

**Département de génie logiciel et des T.I.**

Rapport de Laboratoire

|  |  |
| --- | --- |
| **Numéro du laboratoire** | 03 |
| **Nom du laboratoire** | Le style architectural Invocation implicite |
| **Étudiant(s)** | Jean-Simon Bonin  Xavier Duval  Martin Gingras |
| **Code(s) permanent(s)** | DUVX20048607  BONJ02098701  GINM10108708 |
| **Cours** | LOG 430 |
| **Session** | Hiver 2012 |
| **Groupe** | 01 |
| **Chargé(e) de laboratoire** | Samir Djeffal |
| **Date** | 12 mars 2012 |

Table des matières

[Introduction 3](#_Toc319246851)

[Première partie 4](#_Toc319246852)

[Deuxième partie 5](#_Toc319246853)

[Discussion 7](#_Toc319246854)

[Conclusion 8](#_Toc319246855)

# Introduction

Ce laboratoire porte sur les liens entre une architecture du style « Invocation implicite » et les implémentations orienté objet qui la compose. Dans un premier temps, nous avions à modifier une architecture existante afin de supporter quelques fonctionnalités supplémentaires tout en prenant soin de respecter ce style architectural. La deuxième partie consistait à analyser les interdépendances de cette architecture avant et après les modifications au système avec l’outil de création de matrice de dépendance Lattix LDM.

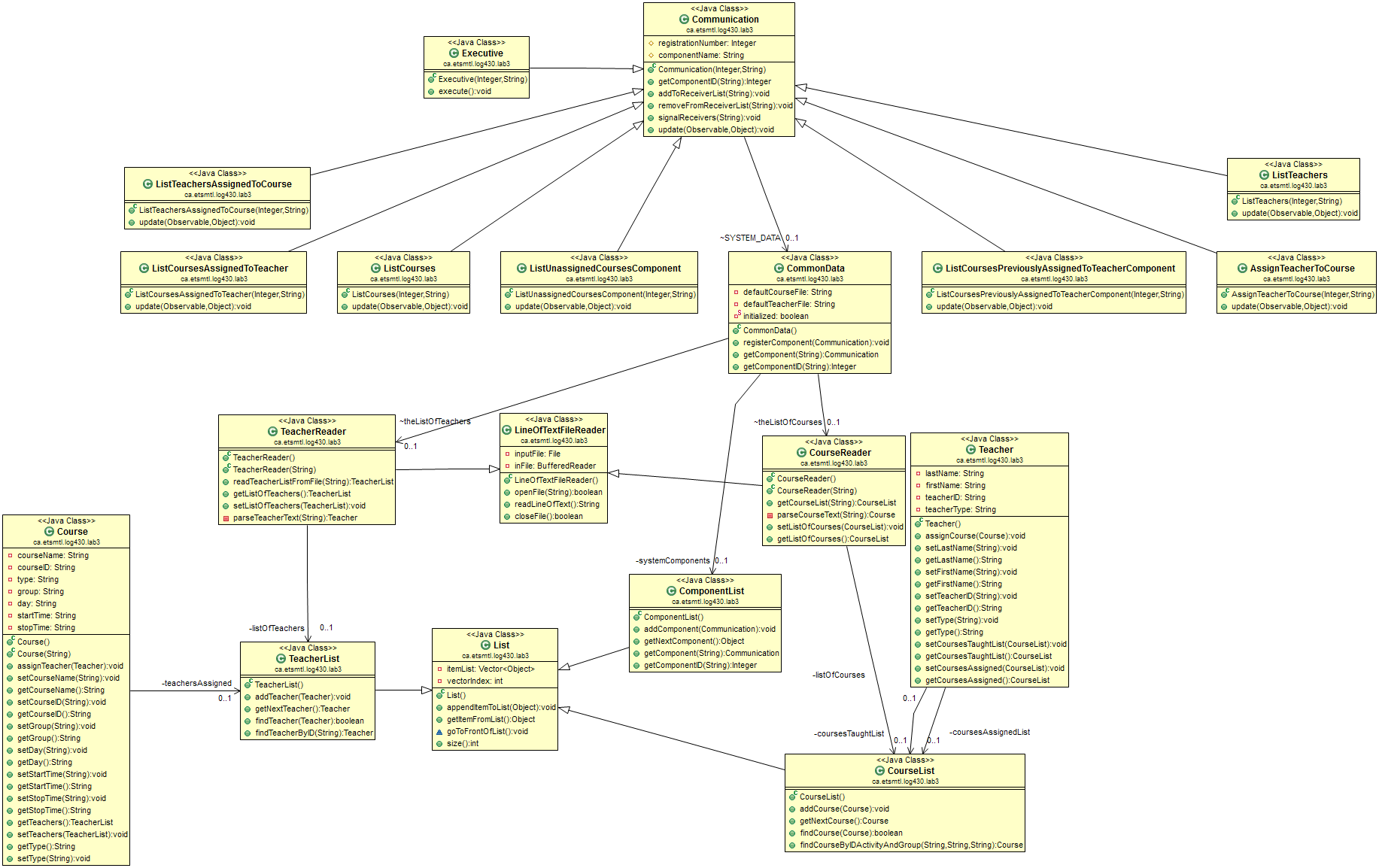
La suite de ce rapport contient quatre sections. D’abord, une section présentera une description sommaire de notre implantation pour ce laboratoire. Dans la section suivante, nous répondons aux questions de l’énoncé de ce troisième laboratoire et effectuons l’analyse architecturale du système. La troisième section est une discussion comparant les trois laboratoires sur lesquels nous avons eu à travailler cette session. Une conclusion vient clore le document en faisant un retour sur les objectifs du laboratoire.

