École de technologie supérieure
L'ÉTS est une constituante du réseau de l'Université du Québec
Le génie pour l'industrie

Rapport de laboratoire

N^{o} de laboratoire	2
Étudiant(s)	Sébastien Lago
	Marc-André Allard
	Israël Hallé
Code(s) permanent(s)	LAGS04128102
	HALI17049101
	ALLM09029106
Cours	LOG430
Session	Hiver 2014
Groupe	02
Professeur	B. Galarneau
Chargés de laboratoire	
Date de remise	19 février 2014

Contents

Introduction

Être capable de mapper l'implémentation orientée objet à une architecture précise est important pour pouvoir créer une architecture logicielle cohérente. Plusieurs outils facilitent la transition entre le code et une architecture, par exemple les diagrammes de classes et les matrices de dépendances. Dans la première partie de ce laboratoire, nous appliquerons les mêmes modifications au code que lors du laboratoire précédent. La différence est que les modifications seront effectuées au sein d'une autre architecture, qui est à invocation implicite. Ensuite, nous procéderons à l'analyse de cette nouvelle architecture pour la comparer avec celle en couches.

Implémentation

Lors de ce deuxième laboratoire, l'implémentation a été très rapide et a nécessité que très peu de nouveau de code. En effet, étant donné que les fonctionnalités demandées étaient les mêmes que le premier laboratoire, il a été possible de réutiliser une majeure partie du code de notre première implémentation. Une partie importante du temps a donc été alloué à comprendre le fonctionnement de l'implémentation du logiciel fourni, suivi par l'intégration des nouvelles fonctionnalités en suivant l'implémentation actuellement en place.

Ainsi, pour chaque nouvelle fonctionnalité demandée, une classe héritant de Communication a été créée et l'implémentation de la fonctionnalité a été insérée dans la méthode « update ». Par la suite, les trois nouvelles classes sont instanciées dans « SystemInitialize » et « Executive » a été mise à jour pour signaler aux nouvelles fonctionnalités. Finalement, le code pour afficher le menu ainsi que les différents éléments restants requis pour les nouvelles fonctionnalités ont pu être repris sans avoir eu à être modifiés.

