École de technologie supérieure

Rapport Technique

présenté à l'École de technologie supérieure
dans le cadre du cours LOG792
Projet de fin d'études en génie logiciel

Incidents

Outil simplifié de suivi et d'application des processus de développement logiciel en TPO

Anthony Plourde
PLOA12078800

Département de génie logiciel et des TI

Professeur superviseur
Alain April

Montréal, 19 décembre 2011
Automne 2011

Remerciements

Merci à Luc Vandal d'Edovia d'avoir cru en ce projet.
Son appui et sa participation ont permis la réalisation d'*Incidents*.

Incidents

Outil simplifié de suivi et d'application des processus de développement logiciel en TPO

Anthony Plourde
PLOA12078800

Résumé

Le logiciel *Incidents* est un outil permettant aux développeurs de chez Edovia de toujours savoir ce qu'ils ont à faire, en un seul clic. Que ce soit de répondre à des interventions des clients, de corriger une anomalie, d'implémenter une nouvelle fonctionnalité ou encore d'effectuer une revue de code, le logiciel guide intuitivement le développeur à travers les différentes étapes du développement. L'outil permet de réduire les pertes de temps accordés à la gestion du processus de développement en rassemblant à un seul endroit, toutes les facettes principales du processus de développement typique des très petites entreprises comme Edovia.

Historique des révisions

Date	Description	Révision	Auteur
06-12-2011	Document initial avec table des matières	0.1	Anthony Plourde
07-12-2011	Introduction et chapitres 1 à 4	0.5	Anthony Plourde
08-12-2011	Chapitre 5	0.6	Anthony Plourde
14-12-2011	Chapitres 6 et 7	0.8	Anthony Plourde
16-12-2011	Chapitre 8, conclusion, intégration des annexes et révision finale	1.0	Anthony Plourde

Table des matières

Liste des figures

Liste des tableaux

Glossaire

Terme	Définition
Edovia	Compagnie partenaire de ce projet. La compagnie ouvre dans le développement d'application iOS et Mac
Mac OS X HIG	Mac OS X Humain Interface Guidelines. Ligne directrice à respecter dans les interfaces utilisateurs des applications Mac.
Tender	Service de soutien aux utilisateurs en ligne permettant aux utilisateurs de communiquer avec l'entreprise vendeur.
Talker	Service de clavardage de groupe en ligne avec historique des conversations.
Lighthouse	Service de gestion des billets en ligne permettant la gestion des anomalies à corriger et des tâches à effectuer lors du développement des logiciels.
Beanstalk	Service de dépôt SVN en ligne.
SVN	Subversion. Service de gestion des versions basé sur un concept de dépôt centralisé permettant la collaboration de plusieurs utilisateurs sur un même dépôt.
Requête de soutien	Message provenant d'un client utilisateur d'une application développé par l'entreprise pour demander de l'aide avec l'utilisation d'une application, pour reporter une anomalie ou encore proposer une fonctionnalité. Ce message est reçu via le service de soutien (Tender).
Incident	Entité regroupant les informations d'un billet, d'une requête de soutien et de propagation SVN reliés à une tâche à effectuer ou une anomalie à régler lors du développement logiciel.
Propagation	Effectué lorsque l'on soumet une modification à un dépôt SVN. En anglais, un "commit".
Message de propagation	Message qui accompagne une propagation afin d'indiquer ce qui a été modifié. En anglais, "commit message".
LOG792	Sigle du cours du projet de fin d'études à l'École de technologie supérieure.
ÉTS	École de technologie supérieure. Université, située à Montréal, faisant partie du réseau de l'Université du Québec, se consacrant aux disciplines de l'ingénierie.
PFE	Projet de Fin d'Études. Projet exécuté en fin de Baccalauréat à l'École de technologie supérieure.
Contrôle de version	Logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. Il permet notamment de retrouver les différentes versions d'un lot de fichiers connexes. Les *logiciels de gestion de versions* sont utilisés notamment en ingénierie du logiciel pour conserver le code source relatif aux différentes versions d'un logiciel. (Wikipedia)
Gestionnaire de billet	Logiciel qui permet d'aider les développeurs à suivre un plan de travail établi. Des billets sont alors créés et assignés aux développeurs. Ces billets peuvent représenter une tâche, une fonctionnalité à développer ou une anomalie à corriger dans le cadre du développement logiciel.
Gestionnaire de dépôt	Service permettant la gestion des différents dépôts de contrôle de version.

Terme	Définition
Salle de réunion	Salle de réunion virtuelle représentée par une salle de clavardage de groupe dans le système développé dans ce projet.
API	“Application Programming Interface” : Consiste en une façade à une librairie logicielle ou toute autre entité informatique permettant d’interagir avec celle-ci depuis notre propre logiciel en développement.
REST	Service web permettant l’interaction via de simples requêtes HTTP. Typiquement, ce genre de service prend, dans ses requêtes HTTP, des paramètres et/ou un corps au format XML ou jSON et retourne une réponse HTTP au format XML ou jSON.
XML	“Extensible Markup Language” est, comme son nom l’indique, un langage de balisage extensible, permettant de présenter de l’information en respectant une syntaxe définie.
jSON	Comme le XML, jSON est un langage de balisage, mais avec une tout autre syntaxe, dérivée du JavaScript. jSON est reconnu pour être plus léger que le XML.
Objective-C	Langage de programmation dérivé du langage C, utilisé lors du développement Apple, sous le cadriciel Cocoa.
Cocoa	Cadriciel (“framework”) développé par Apple permettant le développement d’application Mac, iPhone et iPad. Lorsqu’on parle de développement sous une de ces plateformes, on parle souvent de “Développement Cocoa”.
Beta testing	Forme de test de logiciel où de réels utilisateurs sont choisis comme cobayes. Ceux-ci, à force d’utiliser le logiciel, trouvent les anomalies.
UML	“Unified modelling language” : langage formel permettant de représenter graphiquement la structure ou le comportement de composantes logicielles.

Introduction

En tant que futur ingénieur logiciel, il est primordial de savoir mené à terme un projet logiciel. Des décisions importantes devront être prises dans de tels projets. Plusieurs de ces décisions concerneront l'analyse des exigences, l'architecture et la conception logicielles. En effet, ces trois phases du cycle de développement sont indispensables à la création d'un logiciel de qualité.

Dans le cadre du cours LOG792, projet de fin d'études au baccalauréat à l'*École de technologie supérieure*, il est demandé d'effectuer un projet de conception logicielle. Le projet mis en route lors de ce cours est un logiciel ayant pour but de faciliter l'application et le suivi des processus de développement chez Edovia, une très petite entreprise oeuvrant dans la création d'application iPhone, iPad et Mac. Le projet se voulait au départ un projet se limitant à l'analyse et la conception du logiciel, mais étant donné la volonté de l'entreprise d'avoir le logiciel en main le plus tôt possible, le projet est allé un peu plus loin. En effet, le logiciel fut implémenté en grande partie et offre la plupart des fonctionnalités décrites lors de l'analyse et la conception.

Le présent document se veut une synthèse du travail accompli tout au long du projet. Tout d'abord, la problématique sera expliquée pour ensuite faire place à la solution proposée. La méthodologie utilisée lors du projet sera abordée, l'analyse des risques sera dévoilée, un résumé de l'analyse des exigences, de l'architecture, de la conception et de l'implémentation seront présentés. Enfin, le futur du projet sera discuté et quelques recommandations seront proposées.

Chapitre 1
Contexte et problématique

Edovia

Oeuvrant dans la création d'application iPhone depuis le lancement de celui-ci en 2008, Edovia[1] est une très petite entreprise montréalaise opérée par Luc Vandal. Avant 2008, Edovia oeuvrait dans la conception d'un plugiciel antispam pour Microsoft Outlook, mais cette époque n'a plus vraiment d'importance dans le contexte actuel.

Depuis 2008, l'entreprise s'est consacré au développement de diverses applications iPhone tel que le traducteur *Lingo*, le podomètre *Steps*, le chercheur de taxi *RocketTaxi*, le convertisseur de devises *Currencies*, le pavé numérique utilisant le protocole VNC *NumPad*, le pavé tactile utilisant aussi le protocole VNC *TouchPad* et finalement le client VNC complet *Screens*. *Screens* et *TouchPad* sont également disponibles en version iPad. Depuis l'été dernier, Edovia a ajouté des applications Mac à son éventail d'application Apple, soit une version Mac de *Currencies* et une version Mac de *Screens*.

Le quartier général d'Edovia est situé dans un loft urbain en bordure du canal Lachine. L'entreprise emploi actuellement deux employés, ce qui totalise trois développeurs en incluant le fondateur, Luc Vandal, qui est lui-même le développeur principal de l'entreprise.

Processus de développement actuel

Dans le contexte d'une très petite entreprise comptant moins de 5 employés, les processus de développement manquent souvent de formalisme. C'est ce qui arrive chez Edovia. L'entreprise est pourtant souscrite à de nombreux services en ligne ayant pour but d'améliorer les processus de développement dans le cycle de vie logiciel. Parmi ces services, on retrouve un outil de gestion de soutien aux utilisateurs, un gestionnaire de billet, un gestionnaire de dépôt SVN et finalement un service de messagerie instantanée de groupe pour faciliter le travail à distance. Chacun de ces services fonctionne relativement bien et répond correctement aux besoins qu'il cible. Cependant, l'utilisation quotidienne de l'ensemble de ces services s'apparente davantage à une tâche superflue plutôt qu'à une tâche qui facilite le processus de développement pour une si petite équipe.

Pour mieux visualiser la chose, voici un scénario classique. Un utilisateur reporte une anomalie dans une application. Le développeur est notifié par courriel et doit ouvrir l'application web de gestion du soutien à la clientèle pour consulter le message de l'utilisateur. Une fois avoir constaté que l'anomalie est bien réelle, le développeur ouvre l'application web de gestion des billets pour entrer l'anomalie en y joignant un lien hypertexte vers le message original de l'utilisateur. Une fois l'anomalie corrigée dans le code source, le développeur doit référencer le billet dans son message de propagation SVN. Encore là, il faut préalablement avoir configuré le gestionnaire de billet avec le gestionnaire de dépôt SVN pour que le lien soit possible et cette configuration n'est pas des plus simple. Par la suite, le développeur peut retourner dans le gestionnaire de billet pour assigner le billet à un autre développeur pour que celui-ci fasse une revue de code. Cet autre développeur doit lui aussi aller consulter le gestionnaire de billet pour prendre connaissance du billet, suivre le lien vers le

[1] À Propos - Edovia inc. : http://edovia.com/company

message du client reportant l'anomalie pour bien comprendre le problème, ensuite, suivre le lien vers le gestionnaire de dépôt pour visualiser les changements apportés dans le code pour finalement effectuer la revue. Éventuellement, le billet ainsi que la discussion dans l'outil de soutien devront être fermés et étiquetés comme résolus.

Comme on peut le constater, dû au fait que plusieurs services distincts sont utilisés dans ce processus, un nombre important de manipulations sont nécessaire lorsqu'il vient temps de corriger une anomalie mineure ou encore implémenter une simple fonctionnalité. La lourdeur du processus vient souvent décourager les développeurs. Ceux-ci optent alors pour l'option facile : « je prends connaissance de l'anomalie via l'outil de support, je corrige l'anomalie dans le code source, je propage ma modification via SVN, et voilà ! » Bien que l'équipe soit très petite, la supervision du travail des autres développeurs et le contrôle du travail deviennent difficiles. D'un tel manque de contrôle découlent alors davantage d'anomalies insérées et une mauvaise gestion des tâches et de leurs priorités.

Philosophie Apple : There's an app for that !

Tout le monde a déjà entendu ce slogan de Apple "There's an app for that ! " (Traduction : "Il y a une application pour ça!"). Comme les développeurs chez Edovia sont habitués à cette mentalité, qu'ils développent eux même des applications Apple et qu'ils utilisent eux même des applications Apple dans leur travail, l'utilisation de quatre sites web ne colle pas à leur expérience de développement recherchée. En effet, le fait que chaque service soit une application web ne favorise pas la consultation régulière des développeurs. Même si les développeurs ajoutent les liens dans leurs favoris et qu'ils ouvrent chaque application web quand ils commencent à travailler, les développeurs se retrouvent vite avec une panoplie d'onglet ou de fenêtres ouvertes dans leurs navigateurs et les quatre applications web nécessaires au processus de développement tombent rapidement dans l'oubli, ou sont fermées par mégarde.

Chapitre 2
Objectifs

Unification des services

Enfin, si toutes les facettes du processus décrites dans le scénario typique de développement pouvaient être intégrées dans une seule application simple d'utilisation, les développeurs seraient moins découragés par la lourdeur du processus et l'appliqueraient davantage.

Dans ce sens, l'objectif ultime du projet est de développer et d'implanter un outil qui intègre la gestion du soutien à la clientèle, la gestion des billets, les revues de code et un outil de clavardage. L'application développée devra permettre de lier, de façon transparente, les interventions des clients et les propagations SVN avec les billets de développement.

Le but n'est pas de réinventer la roue dans chacun de ces aspects du projet puisque chacun des services utilisés par l'entreprise répond déjà aux besoins qu'il cible. Dans cette optique, chacun des services offre déjà une interface de programmation (API) publique REST. Celles-ci pourront donc être utilisées dans le projet afin d'assurer en majorité la gestion et la persistance des données.

Précisément, les services externes utilisés chez Edovia sont les suivants :

- **Tender App**[2] (gestionnaire du soutien à la clientèle)
- **Lighthouse App**[3] (gestionnaire de billets)
- **Beanstalk App**[4] (gestionnaire des dépôts SVN)
- **Talker App**[5] (outil de clavardage de groupe)

Application Mac native

Afin de respecter la philosophie Apple, mais surtout d'éviter les pièges des applications web, la solution sera développée comme une application Mac native en utilisant le framework Cocoa.

Dans l'étendue du projet de fin d'études visant principalement l'analyse et la conception logicielle, l'objectif se limitera à produire un prototype fonctionnel de l'application en question. Le prototype devra tout de même respecter l'analyse et la

---

[2] Tender App - www.tenderapp.com

[3] Lighthouse App - www.lighthouseapp.com

[4] Beanstalk App - www.beanstalkapp.com

[5] Talker App - www.talkerapp.com

conception faites dans le cadre du projet afin que ce prototype s'apparente à une ébauche de l'application réelle. De cette façon, l'efficacité de la solution pourra être mise à l'épreuve le plus tôt possible.

Le prototype devra démontrer que l'application simplifie le processus de développement par son nombre de manipulations nécessaires et par le temps d'exécution des scénarios typiques d'utilisation. Ce nombre d'opérations et ce temps d'exécution devront être drastiquement moindres que ceux de la situation actuelle. De plus, le développeur devra avoir une plus forte dépendance à l'application que sa dépendance actuelle aux quatre services web distincts.

Possibilités à long terme

La conception logicielle devra être élaborée de façon à ce que l'application ne dépende pas directement des API des services externes. L'entreprise dépense d'importantes sommes annuellement pour l'abonnement à ces différents services. Éventuellement, l'entreprise pourrait remplacer ces services payants par ses propres services, ou d'autres services gratuits. De façon visuelle, les 4 nuages dans l'illustration ci-dessous pourraient être remplacés, en totalité ou en partie, par un ou plusieurs modules. De cette façon, le projet permettra une économie importante.

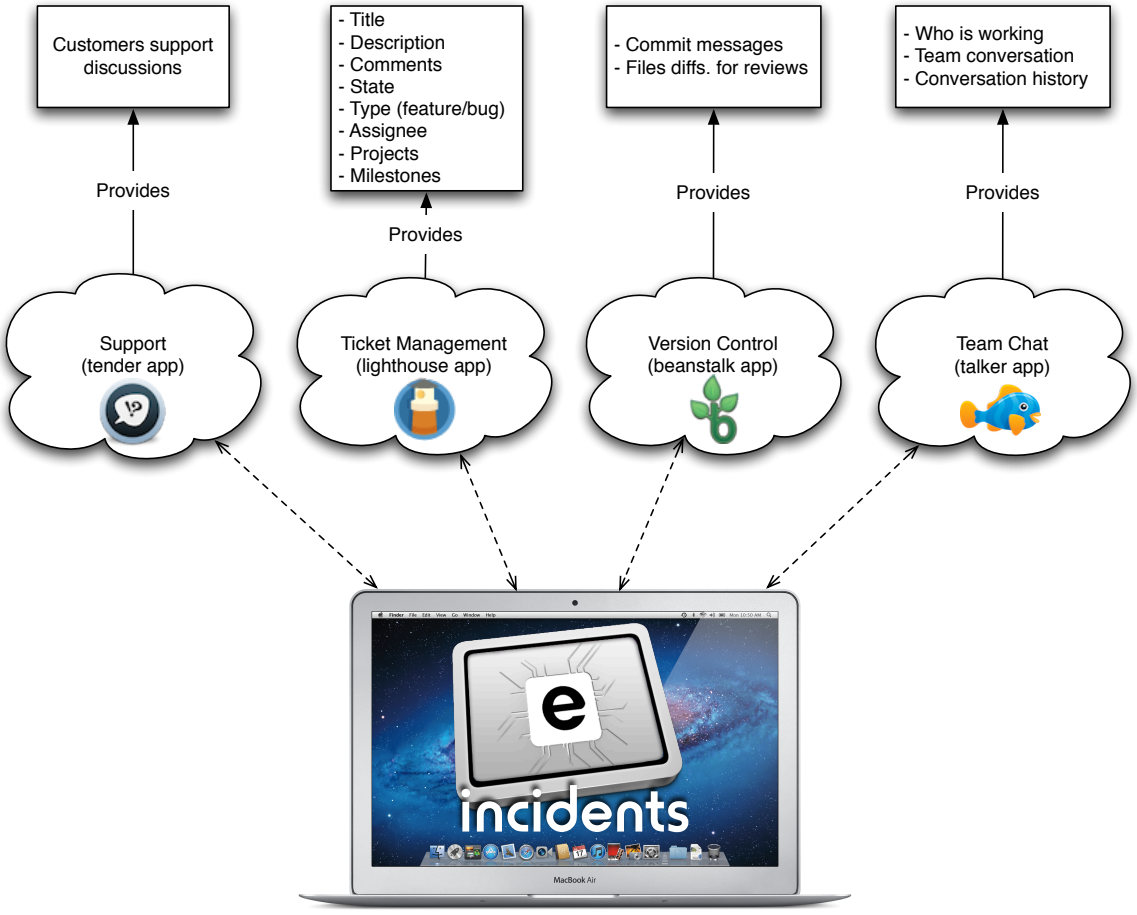


Figure 2.1 - Relation avec les services externes

Une fois les services externes remplacés, si ce jour vient qu'à arriver, l'entreprise n'aura plus aucun lien avec les entités externes et la solution aura donc le potentiel d'être commercialisé, si l'entreprise le souhaite.

Chapitre 3
Méthodologie

ISO/IEC 29110 pour les TPO

Étant donné la petitesse de l'entreprise, des processus particuliers s'appliquent. Heureusement, la norme **ISO/IEC 29110**[6] fut développée en collaboration avec plusieurs professeurs de l'ÉTS, spécialement dans l'optique de l'application de normes de génie logiciel dans les très petites entreprises. Pour chaque phase du cycle de vie logiciel, la norme fournit une trousse de déploiement qui a pour but de proposer de bonnes pratiques. Comme le projet vise principalement **l'analyse**[7] et la **conception**[8]**,** ce sont les deux trousses de déploiement respectives à ces deux phases qui seront utilisées dans le projet. À noter qu'il existe également des trousses pour les tests, la livraison du produit, la gestion des versions, la gestion de projet, la vérification/validation et l'auto-évaluation.

Le travail sera principalement concentré sur la production des artéfacts proposés dans les trousses de déploiement de la norme ISO/IEC 29110. Pour la phase de l'analyse, on retrouve parmi les artéfacts suggérés un document de SRS, un document de cas d'utilisation et un prototype. Pour la phase de conception, on retrouve comme artéfacts un document de design. Tous ces artéfacts se retrouvent en annexe au présent document.

Processus de développement du projet

Une approche itérative sera utilisée. Un premier jet de l'analyse des requis sera fait avec le **SRS et les cas d'utilisation**[9]**,** ensuite un premier jet de conception sera effectué en produisant les diagrammes utiles à la conception du cœur du système, puis finalement un prototype du cœur du système sera implémenté. Ceci marquera la fin de la première itération. Une deuxième itération sera alors entreprise en commençant par raffiner l'analyse des requis, en effectuant la conception des modules plus secondaires du système et en implémentant ces modules dans le prototype. Enfin, une troisième itération sera entreprise, une quatrième, une cinquième, ainsi de suite, jusqu'à la complétion du projet. À noter que le projet sera poursuivi jusqu'à son achèvement si celui-ci n'est pas terminé avant la fin de la session allouée pour le projet de fin d'études.

Une matrice de traçabilité sera maintenue à jour tout au long du projet, afin d'assurer la couverture de chaque fonctionnalité. Suite à l'achèvement d'un artéfact dans une itération, une revue par les pairs sera effectuée par Luc

---

[6] ISO/IEC 29110 - http://profs.etsmtl.ca/claporte/english/VSE/index.html

[7] ISO/IEC 29110 : Trousse de déploiement pour la phase d'analyse
http://profs.etsmtl.ca/claporte/english/VSE/Deploy%20Pack/DP-Software%20Requirements%20Analysis-V1_2.doc

[8] ISO/IEC 29110 : Trousse de déploiement pour la phase de conception
http://profs.etsmtl.ca/claporte/english/VSE/Deploy%20Pack/Deployment_Software_Design_v0%205.doc

[9] Exemple de SRS : Document choisi comme exemple dans le cours de LOG410, à la session d'hiver 2010
http://dl.dropbox.com/u/1334047/SRS%20Eq13.docx

Vandal, le contact de l'entreprise partenaire, Edovia. À mesure que le prototype fonctionnel est développé, celui-ci sera utilisé par les développeurs afin d'assurer une validation sous forme de "beta testing".

Outils

Toute la documentation (document, rapport, diagramme et graphique) sera sauvegardée dans le dépôt partagé de l'entreprise prévu à cet effet. Ce dépôt partagé est desservi par le service Dropbox. À noter que l'entreprise est souscrite à un plan Dropbox corporatif, qui offre des options avancées de gestion des versions et d'un espace de sauvegarde de 350 GO, ce qui est amplement pour la sauvegarde de la documentation.

Les diagrammes UML et tout autre diagramme ou graphique produit dans le cadre de ce projet sont réalisé à l'aide de l'application OmniGraffle, de la compagnie OmniGroup, un logiciel de conception de diagramme exclusif à Mac OS X s'apparentant au Visio de Microsoft. Des « stencils » pour UML2 sont disponibles en téléchargement sur graffletopia.com. Avec ces « stencils » dans OmniGraffle, il est facile de créer des diagrammes UML et d'en respecter la notation.

Étant donné qu'une approche itérative est utilisée et qu'une phase d'implémentation est effectuée de manière incrémentale tout au long du projet par le biais du prototype fonctionnel, beaucoup de code source sera produit. La sauvegarde de ce code source sera assurée par un dépôt SVN desservi par BeanStalk, un des services externes qu'intègrera l'application du projet.

Pour les communications avec le contact de l'entreprise en collaboration, Luc Vandal, l'outil de clavardage de groupe Talker sera utilisé lors du travail à l'extérieur du bureau de l'entreprise, Edovia. Talker est aussi un des services externes intégrés à l'application du projet. Bien évidemment, des courriels seront aussi utilisés comme moyen de communication.

Les artéfacts, ainsi que ce rapport, sont produits à l'aide du logiciel de traitement de texte Pages d'Apple puisque c'est ce qui est utilisé par l'entreprise. Donc, c'est ainsi plus facile pour l'entreprise de consulter le travail effectué.

Chapitre 4
Analyse des risques

Peu d'importance fut accordée à l'analyse des risques durant ce projet étant donné la non-criticité du projet envers l'entreprise. Du moins, quelques risques ont été identifiés, noté d'un impact, d'une probabilité et pour chacun d'eux, un bref moyen de mitigation fut rédigé. Aucun de ses risques ne s'est produit au cours du projet.

Risque	Impact	Probabilité	Mitigation / atténuation
Si le projet n'est pas aussi utile que prévu par le client, alors celui-ci verra ses processus de travail ralentis et donc l'accomplissement de ses objectifs ralentis aussi.	Élevé	Faible	Sonder les différents employés de l'entreprise et d'autres employés d'entreprises qui ont une situation similaire serait une bonne façon de vérifier la viabilité de la solution proposée dans le projet.
Étant donné que l'entreprise partenaire utilise un même employé pour le développement du projet et pour le développement de ses produits, si l'employé se retrouve à passer plus de temps sur le développement des produits, alors la livraison du projet sera retardée.	Moyen	Moyen	Il serait important de prévoir des plages horaires spécifiques à consacrer au projet et de respecter ces plages horaires.
Si le logiciel développé ne permet pas, à long terme, le remplacement des services payants par des services maison, alors l'entreprise ne pourra pas économiser sur les services payants (~200$/mois).	Élevé	Moyen	S'assurer tôt dans la conception que des interfaces sont prévues afin de découpler efficacement le système des services externes utilisés.

Risque	Impact	Probabilité	Mitigation / atténuation
Si l'entreprise ne fait pas d'effort pour adopter la solution développée dans le projet pour son processus de développement, alors les efforts alloués au développement du logiciel seront gaspillés.	Élevé	Moyen	Inclure dans le logiciel développé des fonctionnalités primordiales au développement, afin que les développeurs utilisateurs soient forcés d'utiliser le logiciel et ainsi bénéficier de ses avantages.
Si les objectifs du projet ne sont pas suffisamment mesurables, alors il peut être difficile de valider l'utilité du logiciel développé.	Moyen	Moyen	Établir des exigences non fonctionnelles avec des mesures spécifiques en ce qui concerne l'utilisabilité. (Exemple : nombre de clics nécessaire, nombre d'opération nécessaire, temps nécessaire pour effectuer une tâche, etc.)
Si l'envergure du projet s'avère trop complexe pour le temps disponible, alors la faisabilité du projet peut être mise en péril.	Élevé	Faible	Les services externes centralisés dans le logiciel offrent un nombre important de fonctionnalités. Il sera important de bien définir la portée du projet et de se limiter aux cas d'utilisation essentiels à l'accomplissement des objectifs.
Si les composantes réutilisables ne s'avèrent pas adaptables au projet, alors beaucoup de temps peut être perdu afin de le rendre adaptable.	Moyen	Moyen	Bien évaluer ce qui peut être réutilisé dans les sources d'une l'application développée par une entreprise partenaire avant de débuter la réutilisation. Si l'adaptation est trop complexe, évaluer la possibilité de faire sa propre implémentation.
Si les API publiques des services externes utilisés ne répondent pas à tous les besoins du projet, alors la faisabilité du projet peut être mise en péril.	Élevé	Moyen	Vérifier avec la documentation des API publiques des services externes si toutes les fonctionnalités du projet seront possibles.

Risque	Impact	Probabilité	Mitigation / atténuation
Si le projet prend plus de temps que prévu, alors il se peut que l'entreprise n'ait pas assez de budgets à allouer au projet.	Élevé	Faible	Faire un plan de travail avec des dates de livraison précises, faire un suivi régulier avec le contact de l'entreprise partenaire, et respecter le plus possible le plan de travail.
Si les dates des livrables ne sont pas bien définies, alors il se peut que le projet n'avance pas assez vite et que le budget soit dépassé.	Moyen	Faible	Faire un plan de travail avec des dates de livraison précises, faire un suivi régulier avec le contact de l'entreprise partenaire, et respecter le plus possible le plan de travail.
Si le design s'avère plus difficile à concevoir que prévu, alors la complexité et l'envergure du projet peuvent être affectées.	Moyen	Faible	Faire un design tôt dans le processus de développement.
Si les services externes utilisés ne répondent pas suffisamment rapidement pour une utilisation efficace du logiciel, alors la viabilité de la solution développée peut être mise en jeu.	Élevé	Moyen	Faire un prototype fonctionnel des communications avec les services externes tôt dans le processus de développement.
Si la documentation des API publiques des services externes utilisés est incomplète ou inappropriée, alors il sera plus difficile d'interfacer ces services.	Moyen	Faible	Vérifier, le plus tôt possible, la documentation des API publiques des services externes pour voir si toutes les fonctionnalités du projet seront possibles.
Si un désastre naturel survient, alors la sécurité de tout le travail est basée sur des services externes (Dropbox, SVN), sur lesquels on n'a aucun contrôle.	Moyen	Faible	Vérifier que les services externes utilisés pour le stockage du travail effectué dans ce projet garantissent la récupération des données en cas de désastre naturel.

Tableau 4.1 - Analyse des risques

Chapitre 5
Résumé de l'analyse

Cette section présente un résumé de l'analyse des exigences qui a été faite dans le document de SRS. Ce ne sont pas toutes les exigences qui sont décrites dans cette section. Seulement les aspects des exigences qui ont demandé une attention particulière sont discutés ici. Pour la liste complète des cas d'utilisation, des exigences fonctionnelles et non fonctionnelles, se référer au SRS dans l'**Annexe A**. Pour commencer, voici un diagramme de flux d'information résumant le scénario classique que couvre le logiciel avec ses différentes interactions avec les services externes.

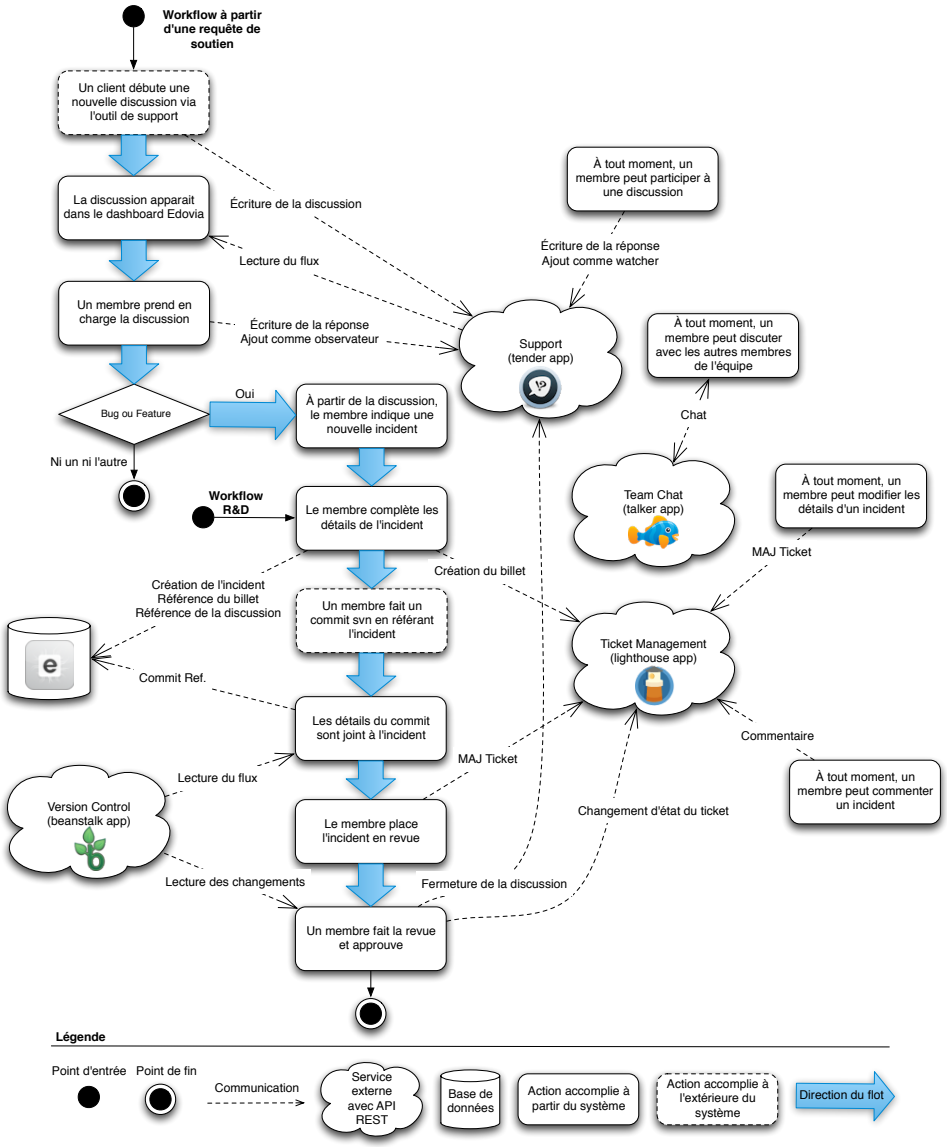


Figure 5.1 - Flot d'utilisation typique et interaction avec les services externes

Préoccupations des exigences fonctionnelles

Affichage en tout temps du nombre d'éléments à traiter

Afin de forcer le développeur à répondre aux interventions des clients, à régler des anomalies, à implémenter de nouvelles fonctionnalités et à effectuer des revues de codes, il est important de toujours afficher le nombre d'éléments en attente de traitement. Évidemment, les développeurs chercheront à faire descendre ces nombres près du zéro. Pour se faire, il est pertinent d'ajouter des badges indiquant, pour chaque type de travail, le nombre d'éléments à traiter. Également, un compte total de tous ces éléments peut être ajouté dans un badge sur l'icône de l'application dans le "dock" de Mac OS.

