

Une taxonomie et un métamodèle pour la détection des défauts de conception

Naouel Moha – Duc-loc Huynh – Yann-Gaël Guéhéneuc

GEODES – Groupe de recherche sur les systèmes
ouverts et distribués et en génie logiciel expérimental

Université de Montréal

{mohanaou, huynhd, guehene}@iro.umontreal.ca

LMO (mars 2006) – LIFL (mai 2006)

Plan

Objectif de notre travail

Modéliser les défauts de conception afin de pouvoir les détecter automatiquement dans le code

- 1 Introduction
- 2 Méthodologie (5 phases)
- 3 Phases 1 et 2 : analyse et taxonomie
- 4 Phase 3 : métamodélisation
- 5 Phases 4 et 5 : modélisation et validation
- 6 Conclusion

Les défauts de conception

Défauts de conception vs. Patrons de conception

“Mauvaises” solutions à des problèmes récurrents dans les architectures à objets

Problèmes

- Pas de représentation précise et structurée
- Sous forme de descriptions textuelles sujettes à interprétation
- Difficile de les détecter précisément et efficacement

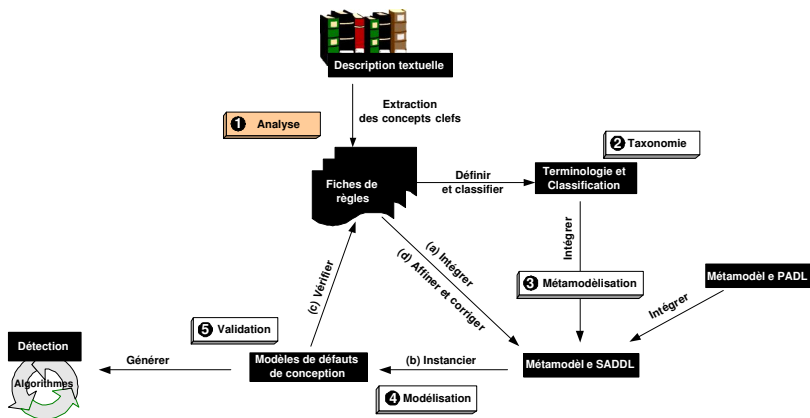
Solution

Représentation précise et structurée des défauts de conception basée sur un **métamodèle** à partir d'une **taxonomie** des défauts

Motivations

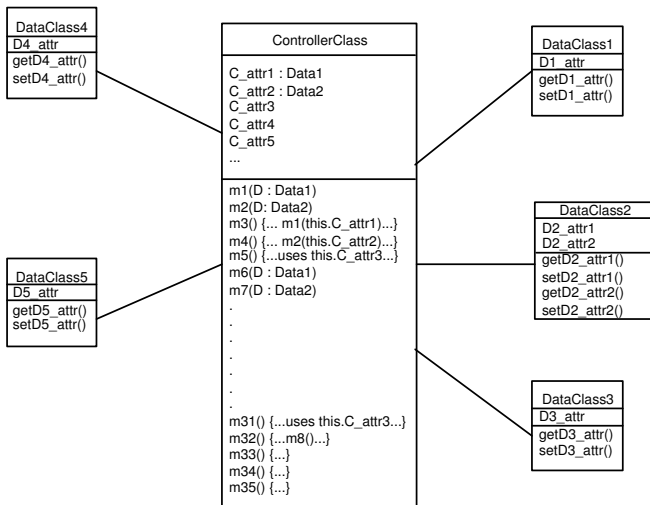
- Pourquoi détecter les défauts de conception ?
 - Améliorer la qualité des architectures à objets
 - Faciliter l'évolution
 - Réduire les coûts de maintenance
- Pourquoi une taxonomie ?
 - Confusion entre : défauts, anti-patterns, problèmes, mauvaises odeurs, anomalies, etc.
 - Définir, comparer et classer les défauts
- Pourquoi un métamodèle ?
 - Limite les ambiguïtés en structurant les concepts
 - Facilite le développement d'algorithmes et d'outils
- Pourquoi une méthodologie ?
 - Définir une démarche systématique à appliquer pour la représentation des défauts
 - Peut être appliquée à d'autres spécifications

