

Efficient Identification of Design Patterns with Bit-vector Algorithm

Olivier Kaczor, Yann-Gaël Guéhéneuc and Sylvie Hamel
{kaczorol, guehene, hamelsyl}@iro.umontreal.ca



GEODES and LBIT
Department of Informatics and Operations Research
University of Montreal



© Olivier Kaczor 2006



Outline

1. Introduction
2. Related Work
3. Our Approach:
 1. Code source transformation
 2. Identification with Bit-vector algorithm
4. Case Study
5. Current and Future Work



1. Introduction

(1/2)

Design patterns:

- Provide good solutions to recurring design problems
- Provide design vocabulary to improve communication
- Encourage designs re-use
- Auto document the source code

“Design patterns help a designer get a design "right" faster”

[Gamma and al. 1994]



1. Introduction

(2/2)

Why detect design motifs:

- Measure the quality of a system
- Help document the source code
- Help understand the design problems and choices

Sub-problem: Identify micro-architectures similar to design motifs

- Program model rarely reflects a design motif completely
- Help to improve the code by applying corrections based on the design motifs

What is a design motif occurrence?

- An occurrence is a micro-architecture where each role of the design motif is played by only one entity



2. Related Work

(1/3)

- Some known identification techniques:
 - Logic programming [Kramer-Prechelt 1996, Wuyts 1998]
 - Constraints programming with explanations [Guéhéneuc-Jussien 2001]
 - Fuzzy logic [Jahnke and al. 1997, Niere and al. 2001]
 - Graphs transformation [Smolarova-Kadlec 2000]
- General problem: limited performance

2. Related Work

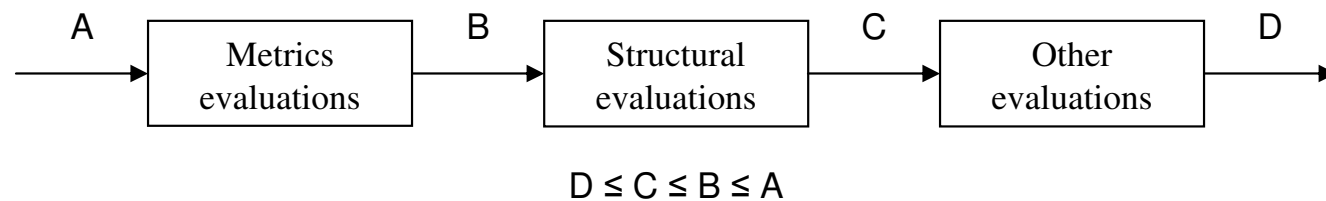
(2/3)

- Metrics to improve efficiency [Antoniol and al. 1998, Guéhéneuc and al. 2004]

Idea : Reduce the search space by removing entities which obviously do not participate in a design motif according to expected metrics values

1. Create fingerprints of design patterns roles with internal attributes of classes playing those roles
2. Use those fingerprints to create identification rules

Multi-stage-filtering:



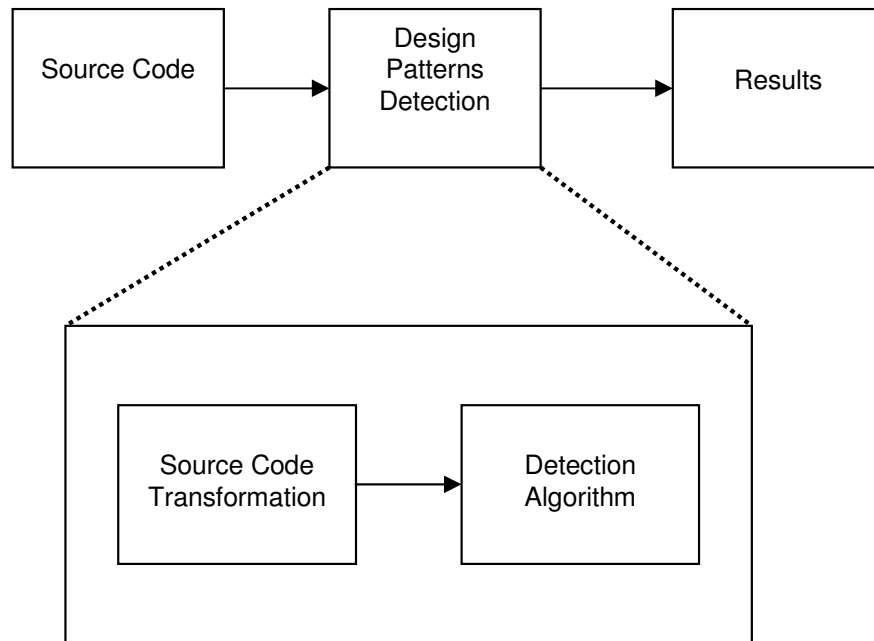


2. Related Work

(3/3)

- Similar problem in bioinformatics:
 - Localising mutated (or not) genes in long anonymous DNA sequences
 - Localising modified (or not) proteins in long amino-acid sequences
- Solutions adopted in bioinformatics:
 - Vector algorithms [Myers 1997, Bergeron-Hamel 2002]
 - Automata simulation [Holub 1997]
 - Dynamic programming alignment [Needleman-Wunsch 1970, Smith-Waterman 1981]

Design patterns detection system:





3. Our approach: Source code transformation

(1/3)

Problem:

- Bioinformatics algorithms work on strings
- Program and design motifs models must be converted in strings

Solution:

- Program and design motifs models can be seen as directed graphs
- Build their string representations by going through every edge in their graphs

3. Our approach: Source code transformation

(2/3)

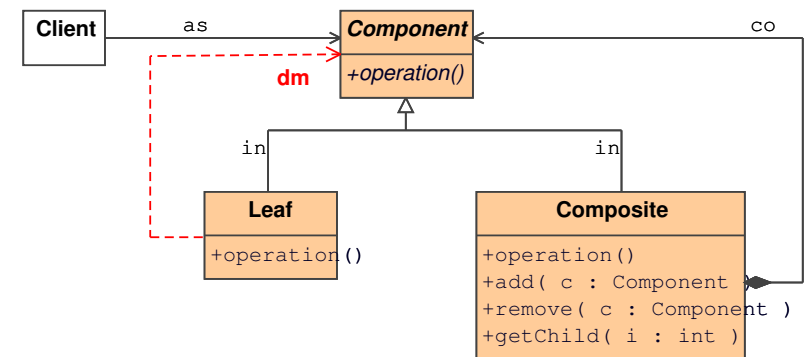
■ Steps to obtain the string representations:

1. Transform the graphs into Eulerian graphs:

- Build an adjacency matrix to identify vertices with unequal in-degree and out-degree
- Compute an optimal list of flows to be added between those vertices with the *transportation simplex*

out \ in	Component	Leaf	Composite
Component	0	1	1
Leaf	0	0	0
Composite	1	0	0

Leaf: out-degree: 0
in-degree: 1



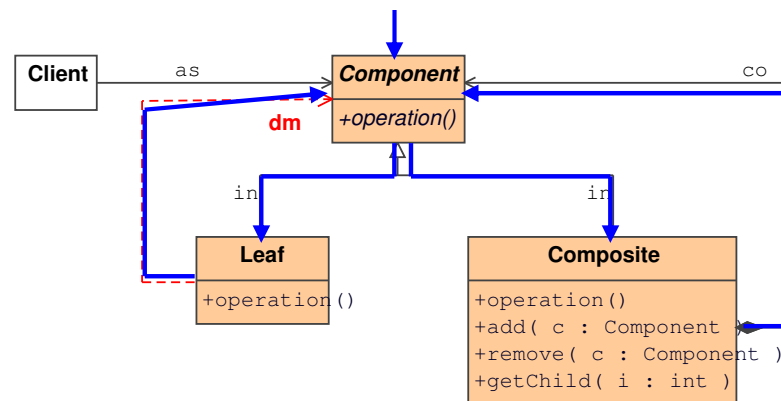
3. Our approach: Source code transformation