Afficher des données à jour

Étant donné que les données se trouvent sur des serveurs web distants, une préoccupation majeure est celle d'afficher, en tout temps, de l'information reflétant le plus possible les valeurs des données distantes. Bien qu'il soit pratiquement impossible de savoir exactement quand les données changent sur le serveur, à moins d'être responsable de l'implémentation du serveur, il est possible de mettre en place un système d'écouteur qui télécharge les données régulièrement pour voir si ceux-ci sont encore à jour. C'est ce qui a été mis en place dans l'application. Le sujet est discuté plus en détail dans le chapitre portant sur la conception.

Notifier l'utilisateur des messages de clavardage non lus

Comme tout bon client de clavardage, celui-ci doit notifier l'utilisateur lorsque de nouveaux messages surviennent. Ceci est un défi en soi puisque le service de clavardage utilise son propre protocole et très peu d'information est disponible sur celui-ci. Le service Talker propose des solutions fonctionnelles en JavaScript, mais rien ne pouvant être intégré facilement dans l'application Mac développée. Donc, une façon simple d'intégrer le clavardage fut d'embarquer une vue web directement dans l'interface de l'application. Bien que ceci permettait de clavarder sans problème, ceci ne permettait pas d'être notifié des nouveaux messages. Par contre, il était possible d'obtenir le contenu HTML de la vue web et d'y exécuter du JavaScript. De cette façon, lorsque la vue web recevrait des nouvelles données, principalement à l'arrivée de nouveaux messages, il serait possible de compter le nombre de nouveaux messages en utilisant du JavaScript.

Filtrer les projets affichés

Cette fonctionnalité était assez préoccupante au départ. Ce n'est qu'au bout d'un certain temps que la solution s'est apparu évidente. Cette fonctionnalité n'aurait qu'à s'appliquer aux listes de discussion et d'incident. Comme ces listes sont téléchargées en continu, le filtre est affecté immédiatement au prochain téléchargement des listes, soit dans un délai de quelques secondes. En effet avant qu'un élément téléchargé au format JSON ou XML soit transformé en objet et retourné dans la liste, une simple vérification du projet qu'il concerne est nécessaire.

Préoccupations des exigences non fonctionnelles

Utilisabilité

Le but principal du logiciel n'est pas réellement de permettre aux développeurs d'accomplir de nouvelles choses, mais plutôt d'accomplir ce qu'ils faisaient déjà, mais d'une façon beaucoup plus naturelle, intuitive et agréable. Donc, tout ça se résume par l'utilisabilité. Bien que cet attribut de qualité soit difficilement mesurable, il est possible de calculer le

nombre d'étapes nécessaires pour effectuer une tâche. En d'autres mots, ce qui est visé est de réduire ce nombre d'étapes considérablement par rapport au contexte initial.

Fiabilité

Comme l'application développée ajoute un intétermédiaire entre l'interaction des utilisateurs et l'affectation des données, il est très important qu'un système fiable soit mis en place. En effet, lorsqu'un utilisateur tente de modifier des données en utilisant l'application, il ne doit pas se soucier si le changement a bel et bien été effectué sur les données distantes. À l'inverse, si quelque chose se passe mal, l'utilisateur doit absolument en être notifié.

Performance

Enfin, une autre qualité très importante dans le logiciel est celle de la performance. En effet, comme des données distantes sont constamment affichées, il est important que celles-ci soient affichées le plus rapidement possible. Le temps d'attente de l'utilisateur doit être réduit au minimum. Des systèmes de cache et d'écouteur de données ("Datasource Listener") sont des moyens qui aident à atteindre cette qualité dans le contexte de l'application.

Mesures, objectifs et résultats

Mesure	Objectif	Résultat	Succès	Méthode utilisée pour atteindre l'objectif
Nombre de clics nécessaire pour consulter un type de travail	1	1	Oui	Une section est dédiée pour chaque type de travail (Soutien aux utilisateurs et travail sur des billets/incidents). Ces sections sont disponibles via un menu toujours visible.
Nombre de clics nécessaire pour connaître la charge de travail complète	0	0	Oui	Le total d'élément à traiter tout type confondu (Discussions, Incidents, messages de clavardage) est affiché dans le badge de l'icône de l'application dans le "Dock".
Temps nécessaire pour récupérer d'une panne	< 1 minute	12 secondes	Oui	Comme c'est une simple application Mac tirant ses données du web, ces données sont en lieu sûr en tout temps et une panne de l'application ne risque pas de causer de problèmes de quelconque nature. L'application se redémarre sans particularité.
Taux de succès de la lecture ou de l'écriture des données distantes	95%	~99%	Oui	Aucun échec n'a été observé lors de la lecture ou l'affectation des données.

Mesure	Objectif	Résultat	Succès	Méthode utilisée pour atteindre l'objectif
Temps moyen de prédiction d'une défaillance	1 / 6 heures	ND	Non	Il est trop tôt pour se prononcer sur les résultats de cette mesure étant donné que le logiciel est encore en version Bêta.
Temps nécessaire pour lire et affecter les données distantes	5 secondes	de 1 à 15 secondes	Non	Parfois, les services web mettent énormément de temps à répondre, probablement dû à l'achalandage du service. Également, l'utilisation directe de requête SQL sur un serveur distant n'est pas optimale pour obtenir les liens entre les incidents et les discussions.
Temps maximal avant que les données soient rafraichies	20 secondes	~15 secondes	Oui	Les DatasourceListener se chargent de cette partie. Ils écoutent constamment les éléments présentés à l'utilisateur et interrogent les données distantes toutes les 10 secondes.
Temps nécessaire avant la réception d'un message de clavardage	2 secondes	~1 seconde	Oui	La fonctionnalité de clavardage de l'application a été implémentée à l'aide d'une simple WebView. Donc, les messages sont reçus aussi rapidement que si le service était utilisé directement via son application web.
Temps nécessaire pour intégrer un nouveau service externe avec API Rest	< 1 semaine	~ 3 jours	Oui	Une fois avoir développé le coeur du système et le module du soutien aux utilisateurs, il fut très rapide pour développé l'équivalent dans le module des incidents.

Mesure	Objectif	Résultat	Succès	Méthode utilisée pour atteindre l'objectif
Nombre de classes à modifier lors du remplacement d'un service externe	1	1	Oui	La classe avec le suffixe Interface et héritant de HTTPServiceInterface est la seule classe à modifier. Ceci a été prouvé alors que l'interface pour accéder aux changements SVN reliés à un incident fut modifiée pour utiliser la ligne de commande SVN plutôt que le service web.
Temps nécessaire pour installer l'application	< 3 minutes	< 1 minute	Oui	Comme c'est une simple application Mac, il suffit de copier/coller le fichier ".app" sur l'ordinateur où l'on désire installer l'application.
Espace disque occupé par l'application	< 10 MB	2.7 MB	Oui	Rien de spécifique n'a été fait pour atteindre cet objectif.

Tableau 5.1 - Objectifs mesurables du projet et résultats

Chapitre 6
Résumé de la conception

Cette section présente un résumé de la conception. Ce ne sont pas tous les aspects de la conception qui y sont présentés, seulement les aspects jugés les plus intéressants ou qui demandaient une attention particulière sont présentés dans ce chapitre.

Pour débuter, voici un aperçu de l'architecture du logiciel.

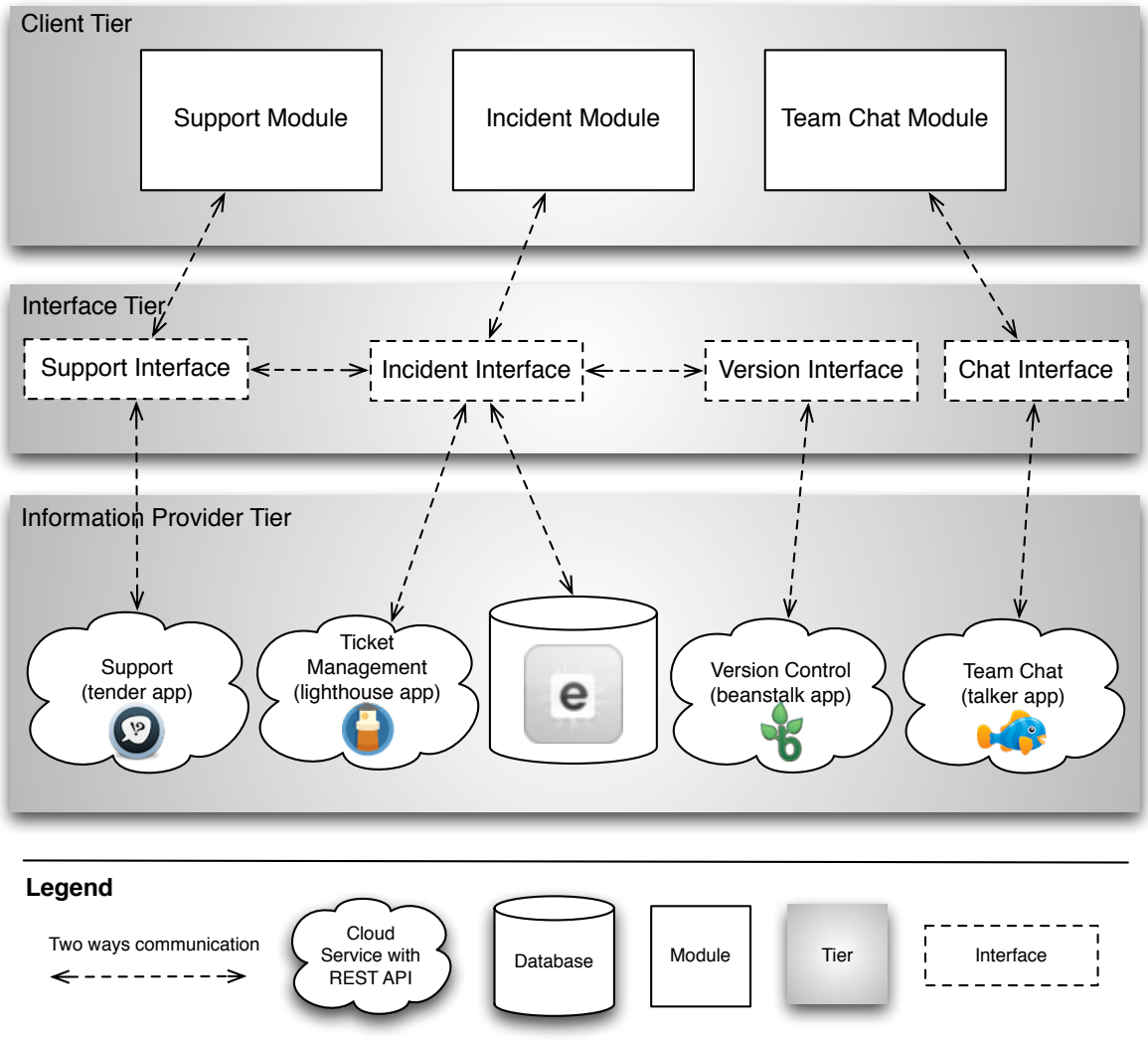


Figure 6.1 - Architecture du logiciel

Le principal but de cette architecture en couche est de découpler l'application cliente des services externes qui fournissent les données. Le tout est découplé grâce aux interfaces qui les relient. Pour chacun des modules de l'application cliente, tout ce dont il a à connaître pour l'accès aux données, c'est son interface associée. Plus concrètement, le module de support doit seulement connaître l'interface de support, le module d'incident l'interface des incidents et le module de clavardage l'interface de clavardage.

Seule l'interface des incidents a des dépendances vers les autres interfaces puisqu'un incident est la combinaison d'une discussion tirée du service de support, d'un bogue à régler ou d'une fonctionnalité à développer tirée du service de gestion des billets et enfin, de propagations SVN tirées du service de contrôle de version. De plus, c'est pourquoi l'interface des incidents fait des accès à une base de données en plus des accès aux services externes. La base de données est là principalement pour entreposer les relations entre les composantes d'un incident.

Voici maintenant une mise en relation de l'architecture avec la conception réalisée.

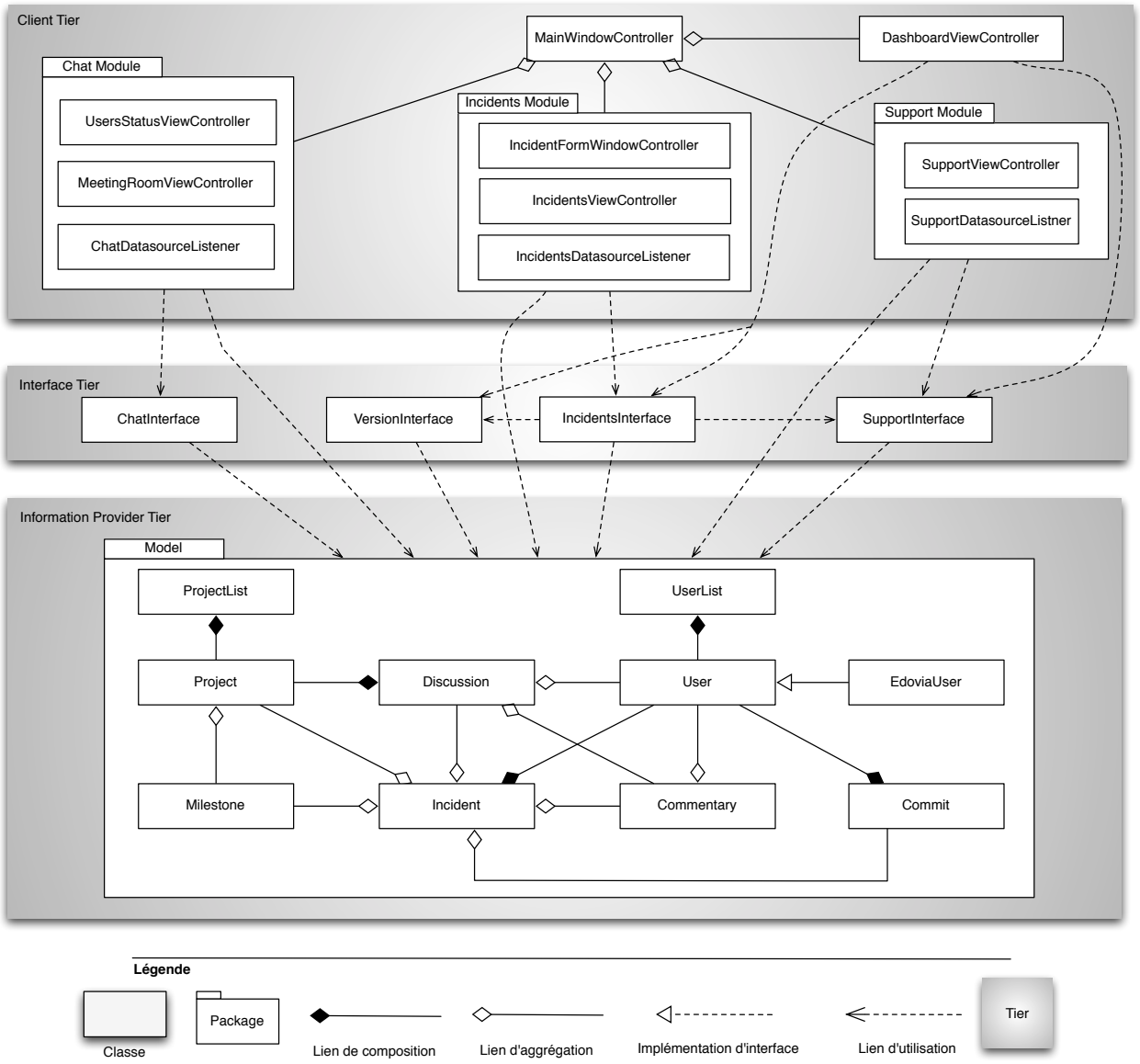


Figure 6.2 - Conception globale

Comme on peut le constater, une seule classe est nécessaire pour chacune des interfaces à implémenter. Les interfaces sont utilisées par les modules clients et les interfaces s'occupent d'interroger les services externes et d'en retourner des objets du modèle. C'est pour cette raison que le modèle du domaine a été placé dans la couche d'accès à l'information. Bien entendu, les modules manipulent ces objets du modèle, c'est pourquoi des liens d'utilisation sont présents entre les modules clients et le modèle. Dans chacun des modules, on peut retrouver une classe suffixée de "DatasourceListener". Ces classes ont pour but de rendre possible la mise en place d'un patron observateur. Leur rôle est d'interroger les classes interfaces dans un fil d'exécution en arrière-plan et de notifier les contrôleurs de vue (suffixé "ViewController") lorsqu'ils détectent que les données en consultation ont changées. Les contrôleurs de vue peuvent utiliser la classe interface directement pour les actions d'écriture sur les données, mais pour les actions de consultation, il est préférable d'utiliser le "DatasourceListener". Le sujet sera abordé plus en détail dans la section "MVC et patron observateur".

Modèle du domaine

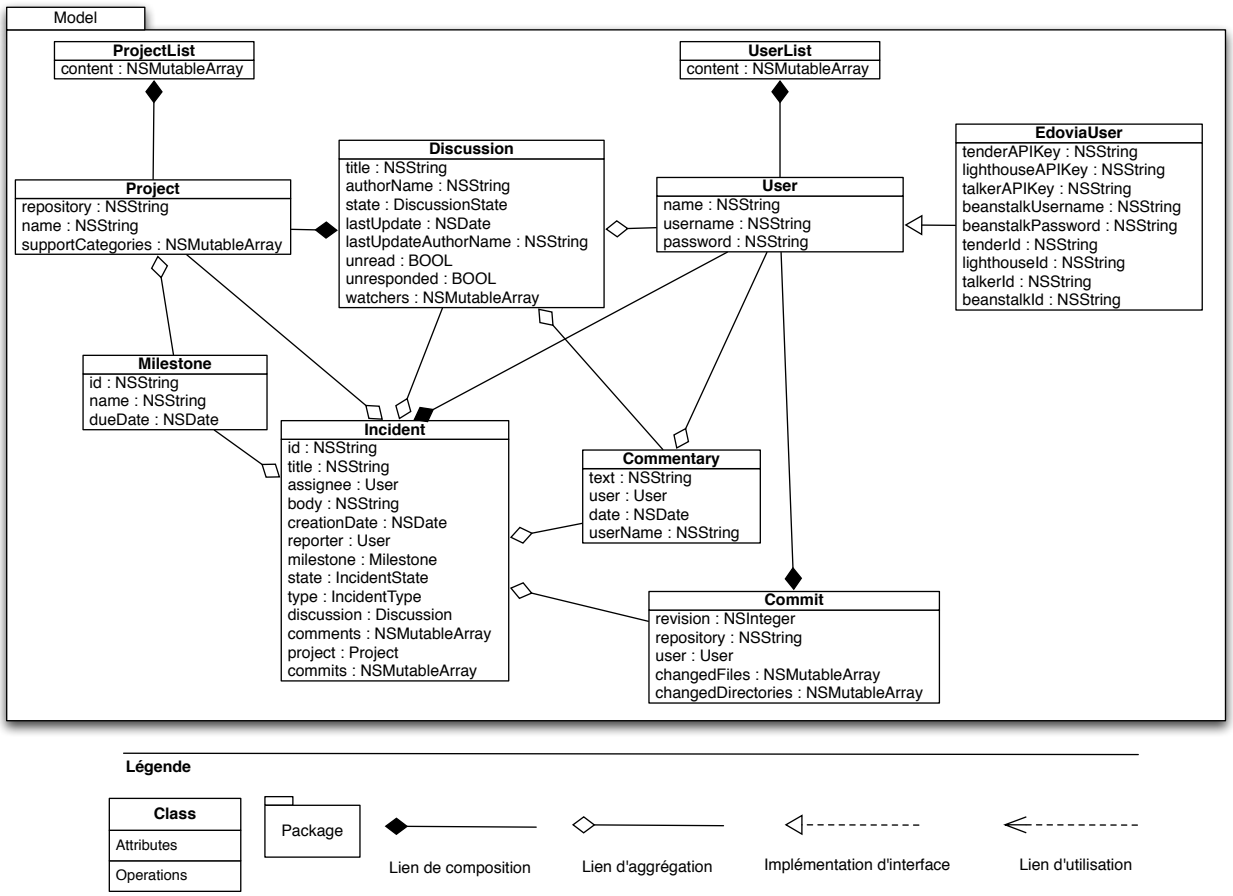


Figure 6.3 - Modèle du domaine

Dans le chapitre 2, à la figure 2.1, il était possible de voir toutes les informations fournies par les différents services externes utilisés dans le cadre du projet. Le diagramme de classe ci-dessus illustre comment chacune de ces informations a été modélisée. Il est important de connaître les objets dont il est question dans l'application avant de continuer plus loin. Pour plus de détails, voir la **section 4.2** du document de conception à l'**Annexe C**.

MVC et patron observateur

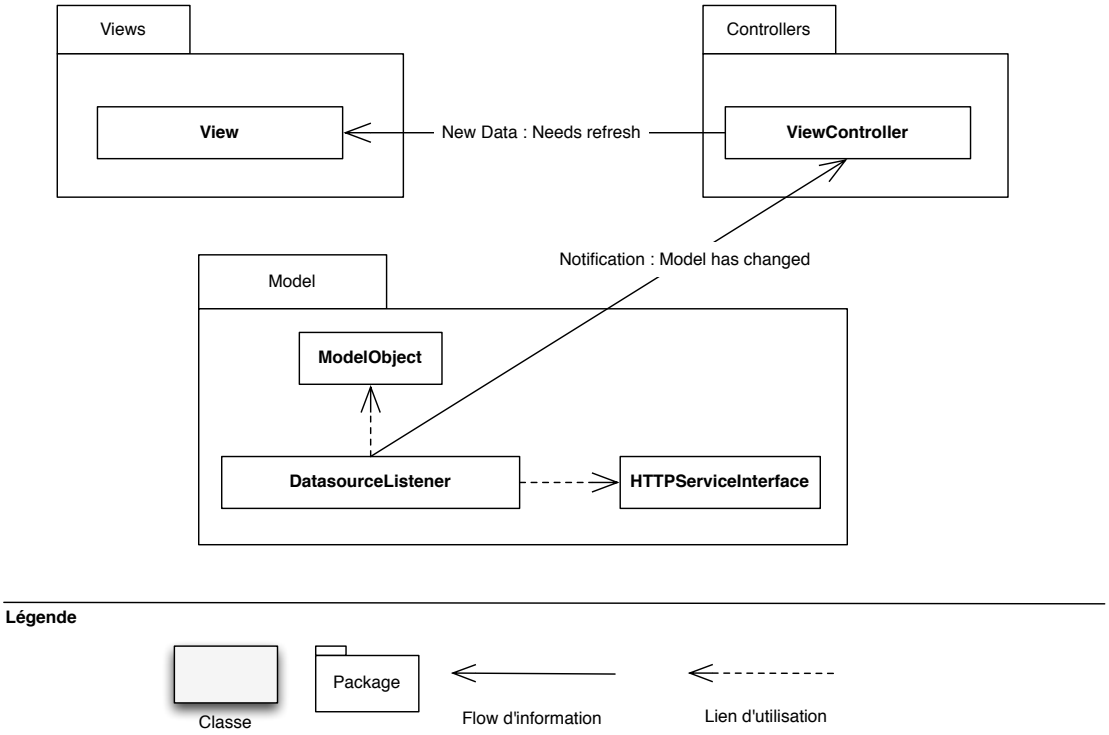


Figure 6.4 - MVC et son patron observateur

Un des défis de taille du projet était de réussir à afficher des données à jour en tout temps. En effet, comme on se trouve dans un contexte de système distribué, si un utilisateur consulte un incident et qu'un autre utilisateur en modifie ses informations au même moment, le premier utilisateur doit voir les modifications dans les plus brefs délais, pour éviter toute source de conflits. La mise en place d'un système de "DatasourceListener" fut la solution choisie pour remédier à ce problème. Plutôt que le contrôleur mette à jour les données continuellement en interrogeant à intervalle (au 10 secondes par exemple) l'interface du service externe, c'est le "DatasourceListener" qui est en charge d'effectuer cette tâche. De plus, le "DatasourceListener" demandera au contrôleur de rafraichir les données de la vue si, et seulement si, celles-ci ont changées. À ce moment, le "DatasourceListener" fournit au contrôleur les informations du changement pour que le contrôleur rafraichisse seulement les bonnes informations dans la vue.

Pour de plus ample information, à ce sujet, voir la **section 3.4** du document de conception dans l'**Annexe C**.

Module de soutien aux utilisateurs

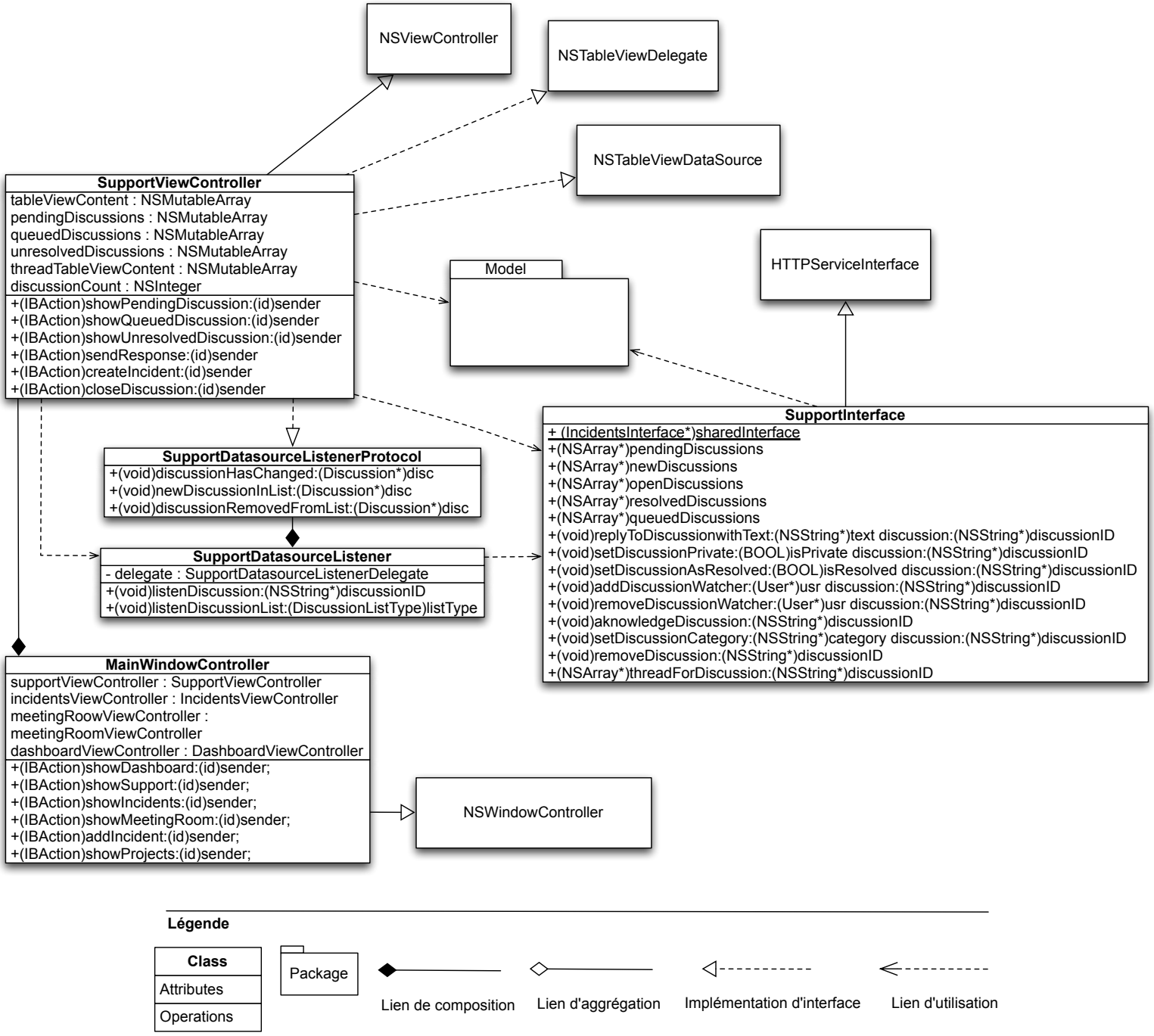


Figure 6.5 - Diagramme de classe du module de soutien aux utilisateurs

Dans ce diagramme, le coeur du module est la classe SupportViewController. Comme discuté plus tôt, cette classe utilise un “DatasourceListener”, en l’occurrence le SupportDatasourceListener, pour s’assurer de continuellement afficher des données à jour dans la vue. Le SupportViewController doit alors implémenter le protocole relié au “DatasourceListener”, le SupportDatasourceListenerProtocol, afin de gérer ce qui se passe lorsque des informations

affichées dans la vue ont changé sur le serveur. Pour plus de détails sur ce module, voir la **section 4.3** du document de conception, dans l'**Annexe C**.

Module des incidents

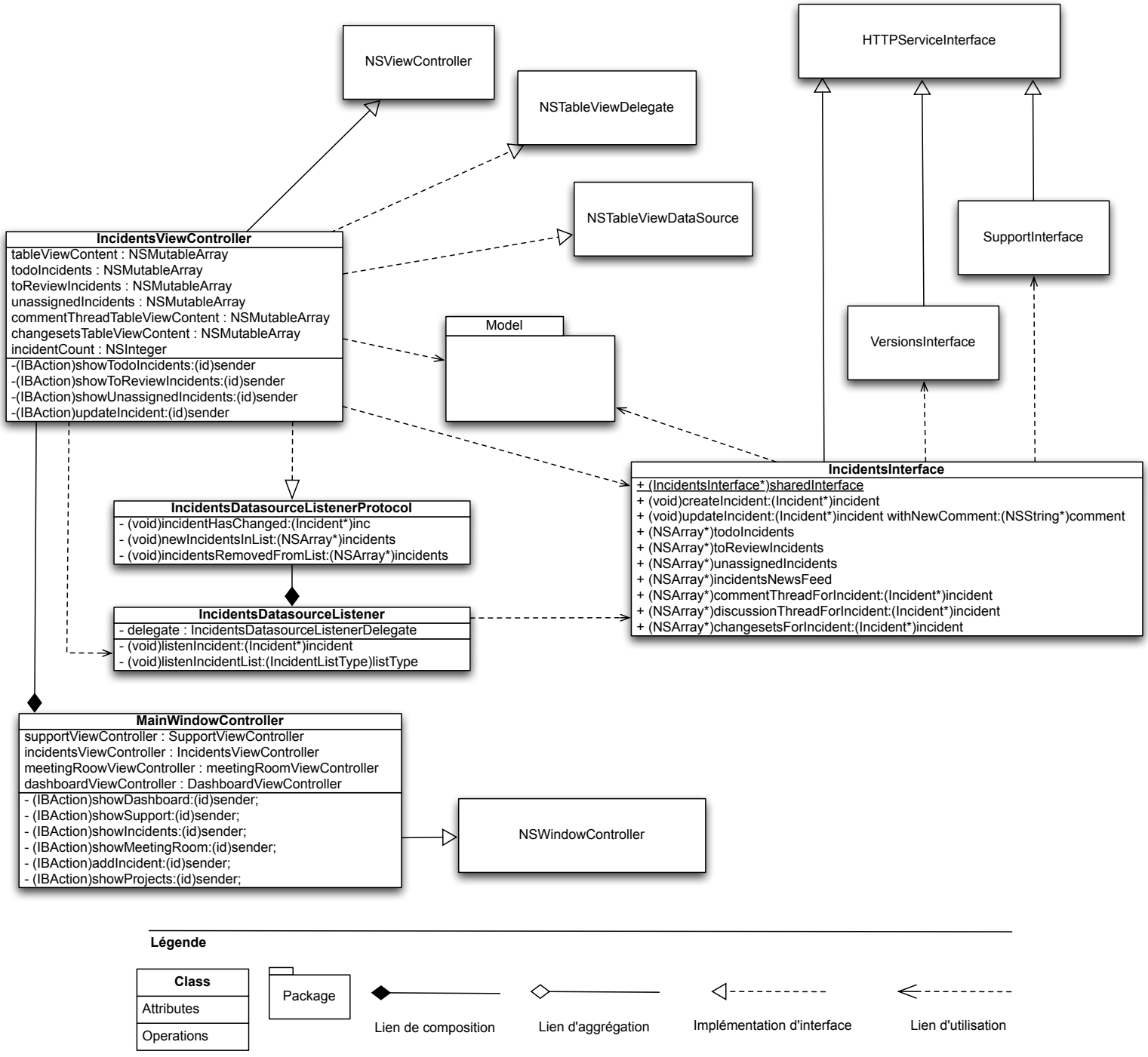


Figure 6.6 - Diagramme de classe du module des incidents

Évidemment, on constate rapidement que le module des incidents et le module du soutien aux utilisateurs se ressemblent grandement. Par contre, la principale distinction repose dans le fait que l'interface des incidents utilise l'interface du soutien aux utilisateurs ainsi que l'interface de versions. Ceci s'explique par le fait que dans le module des incidents, on peut visualiser une discussion relier à un incident ainsi que les changements SVN reliés à un incident. Mis à part cette particularité, le module des incidents est bâti sous la même forme que le module de soutien. Pour de plus amples détails concernant ce module, consulter la **section 4.4** du document de conception, dans l'**Annexe C**.

