



Recovering Binary Class Relationships: Putting Icing on the UML Cake

Yann-Gaël Guéhéneuc
Hervé Albin-Amiot

 Département d'informatique
École des Mines de Nantes

 Software Engineering Laboratory
Department of Computing Science and Operations Research
University of Montreal



Statements

- Maintenance
 - 50% of the total cost of a program
- Program understanding
 - 50% of the maintainers' time
- Any help in understanding a program would reduce its cost!



Context

■ OO maintenance

– Source code

- Most accurate source of information

– Design models

- Useful to understand source code (architecture of a program)
- Non-existent or obsolete

■ **Class diagrams** help in understanding source code



Approach

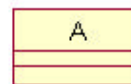
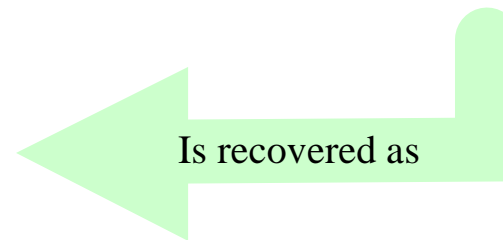
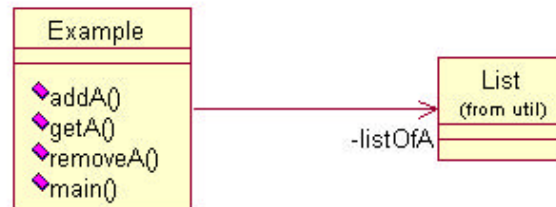
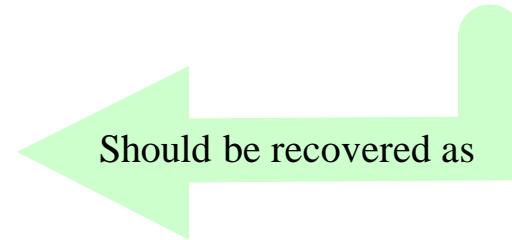
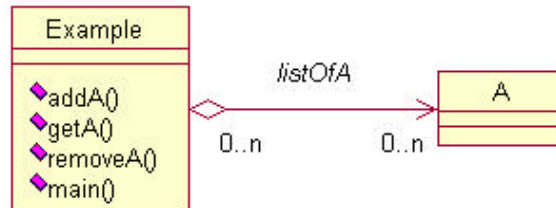
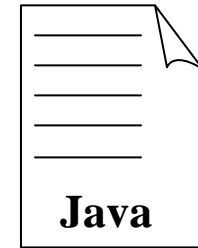
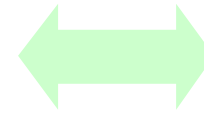
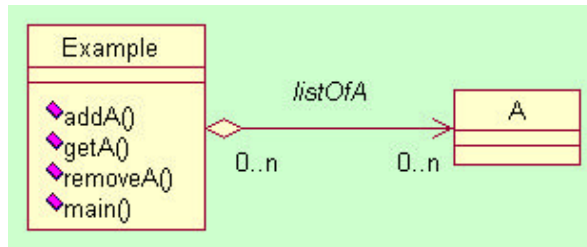
- Automated recovery of class diagrams from program source code
 - Extract relevant information
 - Operational semantics for relevant constituents of class diagram
 - **Consistent mapping**
 - Concepts in programming languages
 - Concepts in modeling languages



