12, rue de la Houssinière
44322 Nantes

Projet multi-modules
Getting Things Done

Génie logiciel à objets 2
Objets répartis
IHM

Louis-Alexandre Celton
Gaetan Hervouet
Damien Levin
Paul Vaillant
2009-2010

Master 2 - ALMA

Table des matières

Première partie

Livrable 1

Chapitre 1

Introduction

Dans le cadre du projet multi-modules en master 2 ALMA, il nous a été demandé de modéliser et concevoir un logiciel permettant de planifier nos tâches au quotidien. La méthode Getting Things Done développée par David Allen se présente sous la forme d'une méthode de gestion des priorités quotidiennes. Cette première partie présente l'analyse de cette méthode. Nous nous interressons dans cette partie à l'analyse de la discipline GTD : ce premier livrable ne concerne donc pas l'application fournissant les fonctionnalités de GTD, mais bien la méthode GTD elle même. Le langage UML (Unified Modeling Language) permet de présenter cette analyse à l'aide des documents suivants :

- Un dictionnaire de données qui permettra d'identifier et de détailler l'ensemble des concepts identifiés,
- un cas d'utilisation général ainsi qu'un sous ensemble de cas modélisés à l'aide d'une description textuelle selon le canevas de CockBurn,
- des diagrammes de scénarios afin d'illustrer les interactions entre l'utilisateur et le système,
- des instantanés illustrant l'impact des cas d'utilisation sur les objets,
- un diagramme d'activités représentant le fonctionnement général de la méthode,
- un diagramme de classes (niveau analyse),
- un ensemble de contraintes exprimées en OCL.

Chapitre 2

Dictionnaire de données

Action	**Définition**	**Traduction**	**Nom Informatique**
Collecte d'informations	Établir une liste de faits qui justifie la réalisation d'une tâche	Opération	Data Collect
Traitement des informations	A partir des faits que l'on a listés, on établit une liste de tâches pertinentes à réaliser.	Opération	Data Process
Organisation des données	Organisation des données selon certains critères : Le contexte, l'environnement dans lequel on se situe, les outils disponibles Temps disponible, Le temps alloué avant la réalisation de la prochaine tâche Capacité physique, les ressources dont on dispose pour réaliser la tâche Priorité : Dans le cas où des tâches se confondent sur les critères précédents, on définit une hiérarchie entre elles pour savoir lesquelles sont les plus importantes	Opération	Data Organisation
Mise au point	Réactualisation de l'ensemble du système de façon périodique ou ponctuelle. Mise à jour du contexte, des listes de tâches, etc...	Opération	Review
Notion	**Définition**	**Traduction**	**Nom Informatique**
Tâche	Tâche, action à réaliser	classe	Task
Projet	Un projet est un conteneur de tâches et de sous projets	classe	Project
Contexte	L' environnement dans lequel la tâche doit être réalisée, ainsi que les outils nécessaires à sa réalisation	classe abstraite	Context

Chapitre 3

Cas d'utilisation

Nous avons procédé à l'analyse de la méthode GTD et modélisé de manière très générale les différents acteurs et actions du système. Le diagramme suivant sera décomposé dans la suite en 4 sous-diagrammes de cas d'utilisation :

- Cas d'utilisation : Collecter des données
- Cas d'utilisation : Traiter des données
- Cas d'utilisation : Organiser des données
- Cas d'utilisation : Réactuliser

La sélection des informations peut engendrer une création de tâches d'où la relation extend. Le cas d'utilisation Revue doit permettre de collecter de nouvelles informations. En effet, lorsque de nouvelles tâches sont traitées, de nouvelles informations peuvent en découler.

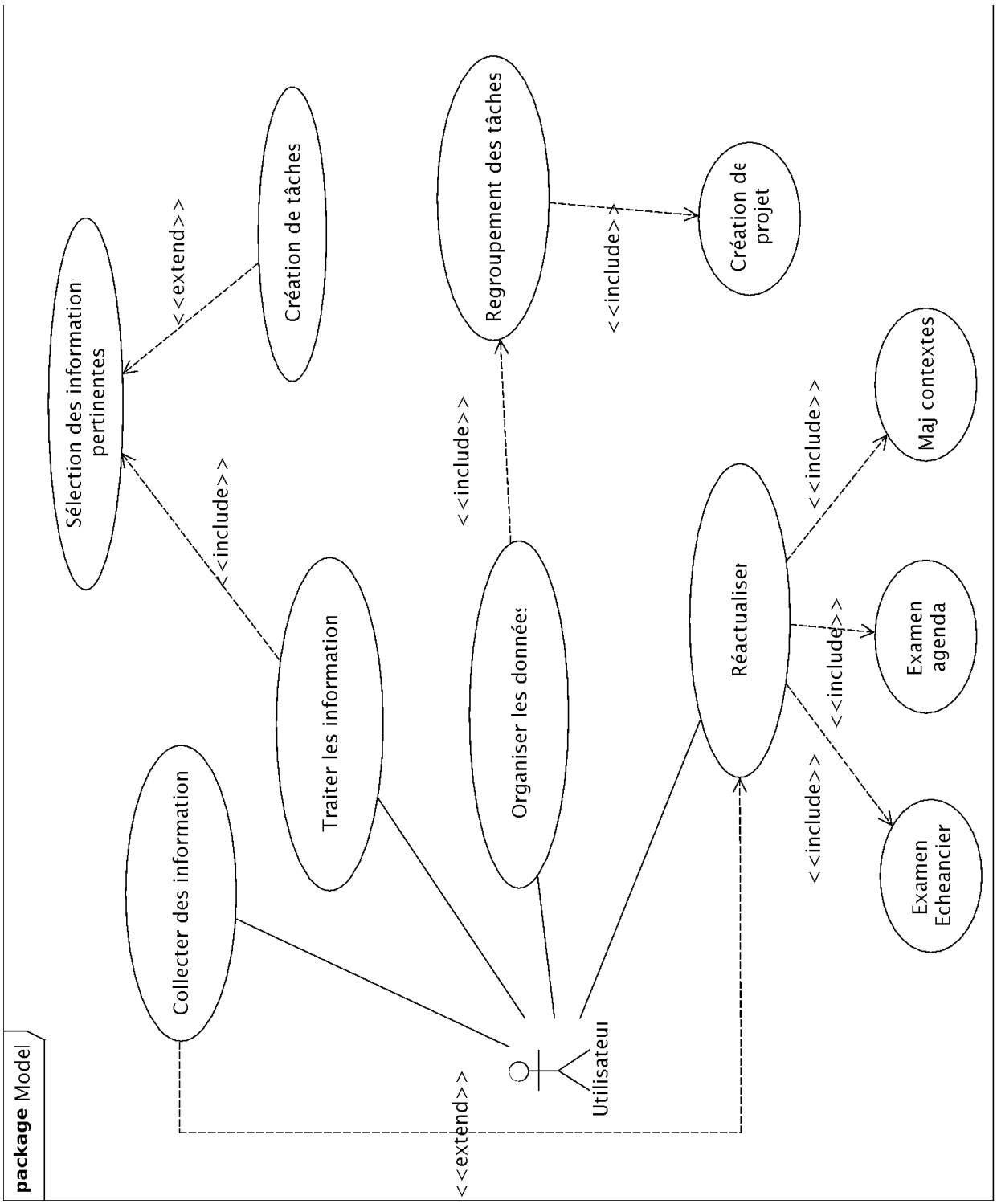

FIG. 3.1 – Diagramme UML de cas d'utilisation - Vision générale du système

3.1 Recensement des taches

Cockburn du recensement des taches

Cas d'utilisation	Collecter des informations		
Acteur	Objectif Utilisateur		
Parties prenantes et intérêts	Le recensement exhaustif de tout ce qui peut justifier une quelconque intervention de notre part : en suspens, inachèvement, en attente, intention, projet, manque, usure, mauvais fonctionnement, problème, insatisfaction, besoin, engagements à tenir,etc. Exemples : cette carte de visite restée dans une poche, cette facture dans la boîte à gants, cette demande reçue, ce dossier qui traîne sur le bureau, cette agrafeuse qui coince, cette course à faire, ce problème à résoudre, cette suggestion à tester, les messages de la boîte vocale, ce projet jamais réalisé, ce souci de santé, ce fauteuil qui grince, les performances de ce collaborateur, toutes ces choses en retard...		
Niveau	Utilisateur		
Portée	Système GTD		
Pré-conditions			
Post-conditions	Les informations sont enregistrées sur un support.		
Scénario nominal	**Etapes**	**Action**	
	1	Recenser tout ce qui justifie une intervention de l'utilisateur	
	2	Enregistrer les informations (postit, système d'information)	
Extensions	**Etapes**	**Condition**	**Action**
	*	A tout moment	Possibilité d'annuler l'action
Contraintes	**Type**	**Description**	
	Correction	Les informations ennoncées sont réelles et correct.	
Priorité	Elevée (5/5)		
Performance			
Fréquences	Tous les débuts de journée		

Scenario du recensement des taches

Le scénario du cas d'utilisation *Collect* est très simple : Il correspond à un travail que doit fournir l'utilisateur. A savoir le recensement des informations sur tout ce qui peut nécessiter une intervention de notre part. La première partie concerne donc le rencensement puis une saisie est effectuée sur le système (post-it ou système informatique).

Diagramme d'objet

Avant *le recensement des tâches*

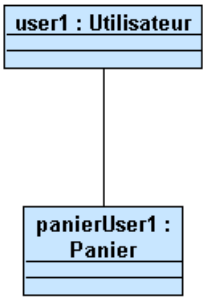

FIG. 3.2 – Diagramme d'objets UML - Avant *le recensement des tâches* le panier est vide

Ce diagramme d'objet représente un snapshot du système à un instant donné. On a ici deux instances d'Utilisateur et de Panier. En effet, le système est tout juste amorcé, l'utilisateur possède donc son panier mais il est vide.

Après *le recensement des tâches*

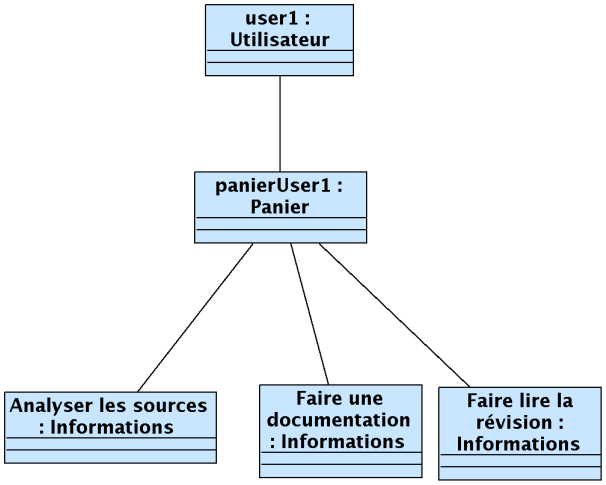

FIG. 3.3 – Diagramme d'objets UML - Après *le recensement des tâches*

Après *Collect*, on retrouve les informations (info1, info2 et info3) que l'utilisateur a saisi après le recensement, dans le système. Celles-ci sont de la forme "information" car elles ne donneront pas forcément lieu à des tâches. Le cas d'utilisation suivant (*Process*) explicite le traitement effectué entre "information" et "tâche".

3.2 Traitement des informations

Cockburn du traitement des informations

Le cas d'utilisation *Process* concerne le traitement des informations. Cette étape se place en seconde position dans l'ordre des étapes de la méthode GTD. La description textuelle selon le canevas CockBurn est présentée ci-dessous :

Cas d'utilisation	Traiter les informations	
Acteur	Utilisateur	
Parties prenantes et intérêts	Une fois la collecte des informations faîte, on détermine si des tâches pertinentes en résultent.	
Niveau	Objectif utilisateur	
Portée		
Pré-conditions	L'utilisateur a procédé à la collecte des informations	
Post-conditions	Les informations ont été traitées, de nouvelles tâches ont été créées si besoin	
Scénario nominal	**Etapes**	**Action**
	1	L'information nécessite une action
	2	La tâche ne peut pas être faite immédiatement
	3	Création d'une tâche
Contraintes	**Type**	**Description**
Priorité	Très élevée (5/5)	
Performance		
Fréquences	Dès que l'utilisateur collecte de nouvelles informations, il doit les traiter.	

Scenario du traitement des informations

Ce scénario complète le cas d'utilisation décrit précédemment. Il présente les interactions entre l'utilisateur et le système dans le cas où l'utilisateur a recueilli toutes les informations susceptibles d'engendrer des actions.

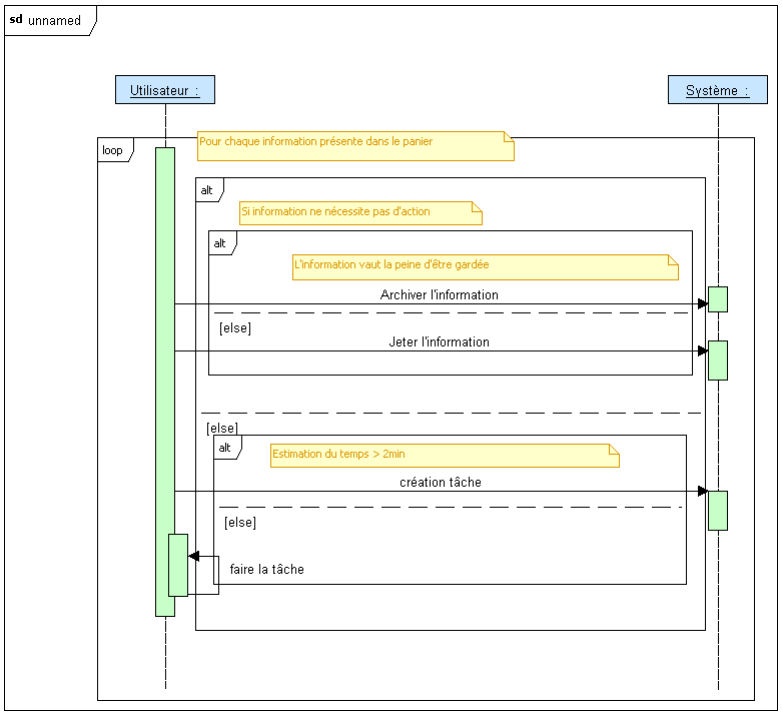

FIG. 3.4 – Diagramme UML - Scénario traitement des données

Diagramme d'objets

Afin d'observer l'impact du cas d'utilisation Traitement des informations sur les objets, voici deux diagrammes d'objets représentant leurs états avant et après le traitement.

Avant le traitement des données

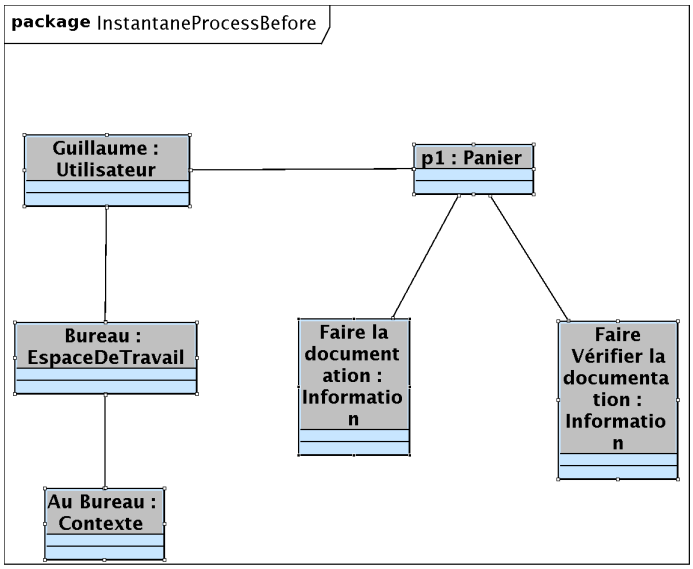

FIG. 3.5 – Diagramme d'objets UML - Avant le traitement des informations

L'utilisateur a procédé à la collecte des différentes informations nécessitant une intervention de l'utilisateur, elles peuvent toutes potentiellement devenir des tâches. Elles sont associées au panier de l'utilisateur. Ce dernier se trouve dans un espace de travail particulier nommé Bureau.

Après le traitement des données

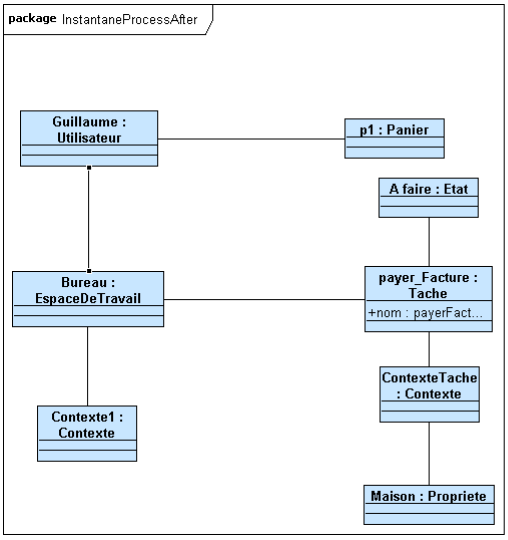

FIG. 3.6 – Diagramme d'objets UML - Après le traitement des informations

L'utilisateur procède au traitement des données : celu-ci consiste à définir si une information nécessite de devenir une tâche à effectuer. Dans le cas présent, l'information 1 entraîne la création de la tâche Payer_facture. Cette dernière est forcément associée à un contexte.

3.3 Organisation des données

Le cas d'utilisation organisation des données constitue la troisième étape de la méthode GTD. Elle à pour but d'organiser les tâches suivant différents paramètres :

- Le contexte de l'utilisateur
- La priorité des tâches
- Le temps disponible pour les tâches
- L'énergie disponible à la réalisation des tâches

Cockburn de l' organisation des données

Cas d'utilisation	Organiser les données		
Acteur	Utilisateur		
Parties prenantes et intérêts	A partir des taches existantes (qui résulte du traitement des données), on les organise au sein d'un projet et par rapport à un contexte donné		
Niveau	Utilisateur		
Portée	Système GTD		
Pré-conditions	On dispose d"une liste de taches résultants de l'étape traitement des données.		
Post-conditions	Les taches sont organisées au sein de projets		
Scénario nominal	**Etapes**	**Action**	
	1	Identification des tâches qui contribuent à la réalisation d'un même travail.	
	2	Regroupement des taches au niveau de projets	
	3	Les tâches sont organisées en fonction du contexte, du temps disponible, de la capacité physique (énergie) et de la priorité	
Extensions	**Etapes**	**Condition**	**Action**
	2.a	Une tâche est à réaliser sans dépendances	On ne créer pas de projet pour celle-ci.
	3.a	Une tâche ne peut pas être réalisée à un instant (t), compte tenu du contexte	On lui attribue une période de réalisation appropriée pour son contexte
Priorité	Très élevée (5/5)		
Performance			
Fréquences	De façon obligatoire après chaque traitement d'information.		