Module de clavardage

Dans le module de clavardage, on retrouve deux vues distinctes. La première est toujours visible et est incrustée à même la fenêtre principale. Elle consiste à afficher les membres de l'équipe ainsi que leur état, à savoir s'ils sont en ligne ou non. La deuxième vue est tout simplement la "salle de réunion", là où les membres de l'équipe peuvent clavarder entre eux. Pour la première vue, on retrouve essentiellement le même principe de contrôleur de vue utilisant un "DatasourceListener" et l'interface du service externe de clavardage. Supposant que ce concept est maintenant maîtrisé grâce aux deux modules précédents, le diagramme suivant démontre un autre aspect du module de clavardage. Pour voir la conception du module de clavardage en détail, consulter la **section 4.5** de l'**Annexe C**.

Étant donné qu'un système de clavardage est un système complexe en soi dû au fait qu'il utilise un protocole particulier, celui-ci n'a pas été complètement implémenté dans le logiciel. Il aurait été trop ardu d'implémenter ce protocole particulier dans l'application en plus de tout le reste. Comme solution alternative, une simple vue web fut incrustée dans l'application, comme si on accédait au service via un fureteur. Cependant, une des exigences concernant le clavardage consistait à être notifié des nouveaux messages, comme il se veut de tout bon logiciel de clavardage. Étant donné qu'une vue web était utilisée, la logique du diagramme suivant fut mise en place. Encore une fois, pour plus de détails, consulter la **section 4.5** du document de conception dans l'**Annexe C**.

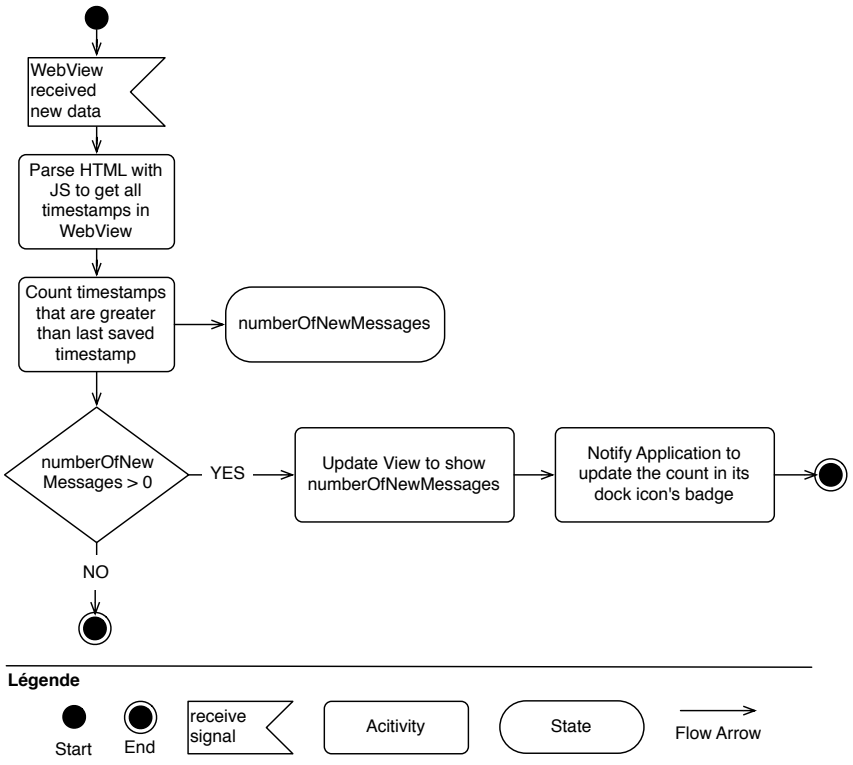


Figure 6.7 - Détection de nouveaux messages de clavardage

Chapitre 7
Implémentation du prototype

Résumé des accomplissements

Pratiquement tout ce qui est discuté dans la spécification des exigences (SRS - **Annexe A**) a été implémenté tout en respectant, à quelques exceptions près, la conception élaborée dans le document de conception (Design Document - **Annexe C**).

Plus concrètement, voici, sous forme de liste, ce qui a été implémenté :

- Module de soutien aux utilisateurs
  - Consultation de trois listes de discussions
    - Discussions en attente de réponse (“Pending”)
    - Discussions observées (“Queued”)
    - Discussions non résolues (“Unresolved”)
  - Consultation d’une discussion
  - Réponse à une discussion
  - Ajouter un observateur à une discussion
  - Fermeture d’une discussion
  - Création un incident à partir d’une discussion
  - Affichage du nombre de discussions à traiter
- Module des incidents
  - Consultation de trois listes d’incidents
    - Incidents à traiter (“To Do”)
    - Incidents en attente de revue de code (“To Review”)
    - Incidents non assignés (“Unassigned”)
  - Consultation d’un incident
    - Consultation des commentaires
    - Consultation de la discussion reliée à un incident
    - Consultation des changements SVN reliés à un incident
  - Mise à jour des informations d’un incident
  - Commenter un incident
  - Création d’un incident
  - Affichage du nombre d’incidents à traiter
- Module de clavardage
  - Clavardage de groupe avec l’équipe
  - Affichage des membres de l’équipe et de leur état
  - Notification des nouveaux messages
- Général
  - Filtre par projet
  - Affichage du nombre d’éléments total à traiter
  - Dashboard (flux de nouvelles)

Éléments non implémentés

D'autre part, certains éléments prévus dans l'analyse des exigences n'ont pas été implémentés.

En effet, un outil de constructeur de message de propagation SVN était prévu (UC12 du SRS - **Annexe A**), et n'a pas été implémenté. L'idée derrière cette exigence consistait à pouvoir construire des messages de propagation SVN à l'aide d'une boîte d'outils visuels permettant du glisser-déplacer dans la boîte de dialogue du message de propagation SVN. À l'aide d'une telle boîte à outils, on aurait pu construire facilement, avec du glisser-déplacer, un message sous la forme de "[incident:54] [assignee:anthony] [state:toreview]". Jusqu'à présent, tout ça aurait pu être implémenté sans trop de souci. Par contre, on aurait voulu que ce que le système traite automatiquement les messages de propagation pour mettre à jour les informations d'un incident. Concrètement, le message de propagation donné en exemple aurait eu comme objectif de lier le changement SVN à l'incident #54 et d'assigner cet incident à l'utilisateur Anthony pour qu'il fasse une revue de code. C'est cette partie qui pose problème. Le logiciel développé est une application cliente. Elle ne peut donc pas s'occuper de traiter tous les messages de propagation reçus sur le serveur SVN. Pour qu'une telle chose soit possible, il aurait fallu avoir un logiciel côté serveur, ayant la responsabilité d'écouter les nouveaux messages de propagation SVN, et d'effectuer les modifications nécessaires sur les incidents.

Le "dashboard" quant à lui a été implémenté, mais de façon incomplète. Tout ce qui a été fait à ce sujet, c'est obtenir les flux RSS du service de gestion de version, Beanstalk, et du service de gestion des billets, Lighthouse. Tous deux offrent des flux RSS résumant les dernières opérations effectuées par les utilisateurs. En fusionnant ces deux flux, et en triant les éléments par date, on obtient un flux de nouvelles de tout ce qui s'est produit dernièrement par rapport aux incidents. Par contre, l'affichage de ces nouvelles a été fait de façon qu'à n'afficher que le corps du message, ce qui donne une liste d'éléments très ennuyeuse. Il aurait été intéressant d'afficher l'utilisateur responsable, le projet concerné avec un code de couleur, un pictogramme indiquant le type de message, la date, etc. De cette façon, le "dashboard" aurait poussé davantage les utilisateurs à consulter son contenu. Bien que tout ça n'a pas été fait, la partie qui a été implémentée prouve que le concept de "dashboard" est réalisable dans le logiciel.

Autres faits saillants

Interface pour SVN

L'API Rest de Beanstalk était censé combler toutes les fonctionnalités reliées à SVN dans le logiciel. La fonctionnalité principale reliée à SVN est celle d'afficher les changements reliés à un incident. Techniquement, cette tâche consiste à trouver les propagations dont le message contient une chaîne de caractère respectant la syntaxe "[#incidentid". Concrètement pour l'incident #58, on recherche "[#58". Cette syntaxe a été choisie pour respecter la syntaxe qui était déjà utilisée pour lier les propagations aux billets dans Lighthouse. On pouvait faire des opérations comme "[#58 state:resolved]", d'où la recherche du crochet ouvrant seulement. Enfin, l'API Rest de Beanstalk permet d'obtenir la liste complète des propagations pour un projet, mais ne permet pas d'en recevoir plus que 30 par requête HTTP. Donc, pour traiter toutes les propagations SVN d'un projet, il faut plusieurs requêtes HTTP, dépendamment du nombre total de propagations. Alors, plutôt que de faire ainsi, ce qui semble très lourd comme procédé, des commandes SVN comme on exécute habituellement dans un terminal sont exécutées via le code de l'application, trouvant ainsi tous les propagations contenant la chaîne de caractère recherchée dans leur message. Le tout peut se faire en une seule étape. Pour pouvoir exécuter des commandes système, c'est la classe NSTask du framework Cocoa qui est utilisée. Du point de vue de la conception, rien n'a dû être changé. Seulement l'implémentation de la méthode *(NSArray*)commitsForIncident:(Incident*)incident* de la classe *VersionInterface* a dû être modifiée. Ceci prouve alors que l'application n'est pas couplée avec les services externes. Pour davantage de précision, par rapport à l'implantation des commandes SVN, consulter la **section 4.6** du document de conception dans l'**Annexe C**.

Connexion à FileMerge

Une fois les propagations SVN affichées dans l'interface usager, un autre défi d'implémentation consistait à afficher les modifications apportées au code source lors de la propagation. Pour se faire, la commande *svn diff* fut utilisée, en spécifiant un logiciel utilitaire pour effectuer la comparaison, le logiciel FileMerge. Ce logiciel est fourni par défaut avec l'environnement de développement Xcode d'Apple, alors tous les utilisateurs l'ont nécessairement d'installer sur leur machine. Pour pouvoir exécuter une telle commande, la classe NSTask est encore utilisée. À noter que l'utilitaire FileMerge se nomme "opendiff" en ligne de commande. Voici concrètement ce à quoi peut ressembler la commande pour afficher les modifications apportées au fichier *file.h* du projet *projetABC* lors de la propagation de la révision *34* :

```
svn diff --diff-cmd http://mysvnserver.com/svn/projetabc/file.h -x opendiff /Applications/Incidents.app/Contents/Resources/svndiff -r 34:33
```

En bleu on retrouve donc les paramètres entrés dans la commande lors de l'exécution. Voici les valeurs des paramètres :

1. L'URL complète du fichier dont l'on veut voir les modifications sur le serveur SVN
2. Le chemin absolu du fichier faisant le lien vers opendiff (le fichier est copié lors de la phase de construction de l'application, on a donc pas à ce soucier de cet emplacement, l'application sait où le fichier se trouve)

Contenu du **fichier**[10] svndiff :

```
#!/bin/bash
BIN=$1
shift 5
$BIN "$@"
```

3. La révision de la propagation
4. La révision antérieure avant cette propagation (révision de la propagation -1)

Enfin, l'exécution de la commande résultera en l'ouverture de FileMerge, montrant les modifications :

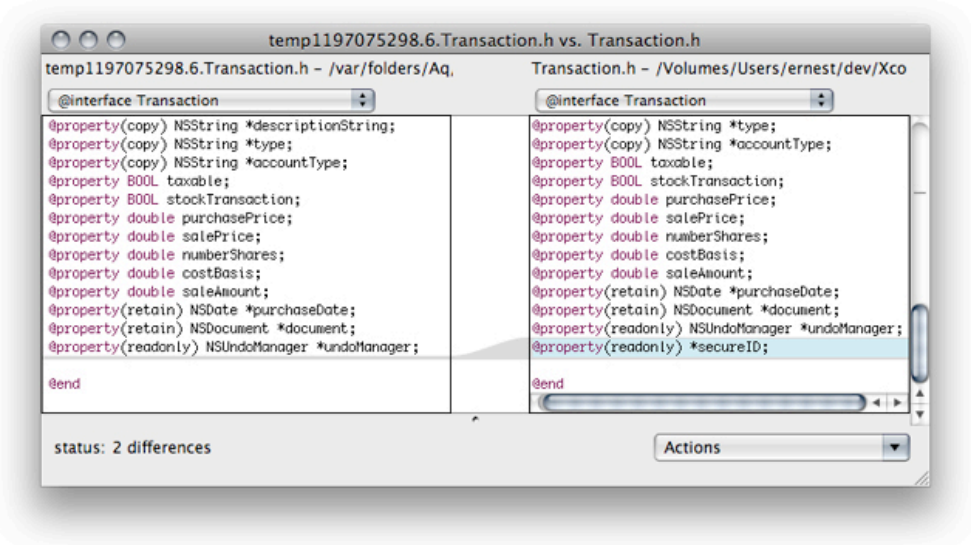


Figure 7.1 - Exemple d'utilisation de FileMerge

[10] Wrapper pour utilitaire svn-diff - *https://modelingguru.nasa.gov/docs/DOC-1944*

Événements asynchrones avec Grand Central Dispatch

Comme le logiciel tire ses données via des services web, il serait inacceptable que l'interface utilisateur soit bloquée pendant que le logiciel est en train de télécharger des données avant de les afficher. Alors, des mécanismes ont dû être mis en place. Traditionnellement, une méthode de "callback" est appelée lorsqu'une opération longue se termine, ce qui a pour effet de multiplier grandement le nombre de méthodes, pour parfois rendre le code difficile à suivre. De plus, il aurait fallu spécifier une telle méthode de "callback" à chaque fois qu'une méthode d'une des interfaces aux services externes aurait été appelée. En d'autres mots, il aurait été impossible de faire ceci :

```
Incident *incident = [ExternalServiceInterface getIncidentWithId:123];
[self updateUIWithIncident:incident];
```

Alors, quelque chose du genre aurait dû être mis en place :

```
[ExternalServiceInterface getIncidentWithId:123 callback:@selector(updateUIWithIncident:)];
```

Bien que la deuxième façon de faire prenne une ligne en moins, il est plus naturel de programmer de la première façon étant donné que la méthode *getIncidentWithId* retourne un objet de type *Incident*. Heureusement, Apple a développé la technologie Grand Central Dispatch (GCD) qui facilite énormément la gestion des processus légers et le parallélisme lors du développement. Grâce à cette technologie, l'implémentation de la première façon est possible en ajoutant une notion de blocs :