Méthodologie en 5 phases



Méthodologie en 5 phases

Blob



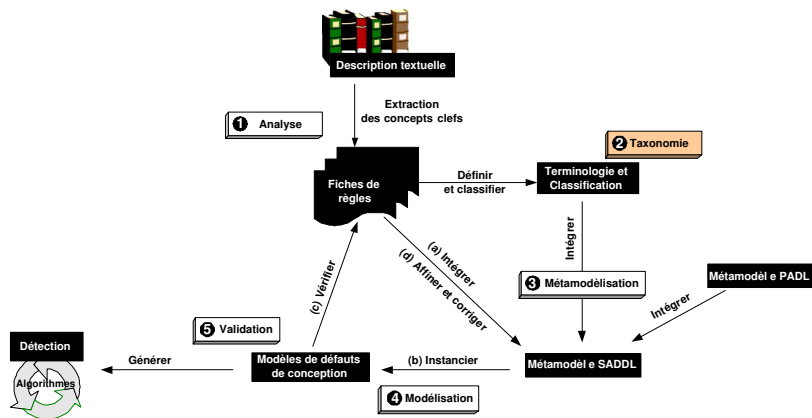
Méthodologie en 5 phases

```
RULE_CARD : Blob {  
  RULE : Blob {ASSOC: associated FROM: ControllerClass ONE TO: DataClass MANY } ;  
  RULE : ControllerClass {INTER LargeClassLowCohesion ClassName } ;  
  RULE : LargeClassLowCohesion {INTER LargeClass ClassLowCohesion } ;  
  RULE : LargeClass {INTER (METRIC: NM, HIGH) (METRIC: NA, HIGH) } ;  
  RULE : ClassLowCohesion {(METRIC: LCOM, HIGH) } ;  
  RULE : ClassName {(SEMANTIC: CLASSNAME, {System, Subsystem, Manager, Driver, Controller}) } ;  
  RULE : DataClass {INTER (STRUCT: METHOD, Accessor) (METRIC: LCOM, LOW) } ;  
};
```

