

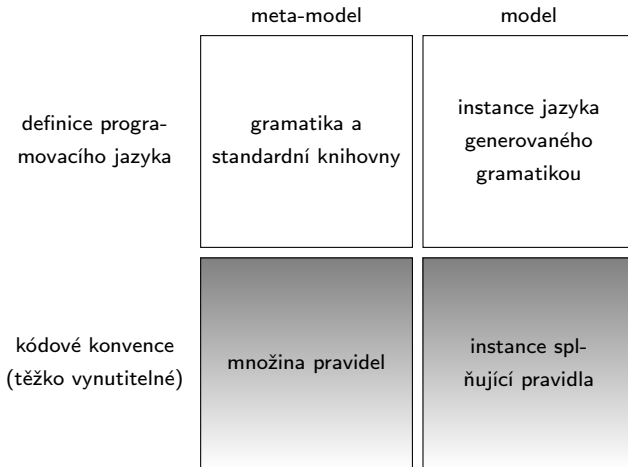
# Validace principů objektového návrhu v kódu

Martin Vejmelka

8. června 2011

- 1 Motivace a zadání práce
- 2 Analýza
- 3 Návrh formalizace pravidel
- 4 Realizace vyhodnocovacího nástroje
- 5 Závěr
- 6 Otázky

# Motivace projektu



# Zadání práce

*Seznamte se se základními principy používanými při objektově orientovaném návrhu a implementaci, konkrétně s low coupling, high cohesion a Law of Demeter. Popište pravidla, která umožní ověřování těchto principů. Vytvořte nástroj, který umožní analyzovat kód v jazyce Java a vyhodnocovat vámi definovaná pravidla. Činnost nástroje ověřte na vzorových příkladech kódu. Při návrhu nástroje se zaměřte na jeho budoucí rozšiřitelnost.*

# Zadání práce - dekompozice

Zadání práce bylo dekomponováno na následující části:

- řešerše souvisejících prací, seznámení se s principy objektově orientovaného návrhu, analýza problémové domény
- návrh formalizace popisu pravidel objektově orientovaného návrhu
- realizace nástroje pro vyhodnocování pravidel popsaných v navrženém formalismu
- testování výsledného řešení na vzorových příkladech kódu

# Analýza

- průzkum existujících řešení
- analýza návrhových konceptů a principů
- analýza problémové domény
  - analýza vstupů – projekty v jazyce Java
  - řešerše nástrojů pro předzpracování kódů do vhodné podoby

# Návrh formalizace pravidel

## Grafový model

$$G = \langle V, E, \rho, K, C, N, \text{Kind}, \text{Classifier}, \text{Name} \rangle$$

- $V$  je množina elementů (v našem případě části kódu),
- $E$  je množina hran (v našem případě vztahy mezi částmi kódu – např. volání funkce, dědičnost),
- $V \cap E = \emptyset$ ,
- $\rho : E \mapsto V \times V$  je zobrazení množiny hran do množiny uspořádaných dvojic vrcholů (incidence),
- $K$  je libovolná množina označení typů vrcholů,
- $C$  je množina klasifikátorů hran,
- $N$  je množina jmen (řetězců),
- $\text{Kind} : V \mapsto K$  je zobrazení, které přiřadí každému vrcholu jeho typ,
- $\text{Classifier} : E \mapsto C$  je zobrazení, které přiřadí každé hraně její klasifikátor (zda se jedná o vyvolání metody, dědičnost, apod.),
- $\text{Name} : V \mapsto N$  je zobrazení, které přiřadí vrcholu jeho jméno (např. jméno třídy, jméno metody).