Analyse architecturale

Vue architecturale

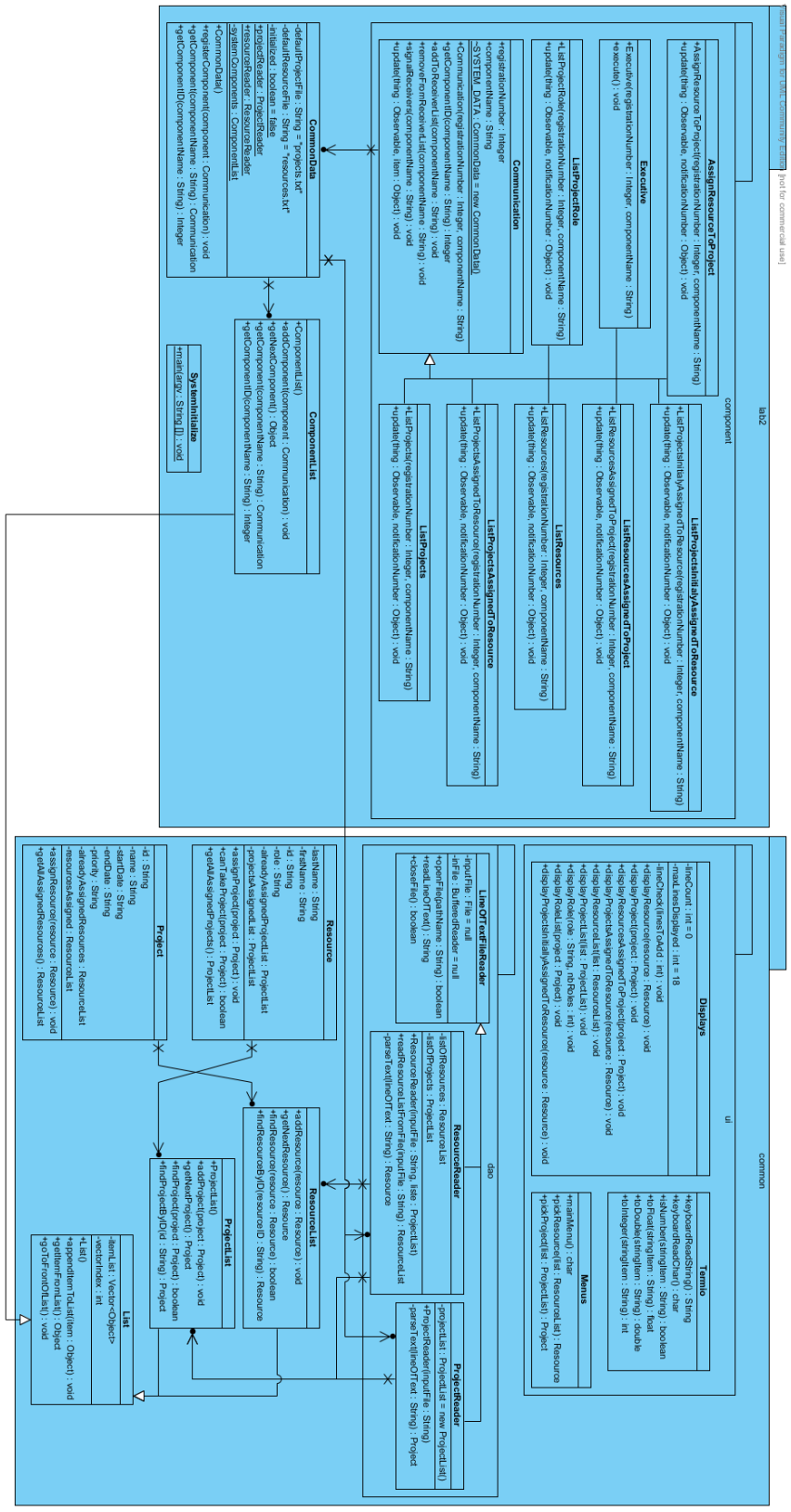

Figure 1 - Diagramme de classe

Le diagramme de classe permet de visualiser l'ensemble des composants de notre système ainsi que les dépendances entre ceux-ci. Nous pouvons donc rapidement voir que le système est totalement indépendant des classes héritant de Communication. Ces classes contenant l'implémentation des règles d'affaires sont utilisées grâce à l'interface de la classe Communication qui permet d'envoyer des signaux afin d'appeler certaines fonctionnalités. Nous pouvons donc voir l'un des plus gros avantages de l'architecture à invocation implicite. Elle permet d'isoler complètement les règles d'affaires du système et de pouvoir en ajouter ou en supprimer sans même avoir à modifier le reste du système.

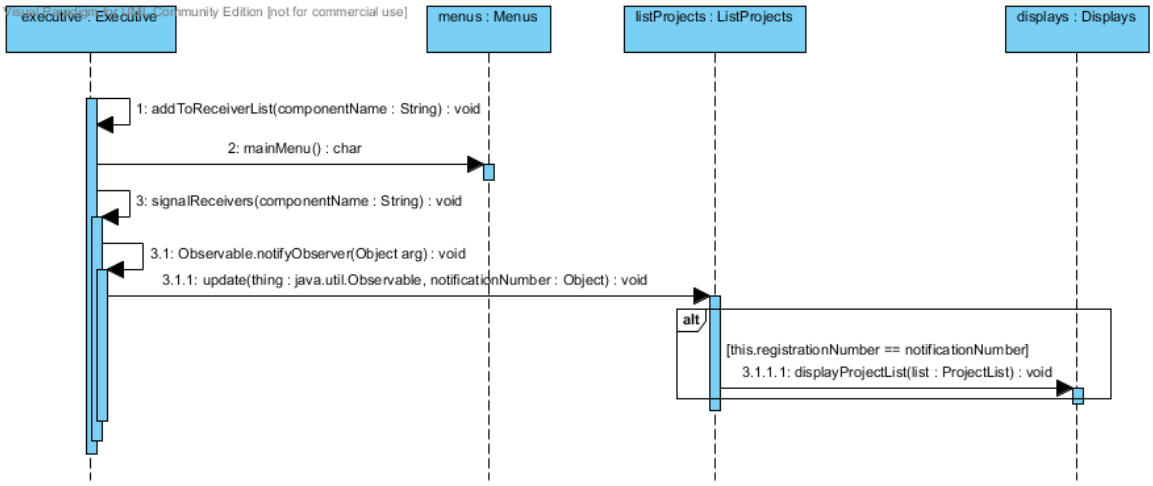

Figure 2 - Diagramme de séquence

Le diagramme de séquence permet quant à lui de voir le déroulement d'une commande au sein de notre système, du menu à l'affichage du résultat. Ce diagramme est particulièrement utile puisqu'il démontre le fonctionnement dynamique de l'invocation implicite.