(3/3)

■ Steps to obtain the string representations:

2. Find an Eulerian circuit and build the strings:

- Resolve the directed Chinese Postman Problem to find the shortest tour of the graphs (an Eulerian circuit)
- Build the strings by running through the circuits

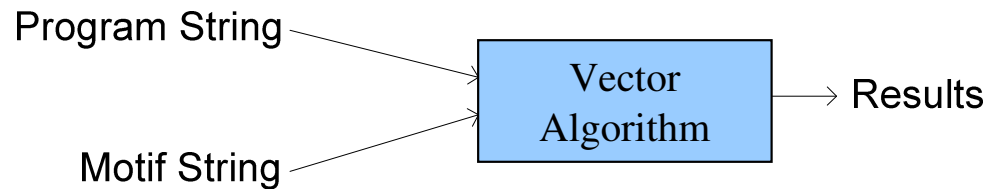


Component in Leaf dm Component in Composite co Component

3. Our approach: Identification algorithm

(1/6)

Objective:



Problem: The design motif string is more like a regular expression than a word

Solution: Iterative bit-vector algorithm

3. Our approach: Identification algorithm

(2/6)

Let $x = x_1x_2\dots x_m$ be a string representing a program model and c a symbol in that string

We define the characteristic vector of c associated to the string x to be:

$$c_i = \begin{cases} 1 & \text{if } x_i = c \\ 0 & \text{otherwise} \end{cases}$$

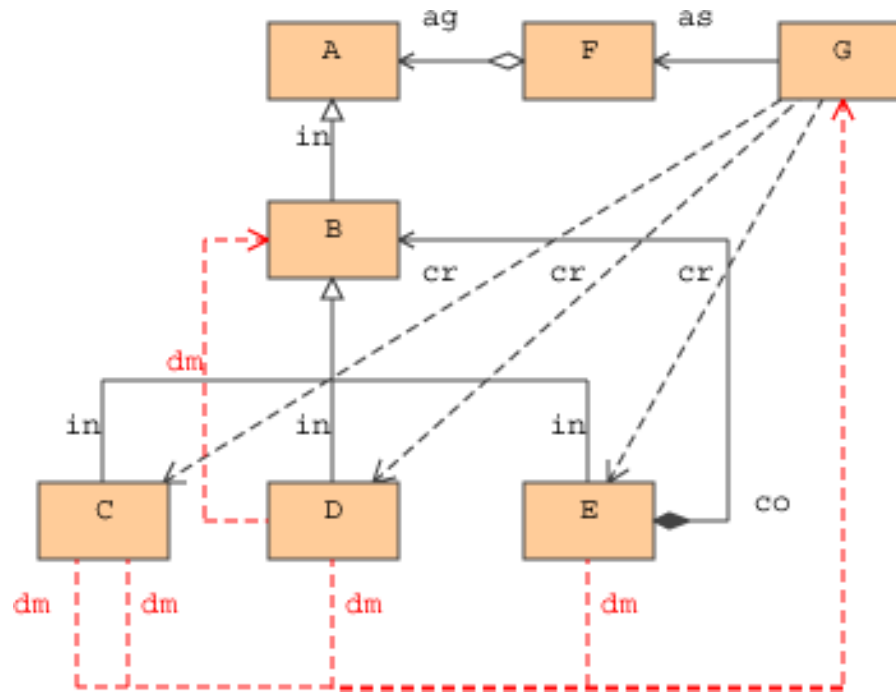
Characteristic vectors are circular sequences of bits on which we operate with standard bit operations:

- and
- or
- left shift
- right shift
- etc.