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0), ^{
    Incident *incident = [ExternalServiceInterface getIncidentWithId:123]
    dispatch_async(dispatch_get_main_queue(), ^{
        [self updateUIWithIncident:incident];
    });
});
```

Dans l'exemple ci-dessus, deux blocs sont imbriqués. Le premier bloc demande à GCD que son contenu soit exécuté dans un fil d'exécution en arrière-plan, et le deuxième bloc demande à GCD que son contenu soit exécuté sur le fil d'exécution principal, soit celui de l'interface utilisateur. Ainsi, l'interface utilisateur sera rafraîchie dès que l'objet *Incident* sera obtenu via l'interface du service externe, comme si on avait implémenté un système de "callback".

Captures d'écran

Figure 7.2 - Formulaire d'authentification la première fois qu'on ouvre Incidents

Figure 7.3 - Nombre d'éléments total à traiter, affiché en permanence sur l'icône du Dock de Mac OS

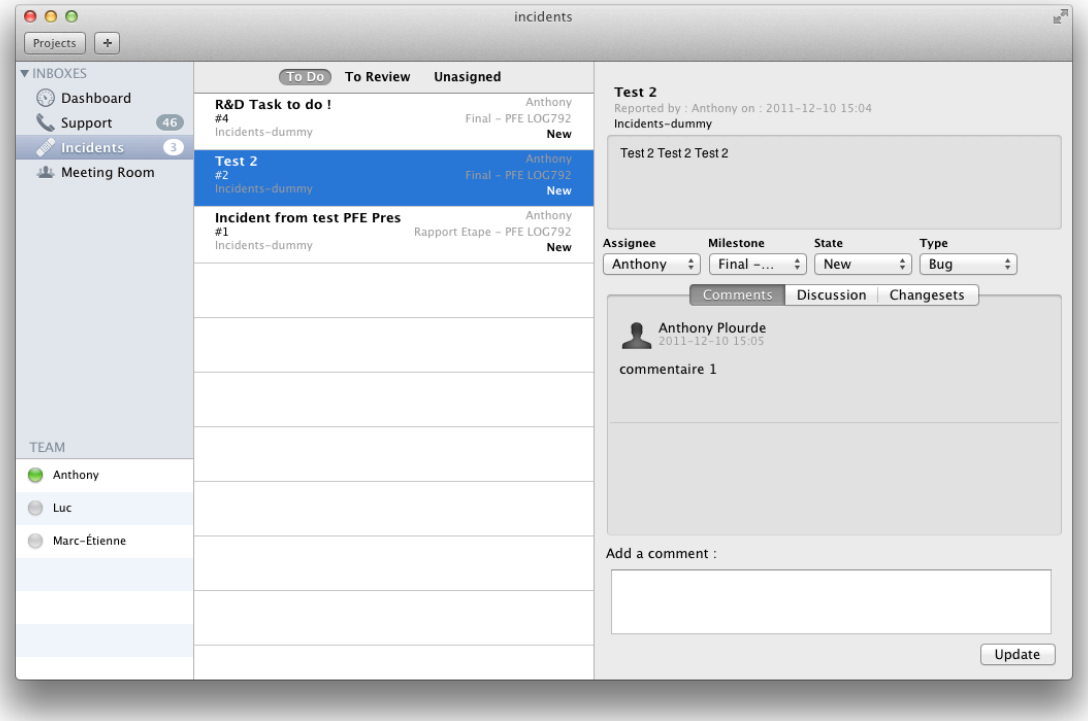


Figure 7.4 - Module des incidents, affichage d'un incident et ses commentaires

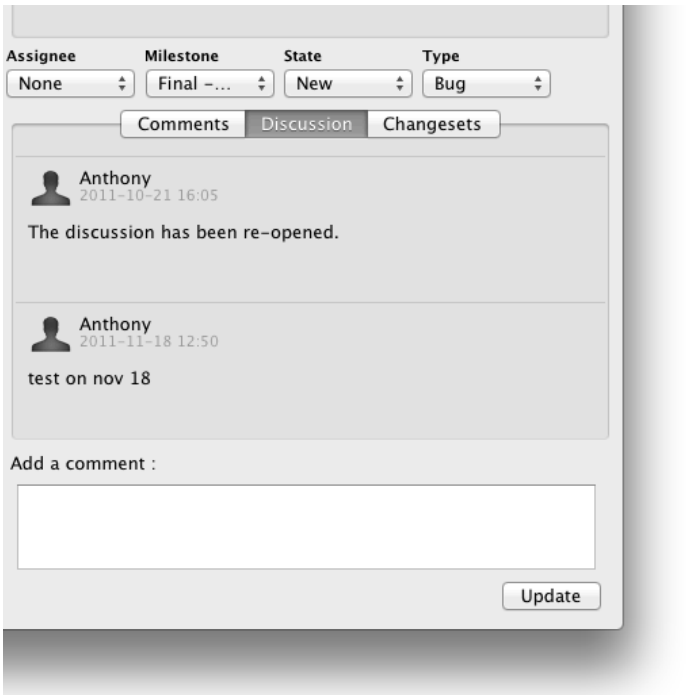


Figure 7.5 - Module des incidents, affichage d'un incident et sa discussion

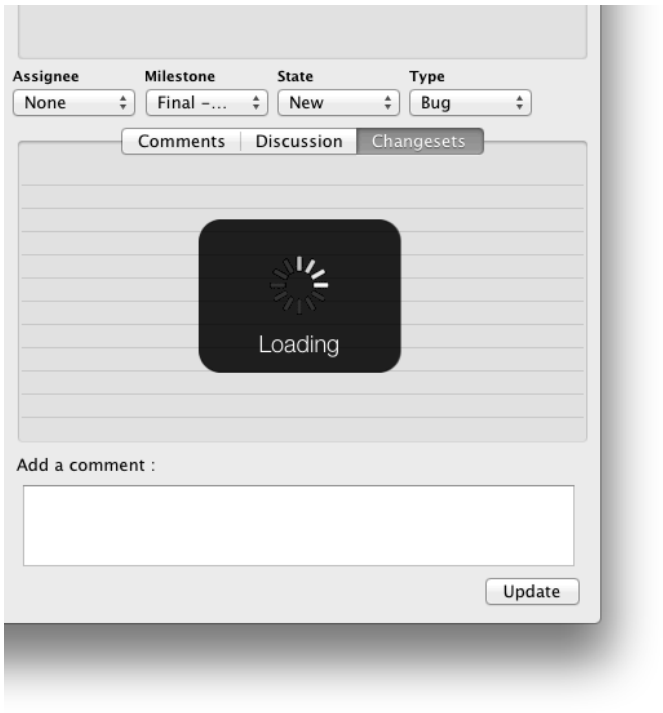


Figure 7.6 - Module des incidents, affichage d'un incident et chargement des changements SVN

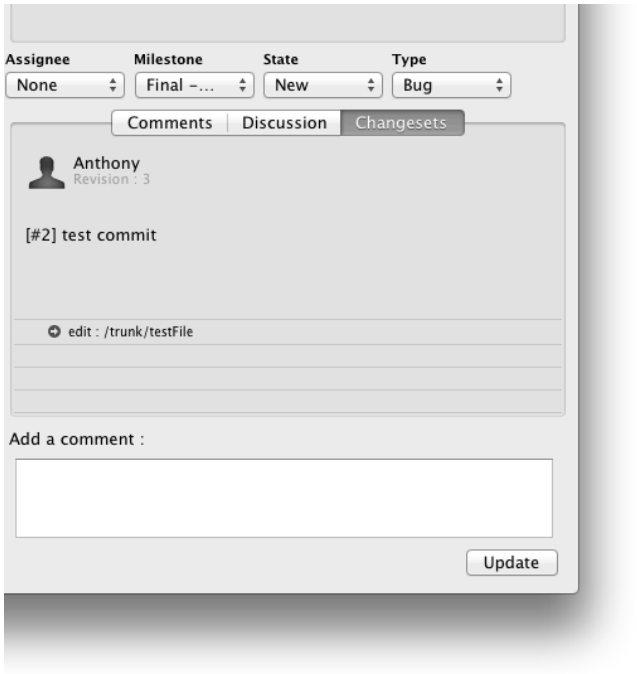


Figure 7.7 - Module des incidents, affichage d'un incident et ses changements SVN. Le fichier "testFile" a été édité lors de la révision 3. L'utilisateur peut visualiser les changements en cliquant sur la flèche.

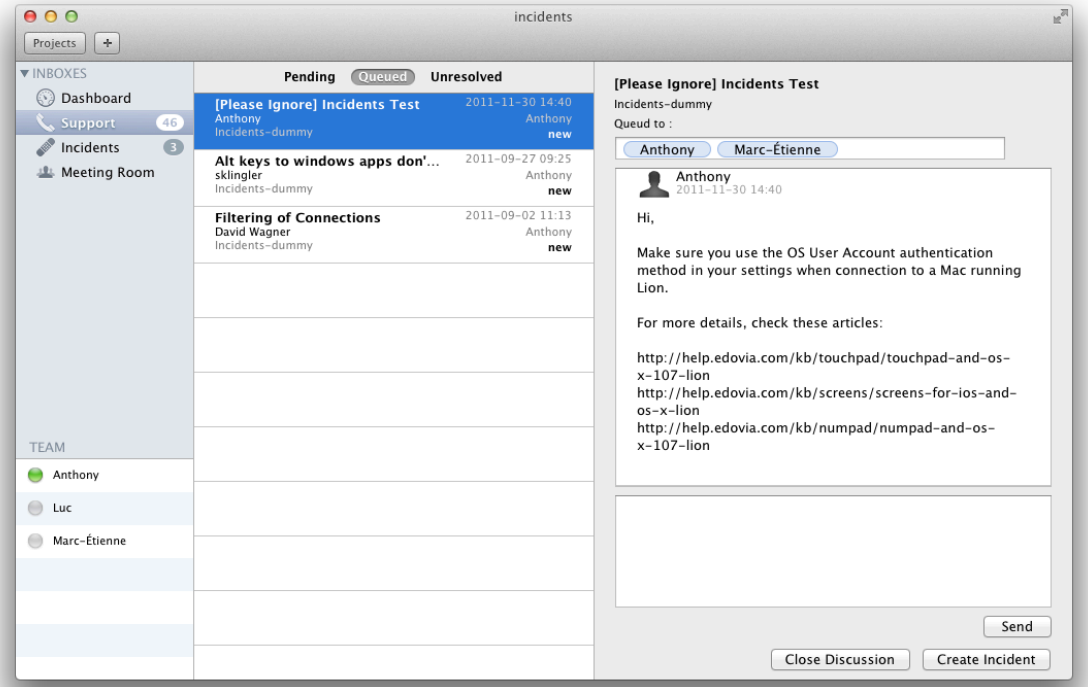


Figure 7.8 - Module de soutien aux utilisateurs, affichage d'une discussion

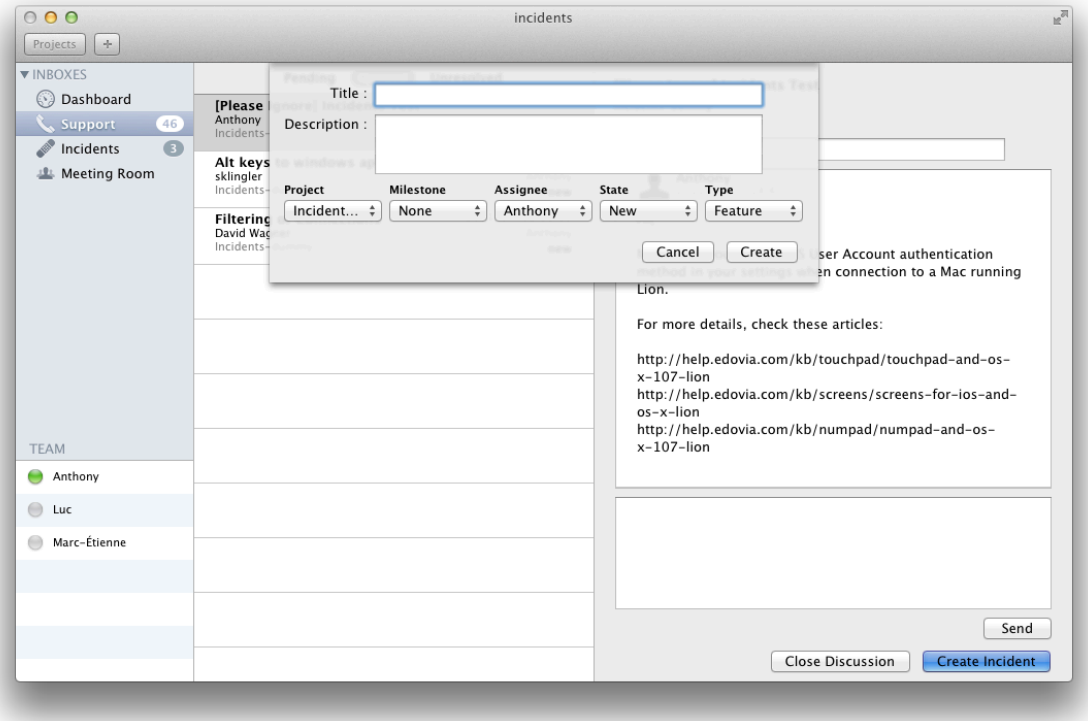


Figure 7.9 - Formulaire d'ajout d'un incident

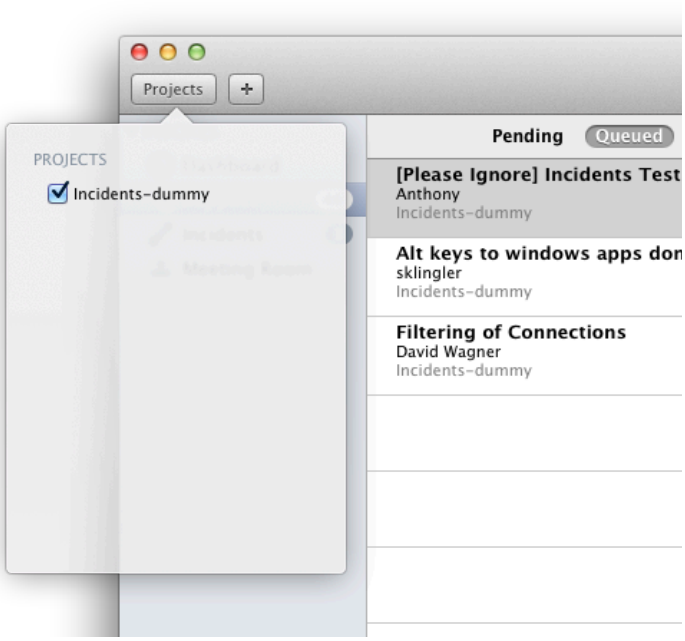


Figure 7.10 - Filtre de projet. Dans un contexte réel, plusieurs projets s'y retrouvent.

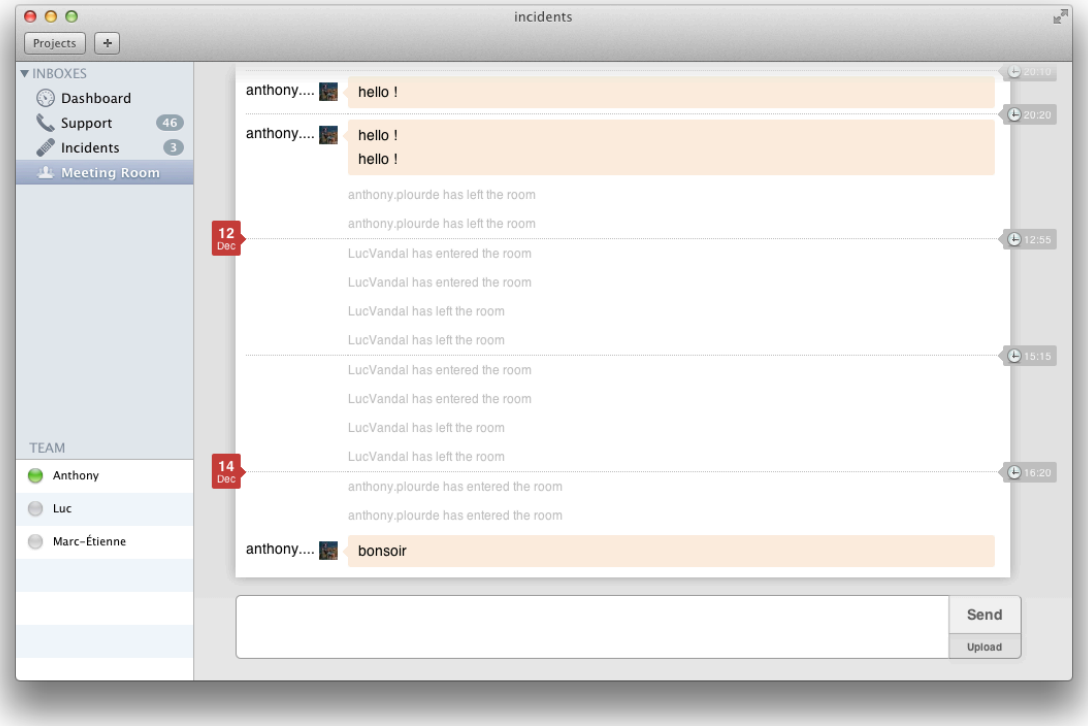


Figure 7.10 - Module de clavardage

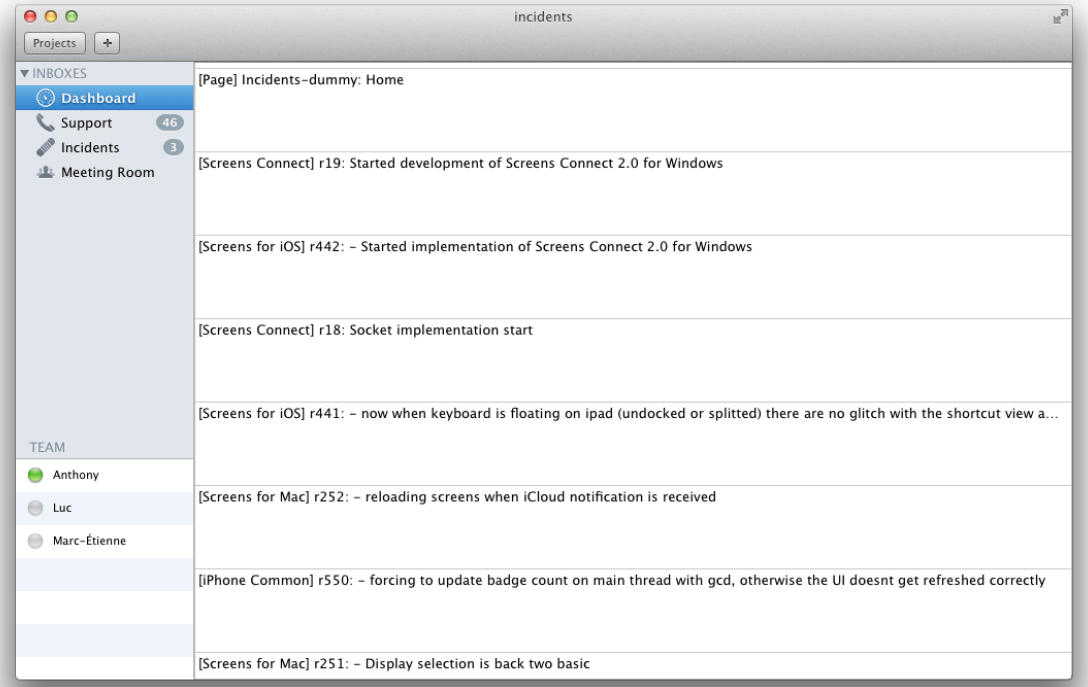


Figure 7.8 - Dashboard

Chapitre 8
Futur du projet

Maintenant que le projet s'est résulté en une application fonctionnelle, l'entreprise Edovia peut donc l'utiliser pour améliorer la gestion de son processus de développement. L'application fut utilisée dans les dernières semaines par certains développeurs de l'entreprise, avec une appréciation notable.

Cependant, Edovia perdra ces deux développeurs cet hiver. Luc Vandal, le fondateur et principal développeur se retrouvera donc seul au sein l'entreprise. L'utilisation d'*Incidents* sera probablement compromise. L'entreprise a déjà entrepris le remplacement de certains services externes, soit le gestionnaire de billet *Lighthouse* et le serveur de dépôt SVN *Beanstalk* pour héberger soi-même des services équivalent, mais gratuit. Les interfaces à ces services externes dans *Incidents* devraient donc être réécrites pour s'assurer de la compatibilité de l'application. Bien entendu, Edovia ne tirera avantage du logiciel qu'au moment où il y aura, à nouveau, plusieurs développeurs à son service. Également, comme l'entreprise n'a pas encore expérimenté à fond l'application, il se peut qu'elle se rende compte que certains aspects sont manquants, ou que d'autres sont superflus. Des ajustements devraient alors être faits.

Enfin, l'entreprise a accordé tous les droits sur l'idée et les sources du logiciel à son développeur principal, en l'occurrence moi-même, l'auteur de ce document, à en faire ce qu'il voulait. Il sera donc possible de pousser l'idée un peu plus loin en se détachant des services externes et en implémentant un logiciel côté serveur qui remplacera tous les autres. Cependant, c'est une éventualité qui devra être évaluée avant d'être concrétisée.

Conclusion

Ce projet a été fait dans le cadre du cours LOG792, projet de fin d'études en génie logiciel. Ce cours visait principalement à effectuer l'analyse et la conception d'un logiciel au choix et de façon autonome. C'est ce qui a été fait, en réalisant le logiciel *Incidents,* un outil qui simplifie l'application et le suivi des processus de développement chez Edovia. Le présent document avait pour but de présenter une synthèse du travail accompli. Brièvement, le travail accompli était concentré sur la réalisation des artéfacts proposés dans les trousses de déploiement d'analyse et de conception de la norme ISO/IEC 29110, soit un document de SRS, un document de cas d'utilisation, un prototype fonctionnel et un document de design. De plus, une matrice de traçabilité a été construite. Tous ces artéfacts sont inclus en annexe.

Le logiciel développé consistait à simplifier la gestion du processus de développement chez Edovia en rassemblant à un même endroit les services et informations fournis par des entités externes tels un gestionnaire de billets, un gestionnaire de soutien aux usagers, un gestionnaire de version et un outil de clavardage de groupe. Le logiciel devait également unifier ces différentes entités pour qu'elles collaborent toutes, de façon transparente, à l'accomplissement du scénario typique du processus de développement. Les exigences décrites dans la phase d'analyse ont été implémentées dans une application Mac native, tout en respectant les diagrammes élaborés lors de la phase de conception. Tout ce travail résulte donc en une application Mac fonctionnelle, qui répond aux besoins initiaux.

Logiquement, et selon quelques mesures, l'application développée améliore nettement l'efficacité du processus de développement chez Edovia. Par contre, c'est seulement à force d'utiliser le système qu'il sera possible de conclure si c'est vraiment le cas. Tout porte à croire que ce le sera.

Recommandations

Sous forme de liste, voici quelques recommandations qui sont à prendre en considération pour quiconque veut réaliser un logiciel semblable, intégrant des services web externes, des téléchargements en arrière-plan, des interfaces utilisateurs Mac, un système de notification, etc.

1. Construire une classe DAO permettant l'accès aux données du service web, découplant ainsi les modules clients du service en question.

2. Éviter que la classe DAO exécute des requêtes HTTP asynchrones et des "callback", limitant ainsi la possibilité qu'une méthode retourne directement le l'objet attendu. Gérer l'asynchronisme à un niveau supérieur.

3. Idéalement, en **Cocoa**[11], utilisé Grand Central Dispatch pour effectuer vos opérations d'asynchronisme et d'exécution en arrière-plan.

4. Toujours consulter la documentation des API des services web que l'on veut intégrer avant de bâtir sa couche d'accès aux données pour éviter de se rendre compte trop tard que l'API n'offre pas ce que l'on veut à faire.

5. En Cocoa, si on ne veut pas que deux classes soient fortement couplées, mais qu'on veut qu'une classe en avertisse une ou plusieurs autres lors d'un évènement, utilisez le NSNotificationCenter.

6. En Cocoa, lorsqu'on a une boucle infinie qui roule en arrière-plan, utiliser un NSAutoreleasePool pour libérer de la mémoire tous les objets "autorelease" qui ont été créés lors de l'itération.

7. Pour créer des interfaces conviviales sous Mac, lire les "**Human Interface Guidelines**[12]" fournies par Apple, et s'inspirer des interfaces des logiciels installés par défaut dans Mac OS X.

[11] Documentation Cocoa - http://developer.apple.com/library/mac/navigation/

[12] Human Interface Guidelines
http://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/AppleHIGuidelines/Intro/Intro.html

Liste de références et bibliographie

Tous les liens de référence ont été présentés à l'aide de note de bas de page tout au long du document. Chacune de ses références est regroupée ici pour une consultation plus globale.

1. À Propos - Edovia inc. : http://edovia.com/company
2. Tender App - www.tenderapp.com
3. Lighthouse App - www.lighthouseapp.com
4. Beanstalk App - www.beanstalkapp.com
5. Talker App - www.talkerapp.com
6. ISO/IEC 29110 - http://profs.etsmtl.ca/claporte/english/VSE/index.html
7. ISO/IEC 29110 : Trousse de déploiement pour la phase d'analyse
   http://profs.etsmtl.ca/claporte/english/VSE/Deploy%20Pack/DP-Software%20Requirements%20Analysis-V1_2.doc
8. ISO/IEC 29110 : Trousse de déploiement pour la phase de conception
   http://profs.etsmtl.ca/claporte/english/VSE/Deploy%20Pack/Deployment_Software_Design_v0%205.doc
9. Exemple de SRS : Document choisi comme exemple dans le cours de LOG410, à la session d'hiver 2010
   http://dl.dropbox.com/u/1334047/SRS%20Eq13.docx
10. Wrapper pour utilitaire svn-diff - *https://modelingguru.nasa.gov/docs/DOC-1944*
11. Documentation Cocoa - http://developer.apple.com/library/mac/navigation/
12. Apple Human Interface Guidelines
    http://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/AppleHIGuidelines/Intro/Intro.html

Annexe A
SRS

Incidents
SRS01
Software Requirements Specification

Historique des révisions

Date	Description	Révision	Auteur
23-09-2011	Objectif et portée	0.9	Anthony Plourde
29-09-2011	Complétion d'un premier jet du document	1.0	Anthony Plourde
06-10-2011	Ajout de RF06 et ajout des liens de dépendance dans le diagramme de UC	1.1	Anthony Plourde
30-10-2011	Complétion des sections 8-9-10-11.	1.2	Anthony Plourde
31-10-2011	Révision, fin de l'itération 2. Changements mineurs dans les exigences	1.3	Anthony Plourde
18-11-2011	Révision, fin de l'itération 3. Changements mineurs dans les exigences	1.4	Anthony Plourde
15-12-2011	Révision finale	1.5	Anthony Plourde

1. Introduction

1.1 Objectifs

Le but de ce document est de définir le comportement extérieur du système Incidents, un outil visant à simplifier le suivi et l'application de processus de développement chez Edovia, une TPE oeuvrant dans la création d'applications iOS et Mac. Ce document décrit les exigences fonctionnelles, non fonctionnelles, les contraintes ainsi que tout autre facteur nécessaire à la compréhension des exigences du système. Ce document sert de base à la conception, à l'implémentation et aux tests du système.

1.1 Portée

L'entreprise est présentement inscrite à de nombreux services en ligne ayant pour but d'améliorer les processus de développement dans le cycle de vie logiciel. Parmi ces services, on retrouve un outil de gestion de soutien à la clientèle, un gestionnaire de billet, un gestionnaire de dépôt SVN et finalement un service de messagerie instantanée de groupe pour faciliter le travail à distance. Chacun de ces services fonctionne relativement bien et répond correctement aux besoins qu'il cible. Cependant, l'utilisation quotidienne de l'ensemble de ces services s'apparente davantage à une tâche superflue plutôt qu'à une tâche qui facilite le processus de développement pour une si petite équipe. Chacun de ces services est sous forme d'application web, ce qui ne favorise pas la consultation régulière des développeurs.

Il existe donc quatre systèmes existants :

- Tender App (gestionnaire du support à la clientèle)
- Lighthouse App (gestionnaire de billets)
- Beanstalk App (gestionnaire des dépôts SVN)
- Talker App (outil de clavardage de groupe)

Ces systèmes sont de tierce partie et resteront inchangés. L'application Incidents communiquera à ces systèmes via des interfaces génériques qui devront permettre une entière transparence des services externes utilisés. Le but d'une telle méthodologie est d'éventuellement pouvoir remplacer les services payants par des services maison.

Dans l'illustration de la page suivante, on peut voir que la portée du projet se limite à l'application native Mac et à ces interfaces communiquant avec les services externes.

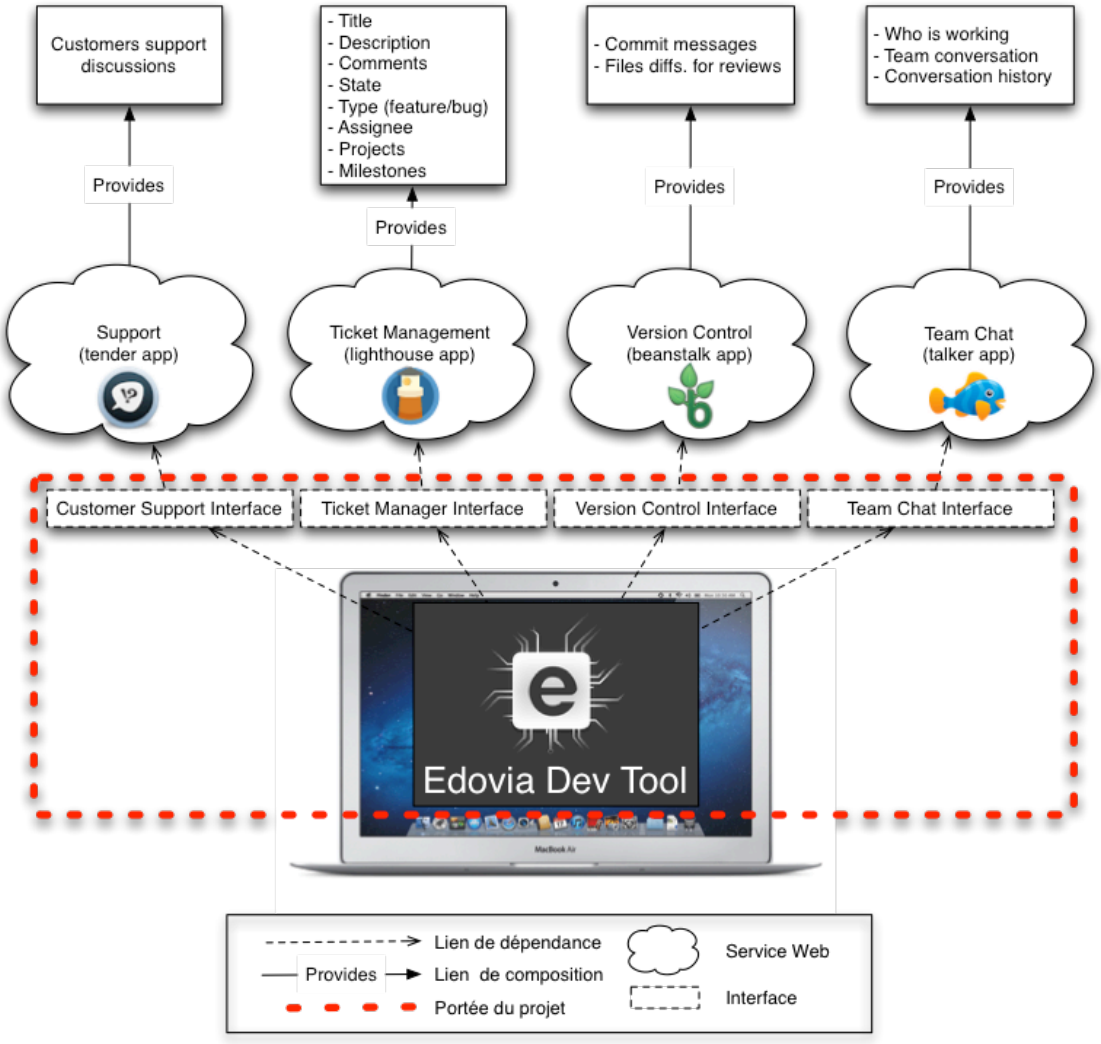


Figure 1. Portée du projet

1.3 Références

1. **Exemple de document SRS et de cas d'utilisation du cours LOG410**
   - http://dl.dropbox.com/u/1334047/SRS%20Eq13.docx
2. **Tender App – Public API**
   - https://help.tenderapp.com/kb/api/introduction
3. **Lighthouse App – Public API**
   - http://lighthouseapp.com/api/introduction
4. **Beanstalk App – Public API**
   - http://api.beanstalkapp.com/introduction.html
5. **Talker App – Public API**
   - https://talker.tenderapp.com/kb/api/rest-api
6. **Mac Developer Program License Agreement**
   - http://developer.apple.com/programs/terms/mac/mac_program_agreement_20110606.pdf

1.4 Hypothèses et Dépendances

Pour récupérer et sauvegarder ses données, le système dépend directement des services externes Tender, Lighthouse, Beanstalk et Talker. Pour pouvoir se faire, le système dépend également des API de type REST de chacun des quatre services externes.

2. Survol du Modèle des Cas d'Utilisation

UC01 – Consulter une liste de requêtes de soutien

Description : Affichage de la liste des requêtes de soutien en attente de traitement, en observation ou non résolues.
Acteur : Utilisateur

UC02 – Consulter une requête de soutien

Description : Affichage des détails d'une requête de soutien suite à la consultation d'une liste.
Acteur : Utilisateur

UC03 – Répondre à une requête de soutien

Description : Permet à l'utilisateur de répondre à une requête de soutien via l'affichage de cette dernière.
Acteur : Utilisateur

UC04 – Ajouter un membre comme observateur

Description : Permet l'ajout d'un membre de l'équipe en tant qu'observateur pour que celui-ci soit notifié des changements. Ceci peut s'appliquer autant pour une requête de soutien ou pour un incident.
Acteur : Utilisateur

UC05 – Terminer une requête de soutien

Description : Lorsqu'une discussion est terminé avec un client ayant fait une requête de soutien, la requête peut-être marquée comme terminée.
Acteur : Utilisateur

UC06 – Créer un incident

Description : À partir d'une requête de soutien relevant un bogue ou une demande de fonctionnalité, un incident peut être créé pour qu'un membre de l'équipe règle le bogue ou implémente la fonctionnalité. Un incident peut-être également créé à partir de rien, dans le cas d'une fonctionnalité ou d'un bogue provenant de l'interne.
Acteur : Utilisateur

UC07 – Modifier un incident

Description : Suite à la création d'un incident, celui-ci peut être modifié soit pour l'apport d'une correction ou simplement pour l'avancement de l'incident dans le processus (changement de statut).
Acteur : Utilisateur

UC08 – Consulter une liste d'incidents

Description : Affichage de la liste des incidents à faire, en attente de revue ou non assignés.
Acteur : Utilisateur

UC09 – Consulter un incident

Description : Affichage des détails d'un incident suite à la consultation d'une liste.
Acteur : Utilisateur

UC10 – Effectuer une revue de code

Description : S'exécute lorsqu'un incident qui se trouve dans la liste des incidents en attente de revus est traité. Le traitement consiste à faire la revue des fichiers de code source modifié et par le changement de statut de l'incident soit de "en revue" à "résolu" dans le cas où les modifications passent la revue, ou de "en revue" à "en cours" dans le cas où les modifications ne passent pas la revue. Un message expliquant les raisons peut être ajouté en commentaire dans le dernier cas.
Acteur : Utilisateur

UC11 – Écrire dans la salle de réunion

Description : Dans la salle de réunion (clavardage de groupe), lorsque l'on écrit un message.
Acteur : Utilisateur

UC12 – Construire un message de propagation

Description : Le système mettra à la disposition de l'utilisateur un outil de construction de message de propagation pour aider l'utilisateur à construire son message de propagation pour qu'il puisse être lié à un incident et même mettre à jour certaines informations de l'incident comme son statut et la personne à qui il est assigné. Le tout effectué avec du glisser-déposer.
Acteur : Utilisateur

UC13 – Commenter un incident

Description : Lorsqu'un incident est affiché, il est possible d'ajouter un commentaire qui sera visible à tout le monde qui consulte l'incident.
Acteur : Utilisateur

UC14 – Filtrer l'affichage par projet

Description : Lorsque l'utilisateur veut seulement travailler sur un(des) projet(s) en particulier, il pourra sélectionner les projets pour lesquels il veut voir s'afficher les incidents et les requêtes de soutien.
Acteur : Utilisateur

2.1 Diagramme de cas d'utilisation

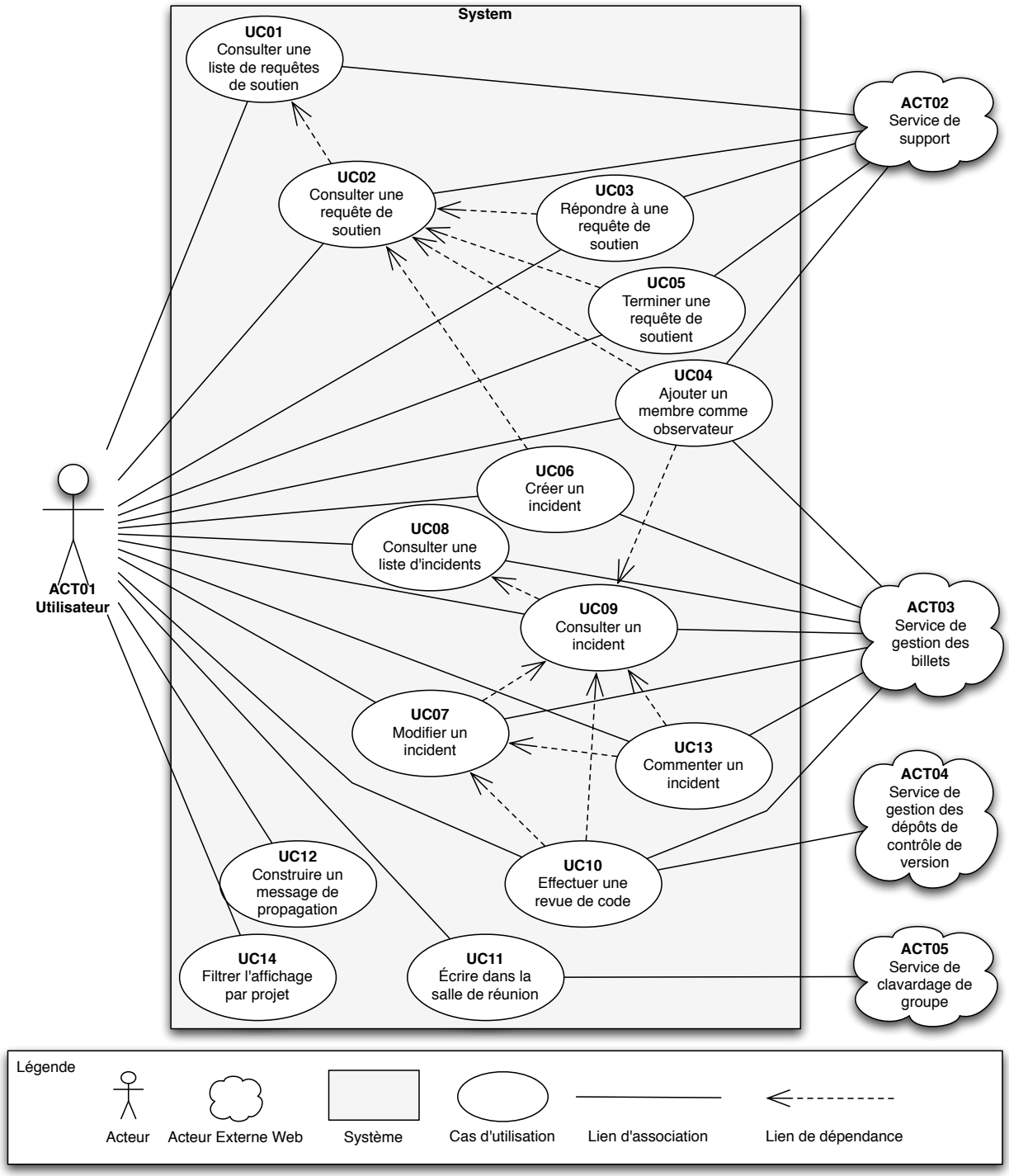


Figure 2. Diagramme de cas d'utilisation

3. Les acteurs

ACT01 – Utilisateur

La personne qui utilise l'outil de développement, un membre de l'équipe d'Edovia.

ACT02 – Service de support

Composante logicielle web qui stocke et fourni les données des requêtes de soutien.

ACT03 – Service de gestion des billets

Composante logicielle web qui stocke et fourni la majorité des données des incidents.

ACT04 – Service de gestion des dépôts de contrôle de version

Composante logicielle web qui stocke et fourni les données des changements du code source.

ACT05 – Service de clavardage de groupe

Composante logicielle web qui stocke et fourni les données des discussions de l'équipe.

4. Les exigences

4.1 Les exigences fonctionnelles

REF01 - Le système doit charger au démarrage la liste des projets et des utilisateurs présents dans la base de données.

REF02 - Le système doit charger au démarrage l'état de l'application comme il était à sa dernière utilisation.

REF03 - Le système doit afficher en tout temps le nombre de requête de soutien, d'incident de revue de code en attente de traitement et étant assignés à l'utilisateur en cours.

REF04 - Le système doit rafraichir les données affichées régulièrement pour refléter le plus fidèlement les données distantes.

REF05 - Le système doit notifier l'utilisateur lorsque des messages de clavardage sont reçus et que l'utilisateur ne se trouve pas dans la salle de discussion. Au même titre, le nombre de messages non lus doit être affiché en tout temps.

REF06 - Le système doit en tout temps afficher seulement les informations relatives aux projets qui sont sélectionnés par l'utilisateur dans la liste des projets.

REF07 - Le système doit en tout temps afficher le nombre d'éléments à traiter, toutes catégories confondues, dans un badge sur l'icône de l'application dans le dock du Mac.

4.2 Les exigences non fonctionnelles

4.2.1 Facilité d'utilisation (Usability)

RENF01 - Le système doit permettre à l'utilisateur de consulter tout type de travail à effectuer en 1 clic seulement.

RENF02 - Le système doit permettre à l'utilisateur de connaitre sa charge de travail complète (nombre d'items à traiter, tout type confondu) d'un seul coup d'œil (sans aucun clic requis).

4.2.2 Fiabilité (Reliability)

RENF03 - Le système doit être tolérant aux pannes et doit pouvoir récupérer d'une telle panne en moins d'une minute.

RENF04 - Le système doit fournir une capacité à affecter et lire les données distantes 95% du temps et doit avertir l'utilisateur lors d'échec.

RENF05 - Le temps moyen de prédiction d'une défaillance du système doit être moindre que 1 fois au 6 heures.

4.2.3 Performance

RENF06 - Le système doit affecter et lire les données distantes et les afficher en moins de 5 secondes.

RENF07 - Le rafraichissement des valeurs à l'écran doit être inférieur à 20 secondes.

RENF08 - Le réception des messages de clavardage doit être inférieure à 2 secondes.

4.2.4 Facilité d'entretien (Maintenability)

RENF09 - Le système doit pouvoir intégrer un nouveau service externe ayant un API Rest en une semaine de travail.

RENF10 - Le système doit permettre le remplacement d'un ou des services externes en ne modifiant que l'interface qui interroge le service et n'ayant pas à modifier les classes du modèle.

4.2.5 Portabilité

RENF11 - Le système doit pouvoir être installé en moins de 3 minutes sur n'importe quelle machine sous MAC OS X Lion (10.7.x).

RENF12 - Le système doit prendre moins de 10 MO lors de son installation sur un disque dur.

5. Documentation en direct pour l'utilisateur et exigence du système d'aide

Étant donné que les utilisateurs de l'outil de développement sont nécessairement des développeurs, et que l'équipe de développement en question est composée de seulement trois individus, aucune documentation d'aide aux utilisateurs ne sera rédigée pour le moment. L'apprentissage de l'utilisation du logiciel se fera sous forme de démonstration.

