

Design Patterns Decomposition in Reverse Engineering Tools

Francesca Arcelli & Claudia Raibulet



Università degli Studi di Milano-Bicocca
DISCo – Dipartimento di Informatica Sistemistica e Comunicazione
Milan, Italy



**1st International Workshop on Design Patterns Theory and Practice
Budapest, Hungary, 2005**

The Role of Design Patterns in Reverse Engineering

□ *Provide additional information related to the **rationale** behind the design*

*=> information about **how** the architecture has been build, as well as **why** it has been built in a specific way*

□ *Problems:*

- *how can be their variants recognized?*
- *how to face scalability?*
- *....*

Classification of Design Pattern Detection Tools

□ Design patterns detection tools classification:

- Manual, semi-automated, automated [Keller et al., 99]
- Static and dynamic approaches
- The entire representation of design patterns (i.e., PTIDEJ, CrocoPat, ...)
- A minimal set of key structures design patterns consist of (i.e., SPOOL, ...)
- Sub-components design patterns are built of (i.e., FUJABA, SPQR, ...)
-

Scope

- ❑ *Which are the sub-components of design patterns?*
- ❑ *How can be design patterns represented through their sub-components?*
- ❑ *Do sub-components represent a first approach toward a fully automated detection approach?*

FUJABA: Design Pattern Definition

- *Design patterns – defined through UML class diagrams and story diagrams (activity + interaction)*
- *Common parts are defined separately as **sub-patterns***
- *Construction of (sub-)patterns of other sub-patterns exploit inheritance and use relationships*

FUJABA: The Role of Sub-Patterns

- ❑ *Reduce the dimensions of the design patterns catalog*
- ❑ *Reused as atomic elements in the detection process*
- ❑ *Introduce more levels of abstraction between source code and design patterns -> both a bottom-up and a top-down detection strategy*

SPQR: Design Pattern Definition

□ *Design patterns are defined through:*

EDPs – Elemental Design Patterns

- Low-level design patterns which express in a formal denotational semantics fundamental OO concepts
- Design patterns are expressed as compositions of EDPs

Rho-calculus = subset of sigma-calculus + reliance operators

- From sigma-calculus inherits type definition, object typing, and type subsumption concepts
- Reliance operators express if elements rely or depend of other elements and to what extent they do so

SPQR: The Role of EDPs

- *EDPs represent the central element of SPQR*
- *EDPs are language and domain independent as design patterns*
- *The detection of EDPs provide a better understanding of source code even not detecting design patterns*

FUJABA vs SPQR

□ *FUJABA*

- A tool for both forward and reverse engineering
- Sub-patterns introduced to reduce the dimensions of the design pattern catalog -> focused on the structure of design patterns
- Sub-patterns organized hierarchically

□ *SPQR*

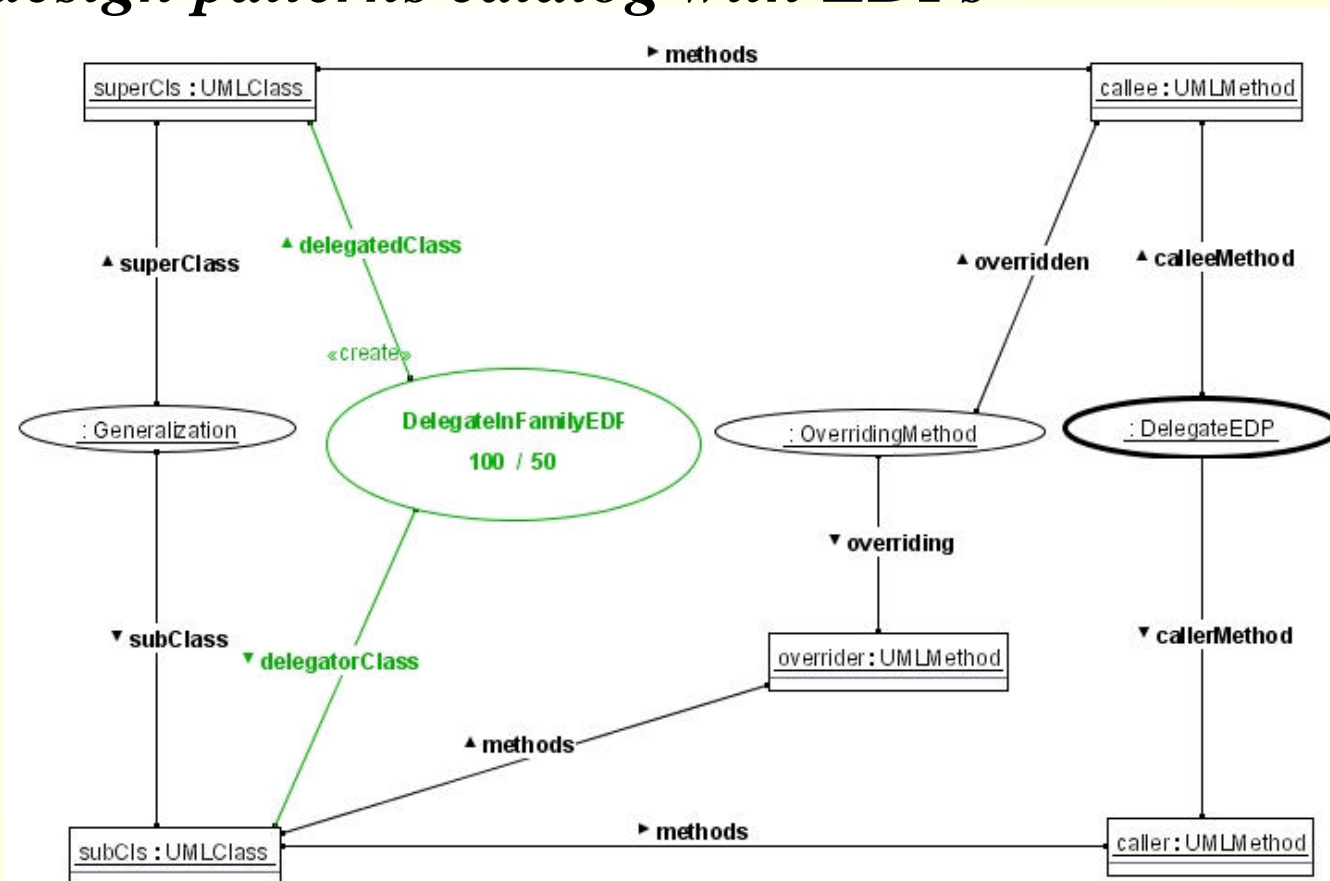
- A tool for design pattern detection
- EDPs play a central role for design pattern definition and detection
- EDPs form a plain abstraction level between source code and design patterns