Nous pouvons donc voir qu'« executive » enregistre la fonctionnalité qu'il propose avec « addToReceiverList ». Notez que toutes les implémentations des fonctionnalités sont préalablement ajoutées dans une liste globale par « SystemInitialize ». Par la suite, la classe « Menus » est utilisée afin d'afficher un menu et récupérer le choix de l'utilisateur. Selon le choix de l'utilisateur, « executive » envoie le signal de la fonctionnalité demandé. Le signal est simplement implémenté à l'aide du patron Observer/Observable. Ainsi, toutes les composantes héritant de « Communication » recevront le signal à partir de la méthode « update ». Dans notre exemple, la composante « listProjects » reçoit le signal et vérifie qu'il est bien du type qu'il traite. Le cas échéant, la composante exécutera sa logique d'affaires et affichera la liste des projets du système. Les autres composantes recevront aussi le signal, mais l'ignoreront étant donné qu'il ne leur est pas destiné.

Déviation du système original par rapport l'architecture à invocation implicite

Nos modifications du système n'ont pas dévié de l'architecture originale et nous avons gardé le même fonctionnement que les anciennes fonctionnalités pour nos nouvelles fonctionnalités. Ainsi, nous avons créé une classe héritant de « Communication », ajouté sa création dans « SystemInitialize » et enregistré le nouveau signal dans « Executive ». Avec ces modifications, le système modifié est pratiquement identique aux systèmes originaux et aucune nouvelle dépendance n'a été ajoutée aux classes existantes.

Comparaison des matrices de dépendances

Plusieurs modifications ont été apportées à l'ancien système pour qu'il respecte l'architecture à invocation implicite. Cependant, le système est tout de même séparé en 2 parties principales : « lab2 » et « common ». Même s'il n'y a pas de dépendance vers le haut, de « common » à lab2, on ne peut pas vraiment les appeler des couches. La raison est que les classes qui s'y trouvent ne sont pas vraiment liées par leur fonctionnalité. Les classes ajoutées dans le lab2 se retrouvent dans le paquet lab2, et les anciennes classes se retrouvent dans « common ». Pour avoir une architecture en couche, il aurait fallu qu'il n'y ait pas d'interdépendance entre le paquet lab2 et le paquet « lab2.component ». Cependant, comme les classes de « common » n'ont pas vraiment été modifiées depuis le système précédent, ils sont encore séparés en couches. Le paquet « common.ui » représente l'ancienne couche présentation et le paquet « common.dao » et « common » représente l'ancienne couche gestion.