Comme on peut le voir l'organisation se divise en deux grandes étapes :

- Le regroupement des tâches participant à un même travail
- L'organisation des tâches en fonction de leurs contextes, des priorités.

Diagramme d'objets de l' organisation des données

Vision du système avant l'étape d'oganisation

Sur le diagramme suivant, les tâches ne sont pas reliées entre elles. Elles sont indépendantes et disposent d'un état et d'un contexte. L'espace de travail est également lié à un contexte, cette relation n'a cependant pas la même signification que pour une tâche : en effet, le contexte par défaut d'un projet regroupe les propriétés que devront (au moins) posséder les tâches affectées à ce même projet. L'exemple présenté ci-dessous illustre parfaitement le fait qu'un espace de travail regroupe des tâches de contextes différents.

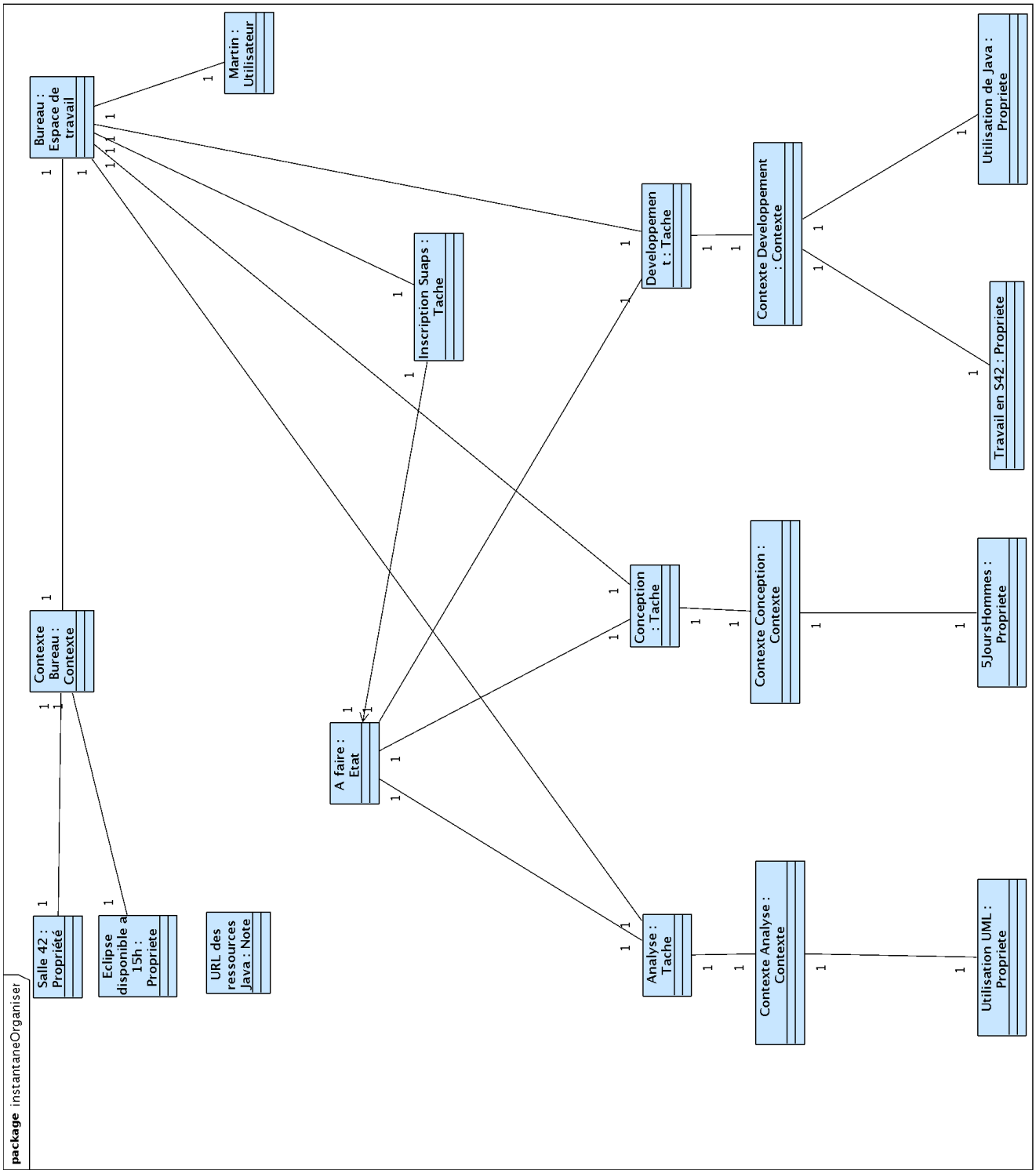

FIG. 3.7 – Diagramme UML Objet - Instantanné avant l'étape organisation

Vision du système après l'étape d'oganisation

Sur le schéma suivant, une étape d'organisation à été effectuée. Ainsi, les tâches sont donc reliées entre elles grâce à la relation *tâche précédente*. De plus, certaines tâches correspondent à la réalisation d'un même travail. C'est pourquoi elles sont contenues dans un projet, ici Projet GLO. On observe également que par rapport au schéma précédent, les tâches sont reliées à l'espace de travail à travers un objet vue. La vue *Agenda* ordonne les tâches selon des critères de temps, se limitant aux tâches courantes et enfin selon le contexte actuel : En effet, dans toutes les *Vues* comme *Agenda* ou *Echéancier*, le contexte courant (au moment où la demande est faite) est un critère de sélection des tâches à ordonner dans des *Vues* . Ici par exemple, le contexte de la tâche *inscription suaps* ne correspond pas au contexte définit dans la vue, c'est pourquoi la tâche n'est pas associée à cette vue.

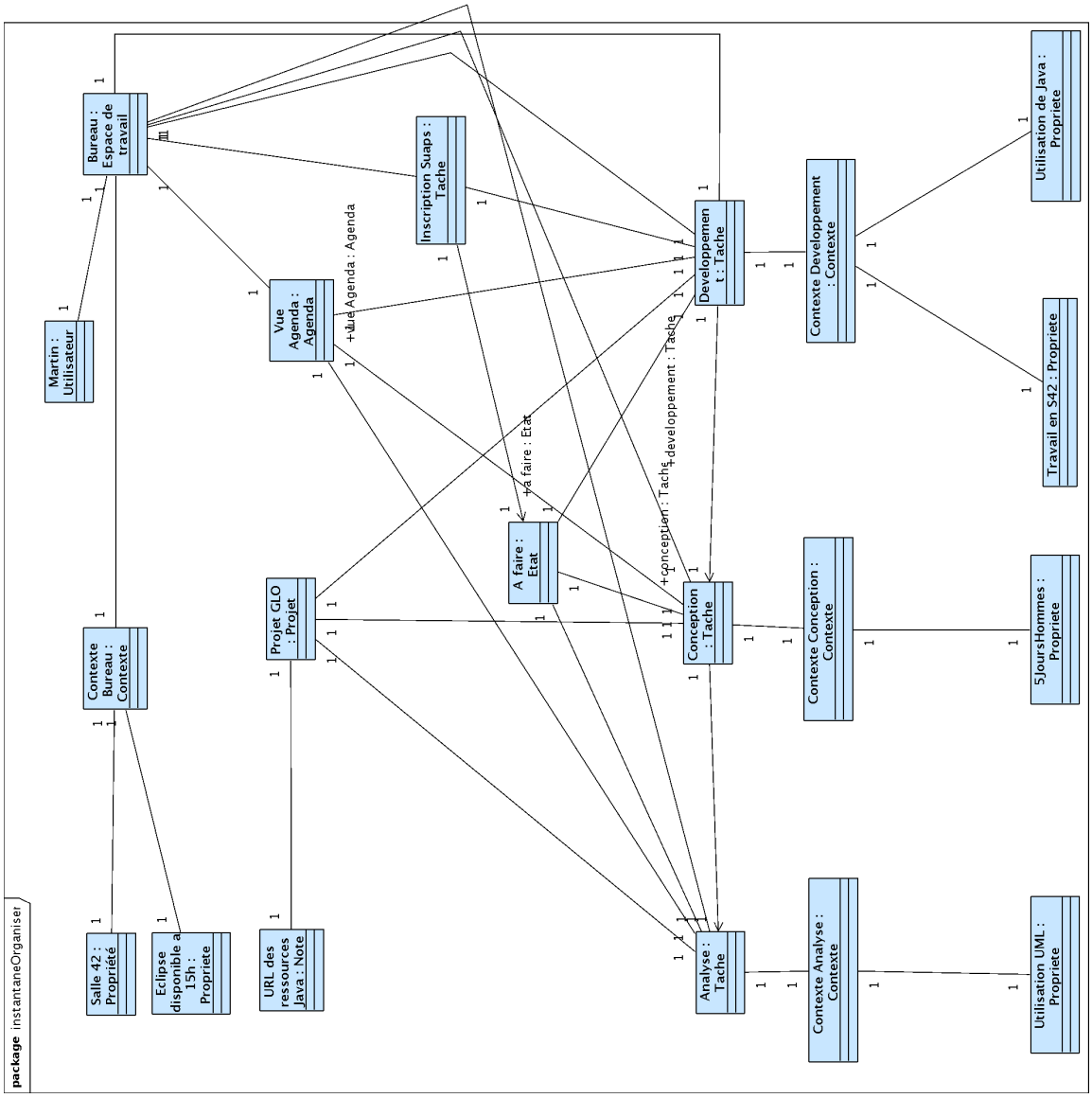

FIG. 3.8 – Diagramme UML Objet - Instantanné après l'étape organisation

3.4 Réactualisation du système

Cockburn de la réactualisation du système

Cas d'utilisation	**Réactualiser**		
Acteur	Objectif Utilisateur		
Parties prenantes et intérêts	Réactualisation régulière et systématique du système : mise à jour des fiches contexte, et ajout des nouveaux éléments au fur et à mesure (au moins une fois par jour), examen du calendrier à venir, des sujets en cours et en réserve, de l'échéancier, mise à jour des listes et dossiers, prospective, nettoyage de l'inutile(au minimum hebdomadaire)		
Niveau	Utilisateur		
Portée	Système GTD		
Pré-conditions			
Post-conditions	Les informations sont mises à jour dans le système.		
Scénario nominal	**Etapes**	**Action**	
	1	Mise à jour des contextes	
	2	Ajout des nouvelles informations (postit, système d'information)	
	2	Examen de l'échéancier et de l'agenda	
	2	Suppression des tâches effectuées, des ressources plus utilisées...	
Extensions	**Etapes**	**Condition**	**Action**
	*	A tout moment	L'utilisateur peut quitter l'action en cours
Contraintes	**Type**	**Description**	
	Correction	Les informations ennoncées sont réelles et correctes	
Priorité	Très Elevée (5/5)		
Performance			
Fréquences	Au moins une fois par jour		

Scenario de la réactualisation du système

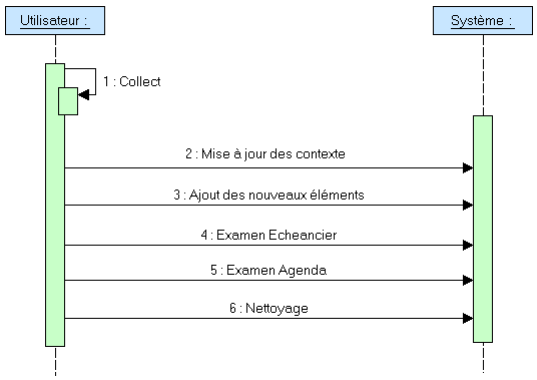

FIG. 3.9 – Diagramme UML - Scénario *Revue*

Le scénario du cas d'utilisation *Revue* utilise celui de Collect. En effet, la revue est un examen, un point sur là où nous sommes au moment présent. On effectue donc de nouveau un collect pour ajouter de nouveaux élements s'il y a lieu puis on passe aux étapes de revue à proprement parler :

- L'utilisateur mets a jour le contexte des tâches existantes s'il a changé,
- il ajoute les nouveaux élements,
- il examine l'écheancier,
- il examine l'agenda,
- enfin, il "nettoie" le système des tâches déja effectuées, des ressources qui ne seront plus utilisées..

Diagramme d'objets

Avant *la réactualisation du système*

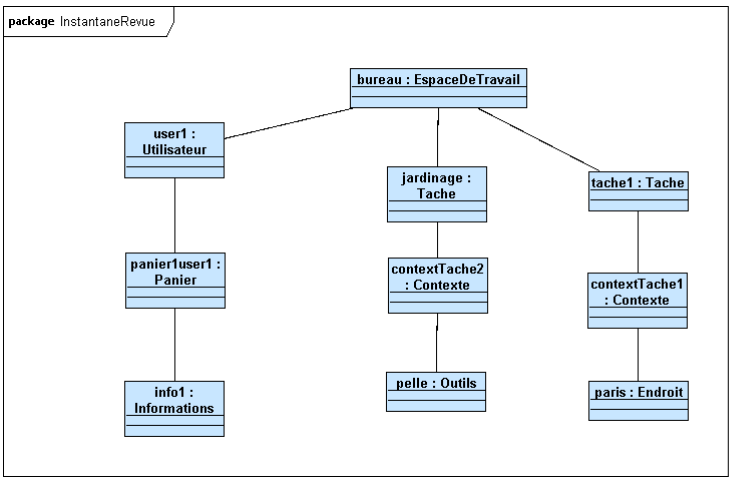

FIG. 3.10 – Diagramme d'objets UML - Avant *Revue*

On a ici le snapshot d'un système en cours d'execution. En effet, l'utilisateur possède un EspaceDeTravail (bureau) qui possède deux tâches préalablement enregistrées (lors du *Process*) :

- jardinage dont le contexte possède la propriété qui spécifie l'outil : une pelle.
- tache1 dont le contexte possède la propriété qui spécifie l'endroit : paris.

Après *la réactualisation du système*

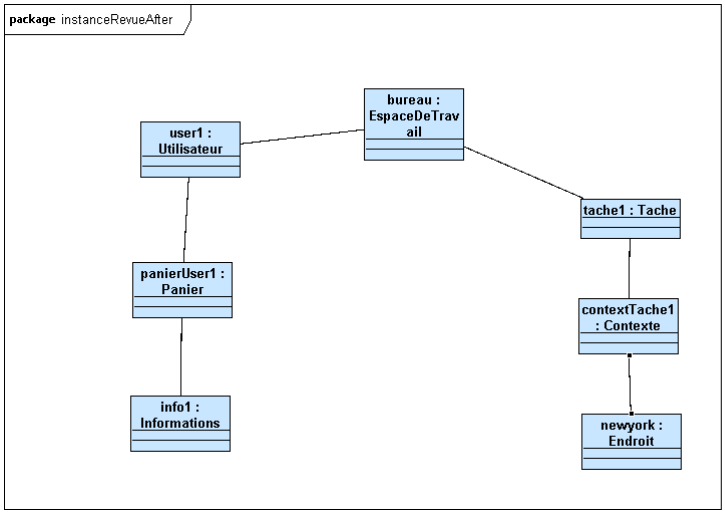

FIG. 3.11 – Diagramme d'objets UML - Après *Revue*

Apres *Revue*, on admet que le lieu du shopping a changé suite à l'annulation d'un ami. Ainsi, après la revue le lieu a été mis à jour, il passe de paris à newyork. De plus, la tâche jardinage ayant été réalisée entre cette revue et la précédente, nous l'avons supprimée.

Chapitre 4

Diagramme d'activités

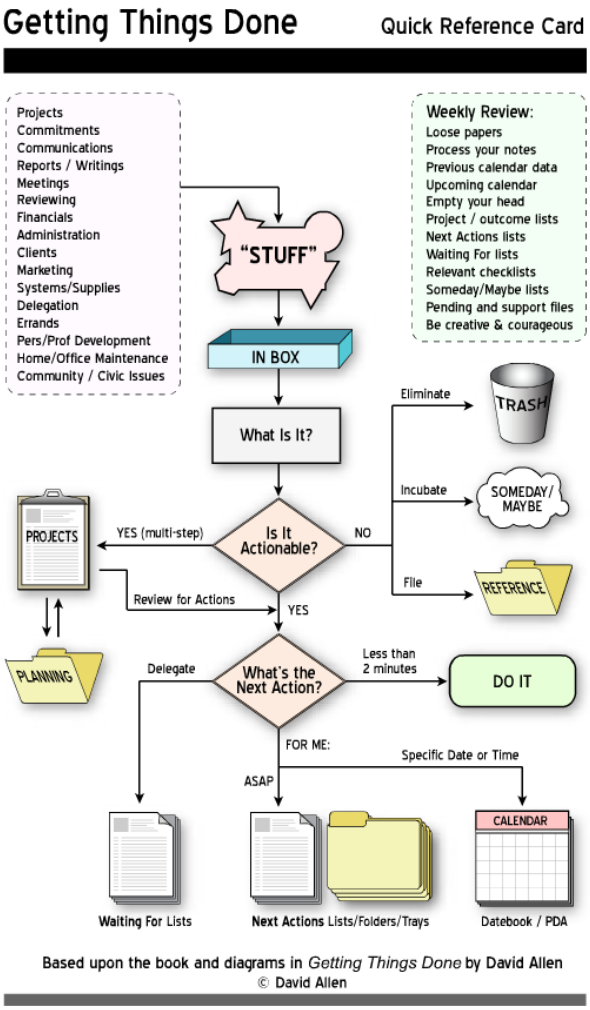

FIG. 4.1 – Diagramme UML d'activités - Vision générale du système (http ://www.produktivniblog.cz)

Chapitre 5

Diagramme de classes

La méthode GTD permet d'organiser les tâches selon plusieurs critères :

- le projet
- le contexte (l'endroit, l'outil ou l'environnement dans lequel la tâche peut être réalisée)
- la priorité
- l'état d'avancement (à faire, déléguée, en attente, finie)

