Université Lille 1

Master IAGL

OPL/IDL

Générateur de diagrammes UML

Auteurs:
Slimane El Ouadi
Sofiane Yousfi

Superviseur:
Dr. Martin Monperrus

9 Février 2016

Sommaire

Introduction

Lors de la réalisation de logiciel, il y a différentes phases que l'on peut classer dans différents domaines : l'analyse fonctionnelle, l'architecture, la programmation, les tests, la validation, la maintenance. L'architecture joue un rôle centrale pour le bon fonctionnement, la lisibilité et la facilité de maintenance du code. Il y a plusieurs manières d'architecturer son code mais également plusieurs manières de représenter cette architecture.

Il y a notamment le diagramme de classes et le diagramme de dépendances des paquetages. Notre travail ici a été de générer ces deux diagrammes pour plusieurs projets via une analyse de code source et le passage des données collectées à une librairie javascript qui se charge du rendu graphique.

Chapitre 1

Principes et outils

1.1 Diagramme UML

UML(*Unified Modeling Language*) est un langage de modélisation graphique conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est souvent utilisé en développement logiciel et en conception orientée objet.
Il existe plusieurs diagramme UML dépendants hiérarchiquement et qui se complètent, de façon à permettre la modélisation d'un projet tout au long de son cycle de vie. Notre travail se focalise sur deux d'entre eux, le diagramme des classes et le diagramme des paquetages.

1.1.1 Diagramme des classes

Le diagramme des classes est un diagramme qui représente les classes intervenantes dans le système d'un logiciel, il permet de représenter chaque classe avec ses attributs, ses méthodes, sa visibilité et ses relations avec d'autres classes

1.1.2 Diagramme des paquetages (dépendances)

Les diagrammes de paquetages sont la représentation graphique des relations existant entre les paquetages (ou espaces de noms) composant un système[2].
Ce diagramme représente donc les packages d'un logiciel et les relations entre eux.

1.2 Spoon[1]

SPOON est une librairie développée par l'équipe SPIRALS[3], elle permet d'analyser et de transformer du code java, elle permet notamment aux développeurs de modifier, insérer ou supprimer des parties du code source avec une grande simplicité grâce aux processeurs.

1.3 GoJs

GoJS est une bibliothèque JavaScript qui permet la mise en œuvre de diagrammes interactifs sur les navigateurs web. Cette bibliothèque permet de construire des diagrammes et des nœuds complexes, des liens et des groupes avec des modèles que l'on peut personnaliser. GoJS est codé en JavaScript mais ne dépend pas d'autre bibliothèques ou Framework JavaScript.

GoJS offre de nombreuses fonctionnalités avancés pour l'interactivité telles que le drag-and-drop, copier-coller, des palettes, des modèles liés aux données, des gestionnaires d'événements, etc.

Pour notre travail nous avons donc choisi cette librairie car elle permet de construire des diagrammes à l'aide d'objets JSON.

Chapitre 2

Travail réalisé

2.1 But

Le but de ce projet est de réaliser un outil qui permet de construire le diagramme des classes et le diagramme des dépendances entre packages(*diagramme des paquetages*) d'un projet en utilisant un format JSON, ensuite, il les affiche sur un navigateur web. Finalement, nous évaluerons notre travail sur un projet open-source afin de valider les résultats.

2.2 Approche

Notre travail consiste à parcourir les classes d'un projet en utilisant SPOON afin de rassembler les information nécessaires pour construire soit le diagramme des classes ou le diagramme des paquetages. Ces information sont ensuite traduits en objets JSON. Un programme génère ensuite un fichier .js utilisant les objets JSON et qui se base sur la librairie GoJs afin de construire les diagrammes, et un autre fichier .html qui fait appelle au fichier .js .

Au lancement, le fichier .html est ouvert sur le navigateur web par défaut, et le diagramme s'affiche.

2.3 Implémentation

2.3.1 Processeurs

Diagramme des paquetages : Afin de construire le diagramme des dépendances entre les packages, nous avons créé deux processeurs, le premier parcours les classes d'un projet et stocke dans une liste les packages en utilisant la classe *CtPackage* de SPOON, le deuxième processeur parcours les classes et utilise la liste construite auparavant pour construire une map qui prend le package de la classe traitée comme clé et la liste des packages importés comme valeur. Les packages importés son déterminé à l'aide de la classe *CtTypeReference* de SPOON.

Diagramme de classes : Il y a plusieurs phases pour construire le diagramme de classes. Le

premier processeur va s'occuper de donner une clé (qui est un entier) à chaque classe parcourus dans le projet et de construire l'objet JSon pour la librairie en y intégrant les informations des attributs et des méthodes. Le second processeur va parcourir pour chaque classe ayant une clé ses super-classes, ses super-interfaces et ses attributs afin de créer le mapping avec les autres classes du projet. Une fois ce mapping créer nous pouvons tout d'abord enlever toutes les classes qui n'ont aucun lien avec les autres, elles n'aparaîtrons pas dans le diagramme final. Ensuite nous pouvons supprimer les attribus qui sont d'un type d'une autre classe du projet car cette relation est remplacé par une association sur le diagramme.

