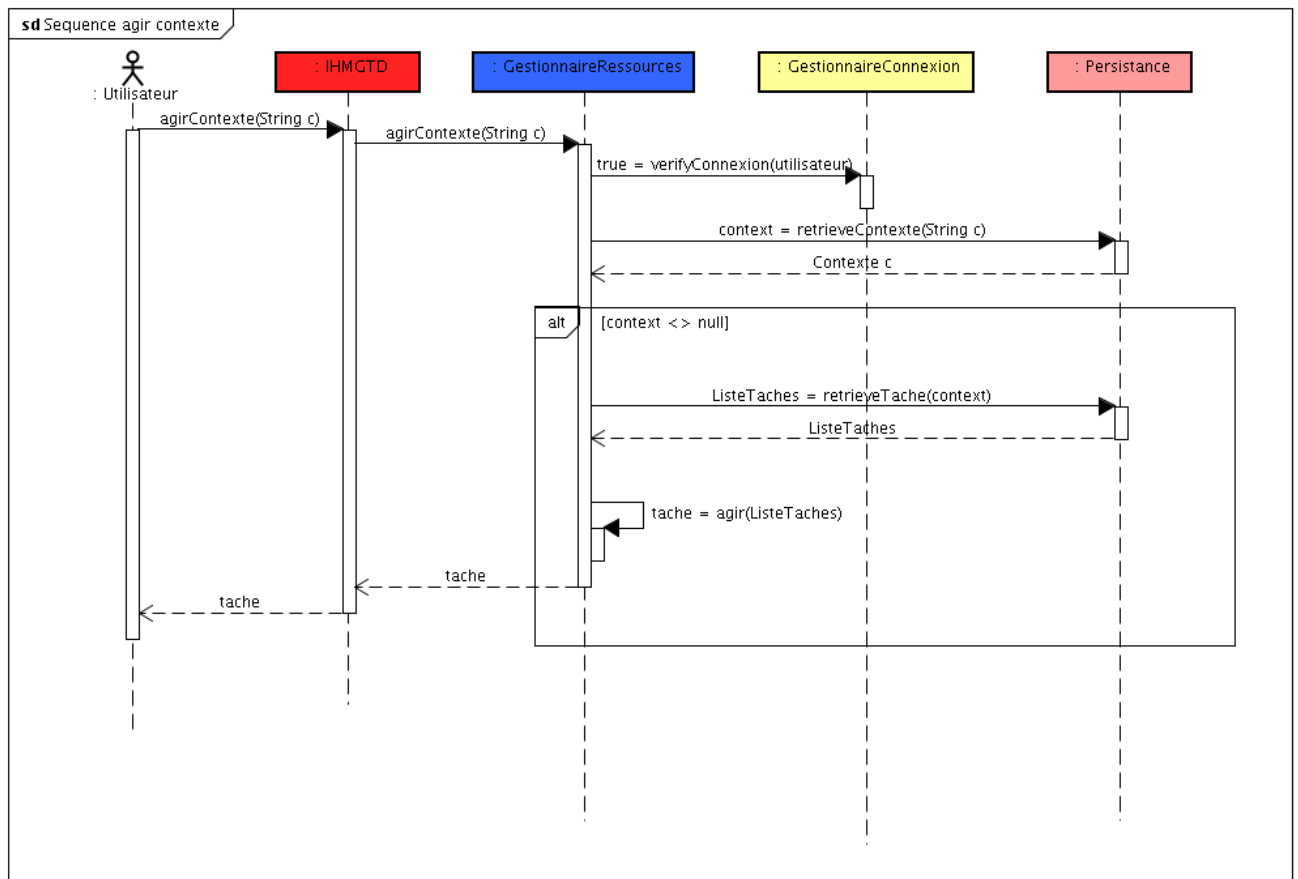


FIGURE 16 : AGIR SUR UN CONTEXTE

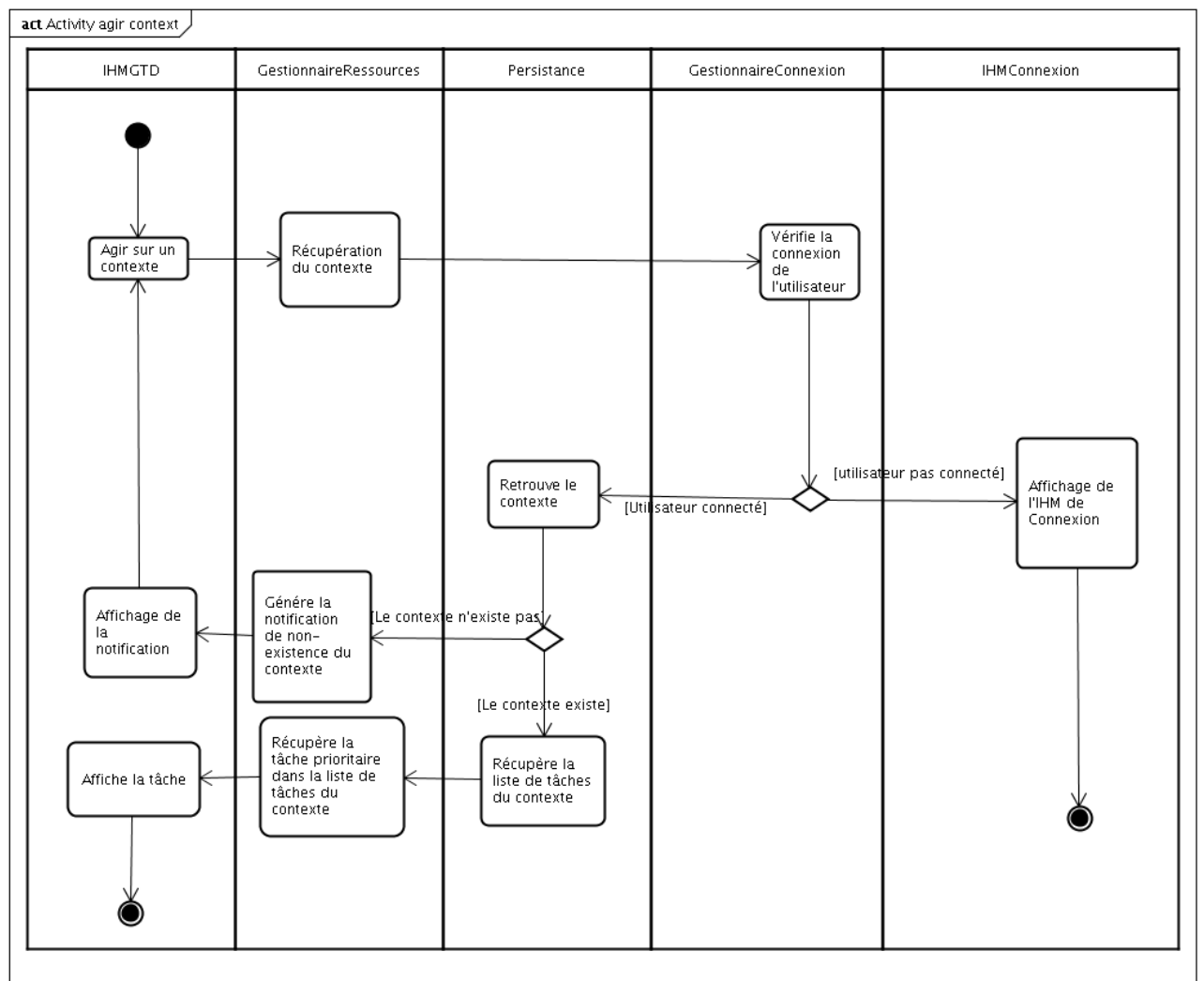


FIGURE 17 : AGIR SUR UN CONTEXTE

## II.5. DIAGRAMME DE SÉQUENCES ET D'ACTIVITÉ DE *REVISER*

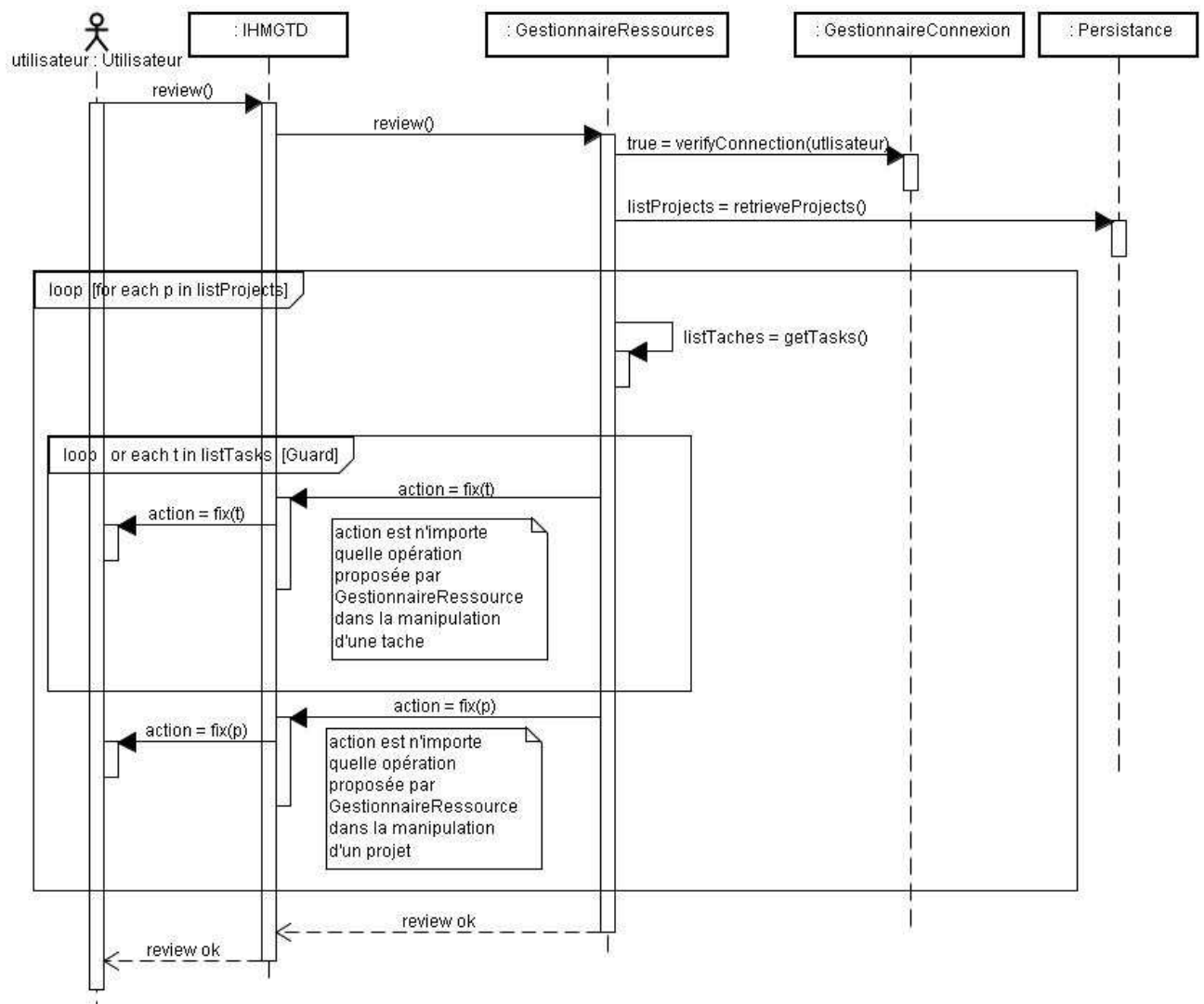


FIGURE 18 : REVISER

## II.6. DIAGRAMME DE SÉQUENCES DE SAUVEGARDE

Le composant Persistence est chargé d'assurer la sauvegarde des données aussi bien auprès de la base de données locale représentée par le composant BDLocale qu'auprès de la base distante représentée par le composant BDDistante.

Le composant Persistence interagit avec le reste du système comme une façade pour le package Sauvegarde. Il délègue les appels qui lui sont fait au différents composants de sauvegarde selon l'état du système de persistance (selon la disponibilité du serveur distant).

La façade Persistence propose aux autres composants du système un ensemble de méthodes suffisantes pour assurer l'ajout, la suppression, l'obtention et la mise à jour des données

persistantes. Il s'agit de méthodes sous la forme CRUD : create, retrieve, update, delete qui prennent en paramètre. Chacun des appels CRUD sur la façade sont délégués aux composants BDLocale et BDDistante.

Nous allons donc intégrer des méthodes CRUD qui seront proposées dans l'interface de BDLocale et dans BDDistante. Ces méthodes seront également remontées à la façade Persistence pour être disponible pour les autres composants.

## II.6.A. SYNCHRONISATION DES INFORMATIONS PERSISTANTES

À l'initialisation du système ainsi que lors d'un éventuel rétablissement de la connexion avec le serveur distant. Il faudra procéder à une synchronisation des données.

