Compromis entre performance et généricité : le cas du diff de modèles

Julien Delplanque[1,2] Anne Etien[2] Nicolas Anquetil[2]

[1]Université de Mons, Belgique
julien.delplanque@live.be

[2]Université de Lille, CNRS, Inria, Centrale Lille,
UMR 9189 - CRIStAL,
F-59000 Lille, France
{*nom.prenom*}*@univ-lille1.fr*

8 Décembre 2016

Plan

Scénario I - Vincent

- Développeur.
- Logiciel écrit en Java d'une taille conséquente.
- Grand nombre de tests unitaires.

Scénario I - Vincent

- Développeur.
- Logiciel écrit en Java d'une taille conséquente.
- Grand nombre de tests unitaires.

$\rightarrow$ Utiliser un diff pour cibler les tests à exécuter après une modification.

Scénario II - Olivier

- Administrateur de base de données.
- Base de données PostgreSQL de gestion du personnel.
- Accord avec un autre organisme pour qu'Olivier fournisse/maintienne le schéma.
- Avec le temps, les deux schémas évoluent indépendamment pour répondre aux besoins différents de deux organismes.

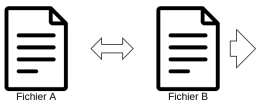
Scénario II - Olivier

- Administrateur de base de données.
- Base de données PostgreSQL de gestion du personnel.
- Accord avec un autre organisme pour qu'Olivier fournisse/maintienne le schéma.
- Avec le temps, les deux schémas évoluent indépendamment pour répondre aux besoins différents de deux organismes.

→ Utiliser un diff pour aider Olivier à cerner les différences entre les deux bases de données et à écrire ses "patches".

Plan

Différence Textuelle



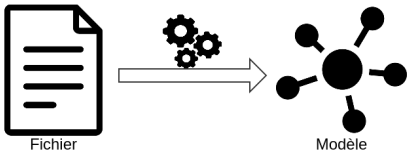
- Ajouts de lignes
- Suppressions de lignes

```
import edu.ntnu.texasai.controller.PokerController;
import edu.ntnu.texasai.dependencyinjection.TexasModule;

-public class Main {
    public static void main(String[] args) {
        Injector injector = Guice.createInjector(new TexasModule());
```

```
import edu.ntnu.texasai.controller.PokerController;
import edu.ntnu.texasai.dependencyinjection.TexasModule;

+public class Play {
    public static void main(String[] args) {
        Injector injector = Guice.createInjector(new TexasModule());
```

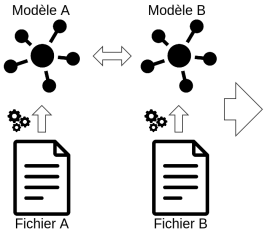
Modélisation

Un modèle est une représentation abstraite et subjective du logiciel.



Exemple : Dans le cas de code Java, on va manipuler des packages, des classes, des méthodes, des invocations, etc. . .

Différence de modèles



Résultats :

- Ajouts d'entités/de relations entre entités.
- Suppressions d'entités/de relations entre entités.
- Renommages d'entités.
- Déplacements d'entités.

Plan

Approche générale

Soit M_A et M_B les modèles à différencier,

1. Faire correspondre les entités de M_A avec celles du M_B ("matching").
   - Changements de version.
   - Renommages.
   - Déplacements.
2. Détecter les entités qui ont été créées (ajouts).
3. Détecter les entités qui ont été supprimées (suppressions).

FAMIXDiff

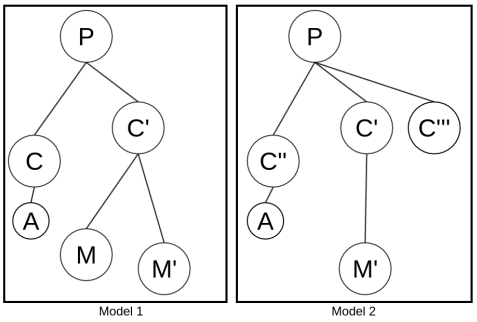


FIGURE – Différence réalisée par FAMIXDiff.

FAMIXDiff

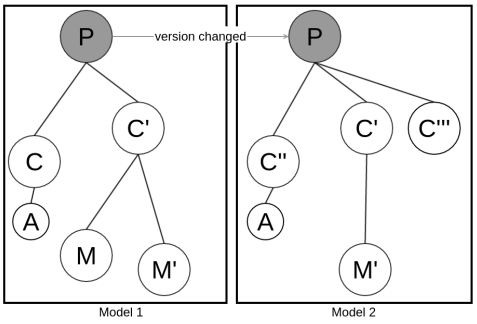


FIGURE – Différence réalisée par FAMIXDiff.