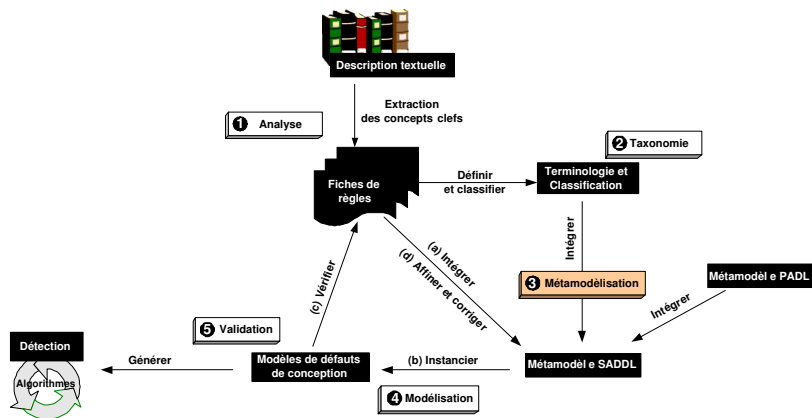
Phase 1 : analyse

- Description littéraire → Synthétique

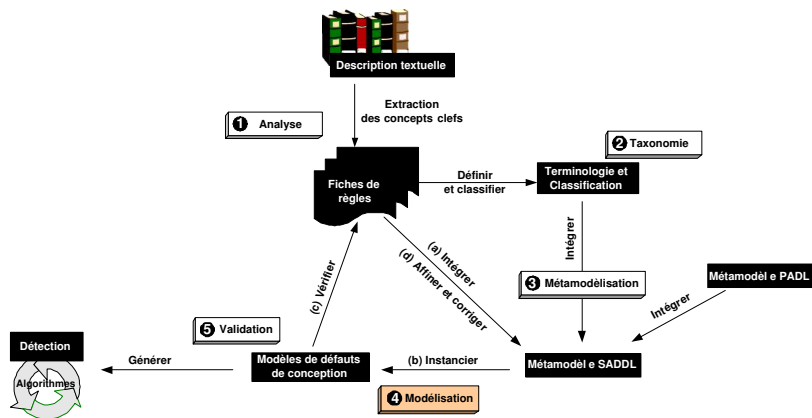
Méthodologie en 5 phases



Méthodologie en 5 phases



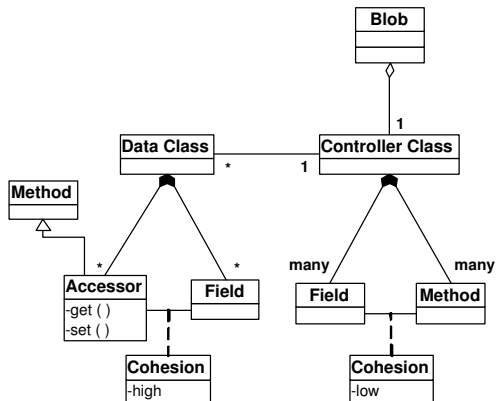
Méthodologie en 5 phases



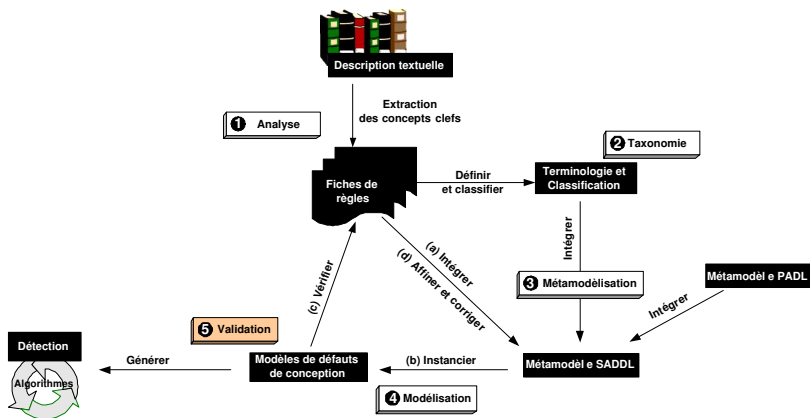
Méthodologie en 5 phases

Phase 4 : modélisation

- Modèle obtenu pour le Blob



Méthodologie en 5 phases



Phases 1 et 2 : analyse et taxonomie

Phase 1 : Analyse

- Extraction des concepts clefs
- Spécification des fiches de règles

Phase 2 : Taxonomie

- Terminologie et classification
 - Les défauts de patrons
 - Les anti-patrons
 - Les symptômes
 - Carte des défauts

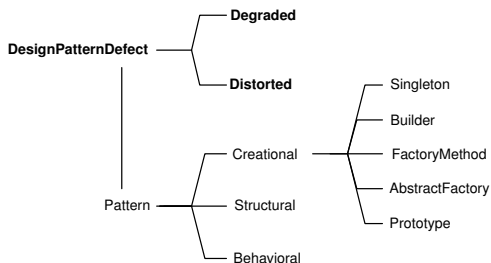
Les défauts de patrons

Terminologie

- Les défauts de patrons sont de mauvaises applications des solutions des patrons de conception [MOH 05c]

Classification

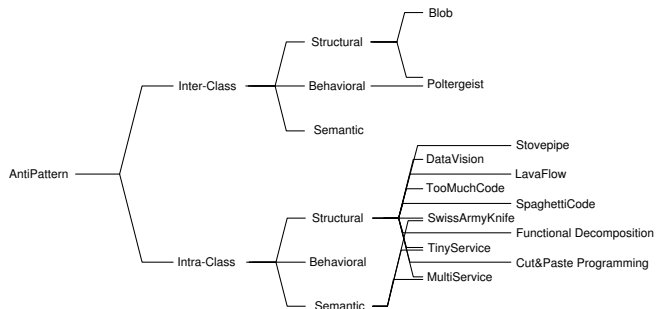
- 2 types :
 - Les patrons déformés
 - Les patrons dégradés
- Basée sur la classification de Gamma *et al.*



Les anti-patterns [BRO 98]