6. Contraintes de conception

CON01 – Mac OS X Lion

Le système doit être compatible avec le système d'exploitation Mac OS X Lion (10.7) pour pouvoir bénéficier de nombreuses fonctionnalités de ce système d'exploitation telles que la restauration automatique, le mode plein écran et plusieurs autres composants visuels intéressants.

CON02 – Application Cocoa

Le système doit être développé en tant qu'une application native Mac (une application web ne serait pas tolérée) et doit donc être conçu avec le « framework » Cocoa d'Apple, utilisant le langage de programmation Objective-C.

CON03 – Utilisation des API Rest publics des services externes

Le système doit utiliser les API Rest des services externes (Tender, Lighthouse, Beanstalk et Talker) pour communiquer avec ceux-ci.

CON04 – Respect des HIG OS X

Le système doit respecter les lignes directrices décrites dans le document *Apple Human Interface Guidelines* [13] en ce qui a trait à ses interfaces utilisateurs.

7. Composants achetés

- Inscription à un plan de service Tender
- Inscription à un plan de service Lighthouse
- Inscription à un plan de service Beanstalk

[13] **Apple Human Interface Guidelines**
http://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/AppleHIGuidelines/OSXHIGuidelines.pdf

8. Interfaces

8.1 Interfaces utilisateurs

Les interfaces utilisateurs sont une importante partie de ce projet puisque les principaux objectifs de ce projet sont d'améliorer l'efficacité d'utilisation des services externes utilisés dans le processus de développement chez Edovia. L'interface en générale est une interface à trois panneaux verticaux, fortement inspirée de l'application Mail d'Apple.

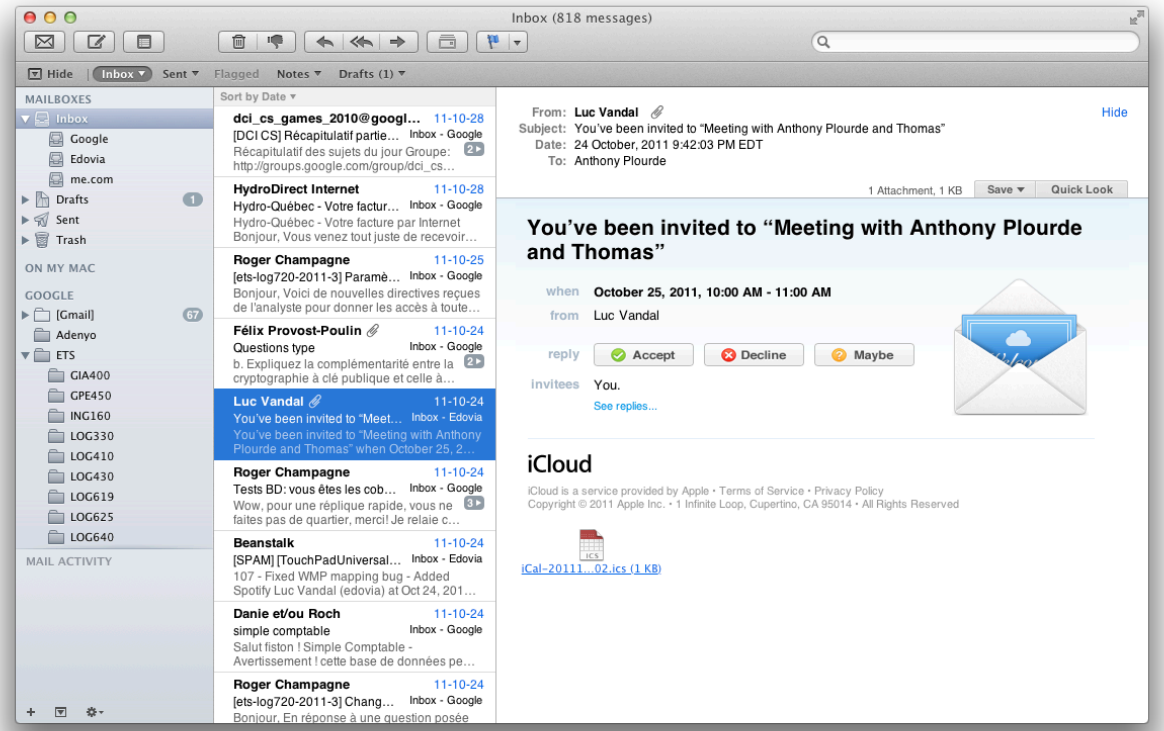


Figure 8.1 - Application Mail d'Apple et son affichage à trois panneaux

Voici d'abord une capture d'écran du prototype qui démontre bien l'interface à trois panneaux et l'explication de ces composantes :

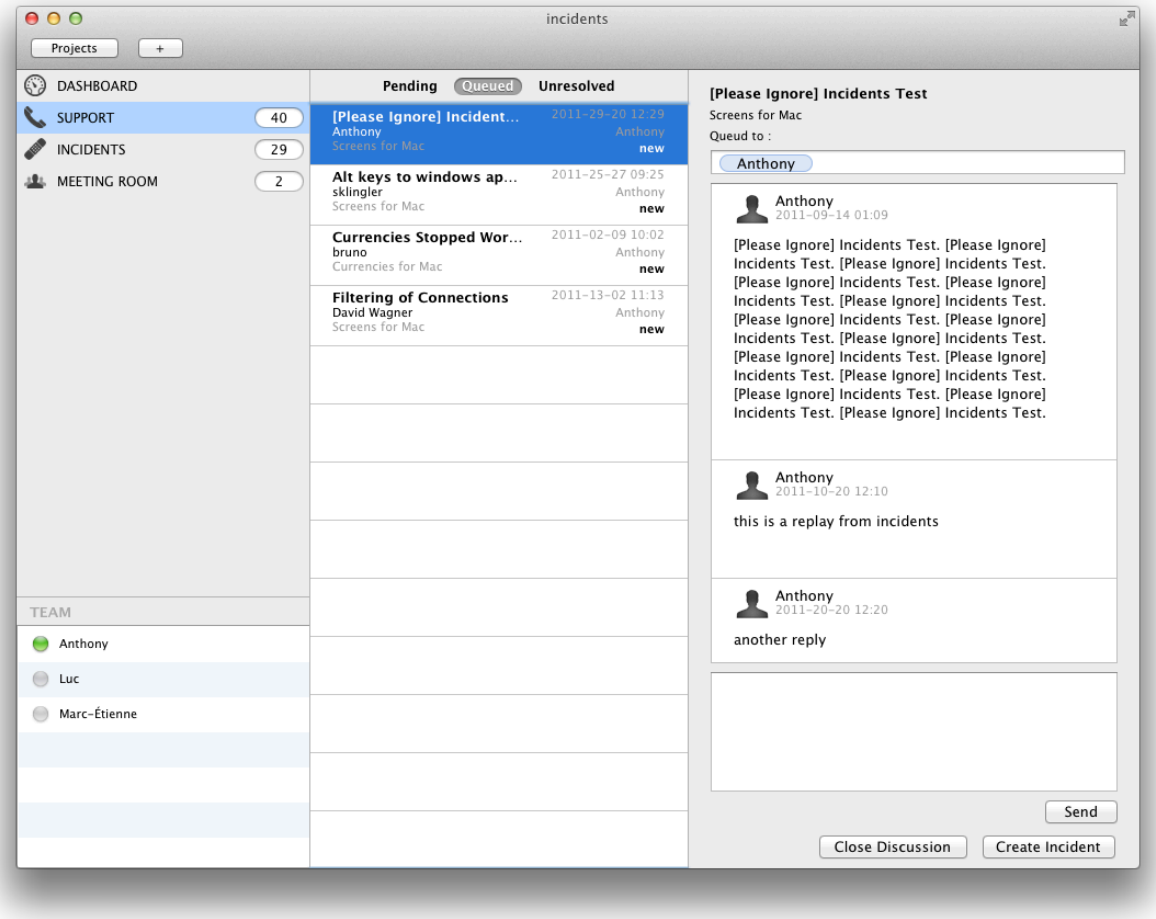


Figure 8.2 - Interface de la vue de soutien aux utilisateurs

Barre de menu de gauche

Cette barre affiche les différentes sections de l'application ainsi que le nombre d'éléments en attente de traitement pour chaque section.

État des membres de l'équipe

En bas de la barre de menu de gauche est affichée la liste des membres de l'équipe de développement d'Edovia. Un témoin vert est affiché lorsque le membre en question est connecté au système. C'est l'API de Talker qui sera utilisé pour savoir qui est connecté.

Panneau central

C'est là où sont affichés les éléments à faire de la section en cours. En haut de cette liste, différents boutons sont accessibles pour changer la sélection de la liste. Par exemple, dans la section de soutien aux utilisateurs, on peut soit voir les nouvelles requêtes de soutien en attente de traitement, les requêtes qui sont assignées à l'utilisateur en cours et celles qui ne sont pas résolues.

Panneau de détail de droite

Lorsqu'un élément de la liste du panneau central est sélectionné, son détail est affiché dans le panneau de droite. C'est également dans ce panneau qu'on peut interagir avec l'élément sélectionné. Fermer un billet ou une discussion par exemple.

Barre d'outils au haut de la fenêtre

Cette barre permet de créer un incident (sans y lier de requête de soutien) via le bouton "+". Également, cette barre permet de filtrer, quels projets sont visibles dans toute l'application. Cette barre d'outils est fortement inspirée de la barre d'outils de iCal, où l'on filtre les calendriers affichés :

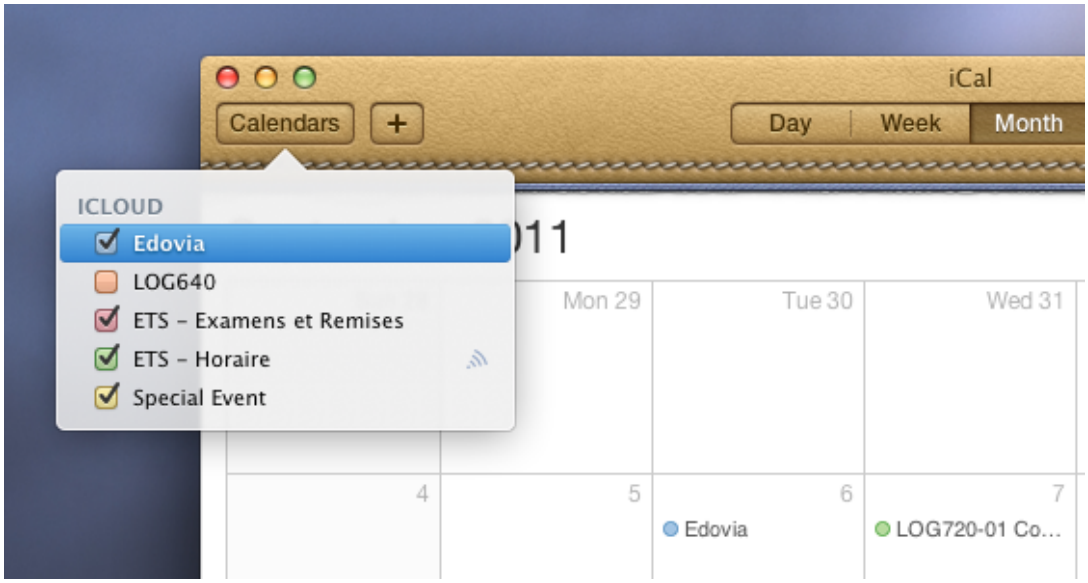


Figure 8.3 - Application iCal d'Apple et son filtre de calendrier

Voici maintenant une capture d'écran du prototype affichant le détail d'un incident. Le même principe que pour les requêtes de soutien est appliqué. Par contre, il y a un élément qui demeure à rectifier. Le détail de l'incident est bien affiché, mais pas la discussion de soutien qui y est associé, ni les fichiers modifiés lors d'un «commit» SVN. Pour régler ce problème, un panneau à onglet sera ajouté à l'endroit de la liste des commentaires, permettant ainsi de naviguer entre les commentaires, la discussion et la liste des fichiers modifiés.

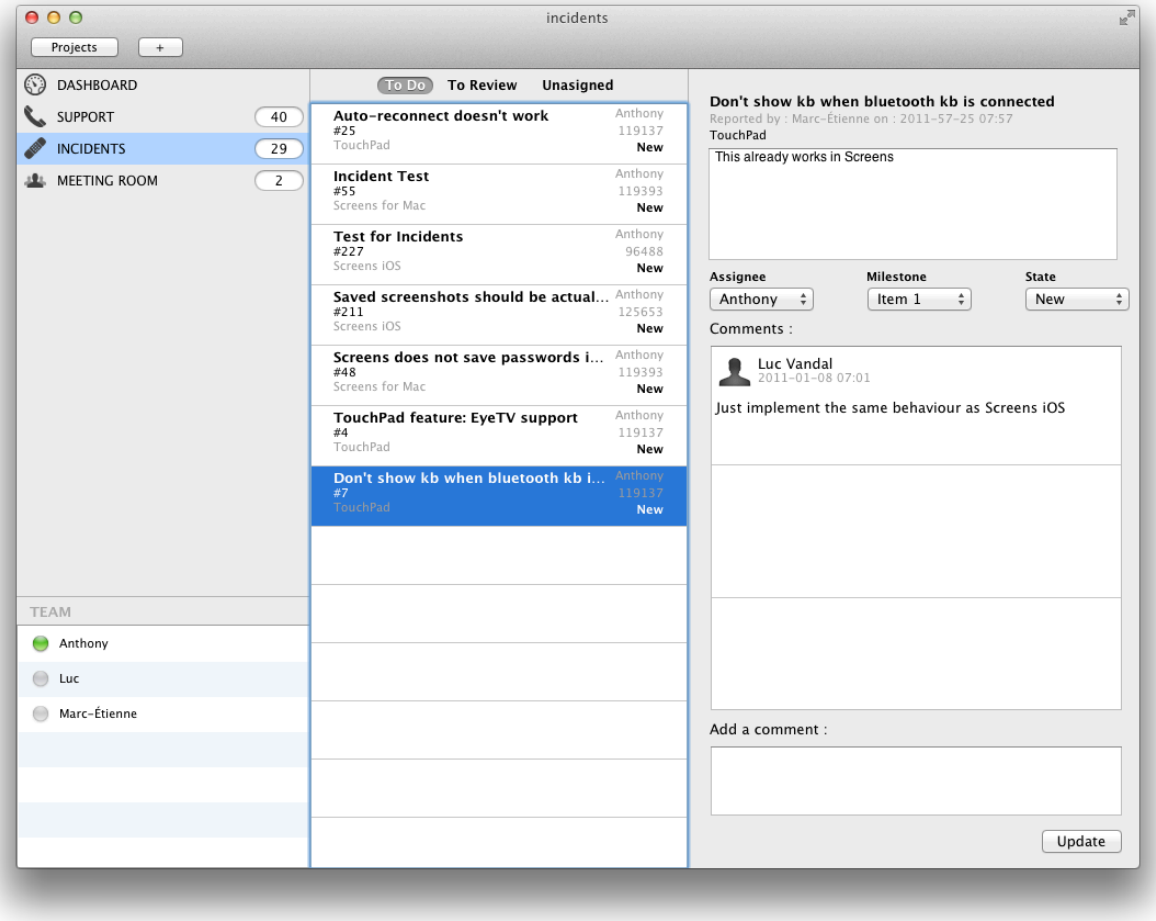


Figure 8.4 - Interface de la vue des incidents

Enfin, voici d'autres captures d'écran pertinentes :

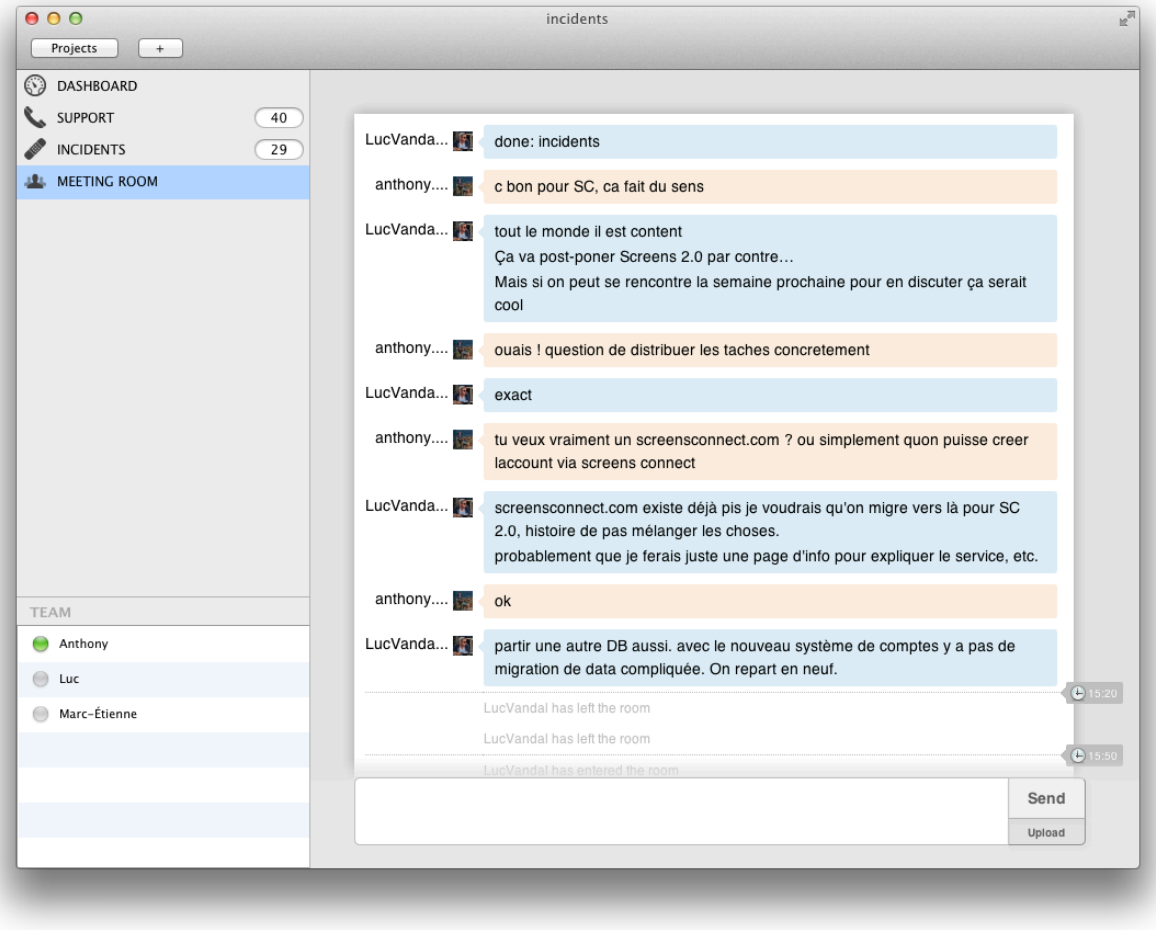


Figure 8.5 - Interface de la vue de clavardage de groupe

Figure 8.6 - Notifications globales dans le dock

8.2 Interfaces matérielles

Il n'y a aucune interface matérielle existante dans le cadre de ce projet.

8.3 Interfaces logicielles

Le système utilisera les interfaces publiques de type REST des services externes suivants : Tender, Lighthouse, Beanstalk et Talker. Ces interfaces sont bien définies et documentées sur les sites web de chacun des services. Une interface logicielle devra être mise en place dans le système pour chacun de ces quatre services afin de rendre transparente l'utilisation des services et de leur API REST. Cette interface permettra aussi au système de ne pas dépendre directement de ces services. Éventuellement, ces services pourraient être remplacés par d'autres services. Pour que le système puisse continuer d'opérer correctement, seules les interfaces devraient être ré implémentés.

8.4 Interfaces de communication

Comme le logiciel entreposera les données concernant les liens entre les différents services dans une base de données MySQL, des requêtes SQL voyageront sur le port Ethernet 3306. Outre l'interface MySQL, il n'y a aucune interface de communication particulière nécessaire au fonctionnement du système excepté les interfaces REST des services externes décrites dans la section précédente,

9. Exigences de Licences

Comme le système ne sera qu'utilisé à l'interne de l'entreprise, aucune exigence de licence n'est requise pour l'instant. Par contre, si jamais l'entreprise venait qu'à remplacer tous les services externes et décidait de vendre sa solution, elle utiliserait probablement le Mac App Store de Apple pour se faire. La licence "**Mac Developer Program License Agreement**[6]" s'appliquerait alors.

10. Remarques légales, de droits d'auteur, et diverses

Edovia permet à Anthony Plourde de disposer de tout code source, documents et diagrammes produits lors du développement du logiciel, et de l'utiliser comme il le souhaite s'il vient qu'à ne plus travailler pour l'entreprise.

11. Standards applicables

Standards de programmation Cocoa

Le code source produit dans le développement de l'application et de son prototype devra suivre ces standards.

Lien : http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CodingGuidelines/CodingGuidelines.pdf

Standards pour les interfaces Mac OS X

Les interfaces de l'application devront respecter ces standards.

Lien : http://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/AppleHIGuidelines/OSXHIGuidelines.pdf

Norme ISO/IEC 29110

La liste des artéfacts à produire durant ce projet est tirée de cette norme.

Annexe B
Document de cas d'utilisation

Incidents
UCS01
Use Cases Document

Historique des révisions

Date	Description	Révision	Auteur
29-09-2011	Création des sections	0.1	Anthony Plourde
05-10-2011	Création des UC 1 à 4	0.4	Anthony Plourde
06-10-2011	Création des UC 5 à 14	1.0	Anthony Plourde
30-10-2011	Révision, fin de l'itération 2	1.1	Anthony Plourde
18-11-2011	Révision, fin de l'itération 3	1.2	Anthony Plourde
16-12-2011	Révision finale	1.3	Anthony Plourde

Objectifs de ce document

Ce document décrit les cas d'utilisation de façon détaillée du système Incidents, un outil d'application et de suivi des processus de développement logiciel chez Edovia inc. Les cas d'utilisations suivants décrivent l'interaction entre les acteurs et le système lors de l'utilisation de l'application.

UC01 - Consulter une liste de requêtes de soutien

Précondition(s) : L'utilisateur est authentifié au service de soutien

Postcondition(s) : Aucune

Points d'extension : Aucun

Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire afficher les requêtes de soutien à traiter, les requêtes de soutien non lues ou encore celles qui lui sont assignées.
2. L'utilisateur sélectionne la section "soutien".
3. Le système demande au service de soutien la dernière liste de requêtes de soutiens affichées par l'utilisateur:
    1. les requêtes de soutien en attente,
    2. les requêtes de soutien en observation,
    3. ou les requêtes de soutien non résolues.
4. Le service de soutien envoie cette liste au système.
5. Le système affiche cette liste.
6. L'utilisateur consulte la liste.

Scénario(s) alternatif(s) :

6a. L'utilisateur désire consulter une autre liste de requêtes de soutien

6a.1. L'utilisateur sélectionne une des listes suivantes:

1. les requêtes de soutien en attente,
2. les requêtes de soutien en observation,
3. ou les requêtes de soutien non résolues.

6a.2. Retour à 5.

UC02 - Consulter une requête de soutien

Précondition(s) : Une liste de requêtes de soutien est affichée → UC01
Postcondition(s) : Aucune
Points d'extension : Aucun
Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire consulter une requête de soutien.
2. L'utilisateur sélectionne une requête de soutien à afficher.
3. Le système demande les détails de la requête de soutien au service de soutien.
4. Le service de soutien envoie les détails de la requête de soutien.
5. Le système affiche les détails de la requête de soutien.
6. L'utilisateur consulte la requête de soutien sélectionnée.

UC03 - Répondre à une requête de soutien

Précondition(s) : Une requête de soutien est affichée → UC02

Postcondition(s) : Aucune

Points d'extension : Aucun

Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire répondre à une requête de soutien.
2. L'utilisateur écrit sa réponse et la soumet.
3. Le système envoie l'information au service de soutien.
4. Le service de soutien enregistre la réponse et ajoute l'utilisateur comme observateur s'il ne l'est pas déjà.
5. Le système affiche la réponse dans le fil de discussion.

Scénario(s) alternatif(s) :

4a. Le service de soutien échoue l'enregistrement de la réponse

4a.1. Le système affiche un message d'erreur.

UC04 - Ajouter un membre comme observateur

Précondition(s) : Une requête de soutien, ou un incident est affiché → UC02 ou UC09
Postcondition(s) : Aucune
Points d'extension : Aucun
Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire ajouter un membre comme observateur à une requête de soutien ou un incident.
2. L'utilisateur ajoute un membre de l'équipe à la liste d'observateur de la requête de soutien ou de l'incident en cours.
3. Le système demande l'ajout de l'observateur au service de soutien ou au service de gestion des billets.
4. Le service interrogé enregistre l'ajout de l'observateur.
5. Le système affiche la liste des observateurs mis à jour.

Scénario(s) alternatif(s) :

4a. Le service de soutien ou de gestion des billets échoue l'enregistrement de la réponse
    4a.1. Le système affiche un message d'erreur.

UC05 - Terminer une requête de soutien

Précondition(s) : Une requête de soutien est affichée → UC02

Postcondition(s) : Aucune

Points d'extension : Aucun

Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire terminer une requête de soutien.
2. L'utilisateur indique qu'il veut terminer la requête de soutien affichée.
3. Le système envoie l'information au service de soutien.
4. Le service de soutien ferme la discussion et ajoute l'utilisateur comme observateur s'il ne l'est pas déjà.
5. Le système affiche que la requête de soutien est terminée.

Scénario(s) alternatif(s) :

4a. Le service de soutien échoue la fermeture de la discussion

4a.1. Le système affiche un message d'erreur.

UC06 - Créer un incident

Précondition(s) : Si utilisation du scénario alternatif 1a, une requête de soutien doit être affichée → UC02
Sinon, aucune précondition.

Postcondition(s) : Un nouvel incident est présent dans la base de données

Points d'extension : Aucun

Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire créer un incident
2. L'utilisateur sélectionne l'option d'ajout d'un incident
3. Le système affiche le formulaire de création d'un incident.
4. L'utilisateur remplit le formulaire de création d'un incident en complétant :
    1. Le titre de l'incident
    2. La description de l'incident
    3. Le projet de l'incident
    4. La personne à qui l'incident doit être assigné
    5. Le milestone
    6. Le type (fonctionnalité ou bogue)
5. L'utilisateur soumet le formulaire.
6. Le système envoie l'information au service de gestion des billets
7. Le service de gestion des billets enregistre l'incident
8. Le système persiste l'incident dans la base de données
9. Si l'incident est assigné à l'utilisateur en cours, le système ajoute l'incident à la liste d'incident à faire et est affiché.

Scénario(s) alternatif(s) :

1a. L'utilisateur désire créer un incident à partir d'une requête de soutien.

1a.1. L'utilisateur sélectionne l'option d'ajout d'un incident se trouvant dans la section d'affichage de la requête de soutien.

1a.2. Retour à 3.

1a.3. Le système complète, à partir des informations de la requête de soutien, les champs suivants :

1. Le titre de l'incident
2. Le projet de l'incident

1a.4. Retour à 4.

7a. Le service de gestion des billets échoue l'enregistrement de l'incident

7a.1. Le système affiche un message d'erreur.

UC07 - Modifier un incident

Précondition(s) : Un incident est affiché → CU09

Postcondition(s) : Aucune

Point d'extension : Aucun

Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire modifier un incident.
2. L'utilisateur modifie les champs désirés parmi les suivants :
    1. La personne à qui l'incident doit être assigné
    2. Le milestone
    3. Le type (fonctionnalité ou bogue)
    4. L'état (nouveau, en cours, en revue ou résolu)
3. L'utilisateur soumet ses changements
4. Le système envoie les modifications au service de gestion des billets
5. Le service de gestion des billets enregistre les modifications
6. Le système affiche l'incident modifié

Scénario(s) alternatif(s) :

5a. Le service de gestion des billets échoue l'enregistrement de l'incident

5a.1. Le système affiche un message d'erreur.

UC08 - Consulter une liste d'incidents

Précondition(s) : L'utilisateur est authentifié au service de gestion des billets

Postcondition(s) : Aucune

Points d'extension : Aucun

Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire afficher les incidents à faire, les incidents en attente de revue ou encore celles qui ne sont assignées à personne.
2. L'utilisateur sélectionne la section "incidents".
3. Le système demande au service de gestion des billets la dernière liste d'incidents affichée par l'utilisateur:
    1. les incidents à faire,
    2. les incidents en attente de revue,
    3. ou les incidents non assignés.
4. Le service de gestion des billets envoie cette liste au système.
5. Le système affiche cette liste.
6. L'utilisateur consulte la liste.

Scénario(s) alternatif(s) :

6a. L'utilisateur désire consulter une autre liste d'incident

6a.1. L'utilisateur sélectionne une des listes suivantes:

1. les incidents à faire,
2. les incidents en attente de revue,
3. ou les incidents non assignés.

6a.2. Retour à 5.

UC09 - Consulter un incident

Précondition(s) : Une liste d'incident est affichée → UC08

Postcondition(s) : Aucune

Point d'extension : Aucun

Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire consulter un incident.
2. L'utilisateur sélectionne un incident à afficher.
3. Le système demande les détails de l'incident au service de gestion des billets, de soutien et de contrôle de version.
4. Le service de gestion des billets, de soutien et de contrôle de version envoie les détails correspondant à l'incident.
5. Le système affiche les détails de la requête de soutien.
6. L'utilisateur consulte la requête de soutien sélectionnée.

Scénario(s) alternatif(s) :

6a. Si la requête se trouvait dans la liste des requêtes non lues.

6a.1. La requête est retirée de la liste.

6a.2. Retour à 5.

UC10 - Effectuer une revue de code

Précondition(s) : Un incident est affiché avec l'état "en revue" et ayant une propagation SVN associée → UC09

Postcondition(s) : Aucune

Point d'extension : UC07 - 3.

Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire effectuer une revue de code
2. L'utilisateur choisit l'option pour afficher les différences d'un fichier modifier
3. Le système affiche les différences du fichier modifier
4. Les étapes 2 et 3 sont répétées autant de fois qu'il y a de fichiers modifiés
5. L'utilisateur désire approuver la modification
6. L'utilisateur modifie le statut de l'incident
7. UC07 - 3

Scénario(s) alternatif(s) :

5a. L'utilisateur désire désapprouver la modification

- 5a.1. L'utilisateur modifie le statut de l'incident pour "en cours" et la personne à qui l'incident est assigné pour la personne qui a effectué la modification
- 5a.2. L'utilisateur ajoute un commentaire
- 5a.3. Retour à 7.

UC11 - Écrire dans la salle de réunion

Précondition(s) : L'utilisateur est authentifié au service de clavardage de groupe

Postcondition(s) : Aucune

Points d'extension : Aucun

Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire écrire un message dans la salle de réunion.
2. L'utilisateur sélectionne la section "salle de réunion".
3. Le système affiche la discussion en cours.
4. L'utilisateur écrit et soumet son message.
5. Le système transmet le message au service de clavardage
6. Le système affiche le nouveau message dans la discussion en cours

UC12 - Construire un message de propagation

Précondition(s) : Le champ de message de propagation est affiché par un client SVN

Postcondition(s) : Aucune

Points d'extension : Aucun

Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire propager ses changements dans le code source via son client SVN préféré
2. L'utilisateur sélectionne l'outil de construction de message de propagation.
3. Le système affiche le constructeur de message de propagation en y affichant
    1. la liste des membres de l'équipe
    2. la liste des statuts d'incident
    3. la liste d'incident à faire
4. L'utilisateur ajoute, via un glisser-déposer, les items à associé à la propagation SVN :
    1. un incident pour qu'il soit lié à la propagation
    2. un statut si le statut de l'incident doit changer
    3. un membre de l'équipe si l'incident doit être assigné à quelqu'un d'autre (pour une revue par exemple)
5. Le système affiche la syntaxe textuelle associée aux items ajoutés au message de propagation.

Scénario(s) alternatif(s) :

5a. L'utilisateur tente d'ajouter un statut ou un membre avant d'avoir ajouté un incident

5a.1. Le système affiche un message comme quoi l'utilisateur doit ajouter un incident d'abord

5a.2. Retour à 4.

UC13 - Commenter un incident

Précondition(s) : Un incident est affiché → CU09
Postcondition(s) : Aucune
Points d'extension : Aucun
Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire commenter un incident.
2. L'utilisateur écrit et soumet son commentaire
3. Le système envoie le commentaire au service de gestion des billets
4. Le service de gestion des billets enregistre le commentaire
5. Le système affiche le nouveau commentaire dans l'incident

Scénario(s) alternatif(s) :

5a. Le service de gestion des billets échoue l'enregistrement du commentaire
    5a.1. Le système affiche un message d'erreur.

UC14 - Filtrer l'affichage par projet

Précondition(s) : Aucune
Postcondition(s) : Aucune
Points d'extension : Aucun
Acteur principal : Utilisateur

Scénario principal :

1. L'utilisateur désire afficher les incidents et requêtes de support concernant un(des) projet(s) spécifique(s).
2. L'utilisateur sélectionne l'option de sélection des projets
3. Le système affiche la liste des projets
4. L'utilisateur sélectionne les projets désirés
5. Le système filtre les affichages en cours selon la nouvelle sélection de projets.