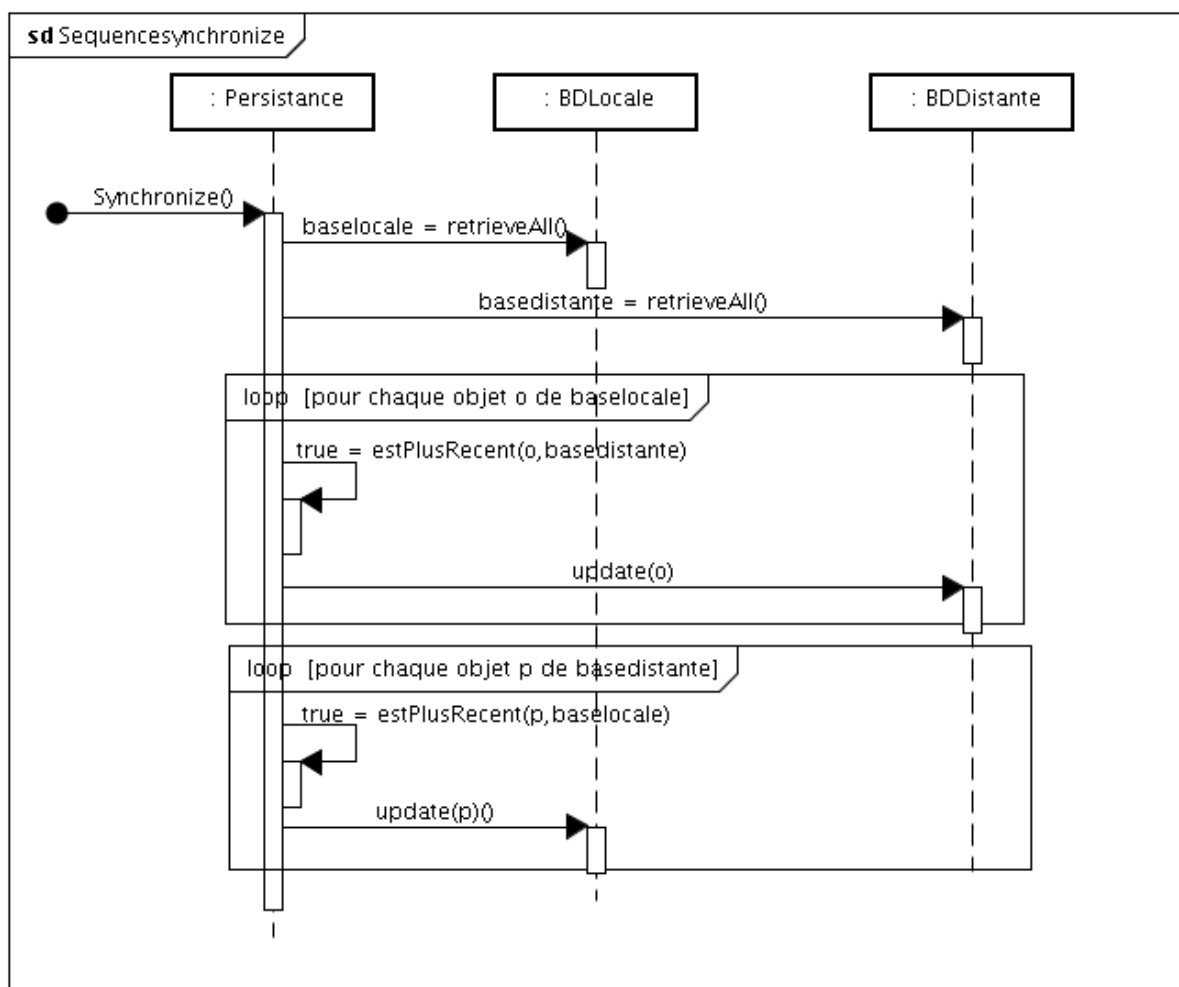


FIGURE 19 : SYNCHRONISATION

## II.6.B. EN IMPRESSION

Le composant Impression permet de sortir dans un fichier au format PDF, une tâche ou un projet et les informations associées.

Il est possible d'omettre le chemin vers lequel le fichier doit être créé. Dans ces cas, il est demandé à l'utilisateur.

---

#### II.6.C. EN XML

---

Le composant XML permet l'export de toutes les informations dans l'espace de travail dans l'état où elles sont au moment de l'export. Le fichier ainsi généré a vocation à être importé dans le client GTD. De même que pour l'impression, le chemin où enregistrer le fichier d'export est demandé s'il n'est pas fourni.

### III. SPECIFICATIONS DES INTERFACES

---

#### III.1. COMPOSANT GESTIONNAIRECOMPTE

---

`verify(String login, String pwd) : Boolean`

Pré-conditions : login et pwd sont des chaînes de caractères non nulles.

Post-conditions : retourne vrai si il le login existe bien et que le pwd correspond à ce login.

`getUserType(String login) : integer`

pré-conditions : login est une chaîne de caractères non nulle. login est l'identifiant d'un utilisateur possédant un compte sur l'application.

post-conditions : L'entier retourné correspond à un type d'utilisateur. Le type d'utilisateur correspond est le type d'utilisateur de l'utilisateur ayant pour identifiant login.

#### III.2. COMPOSANT IHMADMINISTRATEUR

---

`create(String login) : Boolean`

pré-conditions : login est une chaîne de caractères non nulle. login est l'identifiant d'un utilisateur appartenant à la liste des utilisateurs connectés sur l'application. login correspond à un utilisateur de type administrateur.

post-conditions : retourne vrai si l'IHM d'administration correspondant à l'utilisateur possédant l'identifiant login est affichée.

#### III.3. COMPOSANT IHMCONNEXION

---

`connect(String login, String pwd) : Boolean`

pré-conditions : login et pwd sont des chaînes de caractères non nulles.

post-conditions : Retourne faux si le login n'existe pas. Retourne faux si le pwd ne correspond pas au login. Retourne faux si la connexion a échoué. Retourne vrai si l'utilisateur correspondant au login est connecté.

### III.4. COMPOSANT IHMGTD

---

`create(String login) : Boolean`

pré-conditions : login est une chaîne de caractères non nulle. login est l'identifiant d'un utilisateur appartenant à la liste des utilisateurs connectés sur l'application.

post-conditions : retourne vrai si l'IHM de l'application GTD correspondant à l'utilisateur possédant l'identifiant login est affichée.

`addElement(String nom, String element) : integer`

pré-conditions : Les champs en entrée ne sont pas vides.

post-conditions : L'entier retourné est -1 si l'élément n'a pas été ajouté avec succès. Sinon c'est l'identifiant de l'élément créé.

`addProject(String name) : integer`

pré-conditions : name est une chaîne de caractères non nulle.

post-conditions : L'entier retourné est -1 si le projet n'a pas été ajouté avec succès. Sinon c'est l'identifiant du projet créé.

`getProject(String name) : Project`

pre-conditions : name est une chaîne de caractères non nulle.

post-conditions : Le projet retourné est null si name correspond à aucun nom de projet. Le nom du projet retourné est name.

`addTask(String name, integer priorite, integer effort, Etat etat, Date reveil, Date echeance, Project p) : integer`

pre-conditions : name est une chaîne de caractères non nulle. p est un projet non nul.

post-conditions : L'entier retourné est -1 si la tâche n'a pas été ajoutée avec succès. Sinon c'est l'identifiant de la tâche créée.

Si priorite et/ou effort sont nuls, la tâche créée les affecte à 0. Si etat est nul, la créée sera dans l'état par défaut AFaire. La date de reveil est antérieure à la date d'échéance si les deux sont non nulles. Le projet p contient la tâche créée.

`askForPath() : String`

pré-conditions :

post-conditions : le chemin renvoyé pointe bien vers un emplacement accessible en écriture



`agirProjet(String nomprojet) : tache`

pré-conditions : La chaîne en entrée ne doit pas être nulle.

post-conditions : La tâche retournée doit être vide seulement si le projet et ses sous-projets ne contenaient pas de tâches.

`agirContexte(String nomcontexte) : tache`

pré-conditions : La chaîne en entrée ne doit pas être nulle.

post-conditions : La tâche retournée doit être vide seulement si aucune tâche n'était associée à ce contexte.

`cleanBasket() : Boolean`

pré-conditions :

post-conditions : Retourne vrai si le panier de l'utilisateur est vide.

`fix(Element element) : void`

pré-conditions : element est non nul. element est contenu dans la liste d'éléments du panier de l'utilisateur.

post-conditions : une liste d'actions possibles est proposée à l'utilisateur : modifier/ajouter/supprimer une tâche/un projet/un contexte/un contact.

### III.5. COMPOSANT GESTIONNAIRE CONNEXION

---

`connect(String login, String pwd) : boolean`

pré-conditions : login et pwd sont des chaînes de caractères non nulles.

post-conditions : Retourne faux si le login n'existe pas. Retourne faux si le pwd ne correspond pas au login. Retourne faux si la connexion a échoué. Retourne vrai si l'utilisateur correspondant au login est connecté.

`addConnectedUser(String login) : boolean`

pré-conditions : login est une chaîne de caractères non nulle. login est l'identifiant d'un utilisateur de l'application GTD.

post-conditions : retourne vrai si l'utilisateur correspondant au login est ajouté à la liste des utilisateurs connectés à l'application GTD et qu'un timer associé à l'utilisateur est initialisé.

`verifyConnection (Utilisateur user) : boolean`

pre-conditions : user est un utilisateur non nul. user correspond à un utilisateur possédant un compte sur l'application GTD.

post-conditions : retourne vrai le timer associé à user était inférieur à 1800 secondes, et la valeur du timer est réinitialisée.

### III.6. COMPOSANT GESTIONNAIREPANIER

---

`delete(Element element) : Integer`

pré-conditions : element est non nul. element est un élément contenu dans le panier.

post-conditions : Le panier ne contient plus l'instance element.

`createElement(String nom, String nom) : Integer`

pré-conditions : Les chaines passées en paramètre ne sont pas nulles

post-conditions : Le nouvel élément est bien instancié.

`cleanBasket() : Boolean`

pré-conditions :

post-conditions : Retourne vrai si le panier de l'utilisateur est vide.

`fix(Element element) : void`

pré-conditions : element est non nul. element est contenu dans la liste d'éléments du panier de l'utilisateur.

post-conditions : element est supprimé.

### III.7. COMPOSANT GESTIONNAIRERESSOURCES

---

`createProject(String name) : Project`

pre-conditions : name est une chaine de caractères non nulle.

post-conditions : L'instance de Project retournée a pour nom name.

`getProject(String name) : Project`

pre-conditions : name est une chaine de caractères non nulle.

post-conditions : Le projet retourné est null si name correspond à aucun nom de projet. Le nom du projet retourné est name.

`verifyDate(Task t) : boolean`

pre-conditions : t est une tache non nulle.

post-conditions : Retourne vrai si la date de reveil de t est antérieure à la date d'échéance ou si au moins une des deux dates est nulle.

addTask(String name, integer priorite, integer effort, Etat etat,  
Date reveil, Date echeance, Project p) : Integer

pre-conditions : name est une chaine de caractères non nulle.

post-conditions : retourne vrai si une instance de Task possédant les informations fournies a été ajoutée à l'instance de Project p.

Si priorite et/ou effort sont nuls, la tâche créée les affecte à 0. Si etat est nul, la créée sera dans l'état par défaut AFaire. La date de reveil est antérieure à la date d'échéance si les deux sont non nulles.

agir(Liste listetaches) : tache

pre-conditions : La liste ne doit pas être vide

post-conditions : La tâche retournée est bien celle qui avait les priorités les plus hautes.

### III.8. COMPOSANTS BDLOCALE ET BDDISTANTE

---

exists(Element nom) : Boolean

pré-conditions : La chaîne en entrée ne doit pas être nulle.

post-conditions : Un booléen est retourné afin de confirmer ou pas l'existence de l'élément.

connect(String login, String pwd) : boolean

pré-conditions : login et pwd sont des chaines de caractères non nulles.

post-conditions : Retourne faux si le login n'existe pas. Retourne faux si le pwd ne correspond pas au login. Retourne faux si la connexion a échoué. Retourne vrai si l'utilisateur correspondant au login est connecté.

synchronisation() : boolean

pre-conditions :

post-conditions : les bases de données locale et distante ont été synchronisées, elles modélisent des états du système similaires. Renvoie vrai si l'opération est un succès, faux s'il y a un conflit ou si le serveur distant n'est pas joignable

create(Project p) : boolean

pré-conditions : Tous les éléments associés au projet sont connus par le système de persistance (tâches, sous-projets, contexte par défaut et notes).

post-conditions : L'entier retourné est -1 si le projet n'a pas été ajouté avec succès. Sinon c'est l'identifiant du projet créé.

Projet retrieveProjet(String name)

pre-conditions :

post-conditions : L'objet renvoyé est restitué dans l'état dans lequel il a été sauvegardé. Si aucun projet de ce nom n'a été trouvé, c'est une référence nulle qui est renvoyée.

update(Projet p) : boolean

pre-conditions : Le projet p est connu du système de persistance

post-conditions : Les données persistantes concernant p sont représentatives de l'état de p dans l'application au moment de l'appel à update()

delete(Projet p)

pre-conditions : Le projet p est connu du système de persistance. Les tâches associées au projet ont été supprimées préalablement.

post-conditions : Les informations concernant p ont disparues ainsi que les références vers ce projet. Les données persistantes sont cohérentes : il n'y a pas de références à des entités qui n'existent plus.

create(Tache t) : integer

pre-conditions : Tous les éléments associés à la tâche sont connus par le système de persistance (contexte et notes). Dans les données persistantes, dans le projet auquel appartient la tâche, il n'y a pas déjà de tâche avec ce nom.

post-conditions : L'entier retourné est -1 si la tâche n'a pas été ajoutée avec succès. Sinon c'est l'identifiant de la tâche créée.

Tache retrieveTache(Projet p, String name)

pre-conditions : Le projet p est connu du système de persistance.

post-conditions : L'objet renvoyé est restitué dans l'état dans lequel il a été sauvegardé. Si aucune tâche de ce nom n'a été trouvée, c'est une référence nulle qui est renvoyée.

ListeTache retrieveTache(Projet p)

pre-conditions : Le projet p est connu du système de persistance.

post-conditions : La liste n'est pas vide si le projet contenait des tâches

ListeTache retrieveTache(Contexte c)

pre-conditions : Le contexte c est connu du système de persistance.

post-conditions : La liste n'est pas vide si le contexte était associé à des tâches

update(Tache t) : boolean

pre-conditions : La tâche t est connue du système de persistance

post-conditions : Les données persistantes concernant t sont représentatives de l'état de t dans l'application au moment de l'appel à update()

delete(Tache t)

pre-conditions : La tâche t est connue du système de persistance. Les notes associées à la tâche ont été supprimées préalablement.

post-conditions : Les informations concernant p ont disparues ainsi que les références vers ce projet. Les données persistantes sont cohérentes : il n'y a pas de références à des entités qui n'existent plus.

`create(Contact t) : integer`

pre-conditions : Il n'existe pas déjà de Contact avec ce nom

post-conditions : L'entier retourné est -1 si la tâche n'a pas été ajoutée avec succès. Sinon c'est l'identifiant de la tâche créée.

`Contact retrieveContact(String name)`

pre-conditions :

post-conditions : L'objet renvoyé est restitué dans l'état dans lequel il a été sauvegardé. Si aucun contact de ce nom n'a été trouvé, c'est une référence nulle qui est renvoyée.

