Des signatures numériques pour améliorer la recherche structurelle de patrons

Yann-Gaël Guéhéneuc et Houari Sahraoui

Université de Montréal
CP 6128 succ. Centre Ville
Montréal, Québec, H3C 3J7
Canada
{guehene, sahraouh}@iro.umontreal.ca

RÉSUMÉ. Les patrons de conception orientés-objets décrivent de *bonnes* solutions à des problèmes récurrents de conception des programmes. Les solutions proposées sont des *motifs de conception* que les concepteurs introduisent dans l'architecture de leurs programmes. Il est important d'identifier, pendant la maintenance, les motifs de conception utilisés dans l'architecture d'un programme pour comprendre les problèmes de conception résolus et faire des modifications pertinentes au programme. L'identification de *micro-architectures* similaires à des motifs de conception est difficile à cause du large espace de recherche, *i.e.*, les nombreuses combinaisons de classes possibles. Nous proposons une étude expérimentale des classes jouant un rôle dans des motifs de conception avec des métriques et un algorithme d'apprentissage pour associer des *signatures numériques* aux rôles dans les motifs de conception. Une signature numérique est un ensemble de valeurs de métriques qui caractérise les classes jouant un rôle dans un motif de conception. Nous montrons que les signatures numériques permettent de réduire efficacement l'espace de recherche des micro-architectures similaires à des motifs de conception sur l'exemple du patron de conception Composite et du programme JHOTDRAW.

ABSTRACT. Design patterns describe *good* solutions to common and recurring problems in program design. The solutions are *design motifs*, which software engineers introduce in the architecture of their programs. It is important to identify the design motifs used in a program architecture to understand solved design problems and to make informed changes to the program, during maintenance. The identification of *micro-architecture*s similar to design motifs is difficult because of the large search space, *i.e.*, the many possible combinations of classes. We propose an experimental study of classes playing roles in design motifs using metrics and a machine-learning algorithm to associate *numerical signatures* with design motifs roles. A numerical signature is a set of metric values characterising classes playing a given role. We devise numerical signatures experimentally using a repository of micro-architectures similar to design motifs. We show that numerical signatures help in reducing the search space of micro-architectures similar to design motifs efficiently using the Composite design motif and the JHOTDRAW framework.

MOTS-CLÉS. Patrons, rôles, recherche structurelle, micro-architectures, signatures numériques, métriques.

KEYWORDS. Patterns, roles, structural identification, micro-architectures, numerical signatures, metrics.

1 Restructuration pertinente des programmes

Les solutions offertes par les patrons de conception [GAM 94] à des problèmes de conception récurrents sont décrites sous la forme de *motifs de conception* dont les concepteurs s'inspirent pour concevoir leurs programmes. Les motifs de conception déclarent des rôles et les relations entre ces rôles, comme présenté sur la figure 1. L'architecture d'un programme peut contenir de nombreux ensembles de classes ou *micro-architectures* similaires à des motifs de conception lorsque les concepteurs ont utilisé des patrons de conception. La figure 2 décrit succinctement un métamodèle pour de telles micro-architectures.

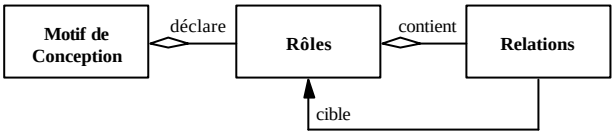

Figure 1 : *Métamodèle pour les motifs de conception*

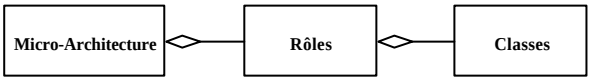

Figure 2 : *Métamodèle pour les micro-architectures*

Lors de la rétro-conception d'un programme, il est important de connaître les motifs de conception utilisés par les concepteurs. En effet, la connaissance des motifs utilisés permet une meilleure compréhension des problèmes résolus lors de la conception du programme. La compréhension des problèmes de conception est une étape nécessaire pour faire des changements pertinents, et de qualité, au programme.

L'acquisition de la connaissance des motifs de conception introduits par les concepteurs dans l'architecture d'un programme nécessite de trouver toutes les micro-architectures similaires à des motifs de conception, *i.e.*, de trouver toutes les classes (ou interfaces) dont la structure et l'organisation sont similaires à des motifs de conception. Ainsi, l'identification des motifs de conception correspond à un problème de recherche de sous-graphes dans des graphes et est ainsi complexe (temps et espace prohibitifs, faibles précisions et rappels) à cause des nombreuses combinaisons de classes possibles et de la taille des programmes [ANT 98]. De plus, les approches existantes utilisent seulement des informations syntaxiques pour identifier des motifs de conception et ont ainsi une faible précision : la sémantique des motifs n'est pas prise en compte.

Une possibilité pour réduire la complexité de l'identification des motifs de conception est de concevoir et d'appliquer des heuristiques qui réduisent l'espace de recherche, *i.e.*, qui réduisent le nombre potentiel de combinaisons des classes. Nous explorons cette possibilité par une étude expérimentale des micro-architectures

similaires à des motifs de conception. Nous cherchons à trouver des signatures numériques communes aux classes qui jouent certains rôles dans des motifs de conception pour faciliter leur identification et ainsi réduire la complexité des algorithmes d'identification des motifs de conception.