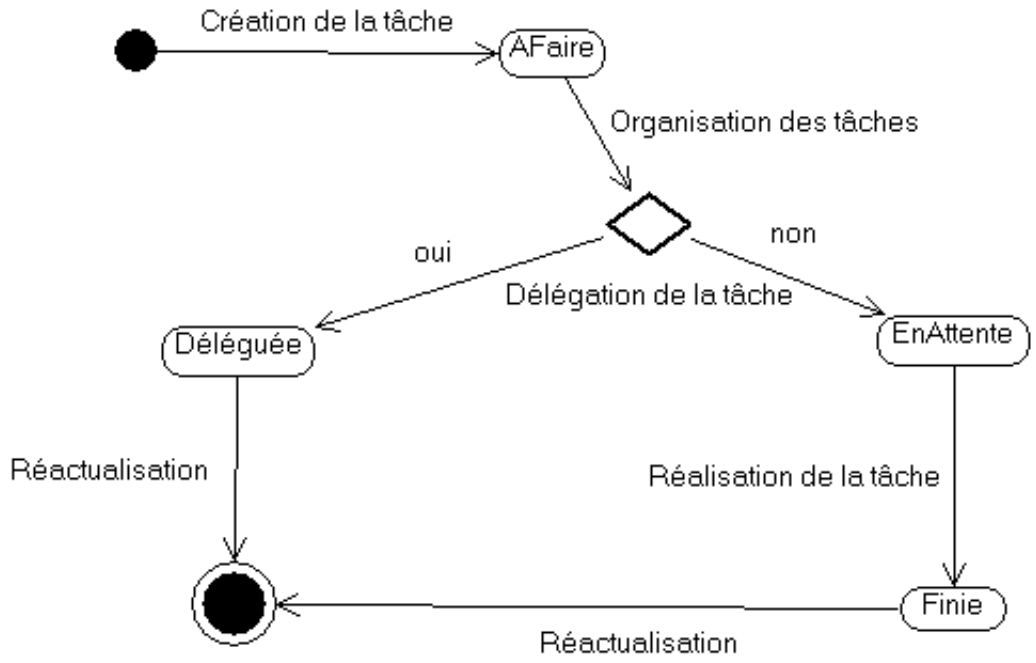

Fig. 5.1 – Diagramme d'états transitions d'une tâche

Un projet regroupe donc des tâches et éventuellement, des sous-projets. A l'intérieur d'un projet, les tâches peuvent être indépendantes les unes des autres ou organisées en séquence. Un projet possède un nom, des notes, et peut préciser le contexte par défaut des nouvelles tâches du projet. Une tâche possède différents attributs :

- son nom
- une date de début
- une échéance
- des notes
- une priorité (valeur comprise entre 1 et 5)
- un taux d'effort demandé (valeur comprise entre 0 et 99)
- une liste de contacts
- une fréquence de répétition (journalière, hebdomadaire, mensuelle)
- une date d'arrêt de la répétition
- des liens (URLs) vers d'autres ressources
- des tags
- un et un seul contexte

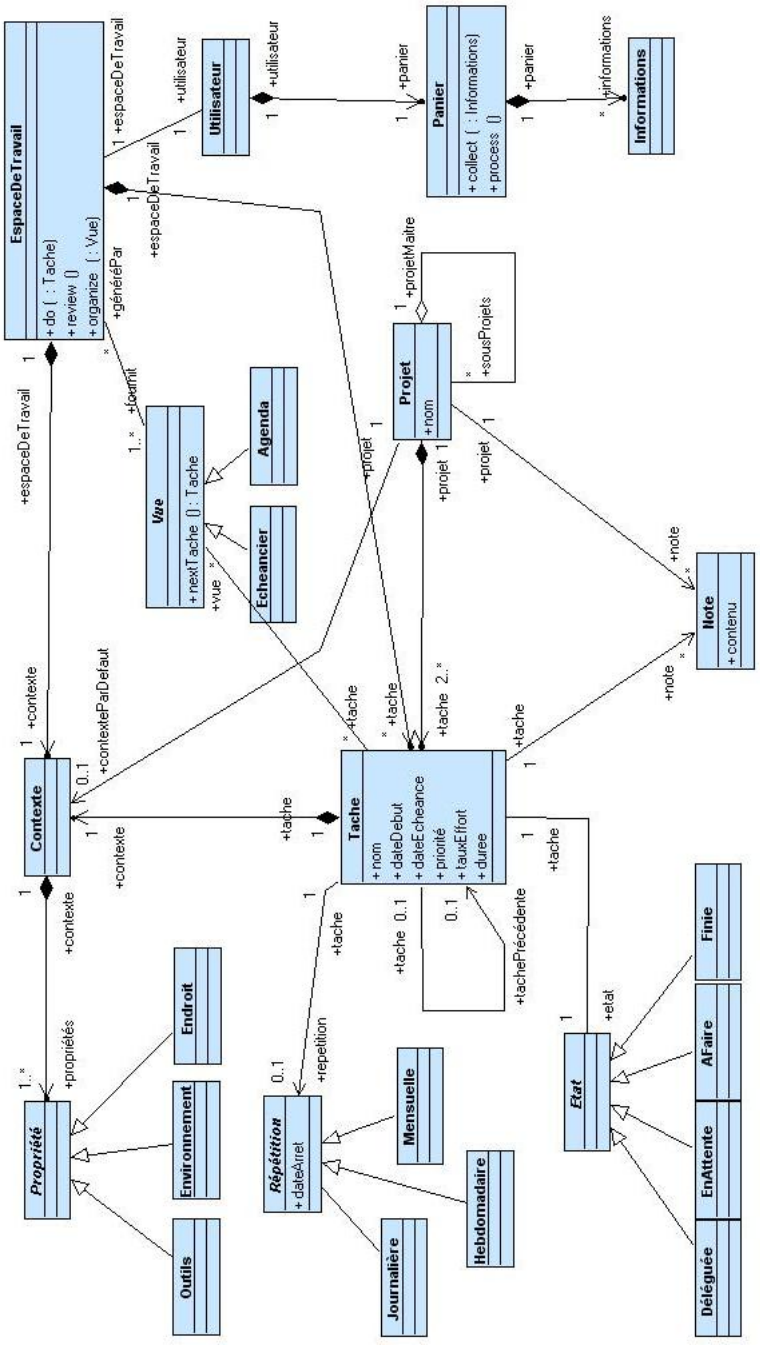

FIG. 5.2 – Diagramme UML de classe

5.1 Contraintes OCL et explications

Les deux classes centrales sont **Projet** et **Tache**. Un projet peut inclure des sous-projets et ce récursivement. Un projet peut aussi contenir une séquence de tâches, dans ce cas les tâches sont vues sous formes d'une liste châinée. Seules les tâches d'un projet peuvent être séquencées :

```
context Tache
        inv :
                self.projet->empty() implies self.tachePr[U+FFFD][U+FFFD]nte->empt
                self.tachePr[U+FFFD][U+FFFD]nte->notEmpty() implies self.tachePr[U+FFFD
                self.projet->notEmpty() implies self.projet = self.tachePr[U+
                self.dateDebut <= self.dateEcheance and
                self.dateEchance >= self.repetition.dateArret and
                self.tauxEffort >= 0 and
                self.tauxEffort <= 99 and
                self.prior[U+FFFD]>= 1 and
                self.prior[U+FFFD]<= 5 and
                self.duree <= (self.dateEcheance - self.dateDebut)
```

Un projet possède aussi un **contexteParDéfaut**, c'est à dire que tous ses sous-projets ainsi que toutes ses tâches incluent son contexte dans le leur :

```
context Projet
        inv :
                self.contexteParDefaut.proprietes->notEmpty() implies
                (self.tache->forAll(each | each.contexte.proprietes->incl
                self.sousProjet->forAll(each | each.contexte.proprietes->
```

Pour effectuer la tâche courante, si celle-ci est séquencée, les tâches la précédant doivent être dans l'état **Finie**. Elle doit aussi être réalisable :

```
context EspaceDeTravail::do(t : Tache)
        pre :
                t.tachePr[U+FFFD][U+FFFD]nte->notEmpty() implies t.tachePr[U+FFFD][U+FFF
                t.tachePr[U+FFFD][U+FFFD]nte.oclInState(Finie) and
                not(t.oclInState(Finie)) and
                self.contexteCourant.proprietes->includesAll(t.contexte
        post :
                t.oclInState(Finie)
```

La tâche courante est fournie à l'**Utilisateur** par la **Vue** de l'**EspaceDeTravail** en fonction de l'ordre chronologique (**Agenda**), des priorités (**Echeancier**)...

```
context Vue
        inv :
                self.generePar.tache->includesAll(self.tache)

context Vue::nextTache() : Tache
        post :
                result = self.tache->first()
```

C'est l'espace de travail qui offre la possibilité à l'utilisateur d'organiser les tâches en fonction des contextes, projets, dépendances (création de séquencement)...

```
context EspaceDeTravail::organise(v : Vue)
        post :
                v.tache = v.tache@pre->asOrderedSet() and
                self.tache->forAll(each | (each@pre.oclInState(AFaire)) i
```

Il offre aussi la possibilité de réactualiser l'ensemble des informations :

```
context EspaceDeTravail::review()
        post :
                self.tache->notExists(each | each.oclInState(Finie))
```

Le **Panier** représente le conteneur dans lequel l'utilisateur va noter (**collect(:Informations)**) tout événements, objets, documents ou idées (**Informations**) qui pourraient requérir une action ultérieure. Ces informations sont ensuite traitées par la méthode **process()**. Celle-ci crée les tâches correspondantes aux informations si c'est pertinent :

```
context Panier::process()
        post :
                self.informations->empty()
```

Chapitre 6

Conclusion

Cette première approche nous a permis d'analyser, de comprendre et de modéliser la méthode GTD. L'implémentation n'est cependant pas imminente. En effet, cette analyse constitue un travail nécessaire à la bonne compréhension du système, mais pour aboutir à l'application, il sera obligatoire d'effectuer une phase de conception concernant, cette fois, l'application elle-même implémentant la méthode GTD.
Ce premier livrable nous a aussi permis d'utiliser OCL une fois de plus, ainsi que de prendre en main l'outil Topcased. Au fur et à mesure de notre réflexion, notre vision de l'application finale s'est éclaircie.
Il faudra cependant confronter notre vision des choses avec celle des autres groupes pour décider d'un standard nécessaire au bon fonctionnement de l'application finale.

Deuxième partie

Livrable 2

Chapitre 7

Introduction

7.1 Objectif

Dans ce second livrable, nous allons décrire l'ensemble des fonctionnalités du logiciel à développer. Ce logiciel devra permettre à l'utilisateur d'organiser ses tâches selon la méthode GTD décrite dans le premier livrable. L'objectif de ce livrable est de fournir une description du comportement de notre application. De ce fait, les contraintes fonctionnelles permettront par le biais de cas d'utilisation de préciser le comportement du logiciel. Nous préciserons cela grâce à des sous cas d'utilisation du livrable 1. De plus, les critères d'utilisation tels que la fiabilité, l'utilisabilité ou la performance feront l'objet de besoins non-fonctionnels.

7.2 Conventions

Dans la suite du document, les notions évoquées dans le premier livrable sont considérées comme acquises. Cependant, lorsque cela sera nécessaire, des références vers le premier livrable seront utilisées. Nous distinguons 4 actions bien distinctes issues du premier livrable :

- La collecte des informations = Collect
- Le traitement des informations = Process
- L'oganisation des tâches = Organize
- La révision des tâches (maj) = Revue

Nous utiliserons donc ces raccourcis pour nommer ci-besoin, l'action courante que l'utilisateur effectue.

7.3 Audience

Ce document est destiné à être utilisé par l'ensemble des participants au projet. Après approbation, il sera considéré comme document de référence.

7.4 Portée du document

Le développement du logiciel va s'appuyer sur ce document. Ainsi, chaque élément décrit dans ce livrable va directement influencer le reste des livrables. Comme il est indiqué précédemment, ce livrable s'appuie sur l'analyse de GTD réalisée dans le livrable 1.

7.5 Définitions, acronymes et abréviations

Afin de bien cerner les éléments relatifs à la méthode GTD (Getting Things Done) la lecture du livrable 1 est nécessaire, en particulier le dictionnaire de données.

7.6 Organisation du chapitre

Dans un premier temps, les besoins fonctionnels seront détaillés. Ces besoins doivent définir le comportement attendu du système. Pour cela nous expliciteront à l'aide de cas d'utilisation.

Dans une seconde partie, les besoins non-fonctionnels permettant de caractériser l'environnement dans lequel le système doit fonctionner, seront détaillées. Ils peuvent être vus comme des contraintes d'utilisation, mais aussi comme des critères de qualité du développement du logiciel.

Chapitre 8

Description générale

8.1 Perspectives du produit

L'objectif du produit est de répondre à un besoin d'utiliser une application se conformant à la méthode GTD. De plus, celle ci doit être utilisable en déplacement. Néanmoins il existe déja un certain nombre d'applications GTD de ce type. Cette application est divisée en différents composants. Ce livrable ne concerne cependant que les composants Serveur Web et Client Web (cf. figure 9.4), qui nous ont été demandés.

8.2 Fonctions du produit

Le logiciel a pour but de fournir l'ensemble des fonctionnalités décrites dans la méthode GTD. Ces fonctions seront décrites plus précisément dans la suite de ce document.

8.3 Caractéristiques et classes d'utilisateurs

Le système sera multi-utilisateur : plusieurs utilisateurs pourront donc se connecter au serveur web. Ces utilisateurs seront à distinguer des administrateurs du serveur GTD disposant de privilèges supérieurs.

8.4 Environnement opérationnel

8.4.1 Matériels

Dans un premier temps, l'application serveur sera installée sur une machine d'une puissance minimum, de manière à pouvoir gérer les multiples connexions. Les spécifications

de cette machine sont au minimum :

- Processeur actuel type Celeron
- Mémoire vive de 2go minimum pour gérer les multiples connexions
- Disque dur standard

L'utilisation de notre application sera effectuée à l'aide d'un client web qui par nature sera légé. N'importe quelle configuration permettant l'affichage de pages web sera fonctionnelle :

- Processeur actuel type Celeron
- Mémoire vive de 1go minimum
- Disque dur standard

8.4.2 Système d'exploitation & client Web

En ce qui concerne l'utilisation du service web, elle sera indépendante du système d'exploitation et utilisable avec la pluparts des navigateurs Internet (Firefox, Google Chrome, InternetExplorer ...)

L'application serveur sera quant à elle déployée sur un serveur Linux, et une distribution type Debian ou dérivées.

8.5 Contraintes de conception et d'implémentation

Il est impératif de prendre en compte tous les éléments décrits dans la méthode GTD (voir livrable 1). De plus, l'architecture de l'application doit être conforme au schéma suivant :

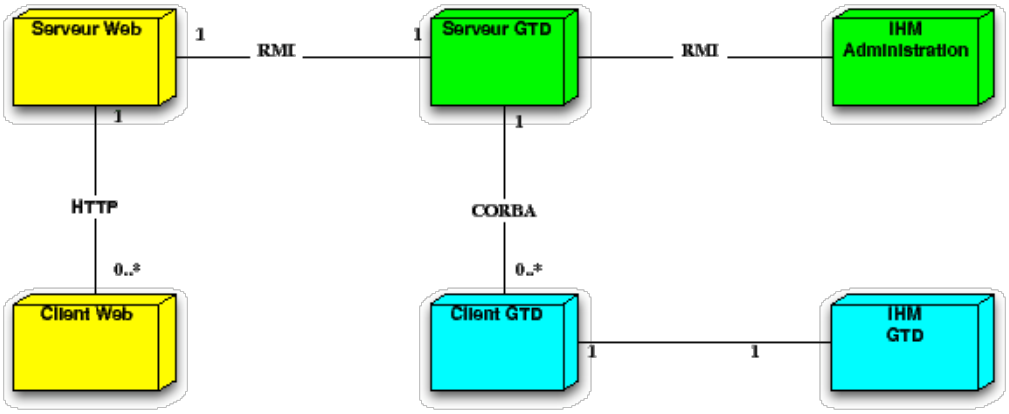

FIG. 8.1 – Architecture Générale

8.6 Documentation utilisateur

Une aide en ligne doit être mise en oeuvre. Une section du site doit être dédiée à celle-ci. Des rollover ou des messages d'aide sur le site guideront l'utilisateur. L'aide en ligne devra suffir à un utilisateur maitrisant les bases de la navigation Web pour utiliser le logiciel. Aucune aide papier ne sera fournie pour l'application web, de par sa nature distante.

Une courte formation sera effectuée lors de la livraison et du déploiement de l'application. Elle sera donnée à l'utilisateur principal du logiciel, lui permettant de la prendre en main rapidement.

8.7 Hypothèses et dépendances

Comme le montre la figure 9.4, l'application est un composite, chaque composant pouvant dépendre d'un autre. Le client Web est ainsi dépendant du serveur Web, qui est lui même dépendant d'un serveur GTD. De même, l'IHM d'administration et le client GTD sont dépendants du Serveur GTD, et l'IHM GTD est dépendante du client GTD.

Il sera possible d'utiliser deux serveur GTD. D'une part un server GTD développé par un autre groupe, et d'une autre part un serveur ToodleDo[1].

[1] www.toodledo.com

8.8 Exigences reportées

L'internationalisation sera réalisée dans une version future du système. L'application sera fournie en français uniquement dans un premier temps.

Chapitre 9

Exigences fonctionnelles

9.1 Identification

Use Case: 1 – Identification

CHARACTERISTIC INFORMATION

Acteur : Utilisateur
Niveau : Tâche principale
Portée : IHM, Base de donnée
Pré-condition : L'utilisateur est non connecté et déja inscrit.
Post-condition : L'utilisateur est connecté.
Priorité : (5/5) : Cette étape est necessaire pour chacun des prochains cas d'utilisation.
Fréquence : Chaque lancement d'application.
MAIN SUCCESS SCENARIO

1 L'utilisateur entre sont identifiant et son mot de passe,
2 Il valide,
3 L'utilisateur est connecté.

EXTENSIONS

1 Renvoyer le mot de passe
2 Création d'un compte

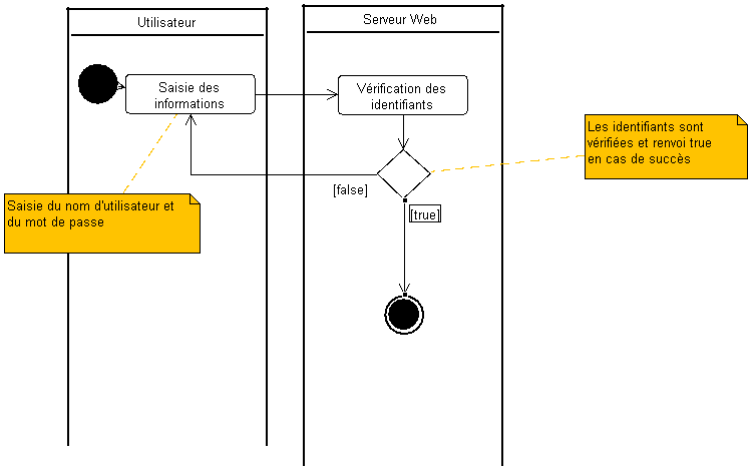

FIG. 9.1 – UML - Diagramme d'activitées - Identification

9.1.1 Exigences fonctionnelles

FONC11 - Saisie des identifiants,
FONC12 - Connexion (avec gestion de session),
FONC13 - Inscription,
FONC14 - Régéneration de mot de passe.