`update(Contact c) : boolean`

pre-conditions : Le contact c est connu du système de persistance

post-conditions : Les données persistantes concernant c sont représentatives de l'état de t dans l'application au moment de l'appel à update()

`delete(Contact c)`

pre-conditions : Le contact c est connu du système de persistance.

post-conditions : Les informations concernant c ont disparues ainsi que les références vers ce projet. Les données persistantes sont cohérentes : il n'y a pas de références au contact c.

`create(Element e) : integer`

pre-conditions : L'élément n'est pas vide.

post-conditions : L'élément que l'on souhaite faire persister a bien été sauvegardé.

`update(Element e) : boolean`

pre-conditions : L'élément e est connu du système de persistance

post-conditions : Les données persistantes concernant e sont représentatives de l'état de e dans l'application au moment de l'appel à update()

`delete(Element e)`

pre-conditions : L'élément e est connu du système de persistance

post-conditions : Les informations concernant e ont disparues ainsi que les références vers cet élément. Les données persistantes sont cohérentes : il n'y a pas de références à l'élément e dans le panier.

`list<Element> retrieveElements()`

pre-conditions :

post-conditions : La liste est vide si le panier ne contenait pas d'éléments.

`retrieve(Tache t, Projet p) : Boolean`

pre-conditions : p et t sont non nuls. p est un projet existant.

post-conditions : retourne vrai si t appartient à p.

`create(Note n) : integer`

pre-conditions :

post-conditions :

update(Note n)

pre-conditions : n est connu du système de persistance

post-conditions : Les données persistantes concernant n sont représentatives de l'état de n dans l'application au moment de l'appel à update()

delete(Note n)

pre-conditions : L'élément n est connu du système de persistance

post-conditions : Les informations concernant n ont disparues ainsi que les références vers cette note. Les données persistantes sont cohérentes : il n'y a pas de références à la note n dans le panier.

list<Note> retrieveNotes(Projet p)

pre-conditions : Le projet p est connu du système de persistance

post-conditions :

list<Note> retrieveNotes(Tache t)

pre-conditions : La tâche t est connue du système de persistance

post-conditions :

create(Contexte c)

pre-conditions :

post-conditions :

update(Contexte c)

pre-conditions : c est connu du système de persistance

post-conditions : Les données persistantes concernant c sont représentatives de l'état de c dans l'application au moment de l'appel à update()

delete(Contexte c)

pre-conditions : L'élément c est connu du système de persistance

post-conditions : Les informations concernant c ont disparues ainsi que les références vers cet élément. Les données persistantes sont cohérentes : il n'y a pas de tâche ou de projet qui font référence à c.

Contexte retrieveContext(Projet p)

pre-conditions : Le projet p est connu du système de persistance

post-conditions :

Contexte retrieveContext(Tache t)

pre-conditions : La tâche t est connue du système de persistance

post-conditions :

`Boolean estPlusRecent(Objet, List<Objet>)`

pré-conditions :

post-conditions : Si l'élément passé en paramètre est plus récent que celui dans la liste, alors la méthode doit renvoyer vrai.

`List<Objet> retrieveAll()`

pré-conditions : La base n'est pas vide

post-conditions : Aucun objet appartenant à la base n'est absent de la liste qui est retourné par cette méthode.

### III.9. COMPOSANT IMPRESSION

---

Dans tous les cas, si le chemin n'est pas fourni en paramètre, il est demandé via `askForPath()` par Persitance.

`exportPDF(Projet p, String chemin)`

pré-conditions : chemin désigne un chemin vers un fichier (emplacement et nom du fichier). Le répertoire doit être accessible en écriture.

post-conditions : le fichier PDF est écrit à l'emplacement demandé et a le nom demandé. Il recense l'intégralité des informations qui ont trait à p et à ses tâches.

`exportPDF(Projet p)`

pre-conditions :

post-conditions : le fichier PDF est écrit à l'emplacement demandé et a le nom demandé. Il recense l'intégralité des informations qui ont trait à p.

`exportPDF(Tache t, String chemin)`

pré-conditions : chemin désigne un chemin vers un fichier (emplacement et nom du fichier). Le répertoire doit être accessible en écriture.

post-conditions : le fichier PDF est écrit à l'emplacement demandé et a le nom demandé. Il recense l'intégralité des informations qui ont trait à t.

`exportPDF(Tache t)`

pre-conditions :

post-conditions : le fichier PDF est écrit à l'emplacement demandé et a le nom demandé. Il recense l'intégralité des informations qui ont trait à t.

### III.10. COMPOSANT XML

---

Dans tous les cas, si le chemin n'est pas fourni en paramètre, il est demandé via askForPath() par Persistance.

`exportXML(Espace_de_travail esp, String chemin)`

pré-conditions : chemin désigne un chemin vers un fichier (emplacement et nom du fichier). Le répertoire doit être accessible en écriture.

post-conditions : le fichier PDF est écrit à l'emplacement demandé et a le nom demandé. Il recense l'intégralité des informations qui ont trait à p et à ses tâches.

`exportXML(Projet p)`

pre-conditions :

post-conditions : le fichier XML est écrit à l'emplacement demandé et a le nom demandé. Il recense l'intégralité des informations qui ont trait à p.

`Espace_de_travail importXML(String chemin)`

pre-conditions : chemin désigne un chemin vers un fichier (emplacement et nom du fichier). Le répertoire doit être accessible en lecture.

post-conditions : le fichier PDF est écrit à l'emplacement demandé et a le nom demandé. Il recense l'intégralité des informations qui ont trait à p et à ses tâches.

`Espace_de_travail importXML()`

pre-conditions :

post-conditions : le fichier XML est écrit à l'emplacement demandé et a le nom demandé. Il recense l'intégralité des informations qui ont trait à p.

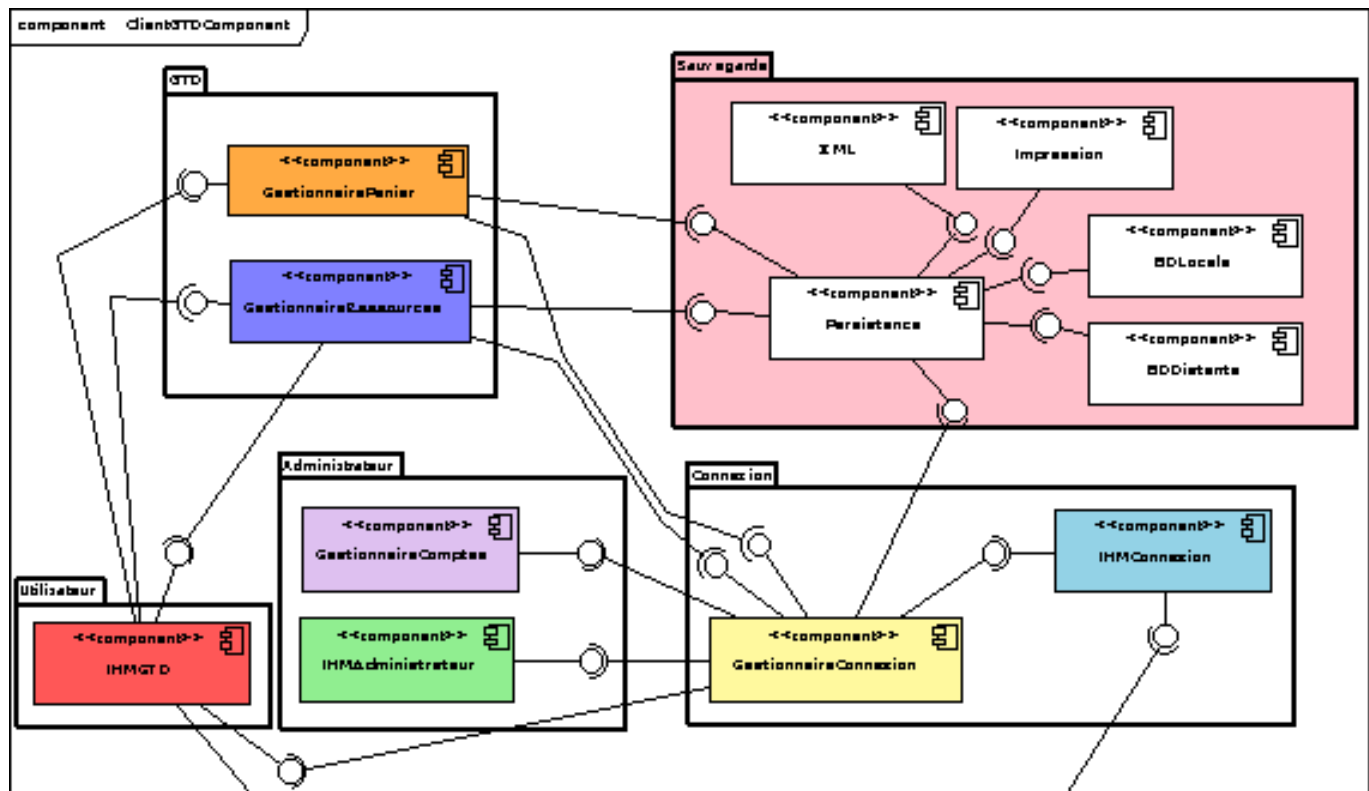
### III.11. COMPOSANT PERSISTANCE

---

Ce composant propose toutes les méthodes qui sont décrites ci-dessus : les même que celles proposées par BDDistante, BDLocale, Impression et XML. Cette agrégat forme une façade pour le reste du système.



#### IV. DIAGRAMME DE COMPOSANTS





# Getting Things Done

Livrable 4 : Architecture

**BRUN Clémentine, LE NY Brendan, PONGE Myrtille,  
RAVENET Brian  
19/11/2009**

## TABLE DES MATIERES

---

---

Introduction .....	1
I. Architecture physique de l'application .....	2
I.1. Les composants matériels.....	2
I.2. Les composants logiciels .....	2
II. Schéma des bases de données.....	4
III. Stéréotypes, étiquettes et sémantique.....	5
III.1. Stéréotypes de base .....	5
III.2. Stéréotypes de persistance.....	5
III.3. Stéréotypes de bases de données .....	5
IV. Règles de traduction de UML vers le code source.....	7
Conclusion.....	10

## INTRODUCTION

---

Au cours des livrables précédents, nous avons défini le cahier des charges ainsi que la spécification des besoins. Nous avons également construit un premier diagramme de classe de niveau analyse, accompagné développé par la spécification des composants de l'application ainsi que de leurs interfaces.

Nous allons ici, établir l'architecture de l'application. C'est-à-dire définir son modèle de base de données, ainsi que les composants matériels nécessaires à son développement et son déploiement. Nous définirons enfin les règles nous permettant de traduire le diagramme de classes UML en du code Java. Pour cela, nous serons aidés d'Acceleo.

## I. ARCHITECTURE PHYSIQUE DE L'APPLICATION

---

Définir l'architecture physique d'une application revient à décrire les composants matériels qui serviront de support à l'application. Ces descriptions s'accompagneront ici d'un diagramme de déploiement permettant de visualiser la manière dont les composants logiciels déployés sur les composants matériels.

### I.1. LES COMPOSANTS MATERIELS

---

L'application client outillant la méthode GTD est un logiciel informatique ne nécessitant aucun composant matériel spécifique. Deux modes de fonctionnement sont possibles, chacun demandant une configuration différente :

- Distant, avec un accès à un serveur au travers d'un réseau. Ce mode nécessite l'existence d'un serveur distant fonctionnel et répondant aux spécifications de l'application, ainsi qu'une connexion réseau avec ce serveur.
- Local, sans serveur distant. Ce mode ne nécessite aucun composant matériel spécifique.

Dans les deux cas l'application devra être déployée sur un ordinateur.

### I.2. LES COMPOSANTS LOGICIELS

---

#### I.2.A. LE CHOIX DU LANGUAGE

---

Une des contraintes principale de notre application est sa portabilité sur les différentes plateformes susceptibles d'être possédées par l'utilisateur. Le logiciel devra ainsi fonctionner de la même manière sur un système MAC OS, UNIX ou Windows. Le langage Java permet de répondre entièrement à cette contrainte, c'est donc celui-ci que nous choisissons pour développer l'application outillant la méthode GTD.

#### I.2.B. LA PERSISTANCE DES DONNEES

---

La persistance des données doit être assurée par deux modes distincts : local ou distant. Nous nous intéresserons dans un premier temps à la persistance locale des données.

Aucun protocole spécifique ne nous a été indiqué pour la gestion de la persistance locale. Notre application sera développée à l'aide du langage Java, nous nous sommes donc naturellement

tournés vers Hibernate<sup>1</sup>. C'est un framework open-source permettant d'utiliser des appels de méthodes objets au lieu des requêtes classiques à la base de données.

Nous nous sommes ensuite interrogés sur la pertinence d'utiliser les EJB<sup>2</sup>, qui sont une implémentation des spécifications d'Hibernate. Ils nécessitent d'être hébergés dans un serveur d'application, un des plus connus étant JBoss. Le déploiement d'un tel serveur est relativement long et coûteux en performances. Or, nous n'utiliserons pas pleinement les possibilités offertes par les EJB (utilisation de composants distribués). De plus, la rapidité de lancement de l'application est un des critères fondamentaux d'utilisabilité spécifié précédemment. Ce cout nous a donc semblé inutile et nous avons choisi de ne pas utiliser la technique EJB.

Nous avons ensuite du choisir la base de données qui permettra la persistance locale de l'application. Nous avons donc réalisées une rapide étude de différentes solutions afin de choisir celle nous correspondant le mieux.

Finalement, nous avons sélectionné la base de données H2<sup>3</sup>. Elle permet d'embarquer la base de données avec le logiciel. Ainsi, il n'y a pas de déploiement local à assurer. Elle répond à l'ensemble de nos critères, et nous a de plus été recommandée par Arnaud Thimel<sup>4</sup>.

La persistance distante est assurée par un serveur dont nous n'avons pas la gestion. Il nous faut donc trouver un protocole permettant d'accéder aux données qui y sont stockées. Ce protocole doit permettre de faire communiquer notre application cliente codée en Java, avec l'application Serveur dont nous ne connaissons pas les détails d'implémentation. Il existe deux solutions répondant à cette problématique : RMI et CORBA. Nous n'avons pas eu à faire le choix du protocole, en effet l'utilisation de CORBA nous est exigée.