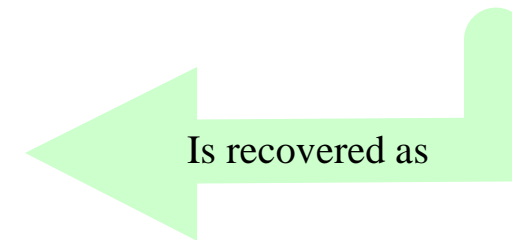
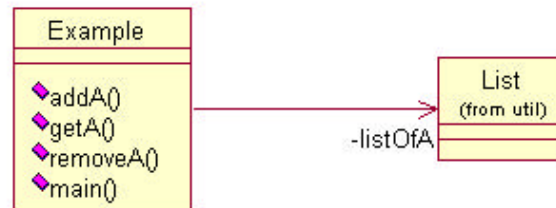
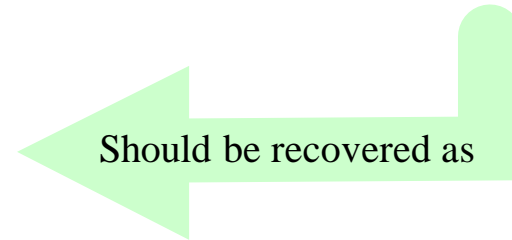
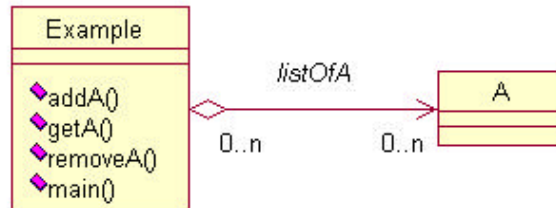
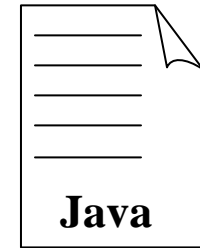
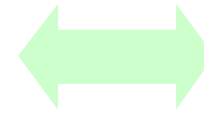
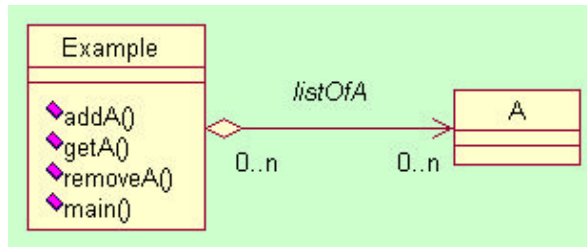
Problem

- Use of our approach on Java and UML
 - UML class diagrams
 - Only a notation! As emphasised during the panel on “Model Driven Architecture”
 - Mapping
 - Semantic continuity for some concepts, such as classes, fields, methods
 - **Discontinuity for binary class relationships**

Problem (cont'd)



Problem (cont'd)





Proposal

- Study of the **reverse-engineering** of binary class relationships
 - “Consensual” definitions
 - At design-level
 - At implementation-level
 - Properties
 - Formalisations
 - Recovery algorithms



Proposal (cont'd)

- Study of the reverse-engineering of binary class relationships
 - Mapping between programming and modeling languages
 - **Sound foundations**
 - Understanding
 - Round-tripping
 - Design patterns identification
 - ... and much more!



Overview

- Motivations
- **Formalisations**
- Recovery
- Validations
- Conclusion

Definitions

- Up to now, no consensual definitions
- Study of more than **30 definitions**
- “Consensual” definitions

Link		
<i>Origin</i>	<i>Means</i>	<i>Target</i>
Instance	Direct	Instance
Instance	Field	Instance
Instance	Field + Lifetime property	Instance

Is described by

Relationship		
<i>Origin</i>	<i>Name</i>	<i>Target</i>
Entity	Association	Entity
Class	Aggregation	Entity
Class	Composition	Entity

“Entity” denotes either a class or an interface

Definitions

■ Association

– At design-level

- Conceptual **link** between two classes
- Multiple instances can be involved

– At implementation-level

- Message sends between instances of two classes, an **origin** and a **target**
- Oriented, irreflexive, anti-symmetric at instance and class level; asymmetric at instance level

Definitions

■ Aggregation

– At design-level

- An aggregation relationship is an association between two classes: The **whole** and the **part**

– At implementation level

- An aggregation relationship exists between classes A and B when the definition of A, the whole, **contains instances** of B, its part



Definitions

■ Composition

– At design-level

- Aggregation where parts are **destroyed** when their whole is destroyed
- Parts might be instantiated before the whole and exchanged

– At implementation level

- Aggregation with constraint on the **lifetimes** and on the **ownership** of the whole and of its part