# Návrh formalizace pravidel

## Pravidlo

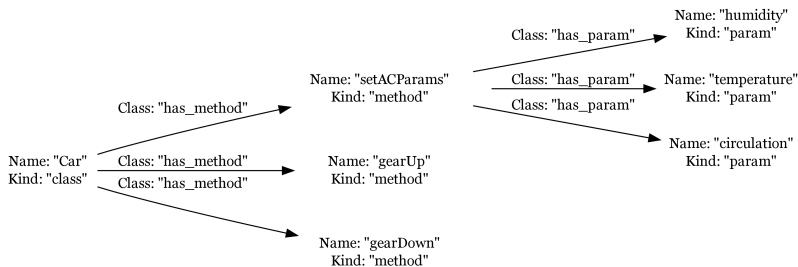
*Pro počet parametrů  $n$  každé metody platí  $n \leq 2$ .*

## Specifikace pravidla

$$\forall v \in V : \text{HasKind}(v, \text{"method"})$$
$$\text{CountLessThan}(\text{OutgoingEdges}(v, \text{"param"}), 2)$$

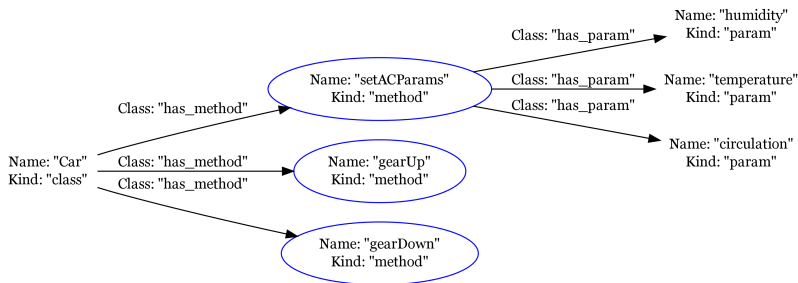


# Návrh formalizace pravidel



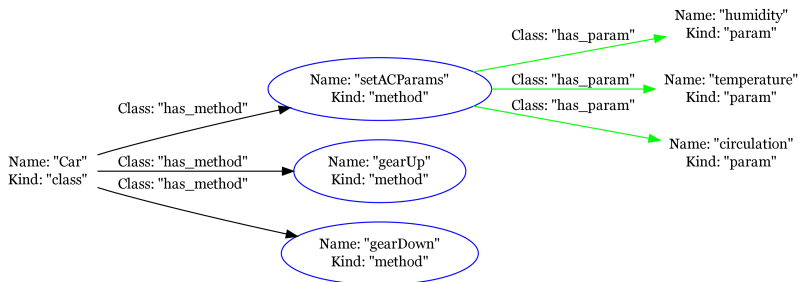
Obrázek: Příklad grafového modelu.

# Návrh formalizace pravidel



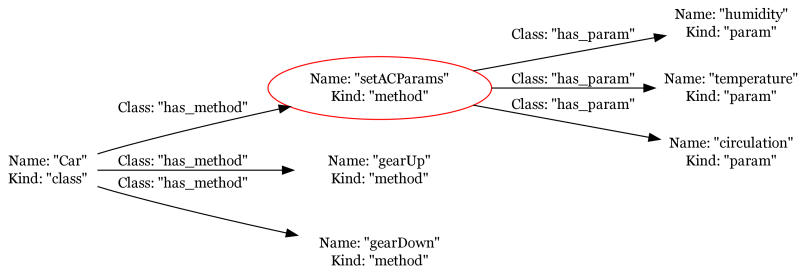
Obrázek: Výběr vrcholů:  $\forall v \in V : HasKind(v, "method")$ .