Nous décrivons un premier dispositif d'expérimentation et les résultats obtenus avec ce dispositif. Nous associons aux rôles définis dans des motifs de conception des *signatures numériques*. Nous inférons ces signatures numériques en utilisant un algorithme d'apprentissage basé sur des règles propositionnelles et des ensembles de valeurs de métriques calculées sur des classes identifiées manuellement comme jouant des rôles dans des motifs de conception. Nous utilisons ces signatures numériques pour réduire efficacement le nombre de classes jouant potentiellement un rôle dans un motif de conception. De part la manière dont les signatures numériques sont inférées, celles-ci introduisent un élément de sémantique dans l'identification de motifs de conception.

La section 2 résume les travaux connexes et conclue sur les limitations des approches existantes. La section 3 décrit notre technique de recherche structurelle de micro-architectures similaires à des motifs de conception avec la programmation par contraintes avec explications. La section 4 introduit les fondements théoriques de notre étude expérimentale des micro-architectures et présente notre premier dispositif d'expérimentation pour inférer des signatures numériques pour les rôles dans des motifs de conception. La section 5 discute les signatures numériques inférées et la réduction de l'espace de recherche. Enfin, la section 6 conclue et présente les travaux futurs.

2 Autres travaux sur l'identification de patrons et les métriques

Le travail présenté dans ce papier recouvre deux domaines de recherche : l'identification des motifs de conception par recherche structurelle et l'étude des programmes et des motifs de conception avec des métriques. Cette section présente brièvement les travaux connexes majeurs dans chacun des domaines.

D'une part, de nombreux travaux portent sur l'identification des motifs de conception. La plupart des approches se basent sur l'appariement structurel entre des groupes de classes, des micro-architectures, et des motifs de conception. Différentes techniques sont utilisées : règles d'inférence [KRA 96, WUY 98], requêtes [CIU 99, KEL 99], réseaux de raisonnement flou [JAH 97], programmation par contraintes [GUE 01, QUI 97]. Par exemple, Wuyts [WUY 98] a développé l'environnement SOUL dans lequel les motifs de conception sont représentés comme des prédicats et les programmes comme des faits (classes, méthodes, champs…). Un algorithme d'inférence Prolog unifie prédicats et faits pour identifier les classes qui jouent des rôles dans des motifs de conception. Le principal problème de l'appariement structurel est la complexité combinatoire inhérente à l'identification de sous-ensembles de classes qui correspondent à la description d'un motif de conception,

i.e., à l'isomorphisme de sous-graphes [EPP 95]. Antoniol *et al.* ont proposé une approche pour réduire l'espace de recherche des micro-architectures [ANT 98] dans laquelle ils utilisent un processus de filtrage en plusieurs étapes pour identifier par des métriques des micro-architectures identiques à des motifs de conception. Pour chaque classe d'un programme, ils calculent des métriques (par exemple, la profondeur de l'arbre d'héritage, le nombre d'associations) et ils comparent les valeurs de ces métriques avec les valeurs attendues pour des rôles dans des motifs de conception. Les valeurs attendues sont inférées théoriquement de la description des patrons de conception. La principale limitation de leur travail est l'hypothèse que l'implantation d'un programme (les micro-architectures) doit refléter la théorie (les motifs de conception), ce qui est rarement le cas. De plus, l'association théorique de valeurs de métriques à des rôles, lorsqu'elle est possible, réduit peu l'espace de recherche.

D'un autre côté, des travaux ont étudié l'impact des patrons de conception sur les valeurs de métriques. En particulier, dans un travail semblable au nôtre, Masuda *et al.* [MAS 99] ont réalisé une évaluation quantitative et une analyse de l'application de motifs de conception. Ils ont construit deux ensembles de programmes : un ensemble contenant deux programmes implantés sans patrons de conception et un ensemble contenant les deux même programmes réécrits avec des patrons de conception. Ils ont calculé les métriques de Chidamber et Kemerer [CHI 93] sur les deux ensembles de programmes et ont comparé les résultats. Ils concluent que l'utilisation de patrons de conception détériore certaines métriques et suggèrent que les métriques de Chidamber et Kemerer ne sont peut-être pas appropriées pour évaluer la qualité de programmes implantant des motifs de conception. Cependant, leur étude expérimentale porte seulement sur deux ensembles de deux programmes, ce qui est un ensemble de données trop petit pour être significatif.

Le travail que nous proposons dans ce papier développe les idées de Antoniol *et al.* et de Masuda *et al.* et résout le problème de la définition des valeurs attendues des métriques en calculant ces valeurs directement depuis un référentiel de micro-architectures similaires à des motifs de conception.

3 Identification de micro-architectures similaires à des motifs de conception

Dans des travaux précédents [ALB 01, GUE 01], nous présentions une technique originale d'identification des micro-architectures[1] similaires à des motifs de conception par une recherche structurelle basée sur la programmation par contraintes avec explications [JUS 00]. Nous montrions que les motifs de conception peuvent se réécrire comme des systèmes de contraintes : chaque rôle est représenté par une

[1] Le terme « micro-architectures » sera maintenant utilisé pour désigner les micro-architectures similaires à un motif de conception.

variable et les relations entre les rôles par des contraintes entre les variables. Toutes les variables ont des domaines identiques : l'ensemble des classes du programme dans lequel identifier les micro-architectures.