6. Le système sauvegarde les préférences de l'utilisateur.

Annexe C
Document
de conception

Incidents
DES01
Design Document

Historique des révisions

Date	Description	Révision	Auteur
29-09-2011	Création des sections	0.1	Anthony Plourde
30-09-2011	Vue architecturale haut niveau	0.2	Anthony Plourde
30-09-2011	Diagramme de classe du modèle du domaine	0.3	Anthony Plourde
17-10-2011	Diagramme de classe et de séquence du module de soutien aux utilisateurs	0.4	Anthony Plourde
25-10-2011	Section 1 et 2	0.5	Anthony Plourde
27-10-2011	Section 3	0.6	Anthony Plourde
28-10-2011	Section 4.3 - Modules du soutien aux utilisateurs	0.7	Anthony Plourde
30-10-2011	Révision fin itération 2	1.0	Anthony Plourde
20-11-2011	Section 4.4 - Modules des incidents	1.2	Anthony Plourde
28-11-2011	Section 4.4 et 4.5 - Modules de clavardage et des versions	1.4	Anthony Plourde
16-12-2011	Révision finale - Diagramme de classes allégés pour les modules de support et des incidents	1.5	Anthony Plourde

1. Introduction

1.1. Objectif

Le but de ce document est de définir l'architecture et la conception du système *Incidents*, un outil visant à simplifier le suivi et l'application des processus de développement chez Edovia, une TPE oeuvrant dans la création d'applications iOS et Mac. Ce document présente les principaux éléments à considérer pour la conception du logiciel. Des solutions concrètes sous forme de diagrammes UML et de vues architecturales seront présentées dans ce document dans l'optique de résoudre les défis et préoccupations de la problématique.

1.2. Portée

L'entreprise est présentement inscrite à de nombreux services en ligne ayant pour but d'améliorer les processus de développement dans le cycle de vie logiciel. Parmi ces services, on retrouve un outil de gestion de soutien à la clientèle, un gestionnaire de billet, un gestionnaire de dépôt SVN et finalement un service de messagerie instantanée de groupe pour faciliter le travail à distance. Chacun de ces services fonctionne relativement bien et répond correctement aux besoins qu'il cible. Cependant, l'utilisation quotidienne de l'ensemble de ces services s'apparente davantage à une tâche superflue plutôt qu'à une tâche qui facilite le processus de développement pour une si petite équipe. Chacun de ces services est sous forme d'application web, ce qui ne favorise pas la consultation régulière des développeurs.

Il existe donc quatre systèmes existants :

- Tender App (gestionnaire du support à la clientèle)
- Lighthouse App (gestionnaire de billets)
- Beanstalk App (gestionnaire des dépôts SVN)
- Talker App (outil de clavardage de groupe)

Ces systèmes sont de tierce partie et resteront inchangés. L'application Incidents communiquera à ces systèmes via des interfaces génériques qui devront permettre une entière transparence des services externes utilisés. Le but d'une telle méthodologie est d'éventuellement pouvoir remplacer les services payants par des services maison.

Dans l'illustration de la page suivante, on peut voir que la portée du projet se limite à l'application native Mac et à ces interfaces communiquant avec les services externes.

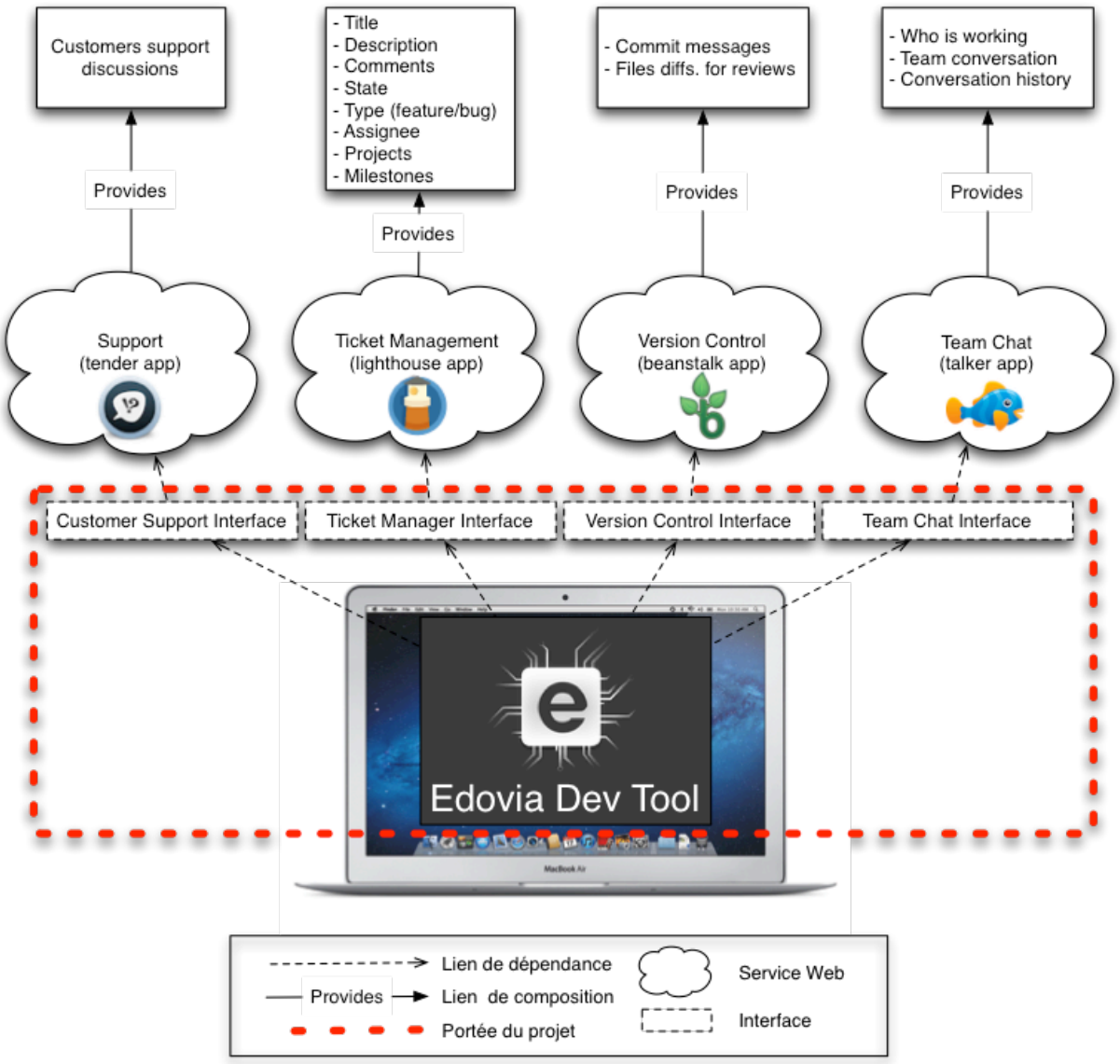


Figure 1. Portée du projet

1.4. Documents de références

- **Incidents - SRS01 - Software Requirements Specification**
  - *Anthony Plourde - Edovia inc.*
- **Deployment Package - Software Architectural and Detailed Design**
  - *École de technologie supérieure - ISO/IEC 29110*
  - Ce gabarit provient de ce document

2. Parties prenantes et préoccupations

2.1. Parties prenantes et leurs préoccupations

Les parties prenantes dans ce projet se limitent aux employés d'Edovia, incluant les stagiaires, les employés à temps partiel et le propriétaire fondateur.

Dans un premier lieu, l'application développée se veut un outil dont l'équipe de développeur tirera avantage lors de son utilisation quotidienne. Par contre, l'application doit être développée en permettant la possibilité de remplacer, dans un avenir certain, des services externes utilisés (service de support, de gestion des billets, de contrôle de version et de clavardage), sans avoir à modifier la totalité de l'application. Ce scénario est viable, puisqu'il permettrait à l'entreprise de sauver d'importantes sommes d'argent reliées aux abonnements de ces services. Cependant, l'entreprise est réticente à aller de l'avant dans cette direction puisque ceci demanderait beaucoup de temps de développement.

Si le projet de remplacement des services externes vient qu'à prendre vie, l'entreprise souhaiterait aller de l'avant avec le projet et d'en faire profiter d'autres entreprises ayant un contexte similaire, en leur **vendant la solution**. Bien entendu, ce scénario est viable, mais l'entreprise n'est pas persuadée qu'elle veut aller dans cette direction avec le projet.

Quoi qu'il en soit, ces éventualités indiquent deux choses. D'abord, l'application doit être **découplée des services externes** auxquels elle est rattachée et aussi que l'application soit **extensible**.

Également, étant donné que l'application offrira d'abord un nombre de fonctionnalités limité comparativement à tout ce qui est possible dans chacun des services externes auxquels elle est rattachée, il est fort possible qu'à force d'utilisation, l'entreprise se rende compte que certaines fonctionnalités devraient être ajoutées. **L'ajout de ces fonctionnalités** ne devrait pas demander de restructuration majeure dans l'application.

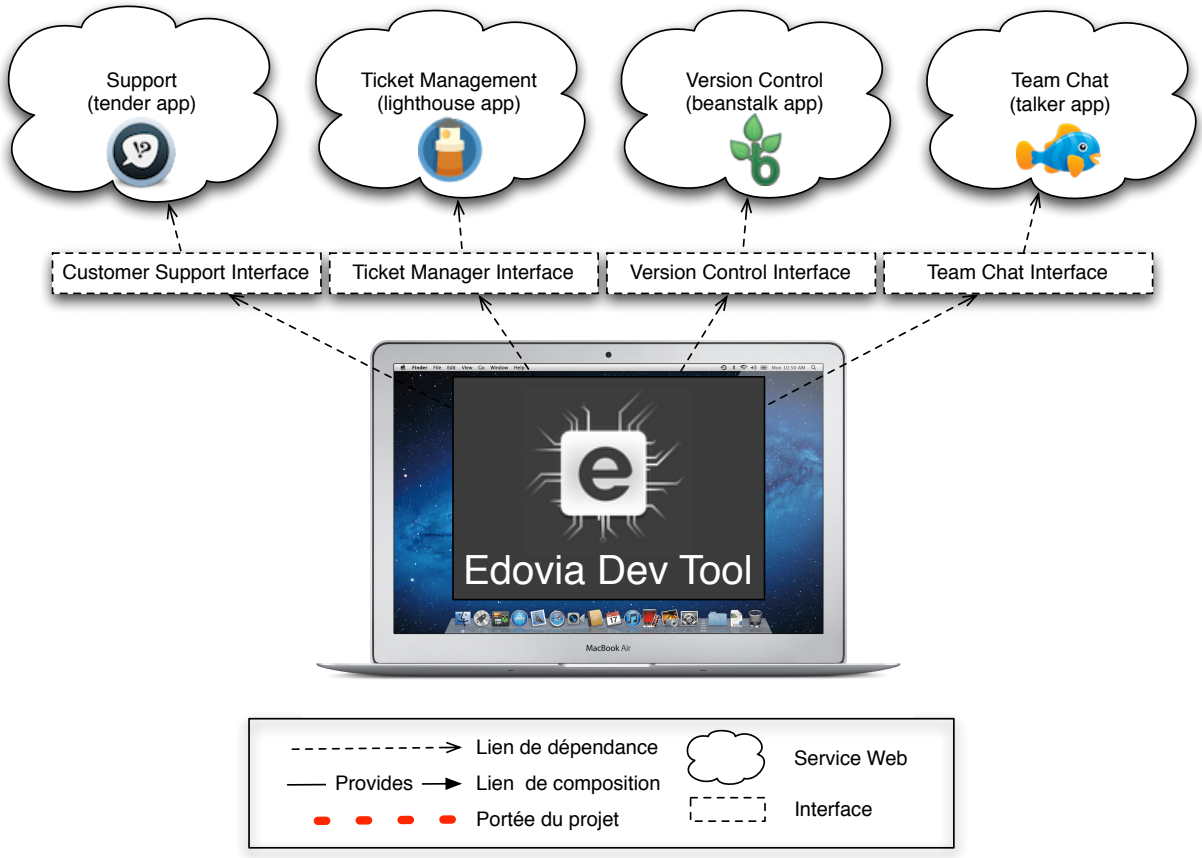
Enfin, une importante préoccupation concerne le fait que l'application soit une application native qui utilise un service web comme source de données. Si aucun mécanisme spécial n'est mis en place pour s'assurer que les données affichées dans l'application soient toujours les données présentes sur le serveur, ceci pourrait amener des scénarios indésirables. Par exemple, si un utilisateur consulte les détails d'une discussion et que pendant ce temps un autre utilisateur modifie les détails de cette même discussion, le premier utilisateur ne saura pas que l'information a changé.

2.2. Relation entre les vues et les préoccupations

Préoccupations	Vues
CNCRN01 - Découplage des services externes	Vue de module en couche démontrant l'indépendance de chacune des couches.
CNCRN02 - Ajout de fonctionnalité qui existe déjà dans les services externes	Vue de module démontrant les composants à modifier pour un tel ajout de fonctionnalité.
CNCRN03 - Ajout de fonctionnalité qui ne pourrait être supporté par les services externes	Vue de module démontrant les composants à modifier pour un tel ajout de fonctionnalité.
CNCRN04 - Extensibilité pour commercialiser le produit	Vue de déploiement démontrant les possibilités d'infrastructure si le produit devenait qu'à être commercialisé.
CNCRN05 - Affichage des données toujours à jour	Vue de module démontrant le patron MVC et son application du patron observateur.

3. Description de l'architecture logicielle

3.1. Vue d'ensemble de l'architecture logicielle



3.2. Vue architecturale de haut niveau

3.2.1. Vue d'ensemble

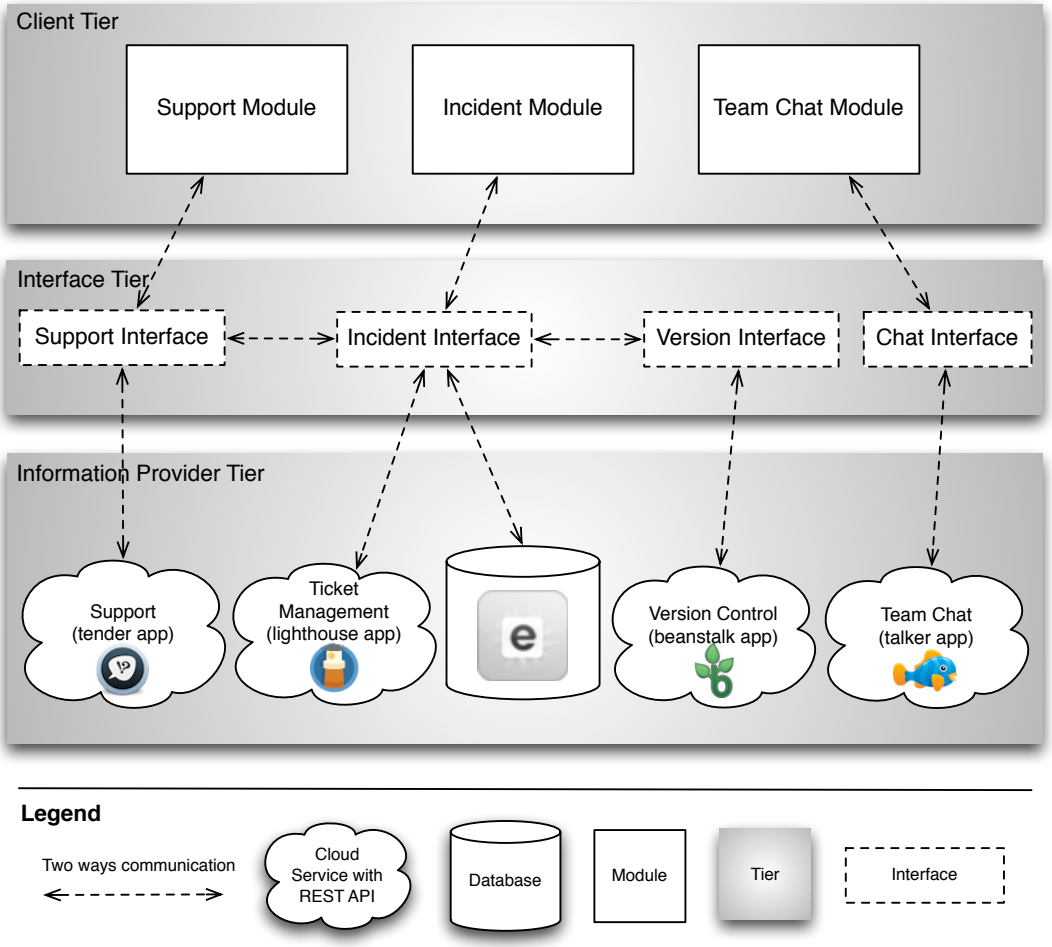


Figure 3.2.1 - Architecture globale

3.2.2. Contraintes de conception s'appliquant à cette vue

- CON03 - Utilisation des API Rest publics des services externes

3.2.3. Préoccupations et exigences résolues par la vue

- CNCRN01 - Découplage des services externes

3.2.4. Description des éléments et des interfaces

Élément et interface	Description
Client Tier	Couche cliente. C'est principalement là que sera développée l'application native. Cette couche comprend tout ce qui touche à la présentation des données, fenêtres, vues, etc. Également, c'est à partir de cette couche qu'on appelle la couche interface pour qu'elle nous retourne les données à présenter.

Élément et interface	Description
Interface Tier	Couche interface qui permet à la couche cliente d'accéder aux données sans avoir connaissance de la source des données réelles. Si un service externe venait qu'à être remplacé, c'est cette couche qui devra être adaptée.
Information Provider Tier	Couche d'accès aux données. C'est cette couche qui fournit les données réelles à la couche interface. La couche interface utilise cette couche pour aller chercher ces données provenant en réalité soit d'un service web ou encore d'une base de données.
Support Module	Ce module s'occupe de gérer tout ce qui concerne l'affichage et l'interaction avec les données retournées par l'interface de support.
Incident Module	Ce module s'occupe de gérer tout ce qui concerne l'affichage et l'interaction avec les données retournées par l'interface des incidents.
Team Clavardage Module	Ce module s'occupe de gérer tout ce qui concerne l'affichage et l'interaction avec les données retournées par l'interface de clavardage de groupe.
Support Interface	Permet au module de support d'accéder aux données en toute transparence. L'interface interroge le service de support Tender via son API REST.
Incident Interface	Permet au module d'incident d'accéder aux données en toute transparence. Pour obtenir les données d'un incident, l'interface interroge le service de gestion des billets Lighthouse via son API REST, mais également la base de données Edovia pour connaître les discussions de support et les propagations SVN en lien avec le billet. Rappelons qu'un incident est la combinaison d'une discussion provenant sur service de support, d'un billet provenant du service de gestion des billets, et des propagations effectuées via le service de contrôle de version. Également, pour connaître les détails des discussions de support et propagation SVN en lien avec l'incident, l'interface doit interroger les interfaces de support et de version.
Version Interface	Permet à l'interface des incidents d'accéder aux données des propagations SVN en toute transparence. L'interface de version interroge le service de contrôle de version Beanstalk via son API REST.
Chat Interface	Permet au module de clavardage de groupe d'obtenir ses données en toute transparence.
Support Cloud Based Service	Tender. Service payant en ligne servant à effectuer du soutien aux utilisateurs. Le service a un API REST qui est utilisé pour gérer les données du service.
Ticket Management Cloud Based Service	Lighthouse. Service payant en ligne servant à effectuer de la gestion de billets pour le développement logiciel. Le service a un API REST qui est utilisé pour gérer les données du service.

Élément et interface	Description
Version Control Cloud Based Service	Beanstalk. Service payant en ligne servant à effectuer du contrôle de version SVN. Le service a un API REST qui est utilisé pour gérer les données du service.
Team Chat Cloud Service	Talker. Service payant en ligne servant à effectuer du clavardage de groupe. Le service a un API REST qui est utilisé pour gérer les données du service.
Edovia Database	Base de données mise en place pour le projet. Cette base de données contient les informations des incidents (lien entre discussion Tender, billet Lighthouse et propagation Beanstalk). Elle contient également l'information sur les projets (lien entre catégories Tender, projet Lighthouse et dépôt Beanstalk) et aussi l'information des utilisateurs membres de l'équipe (lien entre utilisateur Tender, utilisateur Lighthouse, utilisateur Beanstalk et utilisateur Talker).

3.2.5. Raisonnement

Le principal but de cette architecture en couche est de découpler l'application cliente des services externes qui fournissent les données. Le tout est découplé grâce aux interfaces qui les relient. Pour chacun des modules de l'application cliente, tout ce dont il a à connaître pour l'accès aux données, c'est son interface associée. Plus concrètement, le module de support doit seulement connaître l'interface de support, le module d'incident l'interface des incidents et le module de clavardage l'interface de clavardage.

Seule l'interface des incidents a des dépendances vers les autres interfaces puisqu'un incident est la combinaison d'une discussion tirée du service de support, d'un bogue à régler ou d'une fonctionnalité à développer tirée du service de gestion des billets et enfin, de propagations SVN tirées du service de contrôle de version. De plus, c'est pourquoi l'interface des incidents fait des accès à une base de données en plus des accès à un service externe. La base de données est là principalement pour entreposer les relations entre les composantes d'un incident.

3.2.6. Autres vues pertinentes

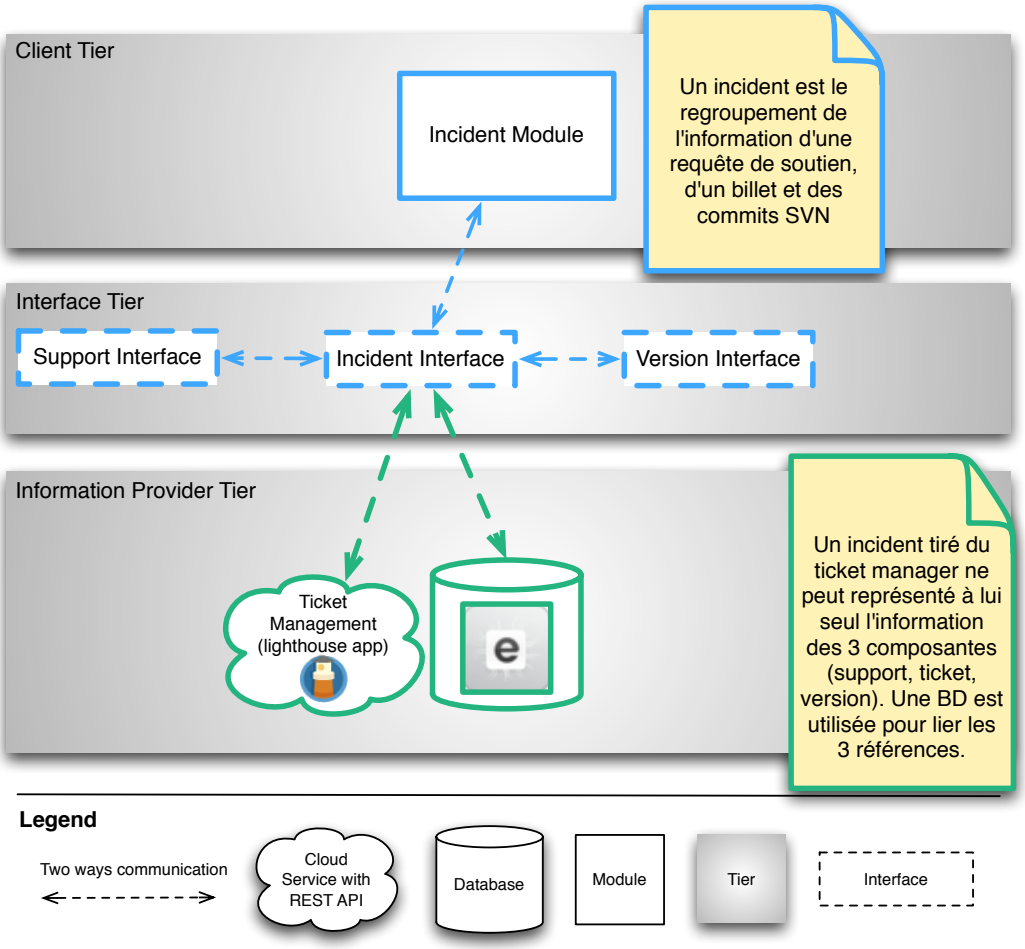


Figure 3.2.2 - Détails des relations des incidents

Projects
id : INT name : VARCHAR lighthouseId : INT tenderId : INT beanstalkId : INT repository : VARCHAR

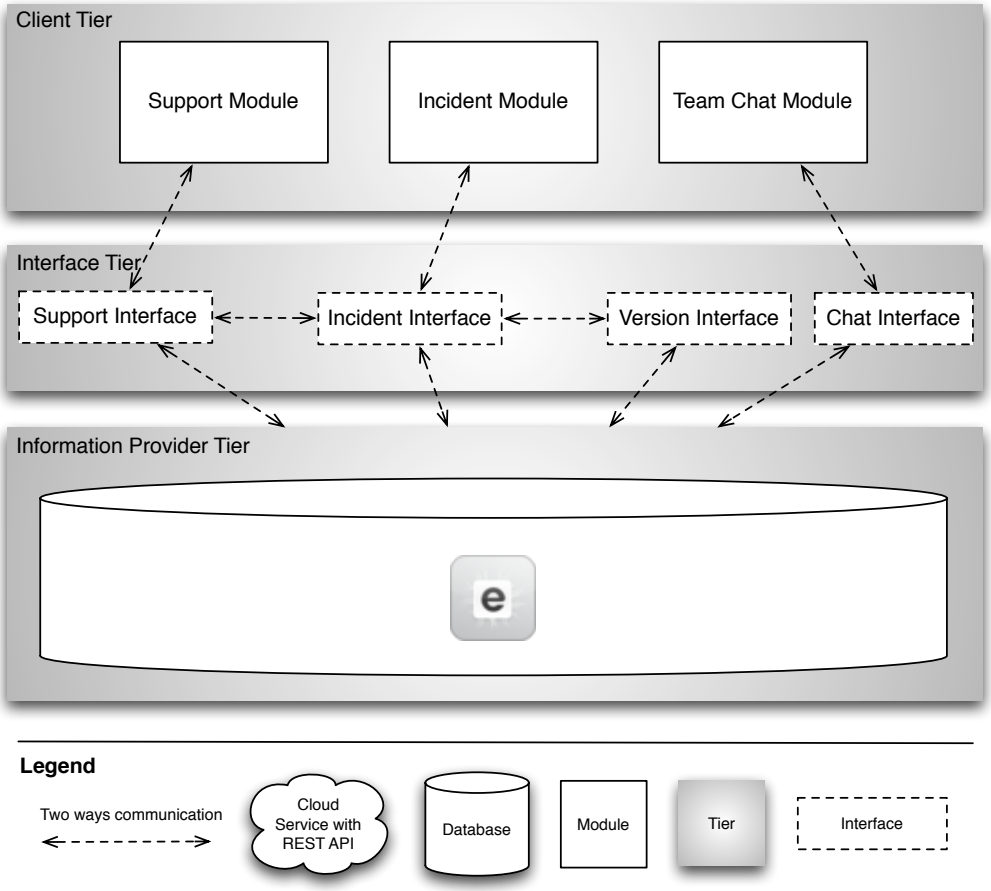
Users
id : INT name : VARCHAR lighthouseId : INT tenderId : INT beanstalkId : INT tenderQueueId : INT email : VARCHAR

Incidents
id : INT lighthouseTicketId : INT tenderDiscussionId : INT beanstalkChangesetIds : SET lighthouseProjectId : INT

Figure 3.2.3 - Table de la base de données

3.3. Vue architecturale de haut niveau - Évolution du projet

3.3.1. Vue d'ensemble



3.3.2. Contraintes de conception

Aucune

3.3.3. Préoccupations et exigences résolues par la vue

- CNCRN01 - Découplage des services externes
- CNCRN04 - Extensibilité pour commercialiser le produit

3.3.4. Description des éléments et des interfaces

Idem à la vue précédente (3.2.4)

3.3.5. Raisonnement

Afin de sauver un maximum d'argent, l'entreprise pourrait retirer, un par un, chacun des services externes pour qu'à la fin, elle n'ait qu'une seule base de données, et potentiellement un service web devant celle-ci, contenant toute l'information qui était autrefois desservie par ces services.

Également, une fois cette étape passée, l'entreprise pourrait décider de commercialiser sa solution étant donné qu'elle ne dépend plus de services externes.

3.4. Vue de module de l'architecture MVC

3.4.1. Vue d'ensemble

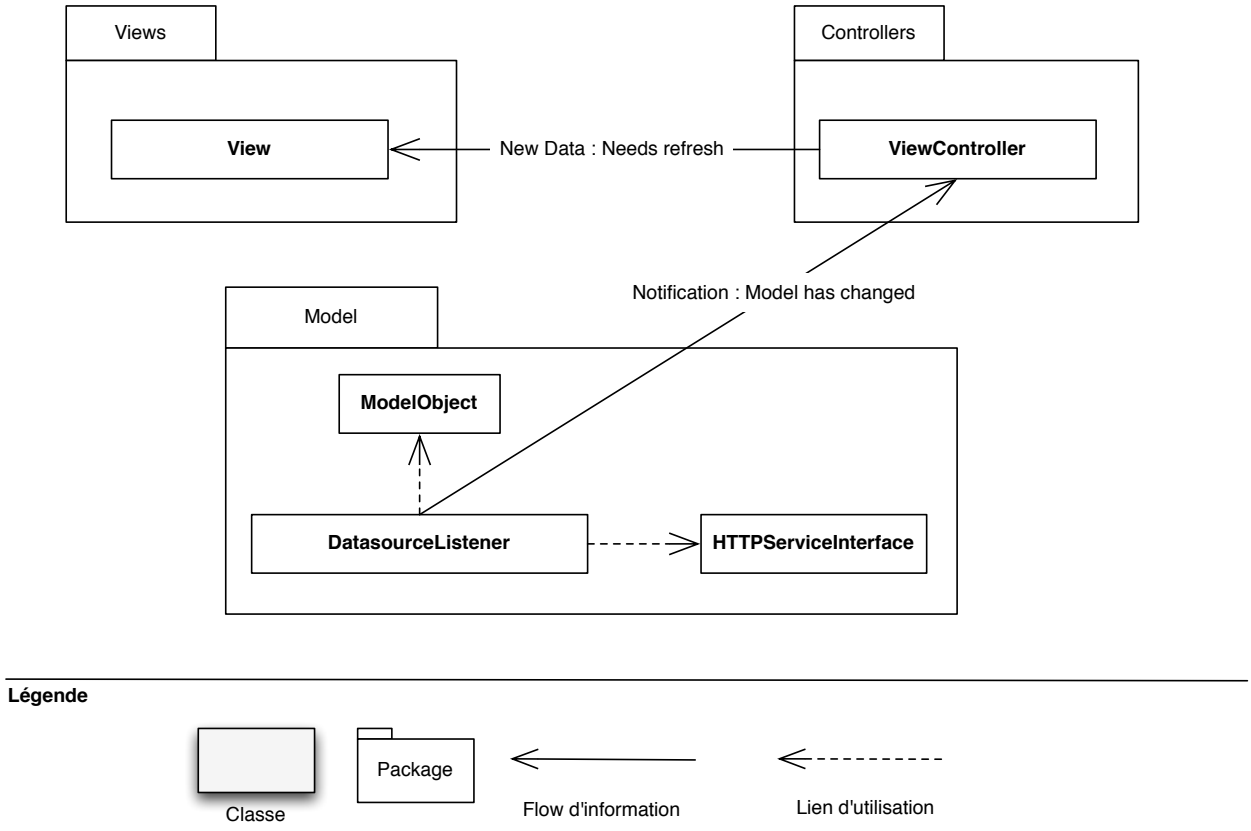


Figure 3.4.1 - Application du patron MVC dans le système

3.4.2. Contraintes de conception

- CON02 - Application Cocoa
- CON03 - Utilisation des API Rest publics des services externes

3.4.3. Préoccupations et exigences résolues par la vue

- CNCRN05 - Affichage des données toujours à jour

3.4.4. Description des éléments et des interfaces

Éléments	Description
Package "Views"	Regroupement de toutes les vues du système.
Package "Controllers"	Regroupement de tous les contrôleurs du système.
Package "Model"	Regroupement des objets du modèle du domaine et de ce qui permet d'interroger les sources de données auxquels ces objets sont rattachés.
View	Une vue du système qui affiche de l'information du modèle du domaine. La vue des incidents, des discussions, etc.

Éléments	Description
ViewController	Un contrôleur de vue du système qui manipule de l'information du modèle du domaine. Le contrôleur de la vue des incidents, des discussions, etc. Ce contrôleur reçoit les événements déclencher par l'utilisateur et prépare l'information pour son affichage dans la vue. Il reçoit les notifications du *DatasourceListener* lorsque les données liées à un des objets du modèle qu'il manipule sont victime d'un changement. Lorsqu'il reçoit une telle notification, il met à jour son information et force la vue à se rafraîchir pour refléter l'information mise à jour.
ModelObject	Un objet du modèle du domaine. Un incident, une discussion, etc. L'information de cet objet sera affichée dans la vue.
DatasourceListener	Objet qui utilise le *HTTPServiceInterface* pour obtenir les données de la source de données reliée au *ModelObject.* Il interroge continuellement la source de données sur un intervalle de temps restreint afin de vérifier si les données reliées au *ModelObject* auquel il est rattaché ont changé. Si oui, il lance une notification. Ceux qui sont abonnés à la notification, en l'occurrence le ViewController, la recevrons. Un DatasourceListener spécifique pour chaque type de *ModelObject* affiché dans les vues devra être implémenté.
HTTPServiceInterface	Interface utilisée pour interroger l'API Rest d'un service externe comme Tender, Lighthouse, beanstalk.

3.4.5. Raisonnement

Bien qu'un patron MVC est élémentaire dans les logiciels, dans le cas où l'accès aux données est distribué, il faut mettre en place un mécanisme supplémentaire puisque la source de données partagée ne peut avertir d'elle-même les logiciels clients qui consultent ses données pour avertir que les données ont changées. Le DatasourceListener a donc été mis en place.

Un DatasourceListener différent sera utilisé pour chaque module, au même titre qu'il existe un HTTPServiceInterface pour chaque module. Plutôt qu'un contrôleur de vue utilise directement le HTTPServiceInterface auxquelles il est associé (ex: SupportViewController est associé au SupportInterface), le contrôleur de vue s'abonnera aux notifications du DatasourceListener pour tous les items qu'il désire manipuler. Le DatasourceListener fera des requêtes en boucle pour obtenir toujours les dernières données des items demandés par le contrôleur de vue. Lorsque le DatasourceListener détectera un changement sur un de ces items, il lancera une notification. Le contrôleur de vue recevra cette notification accompagnée des nouvelles données et mettra à jour la vue.

3.4.6. Séquence d'utilisation du patron observateur

Voici la séquence d'utilisation du patron observateur à travers le patron MVC dans le système. Une séquence semblable se retrouvera dans les diagrammes de séquence des différents modules dans la section conception détaillée de ce document. En effet, c'est le même principe qui est appliqué pour tous les affichages des données, que ce soit des listes de discussion ou d'incident ou encore le détail d'une discussion ou d'un incident. Lorsqu'il s'agit d'une liste, plutôt que de retourner la liste complète lorsqu'on détectera que celle-ci a changé, seulement les items ajoutés ou supprimés seront retournés au contrôleur de vue (voir l'exemple dans le module de soutien).

Bien qu'il n'est pas visible sur ce diagramme, noter qu'un système de cache est également en place dans le *DatasourceListener*. Donc, lorsque l'utilisateur commence à écouter un objet, si l'objet a déjà été écouté auparavant, le *DatasourceListener* lui retournera tout de suite et reprendra son écoute sur celui-ci. À chaque fois que le *DatasourceListener* retourne l'objet pour dire qu'il a changé, il s'assure en même de temps d'ajouter ce nouvel objet dans sa cache. La cache aura une taille limite définie, pour éviter qu'elle devienne trop lourde pour le système.

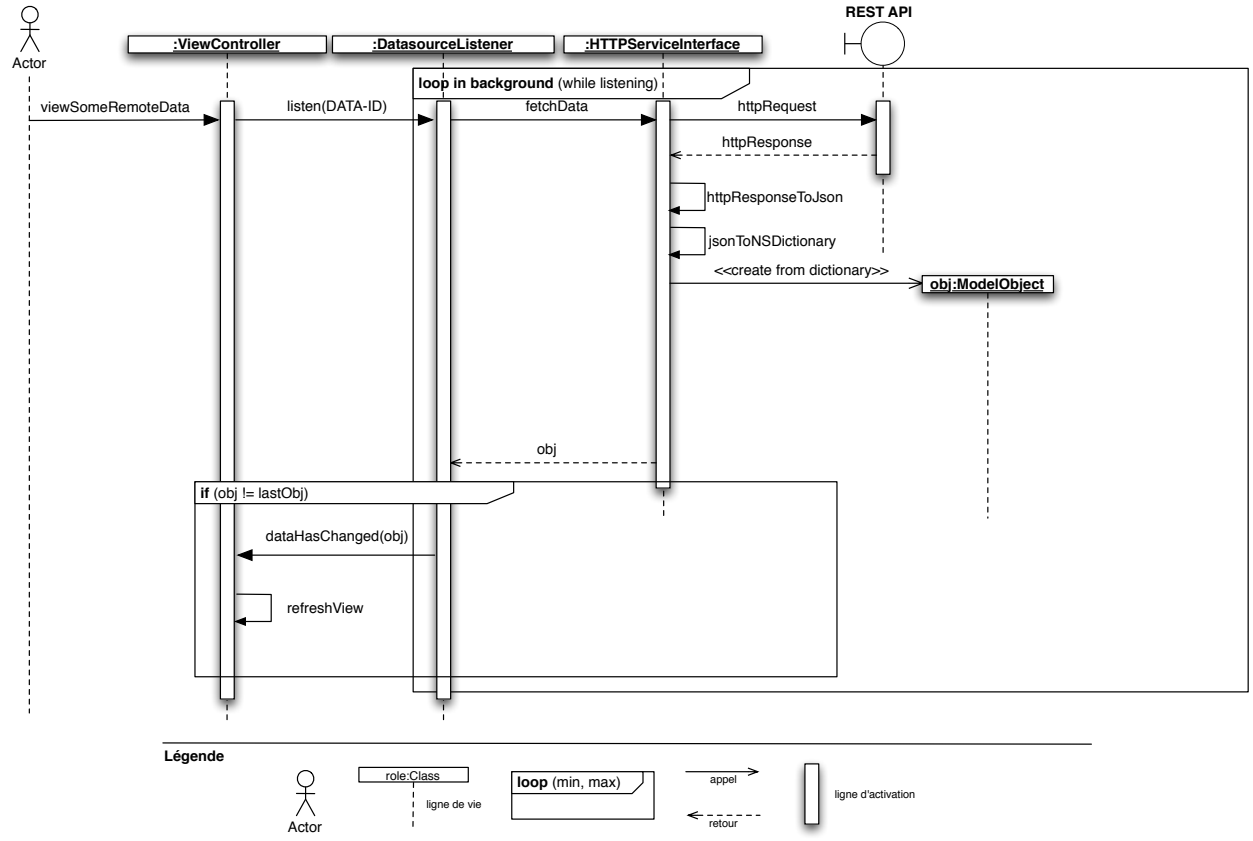


Figure 3.4.6 - Séquence de l'application du patron observateur à travers le patron MVC du système

4. Description de la conception détaillée

4.1. Vue d'ensemble de la conception détaillée

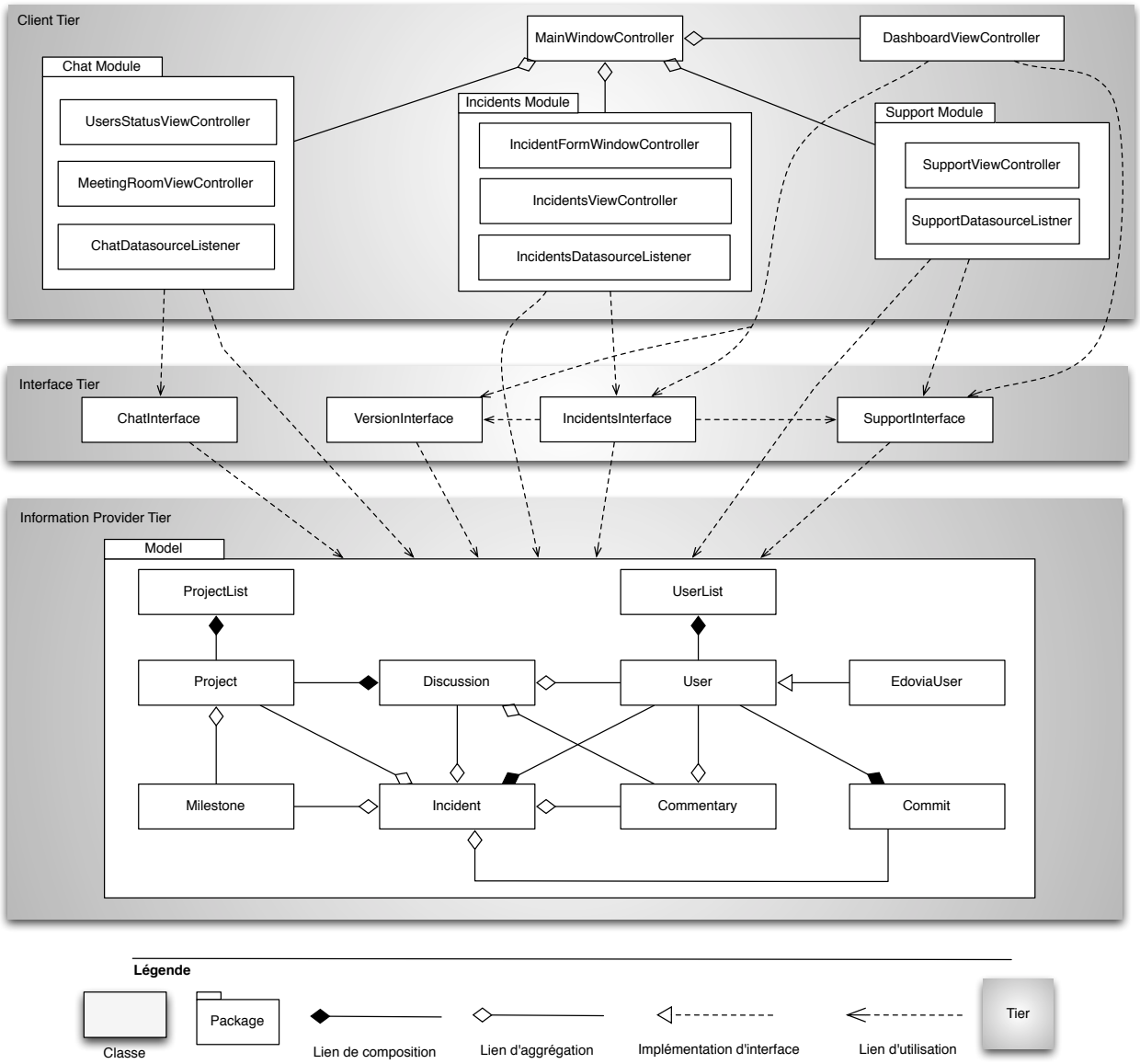


Figure 4.1 - Vue d'ensemble de la conception du logiciel sur les couches de l'architecture

4.2. Conception détaillée du modèle du domaine

4.2.1. Vue structurale

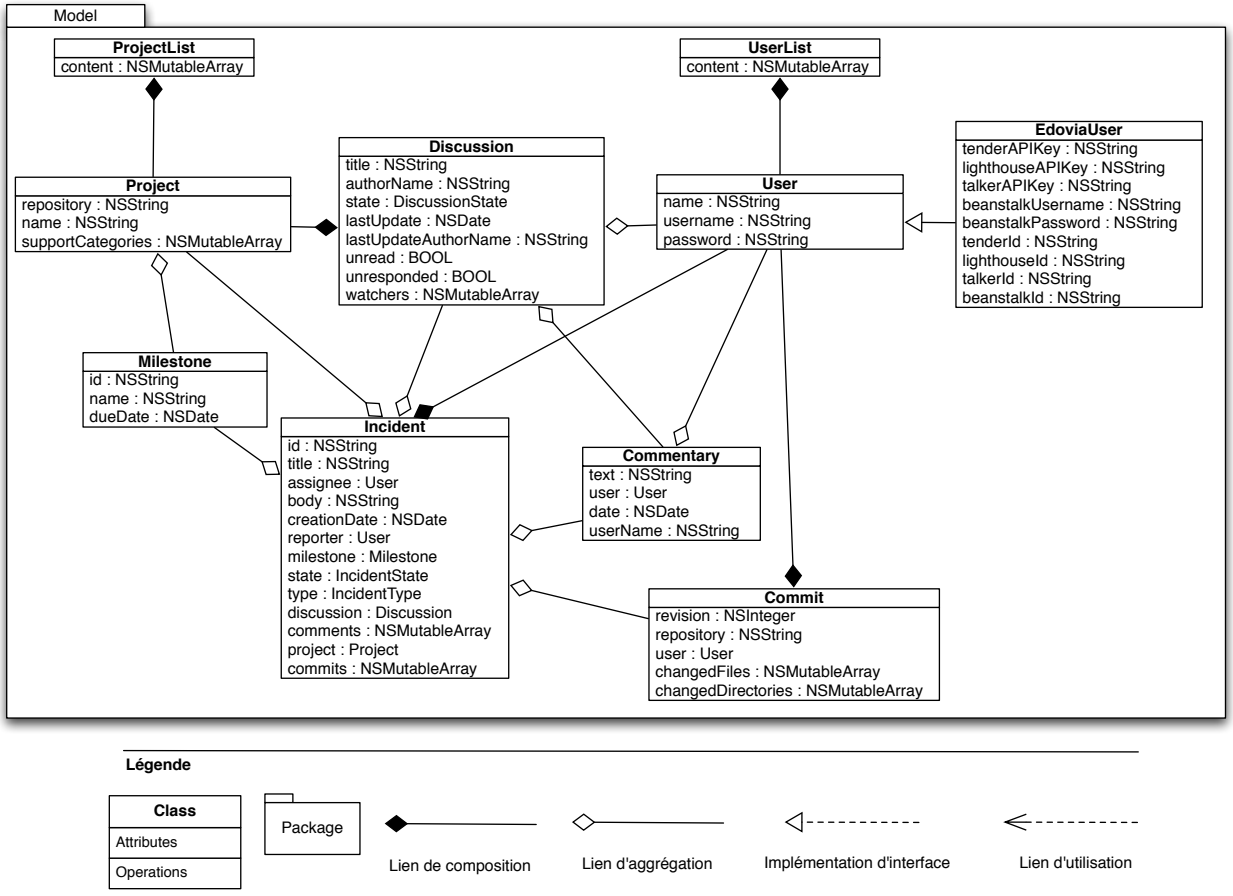


Figure 4.2.1 - Diagramme de classe du modèle du domaine

4.2.2. Vue de comportement

Aucun comportement pertinent étant donné que c'est le modèle du domaine et que sa seule responsabilité est de représenter l'état des objets.

4.2.3. Autre vue pertinente

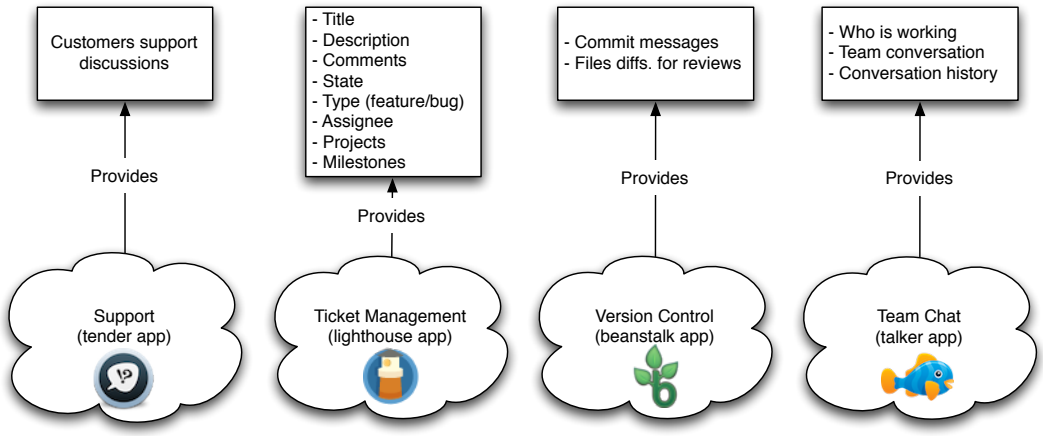


Figure 4.2.2 - Quel service fournit quelles informations

4.2.4. Raisonnement

Concernant le diagramme de classe du modèle du domaine, on peut voir que plusieurs éléments composent l'élément principal de ce projet : l'incident. Comme souvent répété, un incident est principalement composé d'un bogue à régler ou d'une fonctionnalité à développer provenant du service de gestion des billets Lighthouse, d'une discussion provenant du service de support Tender, et de propagation SVN provenant du service de contrôle de version Beanstalk. Comme le billet Lighthouse est le coeur de l'incident, il n'y a pas de classe Billet pour éviter la confusion puisqu'un incident **est** en d'autres mots une sorte de billet.