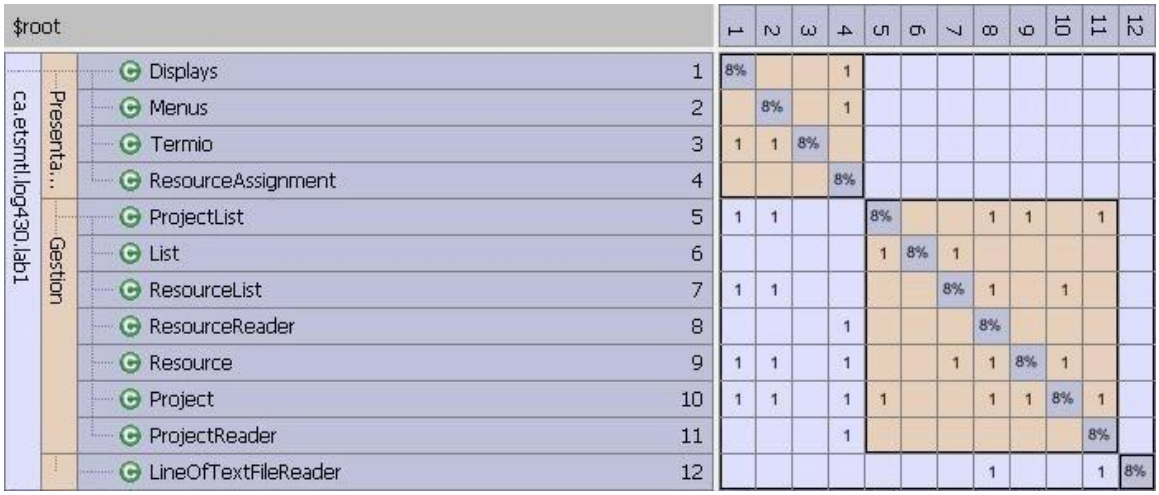

Figure 3 - DSM du laboratoire 1

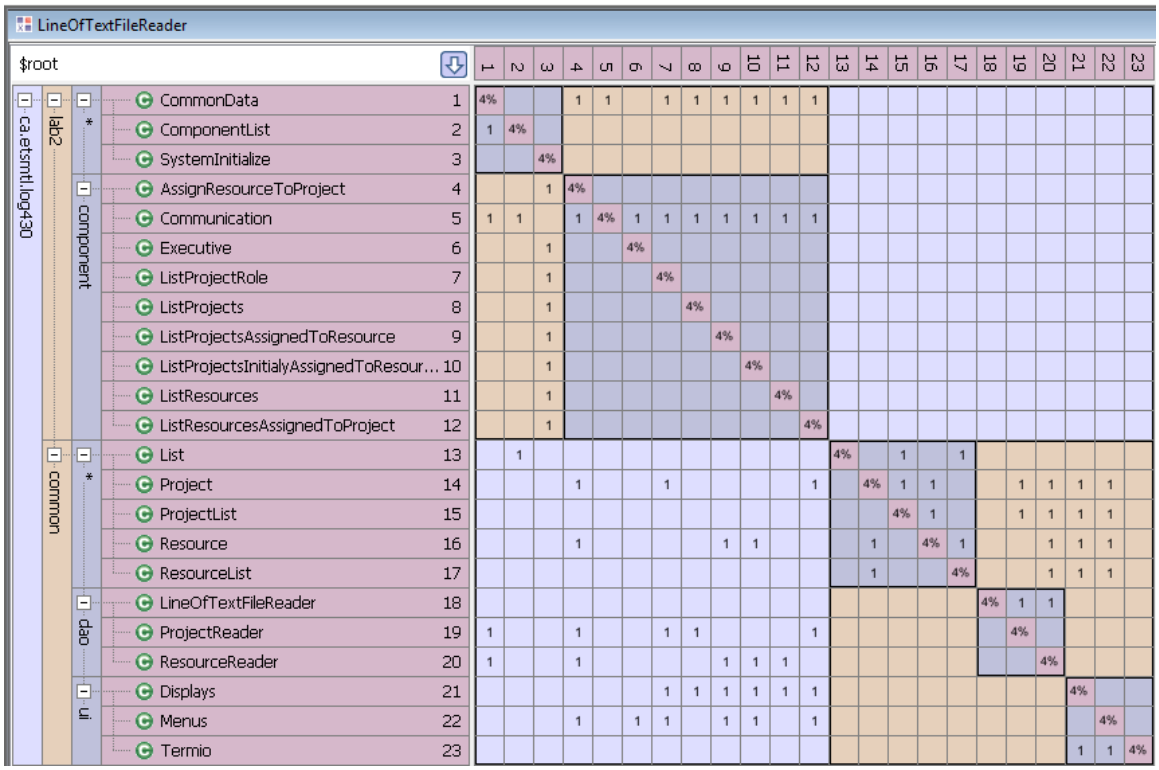

Figure 4 - DMS du système actuel

Différences des modifications par rapport au laboratoire 1

Dans le laboratoire 1, dont l'architecture est en couches, chacune des fonctionnalités du projet est traitée dans une même classe (ResourceAssignment). Cette classe contient la sélection des cas et l'action est exécutée à l'intérieur de celui-ci. Ce cas va chercher les données en utilisant les classes de la couche inférieure puis affiche les données à l'écran par le biais de la classe « Display ». Contrairement au laboratoire précédent, le projet du laboratoire 2, à l'architecture à invocation implicite, traite chaque cas dans des classes à part, qui se nomment des composants. Au démarrage de ce projet, ces composants sont instanciés en leur attribuant un nom et un numéro d'identification. Parmi ces composants, il existe un composant « Executive » qui se charge d'envoyer des notifications aux autres composants. La sélection des cas se fait à l'intérieur de ce composant. Lors de la sélection de celui-ci, une notification est envoyée au bon composant à travers le bus virtuel. En réalité, tous les composants reçoivent la notification. Les composants vérifient l'identifiant passé en paramètre, et seul le composant qui possède cet identifiant réagit à la notification pour traiter le cas demandé par l'utilisateur. Le principal avantage de cette architecture par rapport à celle par couches est que les objets sont moins couplés, car l'invocation des méthodes est implicite. L'autre principal avantage est que l'entretien et la réutilisation du projet sont beaucoup plus simples qu'avec le modèle par couches. Par exemple, pour ajouter un nouveau cas, il suffit d'ajouter une classe « Composant », de traiter ce cas à l'intérieur de celui-ci et de l'enregistrer avec un événement. Un des inconvénients de l'architecture à invocation implicite est que les composants dépendent des classes de la présentation des données. Une modification à l'une de ces classes est

susceptible de se répercuter sur les composants. De plus, lors d'une modification, on doit s'assurer que lors de l'envoi d'une notification, un seul composant répond à cette notification, sinon cela pourrait occasionner des effets indésirables, car on ne peut prédire quand et dans quel ordre cet événement sera traité.

Conclusion