Par exemple, l'identification des micro-architectures similaires au motif de conception Composite, dont la description est montrée sur la figure 3, dans le programme JHOTDRAW, se traduit par le système de contraintes :

Variables :
client
composant
composite
feuille
Contraintes :
association(client, composant)
héritage(composant, composite)
héritage(composant, feuille)
composition(composite, composant)

où les quatre variables *client*[2], *composant*, *composite* et *feuille* ont des domaines identiques, qui contiennent les 155 classes (et interfaces) du programme JHOTDRAW, et les quatre contraintes représentent les relations d'*association*, d'*héritage* et de *composition* suggérées entre les rôles du motif de conception Composite.

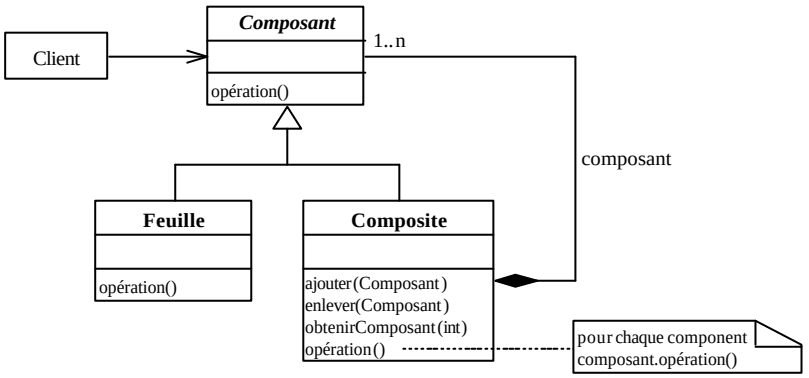

Figure 3 : *Motif de conception Composite*

Le système de contraintes peut alors être résolu par la technique de programmation par contraintes avec explications, laquelle permet (1) de trouver les

[2] On pourrait arguer que Client n'est pas un rôle en soi. Nous ne prenons pas position ici car nous pensons que les signatures numériques pourront également apporter une réponse à cette question.

solutions au système de contraintes et, lorsqu'il n'y a pas (ou plus) de solution, (2) de proposer à l'utilisateur de relâcher dynamiquement les contraintes de son choix pour diriger la recherche de solutions supplémentaires. Une solution est une micro-architecture dont la forme est soit complète par rapport au motif de conception recherché, soit approchée si des contraintes ont été relâchées [GUE03].

L'intérêt de l'utilisation de la programmation par contraintes avec explications est l'utilisation d'un unique modèle d'un motif de conception, le système de contraintes, pour identifier à la fois les formes complètes et les formes approchées du motif. La limitation principale de cette technique, comme de toutes les autres techniques basées sur une recherche structurelle, est la taille de l'espace de recherche. En effet, la définition du système de contraintes implique un espace de recherche de départ contenant un nombre de micro-architectures possibles égal à la taille du domaine des variables élevé à la puissance du nombre de variables :

$$(\textit{taille du domaine des variables})^{(\textit{nombre de variables})}$$

Par exemple, dans l'exemple de l'identification du motif de conception Composite dans le programme JHOTDRAW, l'espace de recherche contient, avant toute propagation de contraintes, 155^4 = 577 200 625 micro-architectures possibles (formes complètes uniquement). L'espace de recherche est donc possiblement très grand même avec un programme de taille réduite et résulte (1) en un temps de calcul important pour propager les contraintes et énumérer les différentes micro-architectures possibles et (2) en un nombre élevé de micro-architectures satisfaisant les contraintes mais sémantiquement différentes du motif de conception, *i.e.*, faussement positives.

Aussi, dans les chapitres suivants, nous explorons la possibilité de réduire efficacement l'espace de recherche des micro-architectures et le nombre de fausses positives par la réduction de la taille du domaine des variables pour ne retenir dans chaque domaine des variables que les classes dont les valeurs d'attributs internes à celles du rôle dans le motif recherché.

4 Signatures numériques pour les rôles dans les motifs de conception

4.1. Construction des signatures numériques

Nous cherchons à identifier les classes jouant des rôles dans des motifs de conception en utilisant leurs attributs internes. Des attributs internes consensuels pour les classes dans les langages de programmation orientés-objets sont :

- la taille/complexité, *e.g.*, le nombre de méthodes, de champs ;
- la filiation, *e.g.*, le nombre de parents, le nombre d'enfants d'une classe dans l'arbre d'héritage, la profondeur de l'arbre d'héritage ;

- la cohésion, *e.g.*, le degré avec lequel les attributs et les méthodes d'une classe sont liés ;
- le couplage, *e.g.*, la dépendance résultante des liens existants entre deux classes.

Plusieurs classes peuvent avoir des valeurs identiques pour un ensemble donné d'attributs internes. En effet, plusieurs classes peuvent jouer le même rôle dans différentes utilisations d'un même motif de conception et une même classe peut jouer des rôles dans plusieurs motifs de conception. Ainsi, les attributs internes d'une classe ne peuvent être utilisés pour distinguer de manière unique les classes jouant des rôles dans des motifs de conception.