---

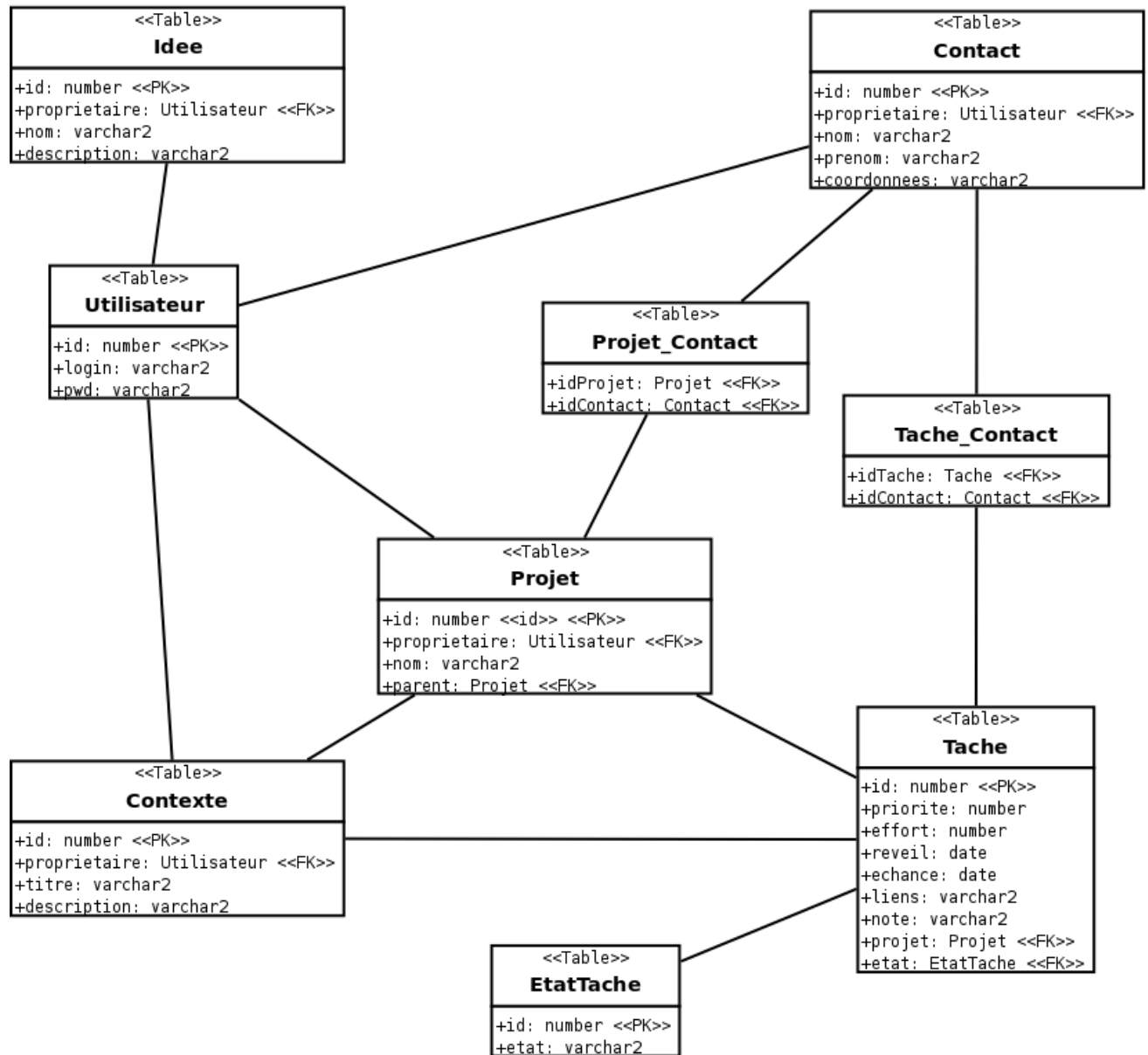
<sup>1</sup> Hibernate: <https://www.hibernate.org/>

<sup>2</sup> EJB: [http://fr.wikipedia.org/wiki/Enterprise\\_JavaBeans](http://fr.wikipedia.org/wiki/Enterprise_JavaBeans)

<sup>3</sup> H2: <http://www.h2database.com/html/main.html>

<sup>4</sup> Arnaud Thimel: Développeur à Code Lutin. Intervenant dans le module Objet Distribués du M2 ALMA.

## II. SCHEMA DES BASES DE DONNEES



### III. STEREOTYPES, ETIQUETTES ET SEMANTIQUE

---

L'introduction d'un nouveau stéréotype permet d'étendre la notation standard d'UML. Il se notera <<Nom du stéréotype>>. Une étiquette est une paire (nom, valeur) qui ajoute une nouvelle propriété à un élément de modélisation (ex : auteur, version, ...).

#### III.1. STEREOTYPES DE BASE

---

→ <<artifact>>

Un artifact correspond à un fichier (ou un ensemble de fichiers) nécessaire au déploiement d'une application.

→ <<component>>

Un component correspond à un composant architectural.

→ <<executionEnvironment>>

Un executionEnvironment correspond à un environnement d'exécution pour une application ou une base de données (JVM, Oracle, JBoss...).

→ <<singleton>>

Une classe typée par ce stéréotype devra posséder une instance unique au sein de l'application.

#### III.2. STEREOTYPES DE PERSISTANCE

---

Nous définissons certains stéréotypes qui seront propres à l'utilisation d'une technologie particulière.

→ <<Entity>>

Applicable à une classe qui devra faire l'objet d'un traitement particulier par Hibernate afin d'assurer sa persistance.

→ <<POA>>

Applicable à une classe qui permettra le transfert de données via la technologie CORBA.

#### III.3. STEREOTYPES DE BASES DE DONNEES

---

→ <<Table>>

Applicable à une classe représentant une relation de la base de données

→ <<PK>>



Applicable à un attribut d'une classe, il désignera l'attribut représentant la clé primaire. La classe doit nécessairement être stéréotypée par <<Table>>.

→ <<FK>>

Applicable à un attribut d'une classe, il désignera l'attribut représentant une clé étrangère. La classe doit nécessairement être stéréotypée par <<Table>>.

### III.4. STEREOTYPES DE DESIGN PATTERN

---

- <<Singleton>>

Applicable à une classe, il permettra d'implémenter le design pattern singleton. Ainsi un attribut static représentant l'unique instance de la classe sera ajouté, ainsi que les méthodes nécessaires.

## IV. REGLES DE TRADUCTION DE UML VERS LE CODE SOURCE

---

### IV.1. REGLES GENERALES

---

Le langage UML est une représentation graphique de l'application qui ne possède pas de sémantique propre. Il nous faut donc spécifier un certain nombre de règles pour traduire UML vers du code source. Ces conventions devront être respectées par l'ensemble des développeurs travaillant sur le projet. Nous obtiendrons ainsi un code cohérent dans sa forme et dans sa mise en œuvre. Notons que le langage qui sera utilisé pour le code source de ce projet sera Java. Celui-ci pose certaines contraintes dans la traduction que nous préciserons en temps voulu.

UML permet de définir plusieurs propriétés concernant un attribut :

- La visibilité : private, public, protected, package
- Les droits : readOnly, ...
- Sa valeur : valeur par défaut, valeur calculée
- Son type : String, Integer, String[1..5]

Nous devons être capable de conserver et traduire l'ensemble de ces propriétés dans le code source, sans pour autant être obligé de les retranscrire telles quelles. Ainsi, un attribut décrit comme public dans la représentation UML ne le sera pas nécessaire dans le code source du moment que nous conservons la possibilité d'accéder à l'attribut à partir de n'importe quelle classe extérieure.

Les attributs déclarés dans le code source devront tous être privés, c'est à dire être accessibles de manière directe uniquement au sein de leur propre classe. Pour assurer la visibilité décrite dans la représentation UML, nous utiliserons des getters/setters. Ce sont les visibilités de ces méthodes qui assureront la conservation de cette propriété.

Les traductions de types, de valeur seront réalisées de manière classique, en utilisant la syntaxe Java adaptée.

Nous utiliserons les règles vues en cours pour traduire les classificateurs (datatype, classe, énumération), les associations (composition, association simple, classe association). Nous n'avons pas cherché à réaliser de règle de traduction de l'héritage multiple, en effet le diagramme de classe que nous avons réalisé n'en comporte pas.

Les standards de codage Java que nous allons utiliser sont quasiment similaire à ceux définis dans la convention SUN. Pour que l'ensemble des développeurs respectent ces standards, nous utilisons l'outil Checkstyle. Ainsi, avant toute validation du code, il faudra vérifier au préalable qu'il respecte bien le fichier Checkstyle fourni.

Afin de profiter du travail de conception détaillée, nous avons eu recours à la génération de code. En effet, nous pouvons déduire de notre diagramme de classe de niveau conception de nombreuses portions du code source de l'application, notamment en ce qui concerne la génération des classes nécessaires à la persistance.

Nous avons d'abord réalisé un script *générique* de génération de classe Java. Parmi les éléments générés, il y a

- ✎ La déclaration d'une classe, en considérant les informations sur la classe étendue et les interfaces réalisées. La visibilité, son package, le fait qu'elle soit abstraite ou concrète est également pris en compte.
- ✎ Les attributs *internes* de la classe, ainsi que les attributs créés à partir d'associations dans le diagramme de classe. Il peut s'agir de collections.
- ✎ Un constructeur sans paramètres, un constructeur avec tous les attributs en paramètres, des getters et des setters dont les visibilités dépendent de la visibilité de l'attribut définie dans le modèle
- ✎ Les signatures des méthodes déclarées dans le modèles et celles nécessaires à la réalisation des interfaces.
- ✎ Des ébauches de commentaire de documentation au format Javadoc

Nous avons ensuite pris appui sur ce travail pour élaborer les autres scripts nécessaires : interfaces, DAO, implémentations de DAO.

Toutefois, nous n'avons pas atteint un degré de maîtrise suffisant du langage de template Acceleo afin de générer quelques portions de code qui auraient pu l'être :

- ✎ Les sections *import* du code Java. Il aurait fallu pour ça pouvoir accéder à un ensemble (sans redondance) des dépendances d'une classe, ce qui n'est pas immédiatement disponible. Nous aurions également parcourir chaque attribut et chaque paramètre de méthode pour ajouter la dépendance qu'il induit à la section import, mais cela pouvait générer des redondances et il nous a semblé trop complexe de les gérer seulement avec le langage Acceleo . Remarquons aussi qu'Acceleo permet de faire appel à du code Java (via la création de *services*), nous aurions pu essayer de faire appel au composant d'Eclipse qui permet de résoudre ce problème (la fonction « Organize Imports » de l'EDI).

- ⌘ Les fichiers de configuration pour JPA/Hibernate. Ces fichiers pourraient être intégralement générés étant donné qu'on peut déduire du modèle le schéma de la base de données. On aurait aussi bien pu générer les annotations JPA pour les ajouter aux classes du modèle.

Nous avons également été ennuyés par les limitations de l'application, notamment :

- ⌘ La difficulté de créer de nouveaux stéréotypes dans TopCased (bien qu'il soit aisé d'appliquer à des éléments du diagramme des stéréotypes existants). Cela nous aurait permis de créer des stéréotypes identifiant les singletons, les états (pattern *state*), et les classes persistantes.

- ⌘ L'impossibilité de créer des classes génériques : l'outil semble le permettre (une propriété est proposée) mais à chaque fois que nous avons tenté de renseigner le champ convenablement, nos informations n'étaient pas sauvegardées et le modèle n'était pas impacté. Nous avons tenté de contourner cette limitation en utilisant un héritage standard mais il en a résulté une complication intense au niveau du code généré.

Malgré cela, nous avons pu développer des scripts qui se sont avérés très efficaces pour générer le code, notamment celui du modèle. En particulier, ces scripts nous ont permis, au cours de la phase finale de programmation, de pouvoir réviser notre modèle et d'impacter immédiatement le code. Cela nous a permis d'accélérer la programmation de l'application en réduisant considérablement le travail nécessaire à un changement du modèle.

D'une façon générale, nous avons eu des difficultés pour appréhender le langage de template d'Acceleo. Il nous a semblé qu'il aurait été utilement complété par une API Java basée sur l'introspection. Ainsi, certaines portions de code auraient été générées par un code manipulant des collections Java (de méthodes, de paramètres...), et générant une chaîne de caractères attendue. Les services Acceleo notamment, pourraient permettre ce genre de création.

Cela dit, nous comprenons le choix des concepteurs d'Acceleo qui voulaient créer un langage permettant de générer du code dans n'importe-quel langage cible.

La manipulation des classes générées était aisée étant donné la possibilité de définir explicitement, dans les scripts de générations, les parties du code qui ont vocation à être écrites par le développeur. Le code ainsi écrit est conservé lors des régénérations du code. Cela permet d'éviter le recours à des patrons de conceptions (tel que Generation Gap) pour combler le manque qu'aurait provoqué l'absence de cette fonctionnalité.

## CONCLUSION

---

A l'issue de ce livrable, nous avons établies toutes les informations nécessaires pour réaliser le développement de l'application. En effet, nous sommes désormais aptes à partager le travail entre les différents membres de l'équipe grâce aux diagrammes de composants, nous connaissons les besoins de techniques de l'application, et nous avons spécifié les règles nécessaires à la traduction du diagramme de classes vers Java.

Nous devons maintenant construire le diagramme de classe de niveau conception afin de générer le code. En parallèle, l'IHM pourra être développée.