Main Benefits of Decomposing Design Patterns

- ❑ *Sub-components are simpler than patterns, hence their detection rules are simpler*
- ❑ *Design patterns can be described formally through their sub-components*
- ❑ *Decomposing design patterns into sub-components do not affect their flexibility -> variants problems impact mainly on sub-components*
- ❑ *Sub-components can be considered independently of design patterns due to the design intends they capture*

EDP Detection

- *EDP4FUJABA – the extension of the FUJABA design patterns catalog with EDPs*



EDP Detection

□ *EDPDetector4Java – identifying EDPs into Java code starting from the EDPs catalog of SPQR*

- New approach for EDP detection by defining EDP as logical functions in terms of variables representing method calls or class properties

Redirect:

$$\bar{A}_3 \cdot \bar{A}_4 \cdot \bar{A}_5 \cdot A_0 \cdot \bar{A}_2$$

$$\text{Redirect} = \sum_{A_0, \dots, A_7} (1, 3)$$

EDP Detection

□ *EDPDetector4Java – identifying EDPs into Java code starting from the EDPs catalog of SPQR*

- New approach for EDP detection by defining EDP as logical functions in terms of variables representing method calls or class properties

Redirect:

<i>Method Invocation</i>	A_3 sender	A_4 receiver	A_5 referrer	A_0 referred	A_2
<i>Redirect</i>	<i>Other Class</i>		<i>Similar</i>		
<i>Conglomeration</i>	<i>Self</i>		<i>Different</i>		
<i>Recursion</i>	<i>Self</i>		<i>Same Declaration</i>		
<i>DelegatedConglomeration</i>	<i>Super</i>		<i>Dissimilar</i>		
<i>RedirectedRecursion</i>	<i>Super</i>		<i>Same Signature</i>		
<i>RevertMethod</i>	<i>Same Class</i>		<i>Dissimilar</i>		
<i>RedirectInFamily</i>	<i>Parent</i>		<i>Similar</i>		
<i>DelegateInLtdFamily</i>	<i>Sibling</i>		<i>Dissimilar</i>		
...		

$$\text{Redirect} = \sum_{A_0, \dots, A_7} \text{Dissimilar} (1, 3)$$

EDP Detection

Characteristic	Logical Variable
Same Method Name	A_0
Same Method Signature	A_1
Same Method Declaration	A_2
Same Class	A_3
Super class	A_4
Same Family	A_5
Super Reference	A_6
This Reference	A_7

□ *EDPDetector4Java* – identifying EDPs into Java code starting from the EDPs catalog of SPQR

- New approach for EDP detection by defining EDP as logical functions in terms of variables representing method calls or class properties

Redirect:

Method Invocation	EDP	sender	receiver	referrer	referred
<i>Redirect</i>	A_3	A_4	A_5	A_0	A_2
<i>Conglomeration</i>		Self		Different	
<i>Recursion</i>		Self		Same Declaration	
<i>DelegatedConglomeration</i>		Super		Dissimilar	
<i>RedirectedRecursion</i>		Super		Same Signature	
<i>RevertMethod</i>		Same Class		Dissimilar	
<i>RedirectInFamily</i>		Parent		Similar	
<i>DelegateInLtdFamily</i>		Sibling		Dissimilar	
...		...		\sum_{A_0, \dots, A_7}	$(1, 3)$

Characteristic

Logical Variable

EDP Detection

Same Method Name

A₀

Same Method Signature

A₁

Same Method Declaration

A₂

Same Class

A₃

Super Class

A₄

Same Family

A₅

Super Reference

A₆

This Reference

A₇

EDPDetector4Java – identifying EDPs into Java code starting from the EDPs catalog of SPQR

- New approach for EDP detection by defining EDP as logical functions in terms of variables representing method calls or class properties

Redirect:

Method Invocation EDP A₃ sender/receiver A₄ referrer/referred A₅ A₀ A₂

Redirect Other Class Ssimilar

	N	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	
Delegated Conglomeration	1	0	0	0	0	0	0	0	1	1
Redirected Recursion	0	0	0	0	0	0	0	0	1	0
Revert Method	0	0	0	0	0	0	0	0	1	1
Redirect In Family	0	0	0	0	0	0	0	0	0	0

Delegated In Ltd Family ...

