

POM, un outil pour le calcul des métriques de qualité

Plan

- I. La méta-modélisation au service de la qualité**
 - A. Les métriques de qualité**
 - B. PADL et CPL**
- II. POM**
 - A. Architecture**
 - B. Implémentation**
 - C. Résultats**
- III. Orientations**

+ Plusieurs métriques mesurant une propriété d'un système logiciel (ex : cohésion d'une classe)

+ Recensement de métriques existantes :

<http://furet.iro.umontreal.ca>

+ Travaux de recherche contribuant à les exprimer de manière simple (ex : Lionel Briand)

+ Principe : une métrique est une expression basée sur l'application d'opérations sur des ensembles.

Ex : $CBO(c) = |\{d \in C \setminus \{c\} : \text{uses}(c,d) \text{ OR } \text{uses}(d,c)\}|$

+ But du projet : fournir un outil qui permet de calculer des métriques selon ce principe

CPL

- ✓ Utilise CFParse (bibliothèque d'IBM) pour la lecture et le parsing des class files (.class)
- ✓ Avantage : analyse du bytecode Java qui va être exécuté, élimination du « code mort », facilité d'analyse du code offert par CFParse (selon la spécification de la JVM)
- ✓ `util.Loader` : classe permettant le chargement d'un répertoire ou d'un fichier JAR contenant le système à analyser
- ✓ Documentation disponible sur demande

PADL

- ✓ Méta-modèle équivalent à un diagramme de classes
- ✓ Classes, attributs, méthodes et paramètres des méthodes
- ✓ Enrichi par rapport au premier méta-modèle de CFParce (ajout des relations d'agrégation, d'association et d'héritage)
- ✓ Possibilité d'y ajouter des informations complémentaires
- ✓ Permet d'extraire les informations nécessaires pour le calcul des métriques
- ✓ Package padl.kernel : constituants du méta-modèle (interfaces Java)
- ✓ Package padl.kernel.impl.v2 : classes implémentant les interfaces
- ✓ Classe padl.util.ModelStatistics : statistiques sur le système analysé (ex : nombre d'interfaces du système)