# Getting Things Done

Livrable 5 : Conception détaillée

**BRUN Clémentine, LE NY Brendan, PONGE Myrtille,  
RAVENET Brian  
06/01/2010**

## TABLE DES MATIERES

---

---

Introduction .....	1
I. Découpage du modèle en packages .....	2
II. Le package fr.alma.model.....	6
II.1. GestionnaireRessources.....	6
III. Le package fr.alma.controler .....	1
IV. Le package fr.alma.persistance.....	2
V. Retours sur le projet .....	4
V.1. Un projet multi-modules .....	4
V.2. Un projet en équipes .....	4
V.3. De nombreux livrables .....	5
Conclusion.....	7

## INTRODUCTION

---

Suite à la description de l'architecture de notre application GTD réalisée dans le livrable 4, nous sommes maintenant en mesure de proposer la conception détaillée de tous les composants du logiciel. Seul le composant concernant l'IHM ne sera pas décrit, en effet, il donnera lieu à un rapport distinct. Ce livrable a pour but de spécifier avec le maximum de détails l'architecture globale du projet (au niveau conception). Nous pourrons ainsi implémenter le système sans faire appel à la « magie ».

Nous justifierons tout d'abord le découpage en package pour lequel nous avons opté, avant de spécifier chacun de ces packages à l'aide de diagrammes UML. Tout au long de ce livrable, nous prendrons soin de dérouler le raisonnement et mettre en évidence les obstacles auxquels nous avons été confrontés et qui nous ont permis d'aboutir à notre conception finale.

Ce livrable étant le dernier du projet, vous trouverez également un chapitre concernant notre retour sur le projet.



## I. DECOUPAGE DU MODELE EN PACKAGES

L'objectif du pattern MVC<sup>1</sup> est de séparer une application en couches et ainsi de dissocier les différentes problématiques. Ainsi, une couche peut évoluer indépendamment des autres. Ce pattern facilite également le travail en équipe en offrant un découpage clair de l'application et ainsi aidant au partage des tâches.

Voici une représentation graphique du pattern MVC permettant de visualiser les interactions entre les différentes couches.

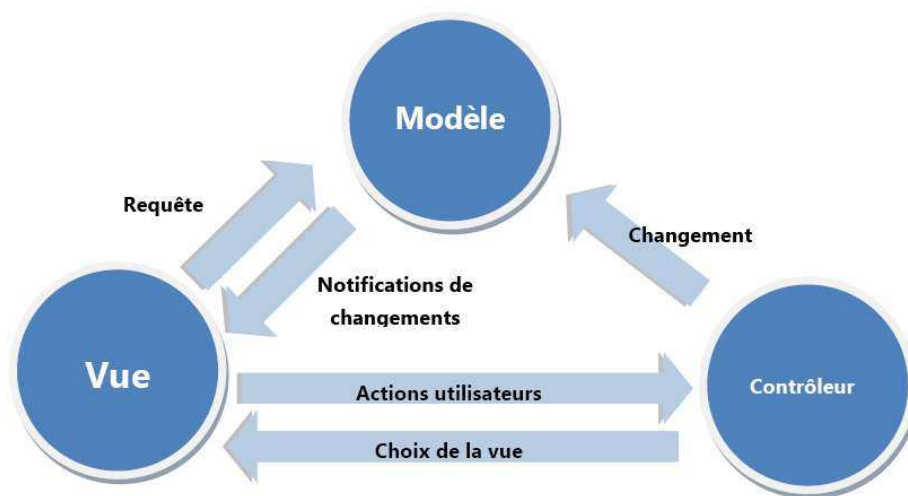


FIGURE 1: REPRESENTATION DU DESIGN PATTERN MVC<sup>2</sup>

En cohérence avec le schéma ci-dessus, l'utilisation de ce pattern a immédiatement donné lieu à la création de trois packages :

- **Fr.alma.view**. Il contiendra l'ensemble du code utile à la création de l'IHM. Ce composant ne sera pas étudié ici, en effet les diagrammes de classes ne sont pas toujours adaptés à la description des composants d'IHM. Cette couche donnera lieu à un rapport distinct.
- **Fr.alma.model**. Il représente les données de l'application et définit l'implémentation des opérations disponibles pour agir sur ces données. Il contient le **GestionnaireUtilisateur** et le **GestionnaireRessource**.
- **Fr.alma.controler**. Il gère l'interface entre le modèle et le client. Il va interpréter la requête de ce dernier pour lui envoyer la vue correspondante. Il effectue la synchronisation entre le modèle et les vues. C'est au sein de ce package que seront traitées les requêtes liées à la connexion d'un utilisateur.

<sup>1</sup> MVC : Modèle Vue Contrôleur.

<sup>2</sup> <http://c-maneu.developpez.com/tutorial/web/php/symfony/intro/images/mvc.jpg>

Les données de l'application doivent être rendues persistantes, nous avons donc créé un quatrième package chargé de gérer les différents modes de persistance fr.alma.persistence.

En réalisant les premières ébauches du diagramme de classe de niveau conception, on s'aperçoit que notre contrôleur devient de taille très conséquente, tandis que le modèle ne devient qu'une couche au-dessus de la persistance. Nous nous approchons donc plus d'une architecture 3-tiers que du pattern MVC. Notre contrôleur associé à notre modèle représenterait le tiers 2 (logique métier).

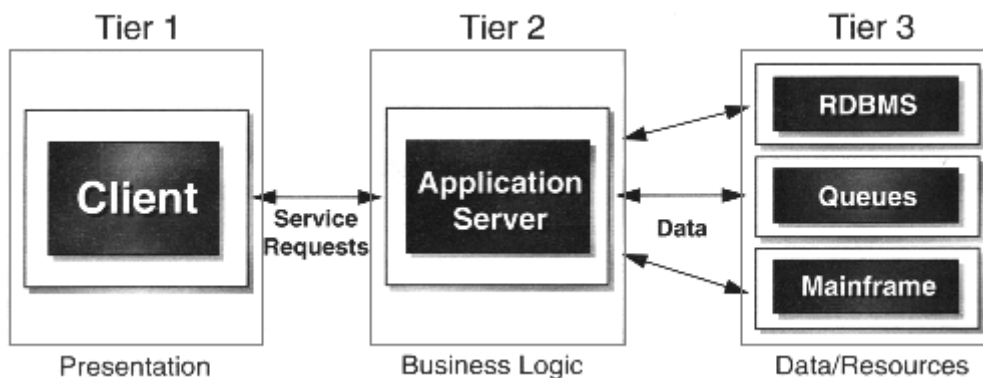


FIGURE 2 : REPRESENTATION DE L'ARCHITECTURE 3-TIERS

Pour réaliser MVC il faudrait mettre dans le package modèle notre modèle et notre contrôleur. Le nouveau contrôleur n'ayant qu'une fonction de filtre sur les données (champs non nul, ...). Nous n'optons pas pour cette solution, car cela ajoute beaucoup de délégations, qui nous paraissent superflues au vue des traitements à effectuer, qui ne sont finalement que des getters et setters.

Nous souhaitons masquer l'implémentation des fonctionnalités qu'offre chaque package aux autres. Nous utiliserons pour cela le design pattern façade qui fournit une interface unique pour accéder au sous système.

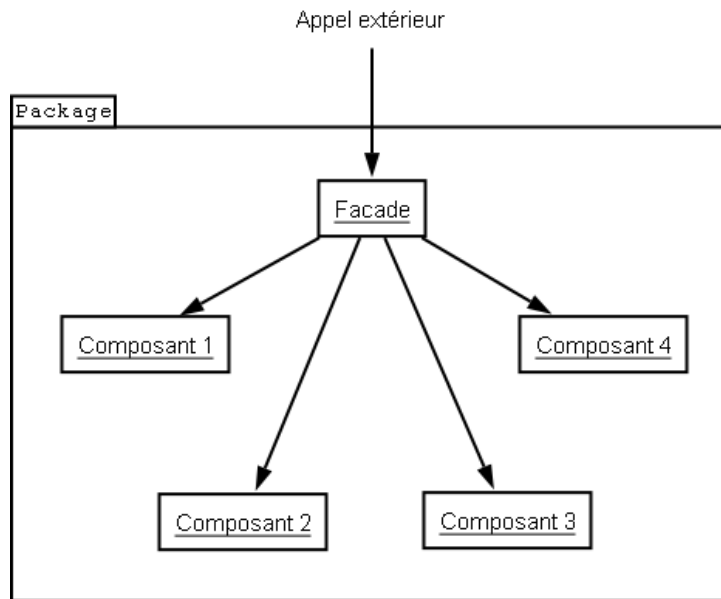


FIGURE 3 : REPRESENTATION DU DESIGN PATTERN FACADE<sup>3</sup>

Les classes implémentant l'interface Facade devront respecter le design pattern singleton. En effet, elles ne devront pas être instanciées plusieurs fois au cours du cycle de vie de l'application.

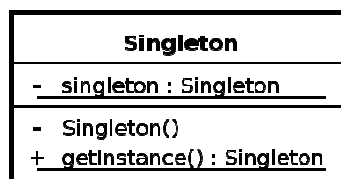
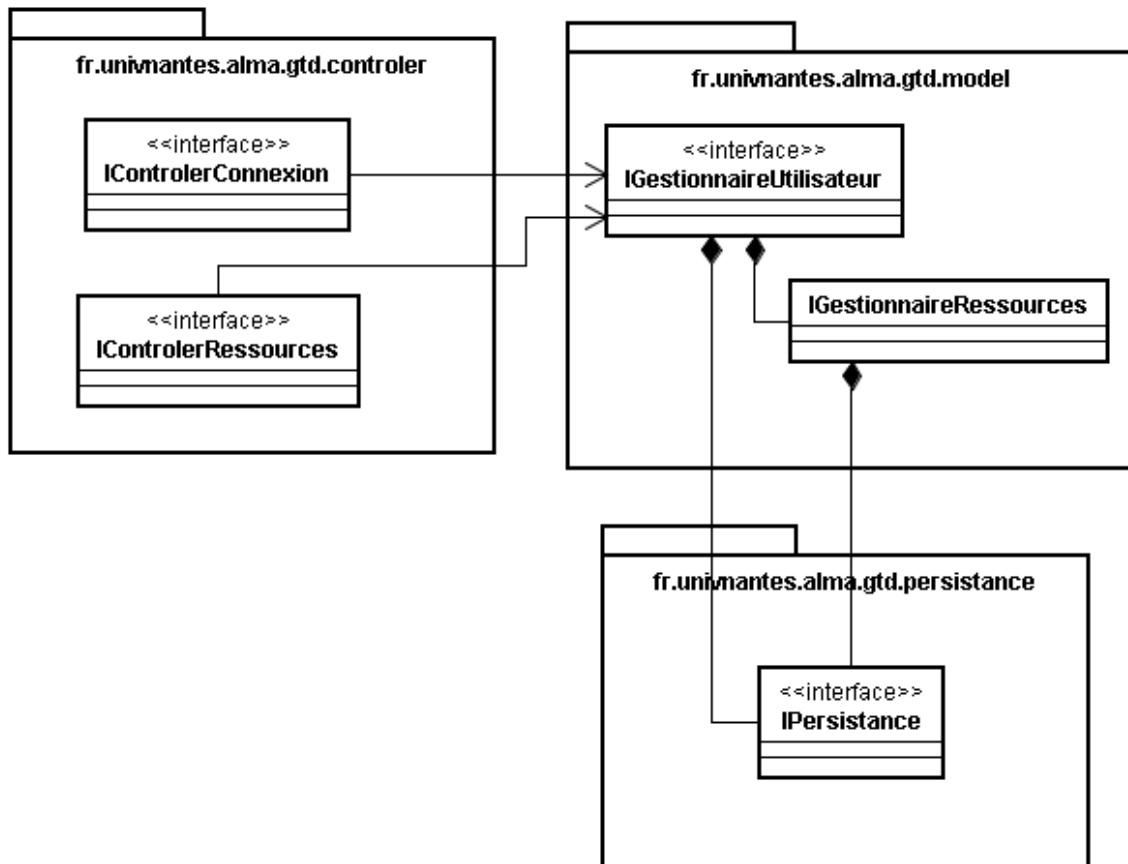


FIGURE 4 : REPRESENTATION DU DESIGN PATTERN SINGLETON<sup>4</sup>

<sup>3</sup> <http://pcaboche.developpez.com/article/design-patterns/programmation-modulaire/images/Facade.png>

<sup>4</sup> [http://en.wikipedia.org/wiki/File:Singleton\\_UML\\_class\\_diagram.svg](http://en.wikipedia.org/wiki/File:Singleton_UML_class_diagram.svg)

Voici les interactions entre chaque composant :



## II. LE PACKAGE FR.ALMA.MODEL

Initialement, nous avons prévu un stéréotype « Entity » permettant à une classe que ses instances devront être rendues persistantes, elle devra donc posséder un identifiant. Malheureusement, nous ne sommes pas parvenus à gérer les stéréotypes dans les scripts Acceleo de génération de code. Nous avons donc choisi de créer une classe abstraite GTDEntity qui contiendra comme unique attribut un identifiant et dont héritera toute classe dont les instances devront être rendues persistantes. Cette solution nous offre finalement l'avantage supplémentaire d'avoir une classe générique représentant toute classe persistante.

### II.1. GESTIONNAIRERESSOURCES

Ce package permettra la gestion des projets de l'utilisateur, cela comprends ses tâches, ses contextes, ses contacts. Voici les classes représentant les données à faire persister hériteront de GTDEntity (Idee, Projet, Tache, Contact et Contexte).

Nous avons tout d'abord imaginé introduire le design pattern Composite sur la classe Projet. En effet, un projet pourra être composé de sous-projet.

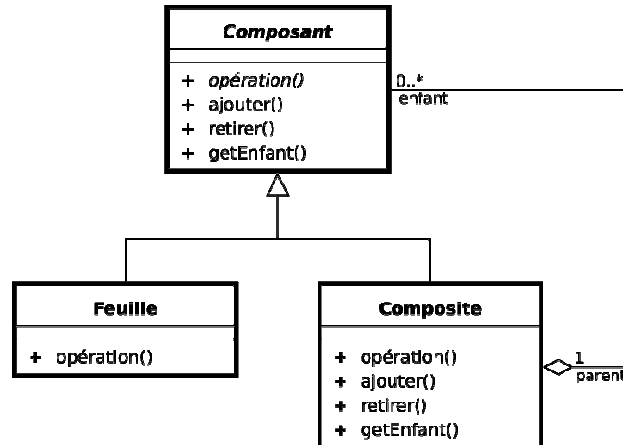


FIGURE 5 : REPRESENTATION DU DESIGN PATTERN COMPOSITE<sup>5</sup>

Finalement nous nous sommes aperçus que **ProjetFeuille** ne serait jamais utilisé. Tout projet est supposé un jour être amené à être composé de projet. Nous ne pouvons donc nous permettre d'introduire une classe limitant la composition. Nous n'avons donc plus que finalement la classe **Projet** qui possède une composition sur elle-même.