Dans le diagramme final, il y a les *extends*, *implements* et les associations simples et *1..** qui sont représentées selon les normes des diagrammes UML. Dans notre cas nous excluons uniquement les classes anonymes et les classes qui n'ont pas de liens avec les autres.

2.3.2 Objet JSON

Dans notre implémentation, les objets JSON sert à formater les informations nécessaires pour construire les diagrammes. Nous avons utilisé les classes *JSONArray* et *JSONObjet* comprise dans le package "org.json" afin de construire nos objets de façon à ce qu'on puisse les intégrer directement dans le fichier Js.
Voici un exemple d'objet JSON tel qu'on utilise dans notre programme :
Objets JSON spécifiant des éléments avec des clés :

```
[
      {"text": "class_1", "key": 1},
      {"text": "class_2", "key": 2},
      {"text": "class_3", "key": 3}
]
```

Objets JSON spécifiant des relations entre des éléments:

```
[
      {"from": "1", "to": 2},
      {"from": "1", "to": 3},
      {"from": "3", "to": 1}
]
```

2.3.3 Génération des fichiers Html et Js

Le fichier Js est générer par une classe java qui contient une suite de code javascript en dur, et qui prend en paramètres les objets JSON correspondants à la liste des éléments(*packages ou classes*) et les liens entre ces éléments.
Le fichier Js utilise la librairie GoJs pour construire les éléments du diagramme comme les classes et les flèches.
Le fichier Html est généré lui aussi par une méthode java de la même façon que le fichier Js. Il contient un espace ou le diagramme est représenté et il fait appelle au fichier Js.

Chapitre 3

Évaluation

3.1 Les projets de test

Afin d'évaluer notre outil, nous avons pris 3 projets open-source sur github, ces projet sont différents au niveau du nombre de packages et de classes. Il s'agit de Jsoup, Joda-time et Commons-math.