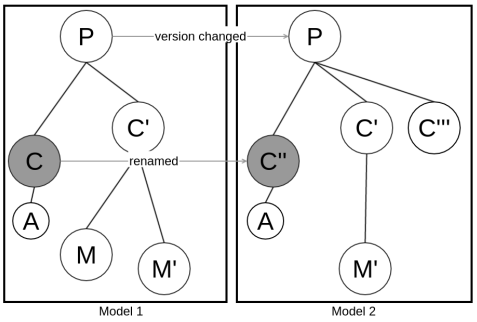


FIGURE – Différence réalisée par FAMIXDiff.

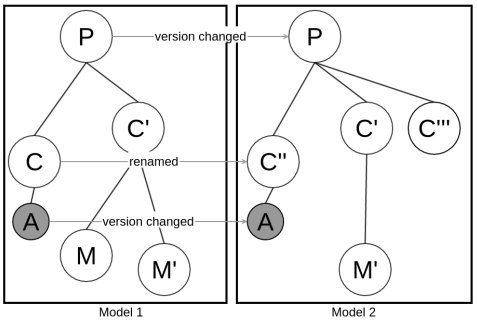


FIGURE – Différence réalisée par FAMIXDiff.

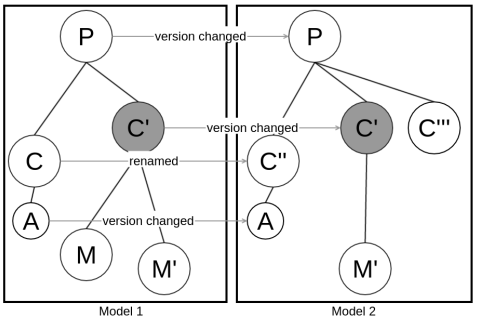


FIGURE – Différence réalisée par FAMIXDiff.

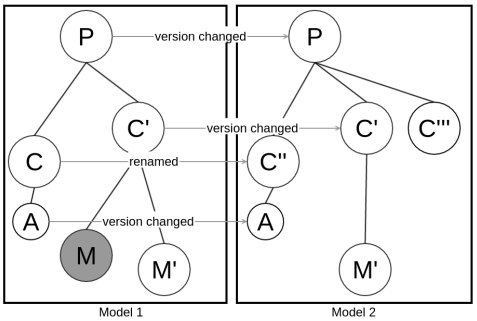


FIGURE – Différence réalisée par FAMIXDiff.

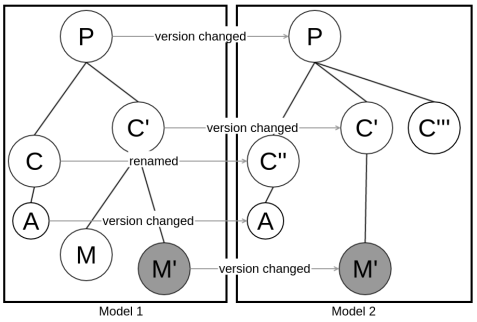


FIGURE – Différence réalisée par FAMIXDiff.

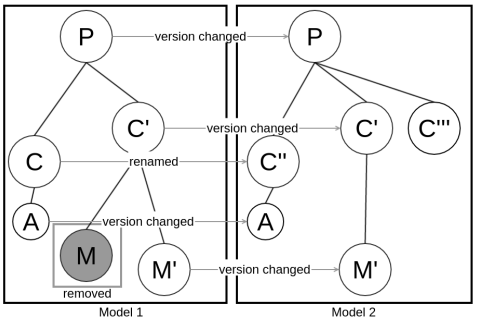


FIGURE – Différence réalisée par FAMIXDiff.

FAMIXDiff

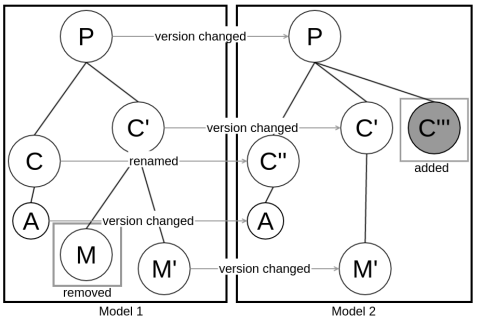


FIGURE – Différence réalisée par FAMIXDiff.

FAMIXDiff

- Approche basée sur UMLDiff [2].
- Suit les liens de compositions pour faire se correspondre les entités des deux modèles.
- Spécialisé dans la différence de modèles représentant des logiciels OO.