9.2 Collect

L'application n'a normalement pas à intervenir dans cette étape. Le recensement des idées est effectivement un processus utilisateur. Cependant, un pense-bête à idées non traitées par l'utilisateur peut s'avérer fort utile dans le cas où l'utilisateur est interrompu dans son processus.Un pense bête permet alors de stocker les idées non traitées.

Use Case: 2 – Collect

CHARACTERISTIC INFORMATION

Acteur : Utilisateur

Niveau : Tâche principale

Portée : IHM, Base de donnée

Partie prenante et interet : Le recensement exhaustif de tout ce qui peut justifier une quelconque intervention de notre part : en suspens, inachèvement, en attente, intention, projet, manque, usure, mauvais fonctionnement, problème, in- satisfaction, besoin, engagements à tenir,etc. Exemples : cette carte de visite restée dans une poche, cette facture dans la boîte à gants, cette demande reçue, ce dossier qui traîne sur le bureau, cette agrafeuse qui coince, cette course à faire, ce problème à résoudre, cette suggestion à tester, les messages de la boîte vocale, ce projet jamais réalisée, ce souci de santé, ce fauteuil qui grince, les performances de ce collaborateur, toutes ces choses en retard...

Pré-condition : Aucune.

Post-condition : Les informations sont enregistrées dans l'application.

Priorité : (1/5) : Cette étape n'est pas indispensable, et peutêtre effectuer sans l'aide du système informatique.

Fréquence : Chaque début de journée.

MAIN SUCCESS SCENARIO

1 L'utilisateur après avoir effectuer sa reflexion, saisie les informations qu'il à recensée dans la boite à idée de l'application.

9.2.1 Exigences fonctionnelles

FONC21 - Saisie des idées,
FONC22 - Stockage des idées,
FONC23 - Suppression des idées.

9.3 Organisation des tâches

Use Case: 3 – Organize

CHARACTERISTIC INFORMATION

Acteur : Utilisateur

Niveau : Tâche principale

Portée : IHM, base de données

Pré-condition : On dispose d"une liste de tâches issues du traitement des données.

Post-condition : Les tâches appartiennent ou non à un projet, sont organisées en séquence ou non.

Priotité : Haute

Fréquence : Après chaque traitement des données.

MAIN SUCCESS SCENARIO

1 L'utilisateur regroupe les tâches par projet en fonction des propriétés communes de leurs contextes et de leurs intérêts communs.

1 Il peut aussi séquencer les tâches d'un projet.

2 Il a aussi la possibilité de créer des sous-projets afin de donner un aspect hiérarchique au projet principal.

EXTENSIONS

1 Si une tâche est isolée, elle n'appartient à aucun projet et n'est pas séquençable.

9.3.1 Exigences fonctionnelles

Pour regrouper les tâches l'utilisateur doit pouvoir :
FONC31 - créer/modifier/supprimer des projets,
FONC33 - affecter ou non des tâches à un projet,
FONC35 - affecter ou non des projets à un projet (sous-projets).

9.3.2 Exigences non-fonctionnelles

FONC36 - Les tâches et sous-projets d'un même projet doivent participer à un même but.
FONC37 - Les tâches d'une même séquence doivent appartenir au même projet.
FONC38 - Les tâches et sous-projets d'un même projet doivent avoir des propriétés communes dans leurs contextes respectif.

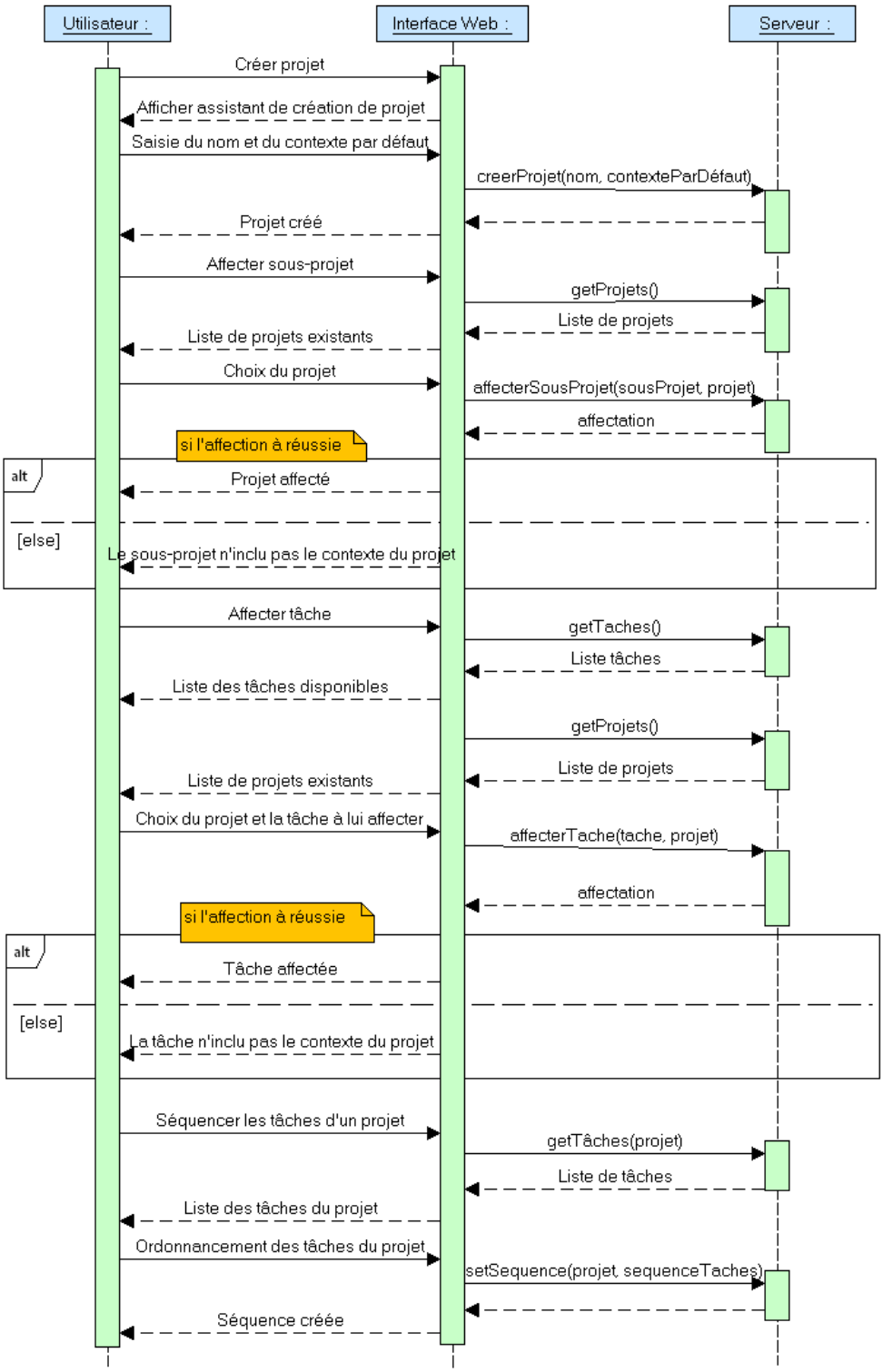

FIG. 9.2 – Diagramme de séquence de l'organisation des tâches

9.4 Affichage des tâches

Use Case: 4 – Affichage des tâches

CHARACTERISTIC INFORMATION

Acteur : Utilisateur

Niveau : Tâche principale

Portée : IHM

Pré-condition : On dispose d'une liste de tâches, de contextes, de projets...

Post-condition : Les tâches sont affichées selon les critères et vues choisies par l'utilisateur.

Priorité : Haute

Fréquence : Selon les besoins de l'utilisateur.

MAIN SUCCESS SCENARIO

1 L'utilisateur affiche les tâches qu'il effectuera selon les projets qu'il a choisit et le contexte courant.

2 Les tâches sont présentées de différentes manières (échéancier, agenda...).

9.4.1 Exigences fonctionnelles

L'utilisateur doit pouvoir via l'IHM :
FONC71- choisir les projets à réaliser.
FONC72- sélectionner la ou les vues dans lesquelles seront affichées les tâches (échéancier, agenda...).

9.5 Réactualisation des tâches

Use Case: 5 – Réactualisation des tâches

CHARACTERISTIC INFORMATION

Acteur : Utilisateur

Niveau : Tâche principale

Portée : IHM, base de données

Pré-condition : On dispose d"une liste de tâches.

Post-condition : On délègue des tâches, celles finies sont supprimées, les vues sont mises à jour suivant le nouveau contexte, création automatique des tâches périodiques.

Priorité : Haute

Fréquence : Au moins une fois par jour.

MAIN SUCCESS SCENARIO

1 L'utilisateur change le contexte courant.

2 Indique les tâches effectuées.

3 Délègue des tâches.

EXTENSIONS

2b L'utilisateur peut annuler ses modifications.

9.5.1 Exigences fonctionnelles

L'utilisateur doit pouvoir :
FONC41 - mettre à jour le contexte.
FONC42 - indiquer les tâches réalisées.
FONC43 - délèguer une tâche.
FONC44 - revenir en arrière.

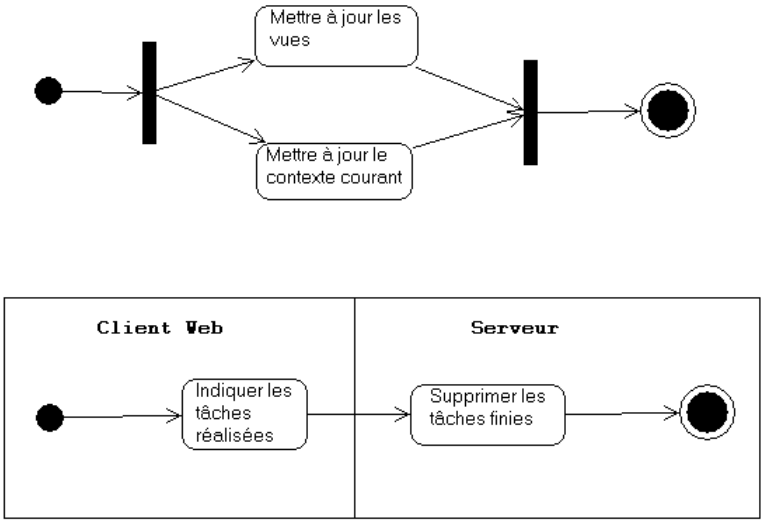

FIG. 9.3 – Diagramme d'activité de la réactualisation des tâches

9.6 Traitement des informations

Use Case: 6 – Process

CHARACTERISTIC INFORMATION

Acteur : Utilisateur

Niveau : Tâche principale

Portée : IHM, Base de donnée

Pré-condition : L'utilisateur est connecté au serveur web.

Post-condition : Une tâche est créée sur les serveurs GTD et ToodleDo.

Priorité : (5/5) : Cette étape est necessaire pour chacun des prochains cas d'utilisation.

Fréquence : De façon ponctuelle.

MAIN SUCCESS SCENARIO

1 L'utilisateur regarde les éléments affichés dans sa boite à idées.

2 L'utilisateur saisie les informations relatives à la tâche qu'il souhaite créer (nom, date debut, date fin, priorité, temps et energie requis ...).

3 L'utilisateur crée la tâche.

EXTENSIONS

1 L'utilisateur oublie de remplir un champ obligatoire.

9.6.1 Exigences fonctionnelles

FONC51 - Affichage de la boite à idées,
FONC52 - Saisie des informations relatives à une tâche,
FONC53 - Création d'une tâche,

9.7 Synchronisation

Use Case: 7 – Synchronisation

CHARACTERISTIC INFORMATION

Acteur : Serveur Web

Niveau : Tâche principale

Portée : Serveur GTD et ToodleDo et Serveur Web

Pré-condition : Le serveur web n'est pas synchronisé avec le serveur GTD ni ToodleDo

Post-condition : Le serveur web et les serveurs GTD et ToodleDo sont synchronisés

Priorité : (5/5) : Cette étape est necessaire pour assurer une persistance des informations.

Fréquence : A chacune des transactions

MAIN SUCCESS SCENARIO

1 L'utilisateur effectue une opération sur le client web
2 Le serveur web enregistre les modifications utilisateur en mettant à jour sa base de données
3 Le serveur web envoi les modifications effectuées aux serveurs

EXTENSIONS

1 La connexion entre les serveurs web et GTD et/ou ToodleDo est interrompue
2 La connexion entre les serveurs web et GTD et/ou ToodleDo se rétablie

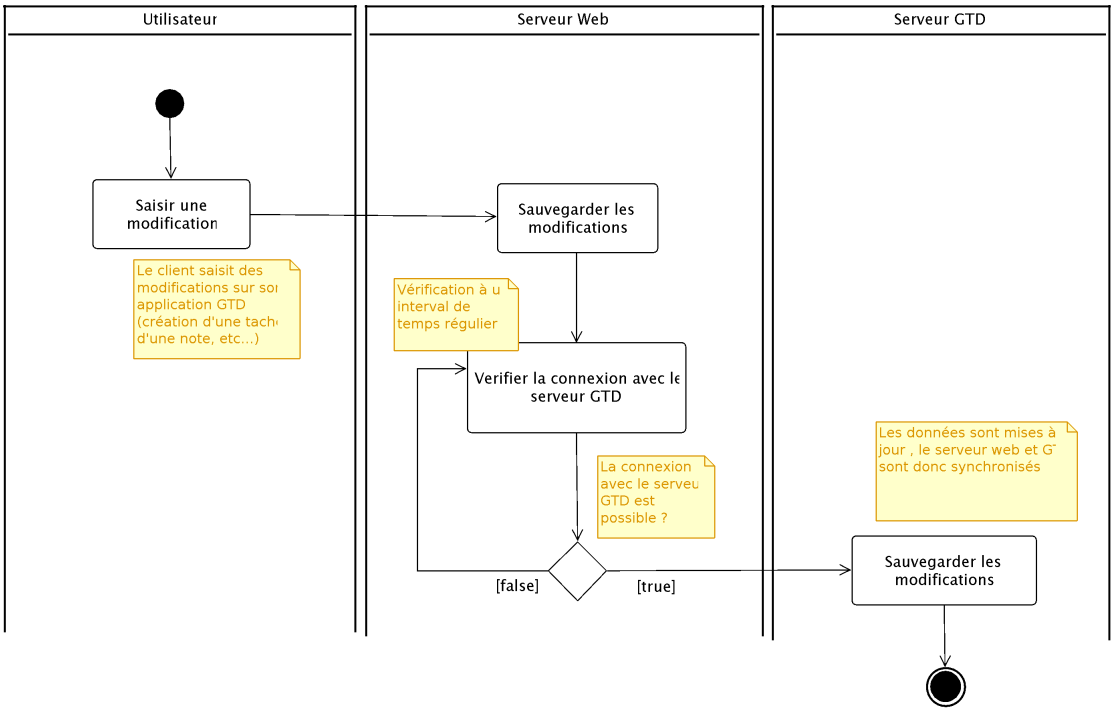

FIG. 9.4 – UML - Diagramme d'activitées - Synchronisation

9.7.1 Exigences fonctionnelles

FONC61 - Enregistrement des modifications sur le serveur web,
FONC62 - Enregistrement des modifications sur les serveurs GTD et ToodleDo
FONC63 - Enregistrement des modifications sur les serveurs GTD et ToodleDo après une perte de connexion

Chapitre 10

Exigences des interfaces externes

10.1 Interface utilisateur

L'application GTD disposera d'une interface Web. Celle-ci devra être concue de façon à être ergonomique. Elle devra respecter une cohérence dans ses couleurs et devra être simple à utiliser. En ce qui concerne sa compatibilité elle devra respecter les standards web tels que le XHTML et CSS3 et ainsi passer les validations W3C. Elle devra reprendre tous les éléments décris dans le chapitre fonctionnalités du logiciel.

10.2 Interface matérielle

Pour la communication entre le Client et le serveur web, les deux postes distants devront disposer d'une interface réseau ethernet. Une connexion ADSL 512k est le minimum requis pour permettre à l'application de fonctionner normalement. De même, la communication entre le serveur web et les serveurs GTD et ToodleDo se fera à travers une connexion ADSL 512K minimum.

10.3 Interface logicielle

La connexion entre le serveur web et les serveurs va se faire grâce à un certain nombre d'interfaces définies par les serveurs GTD et ToodleDo. L'application utilisera ces interfaces à travers RMI et REST pour le server ToodleDo.

10.4 Interfaces ou protocoles de communication

Pour réaliser la communication entre le client et le serveur Web, les données transiterons à travers le protocol HTTPS. L'utilisation de HTTPS se fera pour permettre une

sécurisation des données, HTTPS étant une combinaison de HTTP et d'une couche de chiffrement SSL ou TLS.

La communication entre le serveur web et le serveur GTD se fera grâce à RMI (Remote method invocation). La communication entre le serveur web et le serveur Toodle se fera grâce à REST.

10.5 Contraintes de mémoire

Dans le but de limiter la charge sur le serveur web et GTD, le programme devra limiter au maximum le nombre de ses requêtes. De même, l'envoi de données à travers le réseau se limitera à l'envoi d'objets inférieurs à 500Ko.

Chapitre 11

Autres exigences non-fonctionnelles

Dans ce chapitre, l'ensemble des exigences autres que fonctionnelles est défini. Décrire les exigences qui ne sont pas incluses dans les cas d'utilisation ainsi que les exigences non-fonctionnelles. On peut référer au document de spécifications supplémentaires.

11.1 Utilisabilité

L'application GTD disposera d'une interface Web. Celle-ci devra être conçue de façon à être ergonomique. Elle devra respecter une cohérence dans ses couleurs et devra être simple à utiliser. En ce qui concerne sa compatibilité, elle devra respecter les standards web tels que le XHTML et CSS3 et ainsi passer les validations W3C. L'utilisation du logiciel devra permettre de manipuler des tâches rapidement afin de ne pas rendre son utilisation trop contraignante. Il ne faut pas perdre de vue que ce logiciel est destiné à l'organisation des tâches quotidiennes : son utilisation ne doit pas faire l'objet d'une nouvelle tâche quotidienne !