Redirect = $\sum_{A_0, \dots, A_7}^{\text{Dissimilar}} (1, 3)$

Current and Future Work

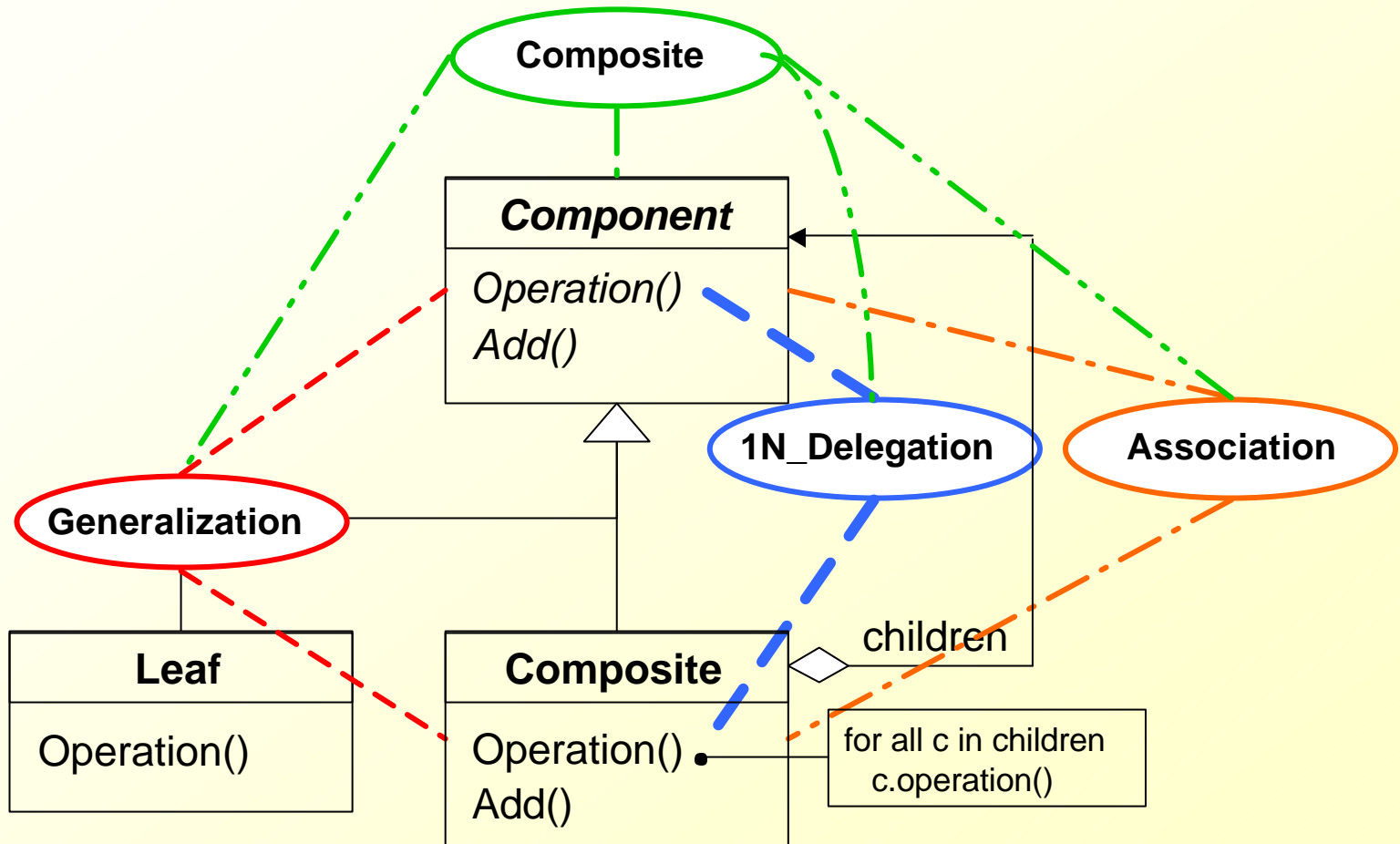
□ *DPDetector4Java*

- extension for design patterns detection
- considering also dynamic analysis
- plug-in for Eclipse

□ *J2POML2EDP – EDP detection according to the SPQR approach for Java source code*

□ *Similarities/Differences between sub-patterns and EDPs*

Sub-Patterns of the Composite Design Pattern



EDPs of the Composite Design Pattern