Limitations

Revenons aux scénarios :

- Vincent utilise du Java, il peut donc utiliser FAMIXDiff.
- Olivier utilise SQL, il est possible d'adapter FAMIXDiff pour qu'il prenne en charge SQL...
- Brice arrive et a besoin de calculer la différence entre deux versions d'un logiciel C...

Limitations

Revenons aux scénarios :

- Vincent utilise du Java, il peut donc utiliser FAMIXDiff.
- Olivier utilise SQL, il est possible d'adapter FAMIXDiff pour qu'il prenne en charge SQL...
- Brice arrive et a besoin de calculer la différence entre deux versions d'un logiciel C...

→ **Besoin d'une solution générique...**

Similarity Flooding

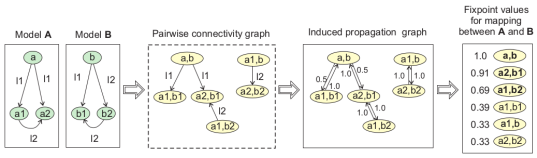


FIGURE – Matching réalisé par Similarity Flooding [1].

Similarity Flooding

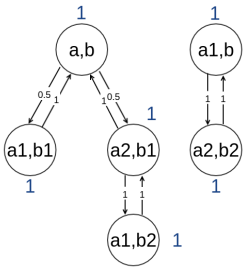


FIGURE – Itération sur l'"induced propagation graph".

Similarity Flooding

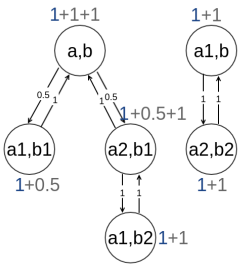


FIGURE – Itération sur l'"induced propagation graph".

Similarity Flooding

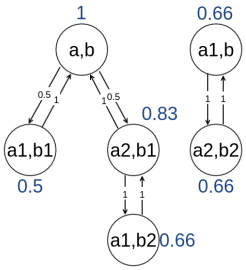


FIGURE – Itération sur l'"induced propagation graph".

Similarity Flooding

- Algorithme générique de “matching” de graphes.
- Basé sur la structure des graphes à faire correspondre.
- Retourne une liste des “matching” possibles associés à des valeurs de similarité.

Limitations

- Gourmand en temps de calcul quand les graphes sont denses.
- Ne fait pas de différence entre les types d’entités contenues dans les noeuds des graphes des modèles.
- Tous les types de liens sont considérés de la même façon.

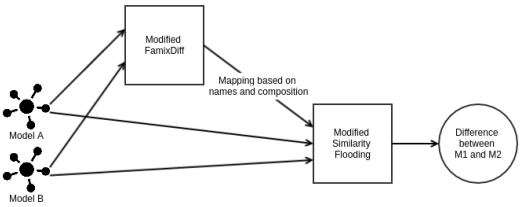
Plan

SFDiff

Approche hybride issue d'une combinaison de FAMIXDiff et Similarity Flooding (SF) :

1. Utilise le principe de FAMIXDiff afin de réduire la taille des graphes en entrée de Similarity Flooding.
2. Utilise une version modifiée de SF qui :
   - Tient compte des "matching" découverts à l'étape 1.
   - N'essaie pas de mettre en correspondance des entités de types différents.
   - Donne plus de poids à la relation de composition.

SFDiff



Première évaluation

Projet	# classes	# changements attendus
LibnotifyBinding (LB)	4	24
Renraku	28	44
RenoirSt	71	100
SmaCC	660	3

Premier résultats

SFDiff :

Projet	Temps (s)	# Changements	Précision	Recall
LB	0.029	21	0.85	0.75
Renraku	0.255	92	0.43	0.9
RenoirSt	0.501	103	0.97	1.0
SmaCC	82.9	36	0.08	1.0

FAMIXDiff :

Projet	Temps (s)	# Changements	Précision	Recall
LB	0.034	23	0.86	0.83
Renraku	0.165	80	0.51	0.93
RenoirSt	0.448	103	0.97	1.0
SmaCC	36.352	4	0.75	1.0

Conclusion

- Plus générique que FAMIXDiff.
- Plus rapide que Similarity Flooding lorsque les modèles sont suffisamment semblables.
- Requiert des modèles dans lesquels la relation de composition existe.

Perspectives

- Régler le problème de précision de la solution proposée.
- Évaluer de façon plus poussée l'algorithme.
- Essayer d'optimiser l'algorithme au niveau du temps de calcul.

Références I

Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm.
Similarity flooding : A versatile graph matching algorithm and its application to schema matching.
In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 117–128. IEEE, 2002.

Zhenchang Xing and Eleni Stroulia.
Umldiff : an algorithm for object-oriented design differencing.
In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 54–65. ACM, 2005.