11.2 Fiabilité

11.2.1 Fréquence et gravité des échecs

Le logiciel développé doit permettre de prendre en charges un maximum d'erreurs survenues lors de l'exécution. Par exemple, le traitement d'une mauvaise saisie utilisateur doit être prévue et traitée en fonction. Dans la suite, nous distinguons les erreurs critiques auquel nous n'avons pas pensées et les erreurs prévisibles (par exemple les erreurs de connexion à la BD).

11.2.2 Temps moyen entre failles

Pour mesurer la fiabilité de notre système et définir des contraintes qualités pour le client nous utiliserons les mesures MTBF (mean time between failure) et MTTF (mean Time To Failure). Notre MTBF de référence pour ce logiciel devra être borné à la valeur 30 jours.

11.2.3 Prédictibilité

Certaines erreurs ne sont pas prédictibles dans le temps de part leur nature, mais vont néanmoins êtres prises en compte lors de l'éxecution du programme. Comme par exemple les erreurs de connexion avec le serveur.

11.2.4 Récupérabilité

En ce qui concerne le MTTR (Mean time to repair), lors d'une découverte d'une faute critique, le temps de réparation est fixé à 48h. Cependant, pour les fautes mineures l'assurance de la correction est assurée dans un délai de 1 mois.

11.3 Exigences de performance

11.3.1 Rapidité

La communication entre le serveur Web et l'interface graphique doit être suffisament rapide pour ne pas ressentir une sensation de lenteur. L'interface graphique ne doit pas se figer lors d'une action effectuée par l'utilisateur. En cas de temps d'attente, une notification sous forme d'une barre de progression doit être affichée. La possibilité de mettre en tâche de fond peut être envisagée.

11.3.2 Efficacité

L'application doit permettre d'effectuer les actions dans un temps imparti qui reste acceptable pour l'utilisateur. Elle doit se limiter à l'utilisation des ressources strictement nécessaires à l'accomplissement des fonctions.

11.3.3 Disponibilité

Le client Web ne peut pas être autonome si le serveur Web n'est pas disponible. Cependant lorsque la connexion est coupée entre le serveur web et le serveur GTD, l'application doit tout de même être fonctionnelle avec ToodleDo, ou l'inverse. Cependant, en cas de coupure des deux serveurs aucune action ne sera réalisable.

11.3.4 Précision

Pour chacune des actions réalisées dans le système, l'opération est déterministe. Etant donné que le système effectue des actions basiques, une précision de 1 est tout à fait réalisable. En effet, dans tous les cas de figure, le résultat escompté sera toujours le résultat exact et non pas une approximation.

11.3.5 Bande passante

Une connexion ADSL 512k est le minimum requis pour permettre à l'application de fonctionner normalement.

11.3.6 Capacité

Le serveur doit permettre d'assurer un minimum de 500 connexions multiples. Les contraintes techniques liées au serveur sont décrites dans ce livrable.

11.3.7 Temps de réponse

Le temps de réponse entre le serveur Web et le client Web doit être inférieur à 40ms. Les temps de réponses entre le client Web et le serveur Web dépendront de la communication entre le serveur Web et les serveurs Toodle et GTD.

11.3.8 Utilisation des ressources

L'utilisation du client Web ne doit pas dépasser 30mo de mémoire vive. Etant donné qu'il est susceptible que l'utilisateur effectue d'autres actions en même temps, il est indispensable que le client Web ne consomme pas trop de ressources.

11.4 Adaptabilité

L'architecture du système doit être assez flexible pour permettre plus tard de :

- rajouter facilement des extensions (nouvelles fonctionnalités...)
- évoluer vers de nouvelles technologies (passer à un nouveau type d'interface sans toucher à la partie serveur par exemple)
- maintenir celui-ci (corrections, nouvelles versions...)
- internationaliser l'interface

11.5 Maintenabilité

11.5.1 Normes de codage

Dans le but de rendre le code source lisible et homogène celui-ci devra respecter la norme de codage java définie par SUN.

11.6 Exigences de sûreté

Le logiciel est considéré comme sûr, aucune perte de données ou de dommage matériel ne peut résulter de son utilisation. Cependant, lors de l'installation du serveur web, l'administrateur devra s'assurer qu'aucun problème de compatibilité ne sera engendré (Par exemple utilisation d'un port déja utilisé).

11.7 Exigences de sécurité

L'application est destinée à une utilisation multi-utilisateurs. Pour gérer la confidentialité des données, une identification lors de l'ouverture du logiciel est demandée. Comme citée dans la section description générale, les demandes de connexion seront cryptées à l'aide du protocole HTTPS.

11.8 Attributs de qualité logicielle

11.8.1 Disponibilité

L'utilisation du logiciel étant possible en mode déconnecté, la disponibilité du serveur GTD devra être assurée avec un minimum de 60% afin d'avoir une base d'informations suffisament à jour.

11.8.2 Tests logiciels

Afin d'assurer une cohérence dans le processus de développement, le code source devra disposé de tests unitaires (par exemple JUnit). L'ensemble de ceux-ci devront couvrir le code à 80%.

11.8.3 Interopérabilité

Le serveur Web étant indépendant du client, l'utilisation d'un nouveau client Web peut être totalement envisageable à partir du moment où les interfaces requises par l'application

déployée sur ce serveur sont respectées.

11.8.4 Portabilité

L'interface logicielle se basant sur les technologies du Web, n'importe quel navigateur récent permettra de l'utiliser. Il sera donc opérationnel sur n'importe quel plateforme (Windows, Unix ...).

Chapitre 12

Classification des exigences fonctionnelles

FONC11	Saisie des identifiants	essentielle
FONC12	Connexion (avec gestion de session)	essentielle
FONC13	Inscription	essentielle
FONC14	Régéneration de mot de passe	souhaitable
FONC21	Saisie des idées	essentielle
FONC22	Stockage des idées	essentielle
FONC23	Suppression des idées	essentielle
FONC31	créer/modifier/supprimer des projets	essentielle
FONC33	affecter ou non des tâches à un projet	essentielle
FONC35	affecter ou non des projets à un projet (sous-projets)	essentielle
FONC36	Les tâches et sous-projets d'un même projet doivent participer à un même but	essentielle
FONC37	Les tâches d'une même séquence doivent appartenir au même projet	essentielle
FONC38	Les tâches et sous-projets d'un même projet doivent avoir des propriétés communes dans leurs contextes respectif.	essentielle
FONC71	choisir les projets à réaliser.	essentielle
FONC72	sélectionner la ou les vues dans lesquelles seront affichées les tâches (échéancier, agenda...).	essentielle
FONC51	Affichage de la boite à idées	essentielle
FONC52	Saisie des informations relatives à une tâche	essentielle
FONC53	Création d'une tâche	essentielle
FONC61	Enregistrement des modifications sur le serveur web	essentielle
FONC62	Enregistrement des modifications sur le serveur GTD et ToodleDo	essentielle
FONC63	Enregistrement des modifications sur le serveur GTD et ToodleDo après une perte de connexion	souhaitable

Chapitre 13

Maquette de l'interface homme-machine

13.1 Connection

Au lancement de l'application l'utilisateur doit pouvoir procéder à toutes les étapes de la méthode GTD. Cependant, la connexion est indispensable à chacune des étapes suivantes. En effet, les données sont stockées dans un compte distant identifié par un login et protégé par un mot de passe. L'interface suivante permet donc de se connecter, s'inscrire et renvoyer son mot de passe en cas de perte de celui-ci.

FIG. 13.1 – Mockup - Connection

Une fois que l'utilisateur est connecté, il peut effectuer l'ensemble des étapes de la

méthode GTD ou seulement celle qui l'intéresse. L'utilisateur doit en effet être libre de choisir ce qu'il veux faire au moment où il lance l'application. Il peut donc le faire par le menu présent en haut de l'interface, présentant dans l'ordre chronologique les différentes étapes.

13.2 Collect

Après avoir cliqué sur l'onglet Collect, l'utilisateur dispose d'une interface offrant les fonctionnalités d'un pense-bête. En effet, Collect est une réflexion de l'utilisateur. La seule aide que pour fournir l'application est un stockage des idées n'ayant pas eu le temps d'être traité dans Process. Il peut donc ajouter (bouton +) et supprimer (bouton -) simplement des idées, ici sous forme de post-it.

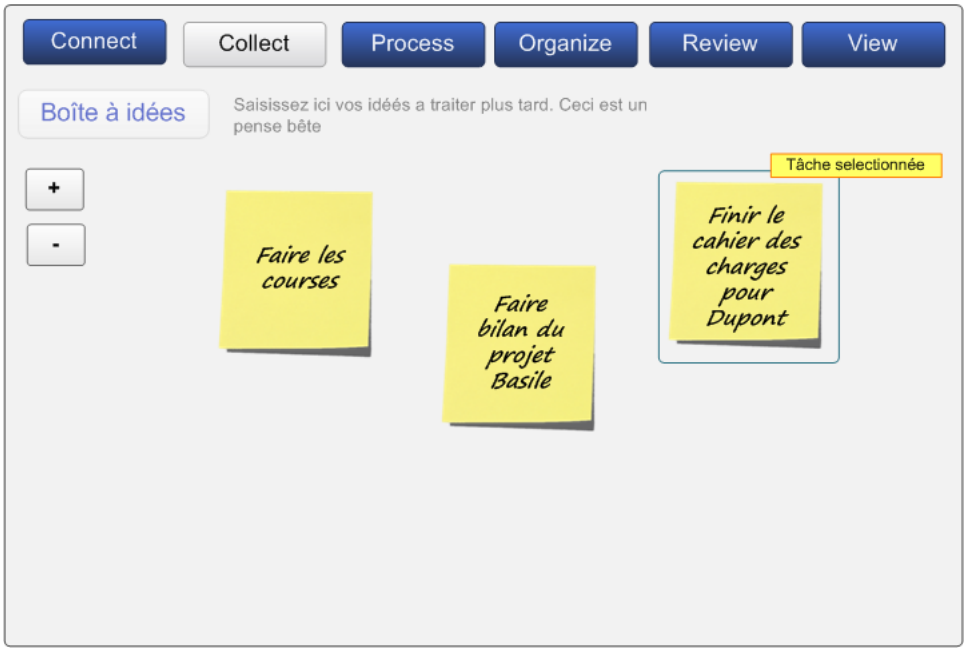

FIG. 13.2 – Mockup - Collect

13.3 Process

Lorsque l'utilisateur clique sur process, il dispose d'une interface affichant les idées qu'il a actuellement dans son pense-bête et d'une interface de saisie de tâches. Il peut donc aisément créer ses tâches avec toutes les informations nécessaires sans oublier celles qui sont dans son pense-bête.

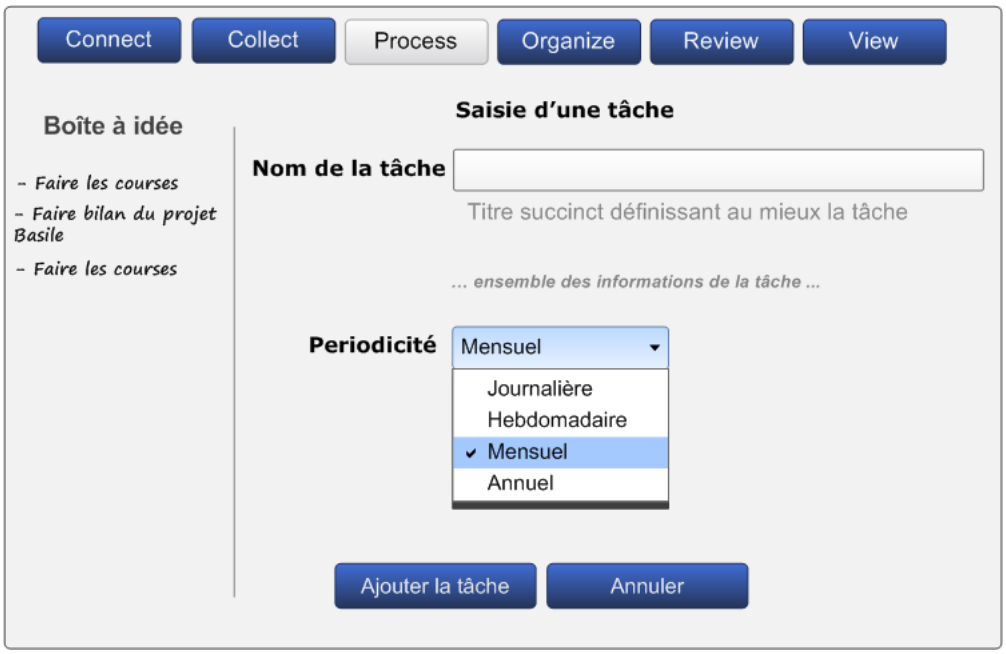

FIG. 13.3 – Mockup - Process

13.4 Organize

Lorsque l'utilisateur clique sur organize, il dispose de deux fonctionnalités :
- Organiser les tâches en projet,
- Déléguer une tâche.

L'organisation inclut la création de projet (bouton +), la définition du contexte par défaut du projet, ainsi que l'ajout des tâches dans un projet par glisser-déposer de la liste 'tâches disponibles' vers 'tâches affectées au projet', en définissant un ordre chronologique. Pour la délégation il suffit de sélectionner une tâche, puis une personne à qui déléguer, et enfin de valider.

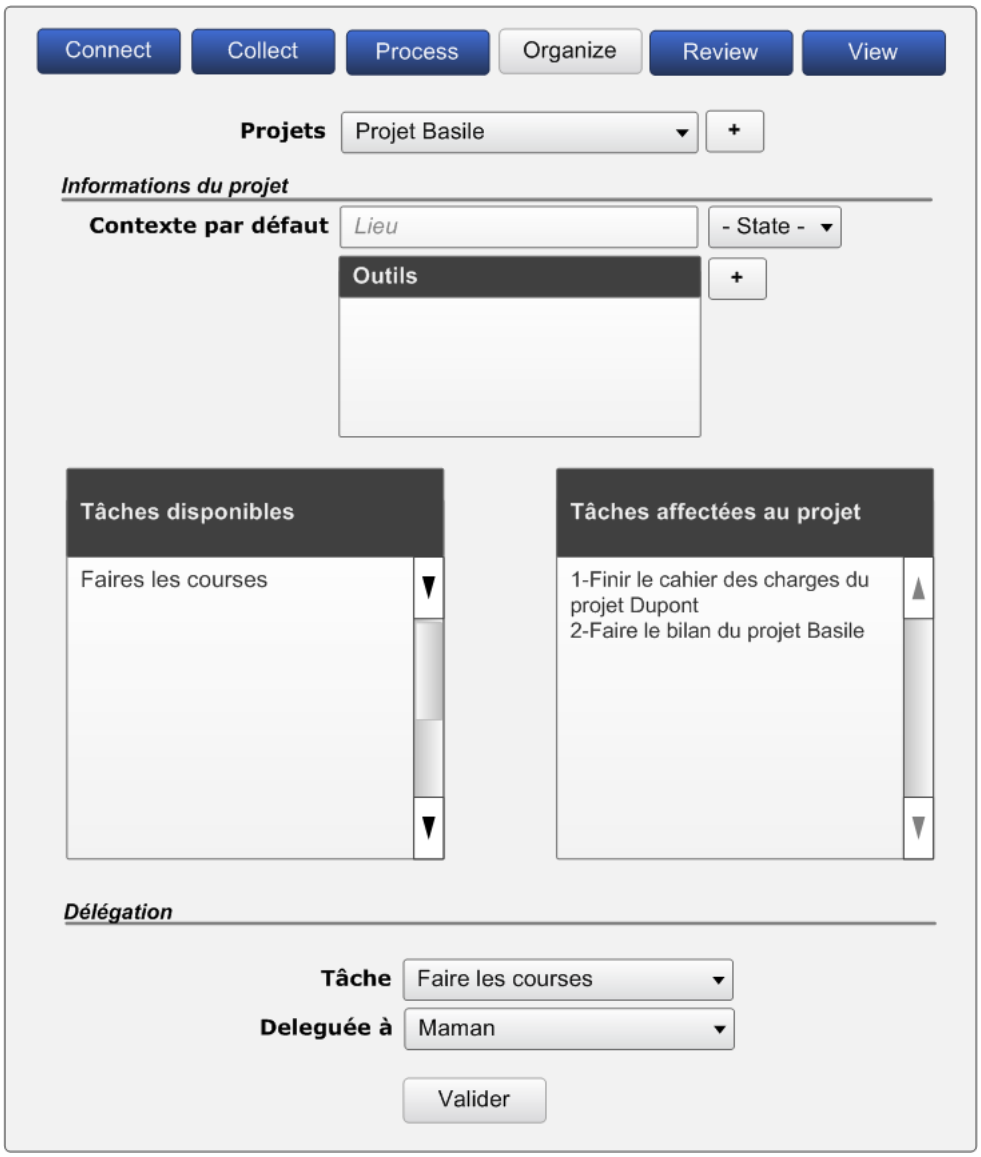

FIG. 13.4 – Mockup - Process

13.5 Review

L'interface Review permet simplement de mettre à jour les informations de certaines tâches. Elle correspond donc à l'interface de process.

FIG. 13.5 – Mockup - Review

13.6 View

La vue est l'opération finale, elle permet d'afficher les tâches en agenda ou en écheancier afin de conseiller l'utilisateur dans la bonne gestion de son temps. Le calendrier affiche les jours occupés et la liste des tâches présentes dans l'ordre de leur priorité depuis la date selectionnée dans le calendrier.