Cependant, les attributs internes d'une classe peuvent être utilisés pour réduire l'espace de recherche des micro-architectures. Nous pouvons utiliser les attributs internes pour éliminer efficacement les classes qui ne jouent *évidemment* aucun rôle dans un motif de conception de l'espace de recherche, *i.e.*, les vraies négatives. De plus, à ce jour, aucune étude empirique n'a validé l'impossibilité d'identifier des classes uniquement par leurs attributs internes ou tenté de trouver expérimentalement des *points communs* quantifiables entre les classes jouant un rôle donné dans un motif de conception.

C'est pourquoi nous étudions l'utilisation des attributs internes des classes pour quantifier les rôles dans les motifs de conception : nous définissons des signatures numériques pour les rôles dans des motifs de conception avec les attributs internes des classes. Nous rassemblons ces signatures numériques en des règles pour identifier les classes jouant un rôle donné. Par exemple, la règle pour le rôle de Singleton dans le motif de conception Singleton pourrait être [GAM 94] :

Règle pour le rôle « Singleton » :
Filiation : nombre de parents faible,
nombre d'enfants faible.

car une classe jouant le rôle de Singleton est normalement haute dans l'arbre d'héritage et n'a habituellement pas d'enfants. La règle pour le rôle d'Observateur dans le motif de conception Observateur pourrait être :

Règle pour le rôle « Observateur » :
Couplage : couplage faible avec les autres classes.

car l'objectif du motif de conception Observateur est de réduire le couplage entre les classes jouant le rôle d'Observateur et le reste du programme.

La création des signatures numériques pour les rôles dans les motifs de conception se décompose essentiellement en deux étapes :

1. D'abord, nous construisons un *référentiel* de classes formant des micro-architectures dans différents programmes. Nous identifions manuellement les rôles joués par les classes dans des motifs de conception. Nous *analysons syntaxiquement* les programmes dans lesquels nous avons identifié des micro-

architectures pour obtenir des modèles de ces programmes. Nous *extrayons des métriques* de ces modèles des programmes pour associer, à chaque classe du référentiel, un ensemble de valeurs de ses attributs internes.

2. Ensuite, nous donnons les ensembles de valeurs de métriques à un *algorithme d'apprentissage* par règles propositionnelles. L'algorithme fournit des ensembles de règles caractérisant les rôles des motifs de conception avec les valeurs des métriques des classes jouant ces rôles. Nous réalisons une *validation croisée* des règles par la méthode « leave-one-out ».

Enfin, nous interprétons les résultats obtenus (ou l'absence de résultats) pour la réduction de l'espace de recherche des micro-architectures.

4.2. Création du référentiel

Nous avons besoin d'un référentiel de classes formant des micro-architectures pour analyser quantitativement ces classes. Nous investiguons manuellement plusieurs programmes pour identifier les micro-architectures et pour construire un référentiel contenant ces micro-architectures, la DPL[3]. Nous créons ce référentiel en utilisant plusieurs sources de données :

- des études dans la littérature, telle que l'étude originale de James Bieman *et al.* [BIE 03], qui dressent une liste des classes jouant des rôles dans des motifs de conception dans différents programmes C++, Java et Smalltalk ;
- notre suite d'outils pour l'identification des motifs de conception, PTIDEJ [ALB 01, GUE 01], qui utilise la programmation par contraintes avec explications pour identifier des micro-architectures dans des programmes ;
- deux travaux dans un cours gradué et dans un cours sous-gradué, durant lesquels les étudiants ont analysé plusieurs programmes Java.

Le référentiel de micro-architectures rassemble :

- pour chaque programme, les motifs de conception pour lesquels nous avons trouvé des micro-architectures ;
- pour chaque motif de conception, les micro-architectures que nous avons trouvées dans le programme ;
- pour chaque micro-architecture, les rôles joués par les classes dans les motifs de conception correspondants.

Nous validons manuellement toutes les micro-architectures avant de les inclure dans le référentiel. La construction du référentiel est un travail long dont la difficulté dépend de la compréhension des programmes analysés. Aussi, nous ne prétendons

[3] DPL est l'acronyme de « Design Patterns Library ».

pas avoir identifié toutes les micro-architectures présentes dans les programmes analysés. Le référentiel n'a pas besoin d'être exhaustif pour un programme donné. Cependant, plus le référentiel contiendra de micro-architectures, plus les signatures numériques reflèteront la sémantique des rôles associés.

Pour l'instant, la DPL inclut les données de six programmes, pour un total de 7 068 classes et 93 micro-architectures représentant 15 motifs de conception différents. Le tableau 1 résume les données dans la DPL. Les deux premières lignes donnent les noms et les nombres de classes (et interfaces) des programmes analysés. Les rangées suivantes indiquent, pour un patron de conception donné (par ligne), le nombre de micro-architectures trouvées similaires à son motif de conception dans chaque programme (par colonne). Le tableau résume également le nombre de rôles définis par un motif de conception et le nombre de classes jouant un rôle dans tous les programmes (deux dernières colonnes). Le nombre de classes jouant un rôle dans un motif de conception montre que seule une fraction de toutes les classes des programmes joue un rôle dans un motif de conception. De plus, certaines classes peuvent être comptées plus d'une fois si elles jouent différents rôles dans différents motifs de conception.