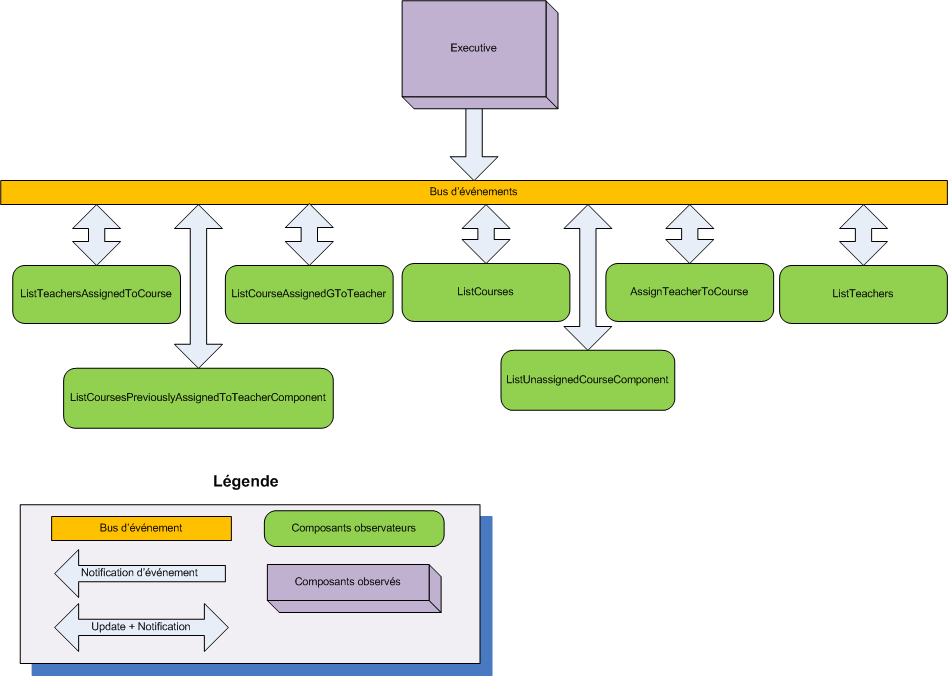
# Première partie

L’implantation pour ce troisième laboratoire c’est très bien passé puisque l’essentiel du code du laboratoire 2 a pu être réutilisé. Chaque méthode que nous avions ajoutée dans la classe « Display » pour le laboratoire 2 ainsi que la classe « Validator » ont tous été réutilisé. L’ensemble des conditions à modifier et des options à ajouter était donc déjà implémenté, les seuls changements étaient donc au niveau des endroits ou faire les appels de ces méthodes. Le plus difficile fut donc de comprendre le fonctionnement de l’architecture afin de déterminer comment déclencher les procédures pour chaque option.