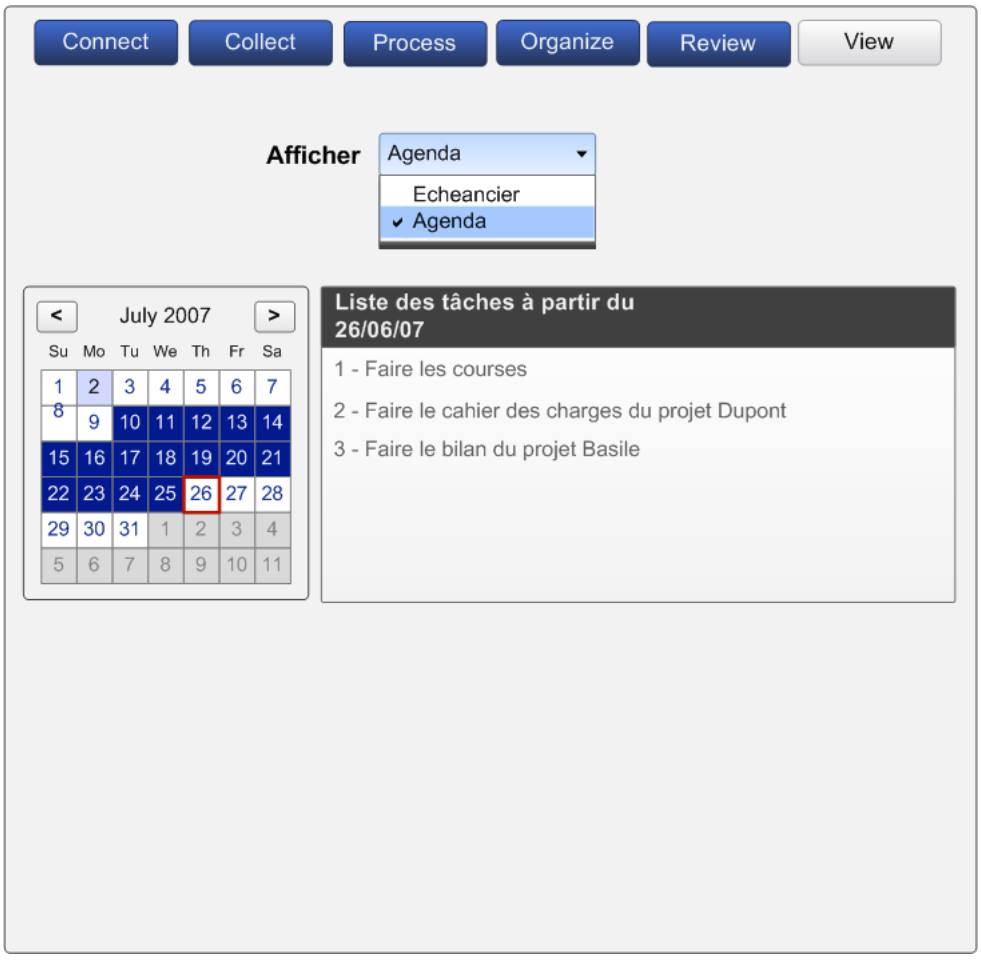

FIG. 13.6 – Mockup - View

Troisième partie

Livrable 3

Chapitre 14

Introduction

14.1 Objectif

Ce troisième chapitre présente la conception de notre application. Ce livrable doit permettre d'améliorer la compréhension de notre vision et faciliter le développement.

14.2 Organisation du chapitre

Le chapitre présente en premier lieu une description des différents composants de l'application et les interactions entre eux. Pour cela, il est indispensable de décrire les interfaces de communication que les composants possèderont. Tous les types présents seront décrits afin que le développeur puisse facilement savoir quels éléments il manipule.

Chapitre 15

Description des composants

L'architecture qui sera détaillée par la suite se limitera à notre partie, a savoir les composants définis par les entités clientWeb et ServeurWeb :

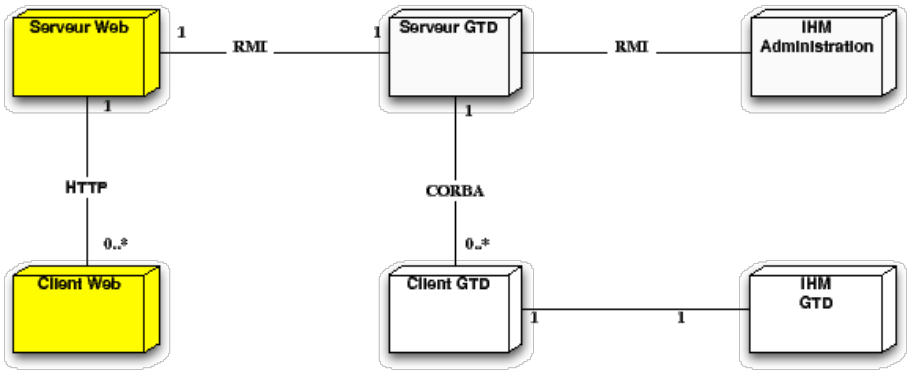

Fig. 15.1 – Architecture Générale - Entités modélisées

15.1 Liste des composants

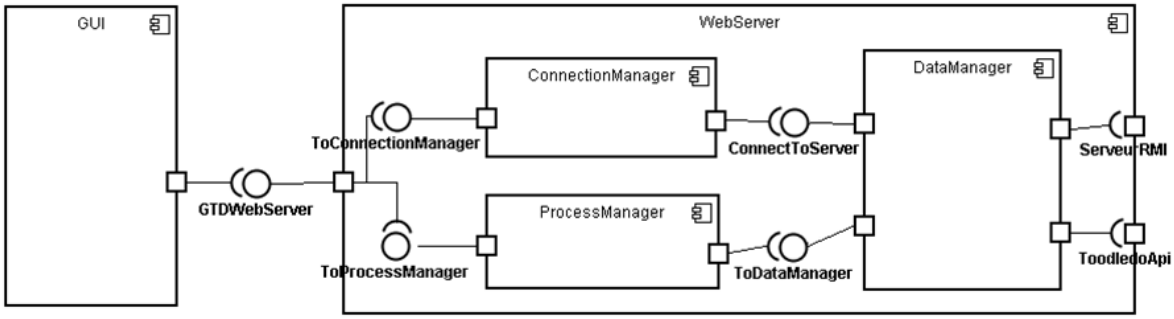

FIG. 15.2 – Architecture Générale - Diagramme de composants

L'application est composée des composants suivants :
- WebClient
- ConnectionManager
- DataManager
- ProcessManager

Chacun d'eux est décrit ci-dessous.

15.2 Composant ClientWeb

L'utilisateur se connectera au serveur GTD à l'aide du composant WebClient. Ce composant est une RIA (Rich Internet Application). Le programme initialement écrit en Java sera transformé à l'aide du compilateur GWT en langage Web (javascript, HTML et css). Il permet donc d'alléger la charge du serveur web : Contrairement à du jsp qui est exécuté sur le serveur web, celui-ci va être exécuté sur le client.

Le composant WebClient va utiliser les interfaces fournies par le WebServer. Celui-ci est en effet un service, les appels vont donc être réalisés via RPC (en javascript). GoogleWebToolkit fournit l'ensemble des outils nécessaires à la réalisation de cette communication.

15.3 Composant ConnectionManager

Ce composant s'occupe de gérer l'ensemble des sessions utilisateurs. En effet notre client Web pouvant être utilisé par une multitude de personnes, le composant s'occupe d'assurer l'identification vers le serveur GTD et/ou ToodleDo, la connexion entre notre serveur Web et le serveur GTD et/ou ToodleDo. Lors d'une perte de connexion entre le

client et le serveur Web, il s'occupera d'envoyer une demande de déconnexion vers le(s) serveur(s).

15.3.1 Gestion des comptes ToodleDo et du serveur GTD

L'application fonctionne avec différents serveurs. Ainsi, le choix qui à été fait consiste à regrouper les informations de chaque serveur (login, password) au sein d'un meta-compte. Celui-ci, dispose de son propre login et mot de passe. Il permet à l'utilisateur de saisir un seul couple login mot de passe pour se connecter aux serveurs ToodleDo et GTD. Les informations concernant ses comptes sont stockées localement sur le serveur d'application.

15.3.2 Liaison vers le client

La communication avec le client sera effectuée à l'aide de l'interface IGTDWebServer. Cette interface expose toutes les méthodes de ConnectionManager et ProcessManager. Le WebServer peut grâce à elle, rediriger les bons appels de connexion/déconnexion vers le composant ConnectionManager, et les appels métiers vers ProcessManager.

15.3.3 Liaison vers DataManager

La communication avec le composant DataManager est effectuée par l'implémentation des méthodes décrites par l'interface du composant ConnectToServer.

15.3.4 Test des connexions et gestion des erreurs

Ce composant doit permettre d'indiquer aux autres composants l'état des connexions en temps réel. Pour cela lors de la phase de connexion, il teste la disponibilité des deux serveurs à savoir du serveur GTD et de ToodleDo. L'interception des exceptions permettra de modifier l'état des connexions à chaque instant. En effet, lorsqu'une exception sera levée de la part du DataManager, il en informera ses observateurs (dont ConnectionManager). Ce dernier mettra alors à jour les données de connexion.

15.4 Composant DataManager

Le composant DataManager va faire le lien entre la couche métier (le composant ProcessManager) et les serveurs GTD / ToodleDo. Il va donc permettre de répondre au problématiques suivantes :

– Permettre la communication entre le ConnectionManager et les serveurs GTD et ToodleDo

- Permettre de renvoyer à la couche métier les données en provenance du serveur GTD et ToodleDo
- Permettre la synchronisation du serveur GTD au serveur ToodleDo
- Permettre la synchronisation du serveur ToodleDo au serveur GTD

15.4.1 Gestion des serveurs GTD et ToodleDo

Le composant DataManager doit rendre l'utilisation des serveurs ToodleDo et GTD transparente pour la couche métier (ProcessManager). Ainsi, les données envoyées et reçues par celle-ci seront génériques et donc de niveau métier. Cependant, lors de la transmission des données du DataManager au serveur GTD et ToodleDo, le DataManager va devoir les adapter au bon format des serveurs. En effet, un certain nombre de concepts diffèrent d'un serveur à l'autre. Par exemple, la notion de notes présente dans le serveur GTD ne l'est pas dans ToodleDo. De plus, il faut considérer différents scénarios possibles en cas de problèmes avec les serveurs :

Gestion des connexions

Comme le montre le diagramme de composant, le DataManager est le seul composant relié aux serveurs. Il va donc devoir gérer les connexions et les transactions et va faire le lien entre la session utilisateur et les deux serveurs. Les couples login/password associés aux serveurs ne sont pas stockés dans ce composant mais dans le composant ConnectionManager.

Le serveur ToodleDo n'est plus disponible

Si jamais le serveur ToodleDo venait à être indisponible, le DataManager doit prévenir le composant ProcessManager, qui lui même informera le GUI. Etant donné que ToodleDo ne dispose pas de toutes les fonctionnalités de GTD, l'utilisateur ne constatera pas de différence sur la GUI. Une synchronisation ultérieure sera nécessaire pour conserver une cohérence des données entre le serveur GTD et ToodleDo.

Le serveur GTD n'est plus disponible

Les fonctionnalités GTD étant plus nombreuses, en cas d'indisponibilité du serveur GTD, l'affichage des données sera limitée sur la GUI. En effet, les données, bien que génériques, seront limitées aux fonctionnalités du serveur ToodleDo. En cas d'indisponibilité des deux serveurs, le Data manager l'informera au ConnectionManager.

15.5 Composant ProcessManager

Le processManager est le composant dédié au traitement métier de l'application.Il fait parti du WebServer et intervient une fois que le ConnectionManager a établit la connexion à un serveur GTD et/ou ToodleDo. Son rôle est donc d'effectuer les calculs et vérification. Ses fonctionnalités sont les suivantes :

- CRUD Idées
- CRUD Tâches
- CRUD Projet
- CRUD Notes
- CRUD Contexte
- Vérification des contraintes métier avant chaque opération
- Synchronisation des données

Couche métier de l'application

Le ProcessManager contient toute la couche métier de l'application. A un niveau plus générique, elle correspond au données stockées dans le serveur GTD. Ces classes seront cependant amenées à 'voyager' entre les composants car elles seront utilisées comme conteneur de données dans les échanges.

Précondition à chaque opération

Avant chaque opération demandée à la couche métier, il est nécessaire de vérifier que les modifications pourront être faites sur les serveurs distants. Pour ce faire, le ProcessManager peut vérifier que la session est utilisable via l'interface qui la relie au ConnectionManager, à savoir IConnManager.

Execution d'une opération

Le processManager est le coeur métier de l'application, il se situe donc entre l'ihm et les données. Il reçoit donc les appels du WebClient, modifie sa représentation interne des donneés puis demande la modification au DataManager. Cette chaine d'execution est la même pour chaque opération. En effet, l'application travaille directement avec les serveur GTD et ToodleDo, il n'est en effet pas concevable de fournir une application mise à disposition sur un serveur Web, sans internet. Le mode non connecté n'est donc pas recevable ; la couche données de l'application est donc le serveur GTD ou ToodleDo lui même.

Synchronisation des données

Le composant processManager permet également d'assurer la synchronisation entre les serveurs GTD et ToodleDo. C'est lui qui prend en charge la mise à jour des données d'un serveur vers l'autre. Pour cela, il va notamment utiliser les dates de modification des entités.

Chapitre 16

Interactions

Dans le but de décrire la collaboration entre les composants et les contraintes de ceux-ci, il est important de montrer quelques exemples de scénarios :

- Connexion au serveurs
- Manipulation de données
- Synchronisation entre les serveurs

16.1 Connexion au serveurs

Le scénario suivant mets en évidence la connexion du clientWeb aux serveurs GTD et ToodleDo. La gestion des connexions est assurée par le composant ConnectionManager. C'est également lui qui va gérer les sessions utilisateurs. Comment on peut le voir, l'objet session va stocker les informations relatives aux connexions sur les serveurs GTD et ToodleDo (couple login/password notamment). Celui ci sera envoyé au DataManager qui va se connecter aux serveurs GTD et ToodleDo.

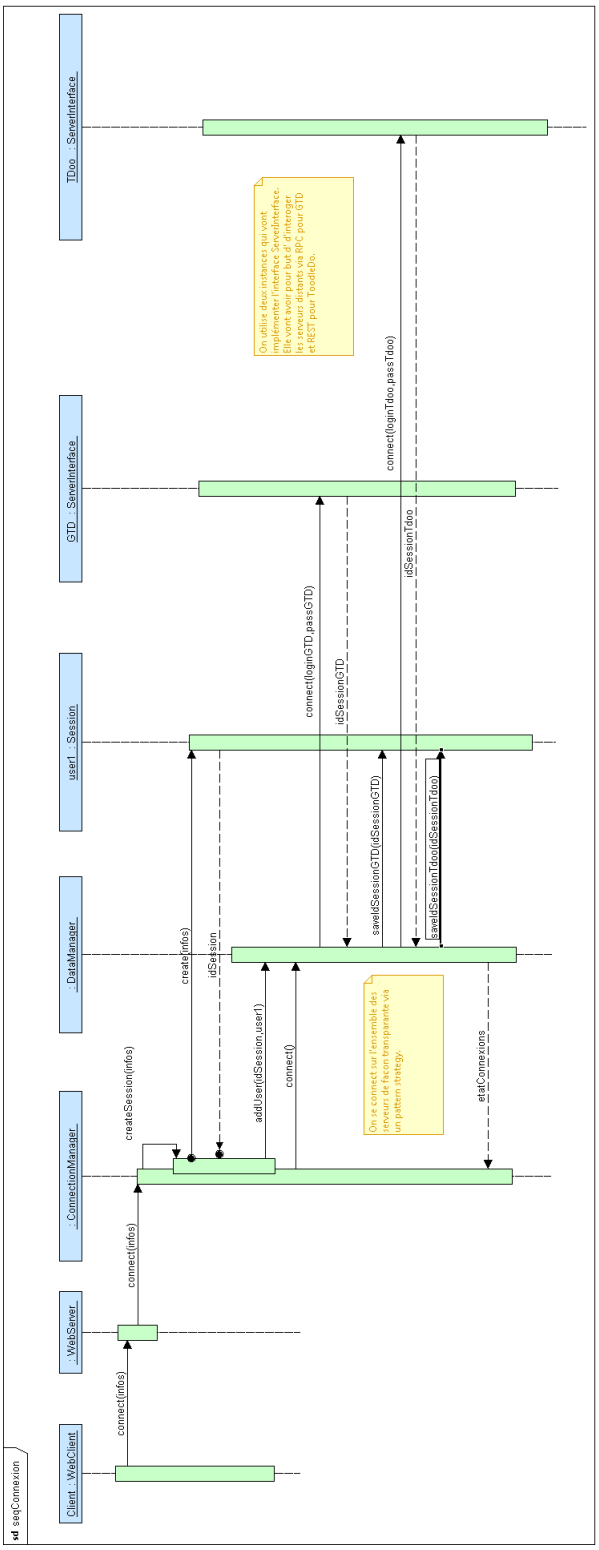

FIG. 16.1 – UML - Diagramme de séquences - Connexion au serveurs

16.2 Affichage des projets

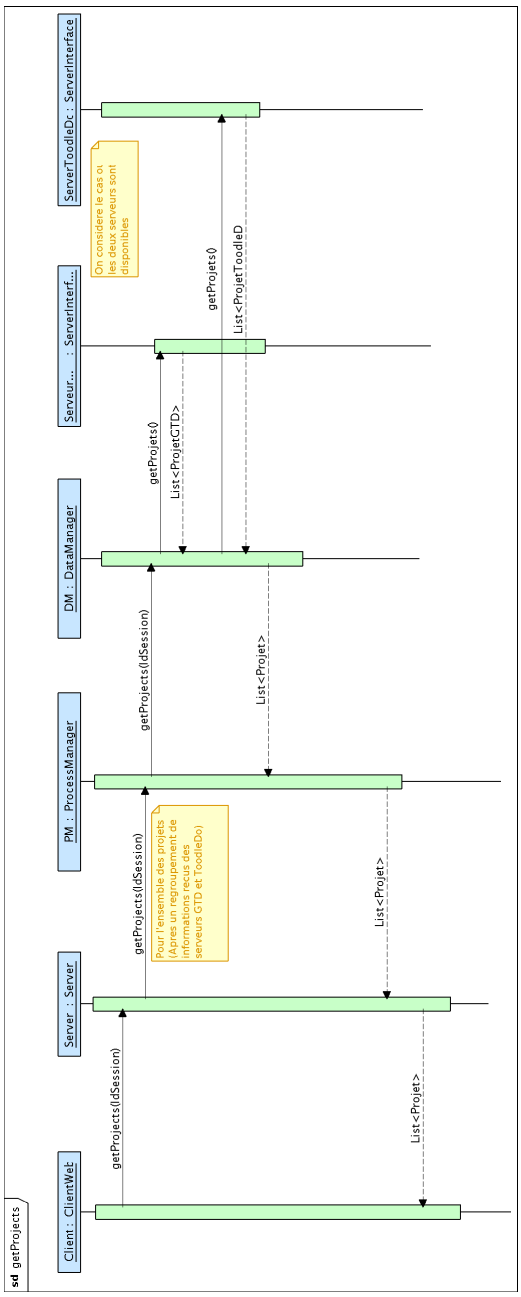

FIG. 16.2 – UML - Diagramme de séquences - Affichage des projets

16.3 Synchronisation

Le processus de synchronisation permet d'avoir des données identiques sur les serveurs GTD et ToodleDo (après une indisponibilité d'un serveur par ex). Dans le scénario suivant, l'utilisateur récupère une liste de projets. Le DataManager qui assure la transparence lors de la récupération des projets, s'aperçoit que les listes sont différentes (il manque un projet sur le serveur ToodleDo). Il va donc ajouter le projet manquant sur le serveur ToodleDo. En ce qui concerne la synchronisation il faut prendre en considération que le serveur GTD est le serveur maitre de l'application.