	JHotDraw v5.1	JRefactory v2.6.24	JUnit v3.7	Lexi v0.0.1α	NetBeans v1.0.x	QuickUML 2001	Total	Nombre de rôles [GAM94]	Nombre de classes jouant un rôle par motif de conception
Nombre de classes	173	575	157	127	5812	224	7 068		
Motif de conception	Nombre de micro-architectures similaires au motif de conception par programme								
Usine Abstraite					12	1	13	5	217
Adaptateur	1	17			8		26	4	230
Constructeur		2		1		1	4	4	24
Commande	1				1	1	3	5	67
Composite	1		1			2	4	4	107
Décorateur	1		1				2	4	64
Méthode Usine	3	1					4	4	67
Itérateur			1		5		6	5	30
Observateur	2		3	2		1	8	4	93
Prototype	2						2	3	32
Singleton	2	2	2	2		1	9	1	9
État	2	2					4	3	32
Stratégie	4						4	3	36
Méthode Patron	2						2	2	36
Visiteur		2					2	4	138
						Total	93	55	1 182

Tableau 1 : *Résumé du référentiel de micro-architectures*

Nous conservons ces données dans un fichier XML, lequel nous permet de parcourir les données pour calculer automatiquement des métriques et différentes

statistiques. Nous analysons syntaxiquement les programmes existant dans la DPL et nous calculons automatiquement des métriques sur leurs classes. L'analyse syntaxique et le calcul des métriques sont réalisés en trois étapes : d'abord, nous construisons un modèle d'un programme avec le métamodèle PADL[4] et ses analyseurs syntaxiques ; ensuite, nous calculons des métriques avec POM[5], une bibliothèque extensible de métriques basée sur PADL ; enfin, nous enregistrons les valeurs obtenues dans la DPL en ajoutant des attributs et des nœuds spécifiques à l'arbre XML. Le tableau 2 présente les métriques que nous utilisons pour associer des valeurs numériques aux attributs internes (taille/complexité, filiation, cohésion et couplage) des classes jouant un rôle dans un motif de conception.

	Acronymes	Descriptions	Références
Size/complexité	NM	Nombre de méthodes	[LOR 94]
	NMA	Nombre de nouvelles méthodes	[LOR 94]
	NMI	Nombre de méthodes héritées	[LOR 94]
	NMO	Nombre de méthodes surchargées	[LOR 94]
	WMC	Poids du nombre de méthodes	[CHI 93]
Filiation	CLD	Profondeur de la classe	[TEG 95]
	DIT	Profondeur de l'arbre d'héritage	[CHI 93]
	NOC	Nombre d'enfants	[CHI 93]
Cohésion	C	Connectivité	[HIT 95]
	LCOM5	Manque de cohésion des méthodes	[BRI 97b]
Couplage	ACMIC	Imports classes-méthodes des ancêtres	[BRI 97a]
	CBO	Couplage entre objets	[CHI 93]
	DCMEC	Imports classes-méthodes des fils	[BRI 97a]

Tableau 2 : *Attributs internes et métriques*

4.3. Construction des signatures numériques

La DPL contient un grand ensemble de données à analyser. Nous utilisons un algorithme d'apprentissage pour trouver les points communs entre les classes jouant un même rôle dans un motif de conception. Nous fournissons les données à un algorithme d'apprentissage par règles propositionnelles, JRIP, implanté dans WEKA, un programme avec code source libre qui rassemble des algorithmes d'apprentissage pour des tâches d'extraction de connaissances à partir de données, aussi appelé « data mining » [WIT 99].

Nous ne fournissons pas à JRIP toutes les données dans la DPL car cela amènerait à des résultats inintéressants à cause de la grande disparité entre les rôles, les classes et les valeurs des métriques. Nous fournissons à JRIP des sous-ensembles des données liés à chaque rôle. Un sous-ensemble σ de données lié à un rôle contient les valeurs des métriques pour les n classes jouant ce rôle dans toutes les micro-

[4] PADL est l'acronyme de « Pattern and Abstract-level Description Language ».

[5] POM est l'acronyme de « Primitives, Operators, Metrics ».

architectures similaires au motif de conception considéré. Nous ajoutons à ce sous-ensemble σ les valeurs des métriques de $3 \times n$ classes *ne* jouant *pas* le rôle, classes choisies aléatoirement dans le reste des données. Nous nous assurons que les classes choisies aléatoirement ont la structure attendue pour ce rôle, *i.e.*, si ce rôle doit être joué par une classe ou par une classe abstraite [GAM 94] pour augmenter leurs similitudes avec les classes jouant le rôle. L'algorithme d'apprentissage infère d'un sous-ensemble σ des règles pour chaque rôle. Nous validons les règles en utilisant la méthode « leave-one-out » [STO 74] avec chaque ensemble de valeurs des métriques dans les sous-ensembles σ. Pour un sous-ensemble σ de m classes (n classes plus $3 \times n$ autres classes), la méthode « leave-one-out » consiste à inférer une règle avec $m-1$ classes puis à vérifier la règle sur la m^e classe, et cela récursivement pour toutes les classes du sous-ensemble.