# 

# Deuxième partie

a) Diagramme de classe:

Vue architecturale:

b)

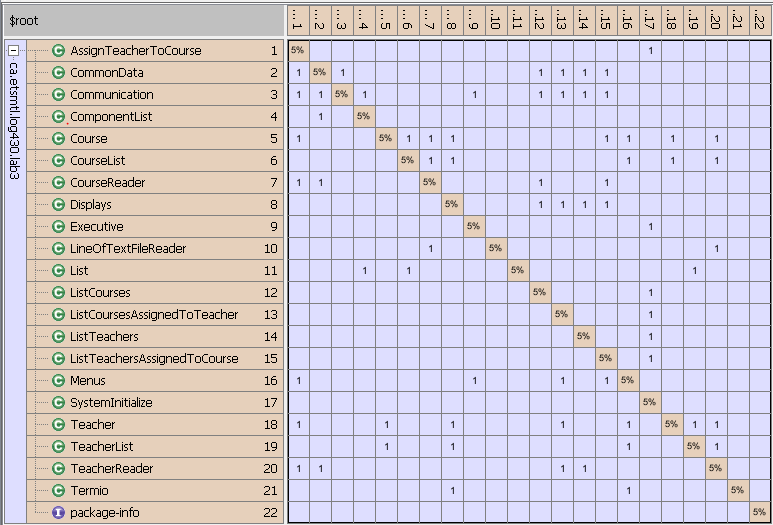
Pour faire des modifications au code d’un projet dont le style d’architecture est l’invocation implicite, il faut bien comprendre le fonctionnement du patron de conception logiciel « Observer ». Dans notre cas, la classe qui est « Observable » est la classe *« Executive »* et les composants logiciels de l’application sont les *« Observers »*. La class *« Executive »* sert à écouter pour enregistrer n’importe quelle modification de l’état de l’application.

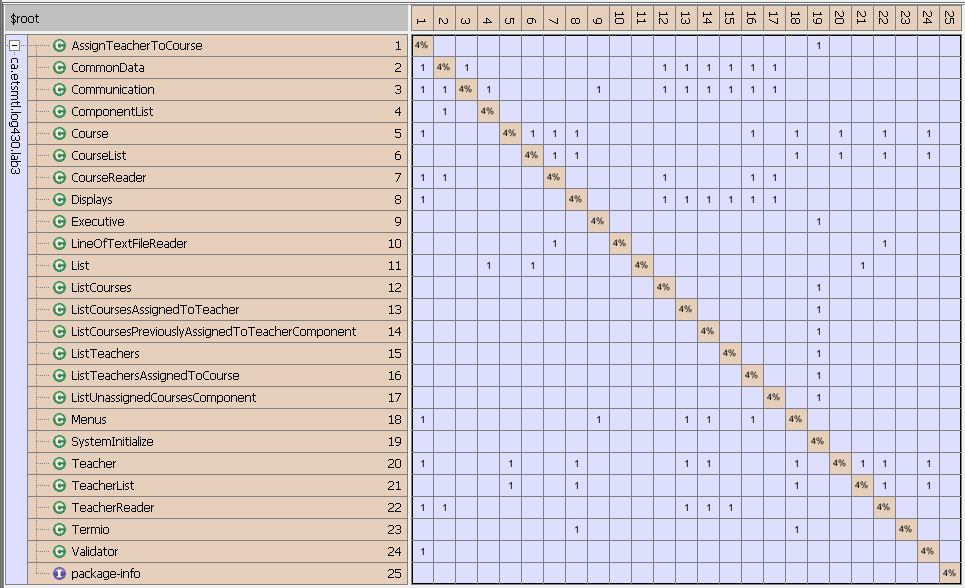
Lorsqu’une modification est détectée, un message est envoyé à tous les composants qui se sont enregistré à la liste des notifications. Selon la nature de l’événement, les composants impliqués lanceront les fonctions correspondantes à la logique d’affaire de l’application. Pour respecter l’architecture d’invocation implicite, toutes nos modifications au code existant devaient respecter le modèle du patron de conception « Observer ». Pour ce faire, nous avons créé de nouveaux composants qui vont s’inscrire à la liste des notifications de *«Executive »* afin d’implémenter les nouvelles fonctionnalités requises.