3. Our approach: Identification algorithm

(3/6)

Example:



A in B in D dm B in E co B in C dm G cr C dm G cr D dm G cr E dm G as F ag A

$$G = \underbrace{0000000000000000}_{14} 10001000100010000$$

$$in = 010100010001 \underbrace{000000000000000000000000}_{19}$$

3. Our approach: Identification algorithm

(4/6)

We match against each other the two strings to build a list of potential occurrences

The algorithm reads triplets of tokens in the design motif string and associates program entities to the roles

For example: *Component in Leaf* *dm Component in Composite co Component*

We retrieve entities before and after an *in* token in the program string with bit-wise operations on the characteristic vectors to have potential entities for the *Component* and *Leaf* roles

3. Our approach: Identification algorithm

(5/6)

Component in Leaf \underbrace{dm} *Component in Composite* *co* *Component*

We search for entities after a *Component* entity and an *in* relationship (ex: B in *Composite*)

$$\rightarrow \rightarrow B = 00001000100010 \underbrace{\dots 0}_{18}$$

$$\rightarrow in = 00101000100010 \underbrace{\dots 0}_{18}$$

$$(\rightarrow \rightarrow B) \wedge (\rightarrow in) = 00001000100010 \underbrace{\dots 0}_{18}$$

$$E = \underbrace{00000000}_{8} 10 \dots 0 \underbrace{10000000}_{6}$$

$$(\rightarrow \rightarrow B) \wedge (\rightarrow in) \wedge E = \underbrace{00000000}_{8} 10 \underbrace{\dots 0}_{22}$$

Triplets							
First (in)		Third (in)			Fourth (co)		
Component	Leaf	Component	Leaf	Composite	Component	Leaf	Composite
A	B	A	B	B	B	C	E
B	C	B	C	C	B	D	E
B	D	B	C	D	B	E	E
B	E	B	C	E			
		B	D	C			
		B	D	D			
		B	D	E			
		B	E	C			
		B	E	D			
		B	E	E			

Occurrences after processing the first third and fourth triplets

3. Our approach: Identification algorithm

(6/6)

A program model rarely reflects a design motif completely

- We include automatic and manual approximation mechanisms
- 1. Association relationships:
 - Composition, aggregation, association and use relationship
 - Conjunction of characteristic vectors
- 2. Entity counts
 - Ex: A Composite without leaf
 - Triplets can be overlooked
- 3. Inheritance relationship:
 - Some entities could be inserted or removed from a hierarchy
 - Parent and children of a class are also possible entities for a role

4. Case Study

(1/3)

- We applied our algorithm on 3 public domain software:
 1. Juzzle v0.5
 2. JHotDraw v5.1
 3. QuickUML 2001
- We used an AMD Athlon at 2Ghz with 1024 Mb RAM

Programs	Sizes	Computation times
Juzzle v0.5	99	5
JHotDraw v5.1	261	37
QuickUML 2001	373	149

Computation times (in seconds) for building the string representation

4. Case Study

(2/3)

We compared our bit-vector technique with Ptidej, a framework implementing:

- Explanation-based constraint programming
- Metric-enhanced explanation-based constraint programming

	CP	CP+M	BV	BV+O
	Abstract Factory			
JHotDraw v5.1	1202	275	218	27
Juzzle v0.5	1	2	0.9	0.3
QuickUML 2001	785	153	97	29
	Composite			
JHotDraw v5.1	$+\infty$	17362	129	25
Juzzle v0.5	45	3	0.5	0.3
QuickUML 2001	$+\infty$	26514	185	27

Identification times (in seconds) of design motifs **with all approximations**

4. Case Study

(3/3)

	Existing Occurrences	Exact			Approximate		
		CP	CP+M	BV	CP	CP+M	BV
		Abstract Factory					
JHotDraw v5.1	0	221	69	221	2994	849	194
Juzzle v0.5	0	0	0	0	9	0	13
QuickUML 2001	13	57	23	57	593	124	118
Composite							
JHotDraw v5.1	70	N/A	0	0	N/A	16983	609
Juzzle v0.5	0	0	0	0	20	0	0
QuickUML 2001	22	N/A	0	0	N/A	4743	513

Number of identified occurrences of the design motifs



5. Current and Future Work

- Improve the precision of the results
 - Combine bit-vector algorithms with empirical metrics values
 - Add *negative* relationships in design motifs
 - Add dynamic information in the string representations
- Try the approach on more and bigger software
- Try other bioinformatics string matching algorithms