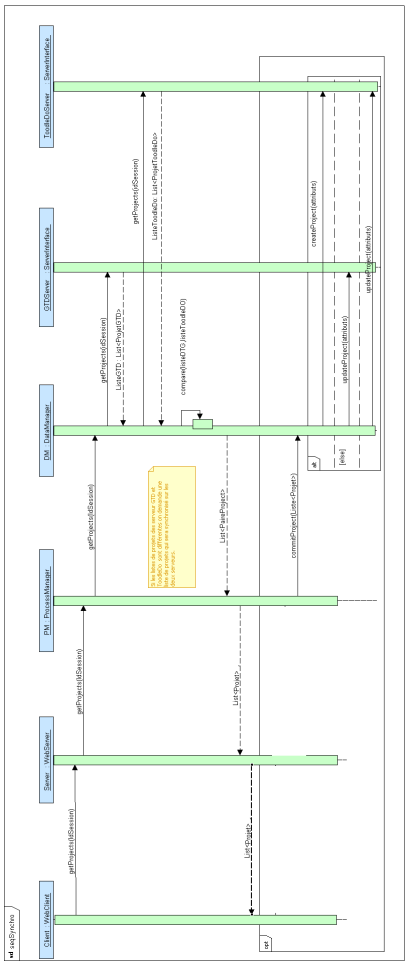

FIG. 16.3 – UML - Diagramme de séquences - Synchronisation

Chapitre 17

Spécification des interfaces

17.1 Composant ProcessManager

Ce composant représente la couche métier de notre application, le coeur de la méthode GTD avec ses concepts et relations. C'est celui-ci qui s'assurera de la création, modification, suppression et cohérence des données.

17.1.1 Interface IProcessManager

- **createTask(session : Session, task : Task) : boolean**
  Créée la tache task dans la couche métier du ProcessManager et demande la création au DataManager sur le Serveur GTD et/ou ToodleDoo.

  ---

  context IProcessManager::createTask(session : Session, task : Task) : boolean
  pre:
  idSession != null **and** idSession != "" **and**
  task != null **and**
  task.**name** != "" **and**
  task.**context** != null **and**
  task.dateBegin <= task.dateFinish **and**
  task.effortRate >= 0 **and**
  task.effortRate <= 99 **and**
  task. priorite >= 1 **and**
  task. priorite <= 5 **and**
  task.time <= (task.dateFinish − task.dateBegin)

- **updateTask(session : Session, task : Task) : boolean**
  Mets à jour la tâche identifiée par l'instance task dans la couche métier du ProcessManager et demande la mise à jour au DataManager sur le Serveur GTD et/ou ToodleDoo. Faire attention que dans une séquence de tâche, il n'y ait pas deux fois la même. Renvoie vrai si la mise à jour a pu être effectuée.

  ---

context IProcessManager::updateTask(session : Session, task : Task) : **Boolean**
pre:
task != null **and** idSession != null **and** idSession != ""
post :
result = task.project−>empty() **implies** task.previousTask−>empty() **and**
task.previousTask−>**notEmpty**() **implies** task.previousTask != task **and**
task.project−>**notEmpty**() **implies** task.project = task.previousTask.project **and**
task.dateStop >= task.repeat.dateFinish

- **delTask(session : Session, taskId : String) : boolean**
Supprime la tâche identifiée par l'identifiant taskId dans la couche métier du ProcessManager et demande la supression sur le serveur GTD et/ou ToodleDoo au DataManager. Il faudra gérer le cas où la tâche elle même précède une autre... Renvoie vraie si la suppression a pu être effectuée.
- **getTask(session : Session, taskId : String) : Task**
Renvoie l'instance de Task (niveau metier) représentant la tâche identifiée par l'id taskId.
- **getTasks(session : Session) : List<Paire<Task>>**
Permet le récupérer l'ensemble des taches.
- **createNote(session : Session, note : Note) : boolean**
Créée la note note dans la couche métier du ProcessManager et demande la création au DataManager sur le Serveur GTD et/ou ToodleDoo.

---

context IProcessManager::createNote(session : Session, note : Note) : **Boolean**
pre:
Note.content != null **and** Note.content != ""

- **updateNote(session : Session, note : Note) : boolean**
Mets à jour la note identifiée par l'instance note dans la couche métier du ProcessManager et demande la mise à jour au DataManager sur le Serveur GTD et/ou ToodleDoo.

---

context IProcessManager::updateNote(session : Session, note : Note) : **Boolean**
pre:
Note.content != null **and** Note.content != ""

- **delNote(session : Session, noteId : String) : boolean**
Supprime la note identifiée par l'identifiant noteId dans la couche métier du ProcessManager et demande la supression sur le serveur GTD et/ou ToodleDoo au DataManager.
- **getNote(session : Session, noteId : String) : Note**
Renvoie l'instance de Note (niveau metier) représentant la note identifiée par l'id noteId.
- **getNotes(session : Session) : List<Paire<Note>>**
Permet le récupérer l'ensemble des notes.
- **createProject(session : Session, project : Project) : boolean**

Créée le projet project dans la couche métier du ProcessManager et demande la création au DataManager sur le Serveur GTD et/ou ToodleDoo. Le **Set(Task)** implique qu'un projet ne contiendra pas plusieurs fois la même tâche, de même pour les sous projets.

```
context IProcessManager::createProject(session : Session, project : Project) : boolean
pre:
        idSession != null and idSession != "" and
        project != null and
        project.tasks->isSet() and
        project.childProjects->isSet()
```

- **updateProject(session : Session, project : Project) : boolean**
  Mets à jour le projet identifié par l'instance project dans la couche métier du ProcessManager et demande la mise à jour au DataManager sur le Serveur GTD et/ou ToodleDoo. Renvoie vrai si la modification a été faite avec succès, par exemple il faudra vérifier que la hiérarchie de projets/sous-projets est bien un arbre.

```
context IProcessManager::updateProject(session : Session, p : Project) : boolean
pre:
        idSession != null and idSession != "" and
        p != null
post:
        result = p.defaultContext->notEmpty() implies
                (p.tasks->forAll(each | each.context.properties->includesAll(self.defaultConte
                p.childProjects->forAll(each | each.context.properties->includesAll(p.defaultC
```

- **delProject(session : Session, projectId : String) : boolean**
  Supprime le projet identifié par l'identifiant projectId dans la couche métier du ProcessManager et demande la supression sur le serveur GTD et/ou ToodleDoo au DataManager. Les éventuelles tâches du projet deviendront indépendante, on veillera aussi que le projet ne contient pas de sous-projets. Renvoie vrai si la suppression a marchée.
- **getProject(session : Session, projectId : String) : Project**
  Renvoie l'instance de Project (niveau métier) représentant le projet identifié par l'id projectId.
- **getProjects(session : Session) : List<Paire<Project>>**
  Permet le récupérer l'ensemble des projets et leurs sous projets.
- **createContext(session : Session, context : Context) : boolean**
  Créée le contexte context dans la couche métier du ProcessManager et demande la création au DataManager sur le Serveur GTD et/ou ToodleDoo.

```
context IProcessManager::createContext(session : Session, context : Context) : boolean
pre:
        idSession != null and idSession != "" and
        context.properties->notEmpty()
```

- **updateContext(session : Session, context : Context) : boolean**
  Mets à jour le contexte identifié par l'instance project dans la couche métier du ProcessManager et demande la mise à jour au DataManager sur le Serveur GTD et/ou ToodleDoo.
- **delContext(session : Session, contextId : String) : boolean**
  Supprime le contexte identifié par l'identifiant projectId dans la couche métier du ProcessManager et demande la suppression sur le serveur GTD et/ou ToodleDoo au DataManager.
- **getContext(session : Session, contextId : String) : Context**
  Renvoie l'instance de Context (niveau métier) représentant le contexte identifié par l'id contextId.
- **getContexts(session : Session) : List<Paire<Context>**
  Renvoie l'ensemble des contextes.
- **createIdea(session : Session, idea : Idea) : boolean**
  Créer l'idée idea dans la couche métier du ProcessManager et demande la création au DataManager sur le Serveur GTD et/ou ToodleDoo.

  context IProcessManager::createIdea(session : Session, idea : Idea) : boolean
  pre:
  idSession != null **and** idSession != "" **and**
  idea.content != ""

- **updateIdea(session : Session, idea : Idea) : boolean**
  Mets à jour l'idée identifiée par l'instance idea dans la couche métier du ProcessManager et demande la mise à jour au DataManager sur le Serveur GTD et/ou ToodleDoo.
- **delIdea(session : Session, ideaId : String) : boolean**
  Supprime l'idée identifiée par l'identifiant ideaId dans la couche métier du ProcessManager et demande la suppression sur le serveur GTD et/ou ToodleDoo au DataManager.
- **getIdea(session : Session, ideaId : String) : Idea**
  Renvoie l'instance de Idea (niveau metier) représentant l'idée identifiée par l'id ideaId.
- **getIdeas(session : Session) : List<Paire<Idea>>**
  Renvoie l'ensemble des contextes

17.1.2 Interface IConnManager

Cette interface permet de communiquer avec le composant DataManager. C'est notamment à travers elle que les demandes de connexion et déconnexion seront effectuées, afin de garantir un niveau de transparence et de généricité.

- **connect(String loginGTD,String passGTD, String loginTdoo, String pass Tdoo) :**
  Vérifie les couples utilisateurs/mot de passes auprès des serveurs (GTD et ToodleDo) et établit la connexion. Retourne null si erreur ou l'id de la session si connecté.

- **disconnect(String session) : void**
  Permet de déconnecter l'utilisateur, en supprimant la session du GTDWebServer.
- **isAlive(String session) : boolean**
  Renvoie true si la session identifiée par la chaîne de caractères 'session' est active, false sinon.
- **createAccount(String login,String pass) : boolean**
  Renvoie true si la création de compte a réussi, false sinon.
- **delAccount(String login) : boolean**
  Renvoie true si la suppression de compte a réussi, false sinon.
- **updateAccount(String session,String login,String pass) : boolean**
  Renvoie true si la modification des informations a réussi, false sinon.

17.2 Composant ConnectionManager

17.2.1 Interface IConnectionCheck

Afin de permettre au composant ProcessManager de vérifier si la session est toujours active sur les serveurs GTDWebServer, ToodleDo et GTDServer, la méthode suivante est définie :

- **isAlive(String session) : boolean**
  Renvoie true si la session identifiée par la chaîne de caractères 'session' est active, false sinon.

17.2.2 Interface IConnManager

Cette interface permet de communiquer avec le composant WebClient, elle est déléguée au WebServer. C'est notamment à travers elle que les demandes de connexion et déconnexion seront effectuées. Afin de garantir un niveau de transparence et de généricité

- **connect(String loginGTD,String passGTD, String loginTdo, String pass Tdo) : St**
  Vérifie les couples utilisateurs/mot de passes auprès des serveurs (GTD et ToodleDo) et établit la connexion. Retourne null si erreur ou l'id de la session si connecté.
- **disconnect(String session) : void**
  Permet de déconnecter l'utilisateur, en supprimant la session du GTDWebServer.
- **isAlive(String session) : boolean**
  Renvoie true si la session identifiée par la chaine de caractère 'session' est active, false sinon.

17.3 Composant DataManager

17.3.1 IToDataManager

Le but de cette interface est d'offrir la possibilité d'enregistrer les données du composant ProcessManager sur les serveurs ToodleDo et GTD de façon transparente. On va ainsi avoir les méthodes suivantes permettant la persistance des données :

- **createTask(Session session ,Task task) : boolean**
  Permet la création d'une tâche sur les serveurs GTD et ToodleDo. Renvoie false si la création a échouée true dans le cas contraire.
- **updateTask(Session session,Task task) : boolean**
  Permet la modification d'une tâche sur les serveurs GTD et ToodleDo. Renvoie false si la modification a échouée true dans le cas contraire.
- **removeTask(Session session,Task task) : boolean**
  Permet la suppresion d'une tâche sur les serveurs GTD et ToodleDo. Renvoie false si la suppression a échouée true dans le cas contraire.
- **getTasks(Session session) : List<Paire<Task>>**
  Permet le récupérer l'ensemble des taches.
- **getTask(Session session ,String taskId) : Task**
  Permet le récupérer la tâche identifiée par l'id taskId.
- **commitIdea(Session session,List<Task>) : boolean**
  Permet de renvoyer les tâches à synchroniser après les avoir choisis sur le GUI.
- **createProject(Session session ,Project project) : boolean**
  Permet la création d'un projet sur les serveurs GTD et ToodleDo. Renvoie false si la création a échouée true dans le cas contraire.
- **updateProject(Session session,Project project) : boolean**
  Permet la modification d'un projet sur les serveurs GTD et ToodleDo. Renvoie false si la modification a échouée true dans le cas contraire.
- **removeProject(Session session,Project project) : boolean**
  Permet la suppression d'une projet sur les serveurs GTD et ToodleDo. Renvoie false si la suppression a échouée true dans le cas contraire.
- **getProjects(Session session) : List<Paire<Project>>**
  Permet le récupérer l'ensemble des projets et leurs sous projets.
- **getProject(Session session ,String projectId) : Project**
  Permet le récupérer le projet identifié par l'id projectId.
- **commitIdea(Session session,List<Project>) : boolean**
  Permet de renvoyer les projets à synchroniser après les avoir choisis sur le GUI.
- **createNote(Session session ,Note note) : boolean**
  Permet la création d'une note sur le serveurs GTD. Renvoie false si la création a échouée true dans le cas contraire.
- **updateNote(Note note) : boolean**

Permet la modification d'une note sur le serveurs GTD. Renvoie false si la modification a échouée true dans le cas contraire.
- **removeNote(Session session,Note note) : boolean**
  Permet la suppression d'une note sur le serveurs GTD. Renvoie false si la suppression a échouée true dans le cas contraire.
- **getNotes(Session session) : List<Paire<Note>>**
  Permet le récupérer l'ensemble des notes.
- **getNote(Session session,String noteId) : Note**
  Permet le récupérer la note identifiée par l'id noteId.
- **commitNote(Session session,List<Note>) : boolean**
  Permet de renvoyer les notes à synchroniser après les avoir choisis sur le GUI.
- **createContext(Session session,Context context) : boolean**
  Permet la création d'un contexte sur le serveurs GTD. Renvoie false si la création a échouée true dans le cas contraire.
- **updateContext(Session session,Context context) : boolean**
  Permet la modification d'un contexte sur le serveurs GTD. Renvoie false si la modification a échouée true dans le cas contraire.
- **removeContext(Session session,String contextId) : boolean**
  Permet la suppression d'un contexte sur le serveurs GTD. Renvoie false si la suppression a échouée true dans le cas contraire.
- **getContexts(String sessionId) : List<Paire<Context>**
  Renvoie l'ensemble des contextes.
- **getContext(Session session,String contextId) : Context**
  Permet de récupérer le contexte identifié par l'id contextId.
- **commitContext(Session session,List<Context>) : boolean**
  Permet de renvoyer les context à synchroniser après les avoir choisis sur le GUI.
- **createIdea(Session session,Idea idea) : boolean**
  Permet la création d'une idée sur le serveurs GTD. Renvoie false si la création a échouée true dans le cas contraire.
- **updateIdea(Session session,Idea idea) : boolean**
  Permet la modification d'une idée sur le serveurs GTD. Renvoie false si la modification a échouée true dans le cas contraire.
- **removeIdea(Session session,String ideaId) : boolean**
  Permet la suppresion d'une idée sur le serveurs GTD. Renvoie false si la suppresion a échouée true dans le cas contraire.
- **getIdeas(String contextId) : List<Paire<Idea>>**
  Renvoie l'ensemble des contextes
- **getIdea(Session session,String ideaId) : Idea**
  Permet le récupérer l'idée identifiée par l'id ideaId.
- **commitIdea(Session session,List<Idea>) : boolean**
  Permet de renvoyer les idées à synchroniser après les avoir choisis sur le GUI.

17.3.2 IConnectToServers

- **connect(Session session) : boolean**
  Permet de connecter la session utilisateur aux serveurs GTD et ToodleDO.
- **disconnect(Session session) : boolean**
  Permet de deconnecter la session associée aux serveurs GTD et ToodleDO
- **createAccount(Session session) : boolean**
  Permet la création d'un compte sur les serveurs GTD et ToodleDO
- **deleteAccount(Session session) : boolean**
  Permet la suppression des comptes associées aux serveurs GTD et ToodleDO

Chapitre 18

Spécification des types utilisés

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

18.1 Session

La classe **Session** va permettre de stocker des informations relatives au serveurs GTD et ToodleDo . Elle possède les attributs suivants :

IdSession : l'id de la session utilisateur

loginGTD : le login du serveur GTD

passwordGTD : le mot de passe du serveur GTD

IdSessionGTD : l'id de la session utilisateur sur le serveur GTD

loginToodleDo : le login du serveur ToodleDo

passwordToodleDo : le mot de passe du serveur ToodleDo

IdSessionToodleDo : l'id de la session utilisateur sur le serveur ToodleDo

18.2 Task

La classe **Task** va permettre de stocker des informations relatives à une tâche. Elle possède les attributs suivants :

name : nom de la tâche.

dateBegin : date de début de la tâche.

dateFinish : date d'échéance de la tâche.

priorite : priorite de la tache.

rateEffort : taux d'effort de la tâche.

time : durée de la tâche.

state : état de la tache EnAttente,Afaire,Finie,Déléguée.

previousTask : la tâche précédant (à réaliser avant) de la tâche.

notes : notes associées à la tâche.

repeat : la répétition de la tache Mensuelle,Hebdomadaire,Journalière

context : le contexte nécessaire à la réalisation de la tâche.

18.3 Project

La classe **Project** va permettre de stocker des informations relatives à un Projet. Elle possède les attributs suivants :

name : nom du projet.

defaultContext : le connexte associé au projet.

tasks : la liste des tâches du projet.

parentProject : le projet parent du projet.

childProjects : la liste des sous projets du projet.

notes : notes associées au projet.

18.4 Context

La classe **Context** spécifie par exemple l'environnement, le lieu, les outils avec lesquels la tâches ou le projet seront effectués. Elle possède les attributs suivants :

properties : propriétés du contexte (lieu, environnement, outils, ...).

