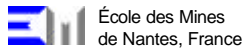


Un méta-modèle pour coupler application et détection des design patterns

Hervé Albin-Amiot
Pierre Cointe
Yann-Gaël Guéhéneuc
{albin, cointe, guehene}@emn.fr





Plan

- Le contexte
- Notre objectif
- Nos besoins
- Travaux connexes
- Notre solution : le méta-modèle PDL
- PDL : principe & mise en œuvre
- Questions ouvertes



Le contexte

Améliorer la qualité des développements par objets en favorisant l'utilisation des **design patterns** pour aider à :

- Implémenter
- Comprendre
- Documenter

3/17

Notre but est d'améliorer...

Les design patterns sont bien connus et éprouvés dans la recherche académique comme dans l'industrie.

Les design patterns se situent à mi-chemin entre le code source et la conception (micro-architectures).

Dans le reste de cette présentation, le GoF (Gamma et al.) est notre référence.



Notre objectif

- Fournir un assistant logiciel aidant à :
 - L'implémentation
 - **Sélection**
 - Paramétrage
 - Génération
- } **Application**
- La compréhension et la documentation :
détection de **formes exactes**

Deux phases principales pour l'implémentation :

- la sélection... ;
- l'application... Qui se décompose elle-même en :
 - paramétrage... ;
 - génération...

Une formes exacte = un design pattern tel que décrit dans le GoF. Une forme exacte suit à la fois la structure du design pattern et aussi son intention, ses collaborations, ... telles que décrites dans le GoF.

En opposition avec une forme dégradée dans laquelle certaines caractéristiques (rôles, relations, ...) ne sont pas vérifiées. Par exemple, dans le design pattern Composite, peut-être la relation d'héritage entre Component et Composite n'est-elle pas vitale pour satisfaire l'intention du design pattern.



Nos besoins

- Formaliser les design patterns
- Décrire les design patterns en tant que :
 - Entités de « **première classe** »
 - Elles savent comment produire du code
 - Elles savent identifier leurs propres occurrences dans du code
 - Entités **manipulables**
 - Sur lesquelles on peut raisonner
 - Que l'on peut adapter à un contexte spécifique
 - Entités avec « **traces** »

Le GoF décrit 23 design patterns, mais ne les formalisent pas.

Nous devons formaliser pour pouvoir manipuler avec des outils ⇒ Décrire les design patterns pour...

- comme des entités de première classes
- comme des entités manipulables

On veut obtenir des entités de premières classes : structure plus une touche de comportement (commun à tous les design patterns et propre à un certain design pattern). Il faut bien voir qu'un design pattern ce n'est pas seulement une structure, c'est aussi un comportement.

On peut comparer cette approche avec la notion de *classe* dans un langage à classes.

L'idée de modéliser les design patterns pour l'application et la détection permet d'apporter une réponse au problème de la trace...



Travaux connexes

- Formalisation :
 - LePUS [Eden 2000]
 - Fragment Model [Florjin *et al.* 1997]
- Application
 - [Budinsky *et al.* 1996]
- Détection
 - [Wuyts 1998], [Antoniol *et al.* 1998]

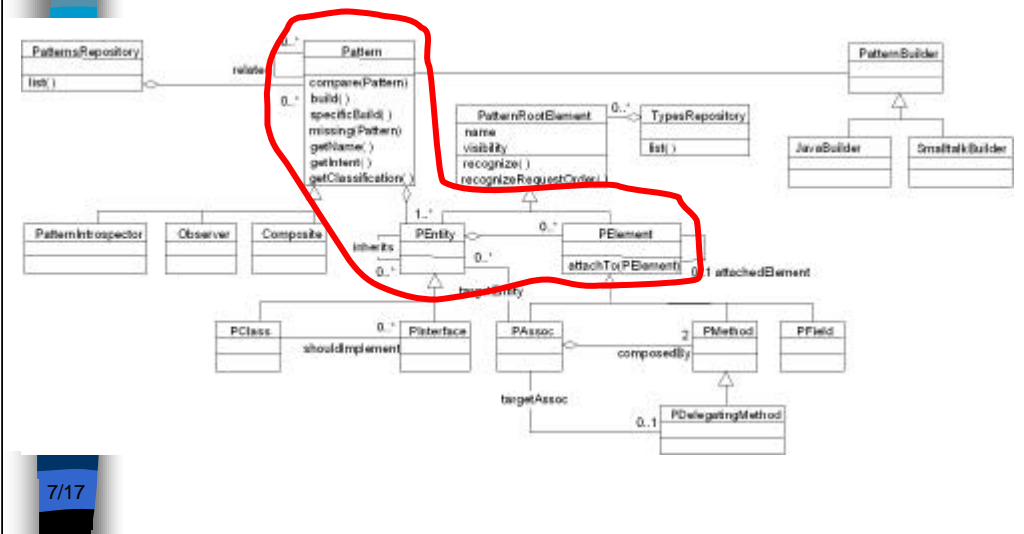
Nos travaux sont à la frontière de différents travaux sur les design patterns : positionnement difficile car on veut faire trois aspects...