Properties

■ Four minimal properties



- Exclusivity, an instance of B (b) is in relation with only one instance of A (a)
- Invocation site, for method invocations from a to b
- Lifetime, time of destruction of b wrt. a
- Multiplicity, number of b 's for one a

Properties

■ Four minimal properties

- Exclusivity, $EX(A, B) \in \mathbb{B}$
- Invocation site, $IS(A, B) \subseteq \{\text{field, parameter, local variable, ...}\}$
- Lifetime, $LT(A, B) \in \{+, -\}$
- Multiplicity, $MU(A, B) \in \mathbb{N} \cup \{+\infty\}$

Formalisations

- Rewriting binary class relationships as **conjunctions** of their properties

- Association

$AS(A, B) =$

$(EX(A, B) \in \mathbb{B}) \wedge (EX(B, A) \in \mathbb{B}) \wedge$

$(IS(A, B) \subseteq \{\text{field...}\}) \wedge (IS(B, A) = \emptyset) \wedge$

$(LT(A, B) \in \{+, -\}) \wedge (LT(B, A) \in \{+, -\}) \wedge$

$(MU(A, B) = [0, +\infty]) \wedge (MU(B, A) = [0, +\infty])$

Formalisations

■ Aggregation

$$\text{AG}(A, B) =$$

$$(\text{EX}(A, B) \in \mathbb{B}) \wedge$$

$$(\text{EX}(B, A) \in \mathbb{B}) \wedge$$

$$(\text{IS}(A, B) \subseteq \{\text{field, array field, collection field}\}) \wedge$$

$$(\text{IS}(B, A) = \emptyset) \wedge$$

$$(\text{LT}(A, B) \in \{+, -\}) \wedge$$

$$(\text{LT}(B, A) \in \{+, -\}) \wedge$$

$$(\text{MU}(A, B) = [0, +\infty]) \wedge$$

$$(\text{MU}(B, A) = [1, +\infty])$$

Formalisations

■ Composition

$\text{CO}(A, B) =$

$(\text{EX}(A, B) = \text{true}) \wedge$

$(\text{EX}(B, A) = \text{false}) \wedge$

$(\text{IS}(A, B) \subseteq \{\text{field, array field, collection field}\}) \wedge$

$(\text{IS}(B, A) = \emptyset) \wedge$

$(\text{LT}(A, B) = +) \wedge$

$(\text{LT}(B, A) = -) \wedge$

$(\text{MU}(A, B) = [1, +\infty]) \wedge$

$(\text{MU}(B, A) = [1, 1])$

Formalisations

■ Emergent characteristics

- Minimality of the set of properties
 - See p. 307 of the proceedings
- Order among binary class relationships
 - Association
 - Aggregation
 - Composition
- Static and dynamic parts
 - $IS(A, B)$, $MU(A, B)$
 - $EX(A, B)$, $LT(A, B)$



Overview

- Motivations
- Formalisations
- **Recovery**
- Validations
- Conclusion



Recovery

- Binary class relationships
 - Not syntactical constructs in source code
 - Expressed with four minimal properties
- Recovery
 - Computation of the properties
 - Inference of **existing** relationships

Recovery

■ Algorithms

- Static analyses
 - $IS(A, B)$, $MU(A, B)$
- Dynamic analyses
 - $EX(A, B)$, $LT(A, B)$

Recovery

■ Algorithms

- Order among the relationships
- Association
 - $IS(A, B)$ with respect to other classes
- Aggregation
 - $MU(A, B)$ with respect to **associated** classes
- Composition
 - $EX(A, B)$
 - $LT(A, B)$