L'algorithme d'apprentissage infère des règles qui expriment la relation expérimentale entre les valeurs des métriques, d'un côté, et les rôles dans les motifs de conception, de l'autre côté. Typiquement, une règle inférée par l'algorithme d'apprentissage pour un rôle à la forme :

Règle pour le rôle « RÔLE » :
- *signature numérique 1, confiance 1,*
- …
- *signature numérique N, confiance N.*

où

signature numérique n : {métrique$_1$ $\hat{I}$ V_{1n}, …, *métrique*$_m$ $\hat{I}$ V_{mn}*}*

et les valeurs d'une métrique *métrique*$_i$, calculées sur les classes jouant le rôle, appartiennent à l'ensemble $V_{ij} \subset$ N. Le degré de confiance *confiance k* est la proportion de classes concernées par une signature numérique dans un sous-ensemble σ. Le degré de confiance est utilisé pour calculer l'erreur et le rappel.

Nous collectons les règles inférées par l'algorithme d'apprentissage et filtrons ces règles pour éliminer les règles non-caractéristiques avec les critères suivants :

1. Nous supprimons toutes les règles inférés de sous-ensembles σ trop « petits », *i.e.*, avec trop peu de classes jouant un rôle donné.
2. Nous supprimons toutes les règles avec un rappel inférieur à 75% ;

Nous décomposons les données dans le référentiel en 56 sous-ensembles σ et inférons autant de règles avec l'algorithme d'apprentissage, lesquelles comportent 78 signatures numériques. Le premier filtrage supprime 20 des 56 règles. Le second filtrage élimine les 16 règles dont le rappel est inférieur à 75%. Les 20 règles restantes sont résumées dans le tableau 3.

Nous normalisons également les règles pour remplacer les valeurs seuils calculées sur les classes du référentiel par des valeurs nominales (« faible »,

« moyen », «grand »…) calculées sur les distributions des différentes métriques dans les classes du programme dans lequel nous cherchons des micro-architectures.

Motif de conception	Rôles	Erreur (en %)	Rappel (en %)
Itérateur	Client	0,00	100,00
Observateur	Sujet	0,00	100,00
Observateur	Observateur	2,38	100,00
Méthode patron	Classe Concrète	0,00	97,60
Prototype	Prototype Concret	0,00	96,30
Décorateur	Composant Concret	4,17	89,58
Visiteur	Visiteur Concret	0,00	88,89
Stratégie	Contexte	3,70	88,89
Visiteur	Élément Concret	2,04	88,78
Singleton	Singleton	8,33	87,50
Méthode Usine	Créateur Concret	4,30	87,10
Méthode Usine	Produit Concret	3,45	86,21
Adaptateur	Cible	4,00	84,00
Composite	Feuille	6,47	82,09
Décorateur	Décorateur Concret	0,00	80,00
Itérateur	Itérateur	0,00	80,00
Commande	Receveur	6,67	80,00
État	État Concret	6,67	80,00
Stratégie	Stratégie Concrète	2,38	78,57
Commande	Commande Concrète	3,23	77,42

Tableau 3 : *Règles inférées, rôles, erreurs et rappels*

5 Utilisation des signatures numériques

Les signatures numériques aident à réduire efficacement l'espace de recherche des micro-architectures. Lors de la recherche de classes dont la structure et l'organisation sont similaires à un motif de conception, l'utilisation des signatures numériques permet de supprimer de l'espace de recherche toutes les classes dont les signatures numériques ne correspondent pas aux signatures numériques attendues pour les rôles du motif de conception.

Par exemple, la règle :

Règle pour le rôle « Cible »
- WMC faible, 24/25.

correspond au rôle Cible dans le motif de conception Adaptateur et indique que les classes jouant le rôle de Cible doivent avoir une signature numérique dans laquelle la valeur de la métrique WMC est faible. Ainsi, lors de la recherche de micro-architectures similaires au motif de conception Adaptateur, nous pouvons supprimer de l'ensemble des classes candidates pour jouer le rôle de Cible toutes les classes avec une forte complexité, *i.e.*, toutes les classes avec une valeur moyenne ou haute pour WMC.

Nous intégrons les signatures numériques avec notre outil PTIDEJ, basé sur la programmation par contraintes avec explications, et pouvons alors réduire l'espace de recherche de deux façons :

- nous pouvons affecter aux variables des domaines qui ne contiennent que les classes dont les signatures numériques correspondent aux signatures numériques attendues pour les rôles ;
- nous pouvons ajouter des contraintes unaires sur chaque variable pour faire correspondre la signature numérique des classes dans son domaine avec la signature numérique du rôle correspondant.

Ces deux façons donnent le même résultat : elles suppriment du domaine d'une variable toutes les classes dont les signatures numériques ne correspondent par aux signatures numériques du rôle correspondant, réduisant ainsi l'espace de recherche en réduisant le domaine des variables.