18.5 Idea

Idea modélise les idées que pourrait avoir l'utilisateur et qu'il lui semble important de noter. En réalité ce sera juste une chaîne de caractères.

content : idée.

Chapitre 19

Conclusion

Cette troisième partie décrit la répartition des fonctionnalités du système à un niveau composant. La description de leurs interfaces permet de définir la façon dont ils vont communiquer, et précise le fonctionnement du système. Ce livrable permet enfin d'avoir une vision plus précise de la façon dont l'application doit être implémentée, notamment par les différents diagramme de scénario. Une analyse plus précise de chacun des composants sera effectuée dans la prochaine partie, en se focalisant sur leur conception précise. Ce raffinage va donc s'appuyer sur ce livrable, en tenant compte des fonctionnalités offertes par le serveur GTD.

Quatrième partie

Livrable 4 et 5

Chapitre 20

Introduction

Dans ce dernier livrable, nous allons parler de la conception détaillée de l'application. On expliquera essentiellement la partie serveur de celle-ci puisque la partie client ne se compose que d'une interface qui sera traitée dans le rapport destiné au module d'IHM. De même, nous n'avons pas de base de données, notre application étant accessible en ligne, il aurait été inutile de dupliquer le serveur GTD.
On rappelle que notre application Web utilise GWT pour l'interface et assure la synchronisation entre deux serveurs GTD : celui développé dans le cadre de ce projet multi-modules et le serveur propriétaire Toodledo.

20.1 Organisation du chapitre

Dans un premier temps, on parlera brièvement de l'architecture physique de l'application. Puis dans un second, on justifiera les différentes décisions de conceptions qui ont été prise. Puis dans un troisième temps, on détaillera la conception finale de notre serveur Web GTD. Puis, pour finir, on présentera quelques scripts de génération de code.

Chapitre 21

Architecture physique

Ici le Client Web s'exécute sur n'importe quel navigateur internet chez l'utilisateur, donc sur sa machine. Cette interface communique avec le Serveur Web via internet et le protocole HTTP (cf. rapport d'Objets distribués pour plus de détails). Le serveur, quant à lui, est exécuté sur une autre machine toujours accessible via internet. Pour des soucis de performances, celle-ci sera assez puissante avec de la mémoire vive en conséquence et disposera d'une bande passante suffisante. Notre serveur Web communiquera avec le Serveur GTD via RMI, ce-dernier pourra être stocké sur la même machine que le notre ou sur une autre distante. Enfin, il comminiquera avec le Serveur ToodleDo via le protocole REST.

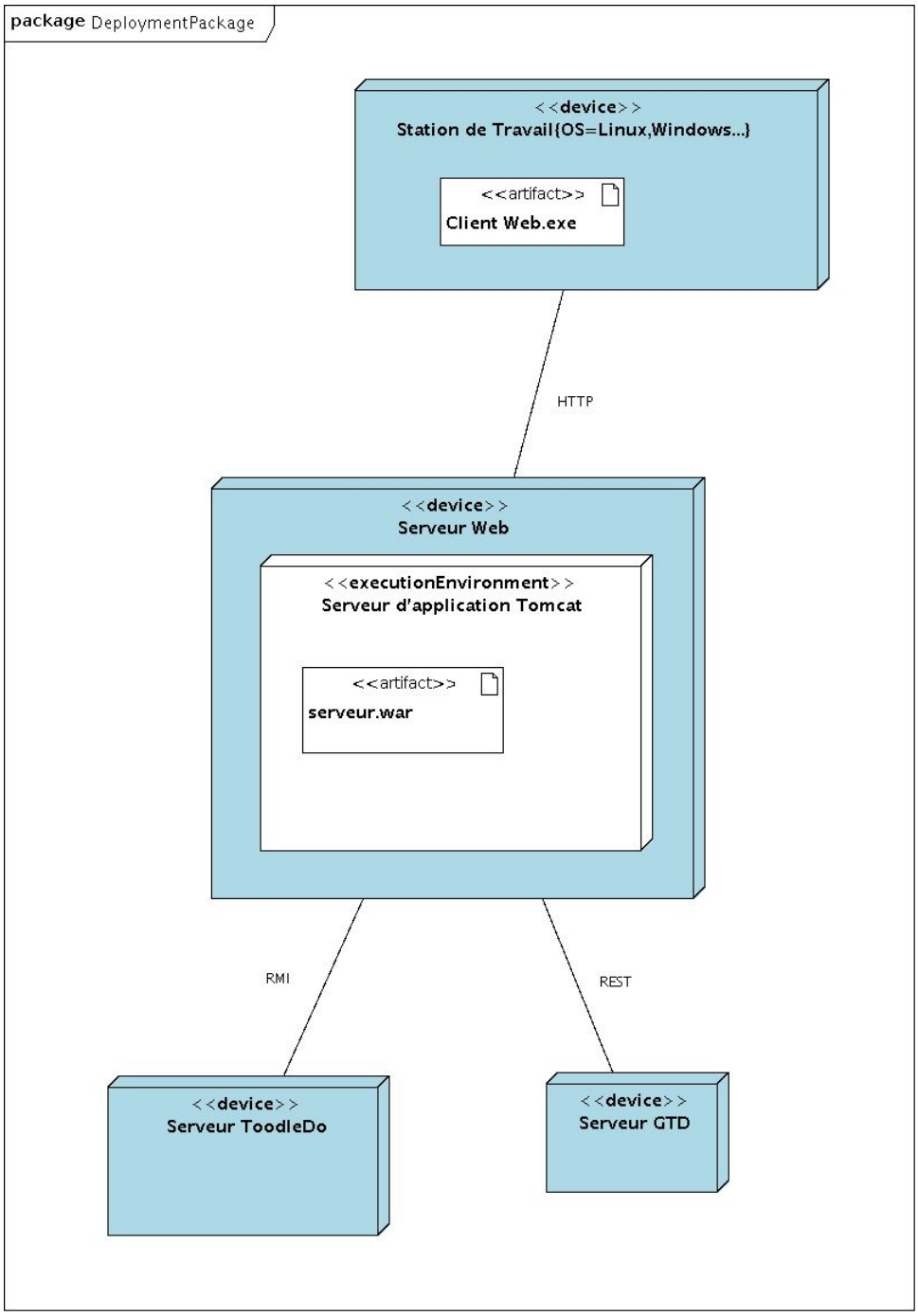

FIG. 21.1 – Diagramme UML de déploiement

Chapitre 22

Décisions de conceptions

Dans l'ensemble, nous sommes restés assez fidèle à nos idées de départ.

22.1 Communication

Le groupe développant le serveur GTD a fait le choix de communiquer avec les autres serveurs de manière asynchrone grâce à un système de CallBack. Le problème est que la communication entre nos composants se fait de manière synchrone, il a donc fallu mettre au point un système permettant de mettre en attente le thread effectuant la requête jusqu'à que celui-ci soit réveillé par l'appel de notre CallBack.

22.2 Synchronisation

Notre application faisant appel à deux serveurs GTD différents, chacun étant accessible par leur propre interface, quand l'utilisateur utilise la notre, il est nécessaire de synchroniser les données. Pour ce faire, nous nous sommes inspirés du modèle SVN, c'est à dire que lorsque par exemple l'utilisateur listera tous les projets, le serveur Web récupèrera tous les projets de chaque serveur GTD dans deux listes distinctes. Celles-ci seront remontées jusqu'à l'interface et ce sera à l'utilisateur de choisir quel projet garder parmi ceux qui diffèrent entre les 2 listes. Une fois les conflits réglés, les choix seront répercutés sur les serveurs.

22.3 Exceptions

Chaque exception levée (RMI, mauvais identifiants, Serveur GTD via le onFailure du CallBack) est remontée jusqu'à l'interface afin de prévenir l'utilisateur de tout problème éventuel.

22.4 Contraintes OCL

Les contraintes OCL spécifiées dans les précédents livrables sont traitées le plus tôt possible, c'est à dire au niveau de l'interface. Celle-ci guide suffisamment l'utilisateur pour éviter que ce-dernier ne saisisse par exemple une mauvaise valeur ou qu'une tâche ne dépende d'elle même.

Chapitre 23

Conception détaillée

23.1 Diagramme de classe

Comme notre architecture est assez modeste, nous n'avons pas utilisé de conteneurs de composants ou autres IOC, tout est centralisé au niveau de la classe statique WebServerManager. Celle-ci récupère les instances de chaque composant et les fournies aux autres, chacun ne peut être instancié qu'une seule fois (patron de conception "Singleton"). Cela permet une certaine indépendance entre les composants qui sont obligés de passer par la configuration (WebServerManager, cf. cours de composant), ceci afin de respecter le plus possible les concepts de la programmation par composant et de bénéficier de ses avantages (variabilités, abstraction de l'implémentation réelle, etc...). De même, l'implémentation utilise au maximum les interfaces, l'approche "mature", pour améliorer la lecture, la modularité, la maintenabilité, la variabilité, l'abstraction... du code.

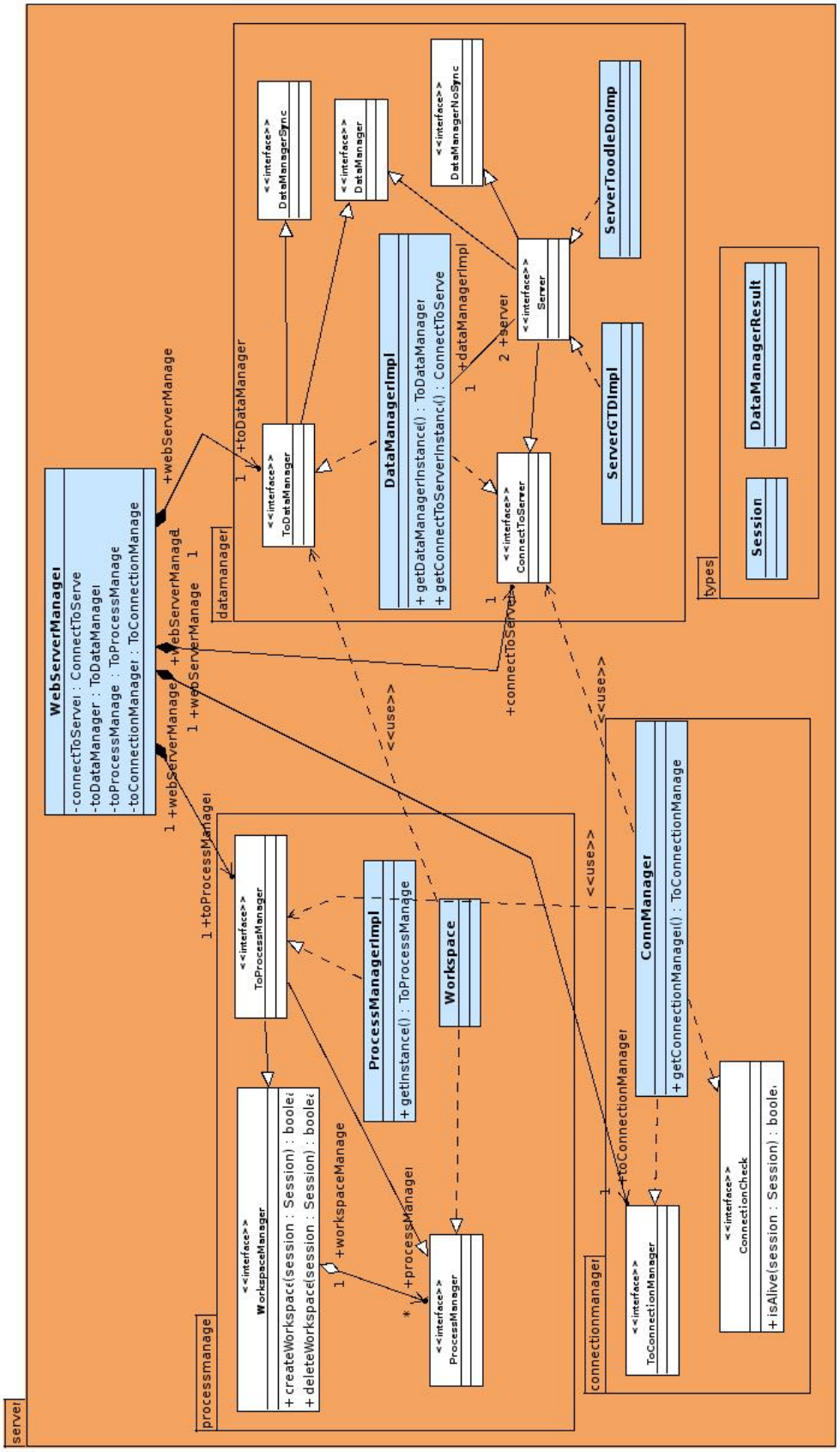

FIG. 23.1 – Diagramme UML de classes

Hormis les méthodes spécifiées dans ce diagramme de classe, toutes les autres méthodes offertes par l'interface ServeurRMI, fournie par le Serveur GTD, sont réparties ainsi :

- CRUD sur les types de bases avec synchronisation : ProcessManager, ToProcessManager, ToDataManager
- CUD sur les types de bases : DataManager
- R avec synchronisation : DataManagerSync
- R sans synchronisation : DataManagerNoSync
- connexion/déconnexion de l'utilisateur : ToConnectionManager, ConnectToServer, Server
- CRUD sur les types de bases sans synchronisation : Server

A préciser qu'un Workspace correspond à un utilisateur. Celui-ci stocke en local les données concernant l'utilisateur afin de limiter les accès aux serveur ce qui permet indirectement d'augmenter la réactivité de l'interface.

23.2 Types de bases

Les interfaces des types de bases ainsi que des implémentations de celles-ci sont fournies par l'équipe du serveur GTD. Dans notre cas, nous avons dû spécialiser ces implémentations pour pouvoir affecter nous même un identifiant à chacun de ces types nécessaire à ToodleDo, contrairement au ServeurGTD. On a pu aussi remarquer que ToodleDo ne dispose pas de tous nos concepts. Par exemple le taux d'effort pour une tâche et les idées ne sont pas présents, ces fonctionnalités ne sont donc tout simplement pas implémentées par ServeurToodleDO et ne sont pas prises en compte lors de la synchronisation. On rappelle que les types de bases sont :

- Projet
- Tâche
- Contexte
- Tag
- Idée

A ceux-ci, nous avons ajouté les types Session et DataManagerResult. Le premier permet de stocker les logins et mots de passes de l'utilisateur pour les deux serveurs ainsi que l'identifiant de session pour le serveur GTD et le token pour le serveur ToodleDo. Ces deux éléments sont renvoyés par leur serveur respectif lorsque l'authentification a réussi. Ils permettent d'identifier l'utilisateur auprès des serveurs et assurent ainsi la sécurité. Un objet Session est instancié quand un utilisateur se connecte et détruit quand celui-ci se déconnecte : l'identifiant et le token ne sont plus valide.

Le second type, DataManagerResult assure la synchronisation entre les deux serveurs. Il contient deux listes génériques, lorsque la lecture d'un type sur les deux serveurs est remontée jusqu'à DataManagerImpl, les deux listes obtenues sont stockées dans un objet DataManagerResult qui sera remonté jusqu'à l'interface.

Chapitre 24

Scripts de génération de code

Du fait de notre architecture, le corps des méthodes CRUD au sein d'une classe qui implémente Serveur RMI sont quasi identiques, seul le nom des données change... voici un exemple :

```
<%for (interfaceRealization.contract.ownedOperation){%>
<%for (ownedParameter[direction == "in"]){%>
* @param <%name%>
<%}%>
<%for (ownedParameter[direction == "return"]){%>
* @return <%resolveType%>
<%}%>
*/
<%visibility%> synchronized <%ownedParameter[direction == "return"].resolveType%> <

CallBackSync<String> cb = new CallBackSync<String>();
//TODO
this.wait();
if (cb.getException() != null) {
throw cb.getException();
}
return cb.getValue();
}
<%}%>

<%script type="Parameter" name="resolveType"%>
<%if (upper == -1){%>List<<%type.name%>><%}else{%><%type.name%><%}%>
```

On peut remarquer le système de synchronisation entre le CallBack et l'appel de méthode. Ici on passe d'une communication asynchrone à une communication synchrone :

lorsque le serveur appel une des méthode du CallBack, celle-ci "notify" (réveille) l'objet appelant qui s'était mis en attente à l'instruction "wait".

La plupart des autres règles de générations de code se base sur la même structure de script, seul le corps des méthodes change. Pour gérer les importations de classes, voici le script utilisé :

```
<%script type="Class" name="gestImport"%>
<%-- superclasse --%>
<%if (superClass != null && superClass.package.name != package.name){%>
<%(superClass.package.name+"."+superClass.name).addToImport()%>
<%}%>
<%-- interfaces --%>
<%for (interfaceRealization){%>
<%(contract.package.name+"."+contract.name).addToImport()%>
<%}%>
<%-- attributs --%>
<%for (attribute){%>
<%if (type.package.name != "UMLPrimitiveTypes"){%>
<%(type.package.name+"."+type.name).addToImport()%>
<%}%><%}%>
<%-- methodes --%>
<%for (ownedOperation){%>
<%for (ownedParameter){%>
<%if (type.package.name != "UMLPrimitiveTypes"){%>
<%(type.package.name+"."+type.name).addToImport()%>
<%}%><%}%><%}%>
```

Chapitre 25

Conclusion

Ce projet aura été assez difficile à mener à terme. Bien qu'intéressant car celui-ci faisait appel à différentes technologies et nous permettait de mettre en pratique les divers modules enseignés ce dernier semestre, un certain manque de coordination entre les différents groupes et un manque d'expertise dans le domaine sont venus ralentir la progression.// En effet, il s'est finalement révélé que notre partie n'était en fait qu'une interface Web, aucune base de données local, aucune persistance, la partie métier de notre serveur était relativement vide. Notre développement, hormis l'interface, dépendait donc très fortement du serveur GTD central qui devait s'accorder avec les différents groupes qui n'avaient pas forcément tous la même vision des choses. Afin de rajouter de la consistance à notre serveur, nous avons inclue le serveur propriétaire ToodleDo qui possède déjà sa propre interface mais qui fournie les API nécessaires pour l'utiliser par nos propres moyens. Le problème a été que tous nos concepts n'étaient pas présent et qui fallait assurer une synchronisation entre les deux serveurs, rajoutant de la difficulté supplémentaire.// Malgré tout, cela a été une bonne expérience et nous a montré que c'est loin d'être évident de mener à bien un projet réparti en plusieurs groupes de travail tout en respectant le cahier des charges et les contraintes de temps imposées.

Table des figures