<sup>5</sup> [http://fr.wikipedia.org/wiki/Fichier:Composite\\_UML\\_class\\_diagram\\_fr.svg](http://fr.wikipedia.org/wiki/Fichier:Composite_UML_class_diagram_fr.svg)

Le pattern State est utilisé pour représenter les différents états d'une tâche : Delegee, AFaire, EnAttente et Finie. Le pattern permet d'offrir des implémentations différentes d'une même méthode en fonction de l'état de l'objet. Ainsi, demander de passer une tâche dans l'état en attente, alors qu'elle est finie provoquera une erreur, alors que si elle est actuellement dans l'état à faire cela ne posera aucun problème.

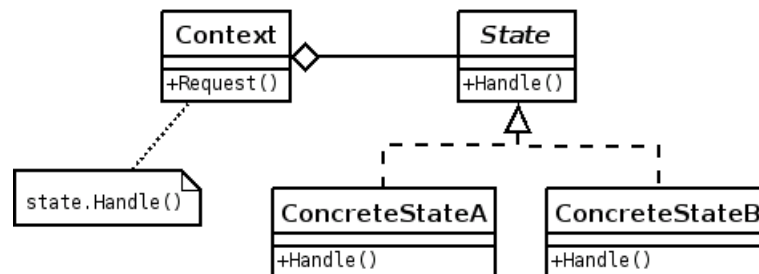
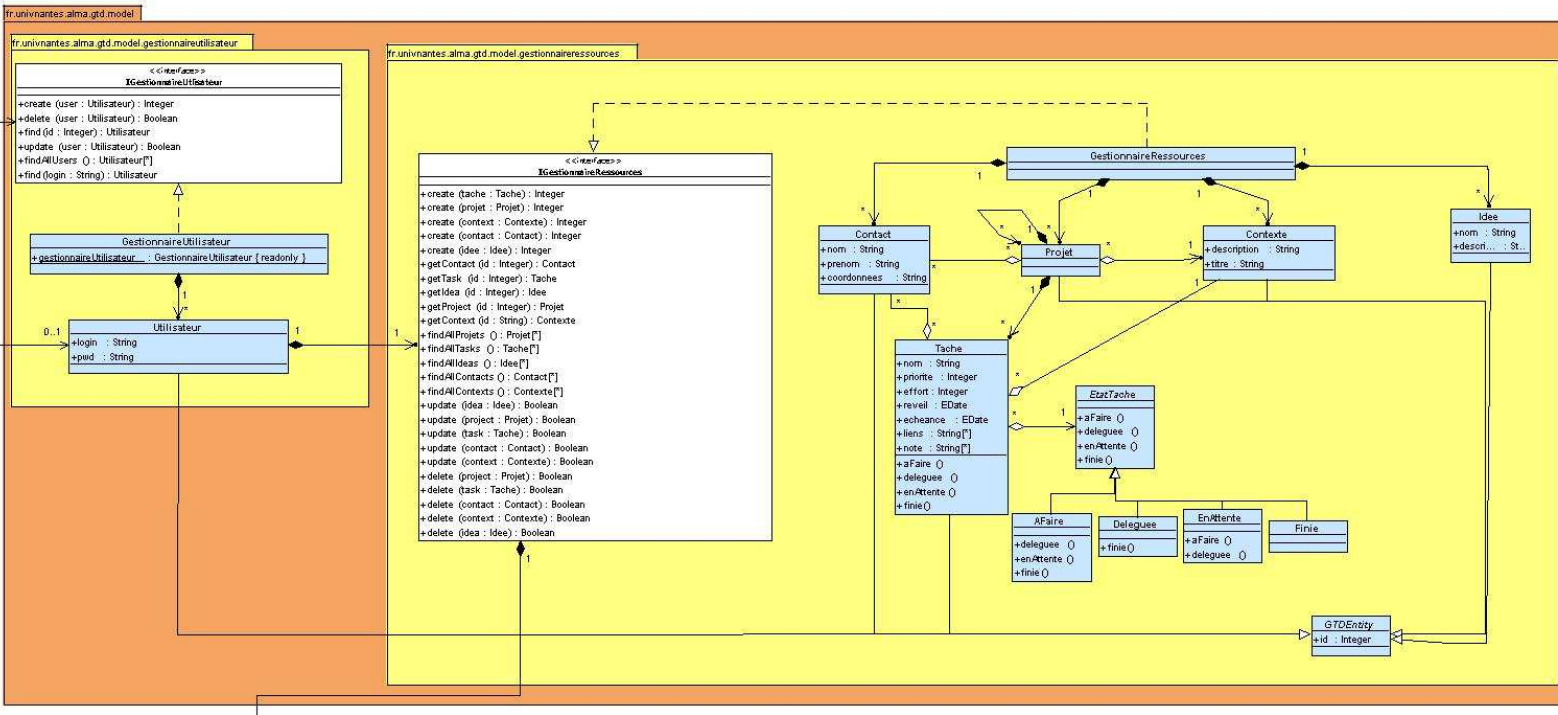


FIGURE 6 : REPRESENTATION DU PATTERN STATE

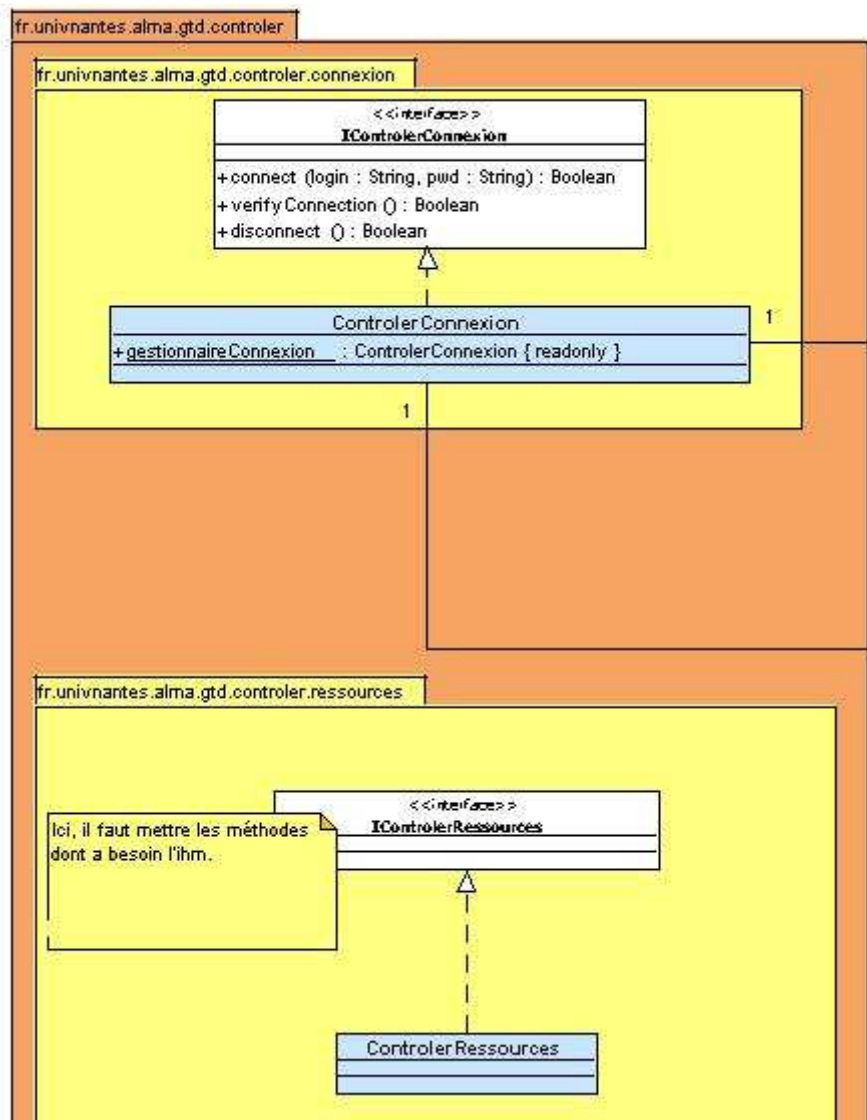
Voici finalement le diagramme complet du modèle :



### III. LE PACKAGE FR.ALMA.CONTROLER

Ce package sera chargé de faire parvenir des données cohérentes au modèle en provenance de l'utilisateur. Il sera également chargé de faire toutes les modifications utiles afin de transmettre correctement des données du modèle à l'utilisateur.

Il contient également le gestionnaire de connexion qui est chargé de suivre le cycle de vie de la connexion d'un utilisateur. C'est ici que pourrait se trouver un timeout de connexion automatique de l'utilisateur, par exemple. En passant toujours par le contrôleur et donc pas le gestionnaire de connexion, nous garantissons que le système ne fournira des données qu'à une personne authentifiée sur le système, et ne lui offrira l'accès qu'à ses données propres.





## IV. LE PACKAGE FR.ALMA.PERSISTENCE

---

Ce package contient l'implémentation des différents modes de persistances : locale ou distante. Chacun des modes est contenu dans un sous package, et implémente l'interface IPersistence. Ainsi, le passage d'une persistance à une autre ou encore la création de nouveaux modes se trouve facilité.

Les persistances locales et distantes font le sujet d'un rapport distinct. Nous ne les présenterons donc pas ici.

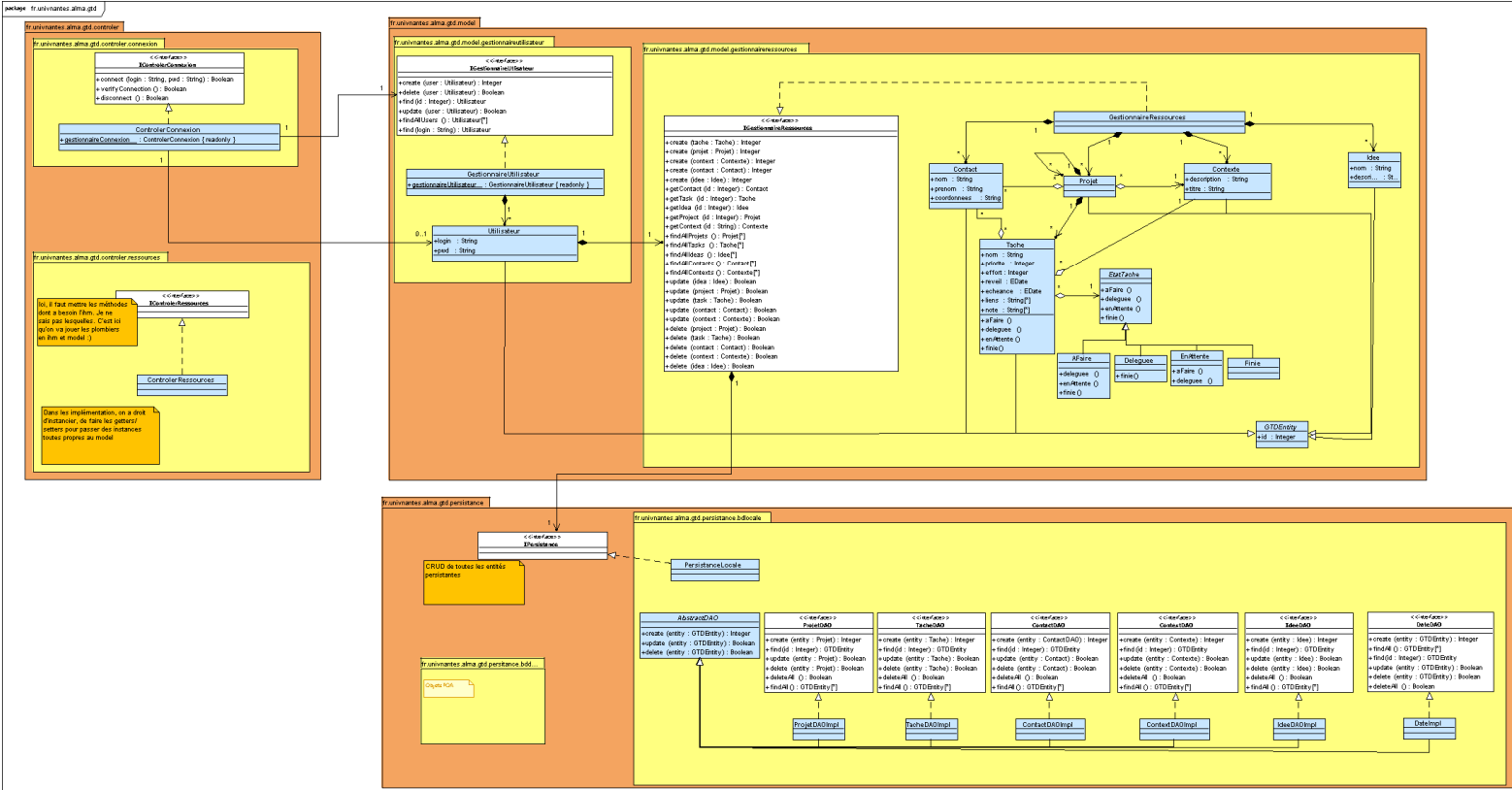


FIGURE 7: DIAGRAMME DE CLASSE COMPLET

## V. RETOURS SUR LE PROJET

---

### V.1. UN PROJET MULTI-MODULES

---

La description de la deuxième année du master ALMA met en avant l'existence de ce projet commun à plusieurs modules. En effet, tout au long de notre cursus universitaire, nous avons eu à réaliser des projets pour chaque module, mais jamais ils avaient été regroupés pour une unique réalisation. En deuxième année, les notions abordées dans les modules sont complémentaires. On ne peut pas construire d'application sans savoir gérer correctement la persistance, sans connaître et maîtriser un certain nombre de designs patterns. Et quel utilisateur voudrait d'un logiciel sans interface graphique. Les trois modules d'objets distribués, de génie logiciel et d'IHM sont donc bien complémentaires pour nous apporter le savoir-faire nécessaire à la réalisation d'applications.