Terminologie

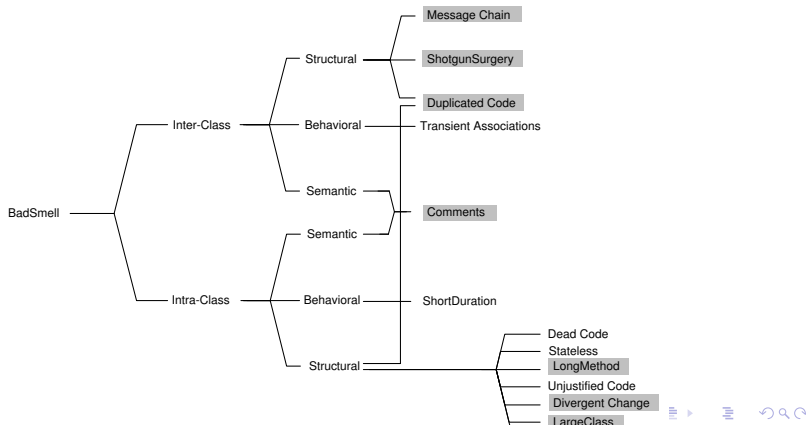
- “Mauvaises” solutions à des problèmes récurrents dans les architectures à objets, dont l’utilisation a des effets négatifs sur la qualité



Les symptômes ou *mauvaises odeurs*

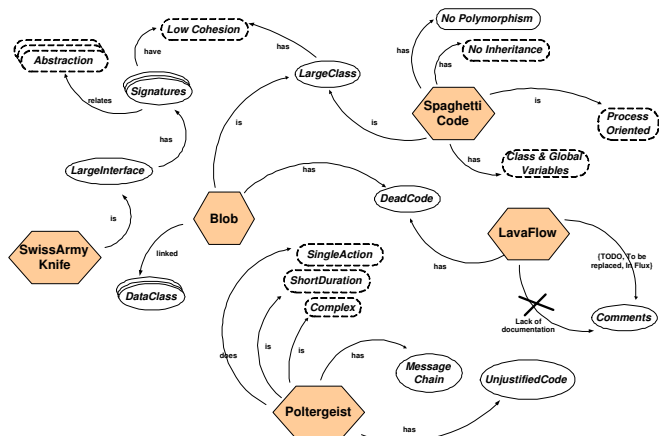
Terminologie

- Indicateurs de la présence possible de défauts
- *Structures dans le code source qui suggèrent la possibilité d'une restructuration du code* [FOW 99]
- Exemples : code dupliqué, larges classes & longues méthodes



Carte des défauts de conception

Relations entre les défauts de conception



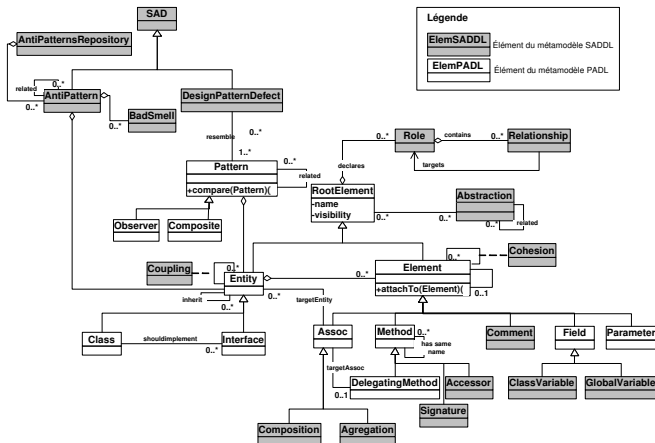
Phase 3 : Métamodélisation

PADL (*Pattern and Abstract-level Description Language*)

- Décrit la structure des programmes OO et les patrons de conception [ALB 02]

SADDL étend PADL (*Software Architectural Defects Description Language*)

- Décrit les défauts de conception



Phases 4-5 : Modélisation et validation

Modélisation

- Instancier des défauts de conception à partir de SADDL sous forme de modèles
- Quinzaine de défauts dont une dizaine d'anti-patterns

Validation

- Modèles ↔ Fiches
- Corriger et enrichir le métamodèle
- Générer les algorithmes de détection