Les nouveaux composants « Observer » que nous avons ajoutés sont les classes « *ListCoursesPreviouslyAssignedToTeacherComponent* » et « *ListUnassignedCourseComponent*. » Ces deux classes sont inscrites pour recevoir les messages de la classe « *Executive »* et implémentent les modifications voulues.

c)

La matrice DSM du système original :



La matrice DSM du système modifié : 

Lorsque l’on compare la matrice DSM du système modifié avec la matrice DSM du système original, on peut constater quelques différences. Premièrement, on remarque la matrice du système modifié contient plus de lignes et de colonnes que celui du système original. Cela peut être expliqué par le fait que pour faire les modifications souhaitées, nous avons dus créer de nouvelles classes dans l’application ce qui augmente le nombre de composants.

Deuxièmement, il est intéressant de regarder les nouvelles relations créées par l’ajout de ces composants dans l’application. Comme on peut le constater, ces classes n’augmentent pas le nombre de relations de manière significative étant donné que les nouvelles relations sont faîtes avec les données communes du programme et avec la classe *Communication.*

Les liens sont donc sensiblement les mêmes quepour les autres composantes du système comme « ListTeacher » ou « ListCourse ». Les seules informations que les nouveaux composants ont besoins de connaître sont les données communes du programme et la logique événementielle de l’application.

d)

Les modifications apportées à ce système sont différentes de celles du laboratoire 2 puisque dans le laboratoire 2, l’ensemble des procédures pour chacune des options du menu sont directement appelé via le « main » de l’application et l’ensemble de la logique pour chaque choix de l’utilisateur est traité à cet endroit. Pour le laboratoire 3, ces appels aux procédures sont invoqués de façon implicite puisque l’application est structurée autour de la gestion des événements. C’est-à-dire qu’au lieu de déclencher la procédure directement, on annonce la diffusion d’un événement et les autres composants qui sont abonné à cet événement démarreront la procédure qui leur est associé. De ce fait, les modifications au laboratoire 3 consistaient plutôt à créer 2 nouveaux composants pour les 2 options a ajouter et à les ajouter à la liste des receveurs qui écouteront afin de recevoir le signal associé à ces options. La logique d’affaire est donc statiquement déterminé avant la compilation pour le laboratoire 2 tandis que dans le laboratoire 3, c’est au moment de l’exécution que le couplage entre les composants est déterminé.

Certain type de changements serait particulièrement simple à intégrer à un système à invocation implicite. En effet, si par exemple nous avons à remplacer un composant par un autre, il n’y aurait aucun effet secondaire sur les autres composants car ceux-ci sont indépendants les uns des autres. De plus, nous pouvons ajouter un composant simplement en l’enregistrant en tant qu’abonné d’un événement sans aucune autre modification aux interfaces des autres composants.

D’autre part, certain autre type de changement pourraient être plus difficile à intégrer, par exemple si un changement implique que l’on doit contrôler l’ordre des réponses à un événement. Les composants qui annoncent des événements n’ont aucun control sur l’ordre des réponses reçu par les composants qui écoute pour cet événement. De plus, il peut être difficile de comprendre et de maintenir le comportement d’un composant qui annonce un événement en étant indépendant des composants qui sont enregistré à cet événement.

# Discussion

Ce laboratoire porte sur les liens entre une architecture du style « Invocation implicite » et les implémentations orienté objet qui la compose. Dans un premier temps, nous avions à modifier une architecture existante afin de supporter quelques fonctionnalités supplémentaires tout en prenant soin de respecter ce style architectural. La deuxième partie consistait à analyser les interdépendances de cette architecture avant et après les modifications au système avec l’outil de création de matrice de dépendance Lattix LDM.

# Conclusion

Ce laboratoire avait pour objectif de nous aider à nous familiariser avec le style architectural « Invocation implicite » et de comprendre les différences fondamentales entre ce style et les autres que nous avons vu au cours des précédents laboratoires. Cet objectif a été atteint.

Nous nous questionnons toutefois sur une tâche qui a officieusement été identifiée comme obligatoire par le chargé de laboratoire : afficher les noms des cours lorsque l’on affiche les sigles pour les cours donnés dans des sessions antérieures de cette année. Il s’agit des listes de cours incluses dans le fichier de données des enseignants (**enseignantsLOG.txt**). L’application initiale ne faisait pas le lien entre ces sigles de cours et les titres des cours qui, eux, sont présents dans un autre fichier et un différent contexte d’utilisation. Les énoncés de laboratoire #2 et #3 ne mentionnent pas cette exigence et nous considérons qu’une telle modification n’est pas suffisamment triviale pour ne pas être officiellement incluse dans les énoncés.