Voici un tableau explicatif de chacune des classes et ses relations :

Classe	Description	Relation
ProjectList	La liste complète des projets de développement effectués par l'entreprise. Singleton.	Contient une liste d'objet *Project*.
Project	Un projet de développement effectué par l'entreprise	Formé d'un nom, du nom du dépôt svn associé pour pouvoir associer un *Commit* à un *Project* et du nom de la catégorie associé dans le service de support pour pouvoir associer une *Discussion* à un *Project*. Également, un *Project* est composé d'une liste de *Milestone*.
Milestone	Jalon d'un projet	Un *Project* peut avoir plusieurs *Milestone*. Un *Incident* fait référence à un *Milestone*.
UserList	La liste complète des développeurs au sein de l'entreprise. Singleton	Contient une liste d'objet *User*.
User	Développeur au sein de l'entreprise	Une *Discussion* peut avoir plusieurs "watchers" qui sont en fait des *User*. Un *Incident* est créé par un *User* et est assigné à un *User*. Un *Commentary* ainsi qu'un *Commit* sont fait par un *User*.

Classe	Description	Relation
EdoviaUser	Développeur au sein de l'entreprise incluant ses informations d'usager dans les services externes Tender, Lighthouse, Beanstalk et Talker. (Si jamais on venait qu'à remplacer tous les services externes, EdoviaUser deviendrait inutile)	Hérite de la classe *User*.
Discussion	Intervention d'un client concernant un produit développé par l'entreprise. Peut-être un bogue trouvé ou encore une suggestion pour une fonctionnalité.	Peut composer ou non un *Incident*. Est composé d'un *Project* et peut être observé d'un ou plusieurs *User* en tant que "watchers".
Commentary	Commentaire entre développeurs sur un incident ou encore un message faisant partie du fil de message d'une discussion. La classe se nomme Commentary parce que "Comment" entrait en conflit avec un mot clé du langage de programmation.	Compose le fil de message d'une *Discussion*. Dans le cas où le *Commentary* est fait par un développeur, un *User* lui est associé. Un ou plusieurs *Commentary* peuvent être ajoutés à un *Incident*.
Incident	Tâche à effectuer par un développeur, soit un bogue à régler ou une fonctionnalité à développer.	Composé d'un *User* qui l'a créé, de possiblement une *Discussion*, un *Project*, un *Milestone*, des *Commentary*, des *Commit* et un *User* à qui *l'Incident* est assigné.
Commit	Propagation SVN effectuée avec un message et une liste de fichiers et de dossiers modifiés, ajoutés ou supprimés.	Compose un *Incident* et est effectué par un *User*.

4.3. Conception du module de support

4.3.1. Vue structurale

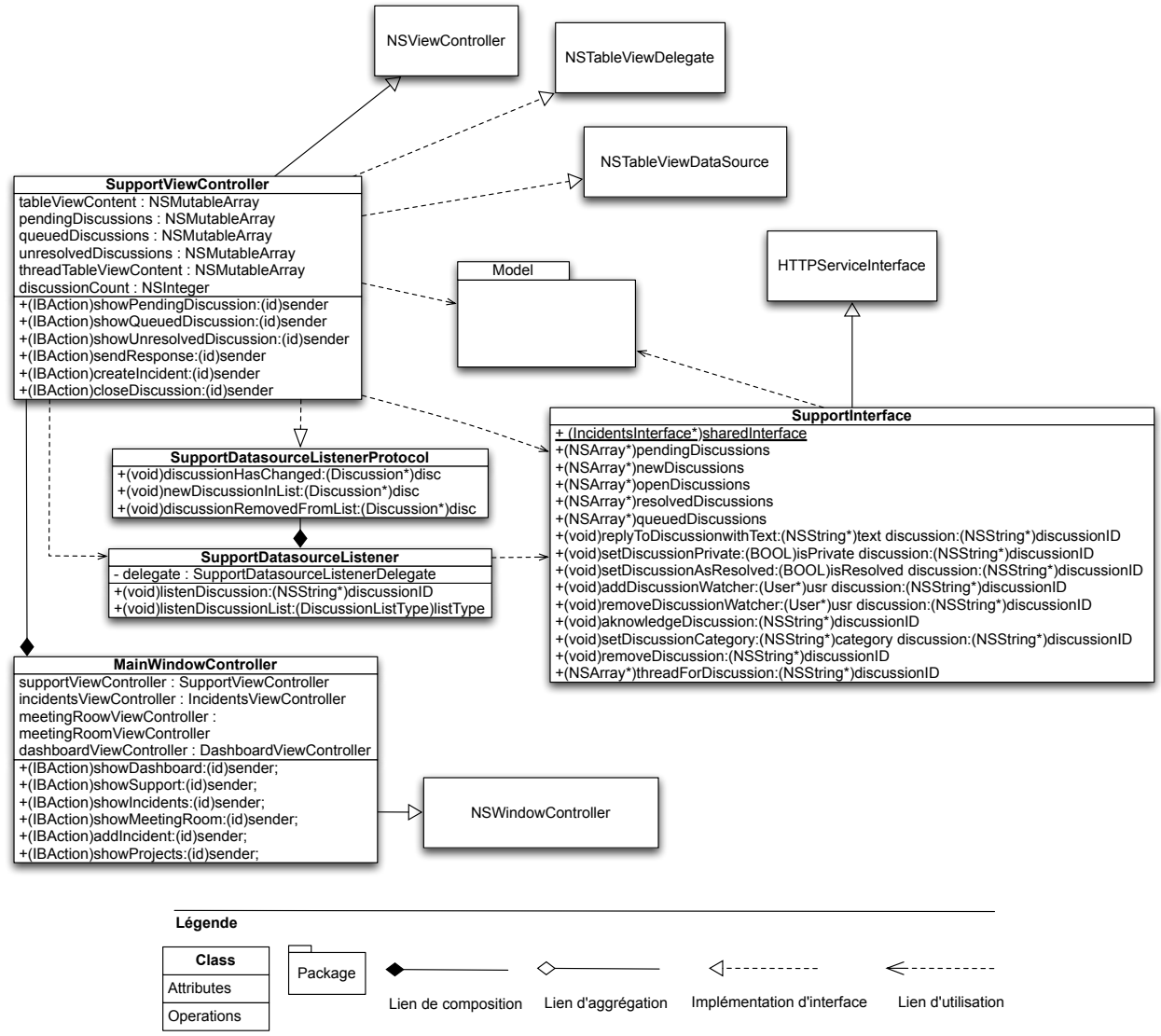


Figure 4.3.1 - Diagramme de classe du module de support

4.3.2. Vue de comportement

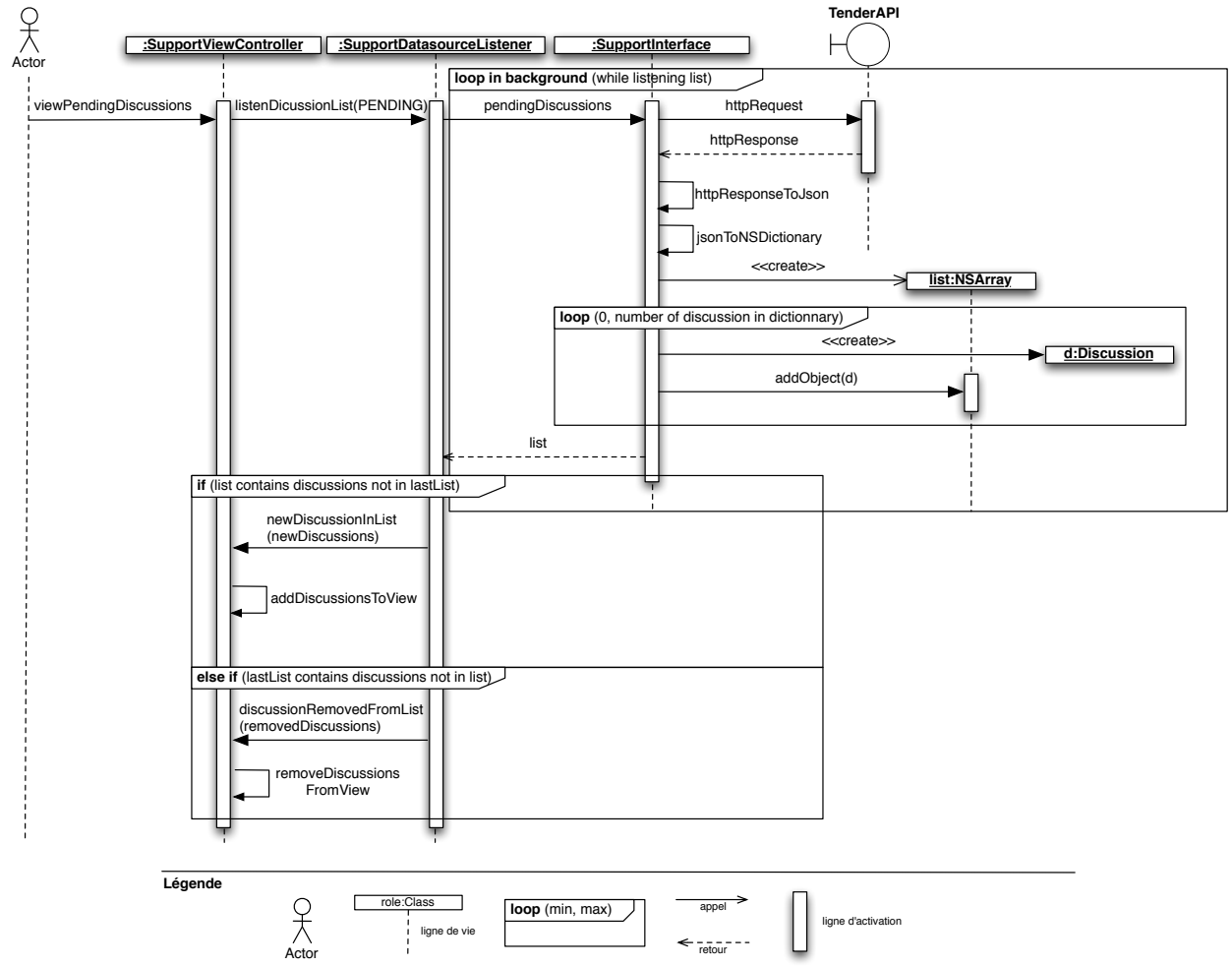


Figure 4.3.2 - Diagramme de séquence pour obtenir la liste des discussions en attente, dans le module de support

4.3.3. Autre vue pertinente

4.3.4. Raisonnement

Structure

À noter que le diagramme de classe ne présente pas les détails concernant les éléments de l'interface utilisateur. Par contre, il présente les méthodes des événements de l'interface utilisateur. Ceux-ci portent le type de retour *IBAction*.

Le coeur du module de support est le *SupportViewController*, c'est de là où sont faites les requêtes à l'interface de support pour obtenir et modifier les données du service de support. Cette classe implémente les interfaces NSTableViewDataSource et NSTableViewDelegate pour permettre l'affichage des données dans une liste et hérite de la classe NSViewController, qui est la classe générale du framework Cocoa pour contrôler une vue. Dans cette classe, on retrouve principalement des attributs de type NSMutableArray qui sont des listes contenant les trois différents états de discussion disponible via l'interface, soit en attente ("pending"), assignée ("queued") et non résolue ("unresolved").

La classe *SupportInterface* offre les méthodes pour obtenir les informations des discussions du service de support via des méthodes comme *pendingDiscussions, openDiscussions, queueDiscussions*, qui retournent tous des listes de

discussions. Ces méthodes permettant d'obtenir les listes sont appelées du SupportDatasourceListener qui notifie le SupportViewController seulement lorsque la liste obtenue du serveur a changé. Les autres méthodes qui permettent de faire des actions sur les discussions sont appelées directement du contrôleur de vue. Dans cette classe, les requêtes HTTP sont faites de façon synchrone bloquante. Donc, il est préférable que celles-ci soient appelées dans un fil d'exécution séparé pour ne pas bloquer l'interface utilisateur qui est sur le fil d'exécution principale le temps de la requête.

La vue pour le module de support se retrouve au sein de la fenêtre principal, c'est pourquoi un lien de composition relie le *MainWindowController* au *SupportViewController.*

Comportement

Pour chaque interaction menant à la consultation de données, la séquence présentée dans le diagramme ci-haut est la même. L'utilisateur demande l'affichage de certaines données, le contrôleur de la vue demande à son *DatasourceListener* d'écouter un item distant en faisant des requêtes en boucle dans un fil d'exécution en arrière-plan. Le *DatasourceListener* utilise l'interface pour interroger le serveur de support. L'interface construit sa requête HTTP, l'envoie au service web, celui-ci retourne une réponse au format XML ou JSON (JSON dans le cas de Tender), la chaîne de caractère JSON est transformée en *NSDictionnary* via la libraire SBJSON. Une fois le dictionnaire en main, l'interface le parcourt et crée ses objets. Dans ce cas-ci, l'interface crée plusieurs instances de *Discussion* et les entrepose dans un tableau. Le dernier tableau obtenu est gardé à chaque itération pour pouvoir comparer prochain tableau obtenu lors de la prochaine itération. On pourra donc comparer s'il y a de nouveaux éléments, ou des éléments qui sont disparus. Dans les deux cas, le contrôleur de vue sera notifié vie les méthodes du protocole *SupportDatasourceListener* qu'il implémente. Une fois notifié, le contrôleur de vue mettra son information à jour et forcera un rafraichissement de la vue qu'il contrôle.

SupportInterface : Facade/DAO/Singleton

La classe *SupportInterface* joue plusieurs rôles en ce qui a trait à la conception du module de soutien. C'est une façade qui permet de faire toute sorte d'interaction avec les discussions. Comme c'est via cette classe que les données sont récupérées et modifiées, il est raisonnable de voir cette classe également comme un DAO (Data Access Object). Comme cette classe doit être utilisée pour tout accès aux données, qu'elle n'a aucun état et qu'il est possible qu'elle soit utilisée à plusieurs endroits, elle sera implémentée comme un singleton.

4.4. Conception du module des Incidents

4.4.1. Vue structurale

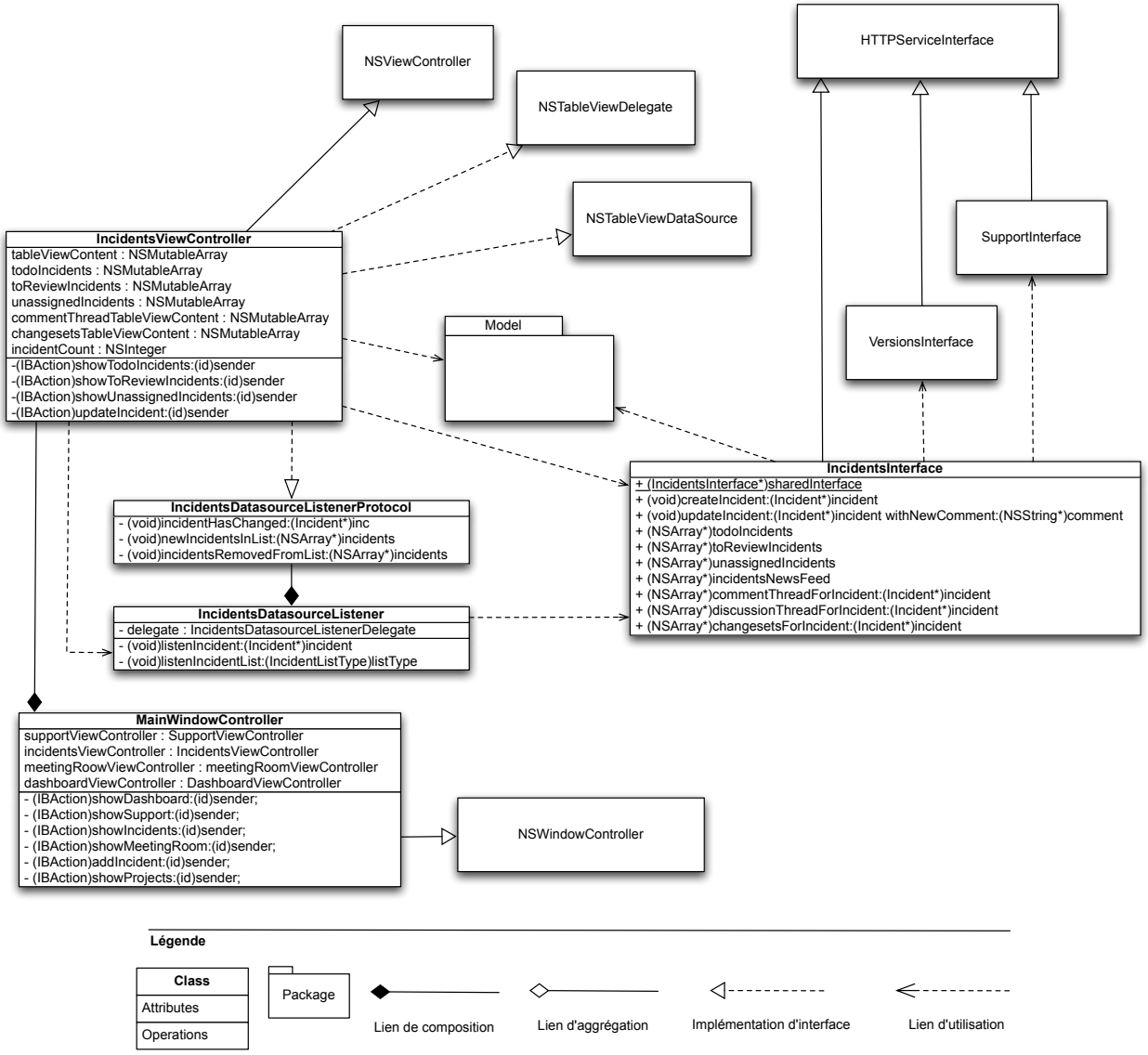


Figure 4.4.1 - Diagramme de classe du module des incidents

4.4.2. Vue de comportement

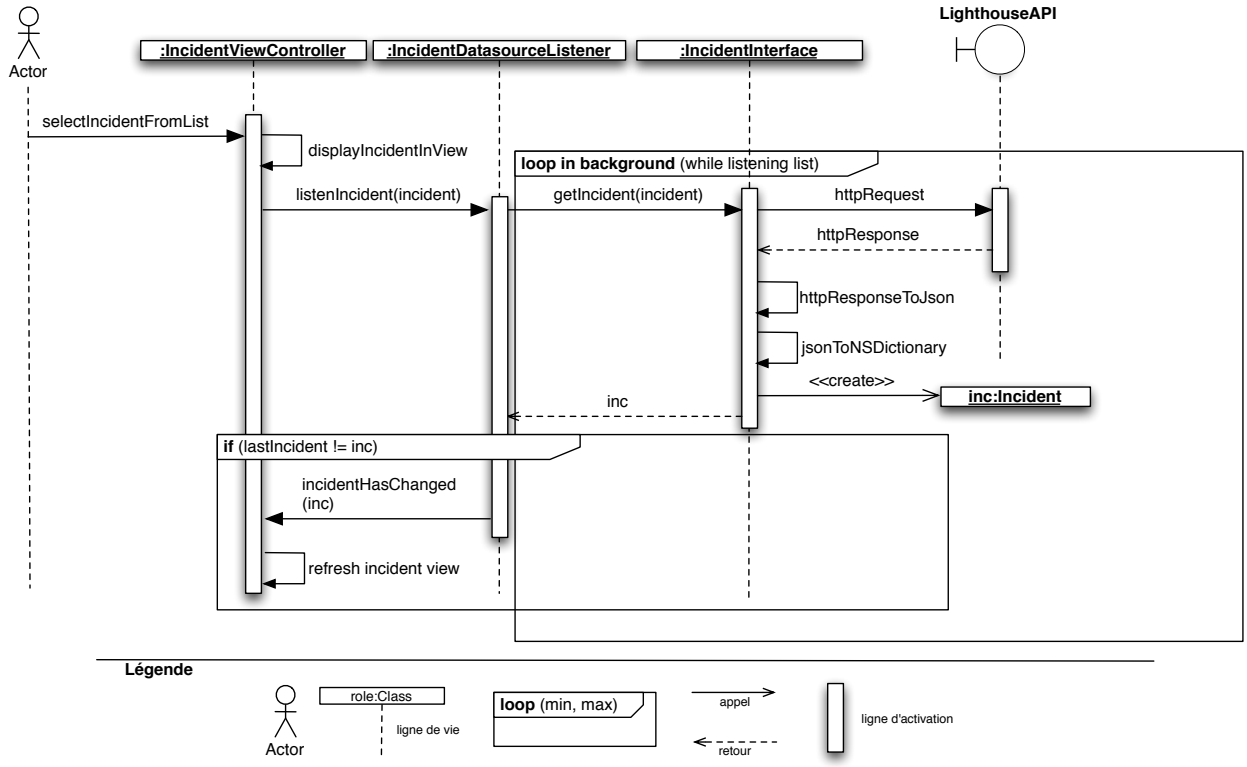


Figure 4.4.2 - Diagramme de séquence pour afficher le détail d'un incident

4.4.3. Autre vue pertinente

4.4.4. Raisonnement

Structure

À noter que le diagramme de classe ne présente pas les détails concernant les éléments de l'interface utilisateur. Par contre, il présente les méthodes des événements de l'interface utilisateur. Ceux-ci portent le type de retour *IBAction*.

Le coeur du module de support est le *IncidentViewController*, c'est de là où sont faites les requêtes à l'interface des incidents pour obtenir et modifier les données du service de gestion des billets. Cette classe implémente les interfaces NSTableViewDataSource et NSTableViewDelegate pour permettre l'affichage des données dans une liste et hérite de la classe NSViewController, qui est la classe générale du framework Cocoa pour contrôler une vue. Dans cette classe, on retrouve principalement des attributs de type NSMutableArray qui sont des listes contenant les trois différents états d'incidents disponibles via l'interface, soit, à faire ("todo"), en attente de revue ("in review") et non assigné ("unassigned").

La classe *IncidentInterface* offre les méthodes pour obtenir les informations des incidents du service de gestion des billets, via des méthodes comme *todoIncidents, toReviewIncidents, unassignedIncidents*, qui retournent toutes des listes d'incidents. Ces méthodes permettant d'obtenir les listes sont appelées du IncidentsDatasourceListener qui notifie le IncidentsViewController seulement si la liste obtenue du serveur a changé. Les autres méthodes qui permettent de faire des actions sur les incidents sont appelées directement du contrôleur de vue. Dans cette classe, les requêtes HTTP sont faites de façon synchrone bloquante. Donc, il est préférable que celles-ci soient appelées dans un fil d'exécution séparé pour ne pas bloquer l'interface utilisateur qui est sur le fil d'exécution principale le temps de la requête. Une autre particularité de cette classe est qu'elle utilise les interfaces des autres modules, soit *SupportInterface* et

VersionsInterface, pour obtenir les discussions des incidents, et pour obtenir les changements au code source des incidents.

La vue pour le module de support se retrouve au sein de la fenêtre principal, c'est pourquoi un lien de composition relie le *MainWindowController* au *SupportViewController.*

Comportement

Pour chaque interaction menant à la consultation de données, la séquence présentée dans le diagramme ci-haut est la même. L'utilisateur demande l'affichage de certaines données, le contrôleur de la vue demande à son *DatasourceListener* d'écouter un item distant en faisant des requêtes en boucle dans un fil d'exécution en arrière-plan. Le *DatasourceListener* utilise l'interface pour interroger le serveur de support. L'interface construit sa requête HTTP, l'envoie au service web, celui-ci retourne une réponse au format XML ou JSON (XML dans le cas de Lighthouse), la chaîne de caractère XML est transformée en *NSDictionnary* via une classe utilitaire XML parser tirée de RestKit. Une fois le dictionnaire en main, l'interface le parcourt et crée ses objets. Dans ce cas-ci, l'interface crée une instance d'*Incident*. La dernière instance de l'*Incident* est gardée à chaque itération pour pouvoir comparer le prochain *Incident* obtenu lors de la prochaine itération. On pourra donc comparer si l'incident a changé depuis la dernière fois. S'il a changé, le contrôleur de vue sera notifié via les méthodes du protocole *IncidentDatasourceListener* qu'il implémente. Une fois notifié, le contrôleur de vue mettra son information à jour et forcera un rafraichissement de la vue qu'il contrôle.

Lazy Loading et Caching

Un incident, en plus d'être composé des informations d'un billet provenant de Lighthouse, peut également être formé d'une discussion provenant de Tender et d'un ou plusieurs changements soumis via Beanstalk. Les informations de la discussion et des changements au code source ne sont pas affichées aux premiers abords lorsque l'utilisateur consulte un incident. L'utilisateur doit cliquer sur un onglet spécifique pour pouvoir afficher soit la discussion ou les changements au code source. Ce n'est qu'au moment où l'utilisateur les demande que la requête au serveur sera effectuée. Une requête au service de soutien sera faite via la *SupportInterface* pour afficher la discussion d'un incident et une requête au service de gestion de version sera faite via la *VersionsInterface* pour afficher les changements au code source. Si toutes les requêtes étaient effectuées pour le premier affichage de l'incident, la latence serait très grande avant que l'information s'affiche. De plus, ce n'est pas à toutes les occasions que l'utilisateur voudra consulter la discussion et les changements au code source, donc ça évite de faire des requêtes HTTP inutiles.

Également, afin de limiter le temps d'attente de l'utilisateur lorsque celui-ci retourne consulter un incident qu'il a déjà consulté auparavant, l'incident s'affichera instantanément tel qu'il était à la dernière consultation. En arrière-plan, le système recommencera à écouter la ressource web auquel l'incident est rattaché afin que l'information se rafraichisse dans le cas où elle aurait changé. Bien entendu, il se peut que l'information en cache ne reflète pas l'information à jour sur le serveur. C'est pourquoi l'information en cache ne sera gardé que pour une durée limitée. L'avantage de la cache est principalement lorsque l'utilisateur consulte plusieurs incidents dans un bref délai. Les chances que l'information des incidents ait changé sont alors très minces.

IncidentsInterface : Facade/DAO/Singleton

La classe *IncidentsInterface* joue plusieurs rôles en ce qui a trait à la conception du module des incidents. C'est une façade qui permet de faire toute sorte d'interaction avec les incidents. Comme c'est via cette classe que les données sont récupérées et modifiées, il est raisonnable de voir cette classe également comme un DAO (Data Access Object). Comme cette classe doit être utilisée pour tout accès aux données, qu'elle n'a aucun état et qu'il est possible qu'elle soit utilisée à plusieurs endroits, elle sera implémentée comme un singleton.

4.5. Conception du module de Clavardage

4.5.1. Vue structurale

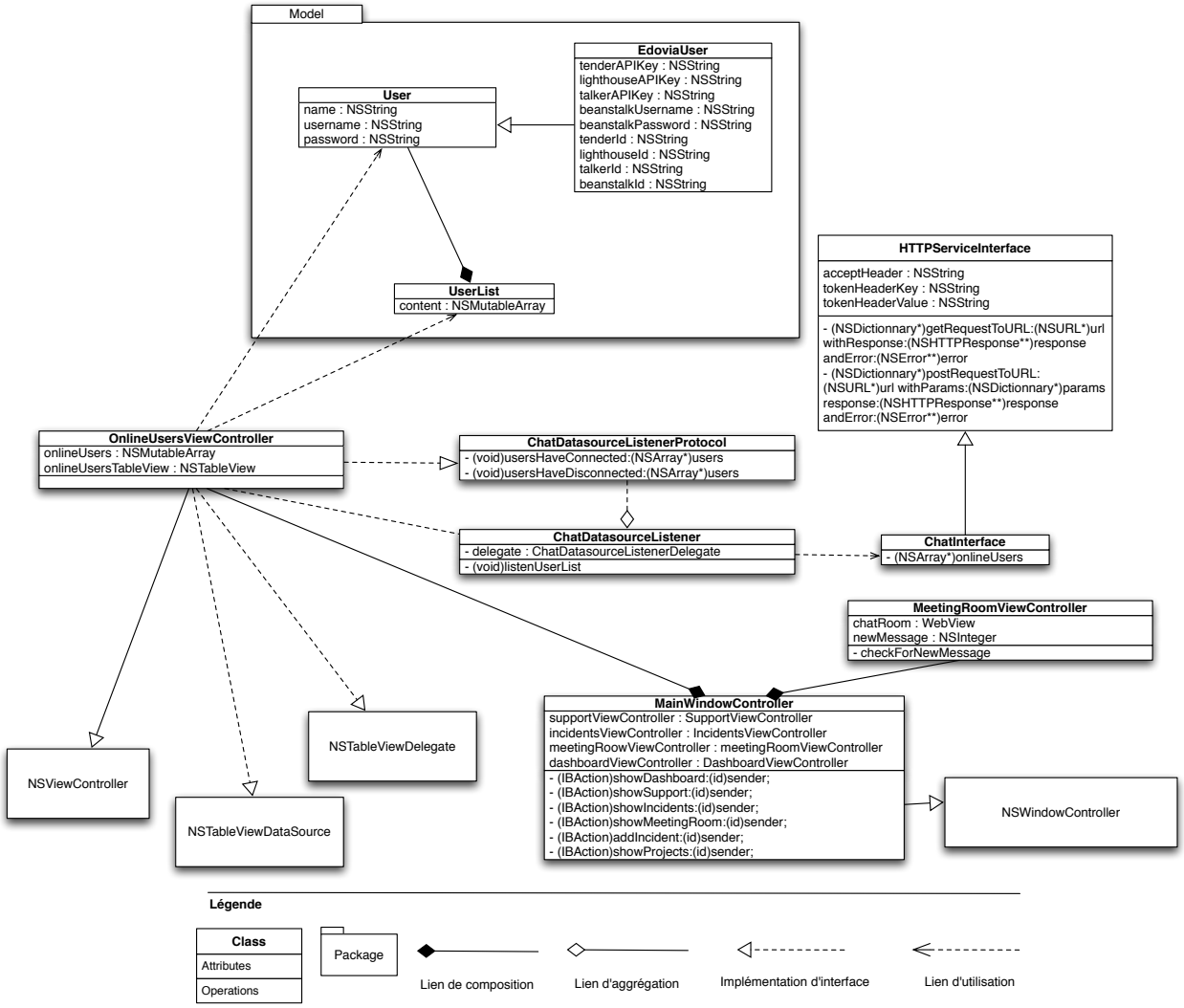


Figure 4.5.1 - Diagramme de classe du module de clavardage

4.5.2. Vue de comportement

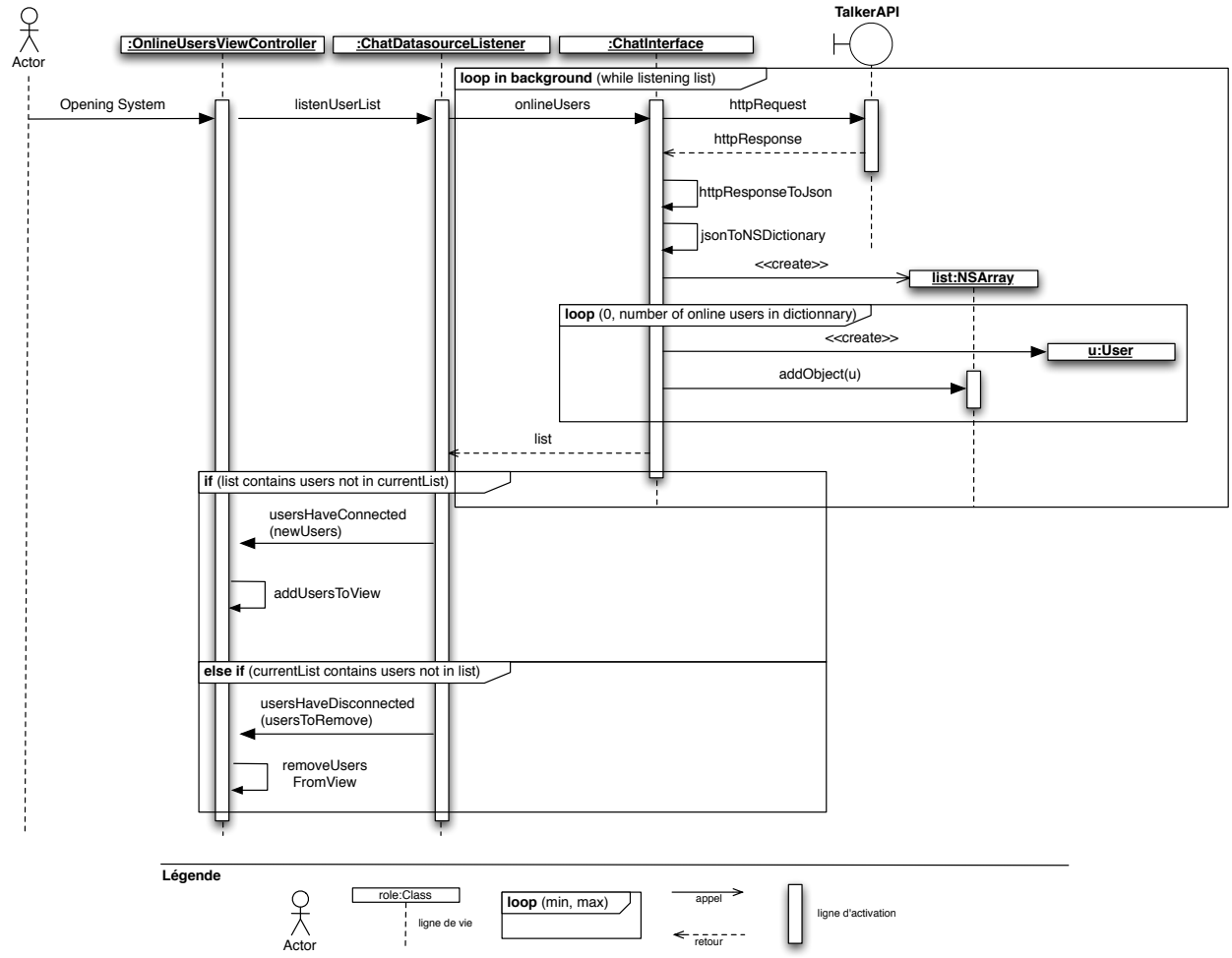


Figure 4.5.2 - Diagramme de séquence pour obtenir le statut des utilisateurs connectés, dans le module de clavardage

4.5.3. Autre vue pertinente

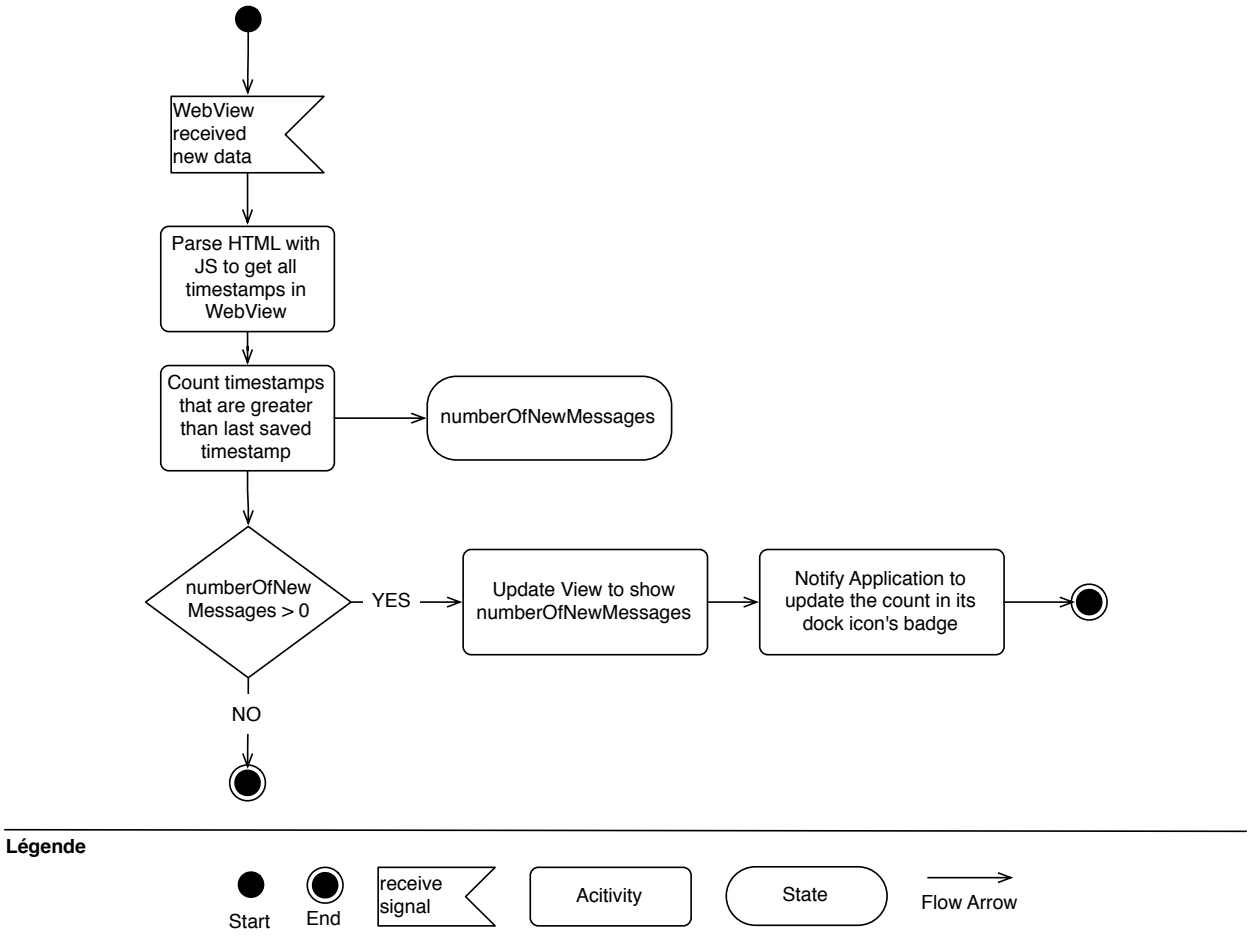


Figure 4.5.3 - Diagramme d'activité démontrant les notifications des nouveaux messages de clavardage

4.5.4. Raisonnement

Le module de clavardage est très simplifié étant donné qu'il serait assez laborieux d'implémenter le protocole du service Talker au sein du système. Le protocole Talker peut s'apparenter au protocole XMPP ou IRC. Comme le fonctionnement désiré est simplement de pouvoir clavarder en groupe via Talker, une simple "WebView" incrustée dans l'application sera suffisante. Un autre aspect relié au clavardage consiste à afficher en tout temps les utilisateurs connectés au système, tel que suggéré dans les interfaces utilisateurs du document de spécifications. Heureusement, l'API Rest de Talker permet d'obtenir la liste des utilisateurs connectés au système de clavardage.

Structure

Deux contrôleurs de vue sont alors nécessaires au module de clavardage. Celui de la classe OnlineUsersViewController pour afficher la liste des utilisateurs connectés, et celui de la classe MeetingRoomViewController pour afficher le "WebView" permettant le clavardage de groupe. La classe OnlineUsersViewController utilise le ChatDatasourceListener pour écouter la liste des utilisateurs connectés sur Talker et elle implémente l'interface ChatDatasourceListenerProtocol afin d'être notifiée lorsqu'un nouvel utilisateur est connecté, ou lorsqu'un utilisateur se déconnecte. Le ChatDatasourceListener utilise ChatInterface pour obtenir ses données provenant du service Rest de Talker. Les deux vues se retrouvent au sein de la fenêtre principale de l'application, c'est pourquoi les deux contrôleurs de vue se

retrouvent agrégés dans le MainWindowController. Le OnlineUsersViewController implémente les interfaces NSTableViewDelegate et NSTableViewDataSource pour pouvoir afficher ses informations sous forme de tableau.

Comportement

Le comportement pour que les utilisateurs connectés soient affichés est sensiblement le même que lors de l'affichage d'une liste de discussion dans le module de soutien ou d'une liste d'incident dans le module des incidents. Le contrôleur de vue s'abonne au Listener pour être notifié des changements à la liste des utilisateurs connectés. Le Listener utilise l'interface de chat pour interroger le service web. Si des changements sont observés, ceux-ci sont signalés au contrôleur de vue pour qu'il mette à jour la vue.

Comme une "WebView" sera utilisée pour afficher la fonctionnalité de clavardage, le nombre de nouveaux messages reçus devra être calculé manuellement. En d'autres mots, aucune notification de la part du service de clavardage ne sera reçue pour dire qu'un nouveau message est arrivé dans la salle de clavardage. Il sera par contre possible d'être notifié à chaque fois que le contenu de la "WebView" aura changé, donc à chaque fois qu'un message est reçu. Un horodatage ("timestamp") sera sauvegardé à chaque fois que l'utilisateur quittera la vue de la salle de clavardage. Alors, à chaque fois que la "WebView" aura changé, une opération JavaScript sera lancée sur le contenu de la "WebView" pour retrouver tous les horodatages ("timestamp") des messages affichés dans la page. Il sera donc possible de compter les horodatages antérieures au dernier horodatage sauvegardé pour connaître le nombre de nouveaux messages. Lorsque l'utilisateur consultera la vue de clavardage à nouveau, le compte de nouveau message sera remis à zéro.

ChatInterface : Facade/DAO/Singleton

La classe *ChatInterface* joue plusieurs rôles en ce qui a trait à la conception du module de soutien. C'est une façade qui permet de faire les interactions avec le service de clavardage. Comme c'est via cette classe que les données sont récupérées, il est raisonnable de voir cette classe également comme un DAO (Data Access Object). Comme cette classe doit être utilisée pour tout accès aux données, qu'elle n'a aucun état et qu'il est possible qu'elle soit utilisée à plusieurs endroits, elle sera implémentée comme un singleton.

4.6. Conception de l'interface à la gestion des versions

4.6.1. Vue structurale

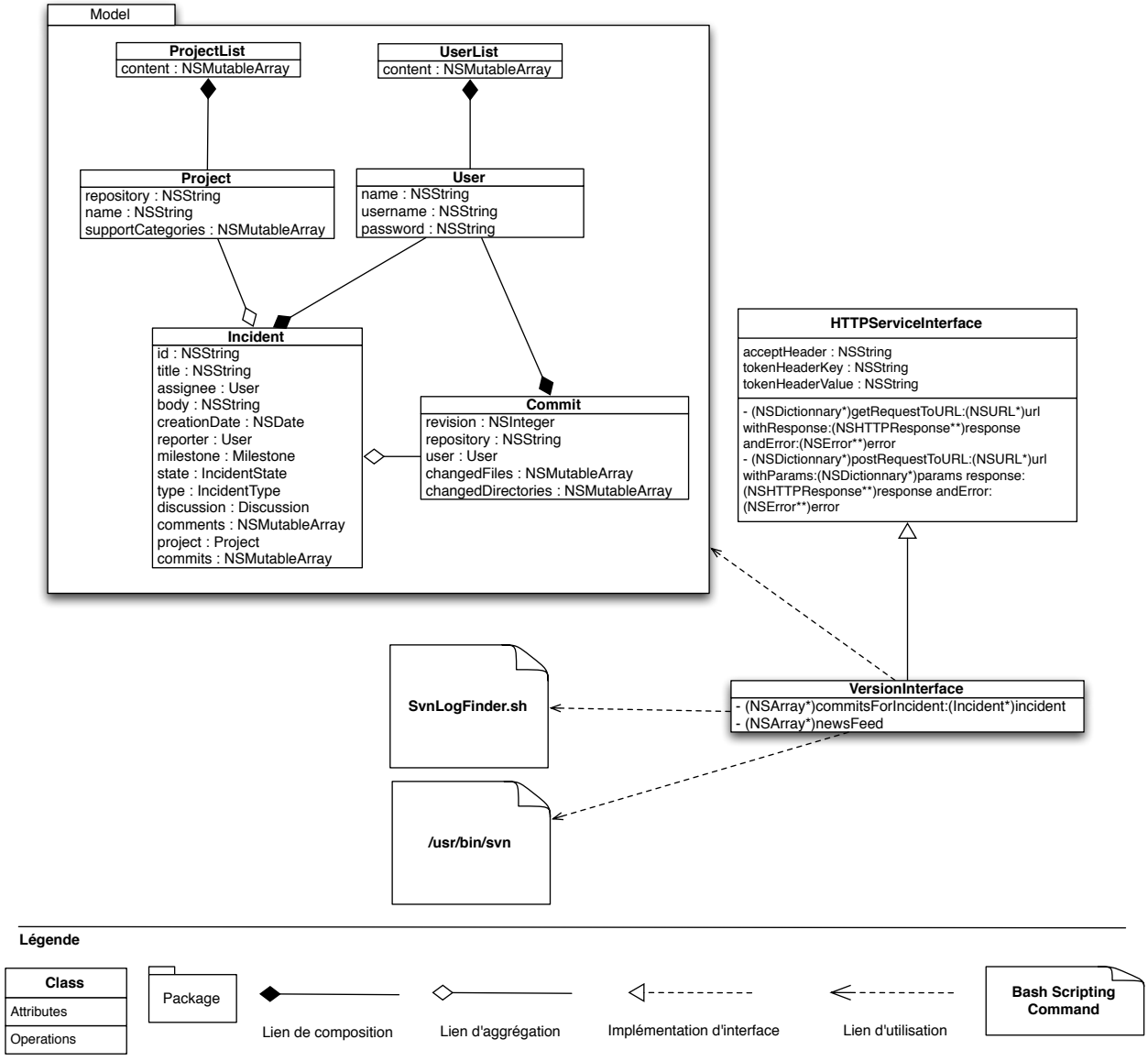


Figure 4.6.1 - Diagramme de classe de l'interface des versions

4.6.2. Vue de comportement

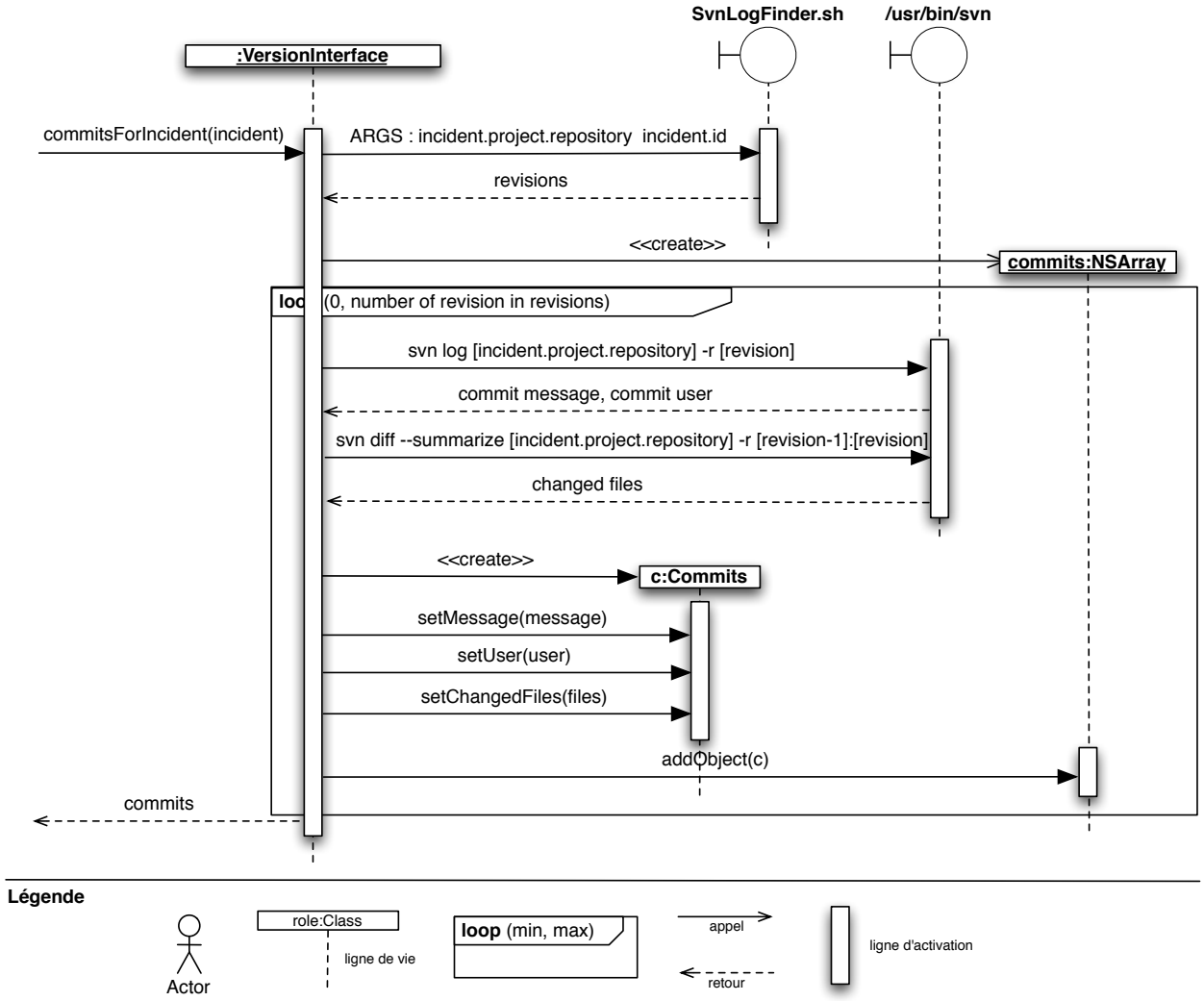


Figure 4.6.2 - Diagramme de séquence pour obtenir les propagations (commits) reliées à un incident

4.6.4. Raisonnement

Au départ, le système devait utiliser l'API Rest de Beanstalk pour trouver les propagations reliées à un incident. Cependant, en consultant la documentation de l'API, il fut remarqué que ce dernier ne permettait pas d'exécuter la tâche avec une seule requête HTTP. La seule façon de faire consistait à demander tous les changements sur le dépôt propre au projet de l'incident, et de vérifier si dans chacun des changements, l'identifiant de l'incident se retrouvait dans le message de propagation. De plus, le service web ne pouvait que retourner 30 résultats par requête. S'il y avait plus que 30 changements pour un dépôt, il fallait alors faire plus d'une requête, ce qui augmentait considérablement le temps pris pour trouver tous les changements reliés à un incident.

Alors, plutôt que d'utiliser l'API Rest de Beanstalk pour trouver les propagations SVN reliées à un incident, des commandes SVN standards en ligne de commande ont été choisies pour exécuter la tâche.

Structure

La classe VersionInterface est une interface qui hérite de HTTPServiceInterface comme toutes les autres interfaces du système. Bien que le service web de Beanstalk ne soit pas utilisé pour trouver les propagations SVN reliées à un incident, le service web est tout de même utilisé pour connaître les dernières activités de l'entreprise sur Beanstalk via la méthode *newsFeed.* C'est la méthode *commitsForIncident* qui retourne les propagations propres à un incident. C'est cette méthode qui exécutera directement des commandes SVN.

Pour trouver les révisions qui sont reliées à un incident, le script Bash SvnLogFinder.sh devra avoir été conçu. Celui-ci exécutera la commande ci-dessous, et prendra en paramètre l'URL du dépôt svn relié au projet de l'incident et le terme recherché dans les messages (Exemple, pour l'incident #21, on recherche "[#21").