Les défauts de patron

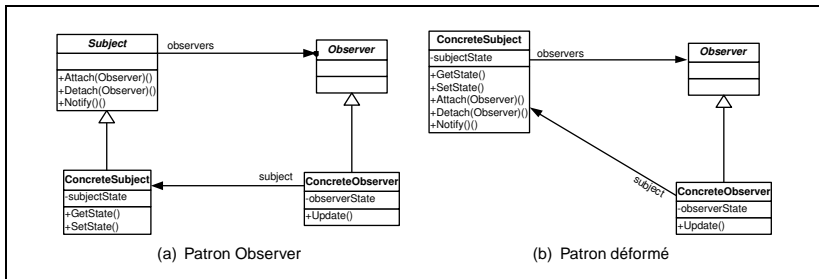


FIG.: Le patron Observer et son patron déformé

Les anti-patterns

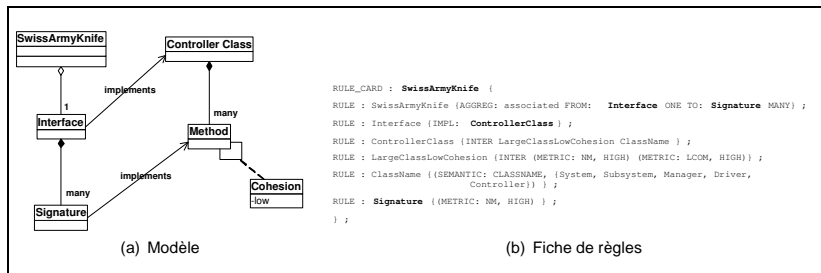


FIG.: Le couteau suisse

Les symptômes

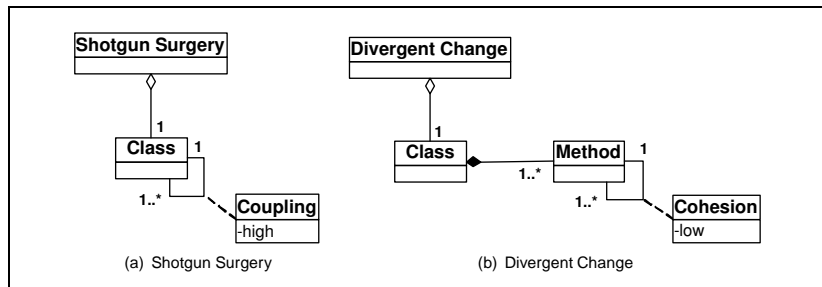
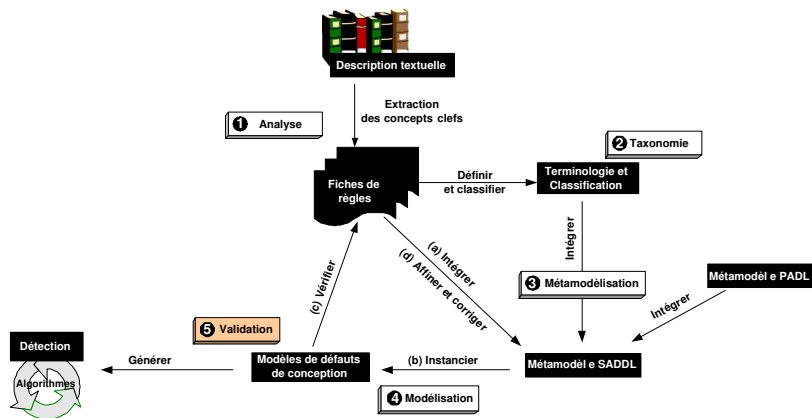


FIG.: Les modèles du Shotgun Surgery et du Divergent Change

Méthodologie en 5 phases



Expérimentations

Objectif

Montrer l'utilité de notre méthodologie et de son implantation (métamodèle SADDL)

Hypothèses

- Possibilité de spécifier un grand nombre de défauts de conception
- La précision de nos algos est raisonnable

Objets

5 programmes open source entre 140 et 1200 classes.

Traitement

- 4 anti-patterns : Blob, Functional decomposition, Swiss Army Knife, Spaghetti Code
- Implémentation des algorithmes de détection
- Validation manuelle : Précision vs. Rappel

Détection des défauts de conception

Blob						
	ArgoUML 113KLOC 1230c	Azureus 192KLOC 1449c	GanttProject 21KLOC 188c	PMD 42KLOC 423c	QuickUML 9KLOC 142c	Total
Effectifs	91	143	19	15	3	271
Precisions	70/91 = 76.9%	82/143 = 57.3%	10/19 = 52.6%	3/15 = 20%	1/3 = 33.3%	166/271 61.5%
Swiss Army Knife						
	ArgoUML	Azureus	GanttProject	PMD	QuickUML	Total
Effectifs	2	22	0	0	0	24
Precisions	2/2 = 100%	22/22 = 100%	- = -%	- = -%	- = -%	24/24 = 100%
Functional Decomposition						
	ArgoUML	Azureus	GanttProject	PMD	QuickUML	Total
Effectifs	14	22	9	13	1	59
Precisions	9/14 = 64.2%	19/22 = 86.3%	0 = 0%	0 = 0%	0 = 0%	28/59 = 47.6%
Spaghetti Code						
	ArgoUML	Azureus	GanttProject	PMD	QuickUML	Total
Effectifs	26	35	8	11	1	81
Precisions	21/26 = 80.7%	29/35 = 82.8%	6 = 75%	6 = 54.5%	0 = 0%	62/81 = 76.6%