3.2 Analyse des résultats

3.2.1 Diagramme des classes

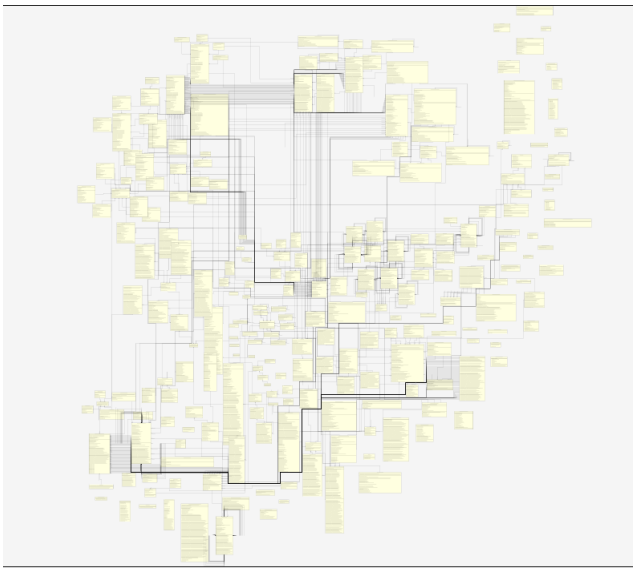

Figure 3.1: Diagramme des classes joda-time

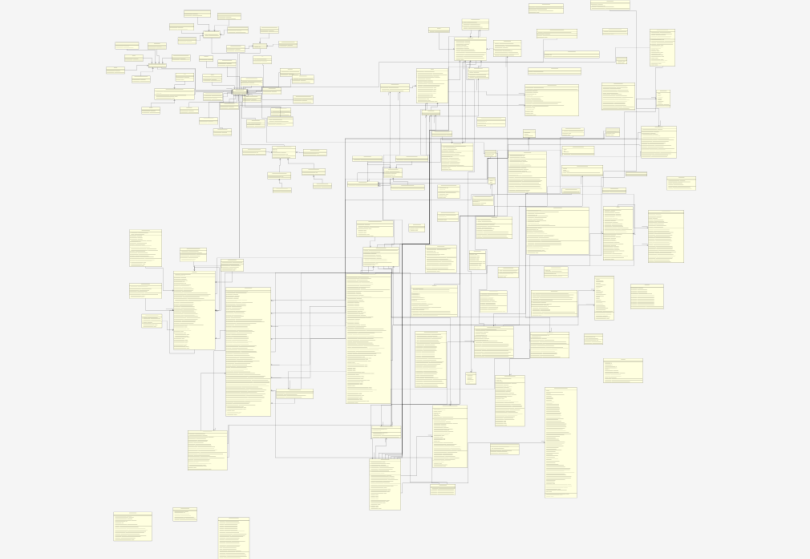

Figure 3.2: Diagramme des classes jsoup

3.2.2 Diagramme des paquetages

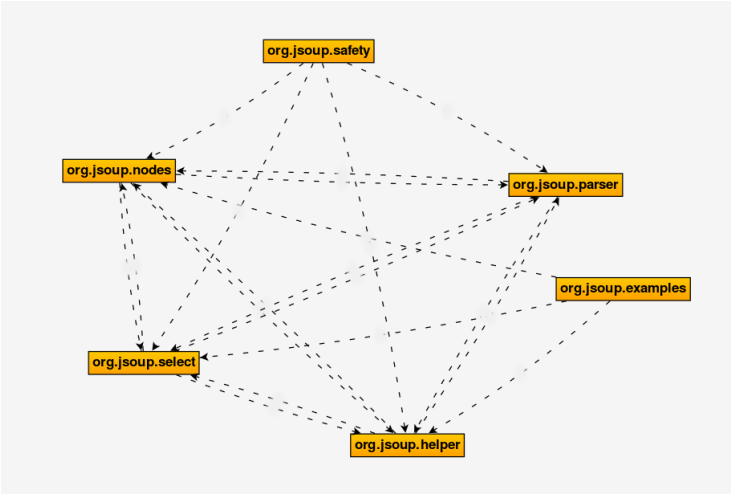

Figure 3.3: Diagramme des paquetage de Jsoup

3.3 Performances

Logiquement, lorsque nous sommes passés à des projets de plus en plus conséquent, nous avons eu le temps d'exécution qui a augmenter. Cela est logique car nors processeurs vont analyser toutes les classes et ensuite construire les fichiers JSON associés.

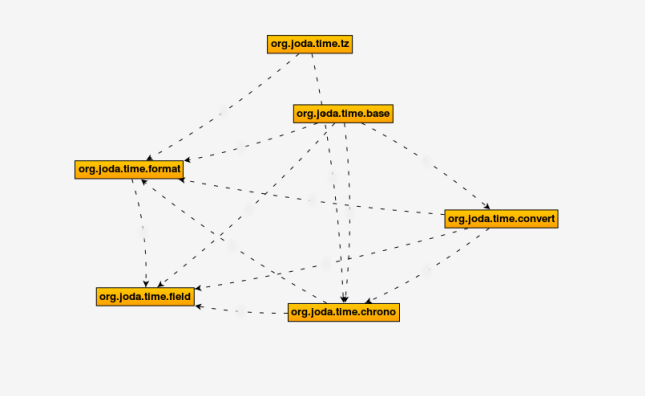

Figure 3.4: Diagramme des paquetage de Joda-time

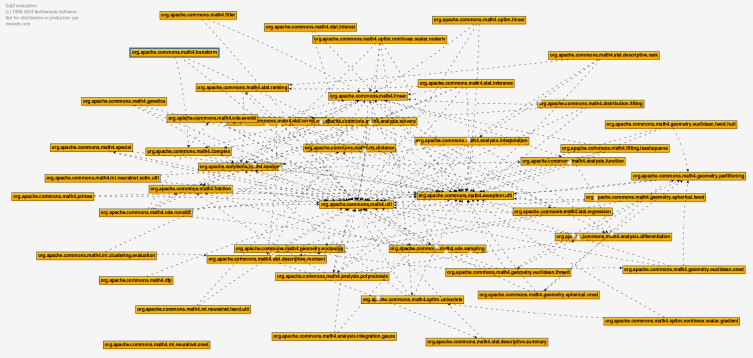

Figure 3.5: Diagramme des paquetage de Commons-math

3.4 Limitations

Nous remarquons clairement que lorsque l'on a un projet avec beaucoup de dépendances entre les paquetages ou entre les classes, nous obtenons un diagramme assez conséquent. Il est difficile de s'y repéré. Par ailleurs, il aurait été préférable d'intégrer cet outil à jenkins sous forme de plugin pour avoir une tâche complétement automatique.

Conclusion

Grâce à spoon, nous avons pu récupérer toutes les données nécessaires au bon fonctionnement de la librarie GoJs et donc à la visualisation des diagrammes. Concernant les petits projets, comme ce que nous faisons actuellement dans le cadre pédagogique, nous pensons que l'outil que nous avons développé nous sera utile. Par contre pour les projets que l'on rencontrera dans la vie professionnel, il est difficile de se repérer des diagrammes de la sorte. Il serait peut être judicieux de permettre à l'utilisateur de choisir une seule classe et d'afficher d'un clic toutes les classes avec lesquelles elle a un lien et l'intégrer à jenkins.

Bibliography

[1] Spoon, http://spoon.gforge.inria.fr/.

[2] Diagramme des paquetages, wikipédia.

[3] spirals, http://www.inria.fr/equipes/spirals.