Dans ce laboratoire, nous avons apporté les mêmes modifications que le laboratoire précédent. Nous avons ensuite analysé ce projet en élaborant une vue de ce système, pour ensuite connaitre toute déviation du système original par rapport à l'architecture à invocation implicite. Pour terminer, nous avons comparé l'architecture de ce système par rapport à celui par couche en comparant leur matrice de dépendances et en identifiant les avantages et inconvénients de chacun d'eux. Il aurait été intéressant de tenter de pousser l'invocation implicite plus loin et d'abstraire la logique d'affichage et de chargement de donnée au sein de composante séparée qui afficherait. Ainsi, nous aurions pu rendre chaque partie de notre système complètement indépendante les unes des autres. Par contre, bien que l'invocation implicite permette de réduire le couplage entre les modules, il est important de noter qu'elle réduit la possibilité d'analyse statique et est donc plus probable de laisser des erreurs d'inattentions passer silencieusement lors de la compilation. Un autre problème de cette architecture est son coût de complexité par rapport à un simple appel de fonction.

Annexe A

Plan de test

Nous avons créé une classe de test unitaire qui permet de tester l'application. Pour ce faire, le dossier « test » fourni avec l'application est requis. Ce dossier contient les fichiers projects.txt et resouces.txt, contenant suffisamment de données pour tester l'application. Ce dossier doit être placé à la racine du programme. Pour lancer les tests, il suffit de lancer la méthode Main() de la classe TestModifications.java. Toutes les nouvelles fonctionnalités ajoutées au système sont testées.

1. La liste des projets auxquels une ressource était déjà affectée avant l'exécution courante du système est affichée.
2. Tous les rôles ayant été assignés à un projet spécifique, incluant les rôles assignés avant l'exécution courante **et** les rôles assignés durant l'exécution, sont affichés
3. Une erreur est lancée quand on tente d'assigner une ressource à un projet si ça l'occupait plus de 100 % de son temps. De plus, l'occupation de la ressource est basée sur la priorité du projet.

Alternativement, il est possible de tester manuellement les nouvelles fonctionnalités :

1. Partez le programme, entrez 5 et choisir une ressource pour afficher les projets initialement associés à cette ressource.
2. Partez le programme, entrez 7 et assignez autant de ressources que voulut à un projet. Ensuite, entrez 6 et choisissez ce projet pour afficher tous les rôles lui ayant été assigné.
3. Partir le programme, entrez 7 pour assigner une ressource à un projet et notez la priorité de ce projet. Puis, entrez de nouveau 7 et associez cette même ressource à un autre projet. Si l'occupation[1] de la ressource est plus de 100 % après avoir été assigné à ce nouveau projet, une erreur est lancée. Sinon, l'association est un succès.

[1] Projet de priorité H = occupe à 100%, projet de priorité M = occupe à 50%, projet de priorité L = occupe à 25&.