Dans le motif de conception Composite, parmi les rôles de Client, Composant, Composite et Feuille, l'algorithme d'apprentissage infère seulement une règle avec un rappel supérieur à 75% pour le rôle de Feuille, comme montré dans le tableau 3. La règle et les signatures numériques associées au rôle de Feuille sont :

Règle pour le rôle « Feuille »
- NMI grand et DIT grand, 23/67
- NIM grand et NMO petit, 45/67
- DIT moyen et NM moyen, 9/67

Nous calculons les métriques du tableau 2 sur les classes du programme JHOTDRAW et appliquons les signatures numériques successivement pour évaluer la réduction de l'espace de recherche. Le tableau 4 montre le pourcentage de réduction de l'espace de recherche pour chaque signature numérique ; elle se situe entre 69,00% et 89,15% pour les classes jouant potentiellement le rôle de Feuille dans une micro-architecture similaire au motif de conception Composite. Le calcul de toutes les métriques a duré moins de deux secondes sur un AMD ATHLON avec un processeur 64 bits à 2 GHz (nous calculons ces métriques une seule fois pour un programme donné) alors que l'identification des micro-architectures prend plus de quatre heures avec l'implantation de référence de la programmation par contraintes avec explications, PALM [GUE01b, JUS 00]. Le calcul des micro-architectures similaires au motif de conception Composite avec un espace de recherche réduit pour le rôle de Feuille prend seulement un peu plus de deux heures.

Signatures numériques	Nombre de classes correspondant à cette signature numérique	Réduction de l'espace de recherche (en %)
NMI grand et DIT grand	20	69,00
NIM grand et NMO petit	7	89,15
DIT moyen et NM moyen	10	84,50

Tableau 4 : *Réduction de l'espace de recherche du rôle Feuille dans JHOTDRAW*

Ainsi, les signatures numériques peuvent apporter une aide appréciable à l'identification des motifs de conception en réduisant efficacement l'espace de recherche. De plus, les signatures numériques introduisent un élément de sémantique dans l'identification des motifs de conception. En effet, les signatures numériques sont inférées depuis un référentiel de classes qui jouent des rôles dans des motifs de conception, *i.e.*, des classes qui respectent la sémantique des rôles tels que définis par le motif de conception. L'utilisation des signatures numériques permet donc également de pallier les limitations des approches purement syntaxiques à l'identification de motifs de conception.

Cependant, il faut distinguer trois types de règles inférées pour évaluer la pertinence de la réduction de l'espace de recherche :

1. Des règles de « bon sens » qui confirment l'intuition que l'on pourrait avoir en étudiant théoriquement les rôles correspondants ;
2. Des règles contraires au « bon sens » qui vont à l'opposé de l'intuition que l'on pourrait avoir sur les rôles et qui sont le résultat d'un manque de données ;
3. Des règles qui ne sont pas vérifiables par l'intuition et qui demandent une étude plus poussée pour confirmer ou infirmer leur pertinence.

Les règles de type 1 permettent de réduire l'espace de recherche en éliminant des domaines des variables les classes dont les signatures numériques ne correspondent pas aux rôles recherchés. La réduction des domaines des variables implique la réduction du nombre de classes faussement associées à un rôle pendant la recherche structurelle et limite donc le nombre de micro-architectures faussement identifiées comme similaire à un motif de conception (fausses positives). Les règles de type 2 ne doivent pas être prises en compte avant d'être confirmées sur un ensemble suffisamment grand de données. Les règles de type 3, quant à elle, doivent être utilisées avec la plus grande attention. En effet, si ces règles s'avèrent correspondre à une réalité, même contraire à l'intuition, alors la réduction de l'espace de recherche aura pour conséquence une réduction du nombre de micro-architectures faussement positives. Mais, si ces règles sont erronées, à cause d'un manque de données, de la qualité des données, de l'algorithme d'apprentissage ou de la méthode de validation, alors la réduction de l'espace de recherche se traduira en une réduction du nombre de micro-architectures identifiées. Des micro-architectures seront alors rejetées pour de mauvaises raisons (fausses négatives).

6 Conclusion et travaux futurs

Nous avons présenté les premiers résultats d'une étude expérimentale des micro-architectures similaires à des motifs de conception. Nous avons construit un référentiel de telles micro-architectures en utilisant différentes sources. Nous avons calculé de nombreuses métriques sur les classes jouant des rôles dans des motifs de conception. Nous avons utilisé un algorithme d'apprentissage par règles

propositionnelles et identifié des signatures numériques communes à toutes les classes jouant un rôle donné dans un motif de conception. Nous avons utilisé ces signatures numériques pour améliorer un algorithme d'identification des motifs de conception utilisant un système de contraintes. L'identification de motifs de conception maintenant se décompose en deux étapes : (1) l'identification des classes candidates pour les rôles-clés dans un motif de conception en supprimant les classes dont les signatures numériques ne correspondent pas et (2) l'identification des classes pour les rôles restant en partant des classes jouant des rôles-clés par recherche structurelle. Ces étapes réduisent efficacement l'espace de recherche, en particulier dans le cas de programmes de grandes tailles, en supprimant les classes qui ne jouent évidemment pas un rôle donné dans un motif de conception.

Nous planifions maintenant de conduire une analyse poussée des motifs de conception et de leur impact sur la qualité en évaluant les caractéristiques de qualité supposées de chaque motif de conception indépendamment (extensibilité, compréhensibilité…) avec notre référentiel et les métriques. Aussi, nous préparons une nouvelle étude avec un référentiel plus large de micro-architectures similaires à des motifs de conception. Dans cette nouvelle étude, nous améliorerons le dispositif d'expérimentation. Un changement important concernera les sous-ensembles σ utilisés pour calculer les signatures numériques de chaque rôle : le nombre de contre-exemples et leurs natures seront choisis aussi proche de la réalité que possible pour obtenir des signatures numériques très précises. Aussi, nous étudierons l'utilisation d'autres métriques orientées-objets.