Cet aspect du projet nous a énormément plu et motivé. Le sujet proposé s'adaptait parfaitement à la mise en commun de ces trois modules. Nous avons donc pu mettre en œuvre les différentes technologies vues en cours.

### V.2. UN PROJET EN EQUIPES

---

Le deuxième aspect innovant de ce projet est le travail en équipe. Habituellement, les projets sont réalisés en binôme, les interactions et les conflits sont donc relativement limités. Dans le cas de ce projet multi-module, nous avons dû tout d'abord travailler par équipe de quatre étudiants pour réaliser notre partie de l'application, mais nous avons également dû collaborer avec un autre groupe de quatre étudiants réalisant la seconde partie de l'application. Ce dernier groupe collaborait avec l'ensemble des étudiants de la promotion. Ce qui donne finalement un projet quasiment commun à toute la promotion.

Ce second aspect du projet nous motivait fortement puisqu'il nous amenait à nous confronter à des problématiques nouvelles de travail en groupe. C'est également celui qui nous a le plus handicapé tout au long de la réalisation du projet. En effet, à aucun moment il n'a été décidé d'une ligne de conduite de projet.

Le travail en équipe de quatre personnes s'est, pour notre part, bien déroulé. Nous avons su partager le travail efficacement, tout en maintenant l'ensemble des participants au courant de l'avancée des travaux de chacun. Cela a été possible grâce à la tenue régulière de « réunions de

travail » et à l'utilisation d'outils collaboratifs : SVN, documents partagés sur le Web, messagerie instantanée, mails.

C'est la collaboration entre les équipes qui a posé de nombreux problèmes. Tout d'abord l'échéancier établi obligeait les équipes développant les clients et celle développant le serveur à avancer en même temps. Ainsi, chaque équipe a analysé le problème à sa façon, écrit ses propres spécifications des besoins. C'est uniquement à la fin de la spécification des composants que les équipes ont commencé à communiquer. On s'est alors rendu compte de divergences profondes sur la manière d'appréhender le sujet. Par exemple l'équipe développant le serveur a vu GTD comme une méthode de gestion de projets réalisés en équipe, alors que nous y avons vu une méthode de gestion personnelle. Nous sommes donc face à deux visions incompatibles. Les seules solutions qui s'offraient à nous étaient :

- ☞ L'une des deux équipes abandonnait sa vision et recommençait le travail d'analyse et de spécification depuis le début : inconcevable !
- ☞ Nous développions notre propre serveur : cela ne répond pas au sujet !
- ☞ Nous essayions de faire rentrer coûte que coûte notre modèle dans le modèle du serveur : après un temps infini à retourner notre projet dans tous les sens, cela nous a semblé impossible sans perdre des données de l'utilisateur.

Nous avons donc du opter pour la seule solution réalisable mais non satisfaisante : réaliser notre application GTD sans la persistance distante, et montrer nos connaissances sur ce sujet et réalisant une petite application de test.

Nous avons donc bien pu constater ici, le besoin fondamental de l'existence d'une méthode de gestion de projet. Idéalement, il aurait certainement fallu nommer un chef d'équipe dans chaque groupe de quatre étudiants, ainsi qu'un chef de projet permettant de garantir une cohérence entre l'avancement du travail de tous les groupes.

### V.3. DE NOMBREUX LIVRABLES

---

Le dernier aspect innovant de ce projet était sa construction de bout en bout par les étudiants. Nous avons ainsi réalisé toutes les étapes du projet, de l'analyse du problème, jusqu'à la génération du code source. De nombreux livrables ont été réalisés pour jalonner les différents stades d'avancement. Nous avons utilisé dans chacun d'eux de nombreux diagrammes UML de différents types.

L'ensemble de ces livrables forment un dossier d'analyse et de conception relativement important. Sa taille nous semble quelque peu disproportionnée par rapport à la complexité du

projet. De nombreux diagrammes sont répétitifs entre plusieurs livrables. Nous n'avons pas toujours su différencier les représentations pertinentes des autres.

Nous avons parfois un peu perdu de vue la finalité du projet (la réalisation simple d'une application GTD) et nous nous sommes parfois égarés dans des détails inutiles. En d'autres termes, nous n'avons pas toujours été « smart » !

Plusieurs membres de notre équipe ont d'ores et déjà réalisés des stages en entreprise et ont donc pu suivre le déroulement d'un projet en conditions réelles. Bien souvent, les équipes en entreprise se contentent d'un dossier de spécification des besoins, accompagnés d'un diagramme de classes ainsi que d'une représentation des données. L'analyse du problème se trouve généralement être le cahier des charges. Les diagrammes de cas d'utilisation sont également couramment utilisés. Mais nous avons peu souvent croisé de diagramme de séquence ou d'activité, ...

Les livrables réalisés en entreprise sont certes souvent trop simplistes et ignorent donc des problèmes qui seront rencontrés ultérieurement dans le projet et qui causeront des retard de livraison. D'un autre côté, nous avons souvent eu l'impression de perdre du temps dans la réalisation des livrables de notre projet. Il faudrait donc être capable de trouver un compromis entre les deux pratiques.

## CONCLUSION

---

Ce livrable est le dernier du projet multi-module. La suite du projet sera l'implémentation du système à l'aide des règles établies dans le livrable 4 et la conception détaillée décrite ici. Certaines modifications surviendront encore certainement au cours de l'implémentation, mais nous garderons soin de toujours conserver une totale cohérence entre notre conception détaillée et le code source de l'application. L'implémentation sera réalisée à l'aide des outils de génération de code proposés par la société Obeo.

Ce livrable clos donc le projet multi-module.



# Getting Things Done

Objets distribués

Le projet GTD d'un point de vue objets distribués

**BRUN Clémentine, LE NY Brendan, PONGE Myrtille,  
RAVENET Brian**

**02/02/2010**

## TABLE DES MATIERES

---

---

Introduction .....	3
a. Communication avec le serveur .....	4
a. Qu'est-ce que CORBA ? .....	4
b. Mise en œuvre ? .....	5
b. La persistance locale .....	7
a. Le choix des technologies .....	7
b. La mise en œuvre .....	9
Conclusion .....	12



## INTRODUCTION

---

Au cours de la deuxième année du master ALMA, nous devons réaliser un projet concernant plusieurs modules distincts. Le travail à réaliser consiste en la construction d'une application outillant la méthode GTD<sup>1</sup>. Le module Génie Logiciel concerne principalement les phases d'analyse et de conception de l'application, le module d'IHM sera noté sur un critère purement graphique, enfin le module d'Objets Distribués permettra l'étude et la mise en place de communication distante ainsi que de la persistance des données. Ce sont ces points que nous aborderons dans ce document.

Dans un premier temps nous nous intéresserons à la partie communication distante avec le serveur, puis nous verrons les moyens mis en œuvre afin d'assurer une persistance locale des données.

---

<sup>1</sup> GTD = Getting Things Done, méthode de gestion de tâches créées par David Allen.

## A. COMMUNICATION AVEC LE SERVEUR

Nous étions chargés de développer une application cliente qui communique avec un serveur distant. Il nous était imposé d'utiliser la technologie CORBA afin de réaliser ce pont entre le client et le serveur.

Dans un premier temps nous étudierons ce qu'est CORBA et ce qu'il permet, ensuite nous pourrions expliquer sa mise en œuvre au sein de notre projet GTD.

### A. QU'EST-CE QUE CORBA ?

CORBA signifie "Common Object Request Broker Architecture". C'est une architecture logicielle pour le développement d'ORB (Object Request Broker). Ces derniers sont des intergiciels permettant la mise en place d'un bus logiciel par lequel les objets pourront envoyer et recevoir des requêtes de manière transparente.

CORBA permet donc de faire communiquer des ordinateurs distants et différents (système d'exploitation, matériel, ...). Pour assurer cette transparence, CORBA utilise des interfaces IDL (Interface Definition Language), dont l'implémentation pourra être générée dans n'importe quel langage souhaité (Java, C++, ...).

Voyons maintenant la séquence d'actions permettant d'envoyer un objet sur le serveur. Tout d'abord voici les différents éléments intervenant dans le traitement d'une requête.

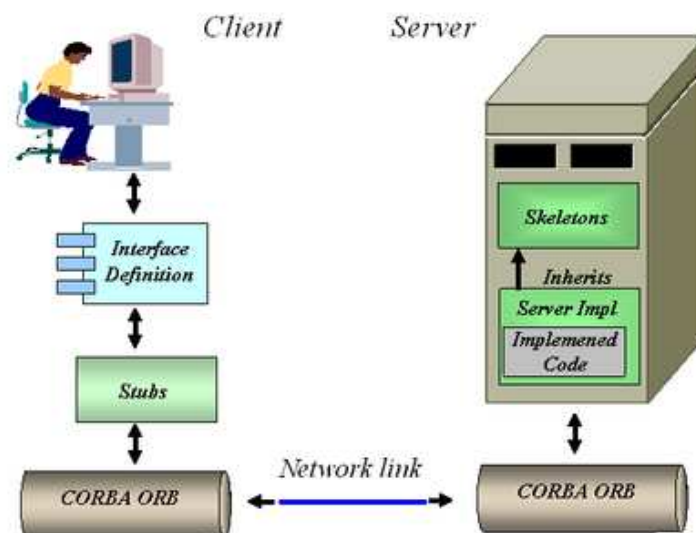


FIGURE 1: INFRASTRUCTURE DE CORBA<sup>2</sup>

<sup>2</sup> [http://upload.wikimedia.org/wikipedia/en/thumb/7/77/Corba\\_Server.png/400px-Corba\\_Server.png](http://upload.wikimedia.org/wikipedia/en/thumb/7/77/Corba_Server.png/400px-Corba_Server.png)

- Interface Definition : l'IDL permet de spécifier la partie visible d'un objet, ses méthodes et ses propriétés. L'interface et l'implémentation est ainsi séparée.
- Stubs : l'amorce cliente , créée grâce aux interfaces IDL.
- CORBA ORB : il masque la complexité des traitements du à l'hétérogénéité des réseaux.
- Skeletons : l'amorce serveur, générée grâce aux interfaces IDL.

Le client détient uniquement les références IOR sur les objets CORBA. Il possède également les interfaces IDL permettant de définir les interfaces des objets CORBA. Lorsque le client invoque une requête, celle-ci est acheminée vers l'objet CORBA par l'ORB CORBA. Ce dernier est chargé du pliage des arguments de la requête. Sur le serveur l'objet CORBA est associé à une implémentation sur laquelle l'opération peut-être réalisée. La réponse de la requête suit exactement le chemin inverse.

## B. MISE EN ŒUVRE ?

---

Notre équipe était chargée du développement d'une application GTD cliente. Nous n'étions pas responsables du développement du serveur. Nous avons alors rencontré de réelles difficultés à la mise en œuvre de CORBA au sein de l'application.

Notre équipe ainsi que celle développant le serveur ont réalisé dans le même temps les livrables d'analyse et de conception du projet. Les échéances fixées ne nous ont pas permis de trouver le moment pour nous mettre d'accord sur ces livrables, pourtant essentiels par la suite. C'est ainsi que de notre point de vue, la méthode GTD est une méthode de gestion personnelle des tâches, tandis que pour le serveur il est tout à fait acceptable de partager des tâches. C'est un exemple parmi d'autres, mais qui montre déjà la divergence de points de vue, et ce, dès l'analyse. Lors de la conception, cela ne s'est pas arrangé, certaines des entités que nous souhaitions voir persister sur le serveur n'existaient tout simplement pas, à l'inverse d'autres étaient « en trop ».

Nous avons envisagé plusieurs solutions afin de remédier à ces divergences :

- ? Nous considérons leurs spécifications comme les seules valides. Outre le fait d'avoir perdu énormément de temps sur les livrables déjà réalisés, cette solution remettait en cause l'ensemble de notre analyse. Nous devions donc redémarrer de zéro et refaire l'ensemble des livrables. Vu le temps imparti pour la réalisation de ce projet, cette solution ne nous a pas semblé viable.
- ? L'équipe réalisant le serveur considérait nos spécifications comme seules valides. Les mêmes arguments que précédemment peuvent alors s'appliquer. De plus, d'autres

équipes dépendaient du serveur, il était donc impensable de la part du serveur de modifier profondément ses spécifications.

- ? Réaliser de la « tuyauterie » afin de faire rentrer notre vision dans la leur. Au vu de la grande divergence d'opinion, cette solution nous a paru irréalisable. En effet, nous ne pouvons pas faire persister la notion de contact, si celle-ci n'existe pas sur le serveur.
- ? Réaliser notre propre serveur. Cette solution ne réponds pas au sujet, ni à nos exigences de délais. Nous l'avons donc très rapidement abandonnée.

Après étude des différentes solutions, nous ne sommes pas parvenus à en choisir une convenable. Nous avons donc décidé de montrer de manière théorique ce qui aurait été réalisé dans le cas idéal.

Nous avons conçu notre application de façon à pouvoir implémenter la persistance des données de multiples façons. Ainsi, l'intégration d'une solution de persistance sur un serveur distant via CORBA est aisée.

Cela revient principalement à implémenter l'interface IPersistence. La programmation de cette classe réalisant IPersistence nécessite de définir les différents éléments qui vont intervenir dans le système de persistance.

Nous devons d'abord définir un ensemble de classes représentant les différentes classes du modèle, dont les instances seraient des représentations des objets CORBA distants, celles-ci agiraient selon le Design Pattern *proxy*.

Les définitions de ces classes sont très similaires, elles doivent toutes contenir un IOR permettant de retrouver l'objet distant. Les opérations et leurs implémentations sont les mêmes, on peut donc factoriser tout cela dans une classe unique. Seul le type d'objet qu'on fait persister varie, on peut donc paramétrer cette classe par un type T. Étant donné que toutes les classes que nous voulons faire persister sont des spécialisations de GTDEntity, T doit être un sous-type de GTDEntity. En Java, les classes seront déclarées avec les signatures suivantes :

```
abstract class ObjetDistant<T extends GTDEntity>
class TacheImpl extends ObjetDistant<Tache> implements Tache
class ProjetImpl extends ObjetDistant<Projet> implements Projet
```

Afin de pouvoir dialoguer avec le serveur, il faut considérer le stockage de l'adresse du registre CORBA. Elle est commune à toutes les classes, on peut donc le définir dans un attribut de la classe implémentant IPersistence.