```
svn log ${1} | perl -e '$/="-"x72;while(<STDIN>){print"$r\n"if(($r)=split)&&/@ARGV/)}' "\[#${2}"
```

source : http://www.perlmonks.org/?node_id=819750

Pour trouver le message de propagation, l'utilisateur et les fichiers modifiés d'une révision, aucun script n'a besoin d'être conçu. Des commandes SVN simples suffiront.

Comportement

Lorsqu'un utilisateur voudra consulter les propagations svn reliées à un incident, le mécanisme de "Listener" habituel sera utilisé. Cependant, du côté de l'interface, plutôt que d'interroger un service web, ce seront des commandes SVN qui seront exécutées.

D'abord la liste des révisions concernées par l'incident sera récupérée avec une commande du type :

```
bash SvnLogFinder.sh https://edovia.svn.beanstalkapp.com/touchpaduniversal 44
```

Le script retournera la liste des révisions sous la forme :

```
r100
r635
r827
```

Pour chaque révision, le message de propagation et l'utilisateur seront récupérés avec une commande du type :

```
svn log https://edovia.svn.beanstalkapp.com/touchpaduniversal -r 100
```

et retournera un résultat sous la forme :

```
------------------------------------------------------------------------
r100 | aplourde | 2011-10-18 16:29:28 -0400 (Tue, 18 Oct 2011) | 1 line

[#41 state:resolved] Implement Touch-hold to start dragging since three finger swiping is now customizable
------------------------------------------------------------------------
```

La liste des fichiers modifiés sera récupérée avec une commande du type :

```
svn diff --summarize https://edovia.svn.beanstalkapp.com/touchpaduniversal -r 99:100
```

et retournera un résultat sous la forme :

```
M      https://edovia.svn.beanstalkapp.com/touchpaduniversal/trunk/TouchPadUniversal.xcodeproj/project.pbxproj
M      https://edovia.svn.beanstalkapp.com/touchpaduniversal/trunk/Constants.h
M      https://edovia.svn.beanstalkapp.com/touchpaduniversal/trunk/Shared/Settings.bundle/en.lproj/Root.strings
M      https://edovia.svn.beanstalkapp.com/touchpaduniversal/trunk/Shared/Settings.bundle/Root.plist
M      https://edovia.svn.beanstalkapp.com/touchpaduniversal/trunk/Shared/Classes/TouchDelegate.h
M      https://edovia.svn.beanstalkapp.com/touchpaduniversal/trunk/Shared/Classes/TouchDelegate.m
A      https://edovia.svn.beanstalkapp.com/touchpaduniversal/trunk/click_off.wav
```

Enfin, pour chaque révision trouvée, tous les résultats seront traités pour encapsuler l'information pertinente dans un objet *Commit.* Chacun de ces objets sera ajouté dans une liste qui sera retournée à l'appelant.

Annexe D
Matrice de traçabilité

Incidents
TMX-01
Traceability Matrix

Cas d'utilisation	Diagramme de conception	Classe ou module de code	Méthode de validation	Résultat de la validation
UC01	4.3.1 et 4.3.2	SupportViewController et SupportInterface	Test fonctionnel	TRUE
UC02	4.3.1	SupportViewController et SupportInterface	Test fonctionnel	TRUE
UC03	4.3.1	SupportViewController et SupportInterface	Test fonctionnel	TRUE
UC04	4.3.1	SupportViewController et SupportInterface	Test fonctionnel	TRUE
UC05	4.3.1	SupportViewController et SupportInterface	Test fonctionnel	TRUE
UC06	4.4.1	NewIncidentViewController et IncidentsInterface	Test fonctionnel	TRUE
UC07	4.4.1	IncidentsViewController et IncidentsInterface	Test fonctionnel	TRUE
UC08	4.4.1	IncidentsViewController et IncidentsInterface	Test fonctionnel	TRUE
UC09	4.4.1 et 4.4.2	IncidentsViewController et IncidentsInterface	Test fonctionnel	TRUE
UC10	4.4.1	IncidentsViewController et IncidentsInterface	Test fonctionnel	TRUE
UC11	4.5.1	MeetingRoomViewController	Test fonctionnel	TRUE
UC12	*Abandon de cette fonctionnalité, voir Chapitre 7 du Rapport Technique*		Test fonctionnel	FALSE
UC13	4.4.1	IncidentsViewController et IncidentsInterface	Test fonctionnel	TRUE
UC14		ProjectsViewController, IncidentsInterface et SupportInterface	Test fonctionnel	TRUE

Exigences fonctionnelles	Diagramme de conception	Classe ou module de code	Méthode de validation	Résultat de la validation
REF01	3.2.3 et 4.2.1	IncidentsAppDelegate, UserList, ProjectList	Test fonctionnel	TRUE
REF02	NA	MainWindowController et IncidentsAppDelegate	Test fonctionnel	TRUE
REF03	NA	MainWindowController, SupportViewController et IncidentsViewC	Test fonctionnel	TRUE
REF04	3.4.1 et 3.4.6	Tous les DatasourceListener	Test fonctionnel	TRUE
REF05	NA	MainWindowController et MeetingRoomViewController	Test fonctionnel	TRUE
REF06	NA	ProjectsViewController, IncidentsInterface et SupportInterface	Test fonctionnel	TRUE

Exigences non fonctionnelles	Diagramme de conception	Classe ou module de code	Méthode de validation	Résultat de la validation
RENF01	NA	MainWindowController	Test fonctionnel	TRUE
RENF02	NA	MainWindowController	Test fonctionnel	TRUE
RENF03	NA	NA	Test fonctionnel	TRUE
RENF04	3.4.1, 3.4.6,	HTTPServiceInterface	Test fonctionnel	TRUE
RENF05	NA	NA	Test fonctionnel	TRUE
RENF06	NA	Tous les DatasourceListener	Test fonctionnel	TRUE
RENF07	NA	Tous les DatasourceListener	Test fonctionnel	TRUE
RENF08	NA	MeetingRoomViewController	Test fonctionnel	TRUE
RENF09	NA	NA	Test fonctionnel	TRUE
RENF10	3.2.1	Toutes les Interfaces	Test fonctionnel	TRUE
RENF11	NA	NA	Test fonctionnel	TRUE
RENF12	NA	NA	Test fonctionnel	TRUE

*Plusieurs exigences fonctionnelles et non fonctionnelles ne sont représentables par des diagrammes UML