Conclusion

Objectif de notre travail

Modéliser les défauts de conception afin de pouvoir les détecter automatiquement dans le code

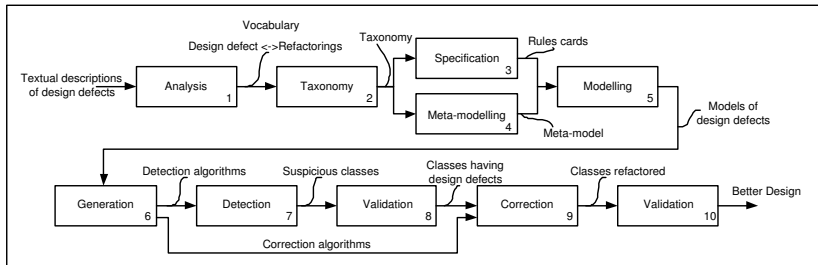
Objectif atteint

- Manque de travaux sur la spécification des défauts
- Nécessiter de représentations précises et structurées
- Une taxonomie et un métamodèle
- Fournir un cadre de développement de techniques et d'outils pour la détection des défauts

À faire...

- Affiner la description des règles (règles comportementales)
- Augmenter notre base de défauts
- Améliorer la précision de nos algorithmes de détection

Travaux à réaliser au LIFL



Objectifs

- Définir un langage pour spécifier des règles de correction des défauts de conception
- Développer des techniques de correction
- Intégrer les techniques de refactorings
- Etudier les techniques de transformation de modèles

Travaux à réaliser au LIFL

Langage de correction : exemple

```
RULE_CARD : Class {  
  RULE: Blob {ASSOC: associated FROM: LargeClass ONE TO: DataClass MANY};  
  RULE: LargeClass { ExtractClass  
                    || ExtractSubclass };  
  RULE: ExtractClass { if (METRIC: LCOM, HIGH) then MoveMethod(LargeClass,DataClass)  
                      && MoveField(LargeClass,DataClass) };  
};
```

Références

- ALB 02** ALBIN-AMIOT H., COINTE P., GUÉHÉNEUC Y.-G., *Un méta-modèle pour coupler application et détection des design patterns*, DAO M., HUCHARD M., Eds., actes du 8e colloque Langages et Modèles à Objets, vol. 8, numéro 1–2/2002 de RSTI L'objet, Hermès Science Publications, janvier 2002, p. 41–58.
- BRO 98** BROWN W. J., MALVEAU R. C., BROWN W. H., III H. W. M., MOWBRAY T. J., *Anti Patterns : Refactoring Software, Architectures, and Projects in Crisis*, John Wiley and Sons, 1st édition, March 1998.
- FOW 99** FOWLER M., *Refactoring – Improving the Design of Existing Code*, Addison- Wesley, 1st édition, June 1999.
- GUÉ 01** GUÉHÉNEUC Y.-G., ALBIN-AMIOT H., *Using Design Patterns and Constraints to Automate the Detection and Correction of Inter-Class Design Defects*, LI Q., RIEHLE R., POUR G., MEYER B., Eds., proceedings of the 39th conference on the Technology of Object-Oriented Languages and Systems, IEEE Computer Society Press, July 2001, p. 296–305.
- GAM 94** GAMMA E., HELM R., JOHNSON R., VLISSIDES J., *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1st édition, 1994.
- MOH 05c** MOHA N., HUYNH D.-L., GUÉHÉNEUC Y.-G., *A Taxonomy and a First Study of Design Pattern Defects*, ANTONIOL G., GUÉHÉNEUC Y.-G., Eds., Proceedings of the STEP International Workshop on Design Pattern Theory and Practice (IWDPTP05), September 2005.