Nous pouvons ainsi assurer la représentation en locale d'objet distants qu'on peut sauvegarder puis récupérer. Il faut toutefois avoir recours à un mécanisme d'identification auprès du serveur. La session d'identification doit être initiée auprès du serveur dès le démarrage de l'application, cette connexion doit être maintenue afin de permettre toute instanciation tardive.

## B. LA PERSISTANCE LOCALE

---

En plus d'assurer une communication avec un serveur distant, notre application doit également être capable de mettre en œuvre une persistance locale des données. Cette fois-ci aucune contrainte technologique ne nous a été fixée. Nous étudierons donc dans un premier temps les choix que nous avons effectués, puis nous expliqueront leur mise en œuvre au sein de notre application.

### A. LE CHOIX DES TECHNOLOGIES

---

Nous étions libres quant au choix des technologies utilisées pour assurer la persistance locale des données de l'application. La communication avec le serveur distant étant réalisée avec CORBA, nous nous sommes tournés vers une tout autre technologie pour la persistance locale : Hibernate. Cela nous permettra de développer de nouvelles compétences autour de J2EE<sup>3</sup>. Ce framework gérant la persistance a les avantages d'être open source et adaptable d'un point de vue architectural. Hibernate peut être utilisé par annotations ou des fichiers de mappings XML. Nous avons déjà manipulé une notion s'approchant des annotations Hibernate : les EJB. Nous avons donc choisi de découvrir la seconde méthode encore très utilisée dans les entreprises.

---

<sup>3</sup> J2EE : Java Enterprise Edition.

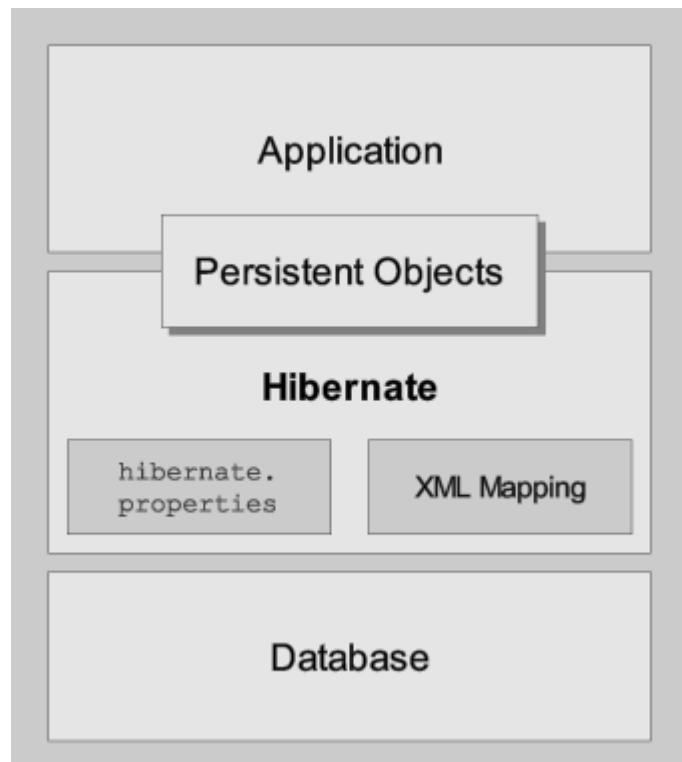


FIGURE 2: ARCHITECTURE D'HIBERNATE

Le framework permettant l'accès à la base de données étant défini, il nous a fallu ensuite choisir un SGBD<sup>4</sup>. Celui-ci devra répondre à plusieurs contraintes : supporter Hibernate et posséder un mode embarqué. Après étude de plusieurs SGBD (SQLite, ...), nous nous sommes tournés vers H2 qui semblait répondre à toutes nos exigences.

Il existe un design pattern qui est classiquement utilisé dans les architectures de persistance utilisant Hibernate. Il est basé sur un pattern Factory :

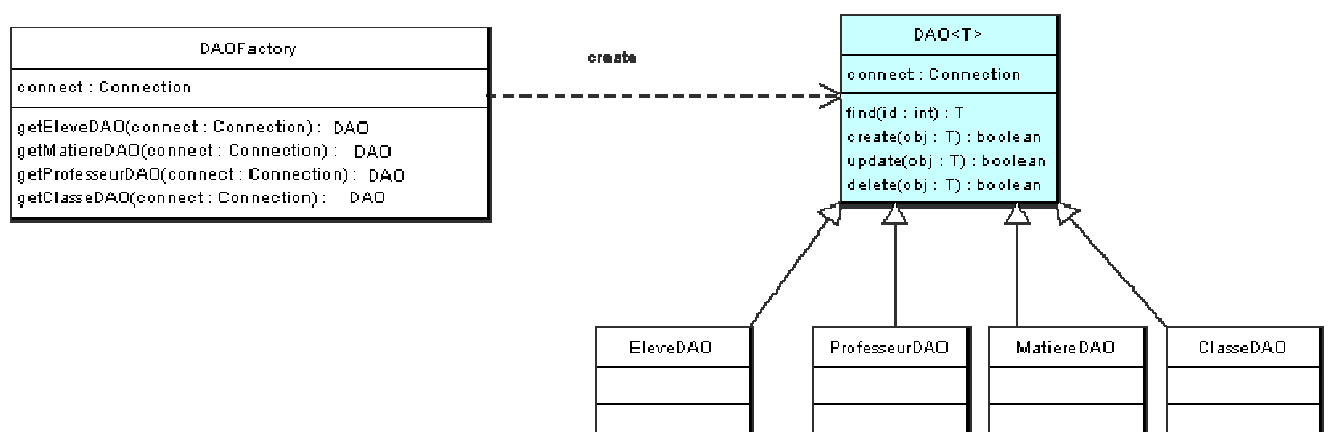


FIGURE 3: PATTERN DAO

<sup>4</sup> SGBD : System de Gestion de Bases de Données.

## B. LA MISE EN ŒUVRE

Le SGBD H2 ne nécessite aucun déploiement, mis à part la configuration de la connexion, aucun traitement particulier le concernant n'est exigé. Sa mise en œuvre est donc réellement simple et efficace.

L'avantage de l'utilisation d'un framework tel qu'Hibernate est de conserver une indépendance entre le modèle et type de persistance choisie. Ainsi, si nous désirons changer de framework, ou de SGBD, cela n'a aucune incidence sur l'architecture et l'implémentation du modèle.

Nous avons vu précédemment qu'idéalement le pattern DAO doit être utilisé pour réaliser des architectures utilisant Hibernate. Malheureusement, nous ne sommes pas parvenus à créer des classes templâtées dans le logiciel de modélisation qui nous était imposé : TopCase. Voici donc l'architecture réelle que nous avons adoptée :

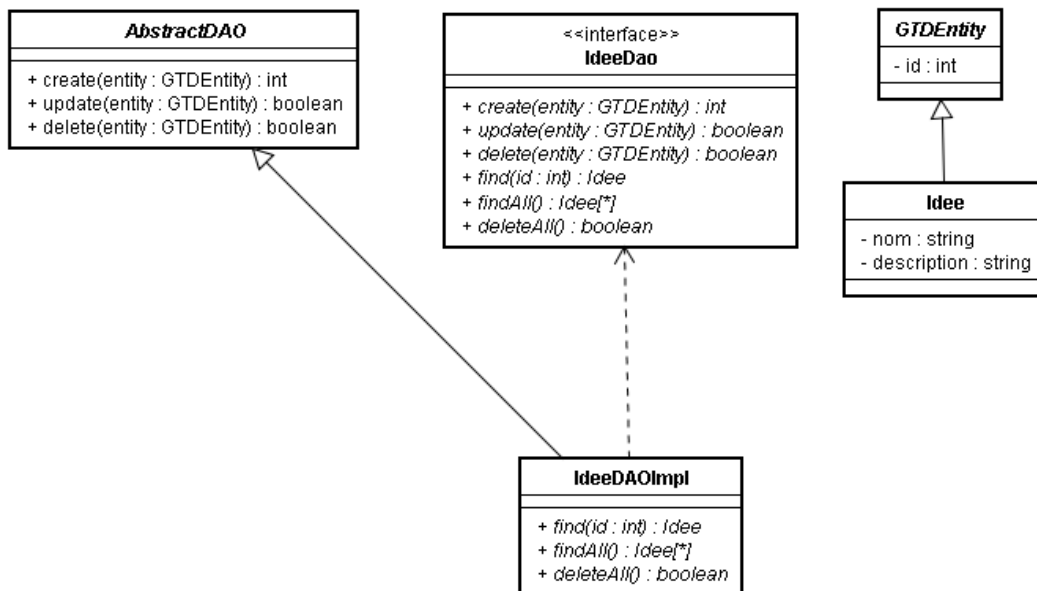


FIGURE 4: ARCHITECTURE REELE DE LA PERSISTENCE LOCALE

Voici maintenant le fichier de configuration d'Hibernate qui permet de lui renseigner tous les paramètres nécessaires à la connexion à la base de données.

### Hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-configuration (View Source for full doctype...)>
<hibernate-configuration>
  <session-factory>
    <property name="dialect">org.hibernate.dialect.H2Dialect</property>
```

```

<property name="show_sql">true</property>
<property name="generate_statistics">true</property>
<property name="hbm2ddl.auto">update</property>
<property name="jdbc.batch_size">1</property>
<property name="connection.driver_class">org.h2.Driver</property>
<property name="connection.url">jdbc:h2:~/Essai</property>
<property name="connection.username">sa</property>
<property name="connection.password" />
</session-factory>
</hibernate-configuration>

```

Une classe nommée HibernateUtil permet de faire toutes les opérations nécessaires à la gestion des sessions. Etudions maintenant les manipulations à effectuer pour rendre une classe persistante. Pour cela nous prendrons l'exemple de Idee.java.

- ✓ Faire étendre la classe de la classe abstraite GTDEntity.java Nous aurons ainsi immédiatement un identifiant et ses accesseurs. De plus, les méthodes de persistance pourront utiliser ce type générique.
- ✓ Vérifier la présence de tous les getters/setters.
- ✓ Créer une interface IdeeDAO.java, définissant les signatures des méthodes nécessaires à la persistance.

```

public interface IdeeDAO{
    public Integer create(final GTDEntity entity);
    public Idee find(final Integer id);
    public Boolean update(final GTDEntity entity);
    public Boolean delete(final GTDEntity entity);
    public Boolean deleteAll();
    public List<Idee> findAll();
}

```

- ✓ Créer une classe IdeeDAOImpl implémentant cette interface et étendant AbstractDAO
- ✓ Créer le fichier de mapping XML d'Hibernate, Idee.hbm.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping auto-import="true"
package="fr.univnantes.alma.gtd.model.gestionnaireressources">
    <class name="fr.univnantes.alma.gtd.model.gestionnaireressources.Idee" >
        <id name="id" column="ideeId" type="java.lang.Integer">
            <generator class="increment" />
        </id>
        <property name="nom" type="java.lang.String"/>
        <property name="description" type="java.lang.String"/>
    </class>
</hibernate-mapping>

```

En rouge nous retrouvons le mapping de l'identifiant, nous indiquons que celui-ci doit être généré par incrément. En bleu nous pouvons voir la définition de l'attribut nom. L'attribut description est défini de la même manière en vert.



- ✓ Enfin, indiquer à Hibernate que le fichier de mapping XML existe, en ajoutant la ligne suivant dans le fichier de configuration :

```
<mapping resource="hibernate/entities/Idee.hbm.xml" />
```

Nous avons parfois été confrontés à des obstacles inattendus. Voici quelques exemples de ces problèmes, ainsi que les solutions mises en œuvre quand elles ont été trouvées:

- 👉 Lors d'une relation one-to-one entre deux entité (à chaque entité de la classe A correspond exactement une et une seule entité de la classe B), les identifiants de chaque entités doivent être identiques.

→ Les identifiants ne doivent pas être générés de manière incrémental mais par « foreign key », exemple :

```
<id name="id" column="utilisateurId" type="java.lang.Integer">
  <generator class="foreign">
    <param name="property">iGestionnaireRessources</param>
  </generator>
</id>
```

- 👉 Nous avons également éprouvé quelques difficultés à rendre persistantes les relations d'héritage.

→ Nous avons choisi de créer une table par hiérarchie d'héritage. Voici le fichier de mapping permettant d'indiquer à Hibernate la manière dont gérer l'héritage :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping auto-import="true"
package="fr.univnantes.alma.gtd.model.gestionnaireressources">
  <class name="fr.univnantes.alma.gtd.model.gestionnaireressources.EtatTache">
    <id name="id" column="etatTacheId" type="java.lang.Integer">
      <generator class="increment"/>
    </id>
    <discriminator column="type" type="string"/>
    <subclass name="fr.univnantes.alma.gtd.model.gestionnaireressources.AFaire">
    </subclass>
    <subclass name="fr.univnantes.alma.gtd.model.gestionnaireressources.EnAttente">
    </subclass>
    <subclass name="fr.univnantes.alma.gtd.model.gestionnaireressources.Deleguee">
    </subclass>
    <subclass name="fr.univnantes.alma.gtd.model.gestionnaireressources.Finie">
    </subclass>
  </class>
</hibernate-mapping>
```

Ici, l'interface EtatTache est implémentée par quatre classes : AFaire, EnAttente, Deleguee et Finie. Le type de chacune des classes sera enregistré dans la colonne « type ».

## CONCLUSION

---

Suite à des obstacles d'ordre organisationnel, nous n'avons malheureusement pas pu mettre en œuvre la persistance distante au travers de Corba. Nous avons pourtant eu un premier aperçu de cette technologie lors des séances de TP, il nous paraissait fort intéressant de la mettre en œuvre dans une application de plus grande envergure, avec des problématiques telles que la persistance du pattern Etat, des relations d'héritages, ...

Malgré quelques erreurs encore présentes dans notre application, la persistance locale fonctionne relativement correctement. Ce fut pour nous l'occasion de mettre en œuvre cette technologie couramment utilisée en entreprise.