Dans notre approche, nous cherchons à formaliser les design patterns pour appliquer et détecter les design patterns.

La plupart des travaux connexes ne s'intéressent qu'à un seul aspect:

- formalisation :
 - LePus : haut niveau d'abstraction de la formalisation, peu ou pas de génération de code, pas de prise en compte des idiomes de programmation propre aux langages ;
 - Modèle à fragments : granularité très (voire trop) fine, axé sur les caractéristique du langage de programmation Smalltalk ;
- application :
 - Budinsky propose un catalogue interactif en HTML des design patterns en vue de la génération de code d'exemple, mais il n'y a pas de possibilité de paramétrer le code générer pour l'insérer directement dans le code de l'utilisateur, de plus la détection n'est pas prise en compte ;
- détection :
 - Wuyts et Antoniol ne représentent que les design patterns structuraux, ils ne proposent pas de réelle formalisation, ni de détection.

Notre solution : le méta-modèle PDL



7/17

On utilise un méta-modèle pour formaliser les design patterns.

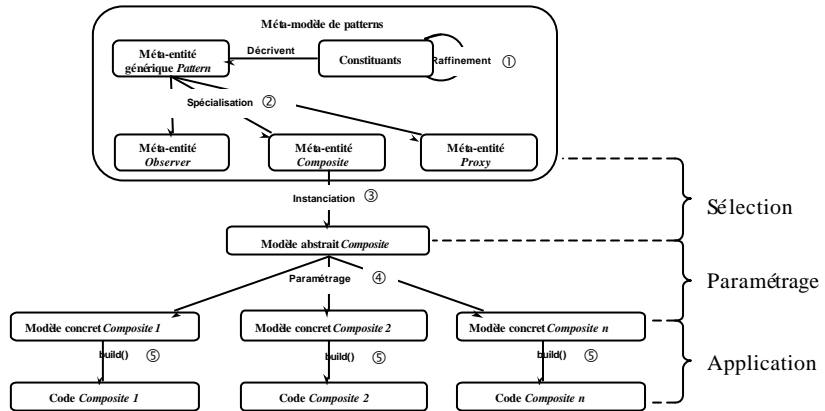
La technique de méta-modélisation est bien connue pour...

Un méta-modèle pour formaliser/décrire les design patterns...

Ce méta-modèle est un compromis entre les constituants pour décrire les design patterns et pour décrire les langages de programmation cible, car on veut faire à la fois de la formalisation, de l'application, et de la détection.

Le méta-modèle est très simple pour décrire les design patterns et le langage Java, il peut/doit être adapté pour la génération d'autres langages.

PDL : principe

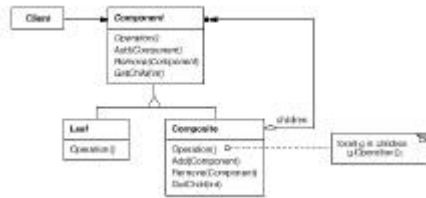


Les différentes étapes d'utilisation de notre méta-modèle, depuis la définition des constituants, jusqu'à l'application.

(Voir l'article dans les actes, ou www.yann-gael.gueheneuc.net/Work/Publications/)

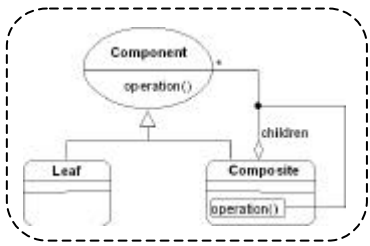
PDL : mise en œuvre

Description



➕ Descriptions informelles du [GoF]

↓ Traduction en un modèle de pattern

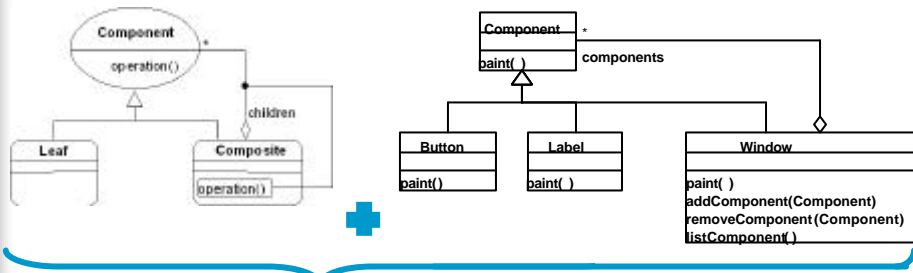


Légende	
	Instance de Pattern
	Instance de PInterface
	Instance de PClass
	Instance de PAssoc
	Instance de PDelegation
	Instance de PMethod

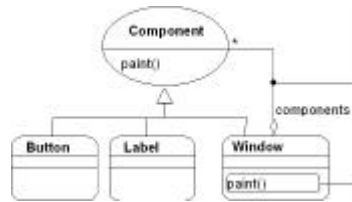
Exemple...

Et on a un catalogue d'autres design patterns...