# Návrh formalizace pravidel



**Obrázek:** Výběr množiny hran:  $OutgoingEdges(v, "has\_param")$ .

# Návrh formalizace pravidel

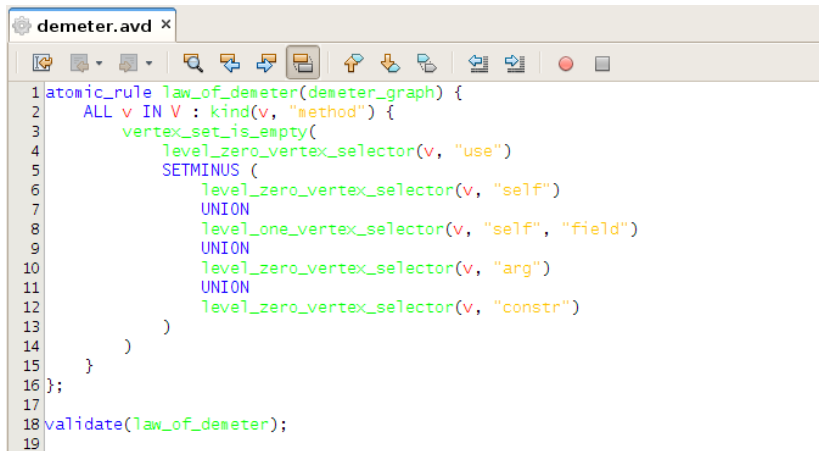


Obrázek: Chyba:  $\text{CountLessThan}(\text{OutgoingEdges}(v, \text{"has\_param"}), 2)$ .

# Realizace vyhodnocovacího nástroje - architektura

- *GraphModel* – množina grafů různých typů pro analyzovaný projekt
- *ValidationModel* – stromová struktura pravidel (operátorů) vygenerovaná z AVD souborů
- *AVD* – ArchVal Definitions – DSL pro serializaci matematických pravidel
- *body rozšíření systému*
  - operátory (predikáty, selektory)
  - analýzy
  - generátory grafů

# Realizace vyhodnocovacího nástroje - editor AVD



```
1 atomic_rule law_of_demeter(demeter_graph) {
2   ALL v IN V : kind(v, "method") {
3     vertex_set_is_empty(
4       level_zero_vertex_selector(v, "use")
5       SETMINUS (
6         level_zero_vertex_selector(v, "self")
7         UNION
8         level_one_vertex_selector(v, "self", "field")
9         UNION
10        level_zero_vertex_selector(v, "arg")
11        UNION
12        level_zero_vertex_selector(v, "constr")
13      )
14    )
15  }
16 };
17
18 validate(law_of_demeter);
19
```

# Závěr

- použití běžného matematického zápisu pro formalizaci pravidel
- rozšiřitelný nástroj pro vyhodnocování pravidel v zavedeném formalismu
- proof of concept - realizace pravidla Law of Demeter
- otestování práce na vzorových příkladech

# Otázky

## Otázka

*Jak hodnotíte váš nástroj v porovnání s obdobnými nástroji uvedenými v analytické části práce?*

## Odpověď

- *zmíněné nástroje jsou vyspělejší a dobře integrované do vývojových IDE,*
- *existující nástroje nezavádí matematický přístup k formalizaci pravidel (poskytují předdefinované množiny pravidel),*
- *práce neměla (alespoň prozatím) za cíl konkurovat existujícím nástrojům, ale vyzkoušet nový/jiný přístup.*



# Otázky

## Otázka

*Je možné některý z nástrojů upravit tak, aby byl schopný pracovat s navrženým způsobem ověřování?*

## Odpověď

- *jádro vyvinutého nástroje nemá přímé závislosti na konkrétní platformě,*
- *nutnost poskytnout kompletní grafový model, případně i přídatné operátory,*
- *možnost využít platformu poskytovanou touto prací.*

# Otázky

## Otázka

*Nebylo možné použít některý ze stávajících formalismů (např. Featherweight Java) pro popis pravidel?*

## Odpověď

- *nástroj je realizován na vyšší úrovni abstrakce,*
- *pravidla jsou definována nad obecným grafovým modelem,*
- *FJ by určitě bylo možné využít – jako elementy v pravidlech by vystupovaly elementy jazyka FJ (příp. FGJ),*
- *pro exaktní formalizaci bychom zvolili vhodné zobrazení programu ve FJ do grafu,*
- *nástroj pracuje nad programy v jazyce Java ( $FJ \subset Java$ ).*

# Otázky

## Otázka

*Chybějící formální sémantika formátu AVD.*

## Odpověď

- *AVD je pouze serializací zápisu pravidel popsaných v práci,*
- *jednalo se o nástroj jak vložit pravidla pro testování ve vhodné podobě do počítače (proof of concept),*
- *v případě budoucího rozvoje práce je určitě vhodné rozšířit existující ANTLR gramatiku o popis významu jednotlivých konstruktů.*

Děkuji za pozornost. . .