Bibliographie

[ALB 01] H. Albin-Amiot, P. Cointe, Y.-G. Guéhéneuc, and N. Jussien. Instantiating and detecting design patterns: Putting bits and pieces together. In proceedings of the 16th conference on Automated Software Engineering, pages 166–173. IEEE Computer Society Press, November 2001.

[ANT 98] G. Antoniol, R. Fiutem, and L. Cristoforetti. Design pattern recovery in object-oriented software. In proceedings of the 6th International Workshop on Program Comprehension, pages 153–160. IEEE Computer Society Press, June 1998.

[BIE 03] J. Bieman, G. Straw, H. Wang, P. Willard, and R. T. Alexander. Design patterns and change proneness: An examination of five evolving systems. In proceedings of the 9th international Software Metrics Symposium, pages 40–49. IEEE Computer Society Press, September 2003.

[BRI 97a] L. Briand, P. Devanbu, and W. Melo. An investigation into coupling measures for C++. In proceedings of the 19th International Conference on Software Engineering, pages 412–421. ACM Press, May 1997.

[BRI 97b] L. C. Briand, J. W. Daly, and J. K. W¨ust. A unified framework for cohesion measurement. In proceedings of the 4th international Software Metrics Symposium, pages 43–53. IEEE Computer Society Press, November 1997.

[CHI 83] S. R. Chidamber and C. F. Kemerer. A metrics suite for object-oriented design. Technical Report E53-315, MIT Sloan School of Management, December 1993.

[CIU 99] O. Ciupke. Automatic detection of design problems in object-oriented reengineering. In proceeding of 30th conference on Technology of Object-Oriented Languages and Systems, pages 18–32. IEEE Computer Society Press, August 1999.

[EPP 95] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. In proceedings of the 6th annual Symposium On Discrete Algorithms, pages 632–640. ACM Press, January 1995.

[GAM 94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, 1st edition, 1994.

[GUE 01a] Y.-G. Guéhéneuc and N. Jussien. Using explanations for design-patterns identification. In proceedings of the 1st IJCAI workshop on Modeling and Solving Problems with Constraints, pages 57–64. AAAI Press, August 2001.

[GUE 01b] Y.-G. Guéhéneuc and N. Jussien. Using explanations for design-patterns identification. In proceedings of the 1st IJCAI workshop on Modelling and Solving Problems with Constraints, pages 57–64. AAAI Press, August 2001.

[GUE 03] Y.-G. Guéhéneuc. Un cadre pour la traçabilité des motifs de conception. Thèse de doctorat, École des Mines de Nantes et Université de Nantes, août 2003.

[HIT 95] M. Hitz and B. Montazeri. Measuring coupling and cohesion in object-oriented systems. In proceedings of the 3rd International Symposium on Applied Corporate Computing, pages 25–27. Texas A & M University, October 1995.

[JAH 97] J. H. Jahnke, W. Sch¨afer, and A. Z¨undorf. Generic fuzzy reasoning nets as a basis for reverse engineering relational database applications. In proceedings of the 6th European Software Engineering Conference, pages 193–210. ACM Press, September 1997.

[JUS 00] N. Jussien and V. Barichard. The PaLM system: Explanation-based constraint programming. In Proceedings of TRICS: Techniques foR Implementing Constraint Programming Systems, pages 118–133. School of Computing, National University of Singapore, Singapore, September 2000. TRA9/00.

[KEL 99] R. K. Keller, R. Schauer, S. Robitaille, and P. Pagé. Pattern-based reverse-engineering of design components. In proceedings of the 21st International Conference on Software Engineering, pages 226–235. ACM Press, May 1999.

[KRA 96] C. Krämer and L. Prechelt. Design recovery by automated search for structural design patterns in object-oriented software. In proceedings of the 3rd Working Conference on Reverse Engineering, pages 208–215. IEEE Computer Society Press, November 1996.

[LOR 94] M. Lorenz and J. Kidd. Object-Oriented Software Metrics: A Practical Approach. Prentice-Hall, 1st edition, July 1994. D. P. Tegarden, S. D. Sheetz, and D. E. Monarchi. A software complexity model of object-oriented systems. In Decision Support Systems, 13(3–4):241–262, March 1995.

[MAS 99] G. Masuda, N. Sakamoto, and K. Ushijima. Evaluation and analysis of applying design patterns. In proceedings of the 2nd International Workshop on the Principles of Software Evolution. ACM Press, July 1999.

[QUI 97] A. Quilici, Q. Yang, and S. Woods. Applying plan recognition algorithms to program understanding. In journal of Automated Software Engineering, 5(3):347–372, July 1997.

[STO 74] M. Stone. Cross-validatory choice and assessment of statistical predictions. In Journal of the Royal Statistical Society, 36:111–147, 1974. Series B: Statistical Methodology.

[WIT 99] I. H. Witten and E. Frank. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann, 1st edition, October 1999.

[WUY 98] R. Wuyts. Declarative reasoning about the structure of object-oriented systems. In proceedings of the 26th conference on the Technology of Object-Oriented Languages and Systems, pages 112–124. IEEE Computer Society Press, August 1998.