} with respect to **aggregated** classes

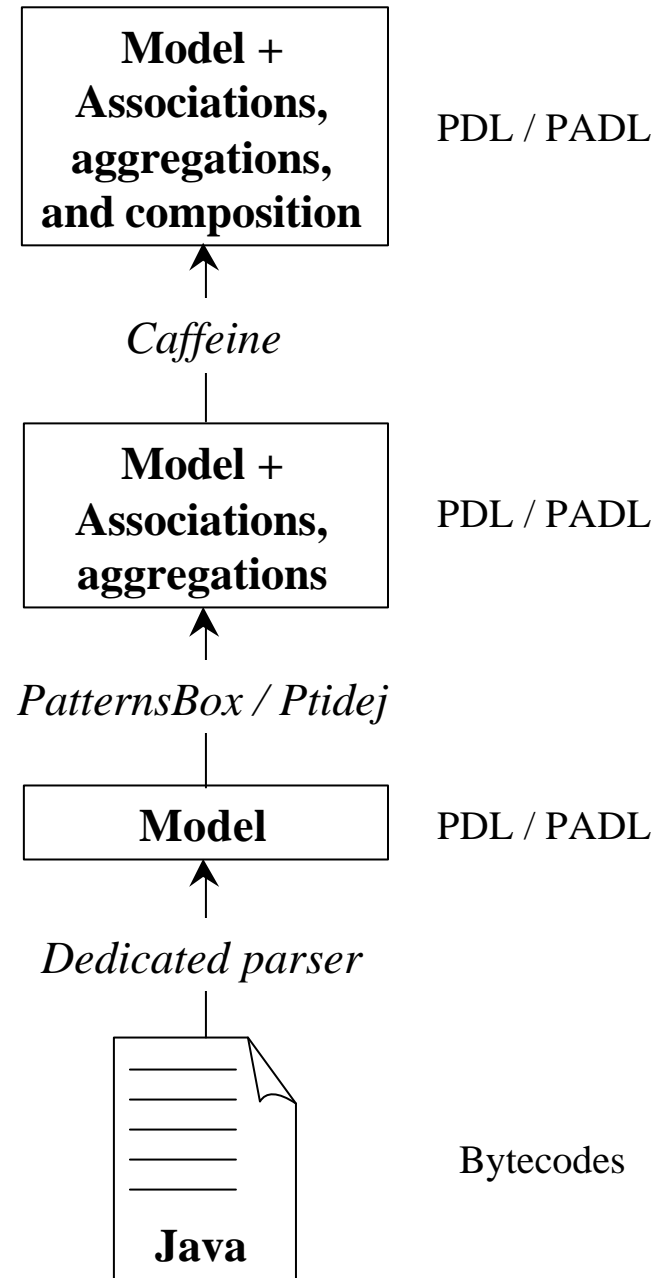


Overview

- Motivations
- Formalisations
- Recovery
- **Validations**
- Conclusion

Implementations

- Meta-model
 - PDL / PADL
- Static analyses
 - PatternsBox / Ptidej
- Dynamic analyses
 - Caffeine





Experimentations

■ Case studies

- Java AWT v1.2.2
 - 367 classes
- JHotDraw v5.1
 - 171 classes
- JUnit v3.7
 - 45 classes

Experimentations

■ Association

- 4,925 for 583 classes but required for aggregation

■ Aggregation

Frameworks	Found	Manual	False Hits
Java AWT	17	20	1
JHotDraw	6	8	0
JUnit	1	4	0
Total	24	32	1

Recall: 0.75

Precision: 0.96

■ Composition

- Precision and recall of 100% (JUnit only)



Experimentations

- Emergent characteristics
 - Quantitative
 - Possible metrics
 - Qualitative
 - Assessment
 - Comparison
 - Practicality



Overview

- Motivations
- Formalisations
- Recovery
- Validations
- Conclusion



Conclusion

- Any help in understanding a program would reduce its cost
- Automated recovery of class diagrams from program source code
- **Mapping for binary class relationships**
 - “Consensual” definitions
 - Formalisations
 - Recovery algorithms



Related work

- **Civello** at OOPSLA 1993
 - Classification of binary class relationships
 - No operational semantics
- **Henderson-Sellers and Barbier** in L'Objet 1999
 - Properties
 - Design-level
- **Jackson and Waingold** at ISCE 1999
 - Operational semantics
 - Flavours of association, aggregation only



Future work

- Detection of other UML constituents
- Static analyses vs. dynamic analyses
- Experimental validations
 - Maintainers
 - Class diagrams with/without binary class relationships