PDL : mise en œuvre Application



Paramétrage

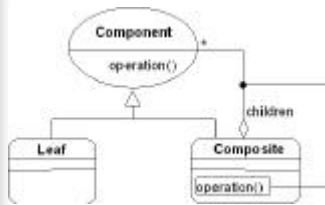


Génération

Code source
Java, Claire

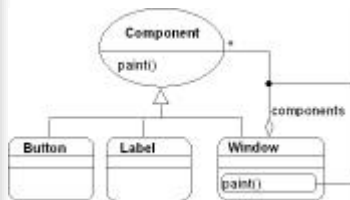
PDL : mise en œuvre

Détection



Design pattern
détecté ?

✚ Comparaison
(arc consistance + algorithmes dédiés)



Reconstruction

Code source Java

Source ou code octal (*byte-codes*)...

Représentation du code source à l'aide des constituants du méta-modèle...

Chaque design pattern est une entité de première classe et donc on peut lui demander de se comparer avec quelque chose et donc dire si ils sont équivalents...

L'algorithme utilisé est AC-*x* + des algorithmes ad-hoc qui permettent d'augmenter l'efficacité et la pertinence pour prendre en compte les idiomes de programmation.

L'algorithme est implémenté au niveau de la classe `Pattern`, et chaque design pattern peut être étendu...



Quelques résultats expérimentaux

- Application
 - Ptidej
- Détection sur plusieurs frameworks :
 - java.awt.*
 - JHotDraw
 - JUnit
 - PatternsBox
 - Ptidej

C'est difficile de présenter des exemples d'application, car il est plus facile d'utiliser l'outil...

Pour la détection, c'est plus facile de donner des résultats concrets...

Détection

Design patterns	Frameworks	NOC	Existing	Détection dans PatternsBox			Time (sec.)
				Hits	Missed	False hits	
Composite	java.awt.*	121	1	1			1
	JHotDraw	155	1	1			0,3
	JUnit	34	1		1		0,4
	JEdit	248					1,4
	PatternsBox	52					0,3
Decorator	java.awt.*	121					0,2
	JHotDraw	155	1	3		2	0,4
	JUnit	34	1	1			0,2
	JEdit	248					0,4
	PatternsBox	52					0,3
Factory Method	java.awt.*	121	3				1,1
	JHotDraw	155	2	2	1	1	0,5
	JUnit	34					
	JEdit	248					0,1
	PatternsBox	52					0,2
Iterator	java.awt.*	121					3,4
	JHotDraw	155	3	3			0,1
	JUnit	34					
	JEdit	248	1	1			0,1
	PatternsBox	52					
Observer	java.awt.*	121					3,4
	JHotDraw	155	2	2			3,2
	JUnit	34	4	4			2,5
	JEdit	248	3	3			1,3
	PatternsBox	52	1	1			2,8
Singleton	java.awt.*	121	3	3			0,7
	JHotDraw	155	2	2			0,5
	JUnit	34					0,4
	JEdit	248					1
	PatternsBox	52	1	1			0,5
		610	29	28	2	3	5,8

Nous avons appliqué six design patterns très utilisés (nous pensons) sur cinq *frameworks*. Les résultats sont donnés sur la dernière ligne...

Nous avons aussi appliqué nos algorithmes à tout les paquetages de Java (`java.*`) et les temps sont restés linéaires.

Mais certains design pattern sont difficilement détectables, comme le design pattern *Façade*.



Réalisation de l'objectif

- Décrire, appliquer et détecter les design patterns
 - Entités de première classe manipulables
- Nous avons un assistant aidant :
 - L'implémentation
 - La détection et la documentation de formes exactes de design patterns(Cet assistant est « relativement » simple)

14/17

Par rapport à notre objectif de départ, notre méta-modèle apporte une solution satisfaisante...

- Par rapport au contexte :

Améliorer la qualité des développements par objets en favorisant l'utilisation des design patterns

Mais au cours de nos recherches, nous avons rencontrés de nombreux problèmes pour lesquels nous n'avons pas de réponse définitive, encore ...

- Formalisation :
 - Un unique méta-modèle est-il suffisant ?
 - Comment représenter les compromis ?
- Application :
 - Comment bien décrire le code source ?
 - Est-ce qu'elle est toujours possible et intéressante ?

16/17

Un unique méta-modèle : Nous pensons que non, nous cherchons à modéliser l'intention des design patterns...

Compromis : Pour l'instant, seulement des solutions ad-hoc...

Source code : FAMIX + définitions sémantiquement opérationnelles des relations d'association, d'agrégation, et de composition...

Possible/intéressante : avec le design patterns Façade, l'utilisateur perd plus de temps à paramétrer qu'à écrire le code lui-même... Mais c'est quand même utile pour la validation, la détection, ...

■ Compréhension :

- Est-il possible de détecter les formes exactes de n'importe quel design pattern ?
- Est-il possible de détecter les formes approchées de n'importe quel design pattern ?

17/17

Formes exactes : Non, mais cela reste une question ouverte...

Formes dégradées : Difficile, quel est le seuil de tolérance entre une forme dégradée et quelque chose qui n'est pas un design pattern ?...