I. La méta-modélisation au service de la qualité

B. PADL et CPL

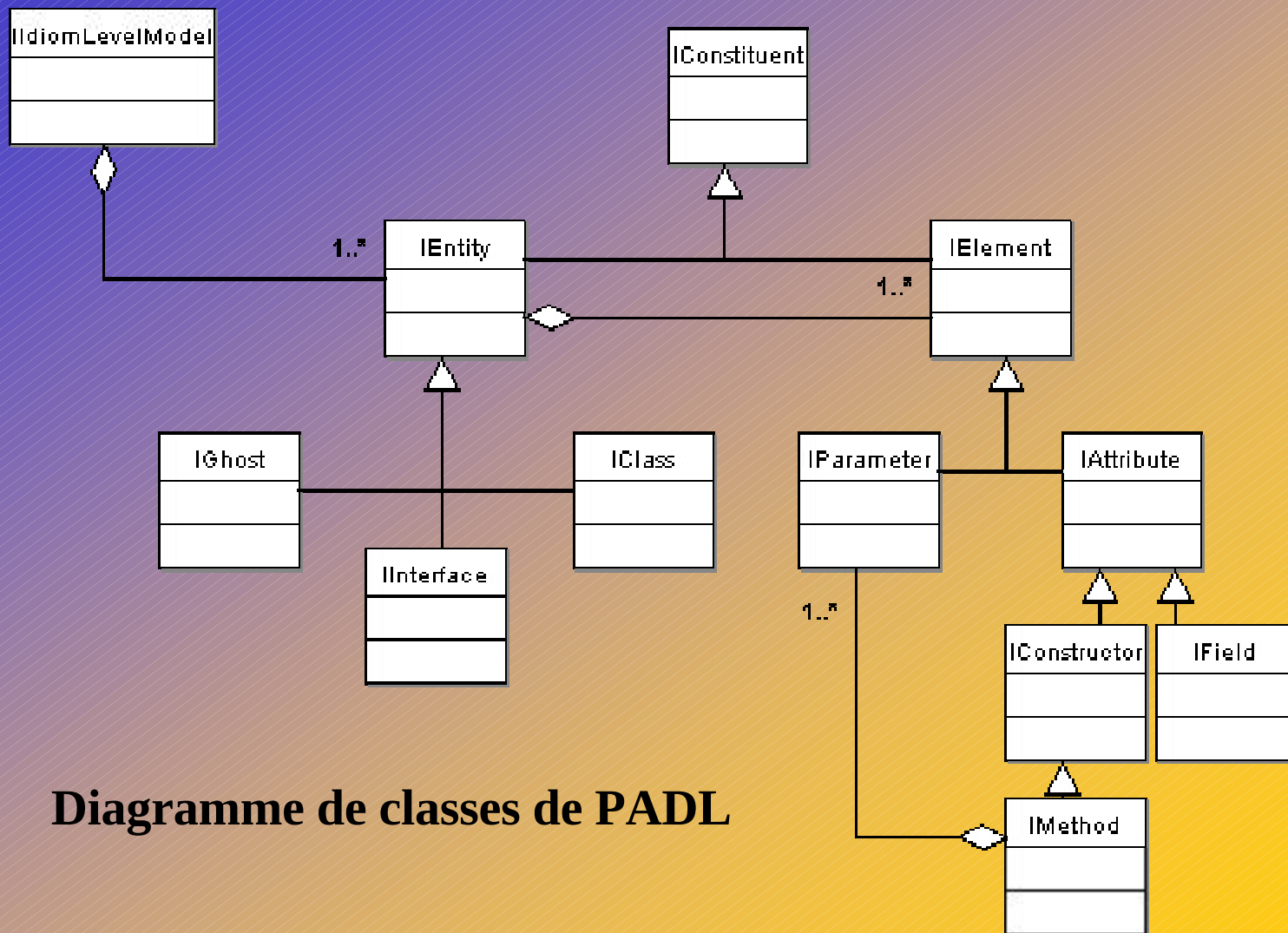


Diagramme de classes de PADL

Ajout des invocations de méthode

- ✓ Dans PADL ClassFile Creator, classe MethodInvocationAnalyzer
- ✓ But : savoir quelle méthode appelle quelle autre méthode ou utilise les attributs d'une classe
- ✓ Analyse du bytecode Java en reconstituant une machine à pile selon la spécification de la JVM. L'algorithme consiste à reconstituer les push et les pop sur la pile mémoire
- ✓ Découpage d'une méthode d'une classe en instructions Java, analyse de chaque instruction pour la reconnaissance de l'appelant et de l'appelé
- ✓ Résultats : appels de méthode statique et dynamique, utilisation des attributs d'une classe par une méthode, affectation

Packages

1. **pom.metrics** : implantation des métriques
2. **pom.operators** : opérations sur les primitives
3. **pom.primitives** : primitives extraites du méta-modèle

✚ Trois étapes pour le calcul de métriques:

1. CPL : parsing des class files

2. PADL : création du méta-modèle à partir du modèle fourni par CPL

3. POM

✓ Calcul des primitives (ex : méthodes héritées d'une classe) en utilisant le méta-modèle de PADL

✓ Résultat de la métrique, obtenue en appliquant les opérations sur ces ensembles (cardinalité, intersection, union, etc.)

Remarque : conférer le tutorial et les tests JUnit dans le projet POM Tests pour des exemples de code

+ Classe

- ✓ Ancêtres, descendants, enfants

+ Méthode

- ✓ Nouvelles, héritées, surchargées, abstraites, etc.

+ Champ

- ✓ Champs accédés par une méthode de la classe,
- ✓ Champs de la classe

Remarque : pour implémenter d'autres primitives, il est préférable de comprendre PADL ou de s'appuyer sur les primitives existantes. Si PADL manque d'informations, il est possible de le compléter. Il faut donc s'appuyer sur CPL et CFParse.

✚ **Implantation des métriques dont la définition était donnée**

✚ **24 métriques disponibles**

✓ **couplage : CBO, ACAIC, DCAEC**

✓ **cohésion : LCOM, cohesionAttributes, ICH, Connectivity**

✓ **héritage : DIT, AID, CLD, NOC, NOP**

✓ **à l'intérieur d'une classe : NMO, NMI, NMA, NOA, WMC**

✓ **etc...**

...dont une version de CBO binaire (entre 2 classes)

- ✚ Tests JUnit pour s'assurer de l'évolution du projet et prévenir les impacts sur les changements de PADL et POM
- ✚ Problèmes de performance lors des tests sur des gros programmes
 - ✓ Solution : allouer une mémoire fixe à la JVM
`java -Xmx1024M -Xms1024M`
 - ✓ Mise en cache des dernières listes calculées

- ✚ Incorporation de POM dans un plug-in Eclipse et un programme stand-alone**
 - ✓ Eclipse : calcul des métriques sur un projet présent dans le workspace d'Eclipse, export en XML
 - ✓ POM stand-alone : calcul des métriques unaires ou binaires, à partir d'un répertoire ou d'un fichier JAR, et export en XML ou CSV (Excel)
- ✚ Travaux ayant utilisés POM**
 - ✓ Publication : Fingerprinting Design Motif Roles
 - ✓ Représentation d'un système logiciel par un modèle physique

POM

- ✓ choix des primitives à utiliser (implémentation de nouvelles si besoin)
- ✓ Ecriture d'un test JUnit
- ✓ Codage de la métrique et exécution du test

POM stand-alone

- ✓ Pris automatiquement en compte grâce à l'utilisation de la réflexion Javz

POM sur Eclipse

- ✓ Etape équivalente à POM
- ✓ Ajout du nom de la métrique dans le fichier POMMetrics.txt

III. Orientations

- + Création d'un plug-in Eclipse à part**
- + Editeur d'expressions de métriques et génération de code Java**
- + Compléter le méta-modèle PADL avec des informations**
 - ✓ Ex : Structures de contrôle**
- + Projet en open-source**

Documents et fichiers disponibles

- ✚ El Hachemi Alikacem – Métriques et modèle générique
- ✚ El Hachemi Alikacem – Modélisation des métriques pour un modèle générique
- ✚ Documentation CFParse
- ✚ CVS Team project set du 29 Juin 2004
- ✚ JVM Specification version 2
- ✚ Lionel Briand, John W. Daly, and Jürgen Wüst- A Unified Framework for Coupling Measurement in Object-Oriented Systems

<http://citeseer.ist.psu.edu/briand97unified.html>

Merci de votre attention

\$1000par question

ou

zaidifar@iro.umontreal.ca